ERAD-SP 2016

Anais da 7ª Escola Regional de Alto Desempenho de São Paulo

03 a 05 de agosto de 2016, São Paulo - SP

Realização

Organização

Patrocínio

ISSN 2178-1397

Sumário

Resumos da Trilha Iniciação Científica - com apresentação de pôster

Resumos da Trilha Pós-Graduação - com apresentação de pôster

Material utilizado nos Tutoriais & Minicursos

Mensagem dos Coordenadores

A computação de alto desempenho popularizou-se e está cada vez mais no nosso dia-a-dia. Ela pode ser encontrada em vários domínios de aplicação e o seu principal propósito é melhorar a vida das pessoas, por meio de soluções computacionais mais rápidas, mais abrangentes e economicamente viáveis. O estado da arte nos sinaliza que estamos cada vez mais dependentes de softwares paralelos em atividades cotidianas e em ambientes industriais, emergindo uma necessidade crescente de respostas rápidas e corretas desses sistemas, os quais, muitas vezes, caracterizam-se por serem críticos, ubíquos, persistentes e adaptativos. Tal demanda estimula o desenvolvimento de conhecimento em torno da computação de alto desempenho considerando as diferentes arquiteturas de computadores, paradigmas de programação e avaliação de desempenho. Tendo-se este norte, observam-se importantes iniciativas que promovem um melhor relacionamento indústria-academia, no que se refere ao desenvolvimento de novos modelos de programação, ao uso eficiente dos computadores paralelos e à disseminação de conhecimento, esta última permitindo às novas gerações se adaptarem ao uso de tecnologias de alto desempenho que ainda nem foram criadas. Nesse caminho, ainda existem desafios para a transferência de tecnologia entre academia e indústria.

É com muita satisfação que chegamos a mais uma ERAD-SP tendo este cenário como pano de fundo! A ERAD-SP - Escola Regional de Alto Desempenho de São Paulo - é o principal evento técnico-científico sobre a computação de alto desempenho no Estado de São Paulo. Realizada desde 2010 pela Comissão Especial em Arquitetura de Computadores e Processamento de Alto Desempenho da SBC, a ERAD-SP visa a estimular o estudo, a pesquisa e o desenvolvimento em Arquitetura de Computadores, Processamento de Alto Desempenho, Sistemas Distribuídos e Aplicações. A ERAD-SP promove a interação entre estudantes, pesquisadores e indústria de modo a estreitar laços de cooperação nessa importante área da computação.

Nessa 7ª edição, a ERAD-SP 2016 ocorre na Faculdade de Computação e Informática (FCI) da Universidade Presbiteriana Mackenzie, nesta maravilhosa e vibrante cidade que é São Paulo. Esta edição da ERAD-SP foi organizada pela Faculdade de Computação e Informática da Universidade Presbiteriana Mackenzie, pela Escola Politécnica da USP, pelo Instituto de Ciências Matemáticas e de Computação da USP-São Carlos, pela Universidade Federal do ABC e pela Universidade Federal de São Paulo.

Inspirado no formato de anos anteriores, o programa técnico da ERAD-SP 2016 é composto por: palestras convidadas, minicursos e tutoriais, desafio de programação paralela, e apresentações de artigos em duas trilhas – pós-graduação e iniciação científica. Neste ano, temos na ERAD-SP seis interessantes palestras que abordam temas relevantes e atuais como: o uso da Computação de Alto Desempenho em diferentes contextos (como pesquisas relacionadas às finanças e aos grandes telescópios), o uso de supercomputadores pela comunidade científica brasileira, e o uso eficiente de energia por softwares paralelos.

A ERAD-SP 2016 recebeu a submissão de 40 artigos técnicos, na forma de resumos estendidos. Foram submetidos 21 resumos de pós-graduação e 19 de iniciação científica. Todos os artigos foram cuidadosamente revisados por no mínimo três membros do comitê de programa, contando com a colaboração de revisores nacionais e internacionais. Os cinco melhores artigos de pós-graduação e os quatro melhores artigos de iniciação científica foram aceitos para apresentação oral, sendo que a taxa de aceitação para apresentação oral ficou em 20%. Além destes, também foram aceitos como pôsteres outros 15 artigos de iniciação científica e 13 de pós-graduação.

Seguindo o mesmo espírito dos anos anteriores, a ERAD-SP 2016 permitiu que trabalhos em andamento e com resultados preliminares fossem aceitos e apresentados como pôsteres, estimulando assim a vinda de estudantes e enriquecendo o intercâmbio de conhecimento na computação de alto desempenho. Todos os trabalhos aceitos na ERAD-SP 2016 serão publicados na BDBComp da SBC.

Esta edição da ERAD-SP também apresentou cinco Minicursos entre básicos e avançados, além de dois Tutoriais. A ideia dessa programação foi de contemplar desde assuntos clássicos, como a programação paralela em diferentes tipos de máquinas, até ferramentas contemporâneas relacionadas à Big Data. Para que isto fosse possível, a ERAD-SP 2016 diversificou criando uma nova categoria, a de Tutoriais. Enquanto os minicursos possuem duração de 3 horas, os tutoriais convidados têm duração de 1,5 hora, dando oportunidade para que assuntos básicos e não contemplados pelos minicursos propostos pudessem ser apresentados.

O Desafio de Programação Paralela consiste na resolução paralela de um problema específico da área de Computação onde as equipes são pontuadas em função do tempo de execução total. A resolução do problema deveria ser realizada antes do evento, em ambiente próprio da equipe, com o resultado sendo divulgado durante o evento. Neste ano o objetivo das equipes foi realizar a paralelização da etapa de aprendizado de uma rede neural multicamadas utilizando a arquitetura Intel Xeon Phi. Foi disponibilizado um ambiente de testes para que as equipes pudessem avaliar suas soluções. Infelizmente, não houveram submissões de soluções para o problema dentro do prazo estipulado pela organização.

Somos muito gratos aos membros do comitê de programa e aos eventuais revisores indicados por eles, os quais nos prestigiaram com seu tempo para as revisões. A participação do comitê contribuiu decisivamente para com a melhoria de qualidade dos artigos avaliados. Sem a colaboração de vocês não conseguiríamos finalizar as revisões em tão pouco tempo.

Agradecemos à Universidade Presbiteriana Mackenzie e em especial à Faculdade de Computação e Informática por ceder sua infraestrutura e abrigar a ERAD-SP 2016. Agradecemos também a CAPES e o CNPq pelo auxilio financeiro concedido para a organização do evento. Agradecemos também aos patrocinadores, a ABRAMTI e a SGI, que acreditam na ideia da ERAD-SP 2016 e se fizeram presentes tanto financeiramente quanto na colaboração de divulgação do evento.

Finalmente, gostaríamos de agradecer aos convidados, aos autores dos artigos submetidos e a todos os participantes desta Escola por fazê-la acontecer. Esperamos que a ERAD-SP 2016 seja proveitosa e contribua para a ampliação e consolidação da Computação de Alto Desempenho no Estado de São Paulo e no Brasil. Desejamos a todos uma ótima Escola!

São Paulo, agosto de 2016

Calebe de Paula Bianchini (MACKENZIE)
Liria Matsumoto Sato (Poli/USP)
Coordenadores da ERAD-SP 2016

Comitês

Coordenação Geral

Calebe de Paula Bianchini	Universidade Presbiteriana Mackenzie
Liria Matsumoto Sato	Universidade de São Paulo (EP)

Coordenação Geral de Programa

Paulo Sérgio Lopes de Souza	Universidade de São Paulo (ICMC)

Coordenação de Minicursos e Tutoriais

Denise Stringhini	Universidade Federal de São Paulo

Coordenação Local

Luciano Silva	Universidade Presbiteriana Mackenzie
Takato Kurihara	Universidade Presbiteriana Mackenzie

Coordenação de Patrocínios

Vivaldo José Breternitz	Universidade Presbiteriana Mackenzie

Coordenação do Desafio de Programação Paralela

Álvaro Fazenda	Universidade Federal de São Paulo
Raphael Camargo	Universidade Federal do ABC

Comitê de Programa

Aleardo Manacero Jr.	IBILCE/UNESP
Alexandre Rocha	IFT/UNESP
Alexandro Baldassin	IGCE/UNESP
Alfredo Goldman	IME/USP
Alvaro Fazenda	ICT/UNIFESP
Arlindo da Conceição	ICT/UNIFESP
Caetano Miranda	IF/USP
Calebe Bianchini	Mackenzie
Celso Hirata	ITA
Celso Mendes	University of Illinois
Daniel Cordeiro	EACH/USP
Daniel Pedronette	UNESP
Denise Stringhini	UNIFESP
Edson Borin	IC/UNICAMP
Eduardo Rodrigues	IBM Research Brazil
Emilio Francesquini	IC/UNICAMP
Francisco Massetto	UFABC
Francisco Monaco	ICMC/USP
Hélio Guardia	DC/UFSCar
Hermes Senger	DC/UFSCar
Igor Freitas	Intel
Jairo Panetta	ITA
Jó Ueyama	ICMC/USP
Júlio Estrella	ICMC/USP
Liria Sato	EP/USP
Luciana Arantes	Université de Paris VI
Luciano Silva	Mackenzie
Luis Scott	UFABC
Luiz Bovolenta	UNESP
Luiz Fernando Bittencourt	UNICAMP
Marco Netto	IBM Research
Márcio Castro	UFSC
Norian Marranghello	IBILCE/UNESP
Paulo Sérgio Lopes de Souza	ICMC/USP
Philippe O. A. Navaux	UFRGS
Raphael Camargo	UFABC
Renata S. Lobato	IBILCE/UNESP
Renato Ishii	UFMS
Ricardo Menotti	DC/UFSCar
Ricardo Santos	UFMS
Roberta Spolon	UNESP/BAURU
Rodolfo Azevedo	UNICAMP
Rodrigo Mello	ICMC/USP
Sandro Rigo	UNICAMP
Sarita Bruschi	ICMC/USP

Siang Song	IME/USP
Vanderlei Bonato	ICMC/USP

Revisores Externos

Marcos Amaris	IME/USP
Pedro Bruel	IME/USP
Thiago Okada	IME/USP

O presente trabalho foi realizado com o apoio da CAPES, entidade do Governo Brasileiro (MEC) voltada para a formação de investimentos na formação de recursos humanos de alto nível no país, bem como do acesso e divulgação da produção científica e promoção da cooperação científica nacional e internacional, por meio do programa PAEP, Programa de Apoio a Eventos no Pais; do CNPq, entidade do Governo Brasileiro (MCTIC) voltada para o fomento da pesquisa científica e tecnológica e o incentivo da formação de pesquisadores brasileiros, por meio do ARC, promoção de eventos científicos, tecnológicos e/ou de inovação; da ABRAMTI, Associação Brasileira de Melhoria em TI, organização sem fins lucrativos, cuja missão é incentivar e propiciar as melhores técnicas e práticas de engenharia de software; da SGI, empresa de soluções inovadoras para computação de alto desempenho.

Minicursos

Básicos

Minicurso I : Um Estudo sobre a Expansão das Diretivas de Compilação para Interceptação de Código OpenMP

por Rogério Gonçalves (UTFPR) e Alfredo Goldman (IME-USP)

O objetivo é apresentar uma introdução ao OpenMP mostrando as principais diretivas de compilação para a cobertura de regiões paralelas, laços, seções e tasks, bem como a estrutura do código gerado pela expansão dessas diretivas. Esses conceitos podem ser aplicados no desenvolvimento de bibliotecas para interceptar chamadas que as aplicações fazem ao runtime do OpenMP, o que possibilita a execução de código pré ou pós chamada para a criação de traces, para atividades de logging, para o monitoramento e avaliação de desempenho ou para offloading de código para dispositivos aceleradores.

Minicurso II : Introdução à Programação de GPUs com a plataforma CUDA

por Pedro Bruel (IME-USP) e Alfredo Goldman (IME-USP)

O objetivo deste minicurso é disseminar o conhecimento e o uso de GPUs na pesquisa em Ciência da Computação. Apresentaremos a plataforma CUDA como ferramenta para facilitar a programação e a otimização de projetos para GPUs. Introduziremos algumas características do hardware de GPUs, o modelo de programação da linguagem CUDA C, e ferramentas para profiling e debugging de aplicações CUDA. O objetivo é que os participantes saiam do minicurso com o interesse e a capacidade de se aprofundar nas ferramentas da plataforma CUDA e de utilizar GPUs em seus projetos pesquisa.

Minicurso V : Introdução ao Spark para Data Science

por Thiago Baldim (Reddrummer Tecnologia)

Apache Spark é uma engine para processamento de dados em larga escala. Utilizando infraestruturas baseadas em Hadoop para dados distribuidos, garantindo uma velocidade de processamento até cem vezes mais rápido que o processamento por hadoop. Permitindo execução de map reduces em memória RAM, tendo ganho de processamento dez vezes mais rápido que em disco.

Avançados

Minicurso IV : Introdução à Vetorização em Arquiteturas Paralelas Híbridas

por Silvio Stanzani, Raphael Cobe, Rogério Iope (NCC-UNESP) e Igor Freitas (Intel)

Utilização de técnicas de vetorização de código são essenciais para se obter alto desempenho nas arquiteturas computacionais atuais, que vem oferecendo cada vez mais unidades de processamento vetoriais e com maior capacidade. Nesse sentido, é proposto um minicurso básico de três horas com o objetivo de ensinar os conceitos relacionados com vetorizaçäo e técnicas de acesso eficiente a memória, bem como, ferramentas para explorar tais técnicas, usando os recursos oferecidos pelas arquiteturas Multicore e Manycore (Intel Xeon e Intel Xeon Phi).

Minicurso III : Task Parallelism Runtimes and Architectures

por Guido Araujo (IC-UNICAMP)

In recent years, Task-based Programming/Execution model has proven itself a scalable and quite flexible approach to extract regular and irregular parallelism from applications. In this model the programmer uses a task annotation directive to add annotations to code regions in the program which will later be used by a runtime system to manage the parallel execution of such regions. Programming models for task parallelism like OpenSs and OpenMP 4.X have been proposed which easy the task of programming tasks. These models have been supported by runtimes like Intel IOMP, GCC GOMP, and MTSP. In this course we will describe how to use task parallelism to speed-up program execution, and detail the design of runtimes. We will also discuss some optimization techniques like dependency resolution acceleration, to reduce runtime overhead, and task stealing to improve workload balancing. The course will have a short practical session in which the public will be able to play with task programming and runtime design (bring your laptop).

Tutoriais

Tutorial I: Construindo aceleradores com FPGAs para computação de alto desempenho

por Ricardo Menotti (DC-UFSCar)

A computação reconfigurável tem se tornado cada vez mais importante em sistemas computacionais embarcados e de alto desempenho. A chamada computação heterogênea consiste em usar aceleradores de diferentes arquiteturas para melhorar o desempenho e reduzir o consumo de energia dos computadores. Os FPGAs, usados recentemente como aceleradores, permitem níveis de desempenho próximos aos obtidos com circuitos integrados de aplicação específica (ASIC), enquanto ainda mantém flexibilidade de projeto e implementação. No entanto, para programar eficientemente estes dispositivos, é necessária experiência em desenvolvimento e domínio de linguagens de descrição de hardware (HDL), tais como VHDL e Verilog. Neste tutorial serão abordados os principais aspectos da computação reconfigurável, abordagens para reduzir a complexidade de projeto, bem como seu uso recente na computação de alto desempenho.

Tutorial II: O primeiro passo para se programar dez milhões de cores

por Denise Stringhini, Alvaro Fazenda (SJC-UNIFESP)

Recentemente, o ranking das máquinas mais poderosas do mundo, publicada no site Top500.org, apontou a máquina chinesa Sunway Taihu como número um entre as 500 mais rápidas, a qual tem em sua arquitetura mais de dez milhões de cores. Este tutorial apresenta princípios básicos de programação nestas supermáquinas através da introdução da biblioteca de programação paralela MPI (Message Passing Interface). Serão apresentados conceitos de arquiteturas paralelas, assim como as principais rotinas de comunicação da biblioteca MPI em linguagem C, tipos de aplicações e aspectos relacionados ao desempenho de tais programas.

Palestras

Palestra I: Como Utilizar Computação de Alto Desempenho na Nuvem

por Alex Coqueiro (AMAZON)

A computação de alto desempenho (HPC) vem tendo papel fundamental na resolução de problemas complexos de ciência, engenharia e negócios usando uma infra-estrutura que demanda capacidade de computação muito alta. Nesta palestra será compartilhado como diversas entidades de pesquisa em âmbito global tem adotado soluções de nuvem permitindo aumentar a velocidade da pesquisa executando computação de alto desempenho na nuvem e reduzir custos fornecendo servidores de computação em cluster ou GPU de cluster sob demanda sem grandes investimentos de capital. Serão apresentados técnicas de rede totalmente bissecionada e de alta largura de banda para cargas de trabalho altamente acopladas e de I/O intensiva, o que permite aumentar a escala até milhares de núcleos para aplicações orientadas à grande necessidade de performance.

Palestra II: HPC na Pesquisa de Finanças

por Eli Hadad Junior (Mackenzie)

Palestra III: Sistemas Distribuídos para o Controle de Telescópios. SST e LLAMA: Comparações e Experiências

por Guillermo Giménez de Castro (Mackenzie)

Os telescópios modernos, cada vez mais poderosos em sua capacidade observacional, são também sistemas cada vez mais complexos. A complexidade se dá pelo grande número de sistemas que precisam ser controlados simultaneamente e em tempo real. Hoje em dia, o controle completo dos telescópios se realiza por meio de computadores, e, mesmo que os computadores aumentem a sua capacidade de processamento, faz-se necessário adotar uma política de divisão coordenada das tarefas, ou seja, processamento distribuído. Analisamos aqui dois casos concretos cujos design, construção e operação estamos diretamente envolvidos: o Telescópio Solar para Ondas Submilimétricas (SST por sua sigla em inglês) e o Large Latin American Millimeter Array (LLAMA). O SST foi desenvolvido nos anos 90 e opera ininterruptamente desde o ano 2001. LLAMA está atualmente em construção e se espera comece a observar em 2018. Ambos os instrumentos trabalham numa faixa semelhante de frequências do espectro eletromagnético, e têm modos de operação parecidos. Em termos de software de controle, a filosofia é parecida, mas as tecnologias adotadas diferem substantivamente. O sistema de controle do SST é baseado no sistema oparacional "hard real time" QNX, fazendo uso de seu "low latency micro-kernel", "message queuing" e memórias compartilhadas para coordenar a atividade de dezenas de diferentes processos escritos em linguagem C, que vão desde o cálculo das efemérides astronômicas até a leitura e armazenamento de dados digitais. No caso de LLAMA, ele se baseia em uma distribuição Red Hat Linux, junto com o módulo de kernel Real Time Aplication Interface (RTAI), e usa o modelo de contâiner / componente, adotando CORBA como "middleware", para a coordenação dos diferentes processos desenvolvidos nas linguagens C++, Java e Python.

Palestra IV: Rede SINAPAD e o Supercomputador Petaflópico SDumont

por Antonio Tadeu Azevedo Gomes (LNCC)

Nesta palestra, apresentamos as principais iniciativas de e-Ciência sendo realizadas dentro da rede SINAPAD, um programa do Ministério da Ciência, Tecnologia e Inovação que visa oferecer serviços de HPC para a comunidade científica brasileira. Vamos começar a palestra apresentando os recursos disponíveis no SINAPAD, seu perfil de uso e limitações atuais. Continuamos com a apresentação de alguns serviços de e-Ciência que estão sendo oferecidos dentro do SINAPAD para lidar com algumas dessas limitações e alavancar a qualidade da experiência dos usuários. Apresentamos também o Supercomputador SDumont do LNCC, maior recurso de HPC inteiramente destinado a "Open Science" na América Latina, incluindo sua arquitetura e os requisitos esperados pelas aplicações candidatas a executar no mesmo. Concluímos a palestra apresentando as ações de médio e longo prazo que estão sendo planejadas no SINAPAD visando expandir consideravelmente sua capacidade de e-Ciência, não só para HPC, mas também para Big Data.

Palestra V: Energy-Efficient Parallel Software

por Samuel Xavier de Souza (UFRN)

The growth rate of power dissipation has played an important role in the way computational performance has been sustaining growth. The thermal limitations imposed by the power wall made parallelism an essential part of any scalable computation. Although power remains a concern, energy consumption is generally second to performance as application design goal due to the absence of general programming tools aiming to improve energy efficiency. Researchers attempt to fill this gap by enabling programmers to target energy consumption as a primary design goal by delivering static energy consumption estimation from the compiler's point of view. However, for some more complex computations, runtime optimization of the energy efficiency can be crucial. We present ongoing research that targets such runtime approach aiming to allow the programmer to find optimal runtime configurations based on the profiling of parallel performance metrics and on the frequency-power relationship. The symbiotic relationship between static and runtime approaches should boost capabilities and allow programmers to write applications that are up to one order of magnitude more energy efficient. This would not only have a large impact on battery life, in embedded systems, and energy bills, in HPC systems, but also would make energy harvesting more usable, becoming one the key mechanisms to allow zero-energy Internet of Things.

Palestra VI: Computação Paralela e Eu

por Alfredo Goldman (IME-USP)

Quando iniciei minha pesquisa em 1992 havia um grande interesse em redes de interconexão e computação paralela só era possível em máquinas dedicadas. Recentemente, tivemos algumas mudanças de paradigma, onde por vezes a computação de alto desempenho está completamente relacionada a sistemas distribuídos. Traga as suas dúvidas e comentários para discussão!

Análise de Desempenho e Paralelização de Algoritmos para Redes Neurais Profundas

Carlos A. T. Aguni , Alex E. Kawahira , Daniel Cordeiro

[1]Escola de Artes Ciências e Humanidades
Universidade de São Paulo

{carlos.aguni,alex.kawahira,daniel.cordeiro}@usp.br

Abstract. *This paper analyzes the performance of a parallel implementation for the training process of the Back-propagation Algorithm in Deep Neural Network. Our testbed is an SMP machine with an Intel® Xeon Phi Coprocessor (Many Cores architecture). We study its suitability for optimizing a machine learning process, whereas this kind of accelerator card is becoming more affordable and popular among high performance enthusiasts and researchers.*

Resumo. *Este artigo analisa o desempenho do algoritmo de Retropropagação em Redes Neurais Artificiais Profundas implementado de forma paralelizada em uma máquina SMP com uma placa aceleradora Intel® Xeon Phi (arquitetura Many Cores), propondo o estudo e a adequabilidade de seu uso para a otimização de processos de aprendizado de máquina, pois placas aceleradoras estão cada vez mais baratas e mais populares entre os praticantes de computação de alto desempenho.*

1. Introdução

O progresso da ciência da computação inspira cientistas e desenvolvedores do mundo inteiro a investigar novos métodos para responder a seguinte pergunta: como fazer com que computadores "aprendam" sozinhos, ou seja, adquiram conhecimento e sejam capazes de inferir novos conhecimentos a partir de novas informações? Mesmo o computador sendo uma máquina capaz apenas de executar sequências de instruções pré-elaboradas, há métodos cada vez mais sofisticados para fazer com que isso seja possível. Mas antes de falar sobre métodos e algoritmos precisamos primeiro definir o que significa o processo de "aprendizagem" para uma máquina. Em 1959, o pioneiro nas áreas de jogos de computadores, inteligência artificial e aprendizado de máquina Arthur Lee Samuel definiu aprendizado de máquina como uma área que proporciona aos computadores a habilidade de aprender sem serem explicitamente programados.

Um dos métodos mais promissores de aprendizado de máquina é o uso de Redes Neurais Artificiais (RNA) que englobam a classe da Inteligência Artificial dedicada a replicar de forma simplificada o modo que funciona um cérebro biológico [Gershenson 2003]. Logo, a abstração da anatomia em um modelo computacional consistiria em entradas (sinapses) que seriam multiplicadas por pesos (intensidade do sinal) gerando uma saída. O processo de aprendizagem seria, então, refletido no ajuste desses pesos.

O processo de treinamento de uma RNA consiste em "treinar" a rede com entradas e ajustar os pesos a partir do resultado obtido. Logo, o problema reside quando há a intenção de se testar a rede em conjuntos maciços de dados (megadados ou conjuntos

de dados não estruturados). Computacionalmente, o problema é complexo. O tempo de execução é diretamente proporcional ao tempo de execução. Em *RNAs Profundas*, uma variante com múltiplas camadas, o problema é ainda mais evidente.

Para fins práticos, é preciso amenizar esse problema para que ele se torne tratável. Uma opção é utilizar as tecnologias de computação paralela conhecidas como *Many Core computing*. Placas aceleradoras de propósito geral (tais como GPUs ou, mais recentemente, a arquitetura Intel® Xeon Phi) permitem a execução de *muitas* tarefas concorrentes [Fang et al. 2013, Lee et al. 2015]. Ao contrário da computação paralela clássica, que prioriza a execução de cada tarefa em uma CPU no menor tempo possível, *Many Core computing* prioriza a execução do maior número de tarefas por unidade de tempo.

2. Redes Neurais de Retropropagação

A RNA utilizada neste artigo é a *Perceptron Múltiplas Camadas* (*Multi Layer Perceptron* - PMC), que consiste em neurônios organizados em pelo menos três camadas (entrada, ocultas e saída) cada uma totalmente conectada às suas adjacentes. PMCs podem ser compreendidas como um mapeamento funcional de um local de entrada para uma saída.

Um tipo de PMC é a Rede Neural de Retropropagação, onde pares de entrada/saída são treinados através de um algoritmo de dois passos supervisados, baseados na regra de aprendizado de correção de erro [Rumelhart et al. 1986].

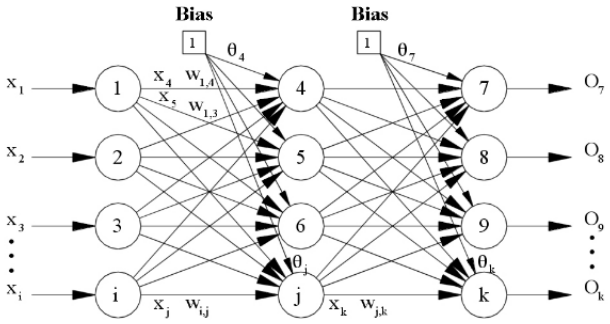


Figura 1. Rede Neural de Retropropagação com uma camada oculta [Lu 2000].

Primeiro passo – Avanço/Ativação: um vetor de entrada é aplicado à primeira camada da rede, que por sua vez propaga para o restante da rede, uma camada de cada vez. Para as camadas seguintes depois da camada de entrada, em cada neurônio é calculada uma soma ponderada dos sinais vindos da camada anterior para produzir uma nova rede de entrada e aplicar uma função de ativação não-linear contínua (geralmente em forma sigmoide) para determinar seu valor da saída. O vetor de saída da rede é o vetor de ativação da camada final de neurônios.

Segundo passo – Retropropagação/Correção: tentativa de correção dos erros em relação ao mapeamento desejado inicialmente. O erro geral da rede de um dado vetor é calculado comparando o vetor de saída com um vetor de resultado esperado. Cada camada oculta deve calcular a contribuição de seus neurônios para o erro final, este resultado é alcançado com cada neurônio em uma camada calculando seu próprio erro de uma soma ponderada dos erros das camadas seguintes. Em seguida o peso de cada neurônio é ajustado proporcionalmente ao seu erro.

A fase de treinamento se estende sobre um número de épocas até que o valor médio de erro desta população fique abaixo de um valor mínimo definido, neste momento, o treinamento é interrompido e é dito que a rede atingiu a convergência. Ao convergir, a rede é capaz de generalizar corretamente vetores de entradas ainda desconhecidos [Pethick et al. 2013].

3. Estratégia de Paralelização

Estudamos o uso de uma placa aceleradora para paralelizar o problema. Utilizamos um computador[1] de memória compartilhada com dois processadores Intel® Xeon CPU E5-2650 de 2.00GHz, 16 GB RAM (*host*) e uma placa aceleradora Intel® Xeon Phi 5110P, um coprocessador de arquitetura *Many Core* de memória compartilhada com 61 cores, cada um com uma frequência de 1.053 GHz. Há dois modos de operação:

- *native*: o código é executado nativamente no coprocessador, todo o código e suas dependências precisam ser carregadas no dispositivo;
- *offload*: a aplicação é executada no *host* e partes do código são carregadas no coprocessador. Este é o modo utilizado neste artigo, portanto a localidade de dados é essencial para um desempenho satisfatório do coprocessador.

Utilizamos o algoritmo e o conjunto de testes disponibilizado publicamente em `https://github.com/buhpc/dnn` como base para a implementação. Durante o processo de aprendizagem, o algoritmo sequencial realiza iterativamente as etapas de *ativação*, *retropropagação* e *correção dos pesos*. O algoritmo é implementado na linguagem C e a biblioteca OpenMPI foi utilizada para a implementação em paralelo.

Porém há outra abordagem com mesma garantia de convergência onde a correção dos pesos é feita somente após realizadas as etapas de ativação e retropropagação de todo o conjunto de treinamento. Dessa forma é possível executar concorrentemente o processo de ativação e retropropagação de cada teste em *threads* independentes.

O algoritmo paralelo foi ajustado manualmente para que as tarefas paralelas realizadas na CPU (*host*) fossem executadas na mesma quantidade de tempo que as tarefas realizadas na Xeon Phi, garantindo, assim, o balanceamento de carga entre as unidades de processamento. A porcentagem de carga de trabalho delegada para o Xeon Phi pode ser vista na Figura 2.

Cada processo mantém, então, uma cópia da matriz de pesos e aplica a correção em uma estrutura secundária ΔW_{CPU} e ΔW_{Phi}. Por fim o *host* se encarregaria de calcular os pesos finais:

$$\Delta W_{\text{Final}} = \Delta W_{\text{CPU}} + \Delta W_{\text{Phi}}$$

4. Resultados experimentais

Foram realizados testes empíricos variando o balanceamento de carga de processo entre processador e coprocessador para encontrar o ponto de maior eficiência da aplicação.

O tempo de execução do código em serial foi de aproximadamente 454s. No melhor caso de teste (com 83,3% da carga executada na Phi) o tempo foi de 42s, ou seja, um *speedup* em torno de 10 vezes mais rápido em relação à implementação serial.

[1]Agradecemos a Profª. Drª. Liria M. Sato (POLI/USP) pelo acesso a esta máquina.

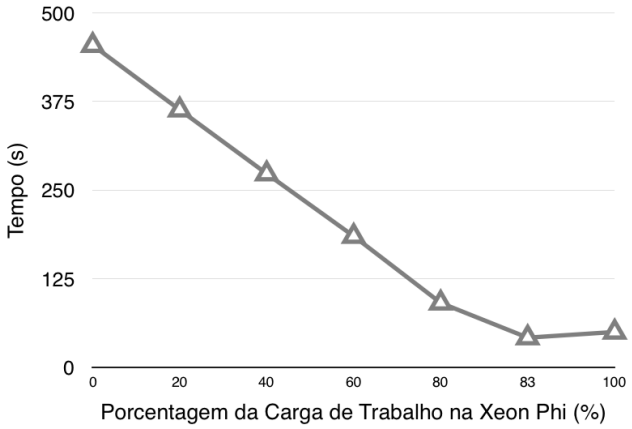


Figura 2. Gráfico de Tempo x Carga de Trabalho na Xeon Phi.

5. Considerações finais

Este trabalho apresenta um método de paralelização para a etapa de treinamento de Redes Neurais de Retropropagação, um tipo de Rede Neural Profunda. Este método explora o uso de placas aceleradoras em uma arquitetura *Many Core*.

Adaptamos a etapa de treinamento deste tipo de rede neural para que fosse possível paralelizá-la em uma placa aceleradora do tipo Intel® Xeon Phi. Mais precisamente, modificamos o cálculo de correção de pesos para que fosse realizado como um ponto de sincronização posterior à iteração de treinamento, fazendo com que cada lote de dados de treinamento fosse processado concorrentemente.

Nossa análise experimental mostrou que foi possível obter um *speedup* de 10 ao executar 83,3% da carga de trabalho nos coprocessadores. Em trabalhos futuros investigaremos também o uso de unidades de processamento gráfico de propósito geral.

Referências

Fang, J., Varbanescu, A. L., Sips, H. J., Zhang, L., Che, Y., and Xu, C. (2013). An empirical study of Intel Xeon Phi. *CoRR*, abs/1310.5842.

Gershenson, C. (2003). Artificial neural networks for beginners. *CoRR*, cs.NE/0308031.

Lee, D., Kim, K.-H., Kang, H.-E., Wang, S.-H., Park, S.-Y., and Kim, J.-H. (2015). Learning speed improvement using multi-gpus on dnn-based acoustic model training in korean intelligent personal assistant. In *Natural Language Dialog Systems and Intelligent Assistants*, pages 263–271. Springer.

Lu, W. (2000). Neural network model for distortion buckling behaviour of cold-formed steel compression members. *Helsinki University of Technology Laboratory of Steel Structures Publications*, 16.

Pethick, M., Liddle, M., Werstein, P., and Huang, Z. (2013). Parallelization of a backpropagation neural network on a cluster computer.

Rumelhart, D., Hinton, G., and Williams, R. (1986). Learning internal representations by error backpropagation. *Parallel Distributed Processing: Explorations into the Microstructure of Cognition*, 1.

Simulação de PaaS no iSPD

João Antonio Magri Rodrigues[1], **Aleardo Manacero Junior**[1], **Renata Spolon Lobato**[1], **Roberta Spolon**[2]

[1]Depto. de Ciências de Computação e Estatística - DCCE
Universidade Estadual Paulista "Júlio de Mesquita Filho" - UNESP
São José do Rio Preto - SP - Brasil

[2]Depto. de Computação - DCo
Universidade Estadual Paulista "Júlio de Mesquita Filho" - UNESP
Bauru - SP - Brasil

joaoa.mr@hotmail.com, {aleardo,renata}@ibilce.unesp.br, roberta@fc.unesp.br

Abstract. *Cloud computing is a term used for computing services offered over the internet. These services cover from simple personal files storage up to servers hosting, being divided into classes accordingly the type of the offered service. One of these classes is known as PaaS, from Platform as a Service, where the provider offers specific operating environments for user's tasks. Besides the use of PaaS is attractive, it is needed to evaluate the efficiency of this service for a given class of tasks. In this work we present a new functionality included to iSPD in order to simulate and evaluate a PaaS service. Results show the efficacy of such simulations.*

Resumo. *Computação em nuvem é um termo usado para serviços computacionais oferecidos pela Internet. Esses serviços abrangem desde armazenamento de arquivos pessoais à hospedagem de servidores, sendo separados em classes de acordo com o tipo de serviço oferecido. Uma destas classes é a de PaaS, de* Platform as a Service*, em que provedores oferecem ambientes computacionais específicos para as tarefas dos usuários. Apesar da atratividade em se usar PaaS, é preciso avaliar sua eficiência para uma dada classe de tarefas. Neste trabalho apresentamos uma nova funcionalidade incluída ao iSPD para simular e avaliar serviços PaaS. Resultados sobre a eficácia dessas simulações são apresentados.*

1. Introdução

Computação em nuvem é o nome dado para sistemas formados por uma coleção de máquinas virtuais interconectadas e provisionadas dinamicamente, com sua disponibilidade baseada em um contrato de nível de serviço [Buyya et al. 2011]. A forma de disponibilizar recursos ocorre através de três classes para ambientes de nuvem [Rittinghouse and Ransome 2009], definidas como:

- ***Software as a Service (SaaS)***, em que aplicações residem no centro de serviço do seu provedor e são acessadas remotamente por seus clientes;
- ***Platform as a Service (PaaS)***, em que o provedor disponibiliza ao cliente um ambiente para desenvolvimento e implantação de aplicações com um alto nível de abstração, de forma que o cliente não necessite conhecer detalhes da plataforma;

- ***Infrastructure as a Service (IaaS)***, em que o provedor oferece o menor nível de abstração ao cliente, que fica responsável por configurar e gerenciar recursos físicos do sistema.

Em função da complexidade e diversidade desse tipo de sistema é necessário ter ferramentas capazes de realizar a análise de seu desempenho, sendo que simuladores são de eficientes nessa tarefa. Para computação em nuvem existem simuladores projetados para isso, tais como *CloudSim* [Buyya et al. 2009], *iCanCloud* [Castañé et al. 2012], e *GreenCloud* [Kliazovich et al. 2010].

O problema nesses simuladores é a sua relativa dificuldade de uso, uma vez que dependem de um maior conhecimento de sua programação. O iSPD (*iconic Simulator of Parallel and Distributed Systems*) [Menezes et al. 2012] é um simulador de grades computacionais desenvolvido como uma interface simples para modelagem e que é capaz de simular serviços de computação em nuvem da classe IaaS. A versão apresentada aqui trata a implementação de novos módulos a fim de simular sistemas da classe de serviço PaaS.

2. Trabalhos relacionados

O uso de simulação para avaliação de desempenho é um fator já reconhecido. Em computação em nuvem isso não é diferente. Temos, por exemplo, o *CloudSim* [Buyya et al. 2009], desenvolvido pelo grupo CLOUDS da Universidade de Melbourne. Nele se pode modelar elementos como máquinas físicas, máquinas virtuais, aplicações, políticas de escalonamento, etc.

Já o *iCanCloud* [Castañé et al. 2012] é uma plataforma desenvolvida pelo grupo ARCOS da universidade Carlos III de Madrid. O simulador foi desenvolvido em OMNet++ e INET [Varga and Hornig 2008] e seu objetivo é realizar avaliações, voltadas para a classe de serviço IaaS, da relação de custo e desempenho de aplicações executadas nos recursos do sistema modelado.

Por outro lado, o *GreendCloud* [Kliazovich et al. 2010], desenvolvido pela Universidade de Luxemburgo, é uma extensão do simulador de redes NS2, possuindo implementação híbrida. A maior parte do seu código foi escrita em C++ e o restante em scripts Tcl (*Tool Command Language*). O simulador é voltado para a avaliação de gastos de energia elétrica dos sistemas simulados.

Uma característica desses simuladores é que demandam de seus usuários conhecimentos específicos em linguagens de programação para a produção de modelos. A proposta do iSPD é erradicar esse problema, oferecendo ao seu usuário uma interface gráfica de fácil utilização para a construção de modelos de simulação e visualização de resultados.

3. Inclusão de PaaS no iSPD

O iSPD (*iconic Simulator of Parallel and Distributed Systems*) [Menezes et al. 2012] é um simulador de grades computacionais e sistemas de computação em nuvem desenvolvido pelo GSPD (Grupo de Sistemas Paralelos e Distruibuídos) da UNESP. Seu objetivo é facilitar a modelagem de ambientes de computação distribuída, oferecendo ao usuário uma interface gráfica intuitiva.

O funcionamento do iSPD está baseado em cinco módulos, sendo que os principais são: **Interface icônica**, em que o usuário realiza a modelagem do sistema a ser simulado; **Interpretação de modelos internos**, responsável por converter os parâmetros de modelos icônicos para modelos de filas; e o **Motor de simulação**, que executa a simulação propriamente dita. As principais modificações para fazer a simulação de PaaS envolvem esses três componentes.

Quanto a interface gráfica, foram adicionados novos componentes que possibilitam ao usuário modelar o serviço de PaaS por meio de uma janela para a inserção de dados. Esses componentes envolvem a configuração de alguns dos parâmetros relativos às máquinas virtuais, como seu custo por unidade de tempo. Também se acrescentou um mecanismo para caracterizar a carga de trabalho a ser usada na simulação como sendo de aplicações específicas, em vez da caracterização original do iSPD, que tratava tarefas apenas como carga computacional.

Ao módulo de interpretação de modelos internos foram adicionados parâmetros relativos ao modelo de simulação PaaS. Isso implica em mudanças na forma de *parsing* dos modelos icônicos. Também implica no acréscimo de parâmetros para a definição de filas no sistema, tais como objetos para provisionamento dinâmico, custos, etc.

Para o motor de simulação, foi adicionado uma nova política de escalonamento, para fazer a alocação ou retirada de máquinas virtuais e realizar o balanceamento de cargas das mesmas durante o processo de simulação. Com isso o motor de simulação pode tratar aspectos de *autoscaling* na alocação das tarefas.

4. Testes e Resultados

Para a validação das novas funcionalidades foram realizados testes comparativos com o CloudSim. Foram consideradas métricas como número máximo de instâncias alocadas, tempo total de execução e custo do serviço, sendo que o custo total de execução da aplicação simulada é calculado pela somatória do preço de cada máquina virtual multiplicado pelo tempo em que esteve ativa. Para o ambiente de nuvem em PaaS se considerou:

- **Aplicação:** composta por tarefas com 100000 Mflop de carga computacional cada, sendo que sempre que uma MV tiver 2000 tarefas um sua fila de execução ocorrerá uma operação de *autoscaling*;
- **Recursos:** cada servidor tem capacidade de processamento de 50000 Mflops e o ambiente físico pode acomodar quantas máquinas virtuais forem necessárias. O custo de cada máquina virtual foi definido em 1$/segundo.

Um resumo das simulações é mostrado na Tabela 1. As colunas 2 e 5 mostram, respectivamente para o iSPD e para o CloudSim, o número de máquinas virtuais disparadas pelo escalonador, na operação de *autoscaling*. As colunas 3 e 6 mostram o tempo simulado de execução de cada conjunto de tarefas, sendo evidente que o aumento no número de máquinas virtuais manteve o tempo de execução relativamente constante, pois as MV's processam no máximo duas mil tarefas cada. As colunas 4 e 7 apresentam os custos do uso das máquinas virtuais.

Em todos os testes o iSPD apresentou grande similaridade de resultados com o *CloudSim*. No que diz respeito ao *autoscaling* é possível notar que há um esforço do sistema em manter constante o tempo total de execução da aplicação quando se aumenta o número de tarefas, implicando no aumento do custo total de uso do sistema.

Tabela 1. Resultado de teste comparativo entre iSPD e CloudSim

	iSPD			CloudSim		
Tarefas	Instâncias	Tempo	Custo	Instâncias	Tempo	Custo
3500	2	4008	8011,8	2	4006	8012,1
4500	3	4014	12025,1	3	4006	12018,1
5500	3	4010	11001,8	3	4006	12018,1
6500	4	4016	16040,9	4	4006	16024,1
7500	4	4012	16029,9	4	4006	16024,1
8500	5	4018	20057,6	5	4006	20030,1
9500	5	4012	19008,3	5	4006	20030,1
10500	6	4018	21001,0	6	4006	24036,1
11500	6	4014	23002,0	6	4006	24036,1
12500	7	4020	25002,1	7	4006	28042,1

5. Considerações Finais

Com a introdução de novas funcionalidades capazes de implementar uma política de provisionamento dinâmico juntamente com um escalonador específico é possível realizar a simulação de um serviço de PaaS no iSPD, oferendo ao usuário agilidade e facilidade de uso para modelagem do sistema e visualização de resultados.

Em relação aos resultados obtidos, o iSPD se mostrou preciso em todos os quesitos ao comparar seus resultados com *CloudSim*.

Referências

Buyya, R., Broberg, J., and Goscinski, A. (2011). *Cloud Computing: Principles and Paradigms*. Wiley.

Buyya, R., Ranjan, R., and Calheiros, R. N. (2009). Modeling and simulation of scalable cloud computing environments and the CloudSim toolkit: Challenges and opportunities. *CoRR*, abs/0907.4878.

Castañé, G. G., Núñez, A., and Carretero, J. (2012). iCanCloud: A brief architecture overview. In *2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications*, pages 853–854.

Kliazovich, D., Bouvry, P., Audzevich, Y., and Khan, S. U. (2010). GreenCloud: A packet-level simulator of energy-aware cloud computing data centers. In *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, pages 1–5.

Menezes, D., Manacero, A., Lobato, R., da Silva, D., and Spolon, R. (2012). Scheduler simulation using iSPD, an iconic-based computer grid simulator. In *Computers and Communications (ISCC), 2012 IEEE Symposium on*, pages 000637 –000642.

Rittinghouse, J. and Ransome, J. (2009). *Cloud Computing: Implementation, Management, and Security*. CRC.

Varga, A. and Hornig, R. (2008). An overview of the omnet++ simulation environment. In *Proc. of the 1st Intl Conf on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, Simutools'08, pages 19:1–19:10. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Detecção Distribuída de Programas Maliciosos para Dispositivos Móveis em Ambientes de Computação em Nuvem

Emanuel Carlos A. Valente[1], **Henrique Yoshikazu Shishido**[1],**Júlio Cezar Estrella**[1]

[1]Instituto de Ciências Matemáticas e de Computação – Universidade de São Paulo (USP)
Caixa Postal 13.566-590 – São Carlos – SP – Brazil

emanuel.valente@usp.br,shishido@usp.br,jcezar@icmc.usp.br

Abstract. *The amount of mobile devices around the world has surpassed the range of 7 billion, and the distribution ratio of malicious applications has also increased significantly. However, researches of threat detection methods on these devices did not follow the same pace. This paper presents a proposal for a distributed detection methodology, optimized for large volumes of malicious programs for mobile devices stored in the cloud using the framework Apache Hadoop. This method has as target application installers for the Android operating system. The results indicate a precision rate of 82.5% and a success rate of 76.13 % to detect malicious files.*

Resumo. *O número de dispositivos móveis em todo o mundo já ultrapassou a faixa de 7 bilhões e o índice de distribuição de aplicativos maliciosos também tem aumentado significativamente. No entanto, a pesquisa de métodos de detecção de ameaças nestes dispositivos não acompanhou o mesmo ritmo. Este trabalho propõe uma metodologia de detecção distribuída otimizada para grandes volumes de programas maliciosos para dispositivos móveis armazenados na Nuvem utilizando o framework Apache Hadoop. Esse método possui como alvo arquivos instaladores de aplicativos para o sistema operacional Android. Os resultados indicam uma precisão de 82,5% e uma taxa de acerto de 76,13% para detecção de arquivos maliciosos.*

1. Introdução

Desde o início dos anos 2000, a distribuição de programas maliciosos (*malware*) não esteve mais relacionada a grupos especializados e isolados como nas décadas anteriores. Atualmente, a distribuição de *malware* é baseada no sistema de monetização *PPI* (*Pay-Per-Install*) [Caballero et al. 2011]. Com a grande oferta de serviços de Computação em Nuvem (*Cloud*), provedores *PPI* vem utilizando serviços na *Cloud* como meio de distribuição de *malware*. Paralelamente, há um crescimento do número de usuários de dispositivos móveis. Em 2014, o total desses aparelhos ao redor do mundo era de aproximadamente 7,1 bilhões [Ericsson 2015] dos quais 2,1 bilhões eram dispositivos do tipo *smartphones* equipados com o sistema operacional *Android*.

Atraídos pelo aumento de usuários móveis, cibercriminosos vem adaptando o modelo de distribuição de *malware* para suportar plataformas móveis com ênfase nos sistemas *Android* e *IOS*. Em 2014, cerca de 98% do total de programas maliciosos para *smartphones* tinham como alvo o sistema *Android* [Storm 2014]. Em abril do mesmo ano, aproximadamente um em cada três instaladores de aplicativos *Android* armazenados em lojas não oficiais eram maliciosos [Hern 2014].

Existe, portanto, uma demanda por métodos de detecção de aplicativos móveis maliciosos que estão armazenados na Nuvem, não só para evitar a distribuição de *malware*, mas para detectar perfis de usuários mal-intencionados que utilizam esse tipo de serviço. Este trabalho propõe uma metodologia de detecção distribuída baseada no framework *Apache Hadoop*[1], otimizada para grandes volumes de arquivos armazenados em um sistema de arquivos distribuído, como o HDFS (*Hadoop Distributed File System)*. Adicionalmente, o método proposto possui flexibilidade para ser aplicado em Computação em Nuvem, pois o sistema de arquivos distribuído pode ser criado sob demanda a qualquer momento baseado nos nós ativos em uma infraestrutura de uma determinada Nuvem. A única restrição é de que todos os nós devem fornecer acesso ao *Hadoop* via serviço *SSH (Secure Shell)*.

2. Métodos

Foram considerados dois grupos de amostras: maliciosas e não maliciosas de aplicativos do sistema *Android*. Cada amostra é constituída de um arquivo instalador, cuja extensão é *'APK'* (*Android Application Package*)[2]. O interesse em cada amostra é de extrair as permissões de acesso requeridas por cada aplicativo durante a instalação, como acesso à câmera, microfone, leitura de *sms* e outras. Por padrão, a plataforma *Android* define essas permissões no próprio arquivo *APK*, em um arquivo de extensão *XML* chamado '*AndroidManifest.xml*'.

Todas as permissões de ambos os grupos foram modeladas como grafos em busca de padrões de acesso requeridos por cada aplicativo durante sua instalação. Dessa forma, a rotina de detecção desenvolvida neste trabalho trata-se de um classificador baseado nos subgrupos de permissões mais acessadas e distintas para cada um dos dois grupos modelados. As duas próximas subseções detalham o processo de análise das amostras e o desenvolvimento do classificador para a plataforma *Hadoop*.

2.1. Análise das Amostras

Para os dois grupos de amostras foi gerado um grafo, do qual cada nó é representado por cada permissão de acesso exigida para a instalação. A modelagem pode ser descrita da seguinte forma: suponha que um aplicativo de um dos grupos solicite permissão de leitura de *sms*, permissão para envio de sms e permissão de leitura do cartão de memória. Isso resultará em um grafo completo de três nós, onde cada nó representa uma permissão. Caso outro aplicativo do mesmo grupo solicite as mesmas permissões anteriores, será gerado um novo grafo idêntico ao anterior. Os grafos idênticos serão representados por um único grafo, cujo peso das arestas será igual ao número de repetições da permissão. Ao final da análise, tem-se um grafo geral para representar um dos grupos.

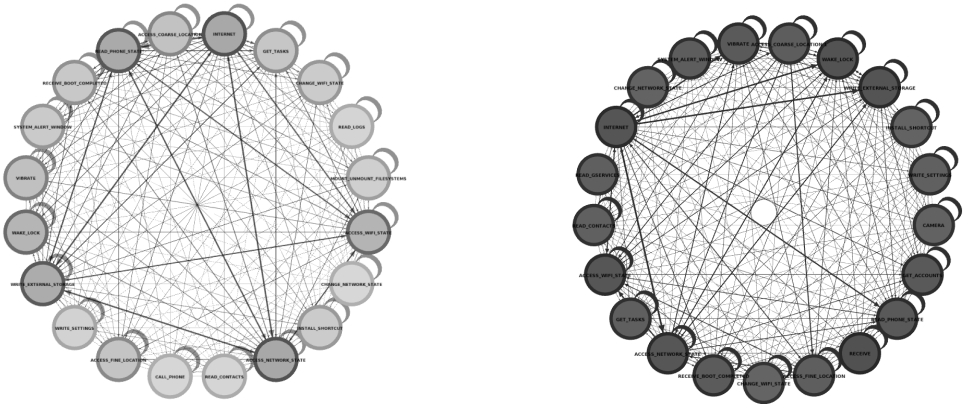
As figuras 1a e 1b ilustram os subgrafos extraídos dos dois grafos resultantes. Ambos possuem os nós de maior grau de seus respectivos grafos principais. Ou seja, cada subgrafo da figura 1 representa o conjunto das permissões requeridas mais acessadas por todas amostras de cada grupo.

2.2. Desenvolvimento do Classificador

O classificador foi desenvolvido baseado nos subgrupos de permissões mais acessadas durante a análise. Sua função é de verificar todas as permissões de acesso requeridas para cada arquivo APK a ser classificado como malicioso ou não. Cada vez que uma permissão é

[1] http://hadoop.apache.org

[2] https://developer.android.com/studio/run/index.html



(a) Subgrafo - Arquivos maliciosos

(b) Subgrafo - Arquivos não maliciosos

Figure 1: Subgrafos (nós de maior grau) extraídos durante a análise das permissões das amostras.

encontrada (i.e., *match*) em um dos subgrupos de permissões maliciosas ou não maliciosas soma-se ou subtrai-se de um (ponto), respectivamente. Dessa maneira, a saída do classificador para cada arquivo é a somatória total da pontuação de classificação. Se a somatória possuir valor positivo, presume-se que o arquivo tem maiores chances de ser malicioso, caso contrário, menores chances de oferecer algum risco ao usuário.

Posteriormente, o classificador foi adaptado para uma aplicação seguindo o paradigma *MapReduce* com duas fases de *map* e *reduce*. A primeira fase é responsável por extrair o arquivo *AndroidManifest.xml* de cada arquivo *APK*, convertê-lo para o formato *ASCII* e armazená-lo no sistema de arquivos distribuído. A segunda fase tem como entrada os arquivos *AndroidManifest.xml* recém-extraídos, que serão comparados com as permissões dos dois subgrupos mais acessados (ver seção 2.1). O resultado final da aplicação *MapReduce* é uma lista de arquivos, seus respectivos *hashes* e a somatória calculada pelo classificador. A lista de arquivos resultante permite detectar usuários mal-intencionados, pois o sistema de arquivo distribuído permite que os respectivos proprietários dos arquivos sejam também listados.

3. Resultados

Para testar o processo de treinamento da aplicação foi utilizada uma base de 42.000 amostras de aplicativos móveis maliciosas obtidas no portal VirusShare[3], e 40.000 amostras não maliciosas obtidas em quatro sites[4]. As amostras foram verificadas como não maliciosas pelo serviço VirusTotal[5].

Como o número de amostras é significativo, decidiu-se escolher o método *Holdout* para o treinamento e teste do classificador. Separou-se $\frac{2}{3}$ das amostras para treinamento e o restante para a realização de teste. A tabela 1 apresenta a matriz de confusão para o resultado dos testes de detecção realizados em uma infraestrutura do *LaSDPC - ICMC/USP* [6] para 14.000 arquivos maliciosos e 13.333 arquivos não maliciosos ($\frac{1}{3}$ de cada amostra).

[3]https://virusshare.com/

[4]http://dl2.melidl.com/android/, http://cirrus.turtil.net/pi2/apk/, http://dl2.download.ir/, http://share724.com

[5]http://virustotal.com

[6]http://infra.lasdpc.icmc.usp.br/cosmos

Matriz de Confusão para classificação		
Tipo da Amostra	**Classificadas como Maliciosas**	**Classificadas como não maliciosas**
Maliciosas	10658	3342
Não Maliciosas	2267	11066

Table 1: Matriz de Confusão

Para realização dos testes foram utilizados dois nós para os dados (*datanode*) e um para os metadados (*namenode*).

Com base nos dados da matriz de confusão, as medidas de acurácia, precisão e *recall* são, respectivamente 0,795; 0,825 e 0,761.

4. Conclusões

Este trabalho apresentou uma abordagem para a detecção de arquivos maliciosos do sistema *Android* baseada em permissões de acesso solicitadas no momento de sua instalação. As contribuições desta abordagem são: (a) desenvolvimento de um método para treinamento e classificação de arquivos maliciosos e não maliciosos; (b) adaptação do framework Apache Hadoop para o método de detecção distribuída de arquivos maliciosos adaptável para ambientes de Computação em Nuvem e (c) detecção de usuários mal-intecionados na Nuvem.

A taxa média de acerto de detecção de arquivos maliciosos foi de 76% e a precisão de 82.5%. É importante levar em consideração de que para melhorar a eficiência do classificador é necessário uma abordagem mais tradicional de detecção, como por exemplo, o uso de assinaturas e/ou verificação de *hashes* dos arquivos, o que demanda um maior processamento para cada análise, aumentando o tempo de resposta. Entretanto, como ambientes de Computação em Nuvem podem armazenar milhares ou até milhões de instaladores de aplicativos móveis, a minimização do processamento é crucial no processo de detecção de grandes volumes de arquivos. Dessa forma, por exigir poucas comparações, podendo ser facilmente paralelizado e/ou distribuído, o método proposto acaba sendo eficaz e eficiente para grandes volumes de arquivos.

5. Agradecimentos

Agradecemos à FAPESP (Processo: 2015/26720-5) pelo apoio financeiro e ao LaSDPC [7] pela infraestrutura oferecida durante o desenvolvimento do projeto.

References

Caballero, J., Grier, C., Kreibich, C., and Paxson, V. (2011). Measuring pay-per-install: The commoditization of malware distribution. In *Proceedings of the 20th USENIX Security Symposium*, pages 187–203. Usenix.

Ericsson, I. (2015). Ericsson Mobility Report. http://www.ericsson.com/res/docs/2015/mobility-report/ericsson-mobility-report-nov-2015.pdf. (Acessado em 21 de maio de 2016).

Hern, A. (2014). One in three android apps on non-google stores are malicious, study finds. [Online; posted 18-April-2014].

Storm, D. (2014). 98% of mobile malware targets android platform. http://www.computerworld.com/article/2475964/mobile-security/98–of-mobile-malware-targets-android-platform.html.

[7] Página oficial do LaSDPC: `http://lasdpc.icmc.usp.br/`

Análise experimental sobre o impacto de funções hash na ocorrência de falsos conflitos em memória transacional

Bruno Chinelato Honorio[1], **Alexandro José Baldassin**[1]

[1]Departamento de Estatística, Matemática Aplicada e Computação –
Universidade Estadual Paulista (UNESP)
Rio Claro – SP – Brazil

brunochonorio@gmail.com, alex@rc.unesp.br

Abstract. *Transactional memory is a new concurrent programming mechanism that can be implemented in hardware, software or in a hybrid form. One of the main concerns about this mechanism is its high computational cost, particularly in software implementations. This work analyses the impact of different hash functions on the performance of software transactional memory systems that make use of a lock table. The results collected through microbenchmarks show that the choice of a hash function has a big impact on the final system performance.*

Resumo. *A memória transacional é um novo mecanismo de programação concorrente que pode ser implementado em hardware, software ou de forma híbrida. Uma das grandes preocupações sobre esse mecanismo é o seu elevado custo computacional, especialmente em implementações em software. Este trabalho analisa o impacto de diferentes funções hashes no desempenho de sistemas transacionais em software que utilizam uma tabela de bloqueios. Os resultados adquiridos através de microaplicações mostram que a escolha da função hash tem um grande impacto no desempenho final do sistema.*

1. Introdução

Mecanismos de sincronização em programação concorrente baseiam-se essencialmente em primitivas como bloqueios e variáveis de condição, que foram elaboradas há muitas décadas, evidenciando uma ausência de novos modelos de programação que ajudem os programadores no desenvolvimento de aplicações paralelas [Sutter and Larus 2005]. A memória transacional (*transactional memory* – TM) é uma nova abordagem para o desenvolvimento dessas aplicações e é considerada uma das mais promissoras para auxiliar no desenvolvimento da escrita de código paralelo [Harris et al. 2010].

A essência de TM é a de transação, ou seja, existe uma sequência de instruções que é executada de forma atômica, sendo que todas as instruções devem ter suas alterações efetivadas no sistema ou, em caso contrário, as alterações realizadas não são efetivadas e a transação é abortada. O suporte transacional pode ser implementado em software (STM), hardware (HTM) ou de forma híbrida (HyTM). Embora abordagens em hardware tendam a obter melhor desempenho, elas geralmente não conseguem executar transações com tamanhos arbitrários. Em contrapartida, um dos grandes problemas com implementações em software é seu elevado custo computacional, já que todos os acessos (leitura e escrita) à variáveis compartilhadas precisam ser instrumentados.

O objetivo desse trabalho é analisar o desempenho de algoritmos para memória transacional baseados em uma tabela de bloqueios. Para isso, foi usado o algoritmo TL2 (Transacional Locking II) [Dice et al. 2006] por ser considerado um dos algoritmos mais eficientes de memória transacional em software. A análise apresentada nesse trabalho visa detectar o impacto de diferentes funções hash, usadas para fazer o mapeamento entre endereços de variáveis compartilhadas e bloqueios (veja Seção 2 para mais detalhes). Os experimentos foram realizados com um conjunto de microaplicações, que reportam diferentes desempenhos para as diferentes funções hash empregadas. Esse resultado indica que a escolha da função hash é importante para um bom desempenho. O artigo é dividido da seguinte forma: a Seção 2 apresenta a descrição do problema, a Seção 3 descreve a metodologia empregada, e a Seção 4 apresenta os resultados obtidos e conclusão.

2. Descrição do Problema

Existe uma classe de algoritmos transacionais que baseiam-se em bloqueios para garantir a consistência da execução. Um desses algoritmos, utilizados neste trabalho, chama-se TL2 (*Transactional Locking II*) [Dice et al. 2006]. No algoritmo, cada transação mantém localmente um descritor que armazena seu estado atual, a versão que iniciou a transação, e os conjuntos de dados lidos e escritos (chamados de conjuntos de leitura e escrita, respectivamente). Cada elemento do conjunto de leitura é um endereço de memória lido pela transação. Além do endereço de memória, cada elemento do conjunto de escrita precisa armazenar o valor a ser escrito, porque a atualização só é efetuada no momento da efetivação. Toda posição de memória lida e escrita durante a transação é mapeada, através de uma função hash, para uma entrada em uma tabela chamada de ORT (*Ownership Record Table*). O conteúdo de cada elemento da tabela depende de seu bit menos significativo, o bit de bloqueio. Se esse bit for zero, então o conteúdo representa a versão corrente de todas as posições de memória mapeadas para a entrada. Caso contrário, a entrada está bloqueada e o seu conteúdo é um ponteiro para o descritor da transação que tem posse da respectiva entrada da tabela.

Pelo fato da estrutura utilizada ser uma tabela hash, há problemas de colisão: dois endereços de memória distintos podem ser mapeados para a mesma entrada, potencialmente gerando o chamado *falso conflito*. Isso acontece dependendo de como a função hash foi confeccionada. A função hash pode não ser eficiente em espalhar os endereços de memória na ORT, levando os endereços de memória que devem ser lidos por diferentes transações a serem mapeados em uma mesma posição da ORT o que, por sua vez, leva a duas transações tentando acessar a mesma posição na ORT, mesmo que os endereços acessados sejam diferentes. Nesse trabalho busca-se fazer uma análise através da avaliação do desempenho de diferentes funções hash, como apresentado na próxima seção.

3. Metodologia Experimental

Para a análise apresentada neste trabalho foram usadas as funções hash apresentadas pelas Equações 1, 2 e 3. Primeiramente, note que todas as funções são limitadas pelo tamanho da ORT ($ORTsize$). A Equação 1 é a padrão sugerida pela implementação do TL2, associando blocos consecutivos de 32 bytes à mesma posição na ORT. A Equação 2 é comumente sugerida em livros-texto e realiza uma operação XOR ($\oplus$) entre os 32 bits mais significativos e o 32 menos significativos do endereço (assumindo uma arquitetura de 64 bits) para obter uma posição na ORT. Finalmente, a Equação 3 apresenta a função

hash multiplicativa de Knuth [Knuth 1997], a qual descarta os bits menos significativos do endereço e multiplica o resultado por um número aleatório. Os bits mais significativos dessse produto são usados para obter uma posição na ORT. $ORTbits$ refere-se ao número de bits necessário para indexar a tabela ORT (no estudo foram utilizados 20 bits).

$$hdefault(addr) = addr >> 5 \mod ORTsize \quad (1)$$

$$hxor(addr) = (Lower\_addr) \oplus (Upper\_addr) \mod ORTsize \quad (2)$$

$$hmult(addr) = (addr >> 5) \times \frac{(\sqrt{5}-1)}{2} >> (64 - ORTbits - 5) \mod ORTsize \quad (3)$$

Enquanto `hdefault` possui uma localidade temporal grande (beneficiando os acessos à cache), as funções `hxor` e `hmult` tendem a espalhar mais os endereços na ORT. Intuitivamente, uma função será melhor ou pior dependendo da relação entre o tempo das faltas na cache (por estar espalhando muito os acessos) e o tempo necessário para se recuperar do cancelamento da transação devido ao falsos conflitos.

4. Resultados Experimentais e Conclusões

Para a condução dos experimentos foi utilizado o micro-benchmark IntSet [Herlihy et al. 2003], o qual estressa operações de atualização (inserção e remoção) e busca em três estruturas de dados diferentes: lista ligada (`LL`), árvore rubro-negra (`RB`), e hash table (`HS`). Esse microbenchmark tem apenas operações de atualização e busca, permitindo que os resultados ofereçam uma visualização melhor do impacto das funções hash, já que o gargalo das aplicações será na qualidade da função hash e no tamanho da cache.

As taxas de atualização usadas neste trabalho foram de 20% e de 60%, que serão referenciadas como `r20` e `rw`, respectivamente. Para todas as aplicações, o tamanho do conjunto inicial foi de 4096, com o intervalo de inteiros sendo 8192. O tempo de duração para realizar as operações foi de 10000 ms. As execuções foram realizadas com 1, 2, 4, e 8 threads. Os experimentos foram realizados em uma máquina servidor com versão do Linux 2.6.32, processador *Intel®Core™* I7 2600k (8 núcleos: 4 físicos + 4 lógicos) com 8GB de memória RAM e compilador GCC versão 4.4.7. As aplicações foram compiladas com flag de otimização `-O3` e os resultados reportados representam a média de 30 execuções.

A Figura 1 apresenta os resultados obtidos. Os valores referem-se às estruturas com hash table (cima), árvore rubro-negra (meio) e lista ligada (embaixo). É possível observar através dos resultados obtidos que o uso de funções hash diferentes afeta diretamente a vazão das aplicações. A implementação `RB` não apresentou resultados muito diferentes ao variar as funções hash, indicando que possivelmente o gargalo da implementação esteja em outra área. Porém, as implementações `LL` e `HT` tiveram resultados opostos. Para `LL`, a função `hdefault` ofereceu os melhores resultados, indicando que a falta na cache é o maior obstáculo de desempenho, portanto possuir uma localidade temporal grande é importante. Para `HT`, a função `hmult` ofereceu os melhores resultados, indicando que a ocorrência de falsos conflitos impacta no desempenho mais do que faltas na cache para essa implementação. Por conta disso, espalhar bem os endereços de memória é crucial para um bom desempenho em `HT`, evitando os cancelamentos.

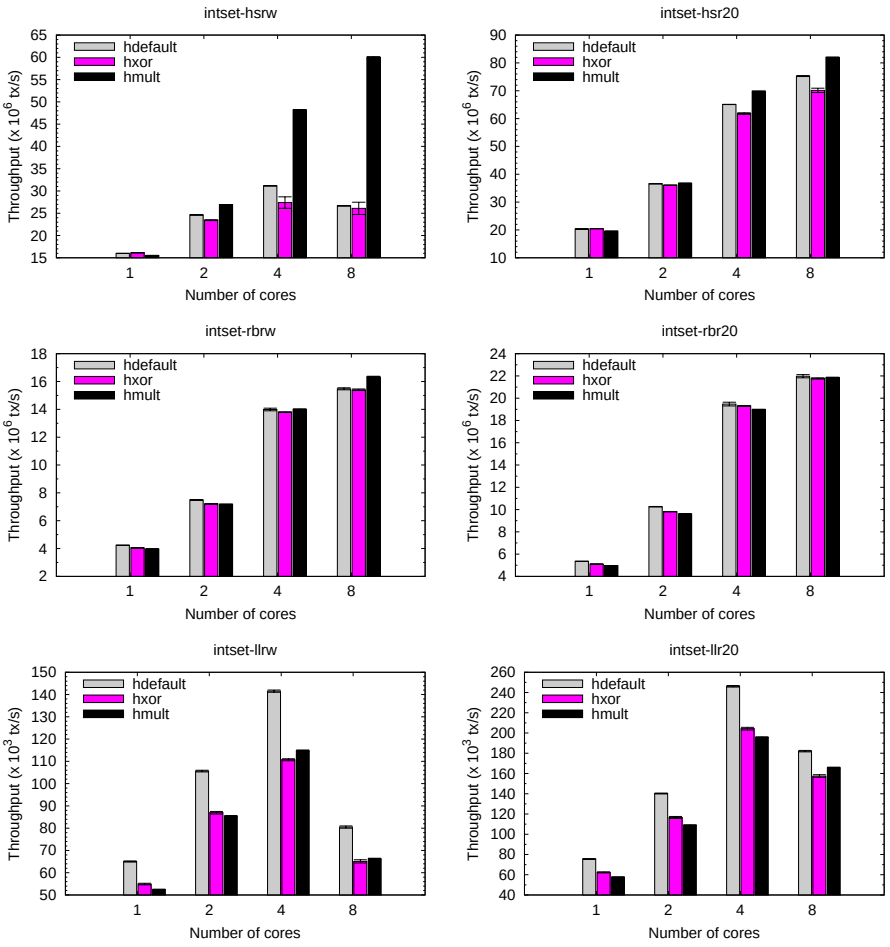


Figura 1. Gráficos de desempenho dos experimentos realizados.

Os resultados coletados nessa avaliação inicial indicam que a escolha da função hash é importante, já que esta pode ter um grande impacto no desempenho final da aplicação.

Referências

Dice, D., Shalev, O., and Shavit, N. (2006). Transactional Locking II. In *20th International Symposium on Distributed Computing*, pages 194–208.

Harris, T., Larus, J., and Rajwar, R. (2010). *Transactional Memory*. Morgan & Claypool Publishers, 2 edition.

Herlihy, M., Luchangco, V., Moir, M., and Scherer, W. N. (2003). Software transactional memory for dynamic-sized data structures. In *Proceedings of the 22nd Annual Symposium on Principles of Distributed Computing*, pages 92–101.

Knuth, D. E. (1997). *The Art of Computer Programming*, volume 3. Addison-Wesley, 2th edition.

Sutter, H. and Larus, J. (2005). Software and the concurrency revolution. *Queue*, 3(7):54–62.

Agrupamento de dados em GPU

Thiago Alexandre D. de Souza[1], **Aleardo Manacero**[1], **Renata S. Lobato**[1], **Roberta Spolon**[2]

[1]Departamento de Ciências de Computação e Estatística
Universidade Estadual Paulista "Júlio de Mesquita Filho" (UNESP)
CEP – 15.054-000 – São José do Rio Preto – SP – Brasil

[2]Departamento de Computação
Universidade Estadual Paulista "Júlio de Mesquita Filho" (UNESP)
Bauru – SP – Brasil

thi.alex@gmail.com,{aleardo,renata}@ibilce.unesp.br

Abstract. *IT industry has witnessed an explosion in the amount of data collected during the past few years. Analyzing this data usually demands clustering algorithms to provide a valuable understanding about the data's subject. These algorithms are time-consuming, demanding a lot of processing. This can be provided through parallelism, which can be done nowadays by GPUs. In line with this context, the goal of this work is the proposal of a parallel version of the Fuzzy Minimals clustering algorithm on GPUs.*

Resumo. *A indústria de tecnologia tem presenciado uma explosão na quantidade de dados coletados nos últimos anos. A análise dessas informações normalmente é feita por algoritmos de agrupamento de dados para fornecer um conhecimento apreciável sobre o campo dos dados. Tais algoritmos são computacionalmente caros, demandando grande poder computacional. Esse poder pode ser oferecido por paralelismo, o que pode ser feito atualmente por GPUs. Nesse contexto, o objetivo deste trabalho é propor uma versão paralela em GPU do algoritmo Fuzzy Minimals.*

1. Introdução

Neste momento estamos testemunhando o maior volume de dados produzido pela civilização durante toda a história. Todo o conhecimento eternizado na literatura impressa durante séculos pode, hoje, estar ao alcance de um simples equipamento digital. Essas e outras atividades corriqueiras produzem uma enorme quantidade de dados sobre situações diversas como, p. ex., comportamento dos indivíduos. Essas informações são usadas pelas empresas para conhecer melhor seus usuários, sugerir produtos, reconhecer tendências e classificá-los de acordo com suas preferências.

Esse grande volume de dados implica em que a análise dessas informações seja feita por meio de algoritmos de agrupamento de dados. Esses algoritmos utilizam critérios de semelhança entre objetos para classificá-los em determinados grupos. Evidentemente essa não é uma tarefa trivial, principalmente quando se aumenta a quantidade de dados a serem processados. Além disso, esses algoritmos possuem um alto custo computacional, pois avaliam cada objeto iterativamente. Portanto, essa é uma área que exige procedimentos escaláveis para dar vazão à demanda crescente.

Nesse sentido, a solução para aumentar o desempenho desses procedimentos normalmente envolve a paralelização do código em uma determinada arquitetura alvo. Como as placas gráficas, inicialmente desenvolvidas para auxiliar o processamento de imagens, têm desempenhado um papel importante na computação de alto desempenho, alcançando resultados expressivos em diversas áreas do conhecimento [Langdon and Banzhaf 2008, Michalakes and Vachharajani 2008, de Camargo et al. 2011], procuramos nesse trabalho explorar o potencial das GPUs para resolver problemas de agrupamento de dados. Como consequência, é proposta uma solução inédita, dentro da literatura pesquisada, para o algoritmo *Fuzzy Minimals*.

2. Trabalhos Relacionados

Diversas abordagens têm sido utilizadas para aumentar o desempenho de algoritmos de agrupamento de dados. O algoritmo *k-means* [Macqueen 1967], devido a sua eficiência em determinados conjuntos de dados, foi a base para as primeiras paralelizações na área. Tais paralelizações utilizavam principalmente o modelo de troca de mensagens [Dhillon and Modha 2000]. Mais recentemente, alguns trabalhos têm explorado outras técnicas como OpenMP [Rao et al. 2009], GPU [Li et al. 2010] e *MapReduce* [Lv et al. 2010]. A principal dificuldade desse algoritmo, contudo, está relacionada ao modo como se classificam os dados, utilizando um particionamento do tipo *hard*, isto é, cada objeto pertence a somente um grupo.

Nesse aspecto, algoritmos que utilizam lógica *fuzzy*, como o FCM [Dunn 1973], desempenham um papel importante. Assim como o *k-means*, há uma extensa literatura em torno da paralelização desse algoritmo [Havens et al. 2012, Modenesi et al. 2007]. Porém, a desvantagem é seu tempo de execução, que cresce exponencialmente à medida que aumenta o tamanho do problema, pois é necessário conhecer a *priori* o número de grupos em que os dados devem ser divididos. Normalmente, para que isso ocorra, diversas iterações são realizadas até encontrar uma solução ótima.

Para contornar tais dificuldades Flores-Sintas *et al* [Flores-Sintas et al. 1998] propôs uma alternativa, chamada *Fuzzy Minimals*. O algoritmo também utiliza a lógica *fuzzy* na busca por representantes dos agrupamentos, chamados protótipos, sem qualquer necessidade de conhecimento sobre o número de grupos, além disso, seus agrupamentos não precisam ser compactos e bem definidos. Adicionalmente, essa solução satisfaz as principais características de um bom algoritmo de classificação, tais como adaptabilidade ao encontrar grupos de diferentes tamanhos e formatos, independência de dados sob a distribuição dos objetos, estabilidade na presença de ruídos e escalabilidade para lidar com crescentes volumes de dados.

3. Desenvolvimento e Resultados Iniciais

A paralelização aqui proposta segue a abordagem de agrupamento de *Fuzzy Minimals*. Assim, se divide o conjunto de dados de maneira uniforme em partições, de modo que cada bloco de *threads* busque por protótipos em apenas uma das partições, evitando o recálculo e aumentando o desempenho, pois o acesso à regiões de memória próximas é mais eficiente. A utilização de apenas um bloco não explora todo o potencial do hardware, pois as *threads* do bloco executam em apenas um multiprocessador, enquanto os demais permanecem ociosos. Porém, a quantidade de blocos deve ser escolhida com atenção,

pois quanto maior é o particionamento dos dados, menor é a precisão do método, isto por que menos objetos significativos estão em cada partição. Um particionamento eficiente deve conter acima de 10% dos dados.

Após processar os dados em GPU, cada bloco irá encontrar protótipos semelhantes aos identificados pelos demais blocos. Isso ocorre pois a busca é local na partição atribuída ao bloco. Para eliminar esse efeito, se utiliza a técnica de agrupamento hierárquico [Johnson 1967] sobre os protótipos identificados pelos blocos. Assim, os objetos semelhantes são aglutinados em um protótipo, produzindo o agrupamento final.

Alguns testes iniciais com o algoritmo desenvolvido estão reproduzidos neste trabalho. Os dados de entrada, ilustrados na Figura 1, representam 10 mil objetos divididos em dois grupos. Os testes foram realizados em uma máquina com processador Intel Core i7 3.4Ghz, 16Gb de memória RAM e GPU GeForce GTS 450.

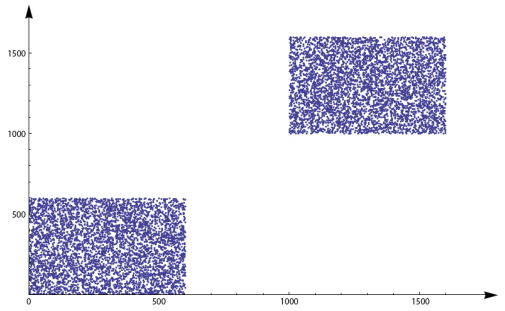


Figura 1. Conjunto de dados utilizado nos testes

Os resultados obtidos são mostrados na Tabela 3. As execuções paralelas em GPU utilizaram um, dois e quatro blocos com 960 *threads*. O número de *threads* por bloco não influencia nos protótipos identificados – note que os protótipos encontrados na execuções serial e paralela com um bloco são iguais. Porém, devido às características da paralelização do método, a precisão do algoritmo diminui à medida que o conjunto de dados é particionado, portanto, quanto mais blocos, menor é a precisão. Essa particularidade justifica a variação dos protótipos encontrados em execuções com mais de um bloco. Deste modo, para equilibar precisão e desempenho, cada bloco deve representar acima de 10% dos dados. Assim, com até quatro blocos, é obtido um *speedup* de 90 vezes.

Tabela 1. Resultados do teste

Execução	Protótipos Encontrados	Tempo de Execução	*Speedup*
Serial	$(332.34, 370.25), (1304.82, 1308.53)$	249.815s	
Paralelo 1 bloco	$(332.34, 370.25), (1304.82, 1308.53)$	33.272s	7.5
Paralelo 2 blocos	$(348.08, 381.45), (1254.14, 1330.71)$	8.985	27.8
Paralelo 4 blocos	$(305.06, 378.14), (1214.71, 1304.95)$	2.770s	90.18

4. Considerações Finais

Algoritmos de agrupamento de dados são uma classe interessante de aplicação de computação de alto desempenho. Das abordagens possíveis, o uso de GPUs demanda soluções eficientes para tratar grandes volumes de dados, principalmente pela dificuldade da identificação adequada no número de grupos. A paralelização do método de Fuzzy Minimals apresentada aqui se mostra promissora, com bons resultados de speedup, apesar de ainda faltarem ajustes para alcançar uma ocupação adequada da GPU.

Referências

[de Camargo et al. 2011] de Camargo, R. Y., Rozante, L., and Song, S. W. (2011). A multi-gpu algorithm for large-scale neuronal networks. *Concurrency and Computation: Practice and Experience*, 23(6):556–572.

[Dhillon and Modha 2000] Dhillon, I. S. and Modha, D. S. (2000). A data-clustering algorithm on distributed memory multiprocessors. In *Revised Papers from Large-Scale Parallel Data Mining, Workshop on Large-Scale Parallel KDD Systems, SIGKDD*, pages 245–260, London, UK, UK. Springer-Verlag.

[Dunn 1973] Dunn, J. C. (1973). A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters. *Journal of Cybernetics*, 3(3):32–57.

[Flores-Sintas et al. 1998] Flores-Sintas, A., Cadenas, J. M., and Martin, F. (1998). A local geometrical properties application to fuzzy clustering. *Fuzzy Sets and Systems*, 100(1):245 – 256.

[Havens et al. 2012] Havens, T. C., Bezdek, J. C., Leckie, C., Hall, L. O., and Palaniswami, M. (2012). Fuzzy c-means algorithms for very large data. *IEEE Transactions on Fuzzy Systems*, 20(6):1130–1146.

[Johnson 1967] Johnson, S. C. (1967). Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254.

[Langdon and Banzhaf 2008] Langdon, W. B. and Banzhaf, W. (2008). A simd interpreter for genetic programming on gpu graphics cards. In *Genetic Programming: 11th European Conference, EuroGP 2008, Naples, Italy, March 26-28, 2008. Proceedings*, pages 73–85. Springer Berlin Heidelberg.

[Li et al. 2010] Li, Y., Zhao, K., Chu, X., and Liu, J. (2010). Speeding up k-means algorithm by gpus. In *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, pages 115–122.

[Lv et al. 2010] Lv, Z., Hu, Y., Zhong, H., Wu, J., Li, B., and Zhao, H. (2010). Parallel k-means clustering of remote sensing images based on mapreduce. In *Proc of the 2010 Intl Conf on Web Information Systems and Mining, WISM*, pages 162–170. Springer.

[Macqueen 1967] Macqueen, J. (1967). Some methods for classification and analysis of multivariate observations. In *In 5-th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297.

[Michalakes and Vachharajani 2008] Michalakes, J. and Vachharajani, M. (2008). Gpu acceleration of numerical weather prediction. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–7.

[Modenesi et al. 2007] Modenesi, M. V., Costa, M. C. A., Evsukoff, A. G., and Ebecken, N. F. F. (2007). Parallel fuzzy c-means cluster analysis. In *High Performance Computing for Computational Science - VECPAR 2006, Rio de Janeiro, Brazil, June 10-13, 2006, Revised Selected and Invited Papers*, pages 52–65. Springer Berlin Heidelberg.

[Rao et al. 2009] Rao, S. N. T., Prasad, E. V., and Venkateswarlu, N. B. (2009). A scalable k-means clustering algorithm on multi-core architecture. In *Methods and Models in Computer Science, 2009. ICM2CS 2009. Proc. of Intl Conf. on*, pages 1–9.

Um Algoritmo Exato em Intel® Xeon® Phi™ para o *Hitting Set* Voltado para Aplicações de Biologia de Sistemas

Danilo Carastan-Santos[1], **David C. Martins-Jr**[1], **Luiz C. S. Rozante**[1]
Siang W. Song[1], **Raphael Y. de Camargo**[1]

[1]Centro de Matemática, Computação e Cognição
Universidade Federal do ABC (UFABC)
Santo André – SP – Brasil

{danilo.santos, david.martins, luiz.rozante}@ufabc.edu.br

{siang.song, raphael.camargo}@ufabc.edu.br

Abstract. *We developed an Intel® Xeon® Phi™ Hitting Set problem (HSP) algorithm suited for Systems Biology applications by introducing several optimizations that better exploit the Xeon Phi's characteristics. With a Xeon Phi 3120A, we compared its performance with a six-core CPU Xeon E5-2620v2 and an NVIDIA Tesla K20c GPU. Although in our experiments the Tesla K20c outperformed the Xeon Phi 3120A, our experimental results also show that the proposed optimizations offered significant performance gains (speedups up to 2.89 over the Xeon E5-2620v2 CPU) that allows us to use the Xeon Phi co-processor to solve HSP instances with thousands of variables in reasonable time.*

Resumo. *Desenvolvemos um algoritmo para o problema do Hitting Set (HSP) adequado para aplicações de Biologia de Sistemas utilizando co-processadores Intel® Xeon® Phi™, introduzindo diversas otimizações que melhor exploram suas características. Com uma Xeon Phi 3120A, comparamos seu desempenho com uma CPU Xeon E5-2620v2 e uma GPU NVIDIA Tesla K20c. Embora em nossos experimentos a Tesla K20c mostrou ter desempenho melhor do que a Xeon Phi 3120A, mostramos também que as otimizações propostas ofereceram ganhos de desempenho significativos (speedups de até 2,89 em relação a CPU Xeon E5-2620v2) que permitiu utilizar os co-processadores Xeon Phi para resolver instâncias do HSP na ordem de milhares de variáveis em tempo razoável.*

1. Introdução

Diversas áreas de estudo possuem problemas teóricos ou práticos os quais podem ser modelados, em parte ou como um todo, como uma instância do problema da Transversal Mínima (do Inglês, *Hitting Set Problem* ou HSP). A área de Biologia de Sistemas não é uma exceção, apresentando aplicações tais como distância reversa genômica [Kolman and Waleń 2007] e inferência de redes de regulação gênica [Ruchkys and Song 2003]. É bastante frequente que os tamanhos de entrada para o HSP dessas aplicações sejam notavelmente grandes (da ordem de milhares de variáveis), fazendo com que a obtenção das soluções exatas seja inviável para os algoritmos atuais.

Uma vez que o HSP é um problema NP-Difícil [Garey and Johnson 1999], existem algumas soluções que tentam contornar esse problema, tais como algoritmos exatos não polinomiais [Shi and Cai 2010] e algoritmos de aproximação [Ruchkys and Song 2003]. Em nossos trabalhos anteriores [Carastan-Santos et al. 2015, Carastan-Santos et al. 2016] propomos um algoritmo em CPU e um algoritmo que faz uso de múltiplas GPUs (do Inglês, *Graphics Processing Units*) homogêneas e/ou heterogêneas para se obter as soluções exatas do HSP, incluindo um mecanismo de ordenação capaz de desclassificar eficientemente as não soluções do problema. Nossos algoritmos foram capazes de lidar eficientemente com conjuntos de entrada contendo milhares de variáveis, por meio de diversas inovações nas estruturas de dados utilizadas nos algoritmos.

Nesse trabalho é apresentada uma extensão do algoritmo exato em CPU de Carastan-Santos *et al.* [Carastan-Santos et al. 2016] capaz de resolver o HSP em co-processadores Xeon Phi. Foram feitas diversas modificações no algoritmo para que as características específicas desse co-processador fossem exploradas. Rodamos nosso algoritmo em um co-processador Xeon Phi 3120A e comparamos seu desempenho com execuções em uma CPU six-core Xeon E5-2620v2 e em uma GPU NVIDIA Tesla K20c. Resultados experimentais mostraram que, apesar de a Tesla K20c apresentar um desempenho melhor do que a Xeon Phi 3120A, foi possível resolver instâncias do HSP da ordem de milhares de variáveis utilizando co-procesadores Xeon Phi em tempo razoável e com um considerável ganho de desempenho em comparação com o algoritmo em CPU, com uma curva de *speedup* crescente em função da quantidade total de variáveis do HSP.

O restante desse texto está estruturado como segue. Na Seção 2 é apresentada uma breve descrição das principais características do nosso algoritmo. Na Seção 3 são mostrados alguns resultados experimentais preliminares. Por fim, na Seção 4 são levantadas as conclusões e as perspectivas de trabalhos futuros.

2. Algoritmo exato para resolver o HSP em co-processadores Xeon Phi

Iremos descrever as principais características do algoritmo aqui proposto em termos das diferenças em relação ao algoritmo exato em CPU para resolver o HSP que já propusemos [Carastan-Santos et al. 2016], as quais estão relacionadas à forma como é feita a conferência dos candidatos a solução, que agora é executada na forma de *offload* no co-processador. A primeira grande diferença tem relação com os processadores vetoriais de 512 bits que estão presentes em cada núcleo dos co-processadores Xeon Phi, o que configura uma vantagem [Jeffers and Reinders 2013]. Assim sendo, rearranjamos o código de modo a explorar esse fato, evitando desvios condicionais e saídas antecipadas de laços de repetição, os quais impediriam a vetorização. Modificamos especialmente o laço de repetição que confere se um candidato a solução do HSP satisfaz uma cláusula da entrada.

A segunda parte com maior custo computacional do algoritmo para resolver o HSP está na geração dos candidatos a solução. Seja X o conjunto de variáveis de uma instância do HSP. No algoritmo em CPU, cada *thread* gera seus candidatos a solução utilizando um sistema de numeração combinatória, que estabelece uma correspondência única entre uma combinação de elementos de X com cardinalidade i e um número inteiro N, $0 \leq N \leq \binom{|X|}{i}$ [Knuth 2005]. Entretanto, o uso de instruções vetoriais no algoritmo desse sistema mostrou-se inviável. Sendo assim, foi utilizada uma abordagem análoga ao algoritmo de GPU de Carastan-Santos *et al.* [Carastan-Santos et al. 2016]. Desse modo, é

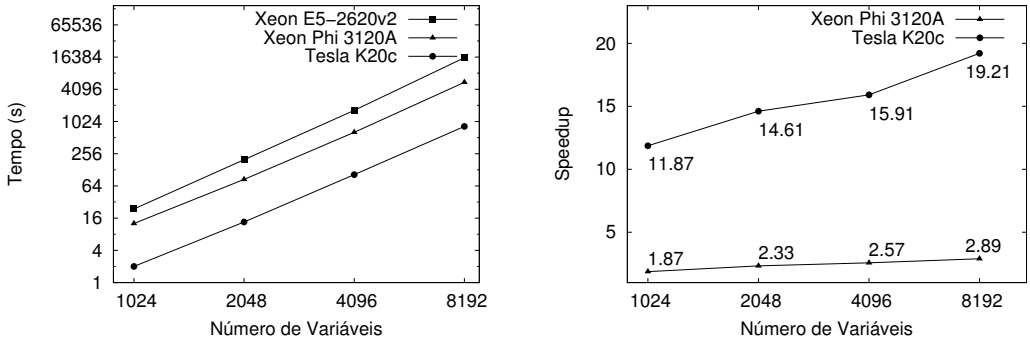
utilizado o sistema de numeração combinatória apenas na CPU, uma vez para cada *thread* da Xeon Phi, com $N = j\kappa$, $0 \leq j < t$, $\kappa = \left\lfloor \binom{|X|}{i}/t \right\rfloor$, onde t é a quantidade de *threads* a serem executadas na Xeon Phi. As consequentes combinações geradas são transmitidas para a Xeon Phi na forma de um vetor. Os demais candidatos a solução são gerados em sequência por cada *thread* dentro do co-processador durante o *offload*, tendo como ponto de partida uma única combinação gerada na CPU.

3. Resultados Experimentais

Abaixo apresentamos os resultados experimentais da implementação do algoritmo para resolver o HSP em co-processadores Xeon Phi. Para uma avaliação comparativa de desempenho, foram utilizados os algoritmos para CPU e GPU de Carastan-Santos *et al.* [Carastan-Santos et al. 2016]. Foi utilizado o compilador da Intel (`icc`) na versão 16.0.1, exceto para a implementação em GPU, onde utilizamos o compilador da NVIDIA (`nvcc`) na versão 7.5. O parâmetro de otimização `-O3` foi utilizado durante a compilação de todas as implementações dos algoritmos. A Figura 1(a) mostra os tempos de execução medidos para o algoritmo em CPU paralelizado em seis *threads* em um processador *six-core* Intel Xeon E5-2620v2, para o algoritmo em GPU com uma GPU NVIDIA Tesla K20c com 2496 núcleos e para o algoritmo proposto com uma Xeon Phi 3120A com 57 núcleos paralelizada em 224 *threads*, em função do número total de variáveis do HSP e para instâncias cujas soluções são conjuntos de cardinalidade 3. A Figura 1(b) mostra os respectivos *speedups* obtidos em comparação com o algoritmo em CPU. Foram obtidos *speedups* de até 2,89 para a Xeon Phi e 19,21 para GPU, ambos crescentes com o aumento do número total de variáveis. Apesar de a Xeon Phi possuir uma pequena quantidade de núcleos se comparada com a GPU, seus processadores vetoriais de 512 bits presentes em cada núcleo permitem uma utilização eficiente dos seus recursos com o aumento da quantidade total de variáveis, enquanto esse aumento não beneficia o desempenho do algoritmo em CPU. No entanto, a quantidade massiva de núcleos da GPU, embora sejam núcleos mais simples em comparação com os núcleos da Xeon Phi, ainda proporcionam um melhor desempenho. Nesse sentido, os *speedups* obtidos para a Xeon Phi nesse trabalho, embora pequenos se comparados com a GPU, são consistentes com os ganhos de desempenho obtidos em outros trabalhos [Intel 2016].

4. Conclusão

Nesse trabalho o algoritmo em CPU para se obter as soluções exatas do HSP de Carastan-Santos *et al.* [Carastan-Santos et al. 2016] foi estendido para que possa ser executado em co-processadores Xeon Phi de maneira eficiente. Foram feitas diversas modificações no algoritmo para que as capacidades de processamento vetorial dos co-processadores Xeon Phi fossem exploradas e também para que a transferência de dados entre a CPU e o co-processador fosse reduzida. Foram obtidos *speedups* significativos (*speedup* máximo medido de 2,89) e crescentes com o aumento do número total de variáveis do HSP. Portanto, mesmo que com a GPU utilizada nos experimentos tenhamos obtido um melhor desempenho do que com a Xeon Phi, mostramos que é possível resolver instâncias do HSP na ordem de milhares de elementos em tempo razoável e alcançando bom desempenho utilizando co-processadores Xeon Phi. É importante observar também que o algoritmo proposto nesse trabalho para resolver o HSP pode ser adaptado para outras aplicações que possam ser modeladas via HSP e que possuam tamanho de entrada da ordem de milhares



(a) Tempo de Execução (escala log-log)

(b) *Speedup* (escala log no eixo x)

Figura 1. Tempos de execução e *speedups* obtidos das implementações em função do número de variáveis do HSP.

de variáveis. Como consequência desse trabalho, pretendemos também desenvolver um algoritmo capaz de resolver o HSP utilizado *clusters* heterogêneos de GPU e Xeon Phi.

Referências

Carastan-Santos, D., de Camargo, R. Y., Martins, D. C., Song, S. W., and Rozante, L. C. (2016). Finding exact hitting set solutions for systems biology applications using heterogeneous gpu clusters. *Future Generation Computer Systems. In Press*.

Carastan-Santos, D., Yokoingawa De Camargo, R., Correa Martins, D., Song, S. W., Silva Rozante, L., and Ferreira Borelli, F. (2015). A multi-gpu hitting set algorithm for grns inference. In *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*, pages 313–322.

Garey, M. R. and Johnson, D. S. (1999). *Computers and Intractability - A guide to the Theory of NP-completeness*. W. H. Freeman and Company.

Intel (2016). Intel® Xeon® Phi™ Coprocessor Life Science Applications Benchmarks. `http://www.intel.com/content/www/us/en/benchmarks/server/xeon-phi/xeon-phi-life-sciences.html`. Online; acessado em 28 de abril de 2016.

Jeffers, J. and Reinders, J. (2013). *Intel Xeon Phi coprocessor high-performance programming*. Newnes.

Knuth, D. E. (2005). *The Art of Computer Programming, Fascicle 3: Generating All Combinations and Partitions*, volume 4. Addison-Wesley, Reading.

Kolman, P. and Waleń, T. (2007). *Reversal distance for strings with duplicates: Linear time approximation using hitting set*. Springer.

Ruchkys, D. P. and Song, S. W. (2003). A parallel solution to infer genetic network architectures in gene expression analysis. *International Journal of High Performance Computing Applications*, 17(2):163–172.

Shi, L. and Cai, X. (2010). An exact fast algorithm for minimum hitting set. *Int. Joint Conference on Computational Science and Optimization*, 1:64–67.

Paralelização de laços aninhados com processamento heterogêneo em sistemas paralelos e distribuídos.

Cleber Silva Ferreira Luz[1] **e Liria Matsumoto Sato**[1]

[1]Departamento de Engenharia de Computação e Sistemas Digitais
Escola Politécnica da Universidade de São Paulo (EPUSP)
São Paulo – SP – Brasil

cleber.luz@usp.br liria.sato@poli.usp.br

Resumo. *Computação paralela e distribuída vem se consolidando como um importante elemento no processamento de aplicações que necessitam de alto poder computacional. Diversas aplicações utilizam laços aninhados com computação heterogênea para processar seus dados. Geralmente aplicação com este perfil consome horas ou até mesmo dias de processamento. Este artigo apresenta uma solução para a paralelização de laços aninhados com computação heterogênea em sistemas paralelos e distribuídos. A proposta visa a diminuição do tempo de processamento das aplicações que contêm laços aninhados com processamento heterogêneos. A metodologia permite realizar uma otimização no uso dos recursos de processamento. Esta otimização é necessária para diminuir a ociosidade dos recursos, causada pela heterogeneidade de tempo de processamento das computações heterogêneas pertencentes aos laços aninhados.*

1. Introdução

Atualmente, computação paralela e distribuída tem sido essencial para se obter um maior desempenho no processamento de aplicações que demandam alto poder computacional. Em aplicações biológicas, por exemplo, geralmente é necessário analisar milhões de amostras experimentais, que muitas vezes, consomem horas ou até mesmo dias de processamento. Aplicações com esse perfil, muitas vezes utilizam laços de repetições aninhados no processamento de dados e muitas vezes, executam computações heterogêneas dentro dos mesmos.

Habitualmente, o processamento destas aplicações analisa e gera uma grande quantidade de dados que consomem horas ou até mesmo dias de processamento. Dessa forma, o processamento de aplicações com este perfil requer ferramentas capazes de manipular grandes quantidades de dados, como também, boas estratégias de processamento, a fim de realizar o processamento de forma rápida e eficiente. Uma solução para aplicações que apresentam laços aninhados é a sua linearização seguida da paralelização.

Diversas classes de algoritmos utilizados nas áreas de análise numérica, processamento de imagem e meteorológica contêm laços aninhados. Quando o algoritmo possui laços aninhados com computações heterogêneas a paralelização de um dos laços poderia implicar em uma subutilização dos recursos de processamento, deixando-os ociosos.

Neste artigo é apresentada uma solução para paralelização de laços aninhados com computações heterogêneas em sistemas paralelos e distribuídos aplicando a técnica de

linearização. A solução paralela também realiza uma otimização no uso dos recursos de processamentos utilizados no processamento das iterações dos laços aninhados, buscando a reduzir sua ociosidade, através de estratégicas que promovem o balanceamento de cargas. Este artigo considera como estudo de caso a modelagem de distribuição de espécies. A solução paralela proposta foi aplicada no processo de modelagem de distribuição de espécies. Para este estudo de caso foram alcançados bons resultados que são apresentados na Seção 4.

A solução proposta pode ser implementada em sistemas de processamento paralelo e distribuídos, tais como, computadores multicores, ou sistemas distribuídos como um *cluster* físicos ou virtuais alocados na nuvem.

2. Paralelização de laços aninhados com processamento heterogêneo em sistemas paralelos e distribuídos.

A solução paralela proposta descreve uma série de procedimentos para paralelizar laços aninhados com computações heterogêneas em ambientes paralelos e distribuídos. A solução proposta também realiza uma otimização no uso dos recursos de processamento, necessária para diminuir a ociosidade dos recursos de processamento, causada pela presença de computações heterogêneas dentro dos laços aninhados.

A otimização é realizada através da definição do número máximo de recursos de processamento, o qual não deverá ultrapassar a quantidade de recursos onde não se teria uma melhora de desempenho no processamento das iterações dos laços. A otimização também realizada realiza um escalonamento entre as computações das iterações do laço linearizado entre os recursos de processamento. Tal escalonamento é realizado visando uma redução de ociosidade entre os recursos de processamento.

Em um paralelismo onde o escalonamento é realizado de forma arbitrária, a distribuição de carga de trabalho não é realizada de forma homogênea. Alguns recursos de processamento podem receber uma carga de trabalho maior, enquanto que outros recursos podem receber uma carga de trabalho menor. Essa desigualdade de atribuição de carga de trabalho pode causar ociosidade entre os processados. A otimização realizada pela solução paralela realiza um escalonamento, onde, não há uma desigualdade de carga de trabalho entre os recursos de processamento.

A solução paralela proposta lineariza os laços aninhados e descreve uma série de procedimentos para tornar o laço linearizado em um laço paralelo. Na solução paralela é considerado que não há dependência de dados entre as computações. O Algoritmo 2 descreve um pseudocódigo da estratégia da solução paralela proposta.

Algorithm 1 Estratégia da solução paralela.

1: linearização dos loops aninhados.
2: preparação das iterações do loop linearizado para o processamento.
3: definição do número de recursos de processamento para processar as iterações do laço linearizado.
4: escalonamento das iterações do loop linearizado.
5: processamento paralelo, em que cada recurso processa as iterações que lhe foram atribuídas.

A solução paralela é composta por: Coleta de tempo, Cálculo do Número de Recursos de Processamento, Escalonamento e Processamento. O mecanismo de Coleta de Tempo realiza a obtenção dos tempos das computações heterogêneas presentes dentro dos laços aninhados. O mecanismo de Cálculo do Número de Recursos de Processamento propõe o número de recursos de processamento que melhora o desempenho da aplicação. Já o mecanismo de escalonamento permite atribuir as iterações do laço linearizado para os recursos de processamento. No mecanismo de escalonamento, a tarefa de maior carga é atribuída para o recurso que contém a menor carga de trabalho. O último mecanismo realiza o processamento paralelo das iterações do laço linearizado.

3. Estudo de Caso: Modelagem de Distribuição de espécies

A modelagem de distribuição de espécies é uma ferramenta numérica que combina observações de ocorrências de espécies com variáveis ambientais. Esta combinação permite realizar uma predição de ocorrência da espécie em determinadas áreas geográficas [3]. A modelagem é dada através de representações matemáticas, fornecidas pelo processamento de diversos algoritmos de modelagem de distribuição de espécie. Para se obter uma boa taxa de acerto de predição, o processo de modelagem executa diversos algoritmos de modelagem que consomem tempos de processamento diferentes. O processamento destes algoritmos é realizado através do processamento de laços aninhados.

4. Resultados

Esta seção, apresenta os resultados da implementação da solução paralela na modelagem de distribuição de espécies. O gráfico apresentado na Figura 1 apresenta o tempo de processamento paralelo e sequencial da fase de modelagem.

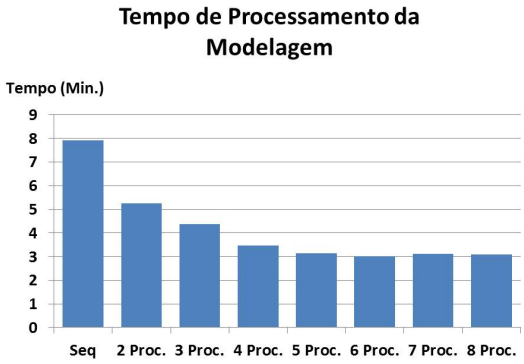


Figure 1. Tempo de Processamento da Modelagem

O tempo de processamento sequencial da modelagem foi de 8 minutos, ao passo que, o tempo de processamento paralelo foi de 3 minutos utilizando 6 recursos de processamento. A solução paralela definiu que 6 recursos é a quantidade ótima de recurso, mesmo adicionando mais recurso, não é possível obter um tempo menor que 3 minutos, como é o caso do processamento utilizando 7 e 8 recursos. Neste experimento, o algoritmo de modelagem GAM consumiu mais tempo de processamento, consumindo

aproximadamente 2 minutos e 40 segundos de processamento, sendo este o tempo aproximado do menor tempo de processamento possível. Dessa forma, é possível concluir que o tempo de processamento mínimo foi alcançado com a menor quantidade de recursos possíveis.

5. Trabalhos Relacionados

Em [4] é apresentado um algoritmo de escalonamento dinâmico para laços aninhados com dependências uniformes em redes com computadores heterogêneos. O trabalho apresentado em [4] vai de encontro com a nossa proposta. Em [4] há um escalonamento de laços aninhados para ambientes heterogêneos. Na metodologia proposta neste artigo, há uma preocupação em linearizar os laços aninhados e processar suas iterações em paralelo. A solução paralela proposta também realiza um escalonamento adequado das computações heterogêneas presentes dentro dos laços aninhados afim de, manter uma boa utilização dos recursos de processamento.

Em [1] é apresentada uma ferramenta para a paralelização de laços aninhados em multicomputadores com memória distribuída. A ferramenta apresentada em [1] utiliza a técnica *grouping* para particionar os laços aninhados com dependência de dados. O resultado desta técnica é um pipeline que fornece um balanceamento na computação dos laços aninhados. A ferramenta apresentada em [1] realiza uma paralelização de laços aninhados em multicomputadores com memória distribuída. A solução paralela proposta se preocupa em escalonar os laços aninhados levando em consideração as computações heterogêneas realizadas dentro dos laços aninhados.

6. Conclusões

Este artigo apresenta uma solução para paralelização de laços aninhados com processamento heterogêneo em sistemas paralelos e distribuídos. A solução proposta neste artigo, também realiza uma otimização no uso dos recursos de processamento. A otimização é necessária para diminuir a ociosidade entre os recursos de processamento, causada pela heterogeneidade dos tempos de execuções das computações heterógenas presentes dentro dos laços aninhados. A solução proposta foi implementada na aplicação de modelagem de distribuição de espécies, onde, foram alcançados bons resultados e que são apresentados neste artigo.

References

[1] Chung-Ta King and Ing-Ren Kau. - Parallelizing nested loops on multicomputers-the grouping approach. *Computer Software and Applications Conference, 1989. COMPSAC 89., Proceedings of the 13th Annual International*.

[2] Gonzalo Vera and Remo Suppi - Integration of heterogeneous and non-dedicated environments for R. 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing

[3] Jennifer Miller - Species Distribution Modeling. *Volume 4, Issue 6, pages 490-509, June 2010*

[4] R., Ionannis and T., Panagiotis - Dynamic Scheduling of Nested Loops with Uniform Dependencies in Heterogeneous Networks of Workstations. *8th International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN'05), 2005*.

Explorando o Paralelismo em Workflows Intensivos em Dados com o Uso de Anotações Semânticas e Informações sobre o Ambiente de Execução*

Elaine Naomi Watanabe[1], **Kelly Rosa Braghetto**[1]

[1]Departamento de Ciência da Computação – Instituto de Matemática e Estatística
Universidade de São Paulo (USP)
Rua do Matão, 1010, Cidade Universitária, 05508-090 – São Paulo, SP, Brasil

{elainew,kellyrb}@ime.usp.br

Abstract. *The use of high-performance platforms is a prerequisite for the execution of activities that handle large volumes of data. A set of interconnected activities, modeled as a workflow, can have its execution controlled by a Workflow Management System (WfMS). The data parallelism can reduce the workflow runtime. However, the WfMSs do not exploit it automatically. This paper proposes the use of semantic annotations for the automatic creation of a parallel model for the execution of activities. In experiments with a workflow that handles 5.8 millions of data objects, the parallelization obtained from the annotations reduced by 88.37% the workflow runtime and 10.35% financial cost.*

Resumo. *O uso de plataformas de alto desempenho é um requisito para a execução de atividades que lidam com grandes volumes de dados. Um conjunto de atividades interligadas, modeladas como um workflow, pode ter sua execução controlada por um Sistema de Gerenciamento de Workflows (SGWfs). O paralelismo de dados pode diminuir o seu tempo total de execução, contudo, os SGWfs não exploram isso de maneira automática. Este trabalho propõe o uso de anotações semânticas para a criação automática de um modelo paralelo para a execução das atividades. Em experimentos com um workflow que lida com 5,8 milhões de objetos de dados, a paralelização obtida das anotações reduziu em 88,37% o tempo de execução do workflow e em 10,35% o custo financeiro.*

1. Introdução

Aplicações que manipulam grandes volumes de dados têm como requisito o uso de plataformas de alto desempenho [Chen and Zhang 2014]. Modelar tais aplicações como atividades interligadas (workflows) permite que Sistemas de Gerenciamento de Workflows (SGWfs) sejam utilizados para o controle do processamento distribuído. Nesses cenários, paralelizar atividades por meio do particionamento de dados é um método adotado para redução do *makespan* (tempo total de execução de um workflow). Entretanto, a ausência de informações sobre o tipo de processamento realizado em cada atividade impossibilita que um SGWf defina, de maneira automática, estratégias para ampliar o paralelismo dos dados [de Oliveira et al. 2015].

*Essa pesquisa foi financiada pela CAPES e pelo NAPSoL-PRP-USP. Os autores agradecem também ao Google pelos créditos concedidos para uso de sua plataforma de nuvem.

Este trabalho propõe o uso de anotações semânticas para caracterizar as atividades de um workflow quanto ao seu tipo de processamento de dados e aos atributos de objetos de dados utilizados. A partir das anotações, é possível reestruturar o workflow de maneira automática, de modo a obter maior paralelismo de dados, considerando também informações sobre o ambiente de execução.

2. Estratégia Proposta para Paralelização de Workflows Intensivos em Dados

O paralelismo de dados em workflows foi explorado por meio da definição de anotações sobre as atividades e sobre os atributos de objetos de dados usados; isso permitiu a reestruturação do workflow e do esquema do banco de dados para ampliar a execução paralela das atividades anotadas. O trabalho considera que os dados manipulados pelo workflow são mantidos por Sistema de Gerenciamento de Bancos de Dados (SGBD).

O tipo de processamento de dados pode ser caracterizado por meio de duas anotações semânticas: *Processamento por Objeto (PO)* e *Processamento por Grupo de Objetos (PG)*. A anotação *PO* define que uma atividade processa cada objeto de dados de entrada individualmente. Já anotação *PG* indica que uma atividade processa os objetos de dados de entrada em grupos; cada grupo é definido por meio de atributo(s) agrupador(es). Cada possível valor (combinação de valores) para o(s) atributo(s) agrupador(es) define um grupo diferente a ser processado.

Foram definidas anotações para identificação da lista de atributos de um objeto de dados necessários para uma atividade (anotação *Seleção de Atributos – SA*) e ordenação dos objetos de dados (anotação *Ordenação de Objetos – OO*).

Essas anotações permitem a modificação da estrutura de um workflow, por meio da definição de réplicas das atividades anotadas e da associação de subconjuntos de dados para cada réplica. As anotações sobre atributos permitem a definição de índices para ordenação e a recuperação somente dos atributos necessários para cada atividade.

3. Experimentos e Resultados Obtidos

Um workflow que manipula 5,8 milhões de objetos de dados foi avaliado em onze cenários. Cada cenário considerou o uso de um banco de dados, o número total de nós disponíveis para a execução das atividades e o uso das anotações semânticas propostas.

Tradicionalmente, os bancos de dados relacionais são mais utilizados devido a suas propriedades transacionais e à linguagem padronizada SQL. Entretanto, os sistemas NoSQL têm controle de concorrência e modelo de dados mais flexíveis, que se beneficiam melhor de ambientes distribuído. Para os experimentos realizados, considerou-se bancos de dados no SGBD PostgreSQL (relacional) e no MongoDB (NoSQL). No caso do NoSQL, foram avaliados um cenário centralizado e dois cenários distribuídos. Como SGWf, adotou-se o Pegasus WMS.

Todos os experimentos foram executados na plataforma de computação em nuvem do Google e o custo monetário foi calculado conforme o tipo de máquina utilizada e o total de disco necessário para a execução de cada cenário. Cada experimento foi executado cinco vezes e os resultados são apresentados com um nível de confiança de 95%.

Foram comparados o tempo total de execução (Figura 1(a)) e o custo monetário (Figura 1(b)) da execução do workflow em cada um dos cenários descritos na Tabela 1.

Tabela 1. Caracterização dos cenários de execução avaliados.

Workflow	SGBD	Uso das anotações	N^o Partições	N^o Réplicas	N^o Máq. SGBD	N^o Nós Execução
W-01	PostgreSQL	Não	1	1	1	1
W-02	PostgreSQL	Não	1	1	1	3
W-03[a]	PostgreSQL	Sim	1	1	1	3
W-04	MongoDB	Não	1	1	1	1
W-05	MongoDB	Não	1	1	1	3
W-06[a]	MongoDB	Sim	1	1	1	3
W-07	MongoDB	Não	1	3	9	3
W-08[a]	MongoDB	Sim	1	3	9	3
W-09	MongoDB	Não	3	3	15	9
W-10[a]	MongoDB	Sim	3	3	15	9
W-11[a]	MongoDB	Sim	3	3	15	9

[a]Cenário envolvendo o uso de anotações semânticas no workflow.

Vale a pena destacar que o uso do MongoDB distribuído requer máquinas reservadas para o roteamento de consultas; para isso, utilizou-se 6 máquinas adicionais nesses casos.

Considerando cenários que utilizaram as anotações propostas para explorar o paralelismo de dados, foi possível obter uma redução de 33,77% do *makespan* e de 7,13% do custo monetário com o PostgreSQL (cenários W-03 $\times$ W-02) e redução de 57,34% do *makespan* e de 3,15% do custo no cenário mais básico de uso do MongoDB (cenários W-06 $\times$ W-05). É importante observar que, com uma mesma quantidade de máquinas com mesma configuração de hardware, obteve-se uma diminuição de 84,45% no *makespan* do workflow devido das anotações e ao uso do MongoDB, em substituição ao uso do PostgreSQL (cenários W-06 $\times$ W-02). Com a adição de mais máquinas – para se ter um banco de dados particionado e mais nós de execução no Pegasus – observou-se um ganho de desempenho ainda maior (redução de 88,36% no *makespan*, em W-10 $\times$ W-02).

A adição de réplicas dos dados no MongoDB gerou um aumento de 38,73% no *makespan* e de 14,64% no custo monetário (cenários W-07 $\times$ W-05). Isso deve-se ao tempo de sincronização entre as réplicas. Mas é importante observar que, com o uso das anotações, o aumento no *makespan* e no custo monetário médios foi menor – 14,86% e 5,40%, respectivamente (cenários W-08 $\times$ W-06). Portanto, as anotações reduziram o impacto negativo no *makespan* e no custo monetário das réplicas para tolerância a falhas.

Comparando os cenários de uso do PostgreSQL (W-01, W-02 e W-03) com os do MongoDB que usam as mesmas quantidades de máquinas (W-04, W-05 e W-06) na Figura 1(b), observa-se que o custo monetário para a manutenção dos cenários com o MongoDB foi maior, apesar das máquinas terem sido usadas por menos tempo. Isso se deu devido a um maior uso de disco nesses casos.

Considerando os cenários em que existem mais de uma partição de dados (W-09, W-10 e W-11), observou-se uma redução no custo monetário de 10,35% tanto em W-10 $\times$ W-09 quanto em W-11 $\times$ W-09, devido ao cálculo do custo por hora. Entretanto, a redução no *makespan* em W-10 $\times$ W-09 (74,40%) é maior do que em W-11 $\times$ W-09 (72,69%). A distribuição por *hash* (cenário W-10) proporcionou um balanceamento de

carga melhor entre as partições de dados do que a por intervalo (cenário W-11).

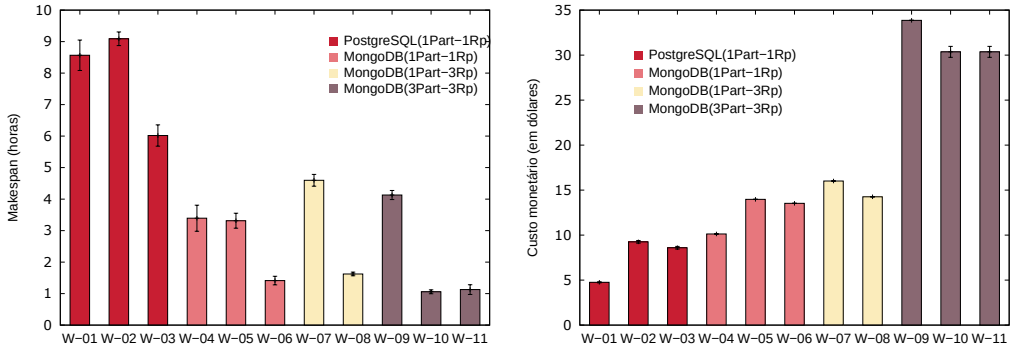


Figura 1. Comparação do makespan e do custo financeiro dos cenários avaliados

4. Considerações Finais

A estratégia proposta para paralelização de workflows intensivos em dados baseada em anotações semânticas e informações sobre o ambiente de execução proporcionou uma redução do *makespan* do workflow avaliado de 88,4% em um sistema NoSQL. Além disso, cenários com as mesmas configurações e mesmo número de máquinas virtuais obtiveram redução do custo financeiro de até 10,35%.

Diferentemente do que ocorre em abordagens como a do Apache Oozie [Islam et al. 2012], para exploração do paralelismo de dados em workflows, a utilização das anotações propostas neste trabalho não requer que o projetista do workflow (e conhecedor do domínio da aplicação) seja um especialista em programação paralela. Até onde pôde-se verificar, não existem trabalhos que tenham avaliado o uso combinado de anotações semânticas sobre atividades e informações do ambiente de execução e do banco de dados para detectar e se beneficiar de oportunidades de paralelismo de forma automática em workflows.

Possíveis trabalhos futuros incluem: a definição de novas anotações e estratégias para o paralelismo de dados; avaliação de mais sistemas NoSQL e do impacto da leitura de réplicas de dados; e experimentos para identificar como o número de partições do banco de dados pode ser definido em função do número de nós para execução do workflow.

Referências

Chen, C. P. and Zhang, C.-Y. (2014). Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. Information Sciences, 275(0):314–347.

de Oliveira, D. E. M., Boeres, C., Neto, A. F., and Porto, F. (2015). Avaliação da localidade de dados intermediários na execução paralela de workflows BigData. In Proceedings of 30th Brazilian Symposium on Databases, pages 29–40. Brazilian Computer Society.

Islam, M., Huang, A. K., Battisha, M., Chiang, M., Srinivasan, S., Peters, C., Neumann, A., and Abdelnur, A. (2012). Oozie: Towards a scalable workflow management system for hadoop. In Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies, SWEET '12, pages 4:1–4:10, New York, NY, USA. ACM.

Avaliação da aplicação de meta-heurísticas para o problema de provisionamento de recursos

Leonildo J. M. de Azevedo[1], **Bruno G. Batista**[2], **Júlio Cezar Estrella**[1]

[1]Instituto de Ciências Matemáticas e de Computação – Universidade de São Paulo (USP)
Av. Trabalhador São-Carlense, 400 – 13566-590 – São Carlos – SP – Brasil

[2]Instituto de Matemáticas e Computação – Universidade Federal de Itajubá (UNIFEI)
Av. BPS, 1303 – 35903-087 – Itajubá – MG – Brasil

leonildo.azevedo@usp.br, brunoguazzelli@unifei.edu.br, jcezar@icmc.usp.br

Abstract. *Nowadays the access to a cloud computing environment is provided on demand, which allows providers to offer transparent services to clients. To ensure compliance with the contract between client and provider, service providers need mechanisms to provide resources to the customer at a fair price. In this context, meta-heuristics in resource provisioning process are analyzed in this paper.*

Resumo. *Atualmente o acesso a um ambiente de computação em nuvem é fornecido sob demanda, o que permite que provedores ofereçam serviços de forma transparente aos clientes. Para garantir o cumprimento do contrato entre cliente e provedor, os provedores de serviços necessitam de mecanismos forneçam os recursos para o cliente a um preço justo. Neste contexto, neste trabalho é avaliado o uso de meta-heurísticas no processo de provisionamento de recursos.*

1. Introdução

Em razão da ascensão tecnológica de *hardware* e ao sucesso da Web, os recursos computacionais estão se tornando cada vez mais acessíveis e com suporte a um alto desempenho, em comparação às máquinas de décadas atrás. Esta evolução tecnológica possibilitou o desenvolvimento de um paradigma denominado computação em nuvem (*Cloud Computing*). A Computação em Nuvem está conceitualmente dividida em três modelos de serviços básicos que são: ***Software as a Service (SaaS)***, ***Platform as a Service (PaaS)***, e ***Infrastructure as a Service (IaaS)***. Essa abordagem pode caracterizar uma divisão abstrata em camadas dos serviços prestados [Buyya et al. 2011]. Neste trabalho, o enfoque está no nível de IaaS.

Prover uma infraestrutura adequada aos seus clientes sempre foi um desafio para os provedores de serviços. Clientes diferentes exigem quantidades e recursos diferentes dependendo de suas aplicações. Por esse motivo, algumas configurações de máquinas virtuais (VMs) são pré-definidas (por exemplo, a Amazon trabalha com diferentes classes de VMs[1]) pelo provedor em Nuvem e o próprio usuário deve determinar a melhor quantidade e capacidade das máquinas virtuais para que o seu sistema funcione adequadamente. Acertar o limiar de quando uma reconfiguração da infraestrutura deve ser realizada é um tópico de grande interesse tanto no âmbito acadêmico quanto industrial. Outro desafio é quantificar o volume de recursos (quantidade de instâncias de máquinas virtuais) que devem ser iniciados ou desligados. Portanto, o provisionamento de recursos levando em consideração o contrato à nível de serviço (SLA - *Service Level Agreement*) não é uma tarefa trivial.

[1]https://aws.amazon.com/ec2/instance-types

Com o objetivo de avaliar a factibilidade de meta-heurísticas para solucionar este problema, alguns algoritmos de otimização foram implementados e analisados. Dentre eles, foram desenvolvidos um algoritmo determinístico e um micro-Algoritmo Genético (μAG). A implementação dos algoritmos e os resultados obtidos são apresentados no decorrer deste trabalho.

2. Trabalhos relacionados

Há na literatura diversos trabalhos que analisam e propõem mecanismos para o gerenciamento de recursos em um ambiente em nuvem. A proposta da Amazon para a reconfiguração automática da infraestrutura de seus clientes é baseada em monitoramento, por meio de alertas e políticas. O escalonamento da AWS (*Amazon Web Services*) pode ser feito baseado em métricas que são geralmente baseadas no consumo de CPU ou em políticas que são basicamente divididas em políticas manuais, políticas dinâmicas e políticas agendadas [Amazon 2015]. O trabalho de [Batista et al. 2015], dentre os encontrados na literatura, é provavelmente o trabalho que mais se aproxima deste escopo. Nesse trabalho, é proposta uma avaliação de desempenho no gerenciamento de recursos em computação em nuvem, onde é implementado um módulo de reconfiguração que tem por objetivo manter a QoS definida no SLA. Contudo, constatou-se a necessidade de técnicas de otimização para esse módulo, pois em alguns casos o modulo de reconfiguração não conseguia manter a QoS contratada. Compreende-se deste trabalho como um complemento ao trabalho de [Batista et al. 2015].

3. Métodos

Para a execução dos experimentos, foi estipulado um SLA para um determinado cliente, tendo como atributos de QoS a capacidade (C^c), o tempo (T^c), a disponibilidade (D^c) e o custo/h (C/h^c). As equações para tais atributos são baseadas no trabalho de [Ko et al. 2008]. A simulação consiste em executar a aplicação do cliente na capacidade descrita no SLA. Com a execução é obtido o tempo de resposta, a disponibilidade e o custo/h retornados pelo CloudSim [Calheiros et al. 2011]. Caso os parâmetros retornados pelo Cloudsim não sejam compatíveis com os descritos no SLA, é aplicado uma meta-heurística para encontrar uma solução (capacidade (C^*), tempo (T^*), disponibilidade (A^*) de custo/h (C/h^*)) que melhor se adeque às exigências do cliente.

Foram elaborados dois algoritmos para a solução do problema, um algoritmo determinístico (vide Algoritmo 1) e um μAG (vide Algoritmo 2). No algoritmo determinístico é explorado todo o espaço de busca dentro de um determinado intervalo, o qual sempre se dá por 50% acima e abaixo do número de VMs que o cliente requisitou. Por exemplo, se o cliente contratou 50 VMs, o intervalo fica entre 25 e 75 VMs, gerando todas as combinações possíveis de VMs disponíveis dentro desse intervalo a fim de obter a melhor configuração para satisfazer o SLA do cliente. Na execução do experimento, primeiro é gerada uma capacidade, em seguida, é executada a aplicação do cliente com essa configuração e obtido um tempo de resposta, uma disponibilidade e o custo/h. Para avaliar o quão próxima essa solução é da ótima (estipulada no SLA) é aplicado um cálculo estatístico denominado Distância de Manhattan [Black 2006]. A cada execução é armazenada a melhor solução e no método determinístico são geradas todas a possibilidades. O algoritmo μAG é uma versão do AG que trabalha com uma população de tamanho menor e um critério de convergência que permite reiniciar indivíduos. A população foi ajustada para evoluir apenas 5 indivíduos, sendo que cada indivíduo representa uma

possível capacidade para o cliente. Uma taxa de mutação de 20% também foi utilizada.

Algoritmo 1: Algoritmo determinístico

```
Entrada: SLA: C^c, T^c, A^c, C/h^c
//varre todo espaço de busca
repita
    //gera uma capacidade para avaliar
    Gera(C*)
    Avalia(C*)
até que todas as possibilidades no intevalo sejam testadas;
//retorna melhor configuração que satisfaz o SLA
retorna
    SLA* : C*, T*, A*, C/h*
```

Algoritmo 2: Algoritmo μGA

```
Entrada: SLA: C^c, T^c, A^c, C/h^c
//gera uma população de 5 individuos
Inicializa População(P)
//Calcula o fitnes de cada indivíduo
Avalia(P)
repita
    //Aplica operadores de recombinação
    Selection(P) Crossover(P) Mutation(P) Evaluate(P)
    se O melhor indivíduo não atualizou por ta iterações então
        Rinicializa(P)
    fim
até satisfazer o tempo limite;
//Retorna o melhor indivíduo
retorna SLA* : C*, T*, A*, C/h*
```

4. Resultados e discussões

Todos os experimentos foram executados no simulador CloudSim versão 3.0.3, utilizando um computador com um processador AMD Phenom(tm) II X6 1090T e com 16 GB de memória RAM, como parte da infraestrutura do LaSDPC (Laboratório de Sistemas Distribuídos e Programação Concorrente) [2]. O CloudSim foi configurado para executar com três tipos de instâncias de VMs, sendo elas baseadas nas instâncias ***m3.medium, m3.large*** **e** ***m3.xlarge*** da Amazon EC2[3].

No primeiro experimento, foi executado o algoritmo determinístico e coletado o tempo de resposta necessário para explorar todo o espaço de busca pela melhor solução dentro do intervalo definido. Esse intervalo possui um limite superior e inferior, sendo o número de VMs contratada mais 50% desse valor para limite superior e o número de VMs contratadas menos 50% desse valor para limite inferior. Por exemplo, se o cliente contratou 50 VMs, o intervalo fica entre 25 e 75 VMs.

No segundo experimento, foi executado o μAG durante 3 minutos ou até que o algoritmo encontrasse a solução ótima (critério de parada). Os experimentos foram executados 100 vezes para cada ambiente (*datacenters* variando de 10 até 100 VMs). Ao final, foi obtida uma taxa de acerto do μAG, ou seja, das cem vezes executadas, em quantas delas o algoritmo conseguiu chegar a mesma solução que o modelo determinístico. Nos experimentos realizados, o μAG obteve 100% de conformidade com o modelo determinístico.

Na Figura 1, observa-se a curva do tempo em relação a cada configuração. A linha contínua com marcador triangular representa o tempo de resposta obtido pelo algoritmo determinístico, a linha pontilhada com marcador retangular representa um limite de tempo de três minutos. Esse limite no contexto do problema é determinado como o tempo de resposta aceitável para o usuário e, por fim, a linha pontilhada com marcador em forma de losango, representa a média do tempo de resposta das 100 execuções para cada configuração em que foi aplicado o μAG. Analisando o tempo de resposta do algoritmo determinístico, observa-se que ele possui limite superior de complexidade polinomial, um tempo acima do tolerado pelo cliente. Por outro lado, o μAG obteve um tempo de resposta consideravelmente menor que o esperado pelo usuário.

[2]http://infra.lasdpc.icmc.usp.br

[3]https://aws.amazon.com/pt/ec2/instance-types/

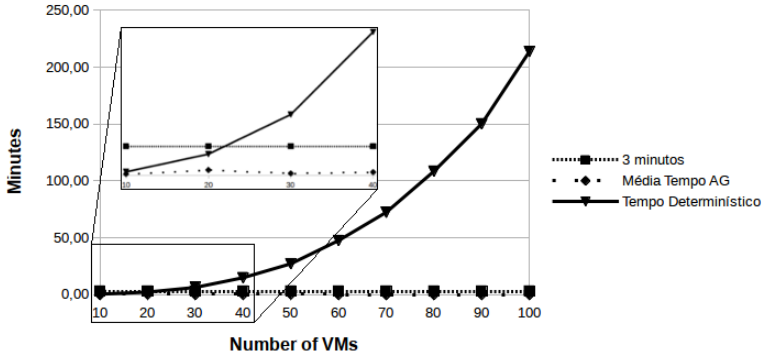


Figura 1. Gráfico do tempo de resposta do modelo determinístico e o μAGs.

5. Agradecimentos

Agradecemos à FAPESP (Processo: 2015-11623-4) pelo apoio financeiro e ao LaSDPC [4] pela infraestrutura oferecida durante o desenvolvimento do projeto.

6. Conclusão

Fornecer ao cliente da nuvem uma infraestrutura eficiente, respeitando o SLA e seus atributos de QoS e ainda tentando minimizar o custo não é uma tarefa trivial. No estudo apresentado neste artigo, alguns atributos de QoS foram mapeados, os quais determinam os critérios do SLA. Além disso, algoritmos que fornecem uma reconfiguração otimizada da infraestrutura baseada em tais critérios foram desenvolvidos e analisados. Os resultados apresentados demonstram que algoritmos evolutivos como o μGA são promissores na solução do problema de provisionamento de recursos, o que estimula o aprofundamento deste estudo. Em trabalhos futuros, novos experimentos serão realizados utilizando um ambiente real, além do desenvolvimento e análise de novos algoritmos meta-heurísticos.

Referências

Amazon (2015). Scaling plan. `http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/scaling\_plan.html`. Accessed: 01/20/2015.

Batista, B. G., Estrella, J. C., Ferreira, C. H. G., Leite Filho, D. M., Nakamura, L. H. V., Reiff-Marganiec, S., Santana, M. J., and Santana, R. H. C. (2015). Performance evaluation of resource management in cloud computing environments. *PloS one*, 10(11):e0141914.

Black, P. E. (2006). Manhattan distance. *Dictionary of Algorithms and Data Structures*, 18:2012.

Buyya, R., Broberg, J., and Goscinski, A. M. (2011). *Cloud Computing Principles and Paradigms*. Wiley Publishing.

Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A., and Buyya, R. (2011). Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50.

Ko, J. M., Kim, C. O., and Kwon, I.-H. (2008). Quality-of-service oriented web service composition algorithm and planning architecture. *Journal of Systems and Software*, 81(11):2079–2090.

[4]Página oficial do LaSDPC: `http://lasdpc.icmc.usp.br/`.

Avaliação de desempenho do algoritmo TOPSIS aplicado no Contexto de Internet das Coisas

Adriller Gênova Ferreira[1], **Luiz Henrique Nunes**[1], **Julio Cézar Estrella**[1]

[1]Instituto de Ciências Matemáticas e Computação – Universidade de São Paulo (USP)
São Carlos – SP – Brasil

adriller.ferreira@usp.br, lhnunes@icmc.usp.br, jcezar@icmc.usp.br

Abstract. *Recently, the number of devices has grown increasingly and it is hoped that increasingly devices will be connected to the Internet. The Internet of Things is composed of small sensors and actuators with Internet access and will play a key role in solving many challenges faced in today's society. The Visual Search for Internet of Things (ViSIoT) proposes a platform to help technical and non-technical users to discover and use sensors as a service. However the in this initial tool prototype don't concerns about the performance of the selection method used, called TOPSIS. In this context, this study presents a performance evaluation of this algorithm under different circumstances.*

Resumo. *Atualmente, o número de dispositivos computacionais tem crescido significativamente, e espera-se que cada vez mais esses sejam conectados à Internet. A Internet das Coisas é composta de pequenos sensores e atuadores com acesso à Internet e desempenhará um papel fundamental na resolução de muitos desafios existentes na sociedade atual. O projeto Visual Search for Internet of Things (ViSIoT) propõe uma plataforma para auxiliar os usuários técnicos e não-técnicos a descobrir e usar sensores como um serviço. Entretanto o protótipo inicial da ferramenta não se preocupa com o desempenho do algoritmo utilizado para a seleção dos sensores, denominado de TOPSIS. Neste contexto, este trabalho apresenta uma avaliação de desempenho de tal algoritmo sob diferentes circunstâncias.*

1. Introdução

Internet das Coisas (IoT) refere-se a uma visão na qual "coisas" tais como objetos, lugares e ambientes estarão interconectados por meio da Internet que passará de uma rede global de computadores para uma rede global de objetos [Koreshoff et al. 2013]. Seu principal objetivo é "criar um mundo melhor para os seres humanos", no qual os objetos presentes no dia a dia das pessoas serão capazes de captar suas necessidades e atuarem sem instruções explícitas [Dohr et al. 2010]. IoT engloba uma vasta quantidade de tecnologias de *hardware* e de *software*, tais como, identificadores de rádio-frequência (RFID, do inglês *Radio-Frequency Identification*), comunicação por campo de proximidade (NFC, do inglês *Near Field Communication*), redes de sensores (SN, do inglês *Sensor Networks*), *middlewares* e *frameworks* [Whitmore et al. 2015]. Dessa maneira, é esperado que os serviços oferecidos neste ambiente sejam sensíveis ao contexto e considerem as preferências e as necessidades dos usuários, de acordo com seu ambiente de atuação. Soluções que ofereçam formas de busca e seleção de sensores em uma escala

global e considerem o contexto no qual estão implantados serão essenciais para o seu crescimento [Romer et al. 2010]. As soluções existentes para a busca e seleção de sensores lidam com problemas de otimização multiobjetivo e utilizam algoritmos de análise de decisão multicritério para auxiliar sua resolução. Nesse contexto, a ferramenta ViSIoT[1] [Nunes et al. 2016] foi desenvolvida para ajudar os usuários a descobrir e usar sensores como um serviço para diferentes finalidades de aplicação. O objetivo deste trabalho é apresentar uma avaliação de desempenho do algoritmo TOPSIS, utilizado para a seleção dos sensores dessa ferramenta, sujeito a variação dos diferentes números de critérios que um usuário pode estabelecer e do número de sensores disponíveis.

2. Trabalhos Relacionados

O protótipo ViSIoT é baseado em uma arquitetura cliente-servidor [Nunes et al. 2015] para realizar experimentos em sistemas distribuídos. Estes estudos são utilizados como base deste trabalho pois foram capazes de configurar ambientes distribuídos de acordo com as preferências de um usuário. Esta arquitetura fornece acesso a um conjunto de sensores disponíveis como serviço, os quais podem alimentar qualquer tipo de aplicação ou middleware para prover sensoriamento como serviço. Além disso, ViSIoT pode abstrair a complexidade de se configurar tais ambientes, por meio de uma aplicação cliente que é responsável por implantar o mecanismo necessário para recuperar as informações desejadas [Nunes et al. 2016]. Os experimentos realizados com o ViSIoT utilizaram uma base de dados com informações geográficas dos sensores fornecidas pela API OpenWeatherMap. Os atributos pertencentes a cada sensor disponível nesta API foram gerados sinteticamente a partir das especificações dos sensores comercializados pela Bird Technologies [2].

3. Otimização Multiobjetivo

Uma variedade de problemas nas áreas de engenharia, indústria, computação e outras áreas envolvem a otimização de múltiplos objetivos. Em muitos casos, estes objetivos são definidos em unidades não comparáveis e apresentam algum nível de conflito entre eles. Em outras palavras, um objetivo não pode ser melhorado sem deteriorar outro objetivo. Por exemplo, uma empresa deseja adquirir equipamentos computacionais que apresentam um alto desempenho, com o intuito de acelerar o processamento de seus dados. Por outro lado, esta empresa também deseja reduzir os custos para adquirir tais equipamentos. Claramente, tais objetivos são conflitantes a medida que deseja-se adquirir equipamentos de alto desempenho ao mesmo tempo em que se reduz os seus custos. Além disso, seus objetivos são expressos em unidades de medida diferentes, dificultando assim sua comparação [Abraham and Jain 2005].

3.1. TOPSIS

O algoritmo TOPSIS (*Technique for the Order of Prioritisation by Similarity to Ideal Solution*) explora os valores dos critérios para fornecer um conjunto ordenado de alternativas. Dentre os domínios de aplicação que utilizam o TOPSIS incluem-se a gestão de cadeia de suprimentos e logística, design, engenharia e manufatura, gerenciamento de

[1] http://visiot.lasdpc.icmc.usp.br
[2] http://www.birdrf.com/

negócios e marketing, saúde e gestão ambiental [Behzadian et al. 2012]. TOPSIS ordena um conjunto de opções de acordo com a distância Euclideana para a solução ideal e ideal-negativa. A solução ideal corresponde ao melhor valor possível de cada critério, enquanto a solução ideal-negativa representa o pior valor de cada critério. As opções são classificadas de acordo com sua proximidade à solução ideal e distância da solução ideal-negativa [Tzeng and Huang 2011].

4. Avaliação de Desempenho

4.1. Ambiente avaliado

O ambiente de experimentos utilizado neste experimento é composto por apenas uma máquina física, que é responsável por hospedar a aplicação com os algoritmos de decisão multicriterio. As configurações de tal máquina correspondem ao Broker do cluster Andromeda e podem ser acessadas no site do Laboratório de Sistemas Distribuidos e Programação Concorrente - LaSDPC [3]. A realização dos experimentos considerou dois fatores: o número de sensores a serem selecionados (de 1.000 a 100.000 sensores com intervalos de 10.000 sensores entre cada experimento) e o número de critérios que um usuário pode definir (de 2 a 6 critérios).

4.2. Experimento

O experimento consiste em avaliar o método TOPSIS, que afeta diretamente o desempenho do ViSIoT. Para isso, foram selecionados de dois a seis números de propriedades, para serem testadas em uma variação de mil a cem mil sensores.

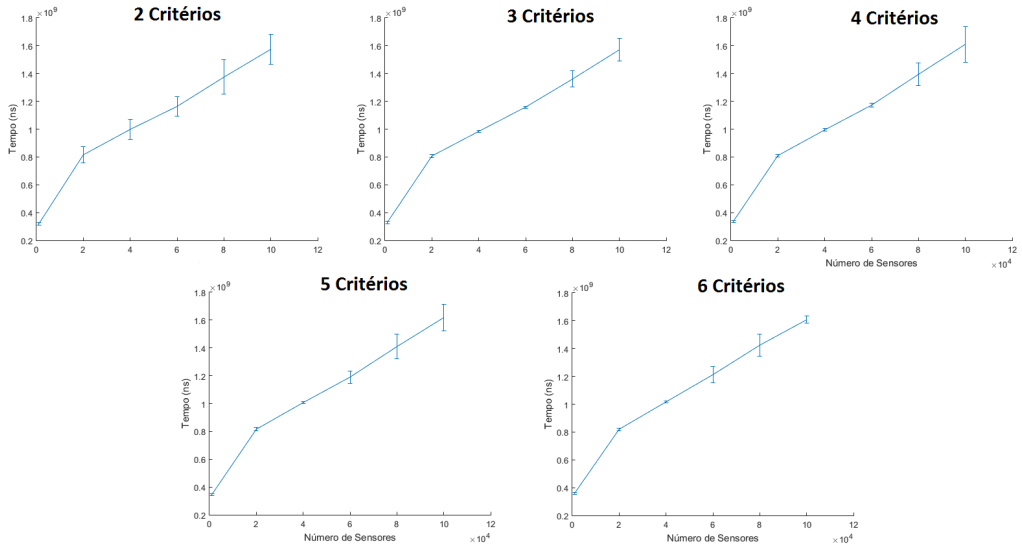


Figura 1. Desempenho avaliado

As funções objetivo utilizadas para maximizar (max) ou minimizar (min) um determinado critério respeitou a ordem: max(bateria), min(preço), min(desvio), max(frequência), min(consumo de energia), min(tempo de resposta) O resultado é o tempo necessário para a tomada de decisão do método, e pode ser verificado na Figura 1.

[3]Página oficial do LaSDPC: `http://lasdpc.icmc.usp.br/`.

É possível observar que aumentando o número de propriedades, o desempenho não alterou significativamente. Porém ao incrementar o número de sensores avaliados, o desempenho caiu drasticamente que pode impactar negativamente ambientes em que o tempo para a tomada de decisão é crucial. Além disso, verifica-se que há uma tendência de aumentar a variação dos tempos obtidos conforme aumenta o número de sensores.

5. Conclusão

No ambiente de Internet das Coisas é esperado que os serviços oferecidos sejam sensíveis ao contexto e considerem as preferências e as necessidades dos usuários, de acordo com seu ambiente de atuação. Os estudos do ViSIoT foram capazes de configurar ambientes distribuídos de acordo com essas preferências de um usuário, entretanto ainda eram escassos os estudos que avaliassem o desempenho do método utilizado. Levando em consideração os resultados mostrados neste artigo, verificamos que o aumento dos sensores utilizados na classificação multiobjetivo afeta com uma maior magnitude o tempo dos resultados obtidos, comparado com o aumento do número de propriedades. Os trabalhos futuros compreendem a comparação do algoritmo TOPSIS com outros algoritmos utilizados para a seleção de sensores.

Agradecimentos

Agradecemos à FAPESP (Processo: 2015/25793-9) pelo apoio financeiro e ao LaSDPC pela infraestrutura oferecida durante o desenvolvimento do projeto.

Referências

Abraham, A. and Jain, L. (2005). Evolutionary multiobjective optimization. In *Evolutionary Multiobjective Optimization*, pages 1–6. Springer.

Behzadian, M., Khanmohammadi Otaghsara, S., Yazdani, M., and Ignatius, J. (2012). Review: A state-of the-art survey of topsis applications. *Expert Syst. Appl.*, 39(17):13051–13069.

Dohr, A., Modre-Opsrian, R., Drobics, M., Hayn, D., and Schreier, G. (2010). The internet of things for ambient assisted living. In *Information Technology: New Generations (ITNG), 2010 Seventh International Conference on*, pages 804–809.

Koreshoff, T. L., Robertson, T., and Leong, T. W. (2013). Internet of things: A review of literature and products. In *Proceedings of the 25th Australian Computer-Human Interaction Conference: Augmentation, Application, Innovation, Collaboration*, OzCHI '13, pages 335–344, New York, NY, USA. ACM.

Nunes, L., Ferreira, C., Nakamura, L., Libardi, R., de Oliveira, E., Kuehne, B., Souza, P., Santana, R., Santana, M., Estrella, J., et al. (2015). Dca-services: A distributed and collaborative architecture for conducting experiments in service oriented systems. *International Journal of Services Computing (IJSC)*, 3(4):14–28.

Nunes, L. H., Estrella, J. C., Nakamura, L. H. V., Libardi, R. M. D. O., Ferreira, C. H. G., Jorge, L., Perera, C., and Reiff-Marganiec, S. (2016). A distributed sensor data search platform for internet of things environments. *International Journal of Services Computing (IJSC)*, 4(1):1–12.

Romer, K., Ostermaier, B., Mattern, F., Fahrmair, M., and Kellerer, W. (2010). Real-time search for real-world entities: A survey. *Proceedings of the IEEE*, 98(11):1887–1902.

Tzeng, G.-H. and Huang, J.-J. (2011). *Multiple attribute decision making: methods and applications*. CRC press.

Whitmore, A., Agarwal, A., and Da Xu, L. (2015). The internet of things—a survey of topics and trends. *Information Systems Frontiers*, 17(2):261–274.

Avaliação de algoritmos de escalonamento com inserção automatizada de escalonadores no RTsim

Fernanda F. Peronaglio[1], Aleardo Manacero[1], Renata S. Lobato[1], Roberta Spolon[2]

[1]Departamento de Ciências de Computação e Estatística
Universidade Estadual Paulista "Júlio de Mesquita Filho" (UNESP)
CEP – 15.054-000 – São José do Rio Preto – SP – Brasil

[2]Departamento de Computação
Universidade Estadual Paulista "Júlio de Mesquita Filho" (UNESP)
Bauru – SP – Brasil

fernanda.peronaglio@sjrp.unesp.br,{aleardo,renata}@ibilce.unesp.br

roberta@fc.unesp.br

Abstract. *Real Time systems have their performance and stability directly linked to the scheduling algorithm used to allocate tasks to CPUs. This dependence has led to the proposal of several different scheduling policies, according to different sets of restrictions and properties. Therefore, it is important to identify wich scheduler is more suited to a given set of tasks. This evaluation can be performed more efficiently by simulation. In this work we present a new modeling interface, added to RTsim (Real Time simulator), aiming to easily create and simulate a new scheduling policy. With this interface a system's designer can define the policy and evaluate its performance through a simple graphical interface.*

Resumo. *O desempenho e estabilidade de sistemas de tempo-real estão diretamente relacionados ao algoritmo de escalonamento usado para alocar tarefas às CPUs. Esta dependência levou à proposição de diferentes políticas de escalonamento, atendendo a diferentes conjuntos de restrições e propriedades. Logo é importante identificar qual escalonador é mais adequado para um dado conjunto de tarefas. Esta avaliação pode ser realizada mais eficientemente atráves de simulações. Neste trabalho apresentamos uma nova interface de modelagem, adicionada ao RTsim (Real Time simulator), buscando criar e simular uma nova política de escalonamento mais facilmente. Com esta interface um projetista de sistemas pode definir uma política e avaliar seu desempenho usando uma interface gráfica simples.*

1. Introdução

Sistemas de tempo-real envolvem aplicações que possuem restrições em seu tempo de execução. Esse tipo de sistema é normalmente usado para ambientes críticos, que têm prazos rígidos para atendimento. Como o não atendimento desses prazos pode levar a consequências graves, é preciso garantir que o ambiente tenha desempenho ótimo do ponto de vista de *througphput* [Farines et al. 2000].

O desempenho de sistemas de tempo-real está ligado ao bom casamento entre as tarefas que serão executadas e o algoritmo de escalonamento a ser utilizado na sua

alocação. Assim, no desenvolvimento desses sistemas é importante avaliar e identificar que escalonadores produzirão o melhor desempenho para o conjunto de tarefas que serão executadas. Uma das formas de avaliar esse desempenho é através de simulação, evitando riscos potenciais na sua avaliação no ambiente físico de tempo-real.

O RTsim (**R**eal-**T**ime **sim**ulator) [Manacero Jr. et al. 2001] é um simulador de escalonadores de tempo-real, originalmente desenvolvido para uso didático, mas que pode ser usado na avaliação dos escalonadores mais comuns na literatura. Entretanto, sua aplicabilidade na avaliação de situações reais é limitada ao número de escalonadores nativos do simulador.

Este trabalho apresenta o desenvolvimento de um componente no RTsim para fazer a inserção automatizada de novas políticas de escalonamento. Isso permite simular diferentes algoritmos sem a necessidade de implementá-los de forma nativa no RTsim.

2. Trabalhos relacionados

A simulação de escalonadores para sistemas de tempo-real pode ser feita por diversas ferramentas propostas na literatura. Com a diversidade de algoritmos e aplicações, esses simuladores foram projetados para atender diferentes objetivos, usando também diferentes formas de integrar escalonadores aos ambientes simulados. Assim, o STRESS [Audsley et al. 1994], trata de sistemas críticos e exige a programação manual das políticas de escalonamento. Já o Simso [Ch'eramy et al. 2013], trata de escalonadores em sistemas multiprocessados, também exigindo a programação manual dessas políticas.

Outros simuladores apresentam um número fixo de políticas de escalonamento previamente implementadas. Esse é o caso do Realtss [Diaz et al. 2007], que trabalha com algumas políticas mais gerais, sendo usado como auxiliar no ensino e pesquisa. Também é o caso do AURTSS [Yaashuwanth and Ramesh 2010], que é voltado para ensino e foi construído em plataforma Web.

Como se pode observar, ou os simuladores apresentam um conjunto fixo, e pequeno, de políticas nativas, ou exigem que o usuário programe manualmente a política de escalonamento a ser avaliada. No primeiro caso se perde em flexibilidade, enquanto no segundo se perde em facilidade de uso. Em nenhum caso existe a possibilidade de se agregar novos escalonadores através de uma interface intuitiva, como se propõe aqui.

3. Automatização da inserção de escalonadores

O RTsim [Manacero Jr. et al. 2001] é um simulador de algoritmos de escalonamento de tarefas de tempo-real. Nele estão disponíveis funcionalidades para simulação de várias políticas de escalonamento, tanto de forma direta como de forma guiada, bastante útil no ensino de tais escalonadores. Na forma guiada é possível ao usuário especificar um conjunto de tarefas e fazer o escalonamento manualmente, verificando se seu entendimento sobre o algoritmo está correto ou não. Essa verificação pode ser feita a cada ponto de escalonamento (modo treino) ou apenas ao final de um certo conjunto de pontos (mode teste). Existem oito escalonadores nativos na ferramenta, sendo cinco para sistemas monoprocessados e três para multiprocessados.

O uso do RTsim para a avaliação do desempenho de outras políticas de escalonamento, que é uma tarefa importante para ambientes reais, é prejudicado pois não permite

que o escalonador seja modelado através de um código programado pelo analista. Como isso descaracterizaria a ideia de modelagem intuitiva, foi preciso criar o componente objetivo deste trabalho. Com ele se gera um escalonador com base nas especificações matemáticas de seu comportamento, inseridas por interfaces gráficas que permitem ao usuário definir os parâmetros da política de alocação das tarefas no sistema.

A escolha de que parâmetros devem ser disponibilizados para a especificação matemática dos escalonadores foi conduzida a partir da avaliação dos algoritmos mais conhecidos da área. Nesse estudo se observou quais características eram mais importantes e utilizadas nas políticas de escalonamento e quais poderiam ser descritas por meio de uma fórmula. Os parâmetros escolhidos são os mais encontrados nos algoritmos de escalonamento, sendo definidos como:

- Período: Intervalo de tempo em que a tarefa ocorrerá novamente;
- Carga: Tempo necessário para que a tarefa termine sua execução;
- Carga Executada: Carga que a tarefa já executou até o momento;
- Carga Restante: Carga que falta para a tarefa terminar de executar;
- Deadline Relativo: Intervalo de tempo entre a chegada da tarefa e o instante máximo para terminar;
- Deadline Absoluto: Instante exato em que a tarefa deve terminar sua execução;
- Tempo Atual: Instante de tempo em que se está realizando o escalonamento.

A interface de formulação da política (Figura 1) apresenta esses parâmetros e uma série de botões de operações aritméticas, permitindo a construção de uma equação. Além disso é possível escolher se a ordenação das tarefas ocorrerá em ordem crescente ou descrescente. Nessa figura optou-se pela ordem crescente, para o algoritmo LST (*deadline* menos tempo atual menos carga ainda a executar).

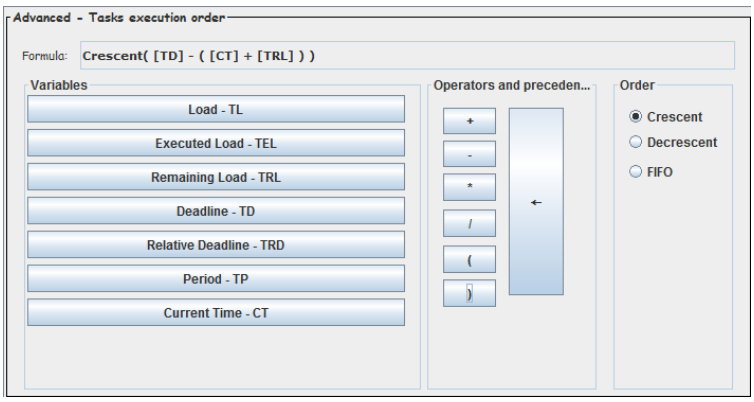


Figura 1. Interface gráfica do módulo para receber a fórmula do escalonamento

Após a definição da política ainda é possível definir tipos de tarefas (periódicas ou aperiódicas) ou ainda o modelo de atendimento de tarefas aperiódicas. Tarefas periódicas podem ser utilizadas em algoritmos de prioridade fixa ou dinâmica, e sua ordenação é sempre feita de acordo com a equação inserida. Já as aperiódicas são tratadas ou por execução em *background*, aproveitando os tempos de folga do escalonador; ou por interrupção, em que executa estas tarefas com prioridade máxima; ou *polling*, em que se cria uma tarefa periódica virtual que funciona como servidor e permite que as aperiódicas executem apenas quando a tarefa servidor for alocada.

4. Testes e Resultados

Os testes da nova funcionalidade envolveram a geração de alguns escalonadores e a validação dos resultados do simulador com resultados teóricos esperados. Em todos os casos a simulação produziu os resultados corretos. Apenas como um exemplo, um dos escalonadores gerados foi o algoritmo *Least Slack Time First* (LST), definido pela equação vista na Figura 1 (`CRESCENT([TD]-([CT]+[TRL]))`). A simulação do LST para um conjunto pequeno de tarefas periódicas produziu o escalonamento visto na Figura 2.

Tarefa	Carga	Período
P1	20	100
P2	40	180
P3	50	250
P4	80	400

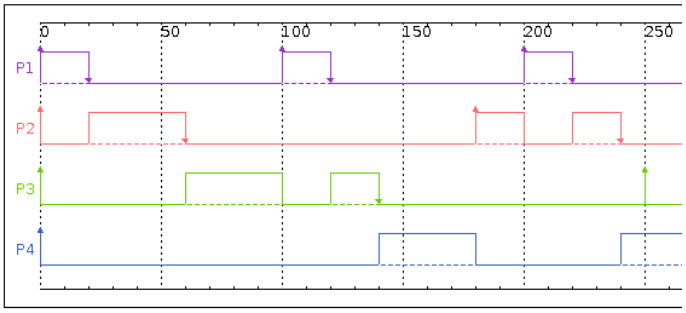


Figura 2. Gráfico de uma simulação do LST gerado de forma automatizada

5. Considerações Finais

Os testes realizados evidenciaram que o novo componente permite ao RTsim simular corretamente políticas não-nativas que possam ser definidas por relações algébricas. Isso dá ao simulador uma maior flexibilidade na avaliação de algoritmos de escalonamento que atendam uma grande variedade de necessidades. Ainda é necessário ampliar a funcionalidade desse componente para a geração de escalonadores para ambientes multiprocessados, trazendo uma maior cobertura de aplicação ao RTsim.

Referências

Audsley, N., Burns, A., Richardson, M., and Wellings, A. (1994). Stress: A simulator for hard real-time systems. *Software-Practice and Experience*, pages 543–564.

Ch'eramy, M., D'eplanche, A.-M., and Hladik, P.-E. (2013). Simulation of real-time multiprocessor scheduling with overheads. In *Intl Conf on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH 2013)*, pages 5–14.

Diaz, A., Batista, R., and Castro, O. (2007). Realtss: a real-time scheduling simulator. In *Electrical and Electronics Engineering, 2007. ICEEE 2007. 4th International Conference on*, pages 165–168. IEEE.

Farines, J.-M., Fraga, J. d. S., and Oliveira, R. d. (2000). Sistemas de tempo real. *Escola de Computação*, 2000:201.

Manacero Jr., A., Miola, M., and Nabuco, V. (2001). Teaching real-time with a scheduler simulator. In *Proc. of the 31st Frontiers in Education Conference*, pages T4D15–19. IEEE Press.

Yaashuwanth, C. and Ramesh, R. (2010). Web-enabled framework for real-time scheduler simulator (a teaching tool). In *Computer Research and Development, 2010 Second International Conference on*, pages 826–830. IEEE.

Análise de Desempenho de Máquinas Virtuais

Eder J. A. de Oliveira[1], Thais K. de Farias[1], Wilber S. C. Costa[1], Weine S. de Oliveira[1], Filippo Valiante Filho[1]

[1]Diretoria dos Cursos de Informática – Universidade Nove de Julho (UNINOVE)
São Paulo – SP – Brasil

eder.jose.a.oliveira@hotmail.com; farias.thais80@gmail.com;
wilberchaves@gmail.com; weine_e@hotmail.com; filippo@uni9.pro.br

Abstract. *This work briefly presents the importance of computer systems performance analysis and some free benchmark tools. The performance of five virtualized systems are evaluated in comparison with the performance of their respective guest systems, in order to have a detailed analysis of the implications of virtualization for the performance of the systems.*

Resumo. *Este trabalho apresenta brevemente a importância da análise de desempenho de sistemas computacionais e algumas ferramentas de software (benchmarks) gratuitas. São avaliados os desempenhos de cinco sistemas virtualizados em comparação com o desempenho de seus respectivos sistemas hospedeiros, de modo a ter-se uma análise mais detalhada das implicações da virtualização no desempenho dos sistemas.*

1. Introdução

O desempenho de um sistema computacional pode ser medido através de uma série de parâmetros quantitativos e, eventualmente qualitativos, em um processo ao qual se dá o nome de análise de desempenho. Através da análise de desempenho é possível comparar sistemas computacionais entre si, avaliar o custo-benefício de aquisição ou expansão de um sistema, avaliar ou prever, o impacto de alterações na configuração do sistema para seu desempenho final [Johnson e Margalho 2011]. O uso de *benchmarks* é uma das técnicas mais comuns para isso.

Neste trabalho, aplicaram-se as técnicas de análise de desempenho em máquinas virtuais, comparando o desempenho dos sistemas virtualizados com o desempenho dos respectivos sistemas hospedeiros, no caso diversos computadores do tipo PC, visando obter parâmetros concretos para o dimensionamento do sistema e configuração da virtualização.

2. Ferramentas de Análise de Desempenho (*Benchmark*)

Foram pesquisadas e avaliadas inúmeras ferramentas de *benchmark* para o sistema operacional Microsoft Windows. Adotou-se como critério de seleção *softwares* que tivessem licença livre, ou quando não houvesse alternativa disponível, licença gratuita, além de documentação detalhada e clara, especificando o que e de que forma é medido no teste. Os *benchmarks* selecionados para medir o desempenho de processamento foram o Linpack [Jack Dongarra 1979], baseado em uma biblioteca de álgebra linear, e o SuperPI [wPrime Systems 2012], que calcula até milhões de casas do número pi. O RAMspeed [Hollander and Bolotoff 2011] foi usado para medir o desempenho da memória de acesso

aleatório (RAM) e o DiskSpd [Microsoft 2014] para as unidades de armazenamento.

3. Metodologia

Após a seleção das ferramentas de benchmark, adotou-se, para virtualização, a plataforma VMware, pela liderança no mercado e compatibilidade com todas as arquiteturas de hardware e software utilizadas no projeto. A versão adotada neste projeto foi a VMware Workstation 12.

Para a padronização dos testes, de modo a gerar resultados diretamente comparáveis, a configuração das máquinas virtuais foi definida com os seguintes critérios:

- A memória RAM foi dividida na metade, garantindo que cada máquina virtual será testada com 50% da memória disponível independentemente da capacidade de cada máquina hospedeira;
- Foram alocados 20GB de espaço em disco para o sistema virtualizado, suficiente para a realização de todos os testes;
- Foi instalada a mesma versão do sistema operacional Windows da máquina hospedeira na máquina virtual;
- Os testes foram realizados variando-se o número de *threads* da CPU, ou seja, um teste com a máquina virtual com apenas 1 *thread*, um teste com o sistema virtualizado usando 2 *threads*, e assim por diante, até que fosse atingida a quantidade total de *threads* disponíveis em cada sistema hospedeiro;
- Para cada teste realizado em uma máquina virtual com N *threads* foi comparado seu desempenho com o desempenho da máquina real em sua totalidade de *threads*.
- Estabeleceu-se uma escala em porcentagem em relação aos resultados da máquina virtual com N *threads* e da máquina real em sua totalidade de *threads*, de modo a permitir a comparação dos diferentes sistemas de forma coerente;
- Após o refinamento dos testes, para a coleta dos resultados os *benchmarks* foram executados automaticamente a partir de um arquivo de lote padronizado em todos os sistemas.

4. Análise dos Resultados

As máquinas físicas adotadas nos testes são detalhadas na Tabela 1.

As métricas adotadas foram o tempo de execução para os testes de CPU e as taxas de leitura e escrita em megabytes por segundo para as memórias principal e secundária. Os tempos médios obtidos foram de 30,05s para cálculos lineares em uma matriz tamanho 4.000 x 4.000 e 100,30s para cálculos de pi até 80.000 casas decimais. Na memória RAM, para blocos de escrita de 1024kB obteve-se a taxa média de 18.838,31 MB/s e nos discos a média de 22.458,33 MB/s na leitura de blocos de 1024kB.

Os resultados dos testes realizados na CPU podem ser vistos na Figura 1. O eixo vertical representa o desempenho medido de forma percentual, onde 100% corresponde ao desempenho da máquina hospedeira. O eixo horizontal corresponde à quantidade de recursos utilizados na virtualização. No caso, a quantidade de threads usadas na máquina

virtual. As medidas são pontuais e as curvas traçadas para mostrar a tendência.

Tabela 1. Configurações das máquinas hospedeiras.

Máquina	CPU	Memória RAM	Placa Mãe	Hard Disk (HD)	Versão do Windows
01	AMD FX8350, 4.00GHz 8 threads	2 x 4GB @669.7MHz DDR3 dual channel	Gigabyte 970A-DS3P	SSD,128GBs, SATA3, Kingston SV300S37A	Windows 7 Ultimate 64-bits
02	AMD FX6300, 3.5GHz 6 threads	2 x 4GB @666.6MHz DDR3 dual channel	ASUSTeK M5A78L-M LX/BR (AM3R2)	160GBs, SATA3, Excelstor J8160S	Windows 10 Professional 64-bits
03	AMD FX8350, 4.62GHz 8 threads	4 x 4GB @850MHz DDR3 dual channel	Gigabyte 990FXA UD3	SSD, 128Gbs SATA3, 850 Evo	Windows 7 Professional 64-bits
04	Intel Core i5-4210U 2.70GHz 4 threads	2 x 4GB @800MHz DDR3 dual channel	Dell 0C1F3F Version A02	1 TB SATA 3.	Windows 8.1 64-bits
05	Intel Core i7-3770K 3.50GHz 8 threads	2 x 4GB @798.1MHz DDR3 dual channel	ASUSTeK P8B75-M LX	2 TB, SATA 3, Seagate ST2000DM001	Windows 10 Professional 64-bits

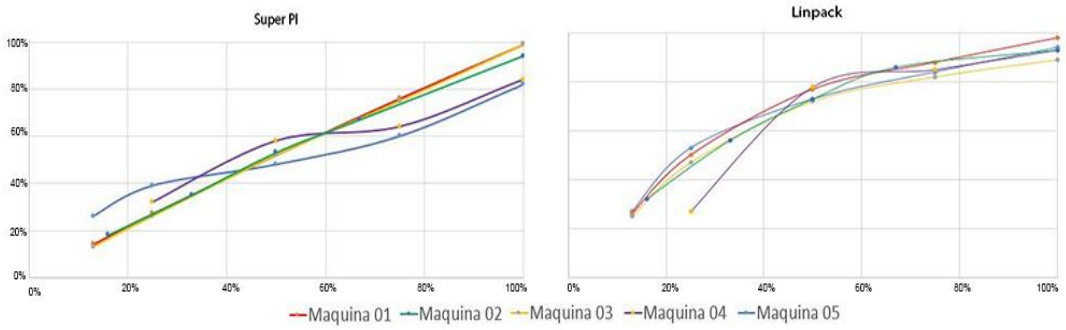


Figura 1. Resultados usando os *benchmarks* SuperPi (esquerda) e Linpack

Analisando-se os resultados, observa-se que o comportamento das máquinas é bastante similar, sendo possível agrupar-se claramente as CPUs AMD, que apresentaram comportamento mais linear e Intel, que apresentaram uma curva mais irregular em função da utilização do *hyperthreading*.

Os resultados dos testes realizados no sistema de armazenamento podem ser vistos na Figura 2, onde os eixos seguem o mesmo padrão da Figura 1. Analisando-se os resultados, observa-se que em relação aos SSDs à uma grande perda de desempenho na virtualização, destaca-se que apenas usando 50% dos *threads* que foi alcançado um resultado satisfatório.

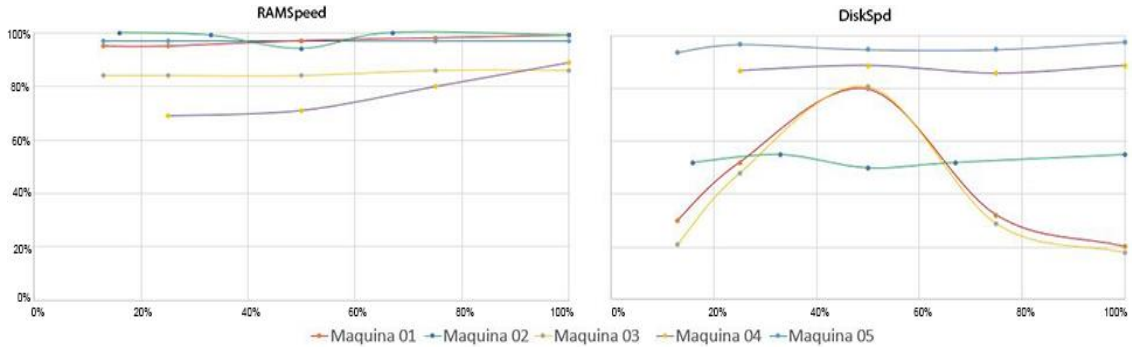


Figura 2. Resultados usando os *benchmarks* RAMspeed (esquerda) e DiskSpd

Conclusão

Observou-se que sistemas virtualizados chegam a até 99% do desempenho do hospedeiro. Em cálculo de ponto flutuante, processadores Intel mantêm um padrão de ganho de desempenho a cada thread alocada, enquanto os AMD apresentam ganhos menos significativos a partir de 50% das threads, devido a sua arquitetura. Para os sistemas com HD convencional o desempenho se manteve constante independentemente da quantidade de threads, enquanto os com SSD conseguem melhor desempenho com 50% das threads alocadas. Concluímos que de acordo com o objetivo da virtualização o melhor a ser feito é virtualizar um sistema que tenha entre 50% a 100% dos recursos do hospedeiro.

Em uma próxima etapa da pesquisa serão avaliados os desempenhos de múltiplas máquinas virtuais executadas paralelamente em um mesmo sistema hospedeiro.

Referências

Hollander, R. and Bolotoff, P (2011) “RAMspeed, a cache and memory benchmarking tool”, disponível em http://alasir.com/software/ramspeed/.

Jack Dongarra 1979 “Linpack, a linear equations tool”, disponível em http://www.netlib.org/benchmark/hpl/.

Johnson, T. and Margalho, M. (2011) “Avaliação de Desempenho de Sistemas Computacionais”, LTC – Livros Técnicos e Científicos Editora Ltda.In: New Trends in Animation and Visualization, Edited by Nadia Magnenat-Thalmann and Daniel Thalmann, John Wiley & Sons ltd., England.

Microsoft Corporation (2014) “Diskspd Utility: A Robust Storage Testing Tool”, disponivel em https://gallery.technet.microsoft.com/DiskSpd-a-robust-storage-6cd2f223.

wPrime Systems 2012, SuperPI website, disponível em http://www.superpi.net. “MultiThreadedPI@LINPACK, union in between Linpack benchamark and SuperPI benchmark”, disponivel em http://hwbot.org/benchmark/multi_threaded_linpack/ .

Estratégias de Alto Desempenho para Rotação de Proteínas

Dionisius O. Mayr[1], **Hélio C. Guardia**[1]

[1]Departamento de Computação – Universidade Federal de São Carlos (UFSCar)
São Carlos – SP – Brazil

dionisiusom@gmail.com, helio@dc.ufscar.br

Abstract. *Protein docking analysis is used in many industries, such as in the search for new drugs. In this analysis, the effect of positioning two molecules is studied while considering their rotations in a tridimensional space. Given the infinite amount of possible combinations in this study, the efficiency of the rotation mechanism is fundamental. The research presented here seeks to investigate efficient ways to realize such rotations. In order to do so, we consider the use of parallel programming strategies using shared memory with multiple threads and accelerators. This project's results shall be used by a mechanism that selects the relevant rotations to be assessed.*

Resumo. *A análise do atracamento de proteínas é utilizada em diferentes indústrias, como na busca por novos fármacos. Nesta análise, estuda-se o efeito do posicionamento de duas moléculas considerando suas rotações num espaço tridimensional. Dado o infindável número de posicionamentos possíveis, a eficiência do mecanismo de rotação é fundamental. A pesquisa apresentada neste trabalho busca formas eficientes de realizar tais rotações. Para tanto, considera-se o uso de estratégias de programação paralela usando memória compartilhada com múltiplas threads e aceleradores. Os resultados do projeto devem ser usados por um mecanismo de escolha das rotações relevantes para avaliação do atracamento de proteínas.*

1. Introdução

Proteínas desempenham um papel central em diversos processos, como na catálise de enzimas [Russel et al. 2004], na replicação do DNA e em diferentes processos industriais. Compreender como proteínas interagem entre si é fundamental para aprimorar uma variedade de procedimentos. Em razão da importância e diversidade de campos de aplicação, simular esta interação é um problema de grande relevância na Biologia Computacional, sendo estudado há mais de 40 anos [Shimoda et al. 2013].

O problema do atracamento de proteínas (*Protein-Protein Docking*) consiste em predizer o complexo proteico resultante da interação entre proteínas baseando-se em seu formato e em suas qualidades físico-químicas [Kohlbacher et al. 2001]. Para tanto, dados das estruturas de proteínas são tipicamente obtidos através de um repositório de moléculas biológicas chamado *Protein Data Bank* (PDB) [RCSB 2016], com acesso livre. O formato PDB usado no armazenamento carrega informações a respeito da molécula e seus aminoácidos, assim como sua estrutura tridimensional.

Após obter dados sobre as proteínas de interesse neste formato, é necessário representá-las de forma eficiente na memória e rotacionar as representações a fim de testar

cada uma das configurações desejadas na investigação do atracamento entre elas. Como uma quantidade muito vasta de configurações devem ser testadas, o estágio de rotação das proteínas se torna relevante para a eficiência deste problema. Em virtude disso, se faz indispensável determinar uma maneira eficiente para realizar a rotação das proteínas. Para tanto, este estudo apresenta os resultados preliminares de um trabalho em andamento, onde considera-se o uso de técnicas de programação paralela com memória compartilhada, aceleradores gráficos e coprocessadores da arquitetura *Many Integrated Cores*.

2. Representação e visualização de proteínas

Para a rotação das representações espaciais de proteínas, é preciso estudar o formato em que suas estruturas estão armazenadas em um arquivo PDB, a fim de determinar como representar suas coordenadas de maneira eficiente na memória. Esta representação deve também possibilitar que o atracamento seja avaliado após a rotação e o encaixe das proteínas. Assim, inicialmente, a representação escolhida neste trabalho foi um vetor de coordenadas espaciais, contendo as coordenadas dos vértices da superfície da molécula estudada. Para a visualização das representações das proteínas é possível armazenar esta representação no formato OFF e também definir faces compostas por estes vértices.

Para verificar se a rotação realizada é coerente, decidiu-se utilizar um mecanismo de visualização de proteínas no espaço. Para tanto, é necessário renderizar a proteína representada no formato favorável às rotações. O mecanismo escolhido foi o software MeshLab [Cignini 2016], capaz de exibir objetos tridimensionais representados no formato OFF como malhas de triângulos. A figura 1 mostra um exemplo da visualização da proteína 1RUY nos formatos PDB e OFF, contendo 181489 vértices.

Para converter uma proteína armazenada no formato PDB, é necessário gerar sua superfície de Connolly partindo do arquivo PDB e então converter esta superfície para o formato OFF. Pode-se realizar tal procedimento utilizando a ferramenta EDTSurf [Xu and Zhang 2016].

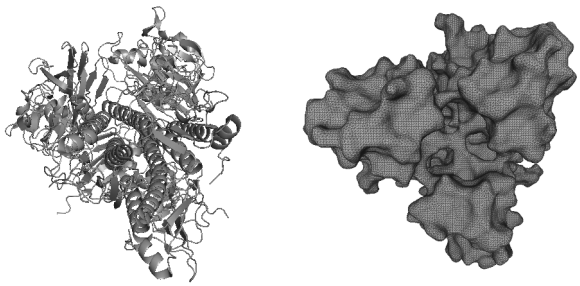


Figura 1. Modelo da proteína 1RUY nos formatos PDB e OFF, respectivamente

3. Rotação de proteínas

A etapa de maior custo computacional deste estudo é a etapa de rotação. Cada um dos vértices da representação espacial precisa ser rotacionado inúmeras vezes a fim de testar as configurações desejadas. Isto justifica o uso do formato OFF, pois ele contém apenas informações relativas às coordenadas tridimensionais dos vértices da malha de triângulos e as faces que estes vértices compõem.

Para determinar as coordenadas resultantes (x', y', z') após aplicar uma rotação de θ graus no sentido anti-horário em torno do eixo z em um vértice (x, y, z) da representação espacial, é necessário:

- Fixar a coordenada z, ou seja, $z' = z$;
- Manter a mesma distância do vértice original e do vértice rotacionado à origem;
- Adicionar θ graus entre a reta que liga o vértice original à origem e a reta que liga o vértice rotacionado à origem.

Isto pode ser realizado da seguinte forma:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad \text{, ou seja,} \quad \begin{aligned} x' &= x \cdot \cos\theta - y \cdot \sin\theta \\ y' &= y \cdot \cos\theta + x \cdot \sin\theta \end{aligned}$$

As rotações em torno dos eixos x e y podem ser realizadas de maneira análoga. Tendo um vetor com as coordenadas de cada um dos vértices da representação da proteína, pode-se rotacioná-la de maneira paralela em torno do eixo z executando o seguinte algoritmo:

Algoritmo 1 Rotação de θ graus em torno do eixo z

```
1: procedure RotZ(vertex * V, double θ, int size)
2:     #pragma omp parallel for shared(V)
3:     for int i = 0 ; i < size ; i++ do
4:         V'[i].x = V[i].x · cos θ − V[i].y · sin θ;
5:         V'[i].y = V[i].y · cos θ + V[i].x · sin θ;
```

3.1. Uso de aceleradores

O paralelismo de *threads* combinado com o uso de aceleradores mostrou ser capaz de reduzir o tempo de processamento requerido [Shimoda et al. 2013] na análise de atracamento de proteínas. Contudo, determinar qual tipo de acelerador e quais técnicas de programação paralela utilizar não é trivial. Assim, o objetivo deste trabalho é explorar como o uso de aceleradores pode auxiliar o processamento e quais são as modificações necessárias para melhor utilizar uma Unidade de Processamento Gráfico (GPU) ou o co-processador Xeon Phi. Diferentes estratégias serão abordadas e analisadas para extrair o melhor desempenho de cada um dos aceleradores.

O uso de *profilers* será um ponto central nesta etapa, a fim de direcionar quais parcelas do código devem ser otimizadas e paralelizadas. Será utilizado o *profiler* OProfile [Cohen 2016] para coletar dados sobre a eficiência do programa ainda sequencial. Após implementar os aprimoramentos utilizando os aceleradores desejados, novos dados serão coletados e analisados. Tais testes serão repetidos para diversas configurações de proteínas, estratégias de programação paralela e diferentes aceleradores a fim de coletar dados de maior credibilidade.

4. Resultados preliminares e conclusões

Tratada de forma direta, a rotação de imagens 3D é um processo embaraçosamente paralelo. Para a representação da proteína exibida na figura 1, por exemplo, a paralelização

utilizando OpenMP em um sistema com 4 cores, e 4.00GHz, com o processador Intel Core i7-4790K obteve um tempo de 1006 milissegundos com 4 *threads* comparado com 3266 milissegundos na versão sequencial após rotacionar a proteína 1000 vezes utilizando o algoritmo 1. A figura 2 mostra o *speedup* de um estudo preliminar de rotação.

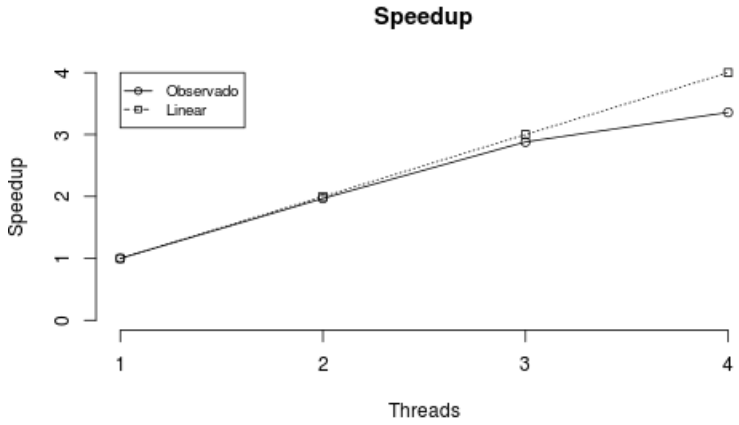


Figura 2. Gráfico de *speedup* da rotação

Novos estudos se fazem necessários sobre a granularidade das operações e a escalabilidade, principalmente considerando o uso de aceleradores. Assim, este trabalho prevê uma análise estatística comparativa entre as diferentes técnicas e aceleradores empregados, estudando as restrições e vantagens de cada método. Gráficos de *speedup*, eficiência e de tempo de execução serão gerados, sempre em função do grau de paralelismo empregado. Com estes gráficos, será analisada a escalabilidade das soluções propostas. De posse desta análise, deseja-se determinar a escalabilidade das soluções e também qual acelerador se mostra mais adequado para este processamento.

Referências

Cignini, P. (2016). Meshlab. Disponível em: http://meshlab.sourceforge.net/ Acesso em: 14 mar. 2016.

Cohen, W. (2016). Oprofile. Disponível em: http://oprofile.sourceforge.net/about/ Acesso em: 16 mar. 2016.

Kohlbacher, O., Burchardt, A., Moll, A., Hildebrant, A., Bayer, P., and Lenhof, H. P. (2001). A nmr-spectra-based scoring function for protein docking. pages 184–192. RECOMB.

RCSB (2016). Disponível em: http://www.rcsb.org/pdb/ Acesso em: 16 mar. 2016.

Russel, R. B., Alber, F., Aloy, P., Davis, F. P., Korkin, D., Pichaud, M., Topf, M., and Sali, A. (2004). A structural perspective on protein-protein interactions. pages 313–324. Current Opinion in Structural Biology.

Shimoda, T., Ishida, T., Suzuki, S., Ohue, M., and Akiyama, Y. (2013). Megadock-gpu: Acceleration of protein-protein docking calculation on gpus. pages 883–889. BCB.

Xu, D. and Zhang, Y. (2016). Edtsurf: Quick and accurate construction of macromolecular surfaces. Disponível em: http://zhanglab.ccmb.med.umich.edu/EDTSurf/ Acesso em: 5 jun. 2016.

Um *back end* baseado no compilador LLVM para aceleração de aplicações em FPGAs

Gabriel N. Tutui[1], **Rafael Gasperetti**[1], **Ricardo Menotti**[1]

[1]Departamento de Computação – Universidade Federal de São Carlos (UFSCar)
Caixa Postal 676 – 13565-905 – São Carlos – SP – Brasil

{{558290@comp}, {rafael.gasperetti, menotti}@dc}.ufscar.br

Abstract. *The performance increase stagnation in general-purpose processors in recent years motivates research in other forms of accelerating applications. The use of FPGAs to run algorithms directly in hardware is a good alternative, given the reconfigurable nature of the platform. However, the low degree of absctraction in hardware description languages makes programming these devices harder. A solution to this problem is the use of techniques to synthesize the applications in hardware from high-level languages, such as C. This article describes the combination of two compilers with this objective.*

Resumo. *A estagnação no aumento de desempenho em processadores de uso geral nos últimos anos motiva pesquisas em outras formas de aceleração de aplicações. O uso de FPGAs para execução de algoritmos diretamente em* hardware*, dada a natureza reconfigurável da plataforma, se mostra como uma boa alternativa. No entanto, o baixo grau de abstração nas linguagens de descrição de hardware torna a programação desses dispositivos mais difícil. Uma solução para o problema são técnicas para sintetizar as aplicações em hardware a partir de linguagens de alto nível, como C. Neste artigo é descrita a combinação de dois compiladores para este fim.*

1. Introdução

Os FPGAs têm sido usados com sucesso em sistemas embarcados e mais recentemente na aceleração de aplicações de alto desempenho. O principal obstáculo para a disseminação desta tecnologia está na complexidade de se projetar as arquiteturas usando linguagens de descrição de hardware. Por esta razão, a geração automática de hardware a partir de programas escritos em linguagens de alto nível sempre foi um objetivo buscado.

Essa ideia vem sendo aperfeiçoada há algum tempo, tendo em vista vários projetos que foram desenvolvidos para tentar solucionar o problema. Um dos exemplos é o compilador Trident [Tripp et al. 2005], que foi uma das primeiras ferramentas com esta finalidade. A partir de programas escritos em linguagem C, a ferramenta produz circuitos que exploram o paralelismo quando se trata de algoritmos de ponto flutuante. Posteriormente, esta ferramenta se tornou um produto comercial.

Outra ferramenta com bastante impacto nessa área é o compilador LegUp [Canis et al. 2011], que é uma ferramenta de código aberto que recebe um código C como entrada e o compila para uma arquitetura FPGA, que também pode ser híbrida, baseada no processador MIPS. Os resultados obtidos neste projeto são

comparáveis com a qualidade de soluções de hardware de ferramentas comerciais de síntese de alto nível.

Atualmente há limitações nas tentativas de geração de hardware otimizado o suficiente para aplicações de alto desempenho e outras [Menotti et al. 2012]. Essas limitações se devem a falta de exploração dos recuros disponíveis nos FPGAs, além do fato dos compiladores se apoiarem em técnicas de software, pouco adequadas à geração de hardware.

A ideia principal deste trabalho é obter automaticamente os resultados das técnicas do compilador LALP [Menotti et al. 2012], a partir da linguagem intermediária do compilador do LLVM[1], que por sua vez é obtida a partir do *front end* Clang [Lattner 2008]. Desta forma, poderão ser aproveitadas otimizações já existentes neste compilador que sejam interessantes para geração de *hardware*, bem como o *back end* usado no compilador LALP.

2. Abordagem

O LLVM [Lattner 2008] é uma infraestrutura de código aberto para construção de compiladores, construído como um conjunto de bibliotecas reutilizáveis. Para entender o projeto, é importante ressaltar a arquitetura tradicional de compiladores:

- Um *front end*, que processa as partes específicas a linguagem por meio de um *parser*, construindo uma representação intermediária agnóstica a linguagem e plataforma a ser passada para a próxima etapa.
- Um *middle end*, que utiliza a representação intermediária para realizar otimizações independentes de plataforma para tornar o código mais eficiente.
- Um *back end*, que traduz o programa da representação intermediária otimizada para código de máquina para a plataforma alvo, e realizando otimizações específicas da plataforma.

A parte principal das bibliotecas do LLVM são ferramentas para construção do *middle* e *back end* — se destacando pela representação intermediária acessível e legível textualmente, arquitetura altamente modular e design independente de linguagem e plataforma. Além disso, o projeto também contempla o Clang, um *front end* para C/C++, que será utilizado em nossa aplicação.

O compilador LALP é baseado na biblioteca de componentes Nenya [Cardoso 2000] e em técnicas de *loop pipelining* avançadas, que resultam em arquiteturas otimizadas para execução de loops em hardware. Os avanços recentes do compilador incluem escalonamento automático no acesso a dados em memórias e suporte a ponto flutuante [Oliveira et al. 2015].

A abordagem usada neste projeto consiste na criação de um novo *back end*, utilizando as ferramentas do LLVM para extrair as informações necessárias para construir um modelo de hardware a partir do programa em C, utilizando principalmente a *Abstract Syntax Tree* (AST) gerado pelo *front end* Clang. Com isso, é possível identificar uma estrutura de componentes da biblioteca Nenya com funcionamento equivalente ao programa original, para ser executado em um FPGA. Com a finalidade de facilitar a implementação

[1] `http://www.llvm.org`

e possibilitar a otimização do circuito com algoritmos clássicos de escalonamento, a estrutura gerada será formada como um grafo — representando cada componente como um nó e cada sinal como uma aresta.

3. Resultados preliminares

A estrutura básica do *back end* foi construída, mapeando as entidades necessárias para a construção do grafo representativo do hardware. Para isso, foram criadas as seguintes classes:

- **Design:** Classe que armazena um conjunto de componentes e sinais, bem como metódos de acesso e manipulação desses, para que possam ser instanciados na geração automática de código HDL. A entidade representa o grafo em si.
- **Component:** Uma definição geral para todos os componentes, de forma que a instanciação e extração das informações destes geralmente é feita por essa classe, com a eventual necessidade de funções virtuais a serem definidas nas herdeiras (classes especifícas para cada componente existente na biblioteca Nenya). A entidade representa um nó do grafo.
- **Port:** Representa as portas - estruturas para a comunicação de dados utilizadas pelos componentes na linguagem, e trata a ligação desses a partir dos sinais (tanto de entrada quanto de saída).
- **GenericMap:** Representa os *generic maps* - estruturas que são utilizadas na atribuição de valores parametrizados para configuração do comportamento dos componentes ou larguras das portas. Uma função importante da classe é a associação dos parâmetros atribuídos.
- **Signal:** Representa os sinais a serem utilizados no programa, guardando informações como tipo e largura de dados. A entidade representa uma aresta do grafo.
- **VHDLGen:** Uma classe, implementada no padrão *Singleton*, que recupera as informações mapeadas de forma a gerar as declarações do código VHDL associado a cada estrutura.

Essa etapa do trabalho foi testada inicialmente com a instanciação e ligação manual dos componentes de programas simples, na qual foi verificada que os códigos VHDL foram gerados e funcionavam corretamente. A geração automática de *hardware*, a partir de um programa em C, foi iniciada e se encontra em andamento. No momento, é possível realizar a extração das informações da AST e identificar alguns dos componentes e sinais necessários para a geração: identificar vetores como blocos de memória, operadores binários de soma e multiplicação, atribuição ampliada (operadores += e $\star$=) e laços de repetição na forma `For()` com incremento unitário e valor inicial zero. Parte dos tratamentos para variáveis de retorno também está implementada, bem como alguns casos de atribuição de sinais entre os componentes. Com essa parte, já foi possível gerar um exemplo simples de um programa capaz de calcular produto vetorial.

A extração das informações foi feita utilizando a biblioteca LibClang, uma interface de alto nível em C que dá acesso às informações geradas pelo compilador. A escolha foi feita pelo fato dessa interface, em contraste com outras disponibilizadas como a LibTooling, ser desenvolvida tentando manter a estabilidade e compatibilidade retroativa entre diferentes versões — fator importante dado a alta frequência de atualizações no projeto do

compilador. A biblioteca fornece funções para navegar na AST e retirar as informações relevantes ao *back end*.

Os códigos gerados com a implementação atual foram os mesmos obtidos em [Menotti et al. 2012], tendo assim os mesmos resultados de *benchmark* em termos de espaço ocupado no dispositivo e tempo de execução, porém partindo de implementações em C ao invés de LALP.

4. Trabalhos futuros

A etapa atual de desenvolvimento deste trabalho concentra-se no refinamento das ferramentas de geração do hardware, de forma que ele contabilize mais casos possíveis — implementar detecção dos demais componentes, tornar a implementação deles mais genérica e melhorar a detecção dos sinais — para que esse possa funcionar com programas mais complexos. Após isso, é planejada a implementação de algoritmos simples de escalonamento para otimização dos circuitos.

Outras abordagens promissoras para geração de hardware são aquelas que usam descrições com paralelismo explícito, como OpenCL e OpenMP. Vários recursos destas bibliotecas são suportadas pelo compilador LLVM e seu uso será considerado em novas etapas deste trabalho. Estas descrições são bastante adequadas para a geração de aceleradores, uma vez que se pode descrever explicitamente a parte do código que será executada o acelerador.

Referências

Canis, A., Choi, J., Aldham, M., Zhang, V., Kammoona, A., Anderson, J. H., Brown, S., and Czajkowski, T. (2011). LegUp: high-level synthesis for FPGA-based processor/accelerator systems. In *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, pages 33–36. ACM.

Cardoso, J. M. P. (2000). *Compilação de Algoritmos em Java para Sistemas Computacionais Reconfiguráveis com Exploração do Paralelismo ao Nível das Operações*. PhD thesis, Universidade Técnica de Lisboa.

Lattner, C. (2008). LLVM and Clang: Next generation compiler technology. In *The BSD Conference*, pages 1–2.

Menotti, R., Cardoso, J. M. P., Fernandes, M. M., and Marques, E. (2012). LALP: A Language to Program Custom FPGA-based Acceleration Engines. *International Journal of Parallel Programming*, 40(3):262–289.

Oliveira, C. B. D., Menotti, R., Cardoso, J. M. P., and Marques, E. (2015). A special-purpose language for implementing pipelined FPGA-based accelerators. In *2015 Forum on Specification and Design Languages (FDL)*, pages 1–8.

Tripp, J., Peterson, K., Ahrens, C., Poznanovic, J., and Gokhale, M. (2005). Trident: an FPGA compiler framework for floating-point algorithms. *International Conference on Field Programmable Logic and Applications*, 0:317–322.

Geração de simulador de eventos discretos aplicado a circuitos sequenciais

Renan Galeane Alboy[1], **Gabriel Covello Furlanetto**[1], **Aleardo Manacero**[1], **Renata Spolon Lobato**[1], **Roberta Spolon**[2]

[1]Departamento de Ciência da Computação e Estatística (DCCE)
Universidade Estadual Paulista "Júlio de Mesquita Filho"
São José do Rio Preto – SP – Brasil

[2]Departamento de Computação
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Bauru -São Paulo - Brasil

Abstract. *Simulators are widely used as tools to analyse, evaluate or validate different systems or even environments. Their use can reduce the time and cost needed to develop a new product since simulations can be performed at very early phases of project. However, most of the available simulation tools are not easy to use or tailored to specific needs. In this work we present a tool that can create a discrete-event simulator, with a GUI-based interface, from relatively simple formulation given by its user. The specification of the required parameters to create simulators systems characterized by transfer function, such as sequential circuits, is present here.*

Resumo. *Simuladores são amplamente usados como ferramentas para análises, avaliação ou validação de diferentes sistemas ou ambientes. Seu uso reduz o tempo e custo necessário para desenvolver um novo produto porque podemos simular desde as primeiras fases do projeto. Entretanto, a maioria dos simuladores disponíveis não são fáceis de usar ou não são projetados para necessidades específicas. Neste trabalho se apresenta uma ferramenta com a qual se pode criar um simulador de eventos discretos, com interface gráfica, a partir de formulação relativamente simples, dada por seu usuário. A especificação dos parâmetros exigidos para criar simuladores de sistemas baseados em funções de transferência tais como circuitos sequenciais, é apresentado aqui.*

1. Introdução

O uso de ferramentas de simulação para avaliar o funcionamento de sistemas, das mais variadas áreas, ocorre com grande frequência. Simuladores permitem reduzir o custo de projetos, desenvolvimento e verificação de sistemas, uma vez que podem atuar antes da produção de um protótipo.

Entretanto o uso de simuladores é dificultado por diferentes motivos. Um problema é que parte das ferramentas disponíveis é difícil de usar, exigindo conhecimentos de programação para a elaboração dos modelos do sistemas avaliados. Outro problema é que nem sempre tratam dos aspectos que necessitam ser avaliados, ou exigindo informações demais ou deixando de tratar algum parâmetro. Um último inconveniente é a disponibilidade das ferramentas, uma vez que muitos simuladores descritos na literatura ou não são

encontrados ou existem apenas em versões comerciais de alto custo. Uma das áreas com tais problemas é a de projeto de circuitos digitais, em que temos simuladores bastante complexos, como o ModelSim [Modelsim 2016], ou com recursos não tão amplos, como o Qucs (Quite universal circuit simulator) [Qucs 2016].

Para contornar parte dos problemas apresentados, principalmente os relacionados com facilidade de uso e adequação de abrangência, é que se propõe uma ferramenta para geração de simuladores baseados em eventos. A ferramenta proposta, Yasc (Yes, a simulator compiler), permite que um usuário qualquer, tal como um projetista de circuitos sequenciais, possa criar um simulador totalmente padronizado às suas necessidades. Esse simulador é criado a partir da especificação de características dos componentes do sistema a ser simulado, tais como representação gráfica, funções de transferência, etc.

2. Trabalhos relacionados

Como o trabalho envolve a geração de simuladores, com foco na simulação de circuitos sequenciais, é importante abordar tanto simuladores de circuitos disponíveis, como simulação de sistemas generalizados.

Assim, na área de projetos de circuitos existem vários simuladores, como por exemplo Modelsim [Modelsim 2016], Circuit maker [Circuit Maker 2016] e Qucs (Quite universal circuit simulator) [Qucs 2016]. Porém, todos estes simuladores apresentam a característica de já oferecerem um conjunto de objetos prontos, não permitindo uma especificação customizada para a simulação, como o Yasc busca oferecer.

Já na linha de simuladores mais gerais encontram-se abordagens com simulação baseada em *templates*, como o KanbanSim e o PowerTrainSim [Doss and Ülgen 2004], sendo que o primeiro simula rotas de entrega e o segundo é utilizado para simular partes específicas da manufatura e testes de componentes do conjunto motor-transmissão, porém sua generalização ocorre mais no uso de *templates* de modelos do que na simulação de aplicações distintas. Outra abordagem, que requer mais conhecimento, é o uso de linguagens específicas de simulação, como DEVS ou SystemC. Nenhuma dessas abordagens apresenta caracteristicas de generalidade e facilidade de uso propostas no Yasc.

3. Yasc e a simulação de circuitos sequenciais

O Yasc é uma ferramenta para gerar simuladores para diferentes contextos, que está sendo desenvolvida no Grupo de Sistemas Paralelos e Distribuidos. O objetivo de projeto do Yasc é que ele seja fácil de usar e gere simuladores fáceis de usar. A geração de simuladores é feita pela inserção pelo usuário, por meio de interfaces gráficas, de parâmetros dos sistemas que se quer simular. Tais parâmetros descrevem aspectos comportamentais do sistema e seus componentes. Isso permite ao usuário criar sua biblioteca de objetos, que posteriormente podem ser usados para modelar um sistema na interface icônica do simulador gerado.

Os principais componentes do Yasc são vistos na Figura 1, sendo:

- **GUI para descrição de elementos gráficos do simulador:** Na descrição de um elemento o usuário fornece sua representação gráfica, assim como um nome para o objeto;

- **GUI para descrição comportamental do simulador:** A descrição comportamental de um objeto permite descrever como o objeto irá se comportar, como por exemplo, se possuirá ou não filas, se eventos terão comportamento instantâneo ou durativo, dentre outras possibilidades.
- **Interpretador e biblioteca:** O interpretador gera uma biblioteca de simulação com os objetos definidos e que serão usados nos modelos a serem simulados;
- **Motor de simulação:** É gerado a partir de componentes de uma biblioteca pré-definida e dos objetos descritos pelo usuário, sendo responsável por executar as simulações de fato;
- **Interface icônica:** Também é gerada a partir de um *framework*, com as inserções dos objetos característicos do sistema a ser modelado e simulado.

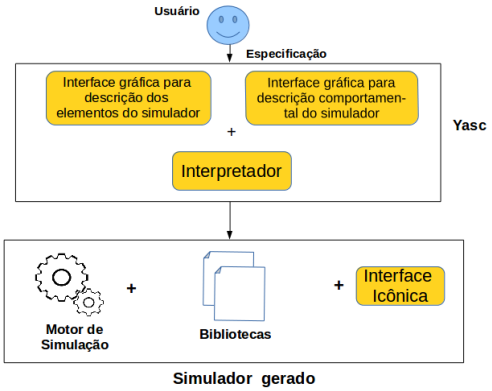


Figura 1. Arquitetura do Yasc [Furlanetto 2016]

3.1. Desenvolvimento

A geração de um simulador de circuitos sequenciais, ou qualquer sistema baseado em transição de estados, envolve a especificação de seus componentes e de funções que descrevam as transições de estado. A especificação de componentes é feita pela GUI de descrição de elementos, permitindo a inclusão de componentes como flip-flops, contadores, etc. Já a definição das transições de estado é o principal componente no Yasc, uma vez que a definição de funções de transferência não é simples.

A definição da função de transferência pode ocorrer de duas formas, fórmula ou tabela. Na descrição por fórmula, mostrada na Figura 2, o usuário especifica a equação da função de transferência, com a criação de variáveis e constantes para definir a função. No caso de circuitos sequenciais o usuário deve ainda habilitar o parâmetro operação lógica, para que ocorra a interpretação da função criada como sendo uma transição de estados.

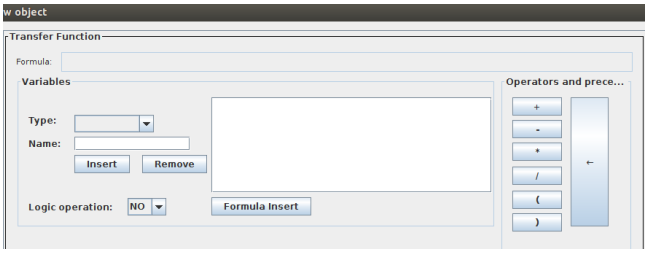


Figura 2. Interface para definir função de transferência através de fórmula.

Já para definir uma função de transferência por tabela, Figura 3, usam-se tabelas-verdade para descrever o comportamento de um dado objeto pela relação entre suas entradas e saídas.

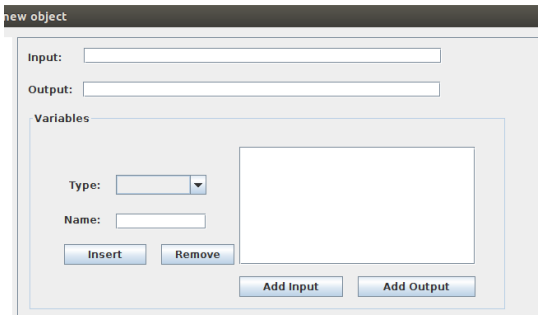


Figura 3. Interface para definir função de transferência através de tabela.

4. Resultados parciais

Em sua fase atual o Yasc já tem implementado os módulos para simulação de sistemas baseados em filas, devidamente testado por Furlanetto [Furlanetto 2016], incluindo a geração de simuladores de redes de computadores por alunos de graduação em computação. O desenvolvimento do trabalho até o momento permite concluir que é possível a criação de simuladores de circuitos sequenciais, assim como qualquer outro tipo de simulador baseado em transição de estados. No momento já se definiu as interfaces para especificação de objetos baseados em transição de estados, assim como componentes necessários para um motor de simulação para este sistema. Falta ainda implementar a biblioteca com os componentes para simular transições instantâneas de estado.

Na sequência deverão ser realizados testes sobre tais componentes, buscando primeiro verificar sua corretude em relação a outros simuladores de circuitos sequenciais, a partir de um simulador gerado pelo Yasc, e depois verificar sua usabilidade na geração de outros simuladores baseados em transição de estados.

Referências

Circuit Maker (2016). Circuit maker documentation. Disponível em http://documentation.circuitmaker.com/, acessado em 06 de maio de 2016.

Doss, D. L. and Ülgen, O. (2004). A case for generic, custom-designed simulation applications for material handling and manufacturing industries. In *Brooks Automation's Worldwide Automation Symposium*, pages 1–5.

Furlanetto, G. C. (2016). Geração de simuladores de filas para diferentes contextos. Dissertação de Mestrado, Universidade Estadual Paulista "Júlio de Mesquita Filho".

Modelsim (2016). Modelsim documentation. Disponível em https://www.altera.com/products/design-software/fpga-design/quartus-prime/overview.html, acessado em 25 de maio de 2016.

Qucs (2016). Qucs (quite circuit simulator). Disponível em http://sourceforge.net/index.html, acessado em 06 de maio de 2016.

Visualização de Padrões de Comunicação em Programas Paralelos através de Grafos: implementação do conversor entre os formatos de *trace* e de descrição de grafos

David Gabriel Oliveira, Denise Stringhini

Instituto de Ciência e Tecnologia – Universidade Federal de São Paulo (UNIFESP)
São José dos Campos – SP – Brasil

david10gabriel@hotmail.com, dstringhini@unifesp.br

Resumo. *Sistemas de processamento paralelo são amplamente utilizados. Tais sistemas são implementados com o objetivo de diminuir o tempo de processamento de grandes quantidades de dados e problemas complexos. Este projeto busca, em última instância, facilitar o entendimento de tais conceitos por meio de demonstrações visuais do funcionamento de sistemas paralelos, através da visualização da comunicação entre processos distribuídos. Esta visualização será realizada modelando-se a execução de determinados problemas através de grafos dinâmicos. Num primeiro momento, foi necessário implementar um conversor entre o formato de trace e o formato de descrição de grafos GEFX.*

1. Introdução

A principal motivação para a implantação de computação paralela é a necessidade de poder computacional não presente em sistemas sequenciais. O conceito de computação paralela nos sugere o princípio de que grandes problemas podem ser divididos e processados paralelamente, seja em sistemas completos operando sem memória compartilhada, ou em processadores multinúcleo (Wilkinson e Allen, 2004). Na teoria, um problema que levaria N minutos para ser processado em um processador *single core*, poderia ser resolvido em N/2 minutos em um processador *dual core* ou com dois processadores *single core* trabalhando concorrentemente.

Ao processarmos um grande problema de forma paralela em configurações com memória distribuída, constataremos que diversos processos do mesmo problema estarão em execução em diferentes núcleos de processamento, no entanto, problemas complexos e com diversos processos tendem a necessitar de comunicação entres suas inúmeras instâncias de processamento, e visto que um processo pode depender de um dado gerado na conclusão de outro processo, será essencial que tenhamos comunicação entre eles. A maneira mais utilizada atualmente para implementar a comunicação de forma explícita é a utilização da biblioteca MPI (*Message Passing Interface*) (Snir et al., 1998).

No estudo de processamento de alto desempenho é usual que se use ferramentas para análise de desempenho e depuração. Este tipo de *software* normalmente gera arquivos de *trace* com os registros de comunicação entre processos. Os arquivos de *trace* incluem registros como qual é o processo que enviou a mensagem, qual(ais) processo(s) recebeu(ram) a mensagem, quanto tempo durou esta comunicação, qual o tamanho da mensagem, além de quando foram criados e finalizados processos específicos. Notando-se a quantidade de dados contidos em um arquivo de *trace*, é de fácil conclusão o porquê desses arquivos serem tão importantes para análise de desempenho e, ao mesmo tempo, serem de

tão difícil análise. São exemplos de ferramentas que geram tais arquivos o EZtrace (Trahay et al, 2011) e o Pajé (Chassin de Kergommeaux e Stein, 2003).

De uma forma mais ampla, o projeto no qual este trabalho se insere propõe a criação de uma galeria de animações de execuções de programas paralelos reais nos moldes da apresentada por Davis e Hu (2011). Para isso, serão usadas ferramentas de geração de traces de execução, que posteriormente serão processados e convertidos em grafos dinâmicos. Estas visualizações serão, então, publicadas para análise de pesquisadores e estudantes da área. Neste contexto, o presente trabalho apresenta a primeira fase deste projeto, que é a conversão do arquivo de trace do formato Pajé para o formato de descrição de grafos GEFX (2016).

2. Visualização através de grafos de interação

Programas paralelos compostos de processos distribuídos podem ser modelados através de grafos de interação ou TIG (*Task Interaction Graph*) (Senar et al. 1998). Esta abordagem pode ser útil, por exemplo, na tarefa de mapear processos de acordo com uma determinada topologia do *hardware* paralelo, o que pode trazer ganhos de desempenho para as arquiteturas atuais (Bathele et al, 2014). Além disso, pode auxiliar a caracterizar os padrões de comunicação de um programa (Stringhini e Fazenda, 2015), o que pode ser útil para que se compreenda o programa paralelo de uma forma mais global, a partir da visualização e da análise das conexões entre os processos paralelos.

No contexto de um TIG, os nós do grafo são modelados a partir dos processos que compõem a aplicação paralela, assim como as arestas representam a comunicação entre eles. Estas informações são coletadas em tempo de execução e armazenadas em um arquivo de *trace*. Este trabalho utiliza a ferramenta EZTrace para coletar as informações sobre os nós e sobre a comunicação entres eles ocorrida durante a execução. Porém, o formato de arquivo gerado pelo EZTrace não é legível. Entretanto, a ferramenta oferece um conversor para o formato Pajé, este sim legível e bem documentado. Assim sendo, este foi o formato escolhido como formato de entrada para o conversor, que coleta as informações de *trace* para gerar o grafo.

O formato de saída do conversor é o formato GEFX, em XML - o que também o torna legível e flexível. Apesar de outros formatos mais simples existirem, o GEFX foi projetado para permitir a inclusão de informações sobre tempo de início e fim de eventos, o que o torna ideal para que posteriormente se possa utilizar a visualização de grafos dinâmicos. O GEFX é um formato aceito pela ferramenta de visualização de grafos e redes complexas Gephi (2016), que será usada para gerar as visualizações e análises.

3. Descrição da conversão e resultados obtidos

O arquivo inicialmente trabalhado, no formato trace, gerado pelo *software* Pajé, contém diversas informações de execução de processos, como criação e destruição de *containers* (neste caso, os processos que serão os nós do TIG) com seus devidos tempos de execução, início e fim de *links* (que serão as arestas do TIG) com identificação de origem e destino, além de tamanho em bytes da mensagem enviada, definição de eventos e seus tipos, e outros vários dados referentes às comunicações entre processos.

Para a criação da representação gráfica das comunicações entre processos são necessárias informações extraídas do arquivo em formato Pajé (Figura 1.a) tais como dados de criação e destruição de *containers* (nós), além de informações de início e fim de *links* (arestas) entre processos, que podem ser coletados fazendo com que o *software* de *parser* faça uma checagem no cabeçalho de definições do arquivo procurando por tais eventos.

Posteriormente, ao achar tais definições de eventos, o conversor coleta os números identificadores destes eventos. O próximo passo é uma extensa varredura no arquivo de *trace* juntamente com a coleta de todas as linhas identificadas pelos números de eventos de interesse inicialmente identificados, além da filtragem dos seguintes dados:

- Para criação de nós: número de identificação, rótulo de identificação e tempo de execução.
- Para criação de arestas: número de identificação, rótulo de identificação, tempo de execução, tamanho da mensagem, origem da mensagem e destino da mensagem.

Figura 1.a: Trechos do formato Pajé

```
%EventDef PajeCreateContainer 7
% Time date
% Alias string
% Type string
% Container string
% Name string
%EndEventDef
...
7 0.0000000000000e+00 "C_Prog" "CT_Program" 0 "Program"
7 0.0000000000000e+00 "P#0" "CT_Process" "C_Prog" "P#0"
7 0.0000000000000e+00 "P#0_T#572798720" "CT_Thread" "P#0" "P#0_T#572798720"
10 0.0000000000000e+00 "ST_Thread" "P#0_T#572798720" "STV_Working"
7 0.0000000000000e+00 "P#13" "CT_Process" "C_Prog" "P#13"
7 0.0000000000000e+00 "P#13_T#2146469632" "CT_Thread" "P#13"
"P#13_T#2146469632"
10 0.0000000000000e+00 "ST_Thread" "P#13_T#2146469632" "STV_Working"
...
42 1.6403584000000e+04 "L_MPI_P2P" "C_Prog" "P#10_T#424773376" "src=10,
dest=9, len=6160, tag=271a" "10_9_
43 1.6405524469000e+04 "L_MPI_P2P" "C_Prog" "P#6_T#251569920" "src=11,
dest=6, len=19040, tag=271b" "11_6_                10011_0x7f6a1834e538"
```

Figura 1.b: Trechos do formato GEFX

```
...
<nodes>
    <node id="11" label="Node 11">
       <attvalues/>
    </node>
...
    <node id="14" label="Node 14">
       <attvalues/>
    </node>
</nodes>
<edges>
    <edge id="8_15_" source="8" target="15">
       <attvalues/>
    </edge>
```

O formato de saída dos dados filtrados é o GEFX (Figura 1.b), ou *Graph Exchange XML Format*, que nada mais é que uma modificação do extensamente utilizado XML, neste caso usado para a descrição de estruturas de redes complexas. O arquivo de saída contém os dados coletados do arquivo de *trace* original, organizados para a geração do grafo, até então estático. O conversor aqui descrito foi implementado na linguagem de programação Python.

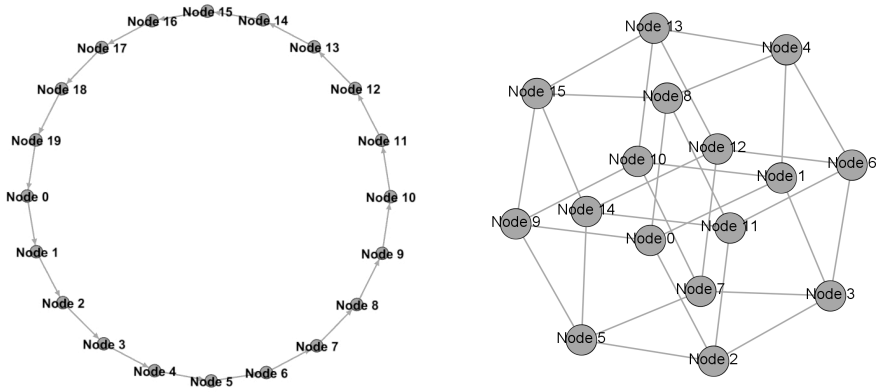


Figura 2: Grafos gerados pelo Gephi a partir do arquivo GEFX

A Figura 2 apresenta o resultado da visualização de dois grafos gerados com a ferramenta Gephi utilizando o algoritmo de desenho de grafos Yifan-Hu. O primeiro

implementa um algoritmo em anel com 20 nós e o segundo é o benchmark BT-MZ do pacote NAS (Bailey, 2011) com 16 nós.

Conclusão

Este trabalho apresentou o primeiro passo na construção de uma galeria de imagens dinâmicas que representarão programas paralelos na forma de TIG dinâmicos. A partir de um arquivo no formato de *trace* Pajé, o conversor aqui apresentado cria uma representação em XML no formato de descrição de grafos GEFX. O grafo gerado pode então ser carregado na ferramenta de visualização Gephi, que pode produzir análises de grafos estáticos e dinâmicos. Os trabalhos futuros incluem, assim, a coleta de informações dinâmicas e a posterior representação dinâmica dos programas paralelos.

Referências

Bailey, D.H.: (2011) Nas parallel benchmarks. In: Padua, D.A. (ed.) Encyclopedia of Parallel Computing. pp. 1254–1259. Springer (2011)

Bhatele, A and Jain, N and Isaacs, K E and Buch, R and Gamblin, T and Langer, S H and Kale, L V. (2014) "Optimizing the performance of parallel applications on a 5D torus via task mapping"

Chassin de Kergommeaux, J. and Stein, B. de Oliveira (2003). Flexible performance visualization of parallel and distributed applications. Future Generation Computer Systems 19 (2003) 735-747.

Davis , T.A. e Hu, Yifan. (2011) The University of Florida sparse matrix collection,, ACM Transactions on Mathematical Software, Vol 38, Issue 1, 2011, pp 1:1 - 1:25.

Gephi (2016) The Gephi website. http://www.gephi.org/. Acesso: maio de 2016.

GEXF (2016) The GEXF website. https://gephi.org/gexf/format/. Acesso:maio de 2016.

Senar , M.A., Ripoll A., Cortés A. e Luque E. (1998) "Clustering and Reassignment-Based Mapping Strategy for Message Passing". Conference: Parallel Processing Symposium,. IPPS/SPDP 1998. Proceedings of the First Merged International ... and Symposium on Parallel and Distributed Processing, 1998.

Snir, M., Otto, S., Huss-Lederman, S., Walker, D., Dongarra, J.(1998) MPI-The Complete Reference, Volume 1: The MPI Core. MIT Press, Cambridge, MA, USA, 2nd. (revised) edn. (1998)

Stringhini, D. e Fazenda A. (2015) "Characterizing Communication Patterns of Parallel Programs through Graph Visualization and Analysis". Euro-Par 2015: Parallel Processing Workshops V. 9523, Lecture Notes in Computer Science pp 565-576

Trahay, F., Ru, F., Faverge, M., Ishikawa, Y., Namyst, R., Dongarra, J. (2011). Eztrace: A generic framework for performance analysis. In: Proc. CCGRID. IEEE, NewportBeach, CA, USA (Maio 2011)

Winkinson, B. e Allen, M. (2004). Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers (2nd Edition) . Pearson Ed., 2004.

Visualização de Progressos de Otimização em Implementações para Alto Desempenho ¶

Vitor H. L. Mesquita[1,2], **Michael M. D. Heo**[1,3], **Bruno S. Silva**[1]

[1]IBM Research Brasil
Rua Tutóia, 1157 - São Paulo - SP - Brasil

[2]Graduando do Curso de Análise e Desenvolvimento na FATEC-SP
Praça Coronel Fernando Prestes, 30 - São Paulo - SP - Brasil

[3]Graduando do Curso de Ciência Da Computação na Universidade Presbiteriana Mackenzie
Rua da Consolação, 930 - Consolação, São Paulo - SP - Brasi

{vleal,mdong,sbruno}@br.ibm.com

Abstract. *High-Performance Computing (HPC) refers to the use of supercomputers to perform processing or evaluation of high complex systems. These evaluations involve weather forecast, process simulation, aerospatial projects among others. A major challenge in this area refers to the utilization of supercomputers to find a parameters configuration that optimizes a given process. This work presents an overview of the graphical interface of an optimization supporting tool for parametric sweeping applications using human expertise - WET.*

Resumo. *Computação de alto desempenho ou High Performance Computing (HPC) refere-se ao uso de supercomputadores para realizar cálculos ou avaliações de sistemas de alta complexidade. Esses cálculos envolvem previsão do tempo, simulações de processos, projetos aeroespaciais entre outros. Um dos desafios nessa área refere-se à utilização de supercomputadores para encontrar um conjunto de parâmetros que otimize um determinado processo. Este trabalho apresenta uma visão geral da interface gráfica da ferramenta para otimizacão de sistemas de varredura paramétrica utilizando a experiência do usuário - WET.*

1. Introdução

Um dos grandes desafios em Computação de Alto Desempenho *(High Performance Computing - HPC)* é a programação paralela e a utilização de suas APIs. A motivação para que novos desenvolvedores trabalhem com programação paralela tende a crescer com o tempo, já que os dispositivos dotados de processadores com mais de um núcleo estão se tornando cada vez mais comuns. Um cluster é um recurso utilizado no cenário da computação paralela, onde um sistema composto de dois ou mais computadores trabalhando de forma conjunta no intuito de processar uma tarefa. Os computadores (denominados "nós") dividem entre si as atividades de processamento, executando a tarefa de maneira simultânea. Uma tarefa (denominada "JOB"), tem como finalidade executar um conjunto de instruções onde seus parâmetros podem ser variados. Assim, utilizando-se

¶Este trabalho é apoiado e parcialmente custeado pela FINEP / MCTI, contrato n° 03.14.0062.00.

sistemas de computação de alto desempenho uma mesma aplicação pode ser executada repetidas vezes e de forma paralela para se encontrar a melhor combinação de parâmetros que otimiza um determinado sistema/processo.

Sistemas de varredura paramétrica ou *Parametric Sweep Applications* são utilizadas por cientistas e engenheiros para avaliação de modelos de simulação e otimização de sistemas. Nesses sistemas, o analista precisa executar várias vezes o mesmo programa e variar seus parâmetros para obter os resultados de interesse. Contudo, a busca exaustiva da melhor configuração de parâmetros para otimizar ou avaliar um determinado modelo/sistema é inviável. Portanto, heurísticas são geralmente utilizadas para se avaliar apenas uma parte do espaço de possíveis combinação de parâmetros.

Nos últimos anos, muitos métodos de otimização para esse tipo de problemas têm sido propostos (otimização combinatória). Contudo, soluções totalmente automáticas também apresentam alguns problemas como os listados a seguir: dificuldade de se obter ou criar um modelo de otimização que reflita todos os aspectos do sistema real, especialmente em otimizações de múltiplos objetivos; mesmo quando o modelo de otimização representa bem o problema em questão, muitas vezes o método aplicado pode demorar para encontrar uma solução e violar restrições de tempo e/ou custo; a experiência e criatividade de especialistas na área em questão são difíceis de se representar principalmente em problemas que envolvem decisões complexas e importantes (exemplo, sistemas de suporte para compra e venda de ações).

No intuito de resolver as questões acima, a ferramenta Workload Explorer Tool (WET) foi proposta para auxiliar analistas na busca de soluções otimizadas dentro de um espaço de amostras discretas. A ferramenta une a experiência e conhecimento humano e técnicas de otimização combinatória para encontrar tais soluções. Este trabalho apresenta uma visão geral da ferramenta WET e mais especificamente o modo de interação desta com o usuário.

2. Trabalhos Relacionados

Parker and Johnson [Parker and Johnson 1995] apresentaram um sistema chamado "SCI-Run", descrito como um modelo de programação de fluxo de dados e programação visual para simplificar tarefas de criação, depuração, otimização e controle de simulações científicas complexas. Meignan et.al [Meignan et al. 2015] exploraram os diferentes papéis que um usuário pode ter em um processo de otimização, como ajustar ou adicionar novas restrições e objetivos, ajudando e guiando o próprio processo de optimização, fornecendo informações relacionadas a variáveis de decisão. Nascimento and Eades [do Nascimento and Eades 2005] propuseram uma aplicação para auxiliar os usuários na assistência do processo de optimização através do conhecimento do domínio aplicado, reduzindo o espaço de busca a ser explorado, e evitando ambiguidade em múltiplas soluções. Abramson et al. [Abramson et al. 2011] desenvovleram um sistema para facilitar a exploração dos parâmetros e suporte para a abstração da plataforma de computação subjacente.

3. WET

A Figura 1 apresenta uma visão geral do modo de utilização da ferramenta. Inicialmente, o analista apresenta as restrições de custo, tempo ou qualidade do experimento (1). Em

seguida, o usuário dispara a execução de um conjunto de tarefas de avaliação (2). Posteriormente o Assistente de Busca executa as tarefas, prediz a violação de *deadlines* e sugere mudanças na estratégia de busca baseada na estratégia *Greedy Randomized Adaptive Search Procedure* (GRASP) [Feo and Resende 1995] (3). Em seguida, o usuário é notificado e o resultado das tarefas é apresentado (4). O analista verifica os resultados e modifica a estratégia de busca (se necessário) (5). Por fim, o usuário notifica o sistema sobre a mudança na estratégia (6).

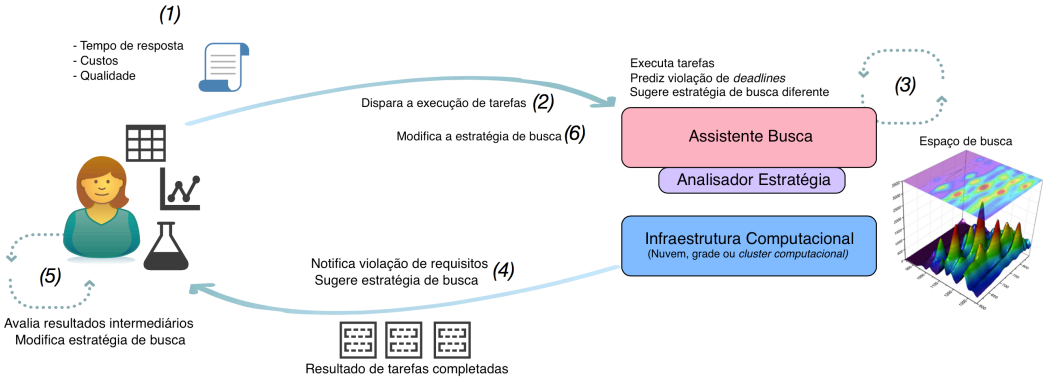


Figura 1. Visão de alto nível do fluxo de informação da ferramenta WET.

A Figura 2 apresenta um exemplo da interface gráfica da ferramenta WET. A interface gráfica da ferramenta WET é baseada em tecnologia WEB e o *framework* D3 é utilizado para geração dos gráficos para apresentação dos dados. Na figura 2 é possível visualizar uma das saídas de diferentes *jobs* que executam para uma aplicação de previsão de inundações *Integrated Flood Model* (IFM) [Singhal et al. 2014]. Nessa avaliação, os parâmetros EP, HC, LANDUSE e PH correspondem aos parâmetros do sistema de previsão de inundação. SCORE corresponde à saída que informa a semelhança entre o modelo predito e a saída do sistema previamente medido. É possível visualizar que parâmetros com os valores em azul apresentam os melhores resultados. Dessa forma, analistas podem utilizar os parâmetros avaliados para configurar o sistema e realizar futuras previsões.

As outras interfaces da ferramenta WET estão relacionadas à configuração de parâmetros e ao controle dos jobs que estão sendo executados no ambiente computacional. É importante mencionar que a ferramenta WET está ainda sendo construída e será disponibilizada como um serviço na plataforma IBM Bluemix.

4. Conclusão

Diversas áreas em ciência e engenharia precisam de várias execuções da mesma aplicação com diferentes parâmetros. Essas execuções são responsáveis pela avaliação de processos complexos e calibração de modelos físicos. A execução de todos os valores possíveis para todos os parâmetros suportados nem sempre é possível dadas restrições de tempo, custo ou energia. Assim ferramentas de apoio a avaliação de sistemas com múltiplos parâmetros é de extrema importância em ambientes de HPC.

Esse trabalho introduz a ferramenta WET que auxilia usuários na execução de seus sistemas, permitindo que o usuário utilize sua experiência para selecionar os parâmetros

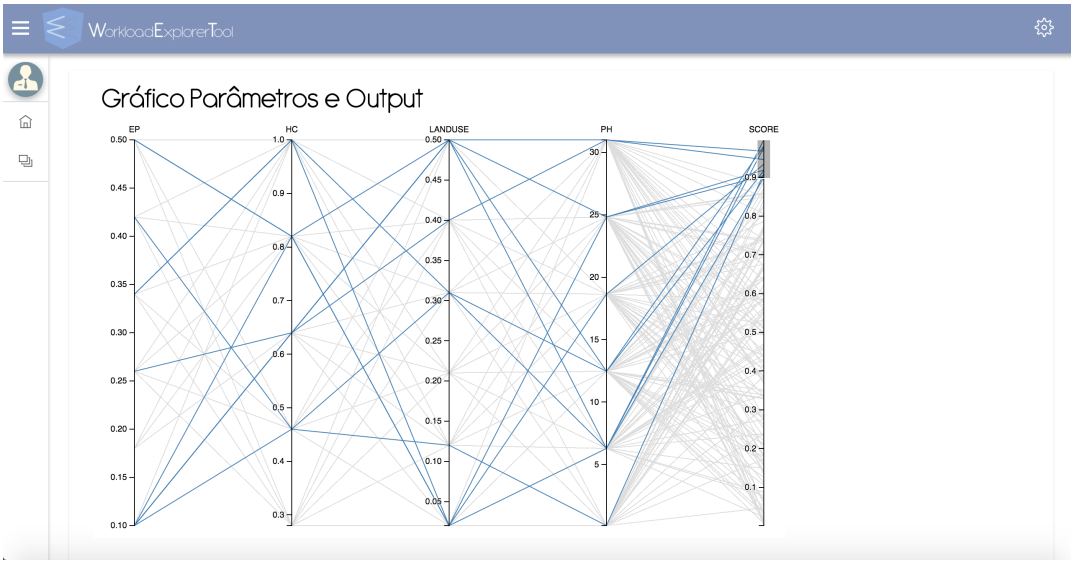


Figura 2. Exemplo de interface gráfica WET.

que melhor se adaptem ao problema em questão. Mais especificamente, este documento apresenta o que vem sendo realizado em relação à interface gráfica da ferramenta WET. Com a utilização da ferramenta WET, usuários podem descobrir de uma maneira simplificada quais os parâmetros mais importantes do sistema e utilizar a experiência do usuário para diminuir o tempo para avaliação do sistema.

Referências

Abramson, D., Bethwaite, B., Enticott, C., Garic, S., and Peachey, T. (2011). Parameter exploration in science and engineering using many-task computing. *Parallel and Distributed Systems, IEEE Transactions on*, 22(6):960–973.

do Nascimento, H. A. and Eades, P. (2005). User hints: a framework for interactive optimization. *Future Generation Computer Systems*, 21(7):1177–1191.

Feo, T. A. and Resende, M. G. (1995). Greedy randomized adaptive search procedures. *Journal of global optimization*, 6(2):109–133.

Meignan, D., Knust, S., Frayret, J.-M., Pesant, G., and Gaud, N. (2015). A review and taxonomy of interactive optimization methods in operations research. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 5(3):17.

Parker, S. G. and Johnson, C. R. (1995). Scirun: a scientific programming environment for computational steering. In *Proceedings of the 1995 ACM/IEEE conference on Supercomputing*, page 52. ACM.

Singhal, S., Aneja, S., Liu, F., Real, L. V., and George, T. (2014). IFM: A scalable high resolution flood modeling framework. In *Proceedings of the International European Conference on Parallel and Distributed Computing (Euro-Par)*. Springer.

Symmetries and the design of supercomputer network topologies

Marcus Campos Silva[1], **Paula Araújo Toyota**[1], **Yuefan Deng**[3,4,5], **Alexandre F. Ramos**[1,2,3]

[1]EACH; NISC – Universidade de São Paulo, São Paulo, Brazil;

[2]InRad, FM; ICESP – Universidade de São Paulo, São Paulo, Brazil

[3]SCSC – National Supercomputer Centre in Jinan, Jinan, China.

[4]AMS Dept – Stony Brook University, New York, USA

[5]Sun Yat-Sen University, Guangzhou, China.

{alex.ramos,marcus.campos.silva,paula.toyota}@usp.br,yuefan.deng@me.com

Abstract. *An application of Lie symmetries for description of the hypercube graph is presented. The usefulness of Cartan classification of the Lie algebras for design of supercomputer network topologies is shown in terms of a graph based on the four dimensional symplectic algebras. We show that this graph enables the construction of network topologies with small average path length and higher numbers of vertices.*

1. Introduction

Applications keep demanding faster computations despite the striking petaflops processing speeds of the fastest supercomputers [1] and exascale remains as a central goal of supercomputing [2, 3]. To reach such a goal one shall need to propose interconnect network topologies capable of optimally combine millions of supercomputers' components such as processing nodes, switchers, routers [4]. Graph theory have been widely used for such a task [5, 6] by means of heavy numerical computations and analytic tools would be welcome. To fulfill such a lacuna group theoretical methods have been recently proposed for the design of supercomputer network topologies [7]. Here we explore that approach by describing well known hypercubic topology by means of usual concepts of Lie algebras. We show that the roots and weights lattice of the fundamental representation of the direct sum of r samples of the $\mathfrak{su}(2)$ algebra corresponds to the r-dimensional hypercube graph [8]. A correspondence rule between the vertices of the hypercube and the weight vectors of an irreducible representation of the algebra is presented while the adjacency matrix is given in terms of the linear combination of the ladder operators of the algebra. That result suggests a systematic procedure for the design of supercomputer network topologies based on the Cartan classification of all Lie algebras and open a new research area on the field [7]. Indeed, we have constructed a new graph optimized for a supercomputer network topology based on the symplectic algebras of four dimensions an analyzed it in comparison with the hypercube and the mesh topologies. The group theoretical approach for designing supercomputer network topologies can be resumed at Table 1 where we establish the relationship among the terminology from computer sciences, graph theory and group theory.

Computer Science	Graph theory	Group theory
Supercomputer	Graph	Roots and Weights lattices
Computing nodes; switch; router	Graph vertices	Weight vectors
Network topology	Graph edges	Root vectors

Table 1. Corresponding terms and concepts in the three disciplines.

2. Methods

We consider the r-dimensional hypercube graph, $\mathcal{G}_h(r)$, of $\nu = 2^r$ vertices labeled by an integer k, with $k = 0, \ldots, \nu - 1$. The adjacency square matrix of $\mathcal{G}_h(r)$ has size $2^r \times 2^r$, and is denoted by ϵ_r. It is written explicitly in terms of a recursion relation starting with the one by one adjacency matrix $\epsilon_0 = (0)$. The matrix ϵ_{r+1} is a $2^{r+1} \times 2^{r+1}$ matrix decomposable as a 2×2 block matrix, with each block being a matrix of size $2^r \times 2^r$. The diagonal (or anti-diagonal) blocks are given by ϵ_r (or 1_r), for $r = 0, 1, \ldots$ and 1_r indicating the $2^r \times 2^r$ identity matrix.

Let us consider the $\mathfrak{su}(2)$ Lie algebra of rank one. The rank of a Lie algebra determines the dimension of the vector space of spanning its root vectors (roots) and weight vectors (weights). The roots of the $\mathfrak{su}(2)$ are $h_0 = 0$ and $\alpha_\pm = \pm 1$ and the weights of the fundamental representation are $w_0 = -1/2$ and $w_1 = 1/2$. The action of the generators of the $\mathfrak{su}(2)$ algebra onto the weights correspond to the vector addition of the weights and the roots. The result of this operation may result another weight of the representation, $w_0 + \alpha_+ = w_1$ and $w_1 + \alpha_- = w_0$, or zero if the new vector is not a weight, *e.g.* $w_0 + \alpha_- = 0$ and $w_1 + \alpha_+ = 0$. Therefore, one may say that the action of the ladder operators of the algebra connects different weights of the representation.

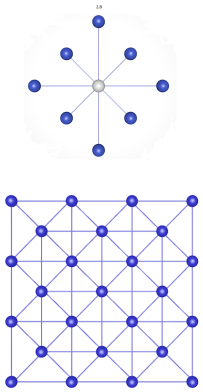


Figure 1. The roots diagram (top) and the roots and weights lattice (bottom) for the $\mathfrak{sp}(4, \mathbb{C})$ algebra for $M = 3$.

That approach is extended to higher dimensional irreducible representations of an algebra or to higher rank algebras, such as the direct sum of r samples of the $\mathfrak{su}(2)$ algebra, namely, $\oplus_{i=1}^{r} \mathfrak{su}(2)$, or to the $\mathfrak{sp}(4, \mathbb{C})$ algebra. The roots of the $\mathfrak{su}(2)$ algebras are $\alpha_{\pm 1} = (\pm 1, 0, \ldots, 0)$, $\alpha_{\pm 2} = (0, \pm 1, \ldots, 0)$, $\ldots$, $\alpha_{\pm r} = (0, 0, \ldots, \pm 1)$ and the weights shall have coordinates on the form

$$(m_1, \ldots, m_r), \text{ with } m_i = \pm 1/2 \text{ for all } i. \quad (1)$$

The $\mathfrak{sp}(4, \mathbb{C})$ algebra shall have roots given by $(\pm 2, 0)$, $(0, \pm 2)$, $(\pm 1, \pm 1)$ and the weights for a given irreducible representation shall be (m_1, m_2) with $m_i = -M, -M + 2, \ldots, M$ or $m_i = -M + 1, -M + 3, \ldots, M - 1$ for $i = 1, 2$ and M a positive integer. The Figure 1 shows the roots and the roots and weights lattice of the $\mathfrak{sp}(4, \mathbb{C})$ algebra.

3. Results

The r-dimensional hypercube is described in terms of the roots and weights lattice of the fundamental representation of the $\oplus_{i=1}^{r} \mathfrak{su}(2)$ algebra and, for simplicity, we shall denote

this lattice as hypercubic graph. The correspondence rule between the vertex addresses of the hypercube and the weights of roots and weights lattice is given as follows. We redefine the weights coordinates on Eq. (1) as $w_k = (\lambda_1, \ldots, \lambda_r)$, where $\lambda_i = m_i + 1/2$ such that the integer k_w is:

$$k_w = \sum_{i=1}^{r} \lambda_i 2^{i-1}, \quad \text{with} \quad k_w = 0, \ldots, 2^r - 1. \tag{2}$$

The inverse of this transformation is a recursion relation. For a given integer k_w labeling a vertex of the hypercube one obtains the address of its corresponding node, $w_k = (\lambda_1, \ldots, \lambda_r)$, on the r-dimensional lattice as follows: $\rho_r = k_w, \;\; \lambda_i = \lfloor \rho_i / \mu^{i-1} \rfloor, \;\; \rho_{i-1} = \mathrm{mod}(\rho_i, \mu^{i-1}), \;\; i = r, \ldots, 2, \;\; \lambda_1 = \rho_1$. The adjacency matrix of the roots and weights lattice is obtained from the ladder operators of the algebra $\mathfrak{g}$ as follows: $A = \mathbb{L}_+ + \mathbb{L}_-$, where $\mathbb{L}_\pm = \sum_{k=1}^{r} \otimes_{j=1}^{r} [(1 - \delta_{kj})\mathbb{I}_2 + \delta_{kj} E_\pm]$.

One may also find a correspondence between the roots and weights lattice of an irreducible representation of the $\mathfrak{sp}(4, \mathbb{C})$ algebra and the corresponding graph that we shall denote as symplectic graph. We start redefining the weights coordinates such that they become $w_k = (\lambda_1, \lambda_2)$ with $\lambda_i = 0, .., 2M$ or $\lambda_i = 1, \ldots, 2M - 1$. Now the weights k_w are labeled by the odd numbers ranging from 1 to $(2M + 1)^2$ and the inverse transformation is given as $\lambda_2 = \lfloor (k_w - 1)/(2M + 1) \rfloor, \;\; \lambda_1 = \mathrm{mod}(k_w, 2M + 1)$. The correspondence between the weights of the roots and weights lattice and vertices of a graph enables one to label the ij-th entry of the adjacency matrix by means of weights $w_p w_q$ with $p = 2i - 1$ and $q = 2j - 1$ and $i, j = 1, \ldots, (M + 1)^2 + M^2$. The adjacency matrix has non-null entries on the rows and columns labeled by weights whose difference $w_p - w_q$ is given in terms of a single root otherwise the entry shall be null.

The symmetry-based approach also enables one to compute the distance between two weights of the roots and weights lattice. The distance between the weights w_i and w_j is given by the smallest number of roots connecting them. This number can be obtained by the vector sum of the two weights. For the hypercubic graph the distance between two weights $w_i = (\lambda_1, \ldots, \lambda_r)$ and $w_j = (\lambda'_1, \ldots, \lambda'_r)$ is $\sum_{i=1}^{r} |\lambda_i - \lambda'_i|$. For the symplectic graph the distance between two weights $w_i = (\lambda_1, \lambda_2)$ and $w_j = (\lambda'_1, \lambda'_2)$ is $(|\lambda_1 - \lambda'_1| + |\lambda_2 - \lambda'_2|)/2$.

The Figure (1) shows that the symplectic graph corresponds to the superposition of two 2D meshes of sizes $M + 1$ and M interconnected by oblique edges. Therefore, we compare the properties of the symplectic graph with the hypercube and the mesh. The number of vertices on the hypercube, the mesh and symplectic graphs are, respectively,

$$2^r, \;\; \mu^r, \;\; (M + 1)^2 + M^2. \tag{3}$$

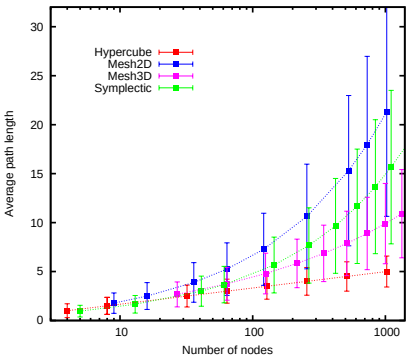


Figure 2. The average path length of four different topologies: r dimensional hypercube $H(r)$; the mesh on two and three dimensions with μ vertices per direction; the $\mathfrak{sp}(4, \mathbb{C})$ graph for different values of M.

On the same order, the graphs diameters are r, $r\,(\mu - 1)$, and $2\,M$, while the maximal (and minimal) vertex degrees are r (and r), $2r$ (and r) and 8 (and 3).

The Figure (2) presents the average path length for the r dimensional hypercube, the 2 and 3 dimensional mesh with μ vertices per direction and the $\mathfrak{sp}(4,\mathbb{C})$ graph defined in terms of different values for M, respectively, indicated by the colors red, blue, pink, and green. The dashed lines are auxiliary interpolations and the actual number of vertices of the graph varies accordingly with the formulas of the Eq. (3). The vertical axis indicates the average distance on the graph while its number of vertices is indicated on the horizontal axis. The solid squares are denoting the average path length of a given graph and the vertical bars denote the standard deviation of the path length distribution. For graphs having up to 100 vertices the symplectic topology has an average path length comparable to the mesh 3D even though it lies on a 2D space. For bigger graphs the average path length of the symplectic graph evaluated here stands between the mesh 2D and the mesh 3D. Notice that for small amounts of vertices, up to 30, the symplectic graph has a path length comparable to the hypercube.

4. Conclusions

The correspondence between the hypercube graph and the roots and weights lattice of the fundamental representation of $\oplus_{i=1}^{r}\mathfrak{su}(2)$ opens new research possibilities. The vertices of the graph are described in terms of the weights of the irreducible representation of the algebra while the adjacency matrix is constructed in terms of the ladder operators. As an example of application of the symmetries we consider the graph constructed with the roots and weights lattices of the $\mathfrak{sp}(4,\mathbb{C})$ algebra and show that it may generate graphs with reduced average path length and greater number of vertices. Further design of supercomputer network topologies can be developed within this framework and shall be explored in the future.

References

[1] www.top500.org.

[2] Peter Kogge. The tops in flops. *IEEE Spectrum*, 48(2):48–54, 2011.

[3] S.S. Pawlowski. Exascale science: the next frontier in high performance computing. In *The 24th International Conference on Supercomputing 2010*, 2010.

[4] D. B. Garzon, C. Gomez, M. E. Gomez, P. Lopez, J. Duato. Towards an efficient fat–tree like topology in Proc. 18th Int. Conf. Euro-Par Parallel Process., 2012, pp. 716?728.

[5] J. Duato, S. Yalamanchili, L. Ni. Interconnection Networks. San Mateo, CA, USA: Morgan Kaufmann, 2002.

[6] W. J. Dally and B. P. Towles. Principles and Practices of Interconnection Networks. San Mateo, CA, USA: Morgan Kaufmann, 2003.

[7] Yuefan Deng, Alexandre F Ramos, and José Eduardo M. Hornos. Symmetry insights for design of supercomputer network topologies: roots and weights lattices. *Int J Mod Phys B*, 26:1250169, 2012.

[8] F. Harary, J. P. Hayes, and H.-J. Wu. A survey of the theory of hypercube graphs Comput. Math. Appl., vol. 15, pp. 277?289, 1988.

Armazenamento de Arquivos Multinuvem com Suporte de Mecanismos de Auditoria – Um Estudo de Caso Aplicado à Ferramenta *FlexSky*

Elisa J. Marcatto[1], Rafael M. O. Libardi[1], Júlio C. Estrella[1]

[1]Instituto de Ciências Matemáticas e de Computação – Universidade de São Paulo
São Carlos, SP – Brasil

elisa.marcatto@usp.br, jcezar@icmc.usp.br

Abstract. *This paper involves the study of Cloud Auditing solutions that are suitable for multi-cloud storage environments. For this, several implementation techniques were assessed with the aim of creating a module which will later be integrated into the multi-cloud storage project FlexSky developed in LaSDPC - Laboratory of Distributed Systems and Concurrent Programming.*

Resumo. *Este trabalho envolve o estudo de técnicas de Auditoria em Nuvem que sejam adequadas a ambientes de armazenamento multinuvem. Para isso, foram avaliados diversos métodos de implementação com o objetivo de se criar um módulo que, posteriormente, será integrado ao projeto de armazenamento multinuvem FlexSky, desenvolvido no LaSDPC - Laboratório de Sistemas Distribuídos e Programação Concorrente.*

1. Introdução

Este projeto teve por objetivo estudar e modelar um mecanismo de auditoria pública aplicada no armazenamento de arquivos em multinuvem e deverá ser acoplado como um módulo ao projeto *FlexSky* [Libardi,2014], desenvolvido no laboratório de pesquisa LaSDPC (Laboratório de Sistemas Distribuídos e Programação Concorrente). A ferramenta *FlexSky* propõe o desenvolvimento e a formalização de um novo modelo de armazenamento em nuvem, consistindo em uma aplicação capaz de armazenar arquivos de maneira dispersa através do uso de vários provedores de armazenamento simultaneamente. A *FlexSky* cria uma abstração entre diversos servidores de armazenamento público, sem infraestrutura adicional, e possibilita ao usuário comum um método de armazenamento distribuído com maior redundância e maior segurança se comparada a abordagens atuais de armazenamento.

A incorporação deste projeto de auditoria à *FlexSky* contribuirá para que esta ferramenta agregue diferenciais atrelados à segurança, à privacidade e à integridade dos dados armazenados. O acoplamento de um módulo de auditoria consiste na implementação de funcionalidades que investiguem e monitorem as atividades de usuários e de provedores de serviços em nuvem, atuando como uma entidade independente e confiável. As inspeções realizadas resultam em análises detalhadas do tipo de falha ocorrido na segurança da aplicação, sendo possível definir a medida de correção mais adequada a se tomar. A Figura 1 evidencia a função do módulo de auditoria na ferramenta *FlexSky*, a qual é de inspecionar o processo de armazenamento dos dados dispersos na nuvem.

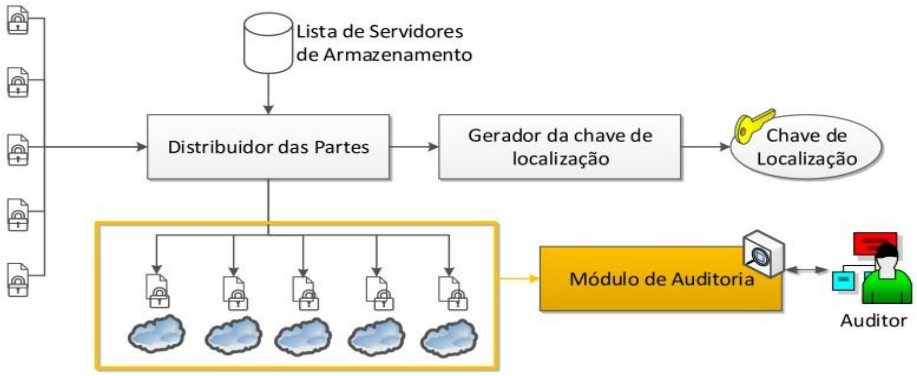


Figura 1. *FlexSky* – Módulo de auditoria integrado ao protótipo

2. Métodos

Neste projeto, foi utilizado o modelo de Auditoria Multipartes baseada em SLA (*Service Level Agreement*), na qual ocorre a inspeção de todas as partes do ambiente de nuvem - cliente, contratante e provedor - através de uma entidade independente e confiável denominada TPA (*Third Party Auditor*). O TPA é composto por inspetores e auditores (agentes de auditoria) e tem a função de coordená-los na auditoria da nuvem. Um outro importante ator na auditoria é a camada de governança do provedor, a qual possui conhecimento sobre o usuário e é capaz de interpretar informações coletadas por inspetores e por auditores.

No processo de auditoria, compara-se os parâmetros coletados nas partes (SLIs - *Service Level Indicators*) com os de referência (SLOs - *Service Level Objetive*) através da ação dos agentes de auditoria, sendo responsabilidade da camada de governança analisar os resultados obtidos por eles. Os SLOs podem se referenciar a intervalos de tempo calculados ao acessar a nuvem ou apenas assumir um valor que represente se houve algum acesso indevido aos arquivos armazenados (*flag* de acesso). A Figura 2 representa o fluxo de informações que ocorre entre os agentes de auditoria e a camada de governança. Diante destas características, o módulo de auditoria verifica, periodicamente, as atividades de armazenamento do usuário e do provedor da nuvem, informando as partes caso algum desvio nos SLIs seja identificado.

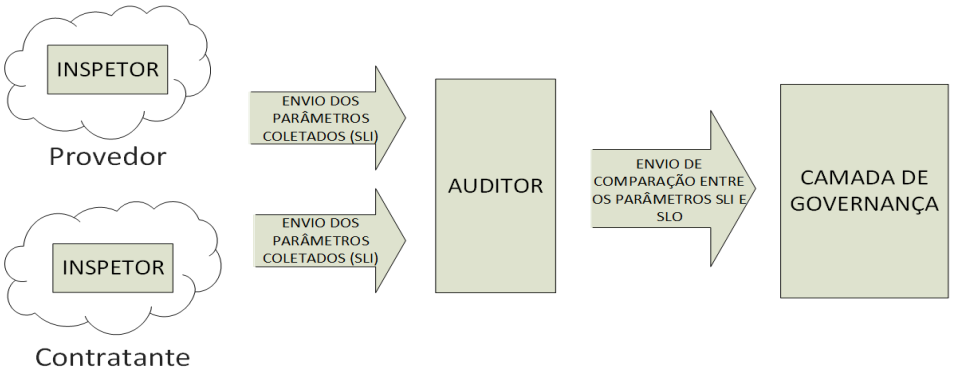


Figura 2. Parâmetros entre agentes de auditoria e camada de governança

Para que fosse analisada a eficiência do mecanismo de auditoria desenvolvido, foram elaborados diversos cenários de teste, para isso foi considerado um ambiente de nuvem pública, a existência de um TPA confiável e de um SLA bem definido entre as partes da nuvem, de modo que os parâmetros analisados respeitem a característica de

serviços mensuráveis estabelecida pelo NIST (*National Institute of Standards and Technology*) no modelo de computação em nuvem. A modelagem desenvolvida neste trabalho envolve tanto a infraestrutura da nuvem, quanto a entidade TPA, tendo o objetivo de simular o uso real de uma ferramenta de armazenamento e as possíveis falhas que podem ocorrer neste processo. Para isso, foram manipulados arquivos no formato *.txt* e diretórios locais para representar, respectivamente, os dados armazenados pelo usuário e a infraestrutura da nuvem. A Tabela 1 exibe algumas das situações analisadas durante os experimentos.

Tabela 1. Descrição de alguns cenários analisados

Tipo de Cenário	Descrição	Implementação
Perda de integridade (I)	O arquivo é violado por algum acesso desconhecido e seu conteúdo é alterado	Sem interação do usuário com a ferramenta, será realizado um acesso no arquivo do diretório que representa a nuvem, sendo modificado o arquivo armazenado
Corrompimento do arquivo armazenado (II)	O arquivo tem seu conteúdo modificado	Sem interação do usuário com a ferramenta, o arquivo do diretório que representa a nuvem terá seu conteúdo modificado
Quebra de privacidade do arquivo (III)	O arquivo é acessado por algo, ou alguém, que não o usuário proprietário	Sem interação do usuário com a ferramenta, o arquivo do diretório que representa a nuvem sofrerá um acesso desconhecido
Falha na infraestrutura do provedor (IV)	Operação do provedor sobre sua infraestrutura impacta no armazenamento do arquivo	O método da classe que modela a ação do provedor terá seu tempo de execução prolongado através da inserção de uma rotina de atraso

3. Resultados e Discussão

Na realização dos experimentos, inicialmente, nenhum dos cenários foi aplicado na execução do modelo, pois era necessário determinar os valores que seriam atribuídos aos SLOs. No caso de SLOs referentes a *flags* de acesso, estabeleceu-se que o valor um representaria uma situação sem alterações, assim, se obtido o valor zero no SLI relacionado, há indicação de um acesso indevido. A partir dos valores SLOs definidos, foi realizada uma série de experimentos baseados nos cenários elaborados, sendo possível identificar, ao final da auditoria, se os desvios evidenciados nos SLIs eram devidos a interferências externas à nuvem ou a falhas associadas ao desempenho da mesma, podendo estar relacionado tanto com o contratante, quanto com o provedor. A Tabela 2 evidencia a análise produzida pela camada de governança após a auditoria das partes para os cenários mencionados na Tabela 1:

Tabela 2. Resultados dos testes para os cenários propostos

Cenário	Valor de SLO estabelecido	Valor de SLI obtido	Análise produzida pela Camada de Governança
(I)	1 (*flag* de acesso)	0 (*flag* de acesso)	Problema com os parâmetros de integridade do arquivo por apresentar conteúdo modificado e acesso indevido
(II)	1 (*flag* de acesso)	0 (*flag* de acesso)	Problema com a integridade do arquivo por conteúdo modificado
(II)	1 (*flag* de acesso)	0 (*flag* de acesso)	Problema com a integridade do arquivo por acesso indevido

(IV)	0 ms (tempo de acesso)	1741 ms (tempo de acesso)	Problema com a infraestrutura do provedor

Os resultados obtidos revelam que os agentes de auditoria são capazes de identificar ocorrências de violação de SLA através da formulação de desvios entre SLIs e SLOs. Já a camada de governança conseguiu destrinchar as interferências externas e internas à nuvem, responsabilizando os elementos que provocaram tais alterações. Dessa forma, é possível observar a viabilidade de se auditar uma infraestrutura de nuvem seja qual for o cenário de degradações. Como o usuário possui um contato direto com a infraestrutura do contratante, mas não com a do provedor, é muito mais custoso para aquele administrar novos armazenamentos do cliente em sua nuvem do que esgotar seus recursos em relação ao contrato estabelecido com o provedor. Assim, era esperado que os valores de SLI coletados na nuvem do contratante fossem maiores do que os do provedor, o que foi possível observar com a conclusão dos experimentos.

4. Conclusão

A elaboração de um mecanismo de Auditoria Multipartes baseada em SLA (*Service Level Agreement*) possibilita a identificação de degradações de nível serviço em um modelo de armazenamento em nuvem e, a partir delas, permite identificar o responsável por estes desvios. Analisando o projeto desenvolvido, pode-se verificar que o método escolhido para se implementar a auditoria é o mais adequado para a ferramenta *FlexSky*, já que permite preservar os princípios de preservação da privacidade e de armazenamento seguro dos dados através de um processo confiável a todas as partes. Dessa forma, este trabalho contribui para a agregação de diferenciais à *FlexSky*, intensificando questões de segurança e de privacidade de arquivos armazenados em nuvem. Assim, é possível oferecer ao usuário uma maneira de garantir a segurança de suas informações armazenadas em um ambiente multinuvem e contribuir para a consolidação do modelo de nuvem no cenário atual.

5. Agradecimentos

Agradeço à FAPESP pelo apoio financeiro (Processo: 2014/12611-7) e ao LaSDPC pela infraestrutura oferecida durante o projeto.

6. Referências

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. (2010). A view of cloud computing. Commun. ACM, 53(4):50–58.

Bessani, A., Correia, M., Quaresma, B., Andre, F., and Sousa, P. (2011). Depsky: dependable and secure storage in a cloud-of-clouds. In Proceedings of the sixth conference on Computer systems, EuroSys ’11, pages 31–46, New York, NY, USA. ACM.

Gens, F. (2008). It cloud services user survey, pt.2: Top benefits and challenges. (Acessado em 14 de março de 2013).

Libardi, R.M.O, Bedo, M.V.N., Reiff Marganiec, S., Estrella, J.C., "MSSF: A Step towards User-Friendly Multi-cloud Data Dispersal," in Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on, vol., no., pp.952-953 – doi: 10.1109/CLOUD.2014.138

Análise de Simuladores de Sistemas para Workflows Científicos

Leonardo Alves Miguel [1], **Henrique Yoshikazu Shishido** [1,2], **Júlio Cezar Estrella** [1]

[1]Departamento de Sistemas de Informação (SSC)
Instituto de Ciências Matemáticas e de Computação (ICMC)
Universidade de São Paulo (USP)
Avenida Trabalhador São-carlense, 400 – Centro – São Carlos - SP – Brasil

[2]Departamento de Computação
Universidade Tecnológica Federal do Paraná (UTFPR)
Av. Alberto Carazzai, 1640 – 863000-000 – Cornélio Procópio – PR – Brasil

{leonardoam@usp.br, shishido@utfpr.edu.br, jcezar@icmc.usp.br}

Abstract. *This paper analyses the architectures of the currently most used workflow simulators, WorkflowSim and DynamicCloudSim, exploring specific characteristics of each scheduling system, seeking to support future development of new features aiming to explore further Cloud Security simulations.*

Resumo. *Este artigo analisa as arquiteturas dos principais simuladores de Workflows Científicos utilizados atualmente, WorkflowSim e DynamicCloudSim, visando explorar as características específicas dos seus respectivos sistemas de escalonamento, com forma de suporte a futuras propostas de adição de novas funcionalidades visando ampliar o suporte a simulações de Segurança em Nuvens.*

1. Introdução

Um cenário comum em e-Science é o ciclo repetitivo de movimentação de dados de um *data-center* a um supercomputador para execução de experimentos e armazenamento de resultados. Em diversas pesquisas, esse processo repetitivo torna-se inviável e de difícil reprodução. Nesse cenário, o conceito de *workflow* viabiliza a execução de uma sequência de tarefas necessárias para a realização de um experimento científico.

A execução de workflows em arquiteturas distribuídas depende do escalonamento das atividades interdependentes do workflow aos recursos computacionais disponíveis. O escalonamento é o processo para determinar em qual momento e recurso uma determinada atividade deve ser executada. Uma abordagem de escalonamento procura otimizar objetivos específicos como, por exemplo, reduzir o tempo de execução de um workflow baseado em um limite de orçamento. Para desenvolver novas técnicas de escalonamento, simuladores de sistemas de gerenciamento de workflows são essenciais para testar a eficácia de novos algoritmos de escalonamento. No entanto, é preciso compreender as características de cada simulador para escolher e adaptá-lo aos diversos cenários existentes de sistemas distribuídos.

Nesse sentido, o objetivo deste trabalho é apresentar uma análise de simuladores de sistemas para gerenciamento de workflows comumente utilizados para testar algoritmos de escalonamento. A contribuição deste trabalho é oferecer uma visão geral das características de cada simulador, permitindo a produção de novas abordagens de escalonamento em ambientes distribuídos.

2. CloudSim

O CloudSim [Calheiros et al. 2009, Calheiros et al. 2011] é um simulador para modelar e simular ambientes de Computação em Nuvem, suportando a criação de Máquinas Virtuais (MVs) em um nó simulado de um *Data Center*, tarefas e seus mapeamentos às MVs adequadas. A Figura 1 ilustra algumas características disponíveis para personalização de cenários na simulação da execução de tarefas. Esse conjunto de componentes fornece desde opções de alto nível, como características das tarefas sendo executadas, até informações sobre as capacidades de hardware, incluindo informações sobre o armazenamento.

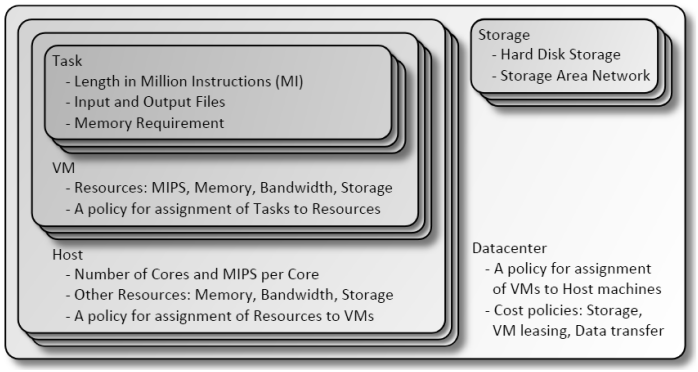


Figura 1. Algumas variáveis disponibilizadas pelo CloudSim para simulação de diferentes cenários

Este conjunto de ferramentas suporta a modelagem e simulação de ambientes de computação em nuvem tanto de redes individuais quanto interconectadas. Além disso, ele expõe interfaces personalizadas para implementação de políticas e técnicas de provisionamento de recursos. Embora o CloudSim seja usado em diversas pesquisas, este possui suas limitações para simular determinados eventos na nuvem. Assim, novas extensões do CloudSim tem sido propostas com o intuito de aprimorar as características de cenários não cobertos pelo CloudSim.

3. WorkflowSim

O simulador WorkflowSim [Chen and Deelman 2012] é uma extensão do CloudSim. O CloudSim possui um modelo de execução que não considera dependências de tarefas, nem o agrupamento das mesmas, suportando apenas a execução de tarefas simples, e ignorando a existência de falhas e *overheads*. Frente a este cenário, o WorkflowSim adiciona múltiplas camadas à camada de escalonamento do CloudSim para cobrir estes problemas.

O WorkflowSim auxilia não só na avaliação de técnicas de escalonamento, mas também considera diversos fatores como falhas e *overheads* na execução e escalonamento de tarefas. A Figura 2 mostra que esse simulador apresenta um módulo para a simulação de falhas, que consiste em um gerador e um monitor de falhas nos nós de execução. Este sistema está conectado à *Workflow Engine* que gerencia os jobs baseando-se em suas dependências, assegurando que uma tarefa que é liberada para execução apenas quando as tarefas das quais ela depende sejam concluídas com sucesso. Caso uma falha ocorra, o módulo *Workflow Engine* gera novas tarefas para assegurar a execução do workflow.

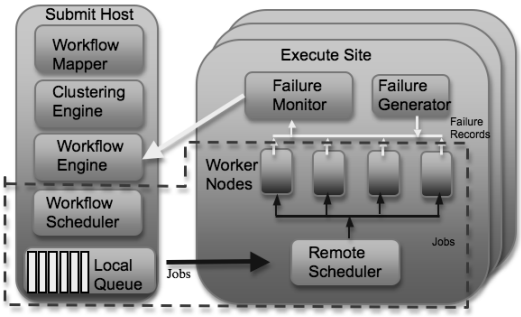


Figura 2. Arquitetura do simulador de workflows WorkflowSim

4. DynamicCloudSim

O DynamicClousSim [Bux and Leser 2015] também estende características do CloudSim, o qual assume que MVs têm performance previsível e estável. Os *hosts* e MVs são configurados com uma capacidade fixa de MIPS (milhões de instruções por segundo por núcleo). Entretanto, em infraestruturas reais como o Amazon EC2, essas suposições não são verdadeiras, embora os fornecedores destes serviços garantam velocidade mínima de processamento, capacidade de memória e armazenamento local para cada VM. O desempenho depende do hardware subjacente e do uso de recursos em MVs no mesmo *host*. Toda vez que uma tarefa é atribuída a uma MV, seu tempo de execução é calculado e o DynamicCloudSim determina se a tarefa irá suceder ou falhar, baseado em uma taxa de falhas definida pelo usuário, utilizando uma distribuição exponencial. Posteriormente, o tempo de execução de uma tarefa que falhou é calculado multiplicando o tempo de execução da tarefa por outro coeficiente definido pelo usuário, simbolizando o *overhead* de falhas. Assim, a falha em uma tarefa é representada por um aumento no seu tempo de execução.

5. Discussão

Como pode ser visto na Tabela 1, WorkflowSim e DynamicCloudSim possuem várias características extras adicionadas ao CloudSim para a interpretação e execução de workflows, e diferem entre si por suas abordagens. O WorkflowSim se preocupa com o impacto do agrupamento de tarefas no tempo de execução, um comportamento comum na execução de workflows, enquanto DynamicCloudSim é voltado aos diferentes *overheads* que são por vezes aleatórios e afetam mais especificamente o tempo de execução.

A tolerância a falhas implementada nesses dois simuladores já provê base adequada para a simulação de ataques. Pode-se entender como um tipo de falha um ataque que impossibilita uma tarefa de encerrar sua execução. Também os *overheads* podem ser tratados de forma a simular ataques de disponibilidade às redes. Ou seja, uma parte das restrições de segurança que precisam ser levadas em consideração pelos algoritmos de escalonamento já são contempladas pela simulação de falhas já disponível.

As maiores dificuldades se dão na simulação do suporte das MVs e ambientes de execução às políticas de segurança. Um exemplo seria o suporte de MVs a um determinado algoritmo de criptografia, essencial na execução de determinado tipo de workflow. Tal simulação requer não somente a alteração das variáveis já disponibilizadas, como também a alteração da representação das MVs para que se possa então estudar algoritmos de escalonamento mais específicos.

Tabela 1. Tabela de comparação das características dos simuladores CloudSim, WorkflowSim, e DynamicCloudSim. Fonte: [Bux and Leser 2015]

Característica	CloudSim	WorkflowSim	DynamicCloudSim
Atributos de performance MIPS, largura de banda, e memória	X	X	X
Atributos de performance de E/S de arquivos			X
Tempo de execução de uma tarefa dependente de outros valores além de MIPS			X
Modelagem de dependência de dados	X	X	X
Parsing de workdlows		X	X
Implementação de escalonadores de workflows		X	X
Modelagem de atrasos em diferentes camadas do SWfMS		X	
Suporte a agrupamento de tarefas		X	
Diferentes MVs em diferentes hosts	X	X	X
Atribuição aleatória de nova MV a host			X
Alocação de recursos baseado em unidades de computação			X
Mudanças dinâmicas de performance de MVs a tempo de execução			X
Modelagem de falhas durante execução de tarefas		X	(X)
Introdução de MVs não mapeadas a hosts			X

Ambos os simuladores, embora com abordagens distintas, possuem suporte à simulação de falhas e contém ferramentas que possibilitam a simulação da recuperação das mesmas. Adicionalmente, ambos disponibilizam interfaces para implementação de algoritmos de escalonamento definidos pelo usuário, bem como políticas de atribuição de tarefas às MVs.

6. Conclusão

Este trabalho apresentou uma visão geral de simuladores de gerenciamento de workflows com o objetivo de apresentar um suporte a escolha de simuladores de acordo com as características do cenário a ser simulado. O WorkflowSim e o DynamicCloudSim são extensões que abrangem características não abordadas pelos CloudSim que permitem a simulação de cenários mais próximos de nuvens públicas. Como trabalho futuro, pretende-se investigar quais dos simuladores oferecem maior suporte para implementação de algoritmos de escalonamento de workflows considerando restrições de segurança.

7. Agradecimentos

Agradecemos à FAPESP (Processo: 2015/25882-1) pelo apoio financeiro e ao LaSDPC [1] pela infraestrutura oferecida durante o desenvolvimento do projeto.

Referências

Bux, M. and Leser, U. (2015). Dynamiccloudsim: Simulating heterogeneity in computational clouds. *Future Generation Computer Systems*, 46:85–99.

Calheiros, R. N., Ranjan, R., Beloglazov, A., De Rose, C. A., and Buyya, R. (2011). Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50.

Calheiros, R. N., Ranjan, R., De Rose, C. A., and Buyya, R. (2009). Cloudsim: A novel framework for modeling and simulation of cloud computing infrastructures and services. *arXiv preprint arXiv:0903.2525*.

Chen, W. and Deelman, E. (2012). Workflowsim: A toolkit for simulating scientific workflows in distributed environments. In *E-Science (e-Science), 2012 IEEE 8th International Conference on*, pages 1–8. IEEE.

[1] Página oficial do LaSDPC: `http://lasdpc.icmc.usp.br/`.

Uma interface de rede com criptografia AES baseada na plataforma NetFPGA

Alexandre D. Negretti[1], **Danilo G. Cardoso**[1], **Cesar Marcondes**[1], **Ricardo Menotti**[1]

[1]Departamento de Computação – Universidade Federal de São Carlos (UFSCar)
Caixa Postal 676 – 13565-905 – São Carlos – SP – Brasil

{{{558214, danilo.cardoso}@comp}, {marcondes, menotti}@dc}.ufscar.br

Abstract. *This paper presents an implementation of AES encryption, largely used method in data protection, using NetFPGA open platform. The NetFPGA combines FPGA Reconfigurable Computing with four Ethernet interfaces of 1Gbps (specifically in the NetFPGA model used in this project). It was taken as base* `Crypto_nic` *project, a basic encryption project already implemented on this platform.*

Resumo. *Este artigo apresenta uma implementação de criptografia AES, método amplamente utilizado em proteção de dados, na plataforma aberta NetFPGA. A NetFPGA combina a Computação Reconfigurável de um FPGA com quatro interfaces de rede Ethernet de 1Gbps (especificamente no modelo de NetFPGA utilizado neste projeto). Foi tomado como base o projeto* `Crypto_nic`*, projeto de criptografia simples já implementado na plataforma.*

1. Introdução

A importância da segurança dos dados transmitidos por equipamentos de rede tem crescido muito nos últimos anos. Entretanto, os métodos aplicáveis que aumentem a segurança devem ser minimamente impactantes no desempenho dos equipamentos de fluxo de dados, para que este não seja comprometido. Para garantir o mínimo de perdas possível o uso de implementação diretamente em hardware se faz necessário, pois oferece um melhor desempenho comparado a implementações em software.

Esse artigo descreve a implementação de uma interface de rede — com criptografia para proteção de envio e recebimento de dados — baseada na plataforma NetFPGA [Lockwood et al. 2007]. O método criptográfico escolhido foi o AES (*Advanced Encryption Standard*), que possui a vantagem de ser altamente seguro e facilmente implementável em hardware [Miller et al. 2009]. Seu funcionamento detalhado não será abordado aqui por questões de espaço, mas convém lembrar que no processo é usada uma chave criptográfica secreta, nesse projeto especificamente, de 128 bits.

A plataforma NetFPGA possui configuração baseada em FPGA, que tem como principal atrativo sua reconfigurabilidade. Isso o torna bastante versátil, pois dessa forma é possível programar o circuito integrado da plataforma de acordo com as necessidades do projeto. A principal característica desta plataforma é a combinação de um FPGA Virtex-II Pro da Xilinx e quatro interfaces de rede integradas. O modelo utilizado nesse projeto foi a NetFPGA 1G, que possui portas Ethernet 1 Gbps. Além disso, seu código fonte é público, o que facilita o seu estudo e aprendizado, de tal maneira que um de seus projetos de referência, o `Crypto_nic`, foi utilizado como base para esse trabalho.

2. Abordagem

Para iniciar os trabalhos com a plataforma NetFPGA foi configurada uma VM (*Virtual Machine*) em nosso grupo de pesquisa. Isso facilita a configuração rápida de um ambiente de projeto e simulação de hardware que, além das ferramentas proprietárias da Xilinx, possui scripts que acompanham a plataforma NetFPGA. Embora o objetivo final do trabalho seja testar a implementação diretamente no hardware, são requeridas muitas simulações neste ambiente até se obter o funcionamento correto desejado. O uso das ferramentas de síntese em uma VM não degrada consideravelmente o desempenho dessas e é plenamente justificado pela facilidade de configuração do ambiente de desenvolvimento.

Para compreender em profundidade o funcionamento da plataforma NetFPGA foram usados os três principais projetos de referência: (i) uma placa de rede, (ii) um comutador Ethernet e (iii) um roteador IP. Nesse trabalho estamos tratando o recebimento de pacotes de dados que devem ser transmitidos pela rede, para isso, usamos como ponto de partida o projeto da placa de rede (`reference_nic`), que configura a NetFPGA como como um conjunto de quatro interfaces de rede.

O projeto instancia oito módulos `rx_queue` que têm a função de tratar o recebimento de dados para cada interface da porta Ethernet e barramento PCI ($4 * 2 = 8$). Do mesmo modo, o projeto `reference_nic` ainda instancia oito módulos `tx_queue` que tratam a transmissão de pacotes por cada interface através da porta Ethernet e barramento PCI. Os módulos `rx_queue` e `tx_queue` fazem a interface entre o FPGA, o controlador MAC e o controlador PCI. Na Figura 1 é apresentado o pipeline completo de processamento do projeto `reference_nic`, na qual os módulos `rx_queue` e `tx_queue` são evidenciados nas extremidades.

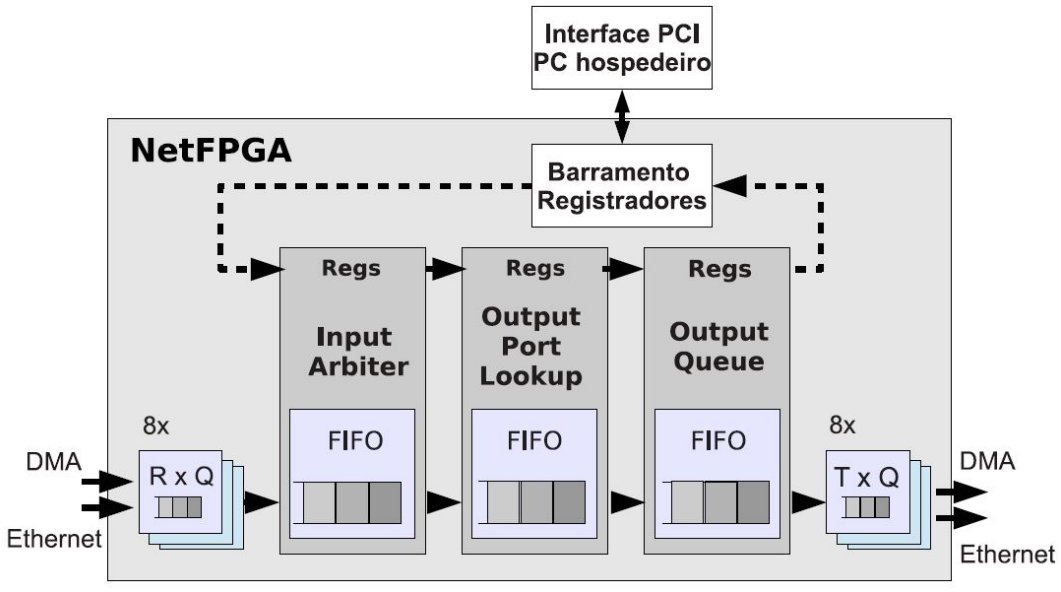


Figura 1. Pipeline de referência da NetFPGA configurada como interfaces de rede padrão [Marcondes et al. 2015]

Como citado anteriormente, o projeto base para este trabalho é o projeto `Crypto_nic`. Esse projeto implementa uma criptografia simples XOR a partir de uma

chave criptográfica.

O projeto segue os estágios de uma fila como apresentado no projeto `reference_nic`, as filas andam 64 bits por vez, ou seja, 8 bytes que serão referidos como uma palavra de um pacote. O pacote do tipo TCP possui um cabeçalho, o qual não pode ser criptografado, já que as informações contidas nele são necessárias para determinar seu destino, saber sua origem, flags, entre outras coisas. Desse modo, o único trecho do pacote que deverá ser criptografado é o trecho seguinte ao cabeçalho — chamado *payload* — que contém verdadeiramente os dados daquele pacote. Tais dados são encontrados a partir dos últimos 16 bytes da quinta palavra do pacote.

O objetivo do trabalho é substituir a criptografia simples (XOR) fornecida pelo `Crypto_nic` pela criptografia de um módulo AES. Entretanto, há um inconveniente, já que o módulo AES trabalha com no mínimo 128 bits e a fila do `Crypto_nic` é de 64 bits.

3. Resultados preliminares

A abordagem deste trabalho não incluiu o desenvolvimento do módulo de criptografia AES, uma vez que já havia alguns implementados na plataforma OpenCores[1], na qual usuários publicam trabalhos desenvolvidos em hardware. Entre os cores analisados e testados, destacaram-se os apresentados na Tabela 1. Os cores foram comparados através da análise de sua documentação e síntese do projeto com alvo na placa Virtex-II Pro 50, que é o FPGA presente na NetFPGA utilizada.

Tabela 1. Percentual de recursos alocados no device Virtex-II Pro 50

Core	Slices	Flip Flops	LUTs	IOBs	BRAMs	Warnings
tiny_AES 128	49%	16%	40%	47%	50%	0
aes-128_pipelined	92%	35%	64%	47%	0%	0
aes_crypto_core	33%	2%	31%	32%	0%	4

Tendo em vista a documentação, os teste realizados, os dados de síntese, e o número de *warnings* (possíveis problemas encontrados na etapa de síntese do módulo), o projeto de módulo de criptografia AES escolhido para ser utilizado dentro do trabalho foi o módulo: tiny_AES 128. Os dados completos da síntese deste core são apresentado na Tabela 2.

Tabela 2. Recursos alocados pelo core tiny_AES no dispositivo Virtex-II Pro 50

Elemento	Usado	Disponível	Percentual
Slices	11608	23616	49%
Flip Flops	7776	47232	16%
4 input LUTs	19008	47232	40%
IOBs	385	812	47%
BRAMs	116	232	50%

[1] `http://www.opencores.org`

4. Trabalhos futuros

Os esforços futuros serão aplicados à sincronização entre o módulo de criptografia AES e o projeto `Crypto_nic`. A maior parte da complexidade está na diferença de fluxo de dados entre os dois módulos, já que o método de criptografia AES só trabalha a partir de 128 bits e o projeto base `Crypto_nic` tem fluxo de 64 bits. As opções de tratamento são variadas, entre elas criptografar dois trechos de 64 bits seguidos do pacote ao mesmo tempo, ou apenas 64 bits de informação do pacote e todo o resto preenchido com bits '0', ou ainda duplicar o trecho de 64 bits do pacote para preencher os 128 bits do módulo AES.

Ainda há uma outra alternativa, é a utilização de outro projeto base da plataforma NetFPGA chamado `sram_arbiter`, que é utilizado em um projeto de `Firewall` da plataforma. O `sram_arbiter` é responsável em tratar entradas maiores que o fluxo do `Firewall`, e a ideia é usar a mesma lógica de modo invertido nesse trabalho.

Todos esses tratamentos são devidos a um problema intrínseco da criptografia em geral, que são os modos de tratamento de `Padding` (atitudes a serem tomadas quando um dado é menor que o fluxo de entrada do módulo criptográfico) [Kaliski 2000].

Por último, outro problema na inserção do módulo AES é o `delay` de processamento da criptografia e os efeitos que isso pode causar no fluxo dos dados. O impacto deste atraso ainda será investigado para a conclusão do trabalho.

Referências

Kaliski, B. (2000). PKCS# 5: Password-based cryptography specification version 2.0.

Lockwood, J. W., McKeown, N., Watson, G., Gibb, G., Hartke, P., Naous, J., Raghuraman, R., and Luo, J. (2007). NetFPGA–An Open Platform for Gigabit-Rate Network Switching and Routing. In *Proceedings of the 2007 IEEE International Conference on Microelectronic Systems Education*, MSE '07, pages 160–161, Washington, DC, USA. IEEE Computer Society.

Marcondes, C., Menotti, R., Goulart, P., Cunha, I., and Vieira, M. (2015). NetFPGA: Programando Processamento de Pacotes em Hardware. In *Anais do SBRC 2015*, Mini-Curso 4, pages 1–49. Sociedade Brasileira de Computação - SBC Livros.

Miller, F. P., Vandome, A. F., and McBrewster, J. (2009). *Advanced Encryption Standard*. Alpha Press.

Mecanismo de criptografia negável aplicado no armazenamento de arquivos multi-nuvem

Victor Marcelino Nunes[1], **Rafael Mira de Oliveira Libardi**[1], **Julio Cezar Estrella**[1]

[1] Instituto de Ciências Matemáticas e de Computação (ICMC)
Universidade de São Paulo (USP)

{victor.marcelino.nunes, mira}@usp.br, jcezar@icmc.usp.br

Abstract. *This work aims to model and implement a deniable encryption mechanism in a multi-cloud file storage platform. This will provide security levels and higher protection to the users of the platform, that even if coerced to provide a key may provide a false key that does not reveal sensitive information.*

Resumo. *Este trabalho tem como objetivo modelar e implementar um mecanismo de criptografia negável em uma plataforma de armazenamento de arquivos em multi-nuvem. Isto irá proporcionar níveis de segurança e proteção mais elevados aos utilizadores da plataforma, que mesmo que coagidos a prover uma chave, poderão fornecer uma chave falsa que não revela a informação sensível.*

1. Introdução

A computação em nuvem surgiu nos últimos anos como um novo paradigma computacional que busca satisfazer a recente promessa de computação por demanda e utilidade. Com o surgimento deste paradigma, cada vez mais organizações estão buscando migrar seus sistemas existentes e seus dados para a nuvem com o objetivo de reduzir o custo com infraestrutura de tecnologia da informação [Gens, 2008, Armbrust et al., 2010]. Porém, não há garantia técnica alguma de que os dados armazenados não serão expostos para pessoas não autorizadas. Uma das maneiras de se melhorar o armazenamento de arquivos é utilizando arquiteturas multi-nuvem. Com isso, é criada uma nova camada de abstração, que consegue obter maior segurança e disponibilidade [Bessani et al., 2011]. Aliado à estas propostas, está a utilização da criptografia para proteger o conteúdo dos arquivos. Dessa forma, somente quem possui a chave pode recuperar os dados criptografados. Contudo, a criptografia usual está sujeita à ataques. Caso outro usuário obtenha a chave de segurança, ela pode facilmente recuperar os arquivos criptografados.

Para solucionar esta questão, o conceito de Criptografia Negável foi introduzido [Canetti et al., 1997]. A criptografia negável tem como foco a utilização de um arquivo falso, que é usado como isca para proteger os dados originais. Ambos os arquivos, o original e o falso, são codificados numa mesma cifra, sendo a utilização de chaves distintas o único modo de recuperá-los. O usuário, portanto, possui duas chaves: uma que decodifica os dados originais e outra que decodifica o arquivo isca. Com isto, a vítima pode fornecer uma chave que não leve ao arquivo protegido, mas sim ao arquivo isca. Isto pode ser particularmente útil em diversos contextos, incluindo no armazenamento em nuvem.

2. Implementação da Criptografia Negável

Para implementar uma mecanismo de Criptografia Negável foi utilizado o método de esteganografia, adaptado de forma correta para representar as devidas funcionalidades. Esteganografia é a arte de esconder informações em meios que não se possa observar a diferença entre seu estado original e o estado com a informação armazenada.

A esteganografia foi utilizada de modo que se possa esconder dois arquivos quaisquer em uma imagem e que ambos possam ser recuperados utilizando sua respectiva chave de segurança. Para tal funcionamento, foram consideradas, para cada pixel da imagem, as três tonalidades que representam sua cor no sistema de cores RGB. Com isso, os bits de cada arquivo substituirão os bits menos significativos de cada tonalidade de cada pixel. Dessa forma, cada pixel resultante dessas alterações não apresentará modificações aparentes, fazendo com que a imagem resultante também não sofra alterações visíveis, validando, assim, a esteganografia. Para diferenciar o arquivo original do arquivo isca, os mesmos são armazenados alternadamente entre os *pixels* da imagem. Além dos dados dos arquivos, o nome e a senha de cada um deles também são armazenados na imagem. A obtenção dos arquivos a partir da imagem é feita com base na chave de segurança, que determinará qual arquivo será recuperado, e na ordem que os bits foram substituídos em cada pixel. A Figura 1 mostra as etapas desde a obtenção dos dados de um arquivo até o armazenamento de parte dele num pixel da imagem.

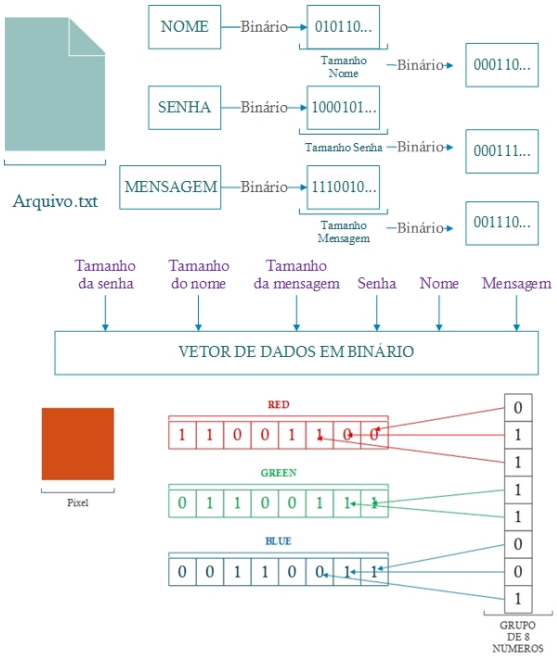


Figura 1. Etapas para esconder parte do arquivo em um pixel da imagem.

3. Experimentos e Resultados

O primeiro teste a ser realizado foi a avaliação de quanto o tamanho do arquivo final aumentou em relação ao arquivo original. Para isso, foi considerado que o arquivo falso possui um tamanho igual ou inferior ao arquivo original. O teste foi realizado com arquivos de diferentes tamanhos e seu resultado é representado pela Figura 2.

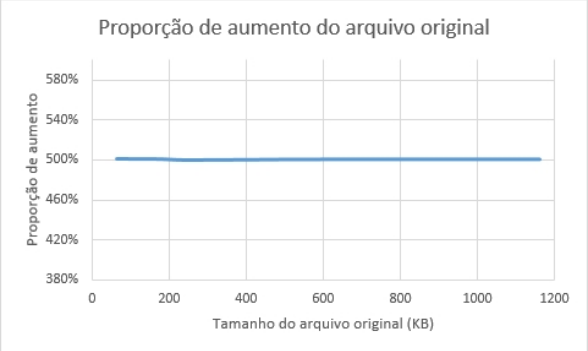


Figura 2. Variação da proporção do aumento do arquivo x Tamanho do arquivo original

Na segunda bateria de testes, foi considerado a integração do módulo de criptografia negável no projeto *FlexSky* [Libardi et al., 2014]. Esta simulação foi feita em um ambiente de núvem privado, no laboratório de pesquisa *LaSDPC - Laboratório de Sistemas Distribuídos e Programação Concorrente* [1]. Este teste foi realizado para avaliar a taxa de transferência e o tempo de *upload* e *download* de um arquivo quando o mesmo passa pelo módulo de criptografia negável. Para isso, foram utilizados arquivos de diferentes tamanhos. Com isso, será possível analisar se o módulo de criptografia negável afetará de forma negativa o projeto no qual ele será inserido. Os dados foram coletados por meio de um arquivo de *log* gerado pelo projeto *FlexSky*. O arquivo de *log* gerado informou o tempo e a taxa de transferência de um arquivo quando o mesmo passa pelo módulo de criptografia negável, já inserido no projeto *FlexSky*.

Os gráficos representados pela Figura 3 e Figura 4 mostram, respectivamente, o tempo variando conforme o tamanho do arquivo e a taxa de transferência variando conforme o tamanho do arquivo. Ambos os gráficos mostram a situação de *upload* e *download* dos arquivos.

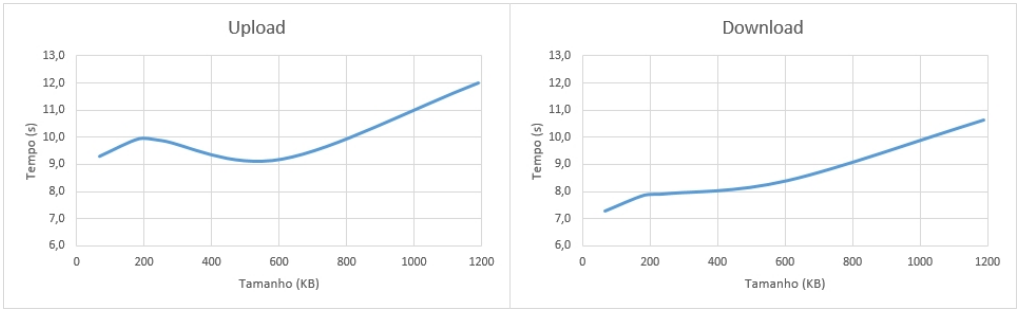


Figura 3. Variação do tempo x Tamanho do arquivo (*upload*/*download*).

4. Conclusão

Os resultados obtidos por meio dos experimentos foram conforme o previsto. Para que o arquivo final possuísse dois arquivos escondidos em si (o arquivo original e o arquivo

[1]Página oficial do LaSDPC: `http://lasdpc.icmc.usp.br/`.

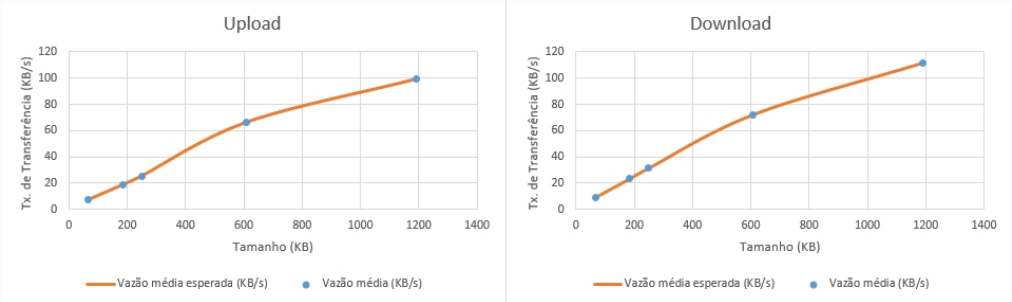


Figura 4. Variação da taxa de transferência x Tamanho do arquivo (*upload*/*download*).

falso), era esperado que o tamanho do mesmo aumentasse em relação ao arquivo original [Figura 2]. Além disso, dado esse aumento, o tempo e a taxa de transferência do mesmo, ao ser armazenado ou baixado da nuvem utilizando o módulo de criptografia negável [Figura 3 e Figura 4], se comportaram conforme o esperado. Apesar do aumento do tamanho do arquivo ter sido de cinco vezes, o tempo e a taxa de transferência do mesmo não impactaram na mesma proporção, mostrando-se, assim, toleráveis ao projeto *FlexSky*. Tal tolerância também se deve ao fato do aumento da segurança que este módulo proporciona ao projeto, além de que todos os resultados obtidos se comportaram conforme o previsto. Com isso, pode ser concluído que o módulo apresentado neste documento não afetou negativamente o projeto no qual foi inserido, possibilitando, assim, sua implementação no mesmo.

5. Agradecimentos

Agradeço à FAPESP (Processo: 2014/12612-3) pelo apoio financeiro e ao LaSDPC pela infraestrutura oferecida durante o desenvolvimento do projeto.

Referências

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. (2010). A view of cloud computing. *Commun. ACM*, 53(4):50–58.

Bessani, A., Correia, M., Quaresma, B., André, F., and Sousa, P. (2011). Depsky: dependable and secure storage in a cloud-of-clouds. In *Proceedings of the sixth conference on Computer systems*, EuroSys '11, pages 31–46, New York, NY, USA. ACM.

Canetti, R., Dwork, C., Naor, M., and Ostrovsky, R. (1997). Deniable encryption. In *Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '97, pages 90–104, London, UK, UK. Springer-Verlag.

Gens, F. (2008). It cloud services user survey, pt.2: Top benefits and challenges. `http://hadoop.apache.org`. (Acessado em 14 de março de 2013).

Libardi, R. M. D. O., Bedo, M. V. N., Reiff-Marganiec, S., and Estrella, J. C. (2014). Mssf: A step towards user-friendly multi-cloud data dispersal. In *2014 IEEE 7th International Conference on Cloud Computing*, pages 952–953.

Estudos sobre a Medição do Desempenho de Sistemas de Processamento de Fluxos de Dados

Fabrício A. Oliveira, André L. S. Gradvohl

Faculdade de Tecnologia – Universidade Estadual de Campinas (UNICAMP)
Rua Paschoal Marmo, 1888 – Jardim Nova Itália, Limeira-SP

falveso@unicamp.br, gradvohl@ft.unicamp.br

Abstract. *This paper presents a general view of online stream processing and the importance of benchmarking these systems. In addition, the paper presents a real performance test of the Apache Storm system.*

Resumo. *Este artigo apresenta uma visão geral do que são os sistemas de processamento de fluxo e a importância da realização de testes para medição de desempenho deles. O texto também apresenta um teste de desempenho do sistema de processamento de fluxo Apache Storm.*

1. Introdução

Sistemas de processamento de fluxos de dados ou *Complex Event Processing* (CEP) *systems* foram criados para lidar com um grande volume de dados que chegam ao sistema como fluxos de eventos. Esses sistemas podem ser utilizados na análise de tendências em Redes Sociais, no Marketing Dirigido, no combate a spams, no Mercado Financeiro, tanto na análise de tendências, quanto no combate a fraudes, ou na própria segurança de uma rede de computadores, dentre outras aplicações [1].

O número de aparelhos eletrônicos ligados à rede e que produzem informação é muito alto o que gera um desafio na área de Tecnologia da Informação no que diz respeito ao processamento dos dados em tempo hábil.

Uma alternativa seria armazenar toda essa informação para análise posterior. Entretanto, essa pode não ser uma alternativa viável, já que a quantidade de dados produzidos é grande. Também não se pode esquecer que muitos sistemas necessitam de análise imediata dos eventos e com baixa latência, quase que em tempo real.

Dado esse cenário, notamos a necessidade de se avaliar o desempenho de sistemas CEP, no que diz respeito à qualidade no tratamento de grandes quantidades de eventos produzidos (*throughput*), taxa de erros que o sistema consegue tratar ou descartar, latência no processamento dos eventos, tolerância a falhas e elasticidade do sistema. Esse artigo relata um trabalho em andamento e se limita a fazer uma análise no tempo médio para o processamento de um evento e o *throughput* do sistema.

2. O Apache Storm

Nessa análise, escolhemos o *Apache Storm* para avaliação por se tratar de um sistema CEP distribuído de código fonte aberto e que já atingiu certo grau de maturidade. Atualmente o *Apache Storm* está na versão 1.0.1, disponibilizada em maio de 2016. O

Apache Storm necessita de um *cluster* para o processamento da informação que chega ao sistema na forma de tuplas.

Um *cluster Storm* é constituído por um nó principal, denominado *nimbus* e por nós do tipo *worker*. O *nimbus* faz o gerenciamento e monitoramento do *cluster*, isto é, verifica a ocorrência de falhas e distribui a carga de trabalho entre os nós *worker*. Um conjunto de processos *ZooKeeper* [2] faz a coordenação entre o nó *nimbus* e os nós *worker*. Todo o estado do processamento é mantido pelo *ZooKeeper* (Figura 1), isso ajuda a estabilizar o sistema e melhorar sua tolerância às falhas.

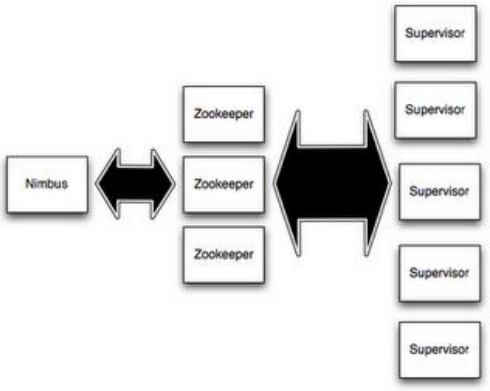


Figura 1 - Estrutura de um cluster storm [5].

Além disso, o *Apache Storm* necessita da definição de uma topologia, conforme ilustra a Figura 2. Uma topologia é um grafo direcionado acíclico (DAG – *Directed Acyclic Graph*) que representa a conexão entre os nós de processamento [3]. O DAG é estabelecido na linguagem de programação *Java* e define o caminho que as tuplas deverão percorrer durante o processamento.

Os operadores, que compõem a topologia, são denominados de *spouts* e *bolts*. Os *spouts* são nós primários na topologia, isto é, filtram o fluxo de entrada e geram uma sequência de tuplas, para posterior processamento. Podem, por exemplo, realizar uma conexão, por meio da *Application Programming Interface* (API) do *Twitter*, e emitir um fluxo de *tweets* na forma de tuplas.

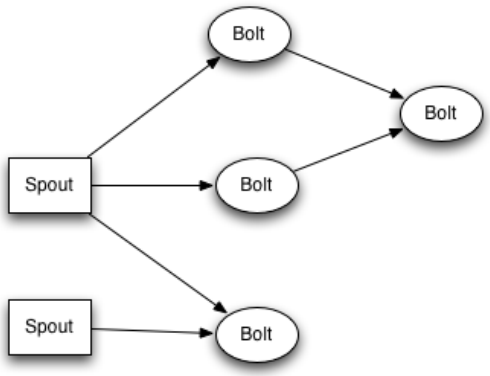


Figura 2 - Exemplo de uma topologia [5].

Os *bolts*, por outro lado, são nós que podem realizar um processamento complexo da informação, isto é, processam as tuplas, agregam fluxos de eventos, dentre outras operações.

A interligação dos nós para compor a topologia é feita por meio de rotinas da API do *Storm*. Apresentamos um exemplo prático da definição de uma topologia em liguagem de programação *Java,* ilustrada na Figura 3. A interligação entre cada *spout* e o *bolts* ou entre os *bolts* é feita por meio do método *shuffleGrouping*.

Na terceira linha do código, na Figura 3, o método *setBolt* paralelizará a execução do *bolt*, ou seja, a execução será feita por três *threads* que representam o *bolt* na topologia. Isso implica que a interligação do *spout* e o *bolt*, pelo método *shuffleGrouping*, faz com que a distribuição das tuplas seja aleatória entre os três processos que representam o *bolt* na topologia.

```
1  TopologyBuilder builder = new TopologyBuilder;
2  builder.setSpout("words", new TestWordSpout, 10);
3  builder.setBolt("exclaim1", new ExclamationBolt(), 3).shuffleGrouping("words");
4  builder.setolt("exclaim2", new ExciamationBolt(), 2).shuffleGrouping("exclaim1");
```

Figura 3 - Definição de uma topologia no Storm [5].

3. Elaboração do Teste

O nosso exemplo, que foi baseado na topologia *ExclamationTopology* [4], consistirá na emissão de tuplas de forma aleatória, lidas de um conjunto de cinco valores, ou seja, ["nathan", "mike", "jackson", "golda", "bertels"]. O processamento consiste em adicionar três símbolos de exclamação no primeiro e no segundo *bolt*. Portando, uma palavra possui seis símbolos de exclamação como sufixo, ao final do processamento.

A marcação de tempo ocorre sempre que uma palavra é emitida ou recebida por um operador na topologia, conforme ilustrado nas figuras 4 e 5. Os valores de tempo da tupla são gravados em arquivo juntamente com o número de identificação de cada tupla. A execução do teste ocorre por dois minutos e meio, ao final, faz-se uma análise dos arquivos gerados.

Observa-se, na Figura 4, nas linhas 6 e 7, que utilizamos o tempo como *id* (identificador único) da tupla. É necessária a especificação de um *id* para que possamos acompanhar o andamento da informação na topologia e o rastreamento de erros. A classe *DadosTupla* foi criada para o armazenamento do *id*, a palavra e o tempo, o que facilita a troca de informação entre os operadores da topologia.

```
1  @Override
2  public void nextTuple() {
3      final String[] words = new String[] { "nathan", "mike", "jackson", "golda", "bertels" };
4      final Random rand = new Random();
5      final String word = words[rand.nextInt(words.length)];
6      long tempTempo = System.nanoTime();
7      DadosTupla dados = new DadosTupla(tempTempo, word, tempTempo);
8      this.inicio = tempTempo;
9      _collector.emit(new Values(dados));
10 }
```

Figura 4 - Emissão da tupla e marcação de tempo pelo *spout*.

```
@Override
public void execute(Tuple tuple) {
    long tempoTemp = System.nanoTime();
    DadosTupla dados = (DadosTupla) tuple.getValue(0);
    dados.setNome(dados.getNome() + "!!!");
    dados.setTempoBolt(System.nanoTime());
    _collector.emit(new Values(dados));
    _collector.ack(tuple);
}
```

Figura 5 - Emissão da tupla e marcação de tempo pelo *bolt*.

4. Medições Obtidas no Teste

Após a execução dos testes e a coleta de tempo e quantidade de tuplas geradas pelos nós de processamento, resumimos o resultado do teste na Tabela 1. Observamos que o tempo de transmissão de uma tupla entre os nós é de aproximadamente 0,20 milissegundos. A latência no método *execute* do primeiro *bolt* foi maior que no segundo. Isso era esperado, pois o primeiro *bolt* deve realizar a transferência da tupla, enquanto que no segundo isso não acontece. Com relação à vazão do sistema, a quantidade de tuplas emitidas pelo segundo *bolt* foi de 7.586 tuplas/segundo, com um total de 1.137.900 tuplas geradas no intervalo de 2,5 minutos de execução.

Tabela 1 - Resultados das medições propostas.

Operador	Latência do método *execute* (milissegundos)
Primeiro *bolt*	0,38
Segundo *bolt*	0,22

5. References

[1] Mendes, M., Bizarro, P. e Marques, P. (2008) “A Framework for Performance Evaluation of Complex Event Processing Systems”. In: Proceedings of the second international conference on Distributed event-based systems pp: 313-316

[2] Apache ZooKeeper. ZooKeeper 3.4 Documentation. Disponível em: https://zookeeper.apache.org/doc/trunk/. Acesso em: 26 de abril de 2016.

[3] Gradvohl, A., Senger, H., Arantes, L. e Sens, P. (2014) “Comparing Distributed Online Stream Processing Systems Considering Fault Tolerance Issues”. In: Journal of Emerging Technologies In Web Intelligence, Vol. 6, No. 2, p. 174.

[4] Storm Starter Project. Example Storm Topologies. Disponível em: https://github.com/apache/storm/tree/v1.0.1/examples/storm-starter. Acesso em: 20 de abril de 2016.

[5] Apache Storm. Tutorial. Disponível em: http://storm.apache.org/releases/current/Tutorial.html. Acesso em: 04 de maio de 2016.

Analytical Performance Prediction of Large-Scale Business Processes*

Waldir Edison Farfan Caro, Kelly Rosa Braghetto

[1]Department of Computer Science, University of São Paulo
Rua do Matão, 1010, Cidade Universitária, 05508-090, São Paulo, Brasil

{waldirc,kellyrb}@ime.usp.br

Abstract. *The execution of automated business processes can be very costly in terms of computational resources. Optimizing their resource utilization while meeting quality of service requirements is highly desirable. This work introduces a new method for generating analytical models to predict performance of resource-aware, large-scale business processes. Our method uses a mean field model to approximate the behavior of a large set of process instances competing for resources. Differently from related works, this approach does not suffer from the state space explosion problem generally associated with Markovian models.*

1. Introduction

Business processes must be capable of handling thousands (even millions) of concurrent requests. Sites such as Amazon, eBay, and Apple Inc. offer hundreds of different services and receive millions of accesses every day. The scale of these scenarios introduces new requirements regarding efficiency, reliability, availability and other quality-of-service related properties, which involve both qualitative and quantitative aspects of the system.

The overall *performance* is perhaps the most important quantitative aspect that directly impacts the system's quality of service. Automated business process activities usually depend on different computational resources to be executed. The expected performance of a business process depends on how resources are provisioned and used. Optimizing resource usage while meeting quality-of-service requirements is an essential task, even on platforms where the number of resources are quasi-infinite, such as in clouds.

Markovian methods are the most commonly used to predict the performance of business processes, but they suffer from the *state space explosion* problem. When we model a business process with several instances executing concurrently, sharing a same set of common resources, the state space of the model tends to grow exponentially in the number of instances, and the numerical solution requires prohibitive computational efforts in terms of processing time and memory consumption.

An interesting technique to overcome this problem is provided by the *Mean Field Theory* (MFT), that approximates a stochastic dynamical system as a deterministic one [Kurtz 1970, Hillston 2005, Tribastone and Gilmore 2011]. The main idea of MFT is to treat a many-body system not by explicitly modeling all two-body interactions but by approximating the interaction of one body with the remaining ones by an average potential (or effective interaction) created by the other bodies. These approximation techniques allow for analyses that can be efficiently computed.

*This work has received financial support from CAPES and NAPSoL-PRP-USP.

This work introduces a new method for generating analytical models to predict performance of resource-aware, large-scale business processes. Our method uses a MFT model to approximate the behavior of a large set of process instances competing for resources. The method allows to predict bottlenecks caused by resource constriction, to plan resource provisioning, and to study other performance related issues of large-scale business processes.

2. Predictive Performance Analysis: An Overview

Predictive performance analysis aims to measure performance of systems before these are implemented. The analytical modeling approach uses mathematical models to describe and numerically analyze certain aspects of interest of a system. Analytical modeling for performance evaluation is generally made with stochastic models that have as underlying formalism a Markov process. From the solution of such stochastic models, one can extract several performance indicators, such as the utilization rate of the modeled resources, the throughput of the tasks, the service time (e.g., the average time to treat a request), etc.

2.1. Analyzing Business Processes Using Deterministic Approximation Models

In the "conventional" analytical modeling with Markovian formalisms, the interacting entities of a system are modeled under a "microscopic" point of view – i.e., entities and their interactions are explicitly described in the model; the global state of the system is defined by the local state of each entity. In the case of a business process, we can see a process instance as an entity of the analytical model and the interaction between them by means of the access to shared resources. For a process with several instances executing concurrently, we can easily obtain a model with a huge state space (> 1 million states).

To account the state-space explosion problem we will model a "macroscopic" view of the business process and its instances, describing all the possible states of one individual instance of the business process and their respective occupation measures. However, the rates of the transitions between these states must be defined taking into account the interactions in the "microscopic" view of the business process.

Related works have already reported the use of deterministic approximations. Benaïm and Boudec [Benaïm and Boudec 2008] studied the behavior of systems composed of identical objects that interact between themselves, in which the evolution of each object is given by a Markov Chain with a finite number of states. Bobbio et al. [Bobbio et al. 2008] reported the use of a mean field method to model and analyze several examples of interacting Markovian queues.

3. Resource-Aware Business Process Models

In this work, we introduce a technique to performance prediction of business processes that uses BPMN process diagrams enriched with information concerning resources requirements. A BPMN process diagram is a directed graph. Our technique can be applied over a subclass of the process diagrams that can be formed from the following BPMN constructors: *start event*, *end event*, *atomic activity*, *sequence flow*, *exclusive gateway*, and *parallel gateway*. Figure 1 shows an example of the diagrams considered in this work.

The execution time of a business process instance is related to the resource requirements of its activities and may vary with the workload of the system. Resource

requirements can be expressed as simple text annotations over the process diagram. For example, the annotation $([R_1, 1] \vee ([R_2, 2] \wedge [R_3, 3]))$ indicates that activity A needs resource R_1 to perform 1 unit of work, or R_2 to perform 2 and R_3 to perform 3.

4. Deterministic Approximation

The mean field method works with continuous state spaces, rather than discrete state spaces as in the stochastic methods. Our method initially uses a *Continuous Time Markov Chain* (CTMC) derived from the BPMN model. This CTMC defines the behavior of a single instance (its possible states). To obtain the CTMC from the BPMN process diagram, we identify three substructures that need a different treatment for their translation into the set of states that will compose the CTMC:

- *Atomic Activity* – its modeling in the CTMC is performed in two phases: (i) the allocation phase, representing the allocation of the resources required by the activity; and (ii) the execution phase, representing the execution of the activity after the allocation of the resources.
- *Parallel gateway* – we need to model the parallel execution of all possible flows that diverge/converge from this gateway. To accomplish this, we model all the permutations that produce the set of possible flows as if they were executed sequentially in the CTMC (benefiting from properties of the exponential distribution).
- *Exclusive gateway* – if divergent, we process each exclusive flow separately; and if convergent, we join all the previous exclusive flows.

From the CTMC for a single instance, we generalize for all the instances. This is accomplished by a *Population Continuous Time Markov Chain* (PCTMC), which represents the counting variables for each state defined in the previous step. Moreover, transitions rules between the states are defined. For each transition, we indicate the number of consumed instances, the produced instances and also the execution rate of the transition. Analyzing the structure and characteristics of the PCTMC, we verify the conditions that must be meet (density dependency and scalability) to guarantee that the stochastic system can be approximated by a deterministic limit [Bortolussi et al. 2013]. This can be intuitively understood by considering the *Law of Large Numbers* (LLN), which states that, for a random experiment performed a large number of times, the average of the obtained results is close to the expected value.

The final approximation is known as *Continuous Time Dynamical System* (CTDS) and is defined as a set of *Differential Ordinary Equations* (ODEs). From the solution of the ODEs, performance indicators of the business process can be extracted. Despite the use of approximations, the solutions obtained with the proposed approach are very close to the exact ones, as shown in Figure 2.

5. Conclusions

Mean-field limit is a good approximation for the behavior of stochastic systems with large populations. In this paper, we have argued that this is the case for business processes with a large number of instances competing for the access to limited resources.

We have briefly described the mapping of a business process, initially modeled as a BPMN process diagram enriched with resource management information, into its

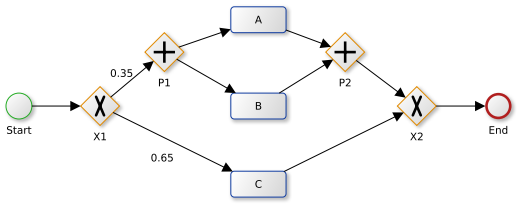
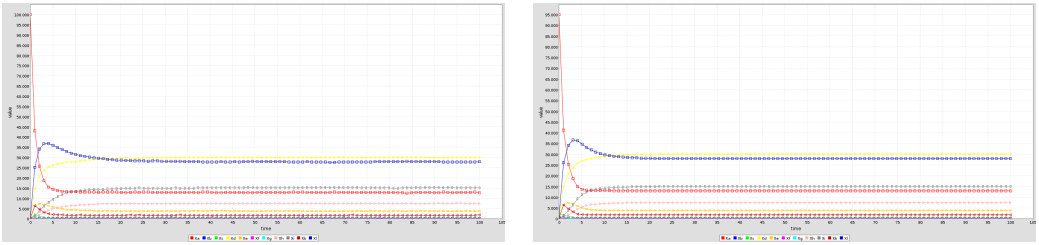


Figure 1. A well-defined BPMN process diagram.



(a) Stochastic behavior (b) Deterministic behavior

Figure 2. Stochastic (a) and deterministic (b) behaviors for the BPMN process diagram in Figure 1.

respective mean-field limit as a set of ODEs describing the behavior of a Continuous Time Deterministic System (CTDS).

The modeling approach we have discussed here overcomes the state space explosion problem of the Markovian models. Our approximation models can be numerically solved in a very efficient manner, by means of ODEs. From the solution of these models, we are able to obtain average indexes in reliable computing times, enabling us to study several performance-related issues of large-scale, resource-aware business processes.

References

Benaïm, M. and Boudec, J.-Y. L. (2008). A class of mean field interaction models for computer and communication systems. *Performance Evaluation*, 65:823–838.

Bobbio, A., Gribaudo, M., and Telek, M. (2008). Analysis of large scale interacting systems by mean field method. In *Proceedings of the Fifth International Conference on Quantitative Evaluation of Systems*, QEST'08, pages 215–224. IEEE.

Bortolussi, L., Hillston, J., Latella, D., and Massink, M. (2013). Continuous approximation of collective system behaviour: A tutorial. *Performance Evaluation*, 70(5):317–349.

Hillston, J. (2005). Fluid flow approximation of PEPA models. In *Second International Conference on the Quantitative Evaluation of Systems*, QEST'05, pages 33–42. IEEE.

Kurtz, T. G. (1970). Solutions of ordinary differential equations as limits of pure jump Markov processes. *Journal of Applied Probability*, 7(1):49.

Tribastone, M. and Gilmore, S. (2011). Scaling performance analysis using fluid-flow approximation. *Rigorous Software Engineering for Service-Oriented Systems*, pages 486–505.

Execução de aplicações Mapreduce utilizando paralelismo local e distribuído

Darlon Vasata[1,2], **Liria Matsumoto Sato**[1]

[1] Escola Politécnica da Universidade de São Paulo
Departamento de Engenharia de Computação e Sistemas Digitais
Av. Professor Luciano Gualberto s/n - CEP 05508-110
São Paulo, SP, Brasil

[2]Instituto Federal do Paraná – Campus Cascavel
Av. das Pombas, 2020 – CEP 85814-800
Cascavel, PR, Brasil

darlon.vasata@ifpr.edu.br, liria.sato@poli.usp.br

Abstract. *The problem of processing great quantities of data is a topic that received significant emphasis in recent years, with its challenges and complexity. We present a library for developing and running applications, following an extension of the Mapreduce model. We focus on obtaining performance, and to achieve this goal, we propose to use threads for intra-node operations and MPI for parallel communication among machines from the distributed environment, in a way that Map and Reduce tasks with different functions can be performed independently.*

Resumo. *O problema do processamento de grandes quantidades de dados é um tema que obteve destaque significativo nos últimos anos, com sua complexidade e desafios. Neste trabalho apresentamos uma biblioteca para desenvolvimento e execução de aplicações seguindo uma extensão do modelo Mapreduce. O foco do trabalho é a obtenção de desempenho nas aplicações, e para atingir tal objetivo, propomos a utilização de threads para a exploração do paralelismo intra-nó e MPI para a comunicação entre as diversas máquinas do ambiente distribuído, de maneira que a execução das tarefas Map e Reduce sejam independentes, e diferentes funções podem ser executadas.*

1. Introdução

A preocupação com o processamento de grandes quantidades de dados teve destaque significativo nos últimos anos, devido à grande geração de dados a partir de fontes como experimentos científicos, geolocalização, redes sociais, dispositivos sensores, dentre outros. A partir desses dados novas informações de valor podem ser obtidas, porém explorá-los requer novos desafios que excedem os atuais limites da computação, como acesso à memória e processamento. A exploração de dados neste cenário tornou-se popular com o nome de Big Data.

O modelo Mapreduce [Dean and Ghemawat 2008] obteve destaque neste cenário, devido à sua facilidade de programação, alta escalabilidade e tolerância a falhas. Neste, o desenvolvedor elabora funções Map e Reduce, utilizando dados organizados em pares

(k, v), de forma que k é uma chave e v um valor associado à chave k. A função Map emite os pares e o ambiente de execução se encarrega de agrupar todos os valores v relacionados à mesma chave k como entrada para a função de Reduce. Esta, por sua vez, processa estes valores e emite os resultados de saída da aplicação. O modelo é executado em diversas fases, com Map, Shuffle, Merge e Reduce. As fases Map e Reduce realizam operações definidas pelo usuário, e operam sobre os pares (k, v). As fases de Shuffle e Merge consistem na troca de dados entre máquinas e agrupamento de valores que possuem a mesma chave, respectivamente. Uma implementação de código aberto bastante conhecida que implementa este modelo é a ferramenta Hadoop.

2. Objetivo

Objetiva-se desenvolver e implementar um modelo de execução para aplicações que seguem uma extensão do modelo de programação Mapreduce, utilizando o padrão MPI para comunicação entre diversas máquinas e *threads* para processamento paralelo intra-nó. Com esta implementação espera-se obter um melhor desempenho na execução das aplicações, comparado a outras ferramentas, como Hadoop, p. ex. Dado que o foco deste trabalho é voltado para o desempenho de aplicações, aspectos envolvendo tolerância a falhas estão fora de escopo.

3. Revisão bibliográfica

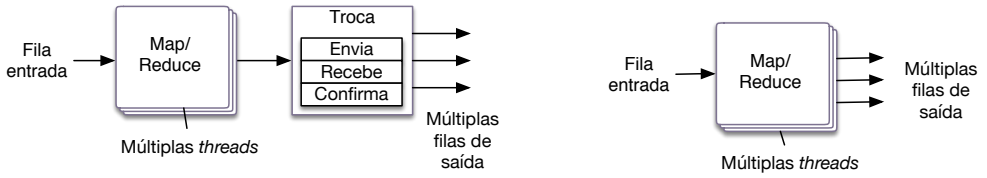
O tema envolvendo Mapreduce e MPI também foi abordado em outros trabalhos. Utilizando um sistema dividido em módulos de gerenciamento e camadas de dados de dois níveis, [Yucheng 2014] obteve resultados com ganho de desempenho de 16 a 24% utilizando MPI, comparado ao Hadoop. Outro trabalho apresenta a utilização de Mapreduce com MPI para indexação distribuída de documentos [Mohamed and Marchand-Maillet 2014]. Sua implementação utiliza uma técnica de *overlapping*, realizando processamento de diversas fases simultaneamente. A principal diferença entre este trabalho e os trabalhos citados consiste na arquitetura da aplicação. Os demais utilizam a arquitetura mestre-escravo, controlando entradas e saídas das diversas fases. Neste trabalho, todas as máquinas executam as mesmas tarefas, de forma paralela e distribuída, com a possibilidade de execução de diferentes operações Map e Reduce no mesmo fluxo de execução.

Trabalhos apontando a utilização de MPI com sistemas de arquivos distribuídos e utilizando vários Maps e Reduces também já foram desenvolvidos [Vasata and Sato 2014, Vasata and Sato 2015].

4. Modelo de execução de aplicações Mapreduce utilizando MPI e *threads*

Uma abordagem híbrida para a execução de aplicações Mapreduce é proposta neste trabalho, de maneira que aplicações sejam executadas em ambiente distribuído e as diversas tarefas de Map e Reduce sejam executadas localmente utilizando *threads*, aproveitando recursos locais de processamento paralelo intra-nó. Diferentes Maps e Reduces podem ser executados, como proposto em [Vasata and Sato 2015].

Com a memória compartilhada é possível que a cópia interna de dados seja reduzida, utilizando manipulação de endereços para a estrutura de dados. Explorando os diversos núcleos de processamento, torna-se possível a execução de diversas operações Map e Reduce em uma mesma máquina simultaneamente.



(a) Com trocas (b) Sem trocas

Figura 1. Blocos de execução com e sem trocas de dados

4.1. Fluxo de execução

O fluxo de execução proposto consiste na execução de estruturas que são instanciadas em todas as máquinas do ambiente distribuído, de forma que estas estruturas possuem variáveis internas e *threads* responsáveis pela execução da aplicação definida. A figura 1 apresenta dois tipos de execução que podem ser utilizados, onde em 1(a) o resultado da operação é transferido a outras máquinas e em 1(b) o resultado é utilizado localmente.

As entradas e saídas dos blocos são compatíveis, permitindo executar blocos em sequência para a realização de tarefas mais complexas, sem a necessidade de manipulação das estruturas de dados. As diversas fases operam seguindo um modelo produtor-consumidor, com bloqueio para a fase de Reduce. Tal bloqueio é necessário para aguardar a finalização da fase de Merge e evitar que o processamento de Reduce seja realizado com dados desatualizados.

4.2. Implementação

O esquema proposto foi implementado utilizando linguagem de programação C, biblioteca MPI para a realização das trocas de dados e Pthreads para a execução paralela das operações Map e Reduce. A estrutura de dados utilizada para os pares (k, v) é realizada de forma que a mesma estrutura seja utilizada para as operações Map e Reduce, permitindo que a execução de diversos blocos seja realizada em sequência.

A comunicação é realizada entre as instâncias do mesmo bloco utilizando funções MPI assíncronas, de forma a não realizar o bloqueio durante o processamento das execuções. Durante o processo de trocas, são utilizadas três *threads*, com diferentes propósitos: enviar, receber e confirmar envios.

4.3. API

Ao programador de aplicações Mapreduce é fornecido a biblioteca `mpi_mr`, com estruturas de dados a serem utilizadas e funções de configuração e controle da execução, além do protótipo de funções que devem serem implementadas.

A figura 2 apresenta algumas das funções que poderão ser utilizadas pelo desenvolvedor, de maneira que a relação de entradas e saídas é definida com endereços para funções (em destaque), que serão chamadas internamente pela aplicação. Também deve ser definido quais serão os destinos dos dados de saída, se os dados serão repassados a uma função que salve resultados em disco ou utilizados como entrada para um próximo bloco de execução. Como as funções de leitura e escrita de dados devem ser implementadas pelo usuário, a utilização do ambiente independe do sistemas de arquivos, ficando a cargo do programador da aplicação Mapreduce sua escolha.

```
1: #include "mpi_mr.h"
2: MpiMrT *map = MpiMrT_new(id_map, mpi_rank, mpi_size, n_threads);
3: MpiMr *red = MpiMr_new(id_red, num_threads);
4: Fila entrada, inter[1], saida[1];
5: MpiMrT_setFuncao(map, Map, param_map, Shuffle, param_shuffle);
6: MpiMrT_setES(map, &fila_entrada,&inter, n_inter);
7: MpiMr_setFuncao(red, Reduce, param_reduce);
8: MpiMr_setES(red, &inter[0], &saidas, n_saidas);
9: MpiMrT_iniciar(mpimr_map); MpiMrT_aguardaFinalizar(mpimr_map);
10: MpiMr_iniciar(mpimr_red); MpiMr_aguardaFinalizar(mpimr_red);
```

Figura 2. Exemplos de funções para aplicações Mapreduce

5. Resultados atuais

Testes preliminares utilizando contagem de palavras foram realizados, mostrando o funcionamento do modelo implementado. Tais testes foram realizados em ambiente com máquinas virtuais, e futuramente serão executados em ambientes com máquinas físicas, com a finalidade de medição de desempenho. Espera-se que a execução de aplicações utilizando a abordagem de paralelismo local e distribuído proporcione melhor desempenho na execução das aplicações, visto que recursos disponíveis localmente poderão ser melhor utilizados.

6. Trabalhos futuros

Apesar da tolerância a falhas não ser abordada neste trabalho, esta pode ser aplicada garantindo que o armazenamento dos dados nas estruturas de dados em árvore exista, utilizando técnicas de replicação. Outro ponto que deve ser levado em consideração são os grupos de comunicação MPI nas máquinas do ambiente distribuído, o que pode ser implementado utilizando bibliotecas com suporte a detecção de falhas, como MPI-ULFM.

Agradecimentos

Agradecimentos à Escola Politécnica da Universidade de São Paulo e ao Instituto Federal do Paraná.

Referências

Dean, J. and Ghemawat, S. (2008). MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM*, 51(1):107–113.

Mohamed, H. and Marchand-Maillet, S. (2014). Distributed media indexing based on MPI and MapReduce. *Multimedia Tools and Applications*, 69(2):513–537.

Vasata, D. and Sato, L. M. (2014). Utilização de MPI com o sistema de arquivos distribuído HDFS. In *Anais do V ERAD-SP*, São Bernardo do Campo.

Vasata, D. and Sato, L. M. (2015). Mapreduce com múltiplos blocos de execução atuando de forma coordenada utilizando entradas e saídas interligadas. In *Anais do VI ERAD-SP*, São José do Rio Preto.

Yucheng, G. (2014). MapReduce Model Implementation on MPI Platform. In *2014 13th International Symposium on Distributed Computing and Applications to Business, Engineering and Science (DCABES)*, pages 88–91.

Sistema Distribuído de Classificação de Imagens aplicado à um Projeto de Ciência Cidadã

Fernanda Beatriz Jordan Rojas Dallaqua[1], Álvaro Luiz Fazenda[1]

[1]Instituto de Ciência e Tecnologia – Universidade Federal de São Paulo (UNIFESP)
Av. Cesare M. G. Lattes, 1201 – São José dos Campos, CEP: 12247-014 – Brazil

fernanda_dallaqua@hotmail.com, alvaro.fazenda@unifesp.br

Abstract. *Recently, scientific projects involving volunteers for analyzing, collecting data and using their own computational resources have been growing due to advances related to information technology and communication resources, which allows a great number of projects in Citizen Science. In this context, the project ForestWatchers [http://www.forestwatchers.net] was launched in 2012 as a Citizen Science joint project developed by UNIFESP, LAC/INPE and Citizen Cyberscience Centre (CCC). ForestWatchers aims to monitor the rainforest by the collaboration of volunteers connected to the Internet around the planet. This work implemented an efficient and scalable Volunteer Computing System that can be integrated into ForestWatchers project. This system allows automatic classification for images by neural network, detecting regions where there was deforestation. To contribute, volunteers need only to be connected remotely through a Web browser and consent to donate their own processing power in computational devices.*

Resumo. *Recentemente, projetos científicos que envolvem voluntários para analisar, coletar dados e ceder recursos computacionais de seus equipamentos pessoais vem ganhando força, impulsionados pelos avanços relacionados a área de tecnologia da informação e comunicação, permitindo o surgimento de uma série de projetos nessa linha, conhecida por Ciência Cidadã. Neste contexto foi lançado em 2012 o projeto ForestWatchers [http://www.forestwatchers.net], um projeto de Ciência Cidadã desenvolvido por pesquisadores da UNIFESP, LAC/INPE e Citizen Cyberscience Centre (CCC), o qual tem como objetivo monitorar as florestas tropicais através da colaboração de voluntários conectados à Internet ao redor de todo o planeta. Neste trabalho foi implementado um sistema de Computação Voluntária eficiente e com escalabilidade que poderá ser integrado ao projeto ForestWatchers. Este sistema permite a classificação de imagens por rede neural, detectando automaticamente regiões onde houve desmatamento. Para contribuir, os voluntários necessitam apenas estar conectados remotamente através de um navegador Web e consentirem em ceder poder de processamento em seus próprios dispositivos computacionais.*

1. Introdução

Florestas tropicais desempenham um papel importante no ecossistema global pois possuem grande diversidade de fauna e flora além de regularem o clima, as chuvas e absorverem grande quantidade de dióxido de carbono. Elas também possuem valor social pois abrigam inúmeros povos indígenas [LUZ et al., 2014].

A cada ano, milhares de hectares de florestas tropicais são desmatados e degradados no mundo todo. Só na Amazônia Legal, no período de agosto de 2014 a julho de 2015, o desmatamento foi de 3.322km², representando um aumento de 63% em relação ao período de agosto de 2013 a julho de 2014. A degradação florestal no período de agosto de 2014 a julho de 2015 totalizou 2.186km², um aumento de 207% em relação ao período anterior [FONSECA et al., 2015]. O termo desmatamento é designado para áreas de floresta que são eliminadas para agricultura, pecuária, reservatórios de água e áreas urbanas. Já degradação diz respeito a áreas que são exploradas para extração de madeira e/ou queimadas.

A tecnologia se mostra uma grande aliada à preservação das florestas tropicais. Imagens de satélites, dados de sensoriamento remoto e algoritmos para classificação de imagens são utilizados para analisar, identificar e quantificar alterações no meio ambiente, sendo possível fiscalizar e controlar o desmatamento e a degradação florestal [LUZ et al., 2014].

O termo Ciência Cidadã (em inglês: *Citizen Science*) é usado para designar atividades e projetos científicos em que há o envolvimento de cidadãos voluntários, individualmente ou em rede, que se dedicam a analisar, coletar ou classificar dados [ARCANJO, 2014]. Esses cidadãos não precisam ter conhecimento científico sobre o assunto tratado nos projetos e não precisam receber treinamento técnico específico. No geral, a motivação dos voluntários é de aprender mais sobre ciência e de ter a chance de participar de uma iniciativa científica ou socialmente relevante [LUZ et al., 2014].

O conceito de Ciência Cidadã é bastante abrangente e engloba projetos que diferem de acordo com o envolvimento do voluntário e a tecnologia utilizada, sendo possível classificá-los em três categorias: Computação Voluntária, Raciocínio Voluntário ou Sensoriamento Voluntário.

Na Computação Voluntária, o voluntário cede poder de processamento ocioso de seu dispositivo para o processamento distribuído de tarefas. O projeto *SETI@home* [http://setiathome.ssl.berkeley.edu/], lançado em maio de 1999 pela Universidade de Berkeley, é o mais conhecido exemplo de projeto nesta categoria. O voluntário participa deste projeto instalando uma aplicação gratuita em seu computador, onde é feita a análise de sinais de rádio captados por telescópios, cujo objetivo é a busca por inteligência extra-terrestre (SETI - *Search for Extraterrestrial Intelligence*). O projeto atingiu grande popularidade, tendo, uma semana após o seu lançamento, 200.000 pessoas inscritas que obtiveram o software necessário ao processamento. Outros exemplos de projetos de Computação Voluntária são: *Prime Grid* [http://www.primegrid.com/], *ClimatePrediction.net* [http://www.climateprediction.net/], *FightMalaria@home* [https://fightmalariaathome.org/], entre outros.

No Raciocínio Voluntário há a participação ativa do colaborador através de sua capacidade cognitiva para analisar dados ou realizar alguma tarefa [ARCANJO, 2014]. Um dos projetos mais bem-sucedidos de Raciocínio Voluntário é o projeto *Galaxy Zoo* [www.galaxyzoo.org], lançado em julho de 2007, onde seus voluntários ajudam a classificar galáxias de acordo com padrões pré-definidos [LUZ et al., 2014]. O *Galaxy Zoo* também obteve grande popularidade, tendo recebido 70.000 classificações por hora apenas um dia após o seu lançamento. Após um ano, mais de 50 milhões de classificações foram recebidas, tendo mais de 150.000 pessoas contribuído com o projeto. Outros projetos de Raciocínio Voluntário são: *FoldIt* [https://fold.it/portal/], *Stardust@home*

[http://stardustathome.ssl.berkeley.edu/], dentre outros.

No Sensoriamento Voluntário os participantes são os responsáveis por coletar os dados ou informações [ARCANJO, 2014]. *Project Noah* [http://www.projectnoah.org/], *Coral Watch* [http://www.coralwatch.org] e *Christmas Bird Count* [http://www.audubon.org/conservation/science/christmas-bird-count] são alguns exemplos de projetos dessa categoria.

O projeto *ForestWatchers* é uma iniciativa de ciência cidadã, lançado em 2012, que visa o monitoramento do desmatamento através da colaboração de voluntários conectados à Internet ao redor de todo o planeta. Atualmente, o projeto conta com três diferentes aplicações de Raciocínio Voluntário: *Best-Tile, Deforestation* e *Correct Classification*.

Neste trabalho teve-se como objeto de aprimoramento a aplicação *Correct Classification*, onde imagens de satélites são avaliadas por um algoritmo de classificação de imagens, criando imagens segmentadas em floresta e não-floresta que depois são revisadas pelos usuários, para avaliação se as designações feitas pelo algoritmo estão corretas ou não [ARCANJO, 2014]. Foram analisadas duas áreas distintas englobadas na Amazônia Legal Brasileira. Entretanto, os objetivos futuros do projeto *ForestWatchers* incluem o monitoramento de diversas outras áreas, tanto no Brasil quanto em outras regiões do globo terrestre, o que poderá ocasionar grande demanda de processamento no atual servidor.

2. Objetivo

O objetivo deste trabalho foi criar um protótipo de sistema de Computação Voluntária, onde o algoritmo para a classificação de imagens da aplicação *Correct Classification* é executado remotamente com eficiência computacional, utilizando os recursos das máquinas dos voluntários, ao invés do processamento central no servidor.

3. Protótipo Desenvolvido

O protótipo foi implementado utilizando as tecnologias HTML5, Web Workers, JavaScript e Node.js, e permite o processamento da RNA utilizando recursos de processamento do voluntário em uma simples interface *web*, diferente de aplicações convencionais de Computação Voluntária, onde é necessária a instalação de softwares específicos.

Parte do protótipo, que demandou a tradução de código legado em C para Javascript, era uma rede neural (RNA) do tipo Multilayer Perceptron (MLP) com backpropagation, capaz de gerar regiões classificadas com diferentes graus de confiança e que posteriormente são avaliadas pelos voluntários.

Além da tradução anteriormente citada, houve a criação de um servidor com tecnologia Node.js, incluindo o framework Sails.js, onde integrou-se a rede neural ao navegador Web.

No protótipo desenvolvido as imagens de sensoriamento remoto são primeiramente carregadas na página do cliente e pré-processadas. Após esta etapa, uma vez obtido o consentimento do cliente em relação à disponibilização dos seus recursos computacionais, o sistema automaticamente abre uma *thread* para execução das tarefas de classificação, através da tecnologia Web Workers, de forma a não atrapalhar a

navegação do voluntário na web através de seu navegador. Esta *thread* recém iniciada tem a função de executar a RNA desenvolvida em JavaScript, o qual demanda poder de processamento numérico na realização da tarefa. Após a classicação, o sistema envia os dados de saída, que se constituem em dados referentes a imagem classificada e de confiança na classificação da RNA, para o servidor. Uma vez concluída todas as etapas aqui descritas, o usuário poderá continuar com a tarefa de correção da imagem, utilizando normalmente o aplicativo *Correct Classification*. A figura 1 demonstra a estrutura de funcionamento do protótipo desenvolvido.

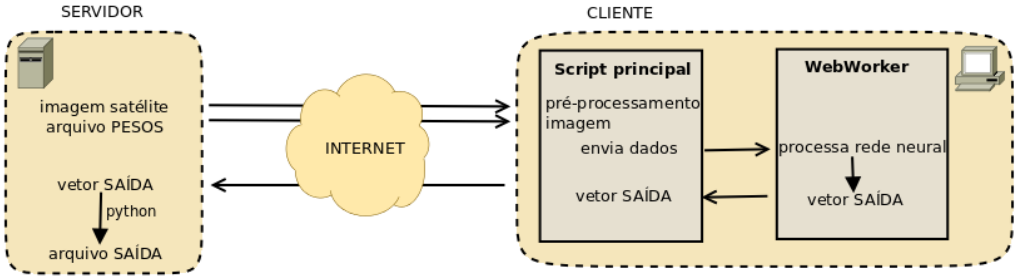


Figura 1. Estrutura de funcionamento do protótipo desenvolvido

4. Conclusão

Neste trabalho, conseguiu-se desenvolver um protótipo que eficientemente classifica imagens de satélite utilizando os recursos computacionais de voluntários remotamente distribuídos, sem a necessidade da instalação de software específico, o que poderia diminuir a sobrecarga no servidor do projeto *ForestWatchers* e possibilitar a expansão do projeto. Esse alívio na carga do servidor do projeto também tornaria possível o desenvolvimento e testes de diferentes classificadores, redes neurais com diferentes pesos ou características, permitindo a busca por algoritmos mais acurados. O uso da tecnologia Web Workers permitiu ainda o desenvolvimento de um algoritmo que demanda intenso poder numérico de processamento, em função da RNA implementada, sem bloqueios na interface de navegação enquanto o processamento está sendo realizado.

Referências bibliográficas

LUZ, E. F. et al. The ForestWatchers: A Citizen Cyberscience Project for Deforestation Monitoring in the Tropics. Human Computation, v. 1, p. 137–145, 2014.

FONSECA, A. et al. Boletim do desmatamento da Amazônia Legal (julho de 2015) SAD. [S.l.], 2015. Disponível em http://imazon.org.br/PDFimazon/Portugues/transparencia_florestal/amazonia_legal/SAD%20Julho%202015.pdf.

ARCANJO, J.d.S. Desenvolvimento e Teste de uma Ferramenta de Avaliação da Qualidade dos Dados e da Confiabilidade dos Voluntários para um projeto de Ciência Cidadã. Dissertação (Mestrado). ICT/UNIFESP. 2014.

A Classic Linear System Solver on Modern Hardware Architecture for Sparse Systems

Nils Urmersbach [1], **Alexandre M. Roma** [1], **Daniel Cordeiro** [2]

[1]Instituto de Matemática e Estatística – Universidade de São Paulo (USP)
São Paulo – SP – Brazil

[2]Escola de Artes, Ciências e Humanidades – Universidade de São Paulo (USP)
São Paulo – SP – Brazil

nils@ime.usp.br, roma@ime.usp.br, daniel.cordeiro@usp.br

Abstract. *In this paper we present our implementations for the Jacobi Method for sparse linear systems in the Compressed Sparse Row (CSR) format using OpenMP, OpenACC and CUDA. Our model problem in this work is derived from the finite difference discretization of the two-dimensional Poisson Equation on rectangular domains.*

1. Introduction

The objective of our work is to implement a simple iterative linear solver for sparse systems of general structure on modern hardware and therefore to exploit modern parallelism schemes, see [Chapman et al. 2008] and [Kirk and Hwu 2013]. To guarantee convergence for our implementation (see Section 2.1) we work with the finite difference discretization of the two-dimensional Poisson equation, an elliptic partial differential equation with applications in electrodynamics, heat transfer, and fluid dynamics.

2. Mathematical Problem

Consider the two-dimensional Poisson equation with Dirichlet boundary conditions

$$\begin{aligned} -(u_{xx}(x,y) + u_{yy}(x,y)) &= 2\pi^2 \sin \pi x \cdot \sin \pi y, \quad x, y \in \Omega = (0, 120) \times (0, 120), \\ u(x,y) &= 0, \quad x, y \in \partial\Omega, \end{aligned}$$

whose exact continuous solution is

$$\hat{u} = \sin(\pi x) \sin(\pi y).$$

The second order derivations are discretized using the centered finite difference schemes

$$\begin{aligned} u_{xx}(x_i, y_j) &\approx \frac{u_{i-1,j} - 2u_{i,j} + u_{i+1,j}}{h_x^2}, \\ u_{yy}(x_i, y_j) &\approx \frac{u_{i,j-1} - 2u_{i,j} + u_{i,j+1}}{h_y^2}, \end{aligned}$$

where the spatial step sizes are defined as

$$h_x = \frac{x_f - x_0}{N+1}, \quad h_y = \frac{y_f - y_0}{M+1},$$

N and M being the number of grid points in the x- and y-direction, respectively, and x_0, y_0 and x_f, y_f are the lowest and highest domain coordinates in the respective direction. For more details on how the linear system of the discrete Poisson equation is set up, see [Isaacson and Keller 1994]. For more details on how sparse matrices are stored in the CSR format, see [Saad 2000] for an example.

2.1. Jacobi Method

Consider the matrix decomposition $A = D - (L + U)$, where D is the diagonal and L and U are the strict lower and upper triangular matrix parts of A. In matrix representation, the Jacobi Method can then be written as

$$G_{Jac} = D^{-1}(L + U) = D^{-1}(D - A) = I - D^{-1}A,$$

so that we have

$$\mathbf{u}^{(k+1)} = G_{Jac}\mathbf{u}^{(k)} + D^{-1}\mathbf{f}_h,$$

G_{Jac} being the Jacobi iteration matrix for matrix A and $\mathbf{f}_h$ being the (discrete) right-hand side vector. Note that the Jacobi method will only converge if the spectral radius $\rho_{Jac} = \max|\lambda(G_{Jac})| < 1$, where λ denotes the eigenvalue.

2.2. Implementations

We considered two implementations using the CSR format and one implementation that does not store a matrix:

- Unmodified Jacobi Method utilizing the CSR format;
- Modified Jacobi Method utilizing the CSR format, and only evaluating the exit criteria every 10 iterations (for the two smallest grids) or 100 iterations (for the other grids), therefore enlarging the parallel region (OpenMP) and diminishing the number of reduction operations which are implicit data transfers from the global to the host memory (OpenACC, CUDA);
- Unmodified Jacobi Method utilizing the five-point-stencil for the discrete laplacian operator in order to evaluate the performance impact of the CSR implementations.

In all three implementations, the NM equations are broken into p (= number of processor cores being used) parts that every processor solves in each iteration using OpenMP SIMD instructions. In the OpenACC/CUDA codes, every worker/thread solves exactly one of the NM equations. Analogously the previous iterative approximation will be updated ($\texttt{u\_old(i)} = \texttt{u(i)}$) either in a chunk (OpenMP) or one by one (OpenACC/CUDA) in parallel. We use reduction operations to determine the infinite norm of the error vector in the same manner. All codes are written in (CUDA-) Fortran95.

3. Results

The OpenMP results were obtained on a Silicon Graphics machine with four AMD Opteron 6376 processors with each 16 cores of 2.3 GHz, and with 8×2 MB L2 cache and 16 MB L3 cache, while the OpenACC and CUDA codes were run on a computer with an Intel Core i7-4770 processor with 3.4 GHz and 8 MB cache, and a NVIDIA Tesla K40

GPU. All four AMD Opteron 6376 processors use the same memory. The Silicon Graphics machine uses Debian 7.0 and gfortran 4.7.2 which supports the OpenMP 3.1 specification, whereas the computer with the Tesla K40 GPU uses Ubuntu 14.04 and pgfortran 15.7-0 which supports all OpenACC 2.0 features. No special flags have been used other than `-mcmodel=large` and `-mcmodel=medium` on the Silicon Graphics and the GPU machine, respectively. For the execution of the OpenMP code we used the CPU affinity "0-63:$\frac{64}{p}$" where p is the number of used processors. The initial approximation is $\mathtt{u\_0} \equiv 0.0$ and the programs iterated until the error $e = \max_{i,j} |u_{i,j} - \sin(\pi i h_x) \sin(\pi j h_y)|$ went below 0.01. All computations are in single precision.

Figure 1 portrays the strong scaling behavior of the CSR implementations including the necessary amount of memory of the respective case to store the previous and current approximation, the right-hand side, and the CSR format. Figure 2 on the other hand shows the strong scaling behavior of the five-point-stencil implementation, considering only the necessary memory needed to store the previous and current approximation, and the right-hand side of the linear system. Both figures are based on the execution time of the respective methods, excluding the time needed to set up the CSR format and the right-hand side of the linear system. Table 1 summarizes all results of this work. All CUDA implementations use blocks of the size (128,1,1) and a grid of the form $(\frac{NM}{128}, 1, 1)$, the gang/vector setup in OpenACC is analogous. No shared memory is used. To emphasize the tremendous effect of parallelism, note that, for example, the serial version of the unmodified Jacobi implementation with $N = M = 8192$ takes roughly 12.25 hours to execute on the Silicon graphics machine. Furthermore, Table 1 reveals the tremendous impact when storing the matrix: Especially the GPU implementations of the five-point-stencil are notably faster (roughly 40%) than the respective CSR implementations, however the latter are more versatile and can solve any sparse system whose Jacobi iteration matrix has a spectral radius $\rho_{Jac} \leq 1$. Furthermore, Table 1 exhibits that there is little difference between the CUDA and OpenACC execution times of these implementations.

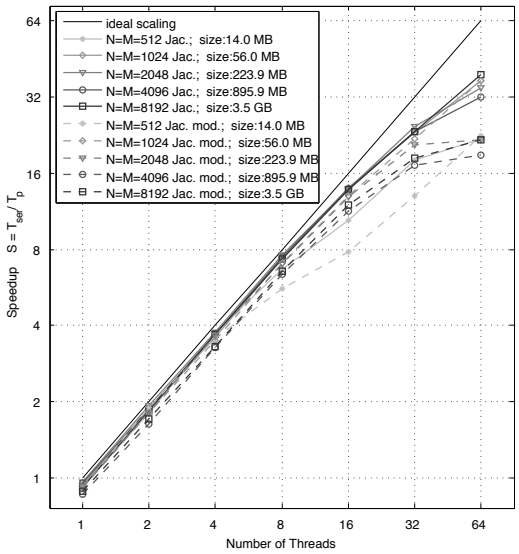


Figure 1. Speedup plot of the Jacobi and modified Jacobi implementations using OpenMP.

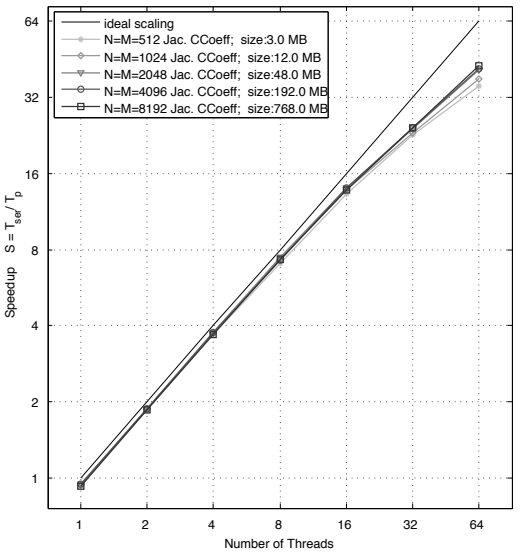


Figure 2. Speedup plot of the Jacobi implementation of the five-point-stencil using OpenMP.

Table 1. Summary of all results. Absolute program execution time in black, relative program execution time in grey (relative to fastest absolute program execution time of the respective implementation).

	N=M=512	N=M=1024	N=M=2048	N=M=4096	N=M=8192
Jacobi					
OpenMP (64 threads)	0.051 s	0.317 s	4.369 s	1 min 14.118 s	20 min 19.251 s
	1.00	1.00	1.48	4.44	5.18
OpenACC	2.005 s	2.074 s	2.976 s	16.689 s	3 min 55.405 s
	39.31	6.54	1.01	1.00	1.00
CUDA	2.024 s	2.071 s	2.952 s	16.975 s	4 min 5.238 s
	39.69	6.53	1.00	1.02	1.04
Jacobi mod.					
OpenMP (64 threads)	0.049 s	0.189 s	2.246 s	56.407 s	14 min 5.732 s
	1.00	1.00	1.00	4.03	4.39
OpenACC	2.014 s	2.086 s	2.949 s	14.286 s	3 min 18.645 s
	41.10	11.04	1.31	1.02	1.03
CUDA	2.027 s	2.062 s	2.849 s	13.984 s	3 min 12.555 s
	41.37	10.91	2.849	1.00	1.00
Jacobi const. coeff.					
OpenMP (64 threads)	0.120 s	0.548 s	3.992 s	54.221 s	14 min 1.558 s
	1.00	1.00	1.67	6.72	7.47
OpenACC	2.014 s	2.043 s	2.385 s	8.073 s	1 min 52.602 s
	16.78	3.73	1.00	1.00	1.00
CUDA	2.022 s	2.059 s	2.469 s	8.958 s	1 min 55.364 s
	16.85	3.76	1.04	1.11	1.02

4. Conclusion

For the three biggest grids used here, the GPU implementations are faster than the OpenMP implementation using 64 cores. This behavior agrees with the hybrid implementation of a geometric multigrid concisely discussed in [Sakharnykh 2016] in which the coarser grid levels (less grid points) run on CPU(s) using OpenMP and the finer grid levels (more grid points) run on the GPU using CUDA. The similar execution times for the discussed GPU codes suggest that OpenACC implementations are preferable, as they are less work-intensive while remaining competitive with CUDA implementations when no optimizations are made to fully exploit the GPU architecture, like using shared memory.

Future works include tiling and domain decomposition for the five-point-stencil implementation, as well as performing a performance/Watt analysis.

5. Acknowledgements

The authors would like to thank the MFLab at UFU and Prof. Goldman from USP for the access to the computers used for this work, as well as CAPES for the financial support.

References

Chapman, B., Jost, G., and van der Pas, R. (2008). *Using OpenMP – Portable Shared Memory Parallel Programming*. MIT Press.

Isaacson, E. and Keller, H. B. (1994). *Analysis of Numerical Methods*. Dover Publications.

Kirk, D. B. and Hwu, W. W. (2013). *Programming Massively Parallel Processors – A Hands-on Approach*. Morgan Kaufman, 2nd edition.

Saad, Y. (2000). *Iterative Methods for Sparse Linear Systems*. SIAM, 2nd edition.

Sakharnykh, N. (2016). High-Performance Geometric Multi-Grid with GPU Acceleration. `https://devblogs.nvidia.com/parallelforall/high-performance-geometric-multi-grid-gpu-acceleration/`. Accessed: 2016-07-21.

Energy aware Heterogeneous OSEP

Cássio H. V. Forte[1], **Aleardo Manacero**[1], **Renata S. Lobato**[1], **Roberta Spolon**[2]

[1]Departamento de Ciências de Computação e Estatística
Universidade Estadual Paulista "Júlio de Mesquita Filho" - UNESP
São José do Rio Preto – SP – Brasil

[2]Departamento de Computação
Universidade Estadual Paulista "Júlio de Mesquita Filho" - UNESP
Bauru – SP – Brasil

cassiohvforte@gmail.com,{aleardo,renata}@ibilce.unesp.br

Abstract. *Advances in the processing power of large datacenters implied in large growth of energy consumption by them. In recent years there is a search for solutions to reduce this consumption, especially in distributed systems, such as computing grids. One option for such systems is to perform the task's allocation to minimize the total power consumption, constrained to performance requirements. In this paper we present a new scheduling algorithm for grid computing, that controls energy consumption, guided by a fairness criteria based in resource ownership.*

Resumo. *Avanços no poder de processamento de grandes centros são acompanhados pelo aumento significativo no consumo de energia deles. Nos últimos anos buscou-se soluções para reduzir esse consumo, principalmente em ambientes distribuídos, como grades computacionais. Uma alternativa para esses sistemas é fazer a alocação das tarefas minimizando o consumo total, sem perda indesejável de desempenho. O objetivo deste trabalho é propor um novo algoritmo de escalonamento para grades computacionais, que controla o consumo de energia a partir de um critério de justiça baseado na posse dos recursos.*

1. Introdução

Nos últimos anos a preocupação com energia atingiu grandes *datacenters*, pois estudos indicam ser eles responsáveis por 3% do consumo mundial de energia elétrica gerando cerca de 2% da emissão global de gases do efeito estufa. Estima-se ainda que o consumo de energia desses sistemas triplique nos próximos 10 anos [Bawden 2016].

Grandes empresas mantêm seus *datacenters* dispersos porém interconectados, formando uma grade computacional. Embora grades desse tipo tenham gerenciamento proprietário, existem ainda as chamadas grades cooperativas, em que os recursos pertencem a diferentes proprietários e estes submetem tarefas para execução a fim de utilizar, quando disponíveis, máquinas de outros usuários e com isso obter melhor desempenho.

Para esse tipo de grade a alocação de tarefas pode ser feita para garantir critérios diversos, entre os quais aparecem com destaque critérios baseados em justiça. Um desses critérios é considerar a propriedade de recursos, buscando garantir que usuários que

ofereçam recursos para compartilhamento possam usar a sua parte sempre que quiserem. Uma proposta nesse sentido foi feita por Falavinha com o algoritmo OSEP (*Owner Share Enforcement Policy*) [Falavinha et al. 2009], restrito a ambientes homogêneos. Forte [Forte 2014] estendeu o algoritmo para grades heterogêneas, dando origem ao algoritmo HOSEP. Neste trabalho se faz uma adaptação do critério de justiça baseada em propriedade para atender restrições de consumo de energia, que é o algoritmo EHOSEP (*Energy aware Heterogeneous OSEP*).

2. Trabalhos relacionados

O tratamento do consumo de energia em sistemas distribuídos tem sido feito de várias formas, muitas operando sobre o escalonamento nesses sistemas, tanto nos níveis macroscópicos quanto microscópicos. Etinski [Etinski et al. 2010] reduz o consumo nas CPUs com a diminuição da velocidade de processamento. Já Liu [Liu et al. 2008] reduz o volume de dados transmitido ou armazenado para poupar energia.

Em outra linha, há algoritmos tratando a infraestrutura tentando, por exemplo, balancear a carga para reduzir o calor gerado, elevar a eficiência e diminuir o consumo da infraestrutura de resfriamento e alimentação dos nós [Patil and Chaudhary 2013] [Chen et al. 2011]. Explora-se ainda condições externas, como variações nas tarifas cobradas pela energia elétrica, para reduzir os custos [Aikema et al. 2011].

3. Formulação do EHOSEP

O algoritmo proposto é baseado nos algoritmos OSEP e HOSEP. O algoritmo OSEP (*Owner Enforcement Policy*) foi proposto para grades de *hardware* homogêneo, a fim de garantir que cada usuário seja atendido por um número de máquinas maior ou igual ao que ofereceu à grade. Em grades heterogêneas trata-se o uso dela considerando o poder computacional de cada recurso, o que é feito no algoritmo HOSEP (*Heterogeneous OSEP*).

Em ambos os algoritmos escalona-se tarefas *bag of tasks*, e a primeira etapa é escolher a tarefa de menor carga na fila do escalonador e que pertença ao usuário com maior defasagem de uso. Enquanto no OSEP a defasagem é dada pela diferença entre o número de máquinas que o usuário ofereceu e o de máquinas que o atende naquele momento, no HOSEP ela é dada pelo diferencial de poder entre máquinas oferecidas e recebidas ($DifPoder_i = \frac{Alocado_i - Porcao_i}{Porcao_i}$).

Caso um usuário tenha $DifPoder$ negativo e não houver máquinas livres é escolhido o recurso de menor desempenho, entre os que atendam ao usuário com maior excesso de poder computacional, podendo ocorrer preempção. Para evitar que preempções causem injustiças maiores que as já existentes, elas ocorrem apenas se a o poder remanescente do usuário que perderá o recurso continuar positivo ou se for maior que o poder remanescente do usuário que receberá o recurso.

3.1. EHOSEP

Os critérios do HOSEP não consideram consumo de energia, que é atualmente um parâmetro importante. Assim, é interessante estender seu conceito com a proposta do EHOSEP (*Energy aware Heterogeneous OSEP*). No EHOSEP, além de se considerar a propriedade que cada usuário tem sobre as máquinas, é considerado também um limite de

consumo, ou seja, o consumo de energia observado na execução das tarefas de um usuário não deve ultrapassar o limite definido por ele.

O limite de consumo de um usuário é dado como porcentagem do consumo total das máquinas que a ele pertencem. Assim, o consumo de energia pode ser visto como parâmetro de nível de serviço contratado. No algoritmo EHOSEP se tenta atender ao critério de justiça do HOSEP enquanto o limite de consumo permitir. Isso leva a modificações no critério de justiça e ao algoritmo de alocação, visto a seguir:

```
1  Ordena em lista os usuários, em ordem crescente de DifPoder. Desempate:
   defasagem absoluta (Alocado_i − Porcao_i) mais negativa;
2  Faça u_i = primeiro usuário da lista;
3  while Não houver escalonado ∧ lista não acabou do
4      if Consumo_i ≤ Limite_i ∧ existe demanda não atendida de u_i then
5          if ∃ conjunto de máquinas livres (M_f, ..., M_g) | ∀M_s ∈
           (M_f, ..., M_g), Consumo_i + Energia_s ≤ Limite_i then
6              Escolhe máquina
               M_h | Desempenho_h/Energia_h = max(Desempenho_f/Energia_f, ..., Desempenho_g/Energia_g);
7              Reserva M_h para a menor tarefa do usuário u_i que estiver na fila;
8              Atualiza dados dos usuários e das tarefas;
9              Encerra algoritmo;
10         else
11             if ∃ usuário u_j | DifPoder_j =
               max(DifPoder_1, ..., DifPoder_m) ∧ DifPoder_j > 0 then
12                 if ∃ Tarefa k de u_j executada em
                   y | [(PR_Preemp_{j,y} ≥ PR_Aloc_{i,y} ∨ (PR_Preemp_{j,y} ≥
                   0 ∧ PR_Aloc_{i,y} ≥ 0)) ∧ (Consumo_i + Energia_y ≤ Limite_i)]
                   then
13                     Realiza preempção da tarefa k do usuário u_j;
14                     Reserva recurso para a menor tarefa, da fila, do usuário u_i;
15                     Atualiza dados dos usuários e das tarefas;
16                     Encerra algoritmo;
17                 end
18             end
19         end
20     end
21     Faça u_i = próximo da lista;
22 end
```

Nesse algoritmo um usuário x, com limite de consumo $Limite_x$ e com consumo atual $Consumo_x$, é atendido com justiça se uma das condições a seguir for satisfeita:

i. estar com poder computacional superior ao oferecido ($DifPoder_x \geq 0$);
ii. se receber nova máquina excederá seu limite de consumo, ou seja, $DifPoder_x < 0 \wedge ((Consumo_x = Limite_x) \vee (\forall$ máquina $y \Rightarrow Energia_y + Consumo_x > Limite_x))$;
iii. se violar o critério de preempção ao receber nova máquina: $DifPoder_x < 0 \wedge (Consumo_x < Limite_x) \wedge (\forall$ usuário $i \mid DifPoder_i > 0$, toda máquina j alocada i é tal que $((PR\_Preemp_{i,j} < PR\_Aloc_{x,j}) \vee ((PR\_Preemp_{i,j} < 0) \wedge (PR\_Aloc_{x,j} < 0))))$;

4. Considerações Finais

O algoritmo EHOSEP, apresentado na última seção, traz uma abordagem de justiça baseada no consumo de energia e, diferente daqueles vistos no trabalhos relacionados, não utiliza as técnicas deles e não procura apenas reduzir o consumo, mas mantê-lo dentro dos limites estabelecidos, enquanto tenta atender ao critério de justiça. O EHOSEP está em fase inicial de testes, utilizando o simulador iSPD [Menezes et al. 2012], por isso ainda não há resultados quantitativos.

Apesar disso, é razoável considerar que a formulação apresentada no algoritmo é matematicamente consistente, pois as garantias de justiça são verificáveis empiricamente. Com isso é possível afirmar que a associação entre os conceitos de porção de propriedade e consumo de energia é viável e deve produzir resultados interessantes para usuários e provedores de recursos em grades.

Referências

Aikema, D., Kiddle, C., and Simmonds, R. (2011). Energy-cost-aware scheduling of HPC workloads. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2011 IEEE International Symposium on a*, pages 1–7. IEEE.

Bawden, T. (2016). Global warming: Data centres to consume three times as much energy in next decade, experts warn. *http://www.independent.co.uk/environment/global-warming-data-centres-to-consume-three-times-as-much-energy-in-next-decade-experts-warn-a6830086.html*.

Chen, H., Song, M., Song, J., Gavrilovska, A., and Schwan, K. (2011). HEaRS: A hierarchical energy-aware resource scheduler for virtualized data centers. In *Cluster Computing (CLUSTER), 2011 IEEE International Conference on*, pages 508–512. IEEE.

Etinski, M., Corbalan, J., Labarta, J., and Valero, M. (2010). BSLD threshold driven power management policy for hpc centers. In *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pages 1–8. IEEE.

Falavinha, J., Manacero, A., Livny, M., and Bradley, D. (2009). The owner share scheduler for a distributed system. In *Intl Conf on Parallel Processing Workshops, ICPPW'09*, pages 298–305.

Forte, C. H. V. (2014). Adaptação do algoritmo osep de escalonamento para sistemas heterogêneos. Monografia de conclusão de curso, disponível em http://www.dcce.ibilce.unesp.br/spd/pubs/mono_Cassio_2014.pdf.

Liu, C., Qin, X., Kulkarni, S., Wang, C., Li, S., Manzanares, A., and Baskiyar, S. (2008). Distributed energy-efficient scheduling for data-intensive applications with deadline constraints on data grids. In *Performance, Computing and Communications Conference, 2008. IPCCC 2008. IEEE International*, pages 26–33. IEEE.

Menezes, D., Manacero, A., Lobato, R., da Silva, D., and Spolon, R. (2012). Scheduler simulation using ispd, an iconic-based computer grid simulator. In *Computers and Communications (ISCC), 2012 IEEE Symposium on*, pages 637–642.

Patil, V. A. and Chaudhary, V. (2013). Rack aware scheduling in HPC data centers: An energy conservation strategy. *Cluster Computing*, 16(3):559–573.

Simulação e Avaliação de Soluções de Software para Arquiteturas com Memórias Não-Voláteis

Mauricio G. Palma, Emilio Francesquini, Rodolfo Azevedo

[1]Instituto de Computação – Universidade Estadual de Campinas (UNICAMP)
Av. Albert Einstein, Nº 1251, CEP 13083-852, Cidade Universitária, Campinas/SP - Brazil

ra180787@students.ic.unicamp.br, {francesquini,rodolfo}@ic.unicamp.br

Abstract. *Recently, new technologies to replace DRAM on current hardware platforms became available in the market. The vast majority of these technologies are non-volatile, i.e., they do not need continuous energy feed to keep their contents intact. However, access to these pieces of hardware is still limited, which makes it difficult to evaluate the performance of software for these new architectures. In this work we propose a simulator capable of evaluating the performance of software systems on these new pieces of hardware. We show preliminary results obtained with a prototype using DRAM and PCM for some of the SPEC2006 benchmarks.*

Resumo. *Novas tecnologias alternativas à memória DRAM começaram a surgir no mercado recentemente. Grande parte destas tecnologias não é volátil, ou seja, não necessitam de alimentação contínua para manter os dados. O acesso a essas tecnologias de hardware ainda é restrito e, portanto, ainda é difícil avaliar o desempenho de soluções de software para estas novas arquiteturas. Neste trabalho propomos um simulador capaz de avaliar o desempenho de sistemas de software quanto a sua utilização nestes novos ambientes de hardware. Apresentamos também alguns resultados preliminares obtidos com a utilização de um protótipo deste simulador comparando o desempenho de aplicações do SPEC2006 utilizando as tecnologias de memória DRAM e PCM.*

1. Introdução

Os dados manipulados por um sistema computacional podem ser classificados de maneira geral em duas categorias. Quando o dado existe apenas durante a execução do programa ele é transiente. Por outro lado, se os dados necessitarem existir por mais tempo, dizemos que ele é persistente. Apesar da manipulação de dados persistentes ser algo recorrente, tradicionalmente as linguagens de programação focam na manipulação de dados transientes, promovendo o acesso aos dados persistentes através de interfaces a sistemas de arquivos ou bancos de dados.

A manipulação dos dados persistentes, entretanto, é normalmente feita em três etapas. Primeiramente eles são copiados para uma área de memória transiente, manipulados e, só então, gravados de volta na área de memória persistente. Esta maneira de trabalhar, apesar de ser a mais comum atualmente, envolve ao menos duas cópias dos dados. Além disto, o programador é forçado a serializar os seus dados para a sua escrita e desserializar para leitura. Algumas linguagens de programação como PS-algol [Atkinson et al. 1983] levaram em conta estes problemas em seus projetos. PS-algol estende a linguagem S-algol com chamadas no estilo chave-valor que automaticamente fazem o mapeamento dos dados transientes para persistentes.

Este mecanismo de manipulação de dados foi moldado a partir da maneira pela qual os sistemas de memória são tradicionalmente construídos. Ou seja, com memória volátil para dados transientes e um dispositivo secundário de armazenamento, tipicamente na forma de um disco rígido ou SSD, para dados persistentes. Recentemente, no entanto, para contornar as crescentes dificuldades enfrentadas no desenvolvimento de memórias DRAM, novas tecnologias de memória começaram a ganhar espaço. Curiosamente grande parte dessas memórias é não volátil, ou seja, podem ser utilizadas tanto para armazenar dados transientes (já que oferecem desempenho próximo à DRAM) quanto para armazenar dados persistentes. Tais tecnologias de memória são conhecidas como *Non-Volatile Memories* (NVMs) [Greengard 2015].

Sistemas computacionais que utilizem NVMs criam novas possibilidades de acesso e manipulação de dados persistentes. Para otimizar o desempenho, por exemplo, pode-se evitar cópias desnecessárias de dados. Alguns trabalhos já tentam antecipar esses novos modelos de programação como o *pmem.io* [Intel 2015]. No entanto, atualmente ainda é difícil obter acesso ao hardware necessário para avaliar experimentalmente tais tecnologias.

Com intuito de antecipar e dar suporte às futuras pesquisas relacionadas à utilização destas novas tecnologias de memória, neste trabalho propomos o desenvolvimento de um simulador. Este simulador será capaz de avaliar o desempenho de soluções que envolvam o uso de memórias de trabalho compostas totalmente ou parcialmente por NVMs. Poderá ser utilizado também para avaliar o desempenho e aplicabilidade de novos mecanismos de manipulação de dados. Apresentamos também alguns resultados preliminares obtidos a partir de um protótipo deste simulador.

2. O Simulador

Simuladores de arquitetura de computadores, assim como manipulação de dados persistentes, são tópicos de pesquisa recorrentes. Diversos simuladores já foram feitos, com diferentes objetivos. Logo, é natural que voltemos nossa atenção às ferramentas já existentes antes de iniciar a construção de um novo simulador. Como base para o nosso simulador escolhemos o ZSIM [Sanchez and Kozyrakis 2013].

O ZSIM é capaz de simular uma máquina com vários processadores que executam instruções da arquitetura *x86* e tem como vantagem um alto desempenho em relação a velocidade da simulação, mesmo quando comparado a outros simuladores modernos. Isto advém do fato do ZSIM ser baseado na ferramenta de instrumentação de arquivos binários PIN [Luk et al. 2005] e executar boa parte do código simulado de maneira nativa. O ZSIM é capaz de manter bons níveis de precisão em seus resultados e, por ser de código aberto, permite a criação de novas instruções de hardware de maneira fácil.

Contudo, o ZSIM não provê um modelo de memória principal tão detalhado, sendo baseado em simples tabelas de latência e velocidade de acesso. Ele tampouco oferece suporte à simulação de NVMs. Para simular memórias não voláteis escolhemos integrar ao ZSIM o simulador de memória NVMain [Poremba and Xie 2012]. O NVMain é capaz de simular diversos tipos de memória, em diferentes configurações. Entre as tecnologias de memória suportadas estão a tradicional DRAM assim como várias tecnologias de NVM tais como PCM e STTM.

Ao unir o ZSIM ao NVMain, pudemos criar a base para um simulador capaz de

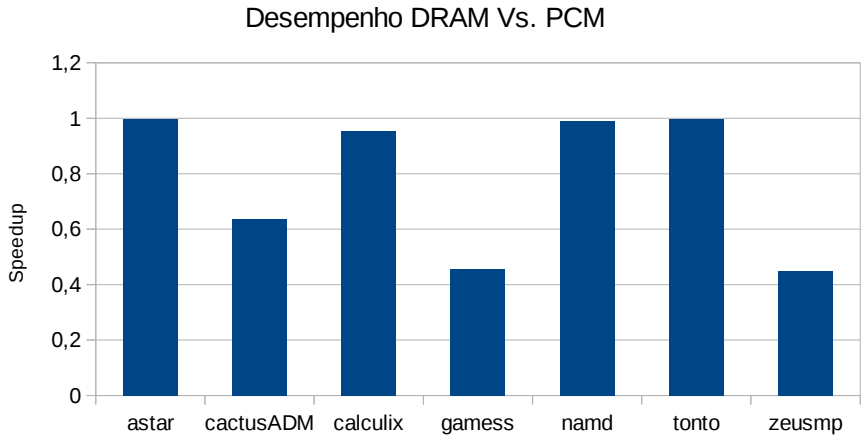


Figura 1. Desempenho de PCM em relação à DRAM

representar uma máquina que tem a memória principal formada totalmente por DRAM ou totalmente por NVMs. Entretanto, em seu estado atual de desenvolvimento, o nosso protótipo ainda não é capaz de simular uma memória híbrida, ou seja, composta por duas tecnologias de memória distintas. Para tanto planejamos alterar a maneira pela qual o ZSIM cria e utiliza controladores de memória, possibilitando o uso de mais de uma tecnologia de memória simultaneamente.

3. Resultados Experimentais

O protótipo desenvolvido já é capaz de simular uma máquina com a memória de trabalho baseada em uma única tecnologia. Desta forma, nesta seção, exploramos duas configurações de memória e o seu desempenho na execução de algumas aplicações do SPEC2006. A primeira configuração utiliza memória volátil baseada em DRAM tradicional, já a segunda utiliza uma NVM com tecnologia Phase-Change Memory (PCM). Os parâmetros para simulação de DRAM correspondem à uma memória DDR3 de 1600 MHz, e os parâmetros para PCM estão de acordo com aqueles descritos por Choi *et al.* [Choi et al. 2012].

A Figura 1 mostra o desempenho ao se utilizar PCM tomando como referência a execução baseada em DRAM para alguns dos benchmarks do SPEC2006. Devido a algumas limitações do protótipo atual, as aplicações foram executadas com as entradas de teste e não com a entrada de referência.

Alguns benchmarks, como *astar* e *namd*, não mostraram muita diferença de desempenho ao utilizar PCM ao invés de DRAM, enquanto outros como o *gamess*, mostraram uma considerável perda de desempenho com a utilização de PCM. Ainda estamos investigando as razões pelas quais alguns benchmarks sofrem maiores impactos que outros. Contudo, análises preliminares indicam como prováveis razões características intrínsecas dos programas (*memory bound/CPU bound*) e o uso da entrada *test* que em alguns casos é pequena suficiente para caber nas *caches* dos processadores.

Outro indicador relevante obtido pela execução dos benchmarks no nosso protótipo é a distribuição dos tempos de acesso à memória. A Figura 2 mostra a média ponderada (pelo número de acessos) do número de ciclos que cada requisição levou. Na figura pode-se constatar que o uso de PCM causa um importante impacto na latência do sistema de memória. Pudemos verificar um aumento de ao menos 3,4X em média na

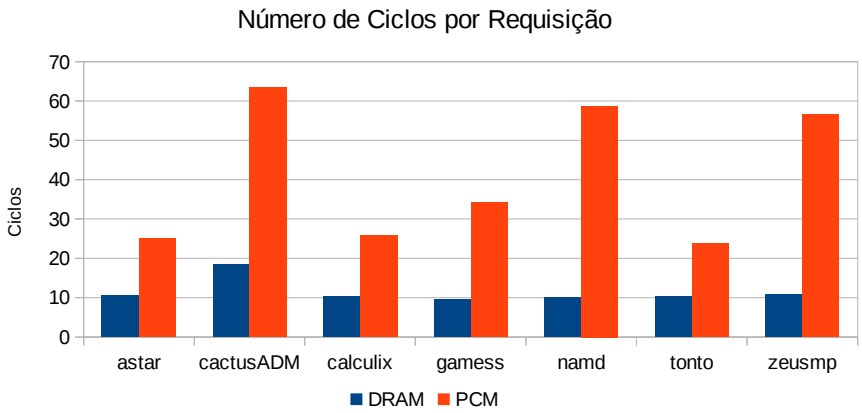


Figura 2. Média ponderada de ciclos por requisição

latência, causando uma potencial perda desempenho.

4. Conclusão

Nossos resultados experimentais corroboraram a nossa expectativa de que a substituição completa da memória DRAM por PCM não traz vantagens em termos de desempenho. No entanto, nós prevemos que as futuras arquiteturas de hardware serão híbridas, com ambas as memórias. Nestas arquiteturas a DRAM pode ser utilizada como uma cache para a PCM otimizando os tempos de acesso à memória, e a PCM como memória de dados persistente de alto desempenho. Essa configuração ainda não é possível no atual estágio de desenvolvimento do simulador, mas já está sendo desenvolvida. Além disto estamos trabalhando na remoção das limitações do protótipo para sermos capazes de executar todos os benchmarks do SPEC2006 com a entrada de referência.

Referências

Atkinson, M. P., Bailey, P. J., Chisholm, K. J., Cockshott, P. W., and Morrison, R. (1983). An approach to persistent programming. *The computer journal*, 26(4):360–365.

Choi, Y., Song, I., Park, M.-H., Chung, H., Chang, S., Cho, B., Kim, J., Oh, Y., Kwon, D., Sunwoo, J., et al. (2012). A 20nm 1.8 v 8gb pram with 40mb/s program bandwidth. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International*, pages 46–48. IEEE.

Greengard, S. (2015). Better memory. *Communications of the ACM*, 59(1):23–25.

Intel (2015). The nvm library. http://pmem.io/.

Luk, C.-K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V. J., and Hazelwood, K. (2005). Pin: Building customized program analysis tools with dynamic instrumentation. *SIGPLAN Not.*, 40(6):190–200.

Poremba, M. and Xie, Y. (2012). Nvmain: An architectural-level main memory simulator for emerging non-volatile memories. In *2012 IEEE Computer Society Annual Symposium on VLSI*, pages 392–397.

Sanchez, D. and Kozyrakis, C. (2013). Zsim: Fast and accurate microarchitectural simulation of thousand-core systems. *SIGARCH Comput. Archit. News*, 41(3):475–486.

Estudo Comparativo de Desempenho e Consumo de Energia em Arquiteturas de Alto Desempenho

Henrique Y. Shishido[1,2], **Leonildo J. M. de Azevedo**[1], **Júlio Cezar Estrella**[1]

[1]Instituto de Ciências Matemáticas e de Computação – Universidade de São Paulo (USP)
Av. Trabalhador São-Carlense, 400 – 13566-590 – São Carlos – SP – Brasil

[2]Departamento de Computação – Universidade Tecnológica Federal do Paraná (UTFPR)
Av. Alberto Carazzai, 1640 – 863000-000 – Cornélio Procópio – PR – Brasil

leonildo.azevedo@usp.br, shishido@utfpr.edu.br, jcezar@icmc.usp.br

Abstract. *The high-performance computing has different architectures such as clusters, multicore processors and GPUs to reduce the execution time of intensive processing applications. However, choosing the architecture for a application is a delicate task. In this sense, the aim of this paper is to present a comparative study of performance and power consumption of high-performance architectures to image processing context. For this we use a multicore processor, cluster and a graphics accelerator for our experiments. The results indicate that the GPU proved to be the best cost/benefit and processing energy consumption.*

Resumo. *A computação de alto desempenho conta com diferentes arquiteturas como clusters, processadores multicore e GPUs para reduzir o tempo de execução de aplicações de processamento intensivo. No entanto, escolher a arquitetura para uma aplicação é uma tarefa delicada. Nesse sentido, o objetivo deste trabalho é apresentar um estudo comparativo de desempenho e consumo de energia entre arquiteturas de alto desempenho ao processamento de imagens. Para isso utilizamos um processador multicore, um cluster e uma aceleradora gráfica para nossos experimentos. Os resultados indicam que a GPU mostrou-se ser o melhor custo/benefício de processamento e consumo de energia.*

1. Introdução

A computação de alto desempenho frequentemente apresenta novas arquiteturas com o objetivo de viabilizar e reduzir o tempo de execução de aplicações que demandam processamento intensivo. Entre essas arquiteturas, destacam-se processadores com múltiplos núcleos (multicore) e as aceleradoras gráficas modernas para processamento de propósito geral (GPGPU). Processadores multicore oferecem desempenho semelhante ao de sistemas multiprocessados, mas com menor custo de comunicação entre os núcleos e menor consumo de energia. As aceleradoras gráficas apresentam um número superior de núcleos e desempenho superior aos processadores multicore. Ambas arquiteturas podem ser incorporadas em clusters de computadores para expandir o poder computacional.

Na literatura diversos trabalhados investigam a elasticidade, eficiência e consumo de energia de aplicações em diversas arquiteturas de alto desempenho. Em estudo realizado entre FPGA (*(Field-Programmable Gate Array*) e GPU aponta que o consumo de energia em FPGA pode ser superior à GPU [Betkaoui et al. 2010]. No trabalho de [Padoin et al. 2013] é mostrado que a carga de trabalho pode afetar drasticamente o consumo de energia em arquiteturas híbridas de CPU e GPU. No estudo da paralelização de um

algoritmo de dinâmica de fluidos para múltiplas GPUs apontou que o consumo de energia da GPU foi consideravelmente inferior aos de CPUs [Jacobsen et al. 2010]. Por outro lado, alguns trabalhos apontam que arquiteturas híbridas CPU-GPU, podem reduzir o consumo de energia em aplicações [Huang et al. 2009].

Algoritmos de processamento de imagens são caracterizados por um volume significativo de processamento que pode durar vários minutos. Um exemplo, é o algoritmo de suavização utilizado para correção de imagens que possuem *pixels* destoantes, causando diferença na qualidade da imagem. Esse algoritmo é comumente utilizado em softwares de edição/reprodução de vídeo e imagem, que buscam a compatibilidade com arquiteturas paralelas para acelerar a manipulação de mídias. No entanto, fica evidente que o desempenho e o consumo de energia são dois fatores relevantes para a escolha de uma arquitetura computacional para este tipo de algoritmo.

Este artigo apresenta um estudo de caso aplicando arquiteturas multicore, GPU (*Graphics Processing Unit*) e aglomerados no processamento de imagens com o intuito de avaliar o desempenho e estimar o consumo de energia de cada arquitetura em um algoritmo de processamento de imagens. As contribuições deste trabalho são: (a) implementações em OpenMP, MPI e CUDA do algoritmo de suavização de imagens; (b) avaliação do desempenho das arquiteturas, e; (c) comparativo do consumo de energia entre as arquiteturas.

2. Métodos

Imagens podem ser divididas de acordo com as cores primárias existentes. Em uma imagem RGB tem-se três canais de cores, em que são representadas as cores vermelha, verde e azul. O algoritmo de suavização tratado neste trabalho usa um operador de convolução que considera a vizinhança de cada pixel da imagem. Este algoritmo assume uma imagem I de um ou mais canais representada pela Equação 1:

$$M_c, c \in \{R, G, B\} \quad (1)$$

onde M_c é a representação matricial de uma de cor primária da imagem. O operador de convolução é aplicado em cada pixel $p_{x,y}$ através de uma máscara N que se estende aos *pixels* vizinhos. Como resultado do operador de convolução é gerado um pixel p', armazenado em uma nova matriz conforme a Equação 2. Cada pixel $p_{x,y}$ é calculado através da média aritmética dos *pixels* presentes na máscara N e atribuído ao posicionamento x, y da nova matriz $M_c^{'}$.

$$M_c'(x, y) = \overline{\sum_{p_i \in N}} \quad (2)$$

Baseado no modelo apresentado foram propostas versões paralelas em OpenMP, CUDA e MPI que consideram que cada elemento de processamento lógico (thread ou processo) execute a convolução nos três canais RGB das posições x, y. Tanto na versão em OpenMP quanto na CUDA, as imagens foram carregadas completamente em memória onde as threads executaram o algoritmo para cada pixel. Ao passo que na versão MPI a imagem foi dividida em X segmentos de acordo com o número de processos. Cada segmento é formado pela altura da imagem original dividida pelo número X de processos, mantendo o número original de colunas. Em caso de sobra na divisão dos segmentos, utilizou-se a abordagem *round-robin* para a distribuição das linhas excedentes para cada

segmento.

3. Resultados e discussões

O código fonte dos algoritmos OpenMP e MPI podem ser obtidos no GitHub[1]. A implementação CUDA está disponível no repositório[2].

Os experimentos foram conduzidos em arquiteturas multicore, cluster e GPU com as respectivas configurações: a) processador AMD Phenom II X6 com seis núcleos, 16GB de memória, Linux Ubuntu Server 14.04 64-bit; b) cluster com 13 nós equipados com processador Intel i7, 32GB de memória, Linux Ubuntu Server 64-bit; c) processador AMD FX-8250, 16GB de memória e aceleradora nVidia Geforce GTX650, 1GB de memória, Linux Ubuntu 15.10 64-bit.

Para avaliar o algoritmo paralelo foi usada uma imagem RGB com resolução de 466 megapixels. O algoritmo foi compilado utilizando o gcc 5.2.1 com flags de otimização -O3. A biblioteca OpenMP foi adotada para a paralelização do algoritmo. A manipulação de imagens adotou a biblioteca OpenCV 2.4.11 para a abertura da imagem e leitura dos *pixels*. A biblioteca OpenMPI 1.8 foi utilizada para a compilação no cluster. E, a biblioteca CUDA 7.5 foi empregada para a compilação do código na versão em GPU.

As execuções seguiram um protocolo de dez repetições para cada versão paralela em que foram retiradas as médias dos tempos de execução. Na versão OpenMP foram realizadas execuções com 1 a 6 threads. Na versão em CUDA as execuções utilizaram o máximo de threads por bloco (32 threads) disponíveis na placa gráfica utilizada. Por fim, na versão MPI foram utilizados os 12 nós físicos variando de 4 a 48 processos.

Para fins de comparação de desempenho e consumo de energia, foi escolhido o menor tempo obtido em cada arquitetura, os quais são representados na Figura 1. O menor tempo da arquitetura multicore foi de 323 segundos, usando todos os seis núcleos do processador, consumindo 125 watts. Nos experimentos com o cluster, o menor tempo foi alcançado utilizando todos os núcleos dos 13 nós, com a marca de 121s e expressivos 1690 watts de consumo de energia. Por outro lado, a arquitetura GPU executou o mesmo volume de instruções com o menor tempo (112 segundos) e, sobretudo, usando menos energia (110 watts) do que as arquiteturas multicore e cluster.

Do ponto de vista comparativo entre arquiteturas, o desempenho atingido pela arquitetura multicore é razoavelmente suficiente para casos em que a resolução da imagem é pequena. Porém, em resoluções maiores, o tempo de processamento pode se tornar inviável frente às restrições de tempo da aplicação. O desempenho obtido com cluster foi maior do que na arquitetura multicore. Porém, o tempo de execução não foi reduzido proporcionalmente ao número de nós utilizados, devido ao significativo gargalo de comunicação entre os mesmos.

A arquitetura GPU obteve o melhor desempenho em relação as duas arquiteturas, devido a compatibilidade do algoritmo com a original função de um acelerador gráfico. Fica evidente que a energia consumida pela GPU é menor em um processador multicore e significativamente aos treze processadores do cluster. Embora os resultados indicam que a arquitetura GPU é adequada para o processamento de imagem, é importante observar que aplicações que não possuem a característica de executar instruções sobre múltiplos

[1] https://github.com/Leonild/smoothMPI.git

[2] https://github.com/rickshido/smoothCUDA.git

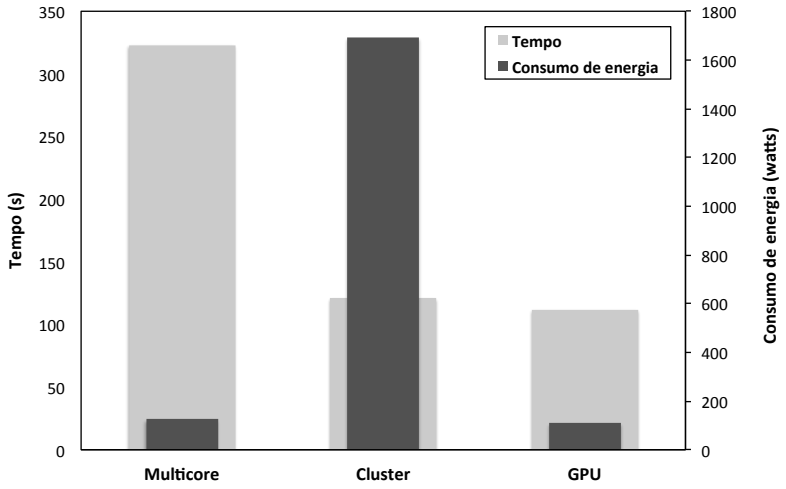


Figura 1. Comparativo dos tempos de execução em relação ao consumo de energia das arquiteturas

dados vetoriais e que possuem diversos desvios condicionais possivelmente não obterão desempenho satisfatório.

4. Conclusão

Neste trabalho foi proposto um estudo comparativo entre as arquiteturas multicore, cluster e GPU por meio da execução de um algoritmo de suavização de imagens. A contribuição foi mostrar por meio de um comparativo a variação do desempenho e do consumo de energia entre estas arquiteturas restrito ao processamento de imagens. Como complemento, os presente estudo destaca que a aceleradora gráfica mostrou-se superior às demais arquiteturas para o processamento de imagens, com desempenho similar a um cluster e consumo de energia próximo a de um processador. No entanto, há situações em que aceleradoras gráficas não conseguem obter um desempenho satisfatório. Como trabalhos futuros, a implementação de outros problemas que envolvam computação intensiva sem a presença de estruturas vetoriais ou matriciais será realizada.

Referências

Betkaoui, B., Thomas, D. B., and Luk, W. (2010). Comparing performance and energy efficiency of fpgas and gpus for high productivity computing. In *Field-Programmable Technology (FPT), 2010 International Conference on*, pages 94–101.

Huang, S., Xiao, S., and Feng, W.-c. (2009). On the energy efficiency of graphics processing units for scientific computing. In *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–8. IEEE.

Jacobsen, D. A., Thibault, J. C., and Senocak, I. (2010). An mpi-cuda implementation for massively parallel incompressible flow computations on multi-gpu clusters. In *48th AIAA aerospace sciences meeting and exhibit*, volume 16, page 2.

Padoin, E. L., Pilla, L. L., Boito, F. Z., Kassick, R. V., Velho, P., and Navaux, P. O. A. (2013). Evaluating application performance and energy consumption on hybrid cpu+gpu architecture. *Cluster Computing*, 16(3):511–525.

Avaliação de desempenho de virtualização: arquiteturas Intel Xeon e IBM Power8

Marcelo Claudio S. Araújo[1], **Luiz Fernando Bittencourt**[1]

[1]Instituto de Computação – Universidade Estadual de Campinas (UNICAMP)
Caixa Postal 6176 – 13.081-970 – Campinas – SP – Brasil

Abstract. *With the popularity of virtualization, research to optimize virtualized environments are of great importance. The Power architecture is an alternative to the x86 architecture to optimize virtualization-based environments. By running benchmarks on a cloud computing infrastructure, this research seeks to analyze the behavior of Power and Intel Xeon architectures as they are overloaded.*

Resumo. *Com a popularização da virtualização, pesquisas para otimização de ambientes virtualizados são de grande importância. A arquitetura Power surge como alternativa à arquitetura x86 para otimização de ambientes baseados em virtualização. Através da execução de benchmarks em uma infraestrutura de computação em nuvens, esta pesquisa busca analisar o comportamentos das arquiteturas Power e Intel Xeon a medida que estas são sobrecarregadas.*

1. Introdução

Nas últimas décadas a capacidade computacional cresceu de forma significativa e as tecnologias de virtualização têm ganhado importância. A virtualização de máquina pode ser definida como um processo de divisão de recursos computacionais a fim de prover um ambiente para execução isolada de um ou mais sistemas operacionais [Chiueh and Brook 2005].

Dois benefícios da virtualização são o compartilhamento de recursos e o isolamento lógico [Sahoo et al. 2010]. O compartilhamento de recursos permite maximizar a utilização do hardware, enquanto o isolamento lógico possibilita esse compartilhamento. Entretanto, devido à falta de garantia de isolamento de desempenho, é importante estudar e entender de quais formas o compartilhamento tem impacto nas aplicações que executam sobre um ambiente virtualizado.

A virtualização obteve uma rápida popularização, impulsionada por servir de base fundamental para a computação em nuvens. Apesar da grande popularidade da plataforma x86, a arquitetura não foi originalmente projetada considerando o conceito de virtualização. Devido à importância da virtualização, outras empresas iniciaram desenvolvimentos de novas soluções. Recentemente, a IBM reestruturou a organização da arquitetura Power, em 2014 empresa lançou a sua oitava versão (Power 8). A plataforma faz uso avançado de *multithreads* simultâneas (alcançando até oito), utiliza memória transacional e aprimoramentos de acesso à memória.

Portanto, dado o protagonismo da virtualização na computação em nuvem, e da computação em nuvem na execução de cargas de trabalho heterogêneas e variáveis, esta pesquisa busca analisar o comportamento das plataformas Intel x86 e Power 8 em cenários de execução concorrente de VMs.

2. Metodologia

A avaliação das arquiteturas foi realizada através da execução de benchmarks em dois servidores da arquitetura Intel Xeon e um servidor Power (Tabela 1). O ambiente de testes foi criado para simular uma nuvem computacional, ou seja, um ambiente onde usuários podem compartilhar um mesmo servidor, onde o OpenStack [Sefraoui et al. 2012] foi utilizado para gerenciar a infra-estrutura.

Tabela 1. Servidores e suas respectivas configurações.

	Processador	Núcleos	*Threads* por núcleo	Memória	Armazenamento	*Clock*
Intel E3	Xeon E3-1240 v2	4	2	16 GB	1 TB	3.4 Ghz
Intel E5	Xeon E5-2620 v3	8	2	16 GB	1 TB	2.4 Ghz
Power 8	Power 8	10	8	128 GB	3,5 TB	3.4 Ghz

O processo de avaliação observou a capacidade de processamento computacional e a largura de banda de memória do servidores. Para medir o desempenho de processamento, foi escolhido o benchmark SPECint CPU 2006. O benchmark STREAM [McCalpin 1995] foi o selecionado para realizar a medição de largura de banda de memória dos servidores. O STREAM realiza as seguintes operações em vetores: cópia (*copy*), expansão de tamanho (*scale*), soma (*add*) e *scale* seguida por uma soma (*triad*).

Para a realização dos testes, máquinas virtuais (VMs) foram alocadas gradativamente (iniciando com uma VM) até uma quantidade máxima M estabelecida, sendo M o número máximo de *threads* do servidor. Para cada quantidade n de VMs, os benchmarks foram executados de forma concorrente. Por suportar uma quantidade de *threads* muito superior aos servidores Intel, o número de núcleos habilitados no servidor Power foi reduzido para dois. A redução de núcleos buscou equiparar a quantidade de máquinas alocadas entre os servidores. O valor de M para o servidor Intel E3 foi de 8, enquanto que para o Intel E5 e Power 8 foi de 16 máquinas virtuais.

Devido à diferença absoluta de desempenho entre as máquinas, consideramos a porcentagem de degradação (PdD) de desempenho dos servidores como métrica de avaliação. A degradação de desempenho reflete a porcentagem com que as máquinas vão ganhando ou perdendo desempenho em relação à sua execução inicial, i.e., com apenas uma VM.

3. Resultados e Discussão

Em relação ao benchmark SPEC, os servidores Intel obtiveram menor degradação de desempenho em comparação com o servidor Power 8. Apesar dos servidores Intel apresentarem desempenho semelhante na execução com uma VM, o E5 obteve nível de degradação inferior à medida que a sobrecarga do sistema aumenta.

A degradação alcançada pelo servidor E3 com sete VMs é equivalente ao E5 com onze instâncias, como pode ser observado na Figura 1. Foi possível observar um pico na degradação dos processadores no momento em que o número de VMs ultrapassa a quantidade de núcleos. Para o servidor Power 8 com 2 núcleos habilitados, em alguns benchmarks (como por exemplo o gcc) a degradação chegou a 1000% com quize VMs. Já com 4 núcleos habilitados, foi possível obter uma melhora no desempenho geral da Power 8. Como esperado, no momento que a quantidade de VMS excedeu o número de núcleos ocorreu um aumento substancial da degradação.

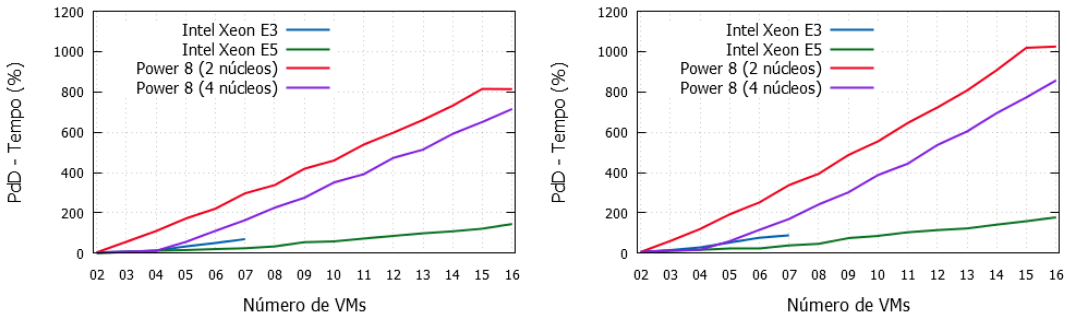


Figura 1. Benchmarks perlbench (esquerda) e gcc (direita) do pacote SPEC.

Após investigar peculiaridades do servidor Power, foi encontrada uma possível justificativa para o comportamento observado. A implementação do *hypervisor* KVM não faz uso completo do SMT com oito *threads*. Atualmente quando vCPUs adicionais ao número de núcleos são alocadas, estas são escalonadas, gerando *overhead*, ou seja, não operam de forma "paralela". Portanto, o desempenho na execução concorrente de VMs na arquitetura Power poderá ser melhorado com o advento de uma implementação adequada do *hypervisor*.

Nas operações realizadas pelo STREAM, a taxa de transferência inicial do servidor E3 supera a do E5 em média em 3.000 MB/s. Nas operações de *copy* e *scale* os servidores Power 8 e E3 possuem valores de taxa de transferência semelhantes. Entretanto, nas operações de *add* e *triad*, o servidor Power, na execução inicial, superou a taxa de transferência do E3 em 2.000 MB/s e 4.000 MB/s, respectivamente.

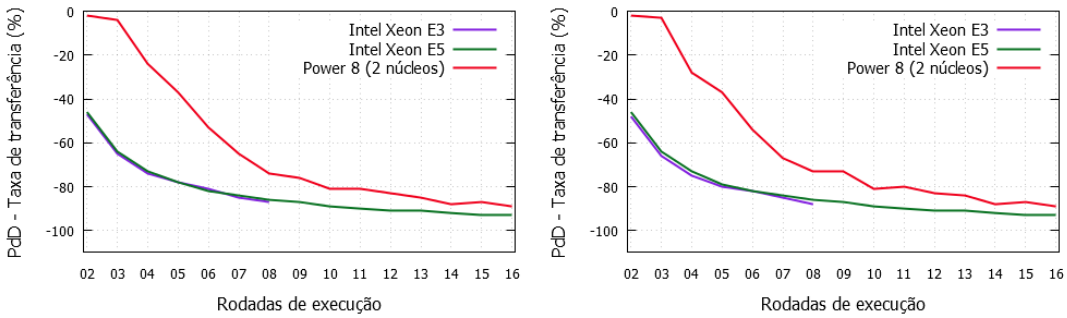


Figura 2. Operação ADD (esquerda) e TRIAD (direita) do benchmark STREAM.

Os resultados das operações *add* e *triad* são apresentados na Figura 2, o comportamento é mantido para as outras operações. Analisando o gráfico, é possível observar que os servidores Intel degradam em cerca de 45% na execução inicial. O servidor Power alcança esse nível de degradação apenas na execução com seis VMS. Entretanto, à medida que os servidores sofrem sobrecarga, os níveis de degradação ficam mais próximos.

O melhor comportamento do servidor Power na avaliação de largura da banda da memória é a utilização de *chips* com a tecnologia Centaur. Os controladores de memória são associados aos *chips* Centaur, funcionando como um *buffer* de memória e até mesmo como uma cache L4. Os *chips* são conectados por três links (dois de leitura e um de

escrita) com velocidade de 9,6 GB/s cada.

A comparação entre a porcentagem de degradação de desempenho e a porcentagem de utilização de recursos computacionais é importante para servir como base para decisões de alocação de VMs em máquinas físicas, de forma a obter desempenho satisfatório para os usuários de uma nuvem computacional e suas aplicações. Através da análise destes resultados preliminares, foi possível observar que o servidor E5 com dez máquinas possui o mesmo nível de degradação que o E3 com sete VMs, utilizando uma menor porcentagem de recursos. Portanto, desconsiderando fatores como custo e consumo de energia, o servidor E5 forneceria um melhor relação entre desempenho por quantidade de recursos.

Com os resultados do benchmark STREAM, foi possível concluir que a Power 8 em comparação com o Intel E5 pode executar com o mesmo nível de degradação e com o dobro de VMs. Portanto, considerando a largura de banda de memória, a Power 8 surge como uma melhor escolha para criação de uma nuvem computacional.

4. Conclusões

Este estudo apresenta resultados preliminares e, portanto, ainda é necessária a realização de mais testes para que só então seja possível estabelecer conclusões robustas sobre o comportamento das arquiteturas, especialmente em cenários com maior quantidade de VMs e núcleos disponíveis. Uma vez que seja possível determinar qual arquitetura possui um melhor comportamento em determinado cenário, um escalonador para uma nuvem híbrida pode ser desenvolvido, com o objetivo de aproveitar o melhor que cada arquitetura tem a oferecer.

5. Agradecimentos

Agradecemos à IBM pela concessão da bolsa de mestrado e pela presteza que sempre demonstrou quando solicitado auxílio durante o desenvolvimento da pesquisa.

Este trabalho recebeu financiamento do *European Union's Horizon 2020 for research, technological development, and demonstration, grant agreement no.* 688941 (FUTEBOL), e também do Ministério da Ciência, Tecnologia e Inovação através da RNP e CTIC.

Referências

Chiueh, S. N. T.-c. and Brook, S. (2005). A survey on virtualization technologies. *RPE Report*, pages 1–42.

McCalpin, J. D. (1995). Memory bandwidth and machine balance in current high performance computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, pages 19–25.

Sahoo, J., Mohapatra, S., and Lath, R. (2010). Virtualization: A survey on concepts, taxonomy and associated security issues. In *Computer and Network Technology (ICCNT), 2010 Second International Conference on*, pages 222–226. IEEE.

Sefraoui, O., Aissaoui, M., and Eleuldj, M. (2012). Openstack: toward an open-source solution for cloud computing. *International Journal of Computer Applications*, 55(3):38–42.

Programação Paralela Unificada para Ambientes Heterogêneos compostos de Multiprocessadores e Coprocessadores

Denilson Souza Bélo, Liria Matsumoto Sato

[1]Departamento de Engenharia de Computação e Sistemas Digitais (PCS)
Escola Politécnica da Universidade de São Paulo (EPUSP)
Av. Professor Luciano Gualberto, tv 3, nº 158 – São Paulo – SP – Brasil

denilsonbelo@gmail.com, liria.sato@poli.usp.br

Abstract. *With the use of hardware accelerators to increase capacity in high-performance computing, there is a need for forms of hybrid programming that can, effectively, make use of all resources. This paper presents a method to perform tasks in heterogeneous architectures composed by processors and co-processors in distributed computing environment.*

Resumo. *Com a utilização de aceleradores de hardware para aumentar a capacidade em Computação de Alto Desempenho, há a necessidade de formas de programação híbrida que possa, de maneira eficaz, fazer uso de todos os recursos. Este trabalho apresenta um método para execução de tarefas em arquiteturas heterogêneas compostas por processadores e coprocessadores em ambiente de computação distribuída.*

1. Introdução

Técnicas utilizadas em arquiteturas paralelas, tais como, GPGPU (*General Propose Graphic Processor Unit*) [Kirk e Hwu 2010] e MIC (*Many Integrated Core*) [Jeffers e Reinders 2013] e multicores de CPUs, auxiliam o processamento de aplicações que necessitam de alto poder computacional. Nos últimos anos, houve um crescente interesse entre os pesquisadores e da industria em criar técnicas de programação que auxiliem o desenvolvedor a obter melhor desempenho com o mínimo esforço. Normalmente essa programação é feita usando um modelo de programação para cada arquitetura separadamente ou, em alguns casos, par a par como Processadores + Coprocessadores. Também existem propostas para adquirir melhor ganho com as três tecnologias. Computadores que se encontram na lista dos mais rápidos do mundo [Top500 2016], como o Tianhe-2, utilizam desses recursos para melhorar o desempenho e, atualmente, há computadores que fazem uso das três arquiteturas [LNCC 2016].

A utilização desses recursos heterogêneos requer uma adaptação das técnicas de programação existentes para cada tecnologia. Há tentativas de "generalização" nesse desenvolvimento em ambientes onde mais de uma arquitetura esteja disponível. Algumas soluções apresentam técnicas de programação separadamente para cada arquitetura. Outras tentam adaptar técnicas utilizadas em uma arquitetura à outra.

Este trabalho apresenta um método unificado e simples de programação híbrida, através de chamadas de funções de uma biblioteca, denominada nesta proposta como

RTElib – Remote Task Execute Library, para obter melhores ganhos no desenvolvimento e um melhor desempenho na execução das aplicações em paralelo. Encontra-se na fase de especificação e de testes de implementação de conceitos que serão aplicados. Este artigo está dividido da seguinte forma: seção 2 apresenta os trabalhos relacionados; seção 3 apresenta o método proposto para integração entre arquiteturas heterogêneas e, por último, na seção 4 é apresentada as considerações finais.

2. Trabalhos Relacionados

A busca por uma solução híbrida para execução em arquiteturas tão diversificadas torna-se um desafio à ciência da computação. As técnicas de unificação mostram pouca ou nenhuma eficiência, pois há uma tentativa de generalizar a execução. Cada recurso de processamento tem suas vantagens. A ideia seria explorá-las para conseguir um melhor desempenho.

Em estudos envolvendo biomedicina, em especial na área da genética, a utilização de sistemas heterogêneos para obter um melhor desempenho é bem explorado. O Escalonador de classificação de interação SNP-SNP oferece uma detecção rápida de interações entre SNPs (*Single Nucleotide Polymorphism*) [Sluga et al. 2014]. Este escalonador, escrito em Python, distribui as tarefas para todos os recursos disponíveis especificados pelo usuário como CPUs, GPGPUs e MIC. Gera um arquivo de resultado final com todos os resultados das unidades individuais. Cada *thread* recebe um par de SNPs e realiza todos os cálculos necessários para calcular a pontuação de união do par. Os resultados experimentais apresentados em [Sluga et al. 2014] demonstram uma perda de desempenho no MIC comparada a execução em ambientes com GPGPUs. Já o desenvolvimento para MIC é muito mais simples, reduzindo o esforço de programação.

Para a execução dinâmica de aplicações orientadas a dados, [Yarkhan 2012] apresenta um ambiente em tempo de execução escalável e de um desempenho competitivo, denominado QUARK (*QUeuing And Runtime for Kernels*), como uma solução de programação para computadores multicore ou sistemas com arquiteturas *multicore* e memória distribuída. Este ambiente oferece uma interface de programação simples para a implementação de um modelo de programação dirigido a algoritmos de álgebra linear. Tais algoritmos são implementados como tarefas que atuam sobre blocos de dados e possuem dependências de dados entre si. O processamento paralelo oferecido é semelhante à execução *dataflow*.

Já a proposta aqui apresentada é voltada para paralelismo de tarefas para computadores com arquiteturas heterogêneas compostos por processadores e coprocessadores, como também, ambientes de computação distribuída.

3. Abordagem Adotada

Para obter uma melhor eficiência na execução de tarefas paralelas, em computadores com processadores e coprocessadores ou sistemas distribuídos, foram especificados um modelo de programação e a arquitetura RTE.

3.1. Modelo de Programação

A execução de tarefas pode ser atribuída a recursos de processamento remoto, como coprocessadores ou nós de um sistema distribuído. A Figura 1 apresenta o modelo de programação.

```
#include <stdio.h>
#include <RTElib.h> //Header Library of RTE

int main(int argc, char *argv[])
{
        . . .

        task_call(recurso, nome_func, argv_func, retorno);
        . . .

        RTE_sync(nome_func);

        return 0;
}
```

Figura 1. Modelo de Programação.

A função **task_call**, disponibilizada na biblioteca **RETlib**, atribui a execução de uma função para um recurso remoto. No momento em que o resultado gerado pela função chamada for utilizado, deve-se chamar a função **RTE_sync** para garantir que a execução da função chamada foi finalizada.

3.2. A Arquitetura RTE – Remote Task Execute

A Arquitetura RTE, (Figura 2), é composta por uma máquina remota **REMOTE_COMPTER** e uma máquina principal **HOST**. **REMOTE_COMPTER** contém um *daemon*, **receiver_daemon**. **HOST** contém dois *daemons*: **sender** e **receiver**. Uma requisição de execução de uma função em **HOST** pelo **user_program** por um recurso remoto é realizada pela chamada da função **task_call**. Esta função cria uma *thread* que é responsável pela execução da função. A *thread* criada informa a requisição ao *daemon* **sender**. Este envia a requisição ao *daemon* **receiver_daemon** em **REMOTE_COMPTER**.

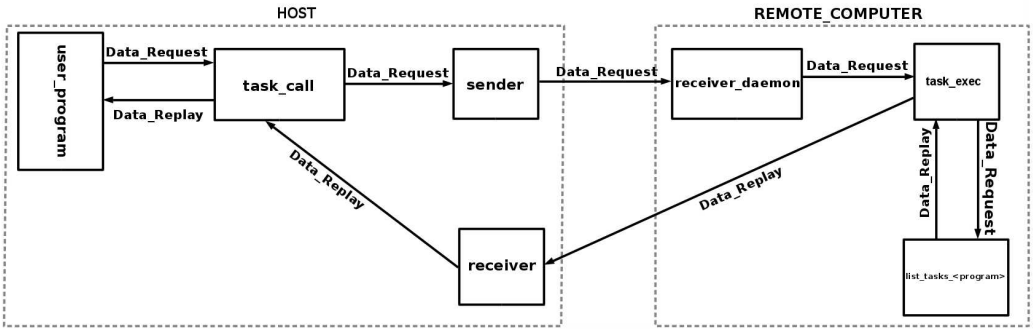


Figura 2. Arquitetura RTE – Remote Task Execute.

Após receber a requisição, o **receiver_daemon** cria uma *thread* **task_exec** que é responsável pela execução da função. Esta coloca em execução um programa

list_tasks_<program> , correspondente ao programa do usuário. Este programa executa a chamada da função solicitada e transfere os dados de retorno e informa o estado da função para **task_exec**. Esta envia os dados de retorno para o *daemon* **receiver** em HOST, o qual retorna-os para a *thread* responsável pelo acompanhamento da execução da função.

Segue uma descrição detalhada dos itens da arquitetura RTE:

HOST: Máquina responsável pela chamada da execução da tarefa remota;
- **user_program:** Programa do usuário responsável pela chamada, através da **task_call**;
- **task_call:** Cria uma *thread* para a execução da tarefa chamada;
- **sender:** *Thread* responsável pelo envio dos atributos para executar a tarefa remotamente no maquina **REMOTE_COMPTER**;
- **receiver:** *Thread* responsável pela recepção dos dados executados remotamente;

REMOTE_COMPTER: Computador remoto onde o receiver_daemon aguarda a requisição para a execução de uma tarefa.
- **receiver_daemon:** Cria uma *thread* para a execução da tarefa chamada;
- **task_exec:** *Thread* responsável pelo envio dos atributos para executar a tarefa em **list_tasks_<program>**;
- **list_tasks_<program>:** programa que contém a lista de tarefas do **user_program** e a executa através do identificador encaminhada pelo **task_exec** e retorna o estado da execução e os dados.

4. Conclusão

O método RTE, apresentado neste artigo, busca integrar arquiteturas heterogêneas compostas de computadores com processadores e coprocessadores ou sistemas distribuídos. A solução proposta oferece uma forma simples e familiar de programação. A implementação desta solução encontra-se em andamento.

Referências

Jeffers, J. e Reinders, J. (2014). *Intel Xeon PHI Coprocessor High Performance Programming.* Morgan Kaufmann. 1-5 p.

Kirk, D. e Wei-Mei, W. (2010). *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann. 280 p.

LNCC. (2016). *A Máquina*. Disponível em: <http://sdumont.lncc.br/machine.php?pg=machine#>.

Sluga, D. et al. (2014). *Heterogemeos computing architecture for fast detection of SNP-SNP interection.* BMC bioinfirmatics, v. 5. Disponível em: <http://www.biomedcentral.com/1471-2105/15/216>.

Top500. (2016). *TOP500 Supercomputing Sites*. Disponível em: <http://www.top500.org/lists/2016/06/>.

Yarkhan, A. *Dynamic Task Execution on Shared and Distributed Memory Architectures.* Tese (Dissertation for the Doctor of Philosophy) — University of Tennessee. Disponível em: <http://trace.tennessee.edu/utk_graddiss/1575>.

Arquitetura para Execução Transacional em Workflows com Domínio Distinto

Fernando Ryoji Kakugawa, Liria Matsumoto Sato

Departamento de Engenharia de Computação e Sistemas Digitais
Escola Politécnica da Universidade de São Paulo
Av. Prof. Luciano Gualberto, tv 3, n 158 – São Paulo – Brasil

fernando.kakugawa@usp.br, liria.sato@poli.usp.br

Resumo. *A utilização de web services tem proporcionado novas possibilidades de desenvolvimento de software, entre elas a composição de serviços, apresentando novas questões como a execução de maneira integral garantido a consistência e o controle de concorrência. A utilização de workflow tem se mostrado adequada para superar esses obstáculos, porém workflows concebidos em domínio distinto faz com que os serviços que sejam comuns não possuam ciência do contexto da execução global, gerando atendimentos que não sejam justos ocasionando deadlock e starvation. Este trabalho apresenta uma arquitetura para execução de worklfows com origem em domínio distinto, sem um coordenador central, mantendo o baixo acoplamento de web services, contemplando o atendimento às requisições de maneira justa, livre de deadlock e starvation.*

1. Introdução

Web services possuem características que os tornam atraentes no desenvolvimento de software, tornando-se infraestrutura básica para áreas como a computação em grade e computação em nuvem. O baixo acoplamento minimiza a dependência entre serviços permitindo autonomia para que sejam concebidos de maneira independente. Com a utilização de *web services* surgem novas aplicações, as quais são constituídas de composição de serviços, agilizando o processo de desenvolvimento, realizando o reuso de rotinas desenvolvidas e disponibilizadas através de um serviço.

Porém, a utilização de composição de serviços concebidos de maneira independente apresenta novas questões para sua execução, entre elas o comportamento transacional de maneira que garanta a consistência de dados, o controle de concorrência através de atendimento justo para as requisições, e execuções que sejam livres de *deadlock* e *starvation*. A utilização de *workflows* para executar composição de serviços tem se mostrado adequado permitindo consistência, possibilitando comportamento transacional. Mas, *workflows* oriundos de domínios distintos faz com que os serviços autônomos não possuam ciência do contexto global da execução, gerando conflito ou execuções injustas, pois cada um deles não possui as informações sobre a execução do *workflow*.

2. Trabalhos Relacionados

A partir da possibilidade de utilizar serviços, começa a haver uma maior demanda no uso de *web services*, aumentando a concorrência e a complexidade em sua execução através da criação de serviços que executam composição entre serviços existentes, apresentando novos desafios que precisam ser transpostos. Para contornar esta situação, esforços são empregados para aprimorar o controle de concorrência e o gerenciamento de transação aos serviços. A seguir são apresentados alguns trabalhos que propõem soluções para o problema.

2.1.Processamento de transações P2P através de grade computacional

Turker *et al.* [1] propõem um ambiente descentralizado para o controle de concorrência, em ambiente de grade computacional, para execução transacional de *workflows* com composição de serviços. O ambiente garante execuções globalmente corretas sem a necessidade de um coordenador global, uma vez que esta abordagem em ambiente de grade inviabiliza o sistema, criando um gargalo com milhares de *peers* acessando um ponto centralizado. Outro aspecto está em como detectar e lidar com os conflitos de maneira que possa, também, ser realizada de forma descentralizada, evitando *deadlock* entre os *worklfows*.

O foco da pesquisa está voltado para o ambiente de grade computacional, onde há um controle maior entre os participantes se comparados aos usuários de *web services* de outros ambientes, utilizando uma abordagem P2P para solucionar os problemas de conflito. Este tipo de solução viola o princípio de baixo acoplamento de *web services*, pois necessita que um *peer* de serviço saiba de outro *peer* de serviço para a comunicação.

2.2.Protocolo baseado em timestamp e two phase commit para Web Service

Maciel e Hirata [2] propõem um controle transacional autônomo e distribuído entre *web services*, apresentando um protocolo baseado na arquitetura REST. Utiliza critérios de prioridade baseados em marca de tempo lógico dos servidores para a não ocorrência de *deadlock* entre requisições, e o protocolo 2PC para que não haja a necessidade de utilizar ações compensatórias em caso de falha na execução da transação, ou em caso de desistência do recurso adquirido.

No entanto, esta solução pode gerar situação de *starvation* de requisições, uma vez que, caso a requisição tenha marca de tempo menor que a marca de tempo do serviço, esta requisição necessita realizar reenvio, o qual pode persistir de maneira indefinida.

3. Arquitetura para execução transacional

A arquitetura contempla a execução transacional entre composições de serviços com origem em domínio distinto que utilizam conjunto de serviços comuns, mantendo características fundamentais de *web services*, os quais compreende desde a autonomia entre os serviços, o baixo acoplamento, a interoperabilidade e a transparência de como o serviço foi concebido ou disponibilizado. A arquitetura está estruturada para oferecer dois serviços, serviço Requisição e serviço Recurso ilustrados na Figura 1, os quais em conjunto promoverão o comportamento transacional para a execução de *workflows*, através do atendimento justo as requisições, evitando *deadlock* e *starvation*.

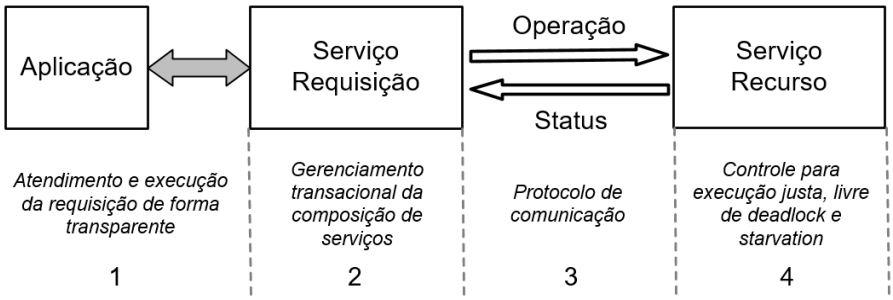


Figura 1 – Arquitetura para Execução Transacional

Os itens a seguir descrevem cada camada da arquitetura ilustrada na Figura 1.

1) A arquitetura desenvolvida é transparente para as aplicações, onde estas aplicações continuam utilizando serviços de maneira habitual, sem a necessidade do conhecimento

prévio sobre a complexidade envolvida em uma requisição, bastando realizar uma chamada comum aos *web services*.

2) O serviço Requisição recebe uma requisição de uma aplicação, a qual solicita múltiplos recursos, executando uma composição de serviços providos pelos serviços Recursos. Para garantir o procedimento transacional desta composição, são executadas operações individuais sobre os serviços Recursos, seguindo-se a ordem necessária de execução determinada por um *workflow*, gerenciando o *status* de cada requisição para efetuar ou não a alocação dos recursos. Ao final da execução retorna uma confirmação de sucesso ou não da requisição realizada para a aplicação.
3) O protocolo de comunicação permite a interação entre o serviço Requisição e serviço Recurso. O serviço Requisição recebe informações do serviço Recurso, sobre o *status* de cada requisição, para que garanta a execução transacional do *workflow*. O serviço Recurso recebe as informações sobre o servidor do serviço Requisição, e a operação desejada de maneira que realize um atendimento justo livre de *deadlock* e *starvation*.
4) O serviço Recurso recebe as requisições individuais dos serviços Requisições, controlando e organizando o acesso e a manipulação do recurso para que sejam realizados de maneira justa. Para o critério de justiça considera-se o instante de envio da requisição pelo serviço Requisição e não o instante que o serviço Recurso recebe a requisição, representado pela marca de tempo (TS - *timestamp*) de relógio lógico de cada entidade. Esta política foi adotada, pois requisições enviadas, a partir de uma mesma origem, para diversas localidades podem atingir seus destinos em instantes de tempo diferentes. O critério de prioridade segue a seguinte ordem:
    - Menores TS's
    - Se TS's são iguais:
        - Menores ID's de requisição
        - Se ID's de requisição são iguais
            - Menores IP's do serviço Requisição

    O gerenciamento das requisições para a alocação de recursos está baseado em três listas:

    - Arrival List (AL) – utilizada para armazenar as requisições recebidas, atuando como uma listagem de pedidos que chegaram e ainda não foram atendidos.
    - Reservation List (RL) – utilizada para armazenar as requisições que obtiveram a reserva do recurso, porém a alocação do recurso ainda não foi efetivada.
    - Wait List (WL) – utilizada para armazenar requisições que não conseguiram a reserva e aguardam a liberação de reservas de outras requisições. Esta lista é reorganizada quando deve incluir uma nova requisição com maior prioridade.

    Os recursos não são alocados imediatamente, mas reservados. Os recursos serão alocados, quando todos os recursos solicitados em uma composição de serviço foram reservados. Caso algumas das reservas não forem obtidas, os recursos reservados são liberados.

    O atendimento no serviço Recurso está baseado no valor do TS de cada requisição. Em uma execução comum (TS maior) as requisições são encaminhadas para lista AL, onde caso haja a quantidade de recurso disponível para o atendimento a requisição será encaminhada para a lista RL.

    Caso uma requisição possua prioridade (TS menor) em relação as requisições atendidas, estas requisições serão transferidas, em ordem crescente pelo TS, para a lista WL. A lista WL têm prioridade em relação às da lista AL para obter a reserva do recurso. Esta política evita o impasse entre requisições, impedindo a situação de *deadlock*.

No atendimento às requisições, tanto as que estão na lista WL quanto as que estão na lista AL, é verificado a QTD_D[1] do recurso. Caso seja suficiente a reserva é realizada transferindo-a para lista RL, caso contrário é verificado a QTD_T[2] do recurso. Caso haja QTD_T disponível, a requisição aguarda o cancelamento de requisições até a disponibilidade do recurso, caso contrário o atendimento será negado. Esta estratégia foi utilizada para requisições que possuíam o recurso e precisou ceder o privilégio obtido devido a uma nova requisição com prioridade maior. Esta abordagem não necessita realizar reenvio de requisição, bastando aguardar, com prioridade em relação a outras requisições, a disponibilidade do recurso, evitando a situação de *starvation*.

4. Conclusão

O desafio é aplicar estratégias para execução de *workflows* distintos, que utilizam serviços em múltiplos servidores, de maneira autônoma entre os serviços sem a necessidade de um coordenador centralizador ou interferir em características básicas de *web services*. Nesta estratégia, as entidades que proveem os recursos devem receber informações para determinar a prioridade de cada requisição para utilizar o serviço. As entidades que realizam as requisições devem oferecer informações para que a requisição seja atendida de forma justa, evitando situação de *deadlock* e *starvation*, e receber informações para que gerencie a execução da composição de maneira transacional.

A definição do critério de justiça, através da política de prioridade baseada no instante de solicitação de uma requisição e não no instante que um serviço recebe a requisição foi o elemento base para o desenvolvimento deste trabalho. Este instante é obtido através de valor de relógio lógico (TS) de cada servidor de serviço. Caso haja um impasse, entre valores de TS, é decidido através do identificador da requisição, e persistindo o impasse é decidido através do identificador da máquina de origem da requisição (número IP). Esta política produziu um critério de prioridade que em caso de impasse no atendimento de requisições haja uma maneira de determinar o privilégio para o uso do recurso impedindo a situação de *deadlock*. O reparo realizado no atendimento da requisição, não elimina o privilégio que esta havia obtido em instante anterior, bastando aguardar a desistência de reservas para obter o privilégio novamente. O mecanismo de reparo faz com que a requisição fique temporariamente suspensa, sem a necessidade de realizar reenvio para obter o recurso novamente, impedindo a situação de *starvation*. A notificação de estado para as operações solicitadas permite que a entidade que está executando a composição de serviços possa gerenciar cada operação solicitada de maneira que possibilite a execução transacional.

Este trabalho está em fase de testes, e os resultados preliminares mostram que é possível realizar o gerenciamento de forma transacional de *workflows* distintos, de maneira autônoma. O gerenciamento transacional para a execução dos *workflows* é realizado através de informações que estão contidas nas mensagens de *request/reply* que as entidades trocam, e as requisições são atendidas de maneira justa, impedindo a situação de *deadlock* e *starvation*.

5. Referências Bibliográficas

1. TURKER, C. et al. **How can we support Grid Transactions? Towards Peer-to-Peer Transaction Processing**. Proceedings of the Second Conference on Innovative Data Systems Research. 2005. p. 174-185.
2. MACIEL, L. A. H. D. S.; HIRATA, C. M. A timestamp-based two phase commit protocol for web services using rest architectural style. **Journal of Web Engineering**, v. 9, n. 3, p. 266-282, Setembro 2010.

[1] QTD_D (quantidade disponível) é a quantidade total de recursos subtraindo a quantidade em reserva dos recursos.
[2] QTD_T (quantidade total) é a quantidade de recursos, desconsiderando as reservas realizadas.

Uma Abordagem Prática no Desempenho de uma Nuvem IaaS com Ênfase na Camada de Armazenamento de Dados

Gustavo C. Bruschi[1], Leandro L. Pauro[1], Roberta Spolon[1], Renata S. Lobato[2], Aleardo Manacero[2], Marcos A. Cavenaghi[3]

[1]Departamento de Computação
UNESP Universidade Estadual Paulista "Júlio de Mesquita Filho" Bauru, Brasil

[2]Departamento de Ciências da Computação e Estatística
UNESP Universidade Estadual Paulista "Júlio de Mesquita Filho" São José do Rio Preto, Brasil

[3]Humber Institute of Technology & Advanced Learning
The Business School Toronto, ON

gustavo@bruschi.net, leapauro@hotmail.com, roberta@fc.unesp.br, renata@ibilce.unesp.br, aleardo@ibilce.unesp.br, marcos.cavenaghi@humber.ca

Abstract. *This article presents a mechanism to monitor and measure the performance of an IaaS cloud multilayers using Apache CloudStack. In order to evaluate the resource consumption of the data storage layer, tests were performed in a private cloud computing, allowing measure the CPU consumption and I/O. We conclude that the number of requests to read and write generated by the other layers, in addition to generating a high consumption of disk in the storage layer directly impacts other resources such as CPU and memory.*

Resumo. *Este artigo apresenta um mecanismo que permite monitorar e mensurar o desempenho de uma Nuvem IaaS multicamadas, utilizando o Apache CloudStack. Com propósito de avaliar o consumo de recursos da camada do armazenamento de dados, foram realizados testes em uma Nuvem computacional privada, possibilitando mensurar os consumos de CPU e E/S. Conclui-se que a quantidade de requisições para gravação e leitura geradas pelas demais camadas, além de gerar um alto consumo de disco na camada de armazenamento, impacta diretamente outros recursos, como CPU e memória.*

1. Introdução

A Computação em Nuvem pode ser definida como um modelo que admite, de forma conveniente, o acesso a rede sob demanda a um *pool* compartilhado de recursos configuráveis de computação como redes, armazenamento, servidores e serviços, que podem ser rapidamente provisionados e liberados com um esforço mínimo de gerenciamento ou interação com o provedor de serviços. [Mell e Grance 2011].

Este artigo tem como objetivo apresentar um mecanismo capaz de extrair e armazenar informações de consumo de recursos computacionais durante a utilização de uma nuvem IaaS criada com Apache CloudStack na camada de orquestração, XenServer

na camada de virtualização e Openfiler na camada de armazenamento de dados, além de validar este mecanismo em um ambiente controlado, possibilitando a análise dos dados obtidos. São apresentados neste trabalho os resultados obtidos com a analise percentual de CPU disponível e o Total de CPU aguardando E/S na camada de armazenamento de dados,

2. StackAct – Mecanismo de Avaliação de Desempenho

O StackAct é um mecanismo desenvolvido em Shell Script que realiza a coleta de dados referentes ao consumo de recursos computacionais e permite a avaliação do desempenho no provisionamento de instâncias em um ambiente IaaS com o Apache CloudStack. Permite realizar o monitoramento e coleta de dados referentes ao consumo de diversos recursos computacionais, como CPU, Memória, Disco, Processos e Rede e foi desenvolvido para permitir o monitoramento nos componentes de uma Nuvem computacional em até 3 camadas, sugeridas neste artigo por: camada do orquestrador, utilizando o Apache CloudStack, camada do *hypervisor* e a camada de armazenamento de dados.

A sequência de eventos realizadas pelo StackAct para coleta e posterior análise dos dados são compreendidas nas seguintes etapas:

- **Definição da Oferta de Serviço:** São as características de recursos computacionais para uma instância na Nuvem. Já deve ter sido criada e estar disponível para utilização no Apache CloudStack.
- **Número de Instâncias Simultâneas:** É possível realizar um teste com a quantidade de instâncias desejadas, desde que o ambiente permita.
- **Definição do Ambiente Alvo:** O mecanismo permite escolher a(s) camada(s) da Nuvem compreendidas no escopo do teste.
- **Definição da Rotina:** A rotina total compreende em 4 etapas, sendo estas – criação e inicialização de uma instância, snapshot, desligamento e remoção da instância. É possível optar pela rotina completa ou parte dela.
- **Início da Coleta SAR:** O *System Activity Reporter* é um conjunto de ferramentas, originalmente desenvolvido para ambiente SOLARIS e posteriormente portado para Linux, que realiza o monitoramento de diversos recursos do sistema, como CPU, Memória e Swap, E/S de leitura e gravação, *System Load Average*, Atividade de Rede, Atividade de Criação de Processo, entre outros.
- **Rotina CloudMonkey:** O CloudMonkey é uma ferramenta criada para automatizar processos para o Apache CloudStack.
- **Tratamento LOG SAR:** Neste ponto foram selecionados 6 aspectos relacionados a consumo de recursos, para avaliação: Total de CPU disponível, Total de CPU consumida por processos de usuário, Total de CPU aguardando E/S em disco, E/S de Leitura de dados, E/S de Gravação de dados e Consumo de Memória.

- **KSAR:** O KSAR possibilita exportar os resultados de comandos SAR em gráficos PDF, JPEG ou exportar para planilhas CSV, este último utilizado na elaboração nos testes.

3. Definições e Execução dos Testes

Para validação do StackAct, foi criado um ambiente controlado composto por 5 servidores físicos: um servidor com processador Intel Core2 Quad com 2.66Ghz e 4GB de memória RAM, onde foi instalado S.O. Linux CentOS 6.6 e o orquestrador Apache CloudStack 4.4.2. Para a camada do hypervisor, dois servidores com as mesmas características do servidor orquestrador, com a exceção que foram adicionados mais 4GB de memória RAM, totalizando 8GB em cada um dos servidores. Foi instalado o hypervisor XenServer 6.2.0 e posteriormente criado um cluster entre os dois hosts. Fazendo a função de armazenamento primário e secundário para o orquestrador CloudStack, foram utilizados mais dois servidores, com as mesmas características dos demais, no qual foi instalado o Sistema Operacional OpenFiler 2.99 para armazenamento dos arquivos das máquinas virtuais, imagens de S.O., snapshot, entre outros. Para compartilhamento das pastas na rede, foi criado um compartilhamento em nível de arquivos utilizando NFS (Network File System), com acesso irrestrito e público as pastas compartilhadas na rede.

Posteriormente foi criada uma oferta de serviços - com 1 core de CPU, 1GHz e 1GB de memória RAM. Assim, foram realizados 20 testes com rotinas completas com até 4 instâncias sendo executadas simultaneamente.

4. Resultados Obtidos

Na Figura 1 é apresentado o percentual de CPU disponível e o Total de CPU aguardando E/S na camada de armazenamento de dados, que demonstra que as requisições de utilização de disco realizadas pelas camadas superiores, em especial a camada do hypervisor – para rotinas com as máquinas virtuais - representa a maior parte do consumo de CPU deste ambiente. Outros fatores que impactam no uso da CPU, como os processos do S.O. ou do próprio sistema de compartilhamentos de arquivos não apresentaram um valor tão expressivo, que demonstra que a tecnologia de disco utilizada – no caso do ambiente proposto um disco SATA2 de 7200RPM pode ser considerado um ponto de atenção para melhorar desempenho na solução.

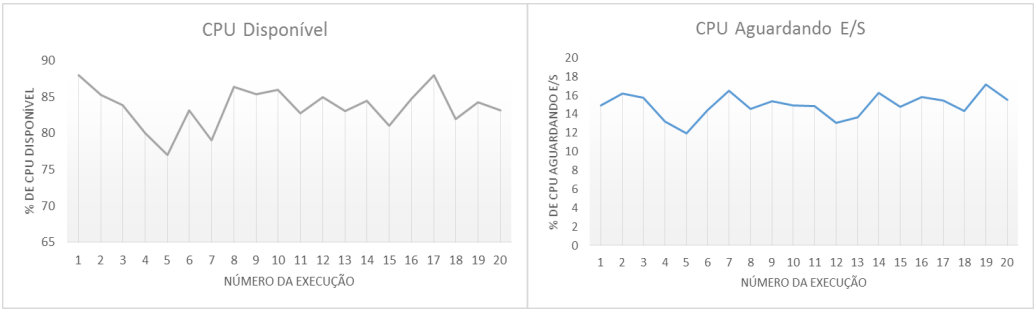


Figura 1 – CPU disponível e CPU aguardando E/S.

Na Figura 2 é apresentado o número de requisições por segundo de gravação e leitura em disco na camada de armazenamento de dados, que demonstra maior número

de requisições para gravação, devido aos processos compreendidos na rotina – criação de instância, snapshot, desligamento e remoção da instância. Mesmo considerado um ambiente controlado, com número limitado da oferta de serviço e rotinas realizadas de forma simultânea, é possível observar um alto consumo do disco, reforçando a relação direta com o desempenho da solução.

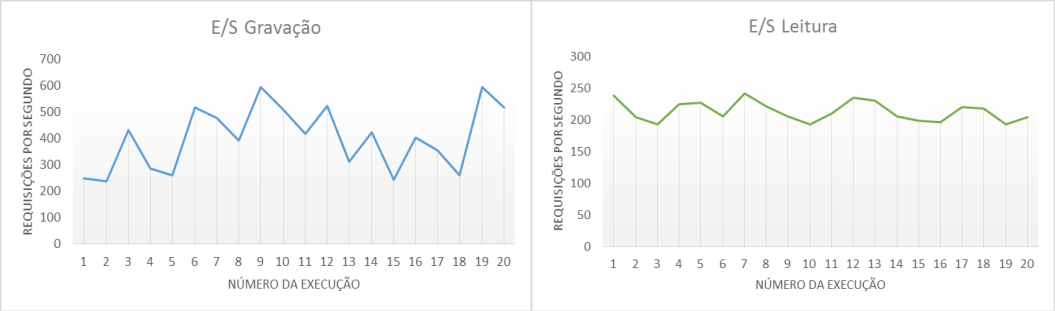


Figura 2 – Requisições de gravação e leitura em disco.

5. Conclusões

Este artigo apresentou um mecanismo de coleta de dados relacionados ao consumo de recursos computacionais em uma Nuvem IaaS multicamadas, denominado StackAct, utilizando o orquestrador Apache CloudStack, o hypervisor XenServer e o sistema de armazenamento Openfiler. Os testes iniciais de desempenho foram realizados em um ambiente de Nuvem privado controlado.

Através de ferramentas *opensource* é possível elaborar um mecanismo de coleta de desempenho que funciona em ambiente com Apache CloudStack e diversos sistemas na camada do *hypervisor* e armazenamento de dados. Utilizando este mecanismo em ambiente Apache CloudStack, é possível realizar a coleta para posterior análise de consumo de diversos recursos computacionais – todos que a ferramenta SAR possibilitar o monitoramento.

Quanto aos resultados dos testes obtidos em ambiente controlado, foi possível detectar que para realização da rotina definida pelo StackAct - compreendida na criação de uma nova instância, execução de um snapshot, desligamento e remoção desta instância - a quantidade de requisições solicitadas pela camada do orquestrador e camada de armazenamento de dados são refletidas na camada de armazenamento, na qual as requisições de E/S de leitura e gravação impactam diretamente no consumo de outros recursos, como CPU.

6. Referências

NIST. (2011) "NIST Programa de Cloud Computing." http://www.nist.gov/itl/cloud/index.cfm>, 01 outubro 2015.

MELL, P., GRANCE, T. (2014) The NIST Definition of Cloud Computing (Versão 15).

Ferramenta para monitoramento e eliminação de inconsistências em ambiente de nuvem

Leandro L. Pauro[1], Roberta Spolon[1], Gustavo C. Bruschi[1] , Renata S. Lobato[2], Aleardo Manacero[2], Marcos A. Cavenaghi[3]

[1]Departamento de Computação
UNESP Universidade Estadual Paulista "Júlio de Mesquita Filho" Bauru, Brasil

[2]Departamento de Ciências da Computação e Estatística
UNESP Universidade Estadual Paulista "Júlio de Mesquita Filho" São José do Rio Preto, Brasil

[3]Humber Institute of Technology & Advanced Learning
The Business School Toronto, ON

leapauro@hotmail.com, roberta@fc.unesp.br

Abstract. *This paper presents an tool for auditing and monitoring in Apache CloudStack. Perform monitoring information, providing greater integrity, reliability, making the decision-making process more accurate for the cloud administrator in return by eliminating the inconsistencies, preventing also the false positives and false negatives alerts, in addition to providing a reduction of the cost in storing the persistent data.*

Resumo. *Este trabalho apresenta uma ferramenta para auditoria e monitoramento em nuvem no orquestrador Apache CloudStack. Realiza um monitoramento das informações, proporcionando maior integridade, confiabilidade do ambiente, auxiliando o administrador da nuvem em ter uma tomada de decisão com maior precisão, através da eliminação das inconsistências, prevenido também o alerta de falsos positivos e falsos negativos. A ferramenta proporciona redução do custo em armazenamento dos dados persistentes..*

1. Introdução

O aumento da utilização da computação em nuvem em ambiente corporativo vem se tornando maior, devido ser financeiramente menos onerosa para seus usuários e os ajuda a reduzir sua dependência de suporte em TI, visto que a promessa da computação em nuvem é prover os recursos computacionais com maior rapidez, sem ter a despesa de aquisição de uma nova infraestrutura, licenciamento de software e treinamento de pessoal [Hassan 2013].

Este trabalho tem como objetivo apresentar uma ferramenta que realiza um sincronismo das informações persistentes armazenadas na base de dados do orquestrador Apache CloudStack, com suas informações de estado atual do ambiente desta nuvem através de API's (Aplications Programing Interfaces).

Para oferecer uma maior integridade, confiabilidade, auxiliar o administrador da nuvem a ter uma tomada de decisão com maior precisão, eliminar as inconsistências sendo estas informações divergentes sobre o mesmo facto, prevenindo o falso positivo e falso negativo, alem de proporcionar, menores custos para o armazenamento de dados persistentes da nuvem.

A organização do trabalho está dividida da seguinte forma: na Seção 2 apresenta os trabalhos relacionados. A Seção 3 apresenta a ferramenta e a realização dos testes. Na Seção 4 apresenta os resultados obtidos e as conclusões finais do trabalho.

2. Trabalhos relacionados

O trabalho desenvolvido por [Xu et al. 2014] realiza coleta de informações dos logs de instancias criadas na nuvem, proveniente de um processo de atualização sem interrupção. Com a utilização das ferramentas Redis, Logstash, ElasticSearch e Kibana, estes logs distribuídos podem ser armazenados em um único log central. Neste log são aplicadas expressões regulares, algoritmos e API's. Tais aplicações permitem a detecção de 91,95% dos erros nas informações analisadas e um aumento na precisão de diagnóstico de causa raiz de 97,13%.

Para o trabalho de [Xu et al. 2015] relata o problema dos administradores de nuvem que recebem falsos positivos ou até, uma inundação de alarmes de diferentes canais de um mesmo evento. A solução foi a utilização do conceito de máquina de aprendizagem a partir do modelo de SVM, obteve um aumento na taxa de precisão de 84,7%, como isso, possibilitou quais eventos deveriam ser desativados.

O trabalho de [Xiaojiang e Yanlei 2013] demonstra que o framework de gestão de recursos físicos e virtuais do orquestrador OpenStack, tem um desempenho baixo na recuperação das métricas do ambiente. Foi desenvolvido um subsistema de monitoramento de recursos, que armazena informações coletadas em um banco de dados de arquitetura NoSQL, para prover otimização na gestão das informações, não apresentou problemas de consistências de dados e falhas de integração com agentes do ambiente.

3. A ferramenta AMFC e ambiente de testes

A ferramenta AMFC realiza o monitoramento de forma sincronizada, através de uma interface de API's, entre os dados persistentes com as informações de estado atual do ambiente de nuvem Apache CloudStack. Proporciona a eliminação das informações inconsistentes, mitigação dos eventos de falsos positivos e falsos negativos, maior precisão do administrador de infraestrutura, sobre qual ação deve-se tomar neste momento, também oferece um menor custo para armazenamento de dados persistentes da nuvem.

Desenvolvida a partir da linguagem Python, pode ser também implementada em outros orquestradores opensource como, OpenStack, Eucalyptus, OpenNebula, etc. Como o que foi realizado neste trabalho com o orquestrador Apache CloudStack.

Para realização dos testes da ferramenta foi criado um ambiente controlado com a utilização de cinco computadores da marca Lenovo ThinkCentre M58 3.0GHz 4GB 500GB, sendo estes, um servidor de gerenciamento do orquestrador com plataforma

Linux CentOS 6.6 e o orquestrador Apache Cloudstack versão 4.4.2, dois computadores com o hypervisor XenServer 6.2.0 que realizam a criação das máquinas virtuais, dois computadores para armazenamento de dados instalado o S.O OpenFiler 2.99.

Nesses testes foi configurado um tipo recurso para posteriormente realizar a criação das instancias, que compreendem 2 Núcleos, 2000 CPU (Mhz), 2048 Memória (MB) e área de armazenamento de 100 GB.

A Figura 1 apresenta o administrador do ambiente, realizando a criação da instancia através da interface de gerenciamento da nuvem, logo após a instancia criada, são realizadas as rotinas provendo assim as informações a serem auditadas e monitoradas pela ferramenta.

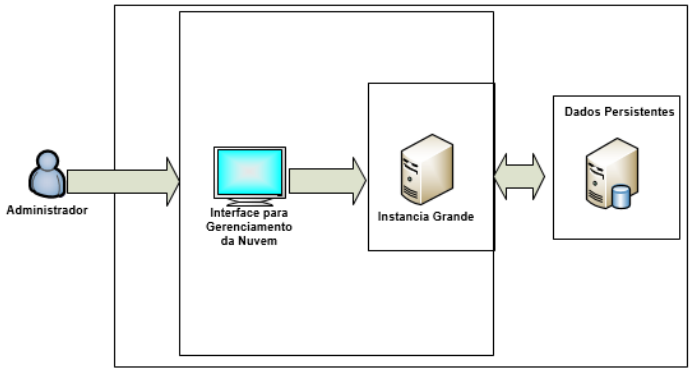


Figura 1 – Monitoramento e eliminação de inconsistências em ambiente de nuvem.

Na realização do experimento, foram definidos 8 diferentes tipos de rotinas de gerenciamento de uma instância de máquina virtual, realizadas através da própria interface de administração do orquestrador, sendo estas: criação de disco de volume de dados, anexar disco de volume de dados, migração da instância para outro host, reiniciar instância, parar execução de instância, apagar instância, recuperar instância e iniciar instância.

Foram criadas dez instancias, sendo que em cada uma foi realizado um número de repetições, por exemplo, na instância chamada VM(3), foram realizadas três vezes o número de cada uma das rotinas citadas, com isso, esta VM(3) obteve um total de 24 rotinas executas ou seja (número de execuções (3) X número de cada rotina) e assim por diante até a VM10, tem-se uma totalização final de 440 rotinas executadas.

4. Resultados e Conclusões

Na Figura 2 é apresentado o cálculo do número de inconsistências / quantidade de rotinas, o que demonstra que a medida que se aumenta o número das rotinas definidas para uma determinada instancia, há também uma maior incidência de inconsistências geradas A ferramenta AMFC proporcionou a identificação destas inconsistências, através do monitoramento entre os dados persistentes e ambiente atual da nuvem.

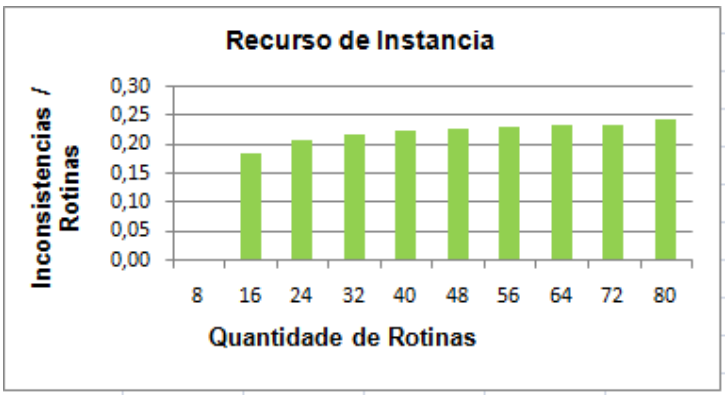


Figura 2 – Resultado do monitoramento de inconsistências

A AMFC obteve ganho em armazenamento de informações persistentes, proveniente da eliminação das inconsistências encontradas e também das informações que eram atribuídas a esta instancia assim que foi expurgada. Realizada uma conversão de 100 GB em 104857600 KB foi possível constatar um ganho de 0,00004120 em termos de proporcionalidade ao final de 440 rotinas executadas.

Este trabalho apresentou a ferramenta AMFC que através da utilização de sincronização por meio de API's entre os dados persistentes com as informações de estado atual da nuvem, realiza a auditoria e monitoramento de informações, no intuito da eliminação de informações inconsistentes que causam o falso positivo e falso negativo, obteve também um ganho em armazenamento de dados persistentes.

Quanto aos resultados dos testes obtidos em um ambiente controlado, foi possível concluir que a ferramenta AMFC eliminou apenas as informações inconsistentes do ambiente, impedindo assim o falso positivo e falso negativo, proporcionando informações consistentes ao ambiente atual e também menor custo de armazenamento para os dados persistentes

5. Referências

Hassan, S.A.Z. (2013) "SONA: A Service Oriented Nodes Architecture for Developing Cloud Computing Applications", IEEE Advanced Computing and Communication Systems (ICACCS), 2013 International Conference on Dec.2013.

Xu,X; Zhu,L; Ingo,W; Bass,l; Sun,D.(2014) "POD-Diagnosis: Error Diagnosis of Sporadic Operations on Cloud Applications." International Conference on Dependable Systems and Networks, 2014 IEEE International, vol., no., pp. 252 - 263, 23-26 June 2014.

Xu,X; Zhu,L; Ingo,W; Bass,l; Sun,D. (2015) "Crying Wolf and Meaning it:Reducing False Alarms in Monitoring of Sporadic Operations through POD-Monitor." COUFLESS, 2015 IEEE/ACM 1st International Workshop on Complex faUlts and Failures in LargE Software Systems, vol., no., pp. 69 - 75, 23-23 May 2015.

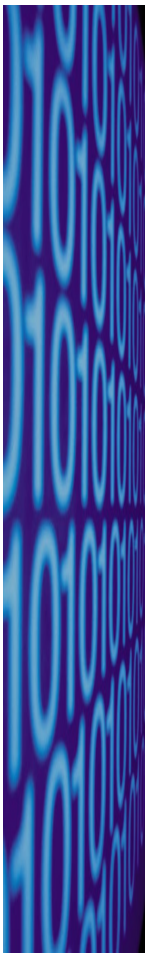
Xiaojiang, L; Yanlei, S. (2013) "The Design and Implementation of Resource Monitoring for Cloud Computing Service Platform." 3rd International Conference on Computer Science and Network Technology, 2013 IEEE International, vol., no., pp. 239 - 243, 12-13 Oct. 2013. Apache. (2015, October 23, 2015).

Construindo ***aceleradores*** com ***FPGAs*** para computação de ***alto desempenho***

Prof. Ricardo Menotti
http://menotti.pro.br/

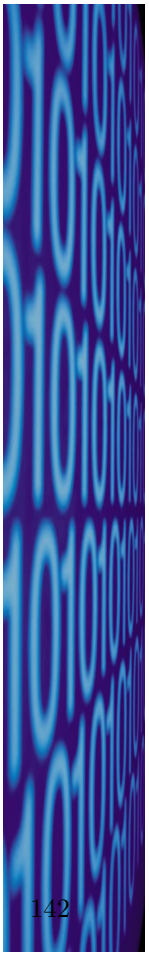
Roteiro

- Computação
  - Principios
  - Heterogênea
  - Reconfigurável
  - Aceleradores
  - Ferramentas
  - Iniciativas
  - Conclusão



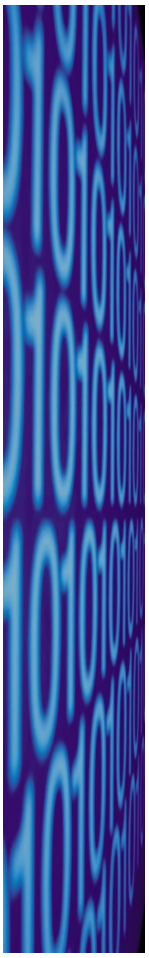
Computação

- Em que consiste?



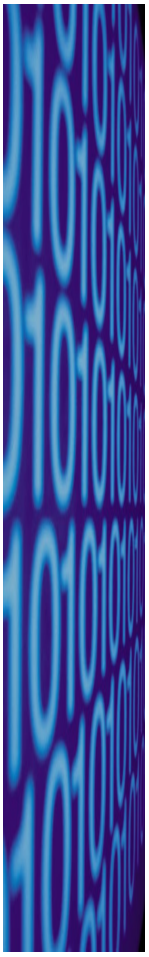
Computação

- Em que consiste?
- Métodos para execução de algoritmos



Computação

- Em que consiste?
- Métodos para execução de algoritmos
  - Software executando em CPU

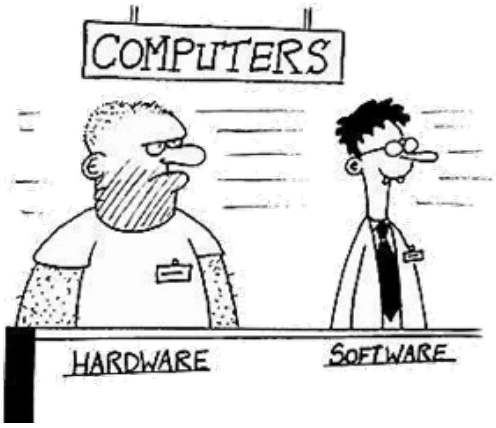


Computação

- Em que consiste?
- Métodos para execução de algoritmos
  - Software executando em CPU
  - Hardware dedicado

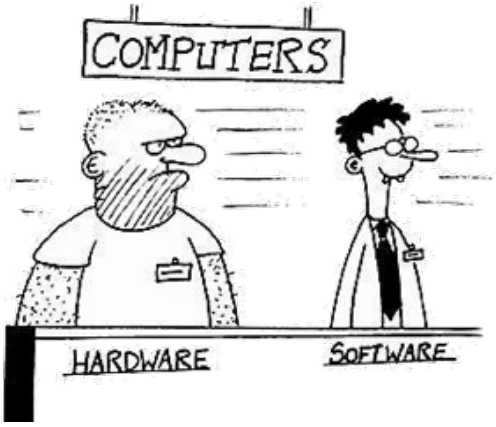
Computação

- Em que consiste?
- Métodos para execução de algoritmos
  - Software executando em CPU
  - Hardware dedicado



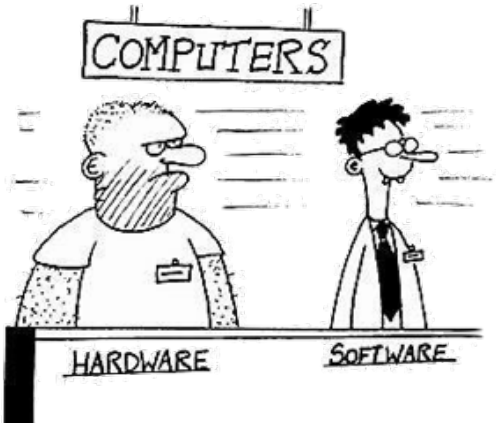
Computação

- Em que consiste?
- Métodos para execução de algoritmos
  - Software executando em CPU
  - Hardware dedicado



Computação

- Em que consiste?
- Métodos para execução de algoritmos
  - Software executando em CPU
  - Hardware dedicado



Computação

- Em que consiste?
- Métodos para execução de algoritmos
  - Software executando em CPU
  - Hardware dedicado

Software

- Processadores de propósito geral
  - A solução para o problema é um programa
- Vantagem: **flexibilidade**
  - O programa pode ser facilmente alterado
- Desvantagem: **ineficiência**
  - O processador não foi projetado para resolver aquele problema em especial

4

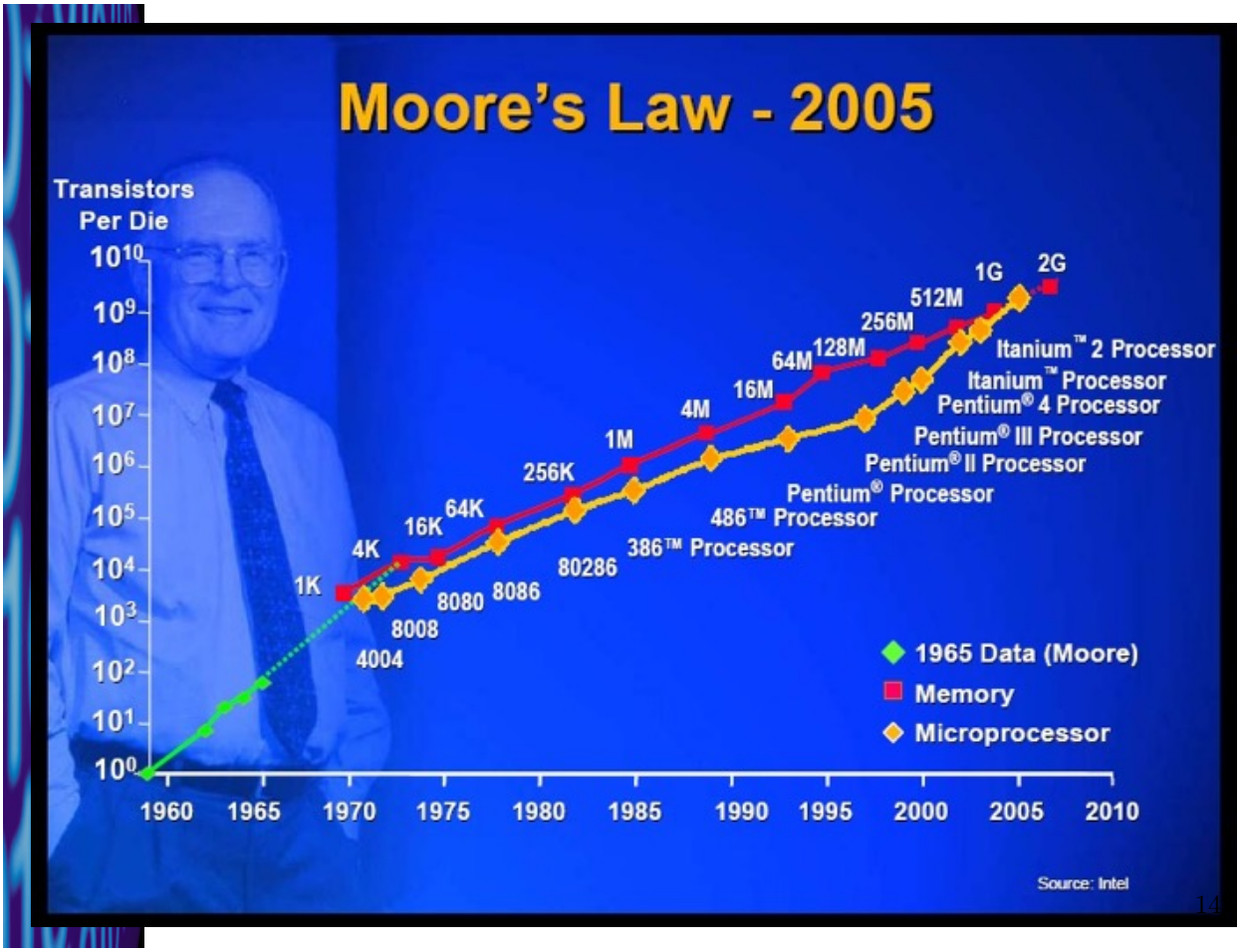
Software

- Processadores de propósito específico
  - Otimizados para determinadas soluções
- Graphics Processing Unit (GPU)
  - Otimizados para processamento gráfico
- Digital Signal Processor (DSP)
  - Otimizados para processamento matemático

5

Hardware

- Application-specific integrated circuit
  - System-on-a-chip
- Vantagem: **eficiência**
  - O chip foi desenvolvido especificamente para resolver aquele problema
- Desvantagem: ~~**flexibilidade**~~
  - Uma vez fabricado, não é mais possível fazer qualquer modificação



TOP 500
The List.

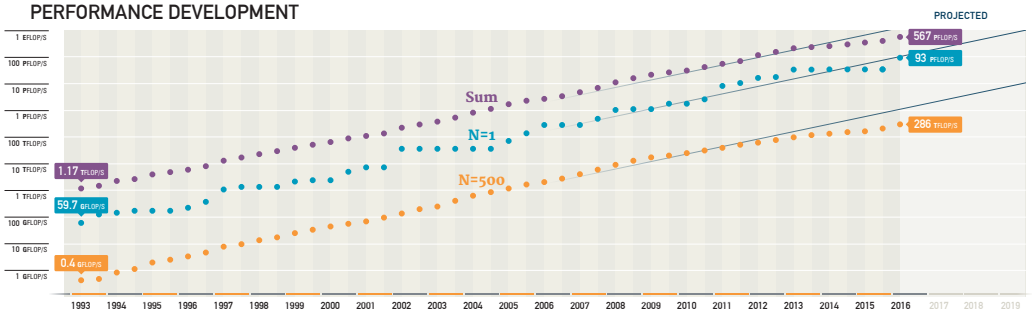
PRESENTED BY

ICL INNOVATIVE COMPUTING LABORATORY · BERKELEY LAB Lawrence Berkeley National Laboratory · ISC GROUP Moving Forward.

FIND OUT MORE AT
top500.org

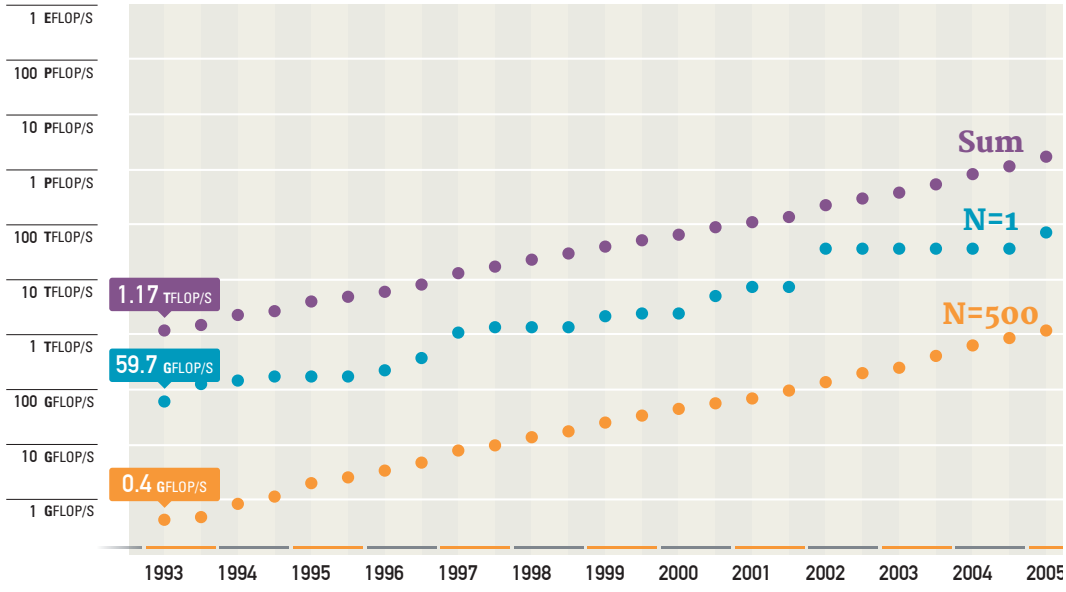
	NAME	SPECS	SITE	COUNTRY	CORES	Rmax PFLOP/S	POWER MW
1	Sunway TaihuLight	Shenwei SW26010 (260C 1.45 GHz) Custom interconnect	NSCC in Wuxi	China	10,649,600	93.0	15.4
2	Tianhe-2 (Milkyway-2)	Intel Ivy Bridge (12C 2.2 GHz) & Xeon Phi (57C 1.1 GHz), Custom interconnect	NSCC in Guangzhou	China	3,120,000	33.9	17.8
3	Titan	Cray XK7, Opteron 6274 (16C 2.2 GHz) + Nvidia Kepler GPU, Custom interconnect	DOE/SC/ORNL	USA	560,640	17.6	8.2
4	Sequoia	IBM BlueGene/Q, Power BQC (16C 1.60 GHz), Custom interconnect	DOE/NNSA/LLNL	USA	1,572,864	17.2	7.9
5	K computer	Fujitsu SPARC64 VIIIfx (8C 2.0 GHz), Custom interconnect	RIKEN AICS	Japan	705,024	10.5	12.7

PERFORMANCE DEVELOPMENT

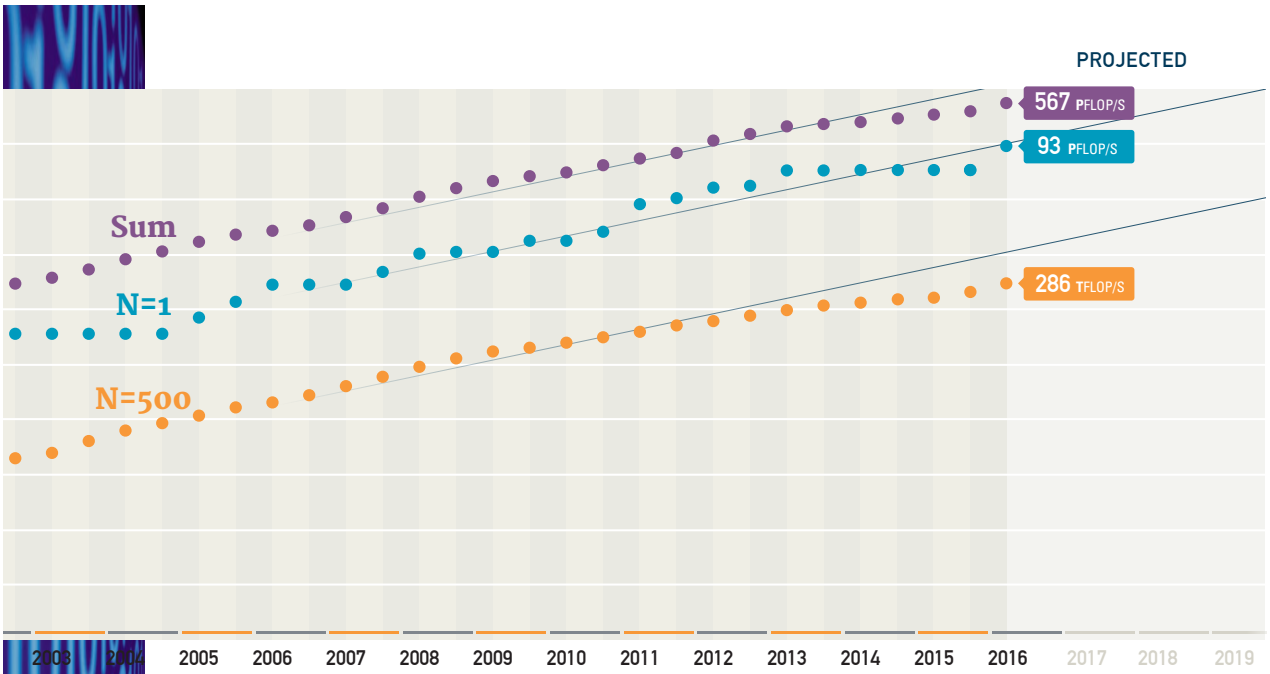


8

PERFORMANCE DEVELOPMENT



9



TOP 500
The List.

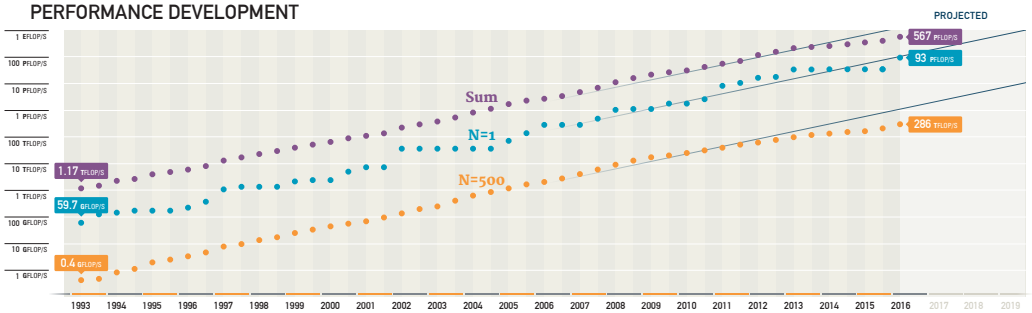
PRESENTED BY

ICL INNOVATIVE COMPUTING LABORATORY
BERKELEY LAB Lawrence Berkeley National Laboratory
ISC GROUP Moving Forward.

FIND OUT MORE AT
top500.org

	NAME	SPECS	SITE	COUNTRY	CORES	RMAX PFLOP/S	POWER MW
1	Sunway TaihuLight	Shenwei SW26010 (260C 1.45 GHz) Custom interconnect	NSCC in Wuxi	China	10,649,600	93.0	15.4
2	Tianhe-2 (Milkyway-2)	Intel Ivy Bridge (12C 2.2 GHz) & Xeon Phi (57C 1.1 GHz), Custom interconnect	NSCC in Guangzhou	China	3,120,000	33.9	17.8
3	Titan	Cray XK7, Opteron 6274 (16C 2.2 GHz) + Nvidia Kepler GPU, Custom interconnect	DOE/SC/ORNL	USA	560,640	17.6	8.2
4	Sequoia	IBM BlueGene/Q, Power BQC (16C 1.60 GHz), Custom interconnect	DOE/NNSA/LLNL	USA	1,572,864	17.2	7.9
5	K computer	Fujitsu SPARC64 VIIIfx (8C 2.0 GHz), Custom interconnect	RIKEN AICS	Japan	705,024	10.5	12.7

PERFORMANCE DEVELOPMENT



TOP 500
The List.

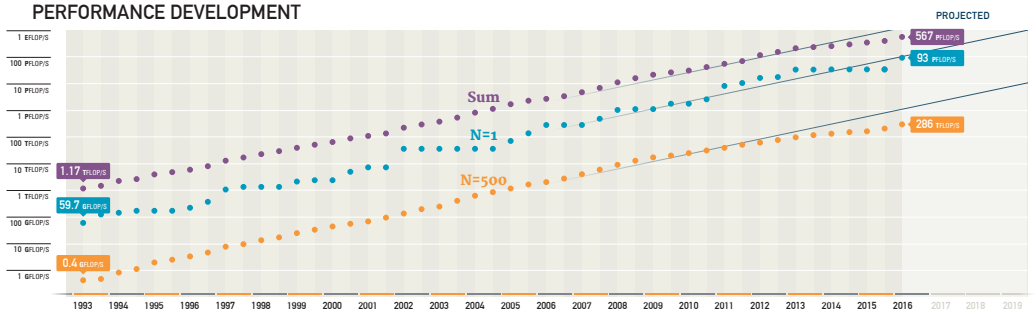
PRESENTED BY

ICL INNOVATIVE COMPUTING LABORATORY · BERKELEY LAB Lawrence Berkeley National Laboratory · ISC GROUP Moving Forward.

FIND OUT MORE AT
top500.org

	NAME	SPECS	SITE	COUNTRY	CORES	Rmax PFLOP/S	POWER MW
1	Sunway TaihuLight	Shenwei SW26010 (260C 1.45 GHz) Custom interconnect	NSCC in Wuxi	China	10,649,600	93.0	15.4
2	Tianhe-2 (Milkyway-2)	Intel Ivy Bridge (12C 2.2 GHz) & Xeon Phi (57C 1.1 GHz), Custom interconnect	NSCC in Guangzhou	China	3,120,000	33.9	17.8
3	Titan	Cray XK7, Opteron 6274 (16C 2.2 GHz) + Nvidia Kepler GPU, Custom interconnect	DOE/SC/ORNL	USA	560,640	17.6	8.2
4	Sequoia	IBM BlueGene/Q, Power BQC (16C 1.60 GHz), Custom interconnect	DOE/NNSA/LLNL	USA	1,572,864	17.2	7.9
5	K computer	Fujitsu SPARC64 VIIIfx (8C 2.0 GHz), Custom interconnect	RIKEN AICS	Japan	705,024	10.5	12.7

PERFORMANCE DEVELOPMENT



Green500 Rank	MFLOPS/W	Site*	Computer*	Total Power (kW)
1	6,673.84	Advanced Center for Computing and Communication, RIKEN	Shoubu - ZettaScaler-1.6, Xeon E5-2618Lv3 8C 2.3GHz, Infiniband FDR, PEZY-SCnp	149.99
2	6,195.22	Computational Astrophysics Laboratory, RIKEN	Satsuki - ZettaScaler-1.6, Xeon E5-2618Lv3 8C 2.3GHz, Infiniband FDR, PEZY-SCnp	46.89
3	6,051.30	National Supercomputing Center in Wuxi	Sunway TaihuLight - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway	15,371.00

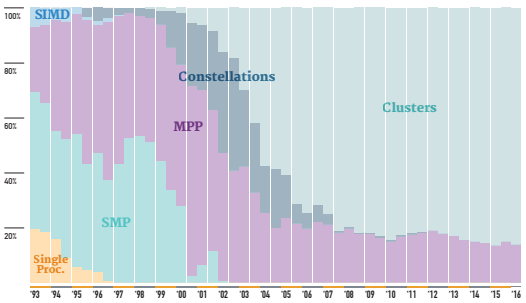
TOP 500
The List.

PRESENTED BY

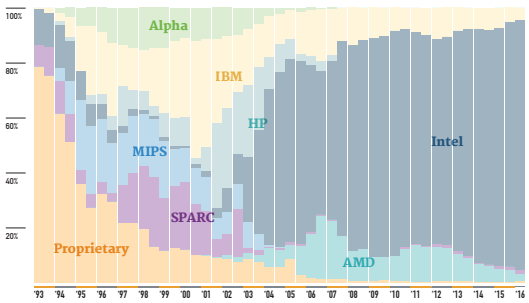
ICL INNOVATIVE COMPUTING LABORATORY · BERKELEY LAB Lawrence Berkeley National Laboratory · ISC GROUP Moving Forward.

FIND OUT MORE AT
top500.org

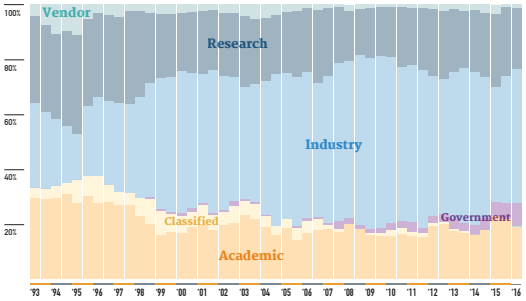
ARCHITECTURES



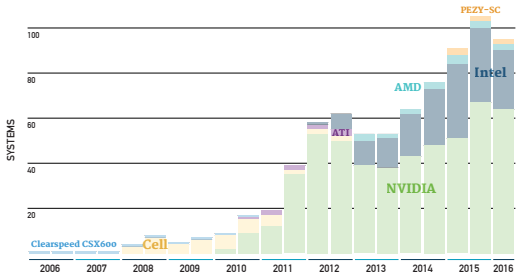
CHIP TECHNOLOGY

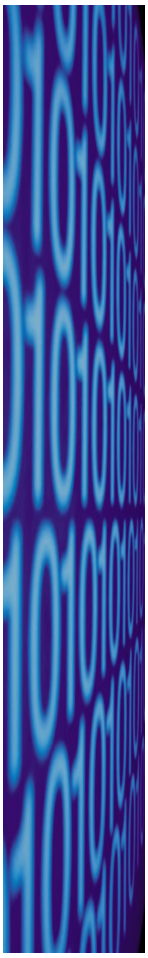


INSTALLATION TYPE



ACCELERATORS/CO-PROCESSORS



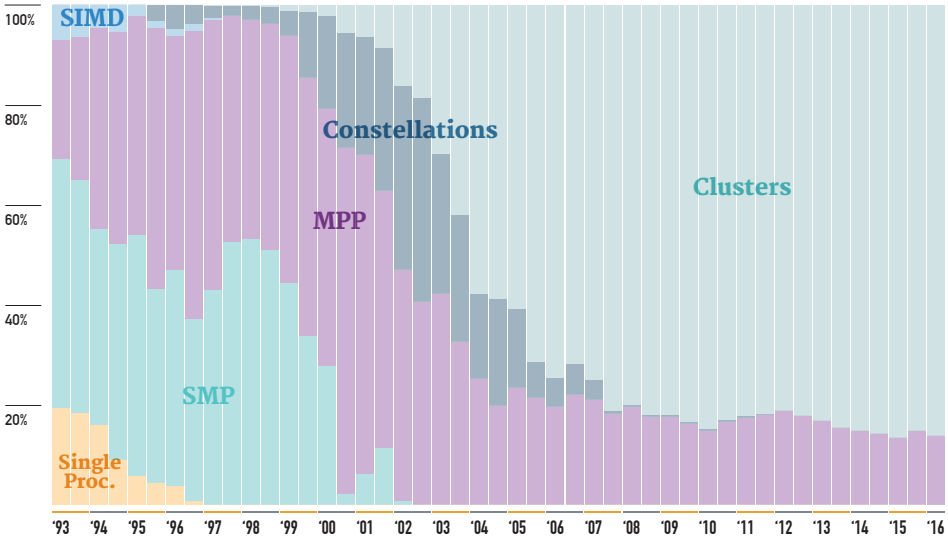


The List.

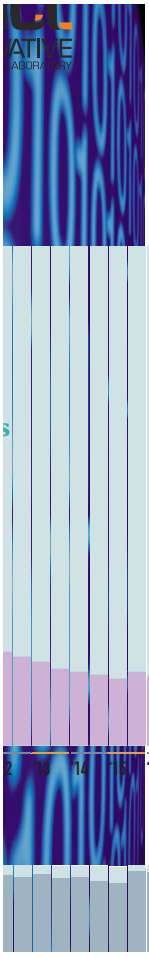
INNOVATIVE COMPUTING LABORATORY

BERKELEY LAB

ARCHITECTURES



INSTALLATION TYPE

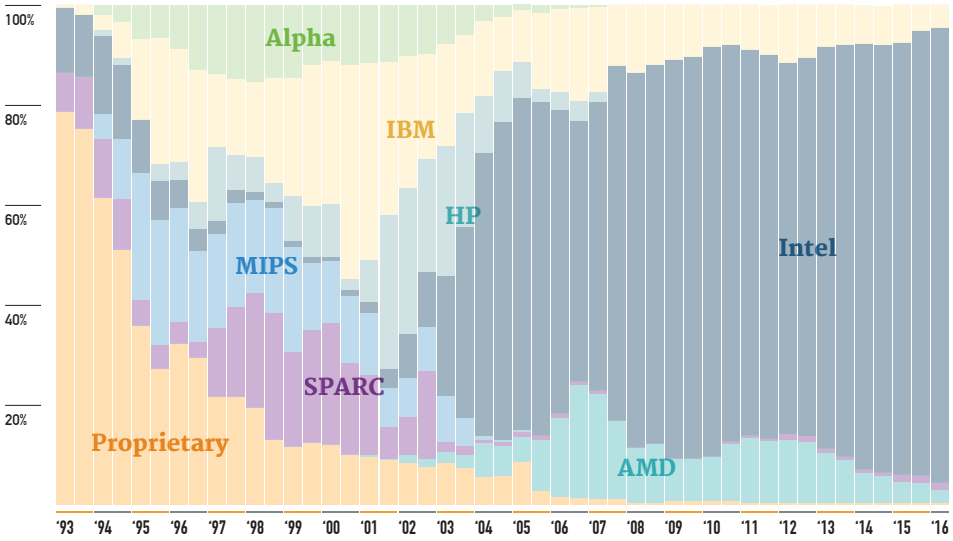


BERKELEY LAB Lawrence Berkeley National Laboratory

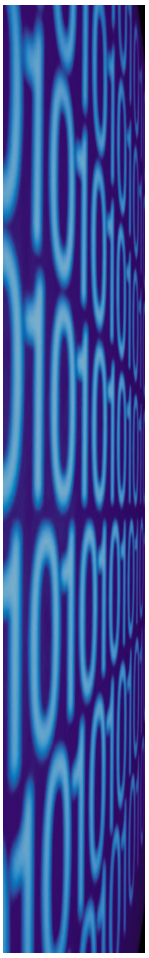
ISC GROUP Moving Forward.

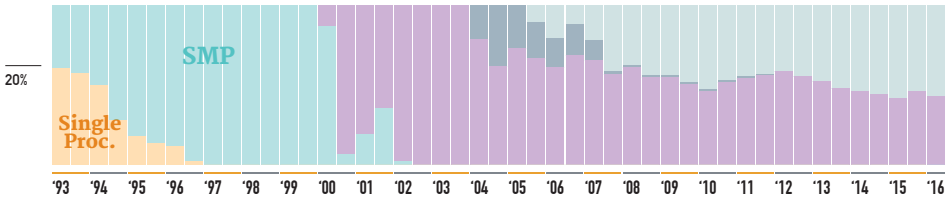
top500.org

CHIP TECHNOLOGY

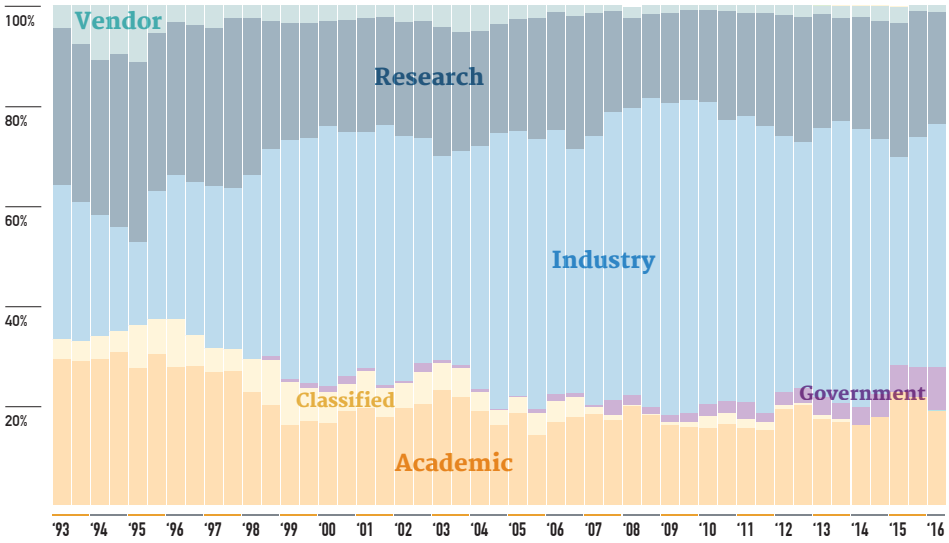


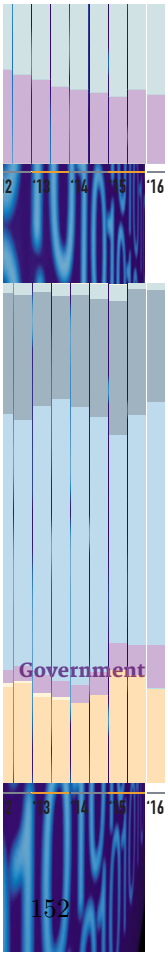
ACCELERATORS/CO-PROCESSORS

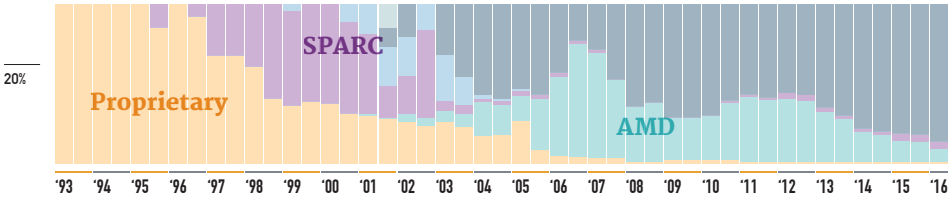




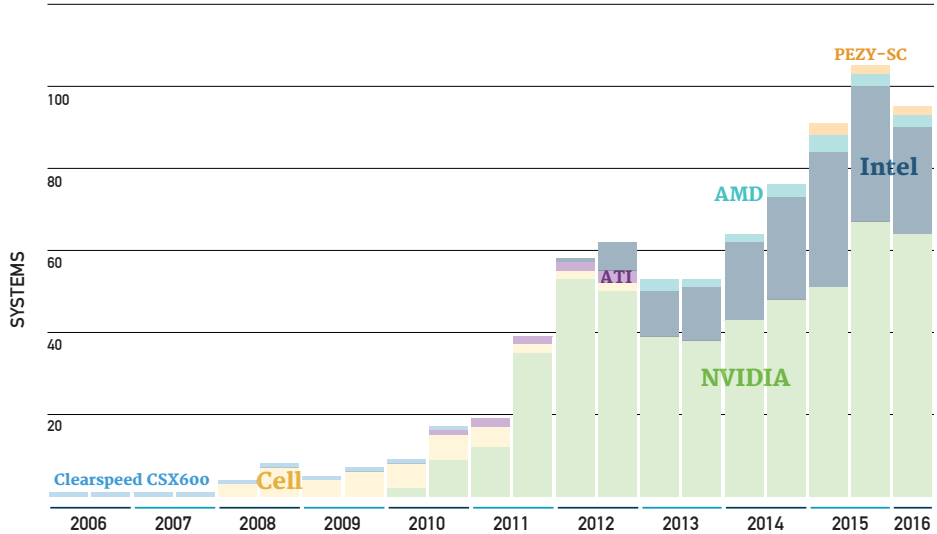
INSTALLATION TYPE







ACCELERATORS/CO-PROCESSORS



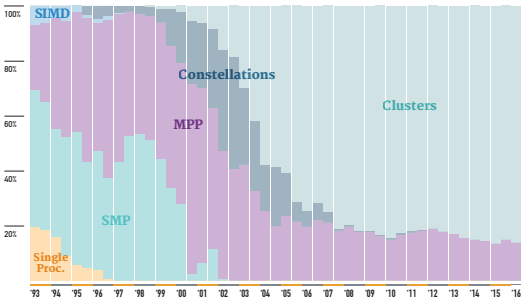
TOP 500
The List.

PRESENTED BY

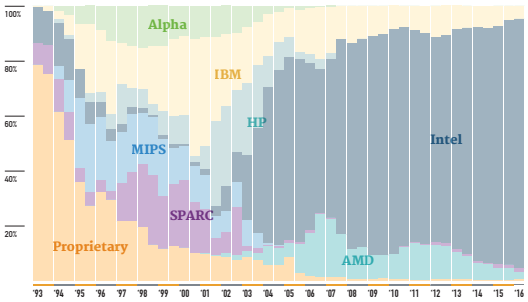
ICL INNOVATIVE COMPUTING LABORATORY · BERKELEY LAB Lawrence Berkeley National Laboratory · ISC GROUP Moving Forward.

FIND OUT MORE AT
top500.org

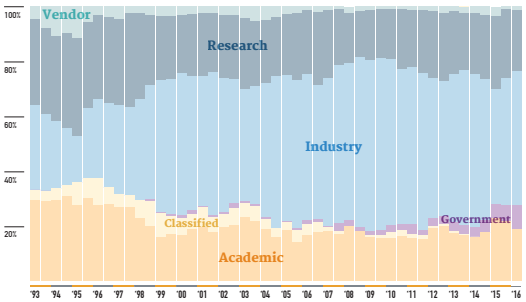
ARCHITECTURES



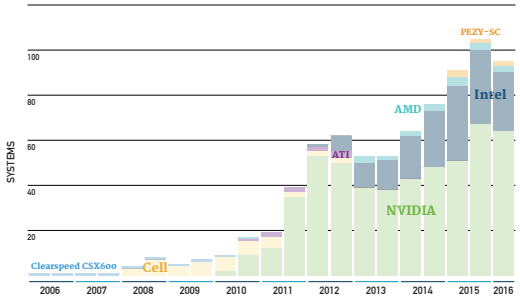
CHIP TECHNOLOGY



INSTALLATION TYPE



ACCELERATORS/CO-PROCESSORS

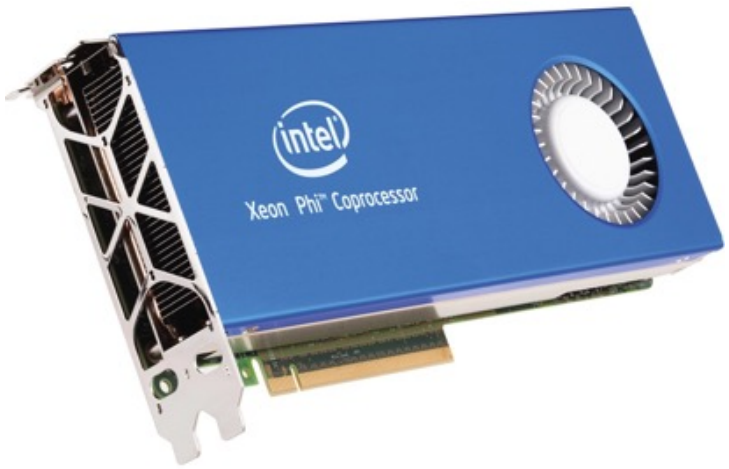


GPU

- *Graphics Processing Unit*
- Programação
  - CUDA
  - OpenCL
- GPGPU

Intel Xeon Phi

- *Many Integrated Cores (MIC)*
- Programação
  - OpenCL
  - OpenMP
  - MPI
  - …



19

Welcome to the Jungle

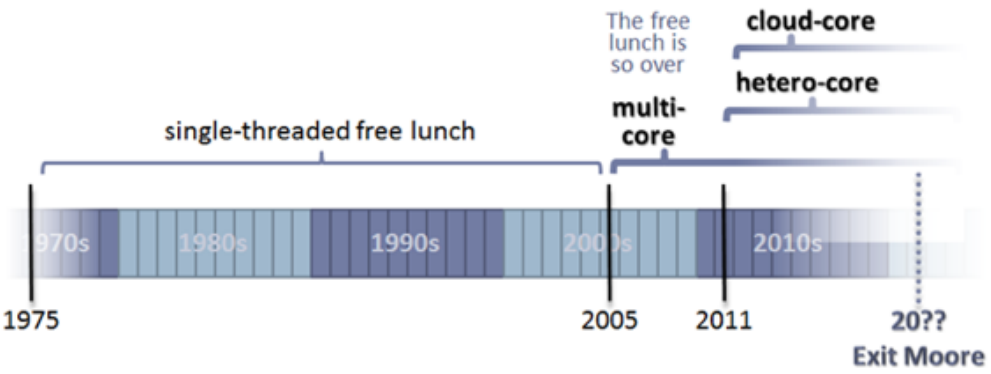
"In the twilight of Moore's Law, the transitions to multicore processors, GPU computing, and HaaS cloud computing are not separate trends, but aspects of a single trend – mainstream computers from desktops to 'smartphones' are being permanently transformed into heterogeneous supercomputer clusters. Henceforth, a single compute-intensive application will need to harness different kinds of cores, in immense numbers, to get its job done.

The free lunch is over. Now welcome to the hardware jungle."

Herb Sutter, *Welcome to the Jungle*, 2012

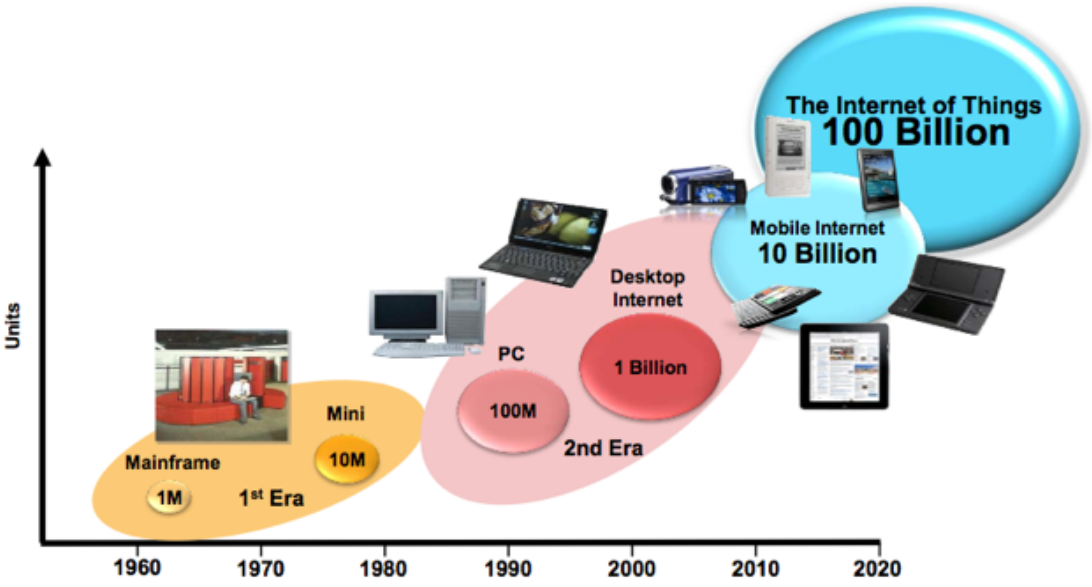
20

Computação heterogênea



Herb Sutter, *Welcome to the Jungle*, 2012

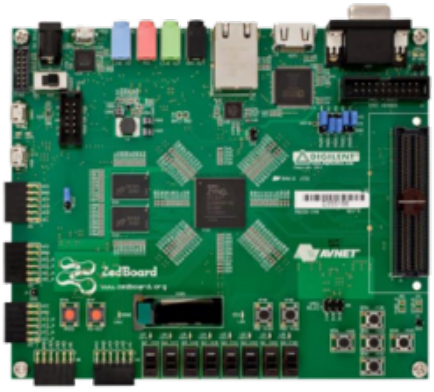
As eras da computação



Jem Davies, *Compute Power with Energy-Efficiency*, ARM, 2013

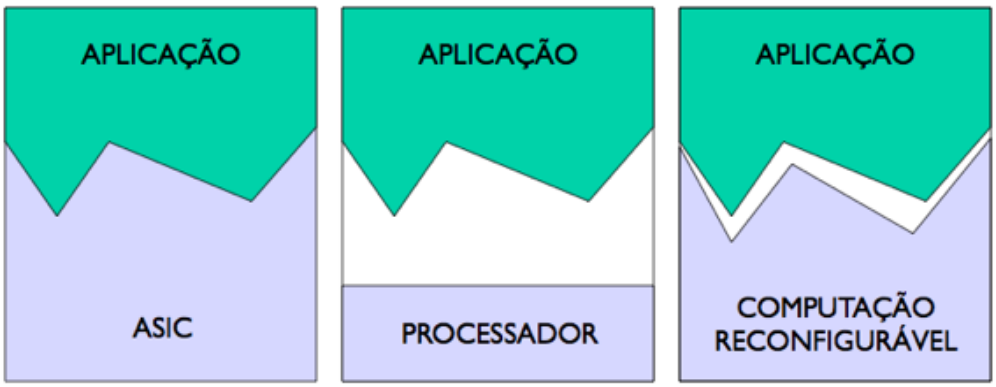
Sistemas Embarcados (IoT)

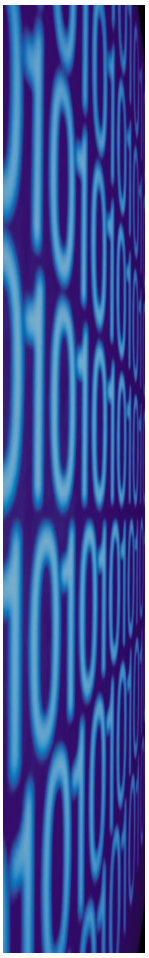
- Texas 66AK2H
- Zynq-7000



ZedBoard.org™ microZed™ picoZed™

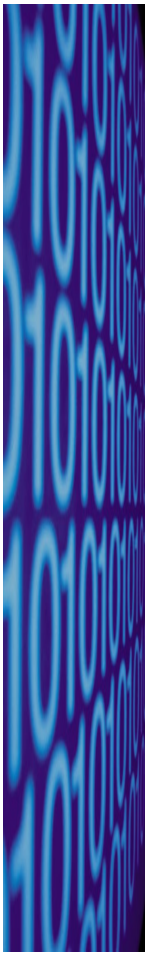
Computação Reconfigurável





Computação Reconfigurável

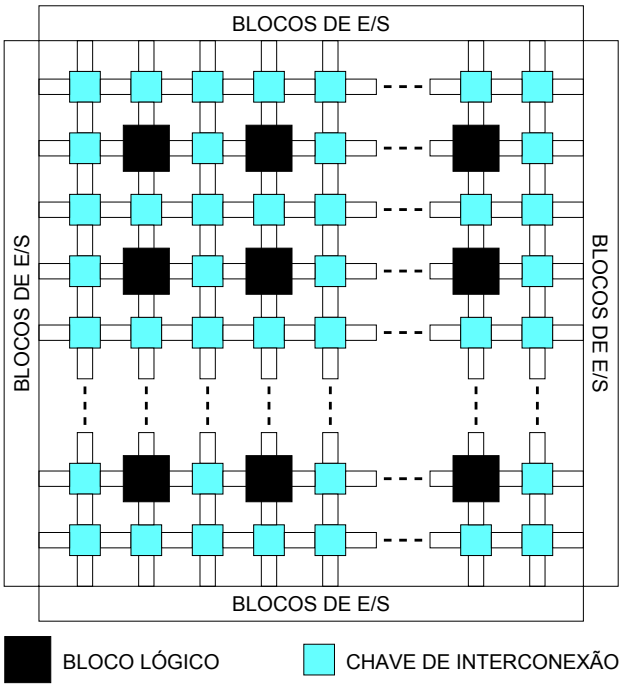
- Field-programmable gate array (FPGA)
  - Principal dispositivo desta categoria
- Vantagens
  - Desempenho
    - Próximo ao do hardware dedicado
  - Flexibilidade
    - O chip pode ser “reprogramado” a partir de modificações no projeto



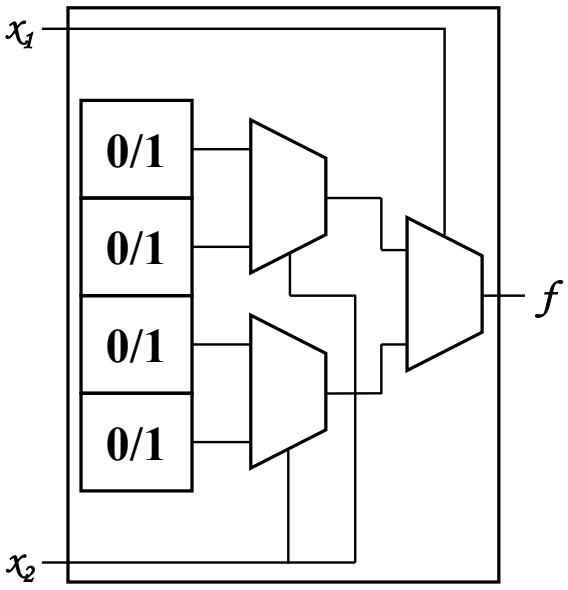
Computação Reconfigurável

- Recursos do dispositivos podem ser configurados para formar uma arquitetura específica;
- Conexão entre elementos formam um caminho de dados personalizado;
- Memórias on-chip configuráveis;
- Vários modelos de execução possíveis simultaneamente.

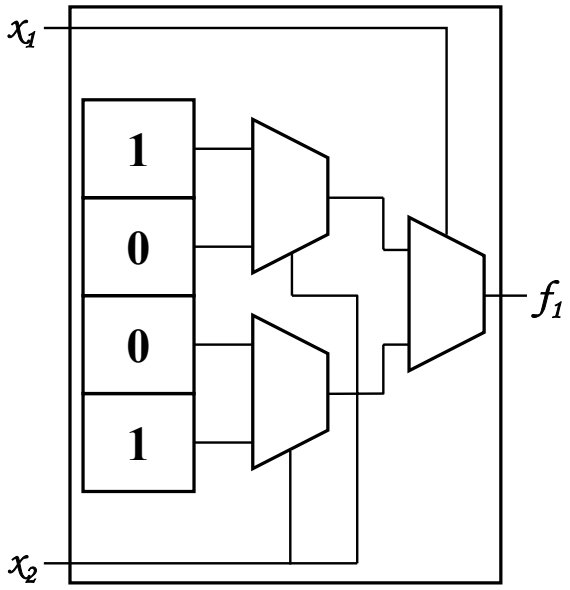
Field-Programmable Gate Array
Visão Geral



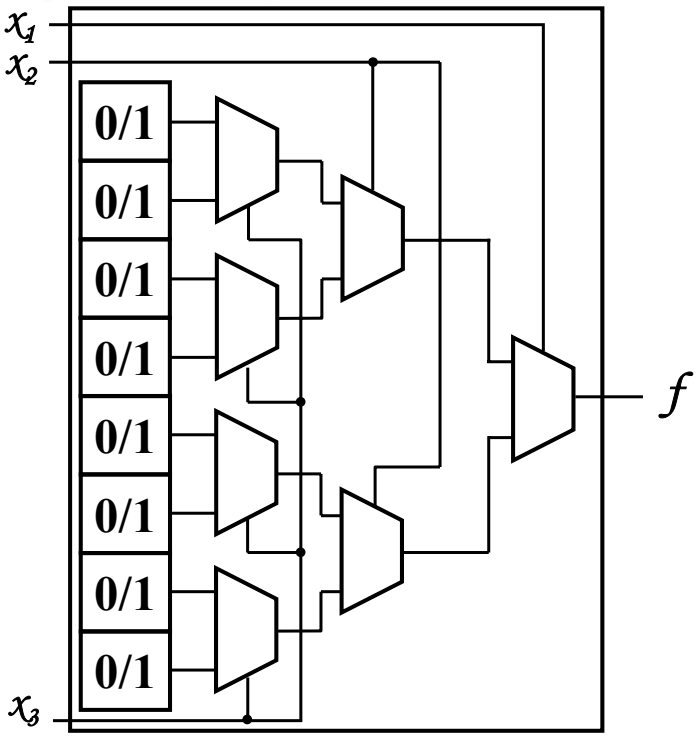
Field-Programmable Gate Array
Look-Up Table (LUT)



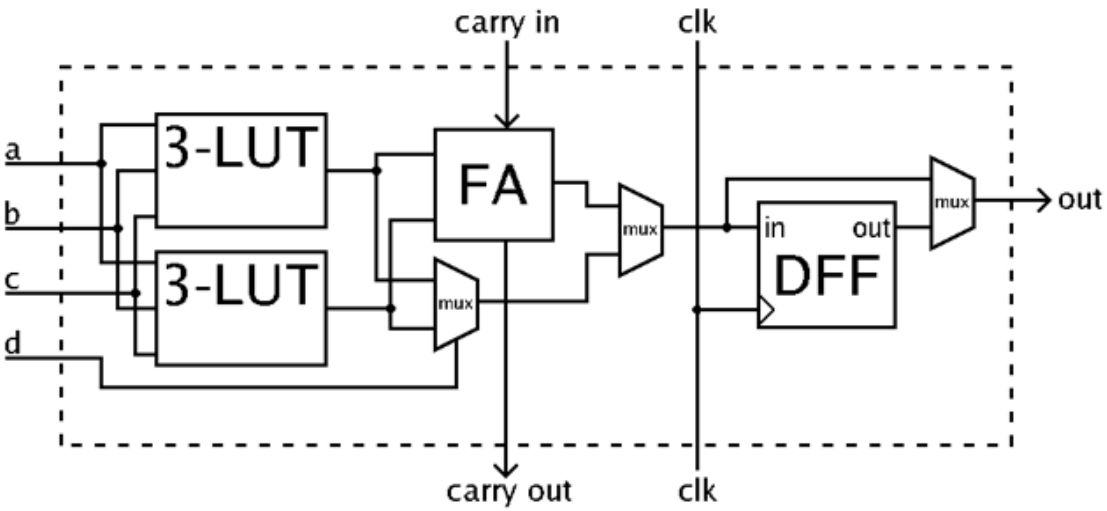
Field-Programmable Gate Array Look-Up Table (LUT)



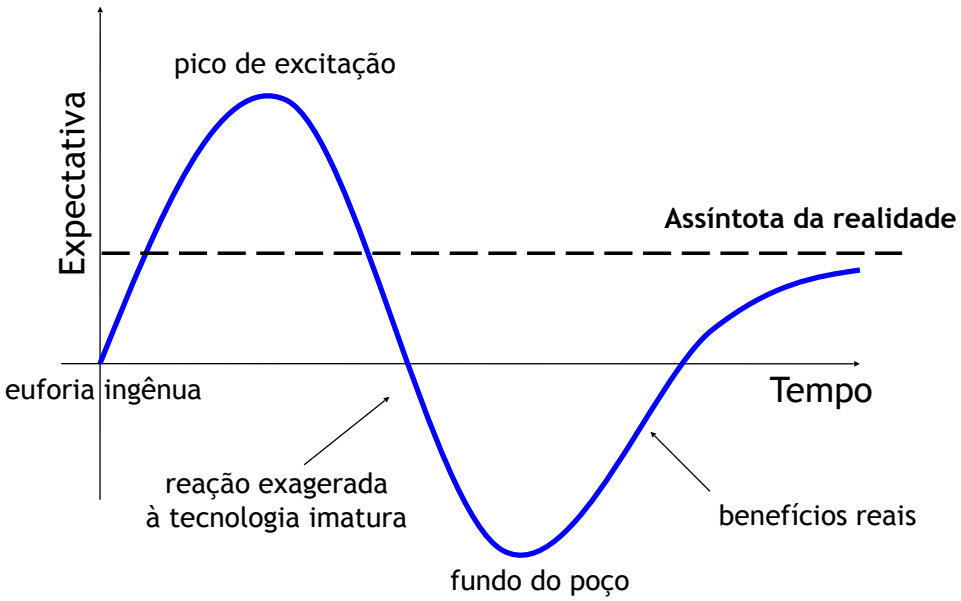
Field-Programmable Gate Array Look-Up Table (LUT)



Field-Programmable Gate Array
Configurable Logic Block (CLB)



Novas tecnologias



Bezdek, J.C, Fuzzy models - what are they, and why, IEEE Trans. on Fuzzy Systems, 1993.

Good news!

Forbes New Posts +15 Most Popular Lists Video 10 Stocks to Buy Now Search

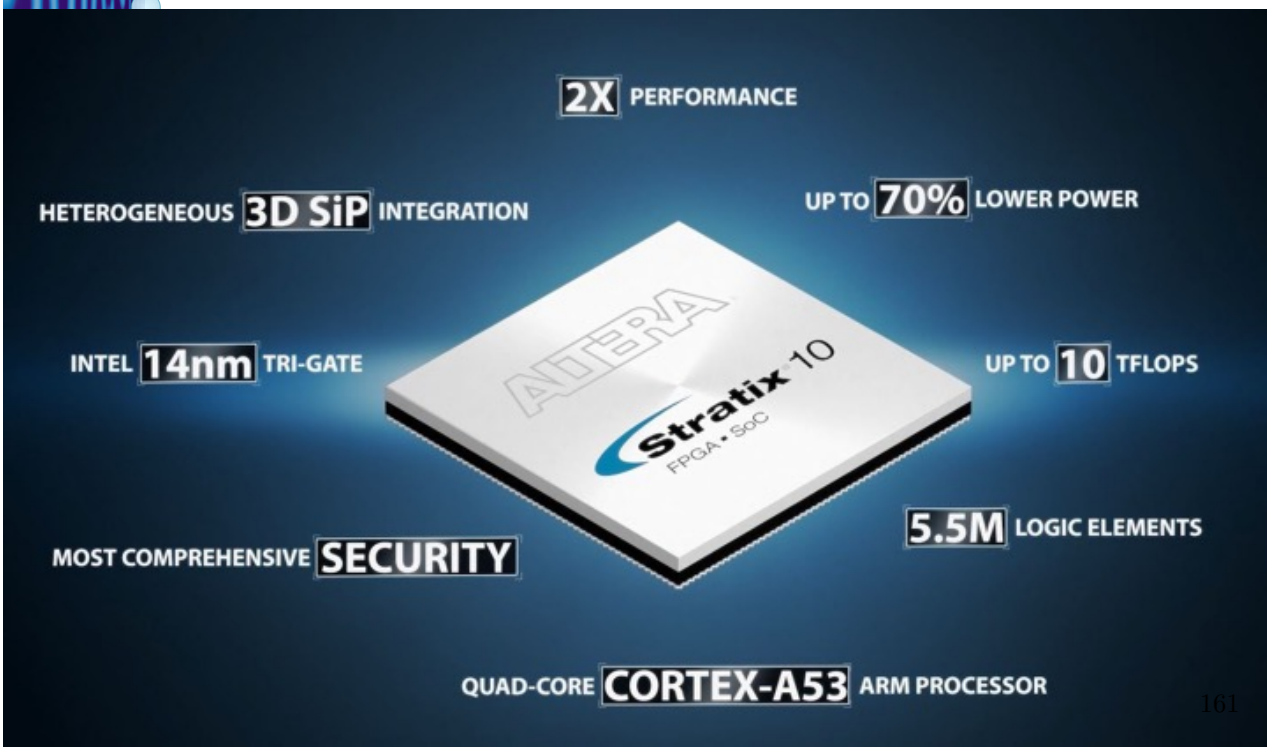
INVESTING 6/01/2015 @ 9:26AM | 3,869 views

Intel Buying Chipmaker Altera For $16.7 Billion

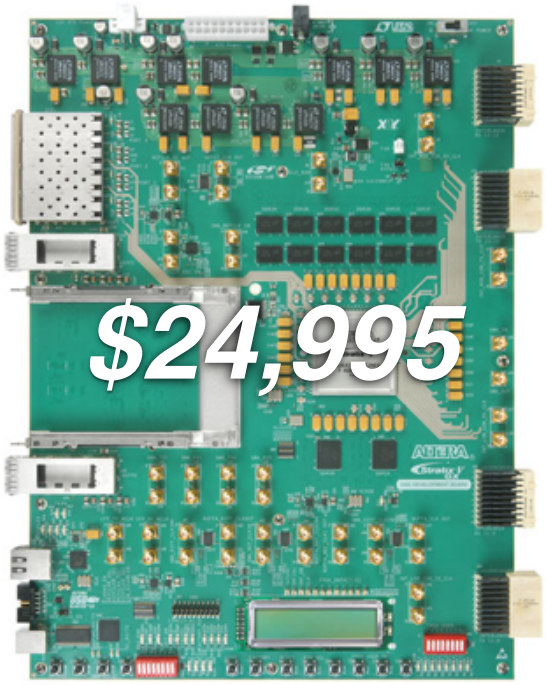
+ Comment Now + Follow Comments

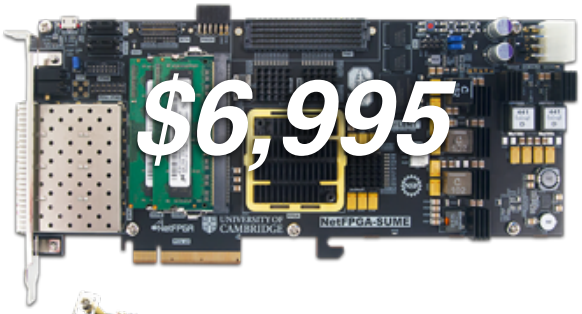
Share

Field-Programmable Gate Array Altera Stratix 10



Kits de desenvolvimento

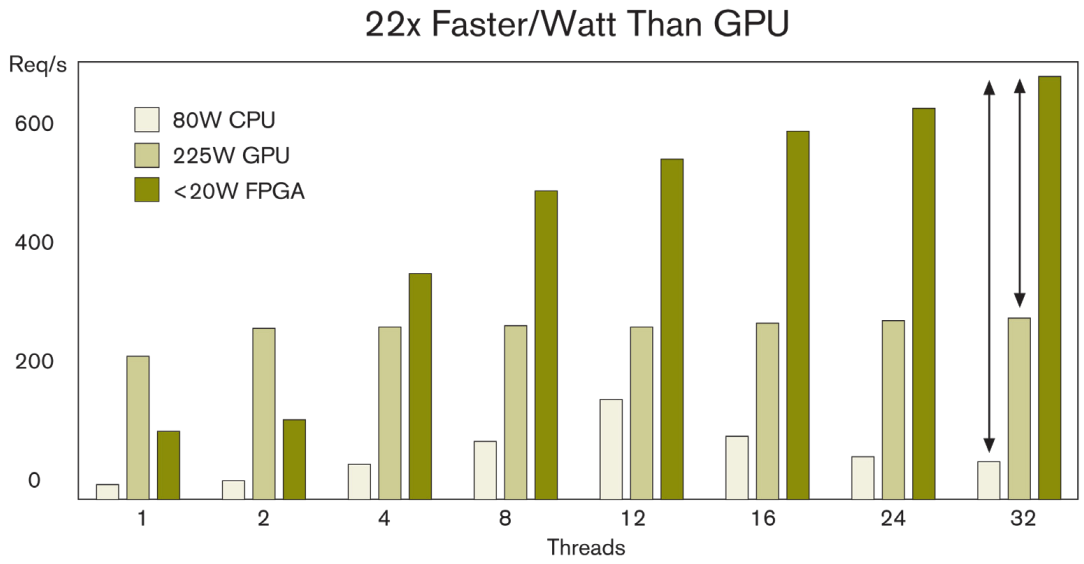




Aplicações

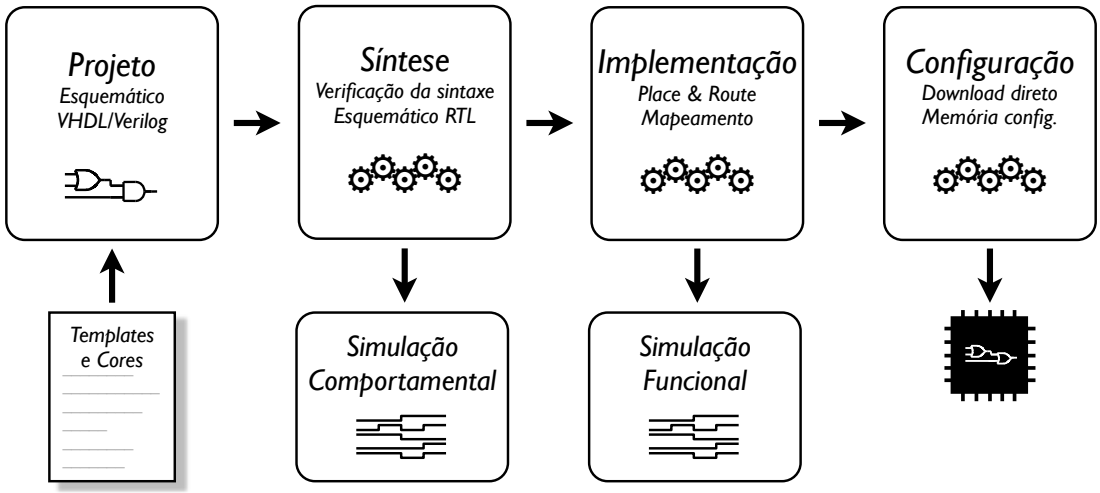
- Prototipação de ASICs
- Redes
- Criptografia
- Financeiras
- Bioinformática
- Petróleo & Gás
- Radioastronomia
- Imagens Médicas
- Visão Computacional
- Computação Cognitiva
- Reconhecimento de Voz

Performance/Watt

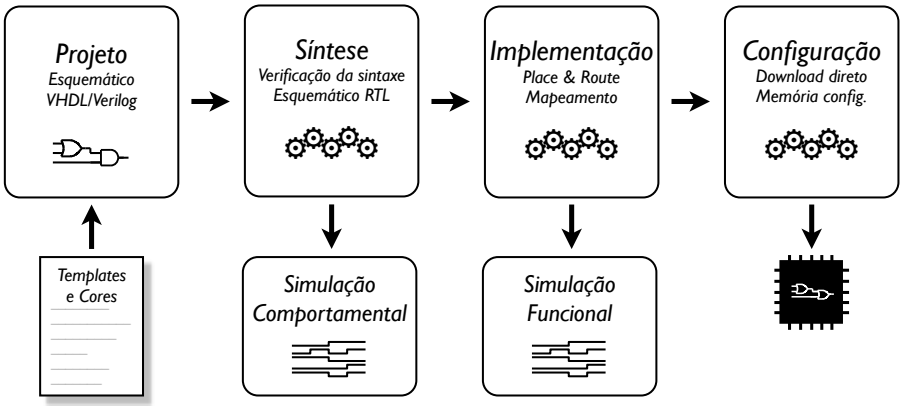


FPGAs provide significantly more hardware acceleration performance/watt
http://www.xilinx.com/products/design-tools/software-zone/sdaccel.html

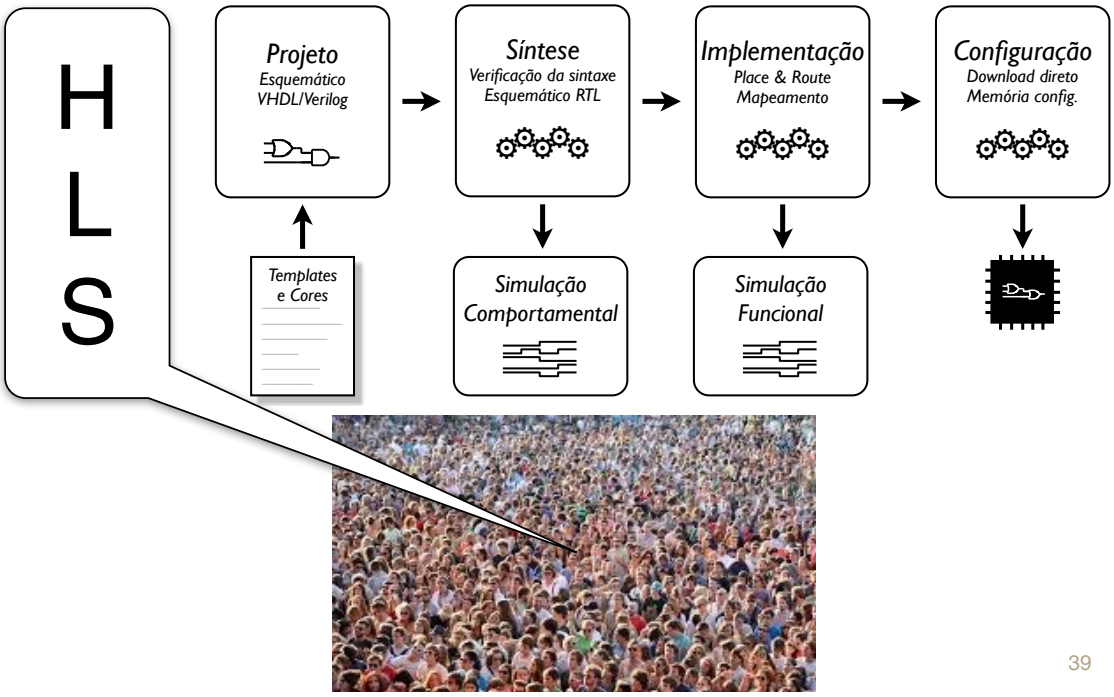
Fluxo de desenvolvimento



Fluxo de desenvolvimento



Fluxo de desenvolvimento



Desenvolvimento

- High-Level Synthesis (HLS)
  - Electronic System-Level (ESL)
- Hardware Description Language (HDL)
  - VHDL & Verilog
- Register-Transfer Level (RTL)
  - FF, Latches e transições entre eles
- Gate-Level

Técnicas de codificação

XILINX®

Chapter 2

XST HDL Coding Techniques

This chapter (*XST HDL Coding Techniques*) gives Hardware Description Language (HDL) coding examples for digital logic circuits. This chapter includes:

Most sections include:

- A general description of the macro
- A sample log file

XILINX®

Técnicas de codificação

Chap

XST HDL Coding Techniques

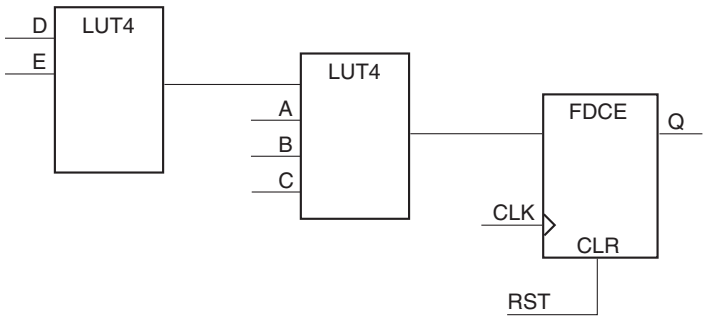
This chapter (*XST HDL Coding Techniques*) gives Hardware Description Langua coding examples for digital logic circuits. This chapter includes:

42

Técnicas de codificação

VHDL	Verilog

`process (CLK, RST)`
`begin`
`  if (RST = '1') then`
`  Q <= '0';`
`  elsif (rising_edge(clk)) then`
`  Q <= A or (B and C and D and E);`
`  end if;`
`end process;`
	`always @(posedge CLK, posedge RST)`
	`  if (RESET)`
	`  Q <= 1'b0;`
	`  else`
	`  Q <= A | (B & C & D & E);`

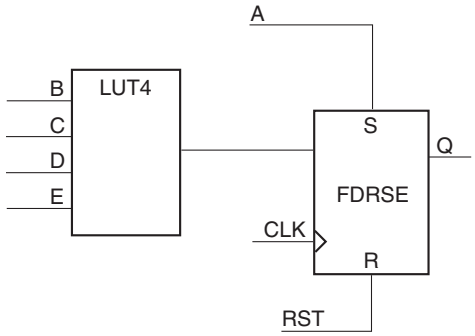


43

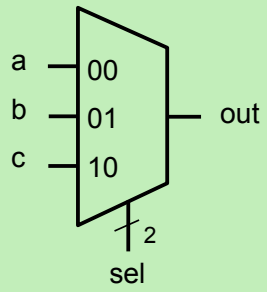
Técnicas de codificação

VHDL	Verilog

process (CLK)
begin
  if (rising_edge(clk)) then
  if (RST = '1') then
  Q <= '0';
  else
  Q <= A or (B and C and D and E);
  end if;
  end if;
end process;
	always @(posedge CLK)
	  if (RESET)
	  Q <= 1'b0;
	  else
	  Q <= A | (B & C & D & E);



Especificação incompleta

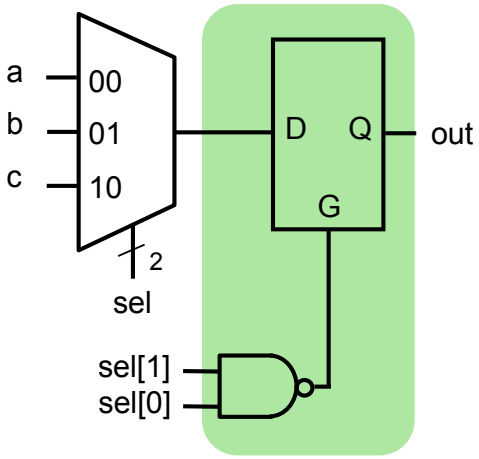


3-to-1 MUX
('11' input is a don't-care)

```
module maybe_mux_3to1(a, b, c,
                      sel, out);
  input [1:0] sel;
  input a,b,c;
  output out;
  reg out;

  always @(a or b or c or sel)
  begin
    case (sel)
      2'b00: out = a;
      2'b01: out = b;
      2'b10: out = c;
    endcase
  end
endmodule
```

http://web.mit.edu/6.111/www/f2007/handouts/L04.pdf

Especificação incompleta




```
module maybe_mux_3to1(a, b, c,
                      sel, out);
  input [1:0] sel;
  input a,b,c;
  output out;
  reg out;

  always @(a or b or c or sel)
  begin
    case (sel)
      2'b00: out = a;
      2'b01: out = b;
      2'b10: out = c;
    endcase
  end
endmodule
```

http://web.mit.edu/6.111/www/f2007/handouts/L04.pdf

Especificação incompleta

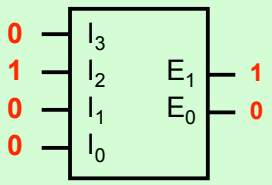
```
always @(a or b or c or sel)
  begin
    out = 1'bx;
    case (sel)
      2'b00: out = a;
      2'b01: out = b;
      2'b10: out = c;
    endcase
  end
endmodule
```

```
always @(a or b or c or sel)
  begin
  case (sel)
      2'b00: out = a;
      2'b01: out = b;
      2'b10: out = c;
      default: out = 1'bx;
    endcase
  end
endmodule
```


http://web.mit.edu/6.111/www/f2007/handouts/L04.pdf

Prioridades das atribuições

4-to-2 Binary Encoder



I_3 I_2 I_1 I_0	E_1 E_0
0 0 0 1	0 0
0 0 1 0	0 1
0 1 0 0	1 0
1 0 0 0	1 1
all others	X X

```
module binary_encoder(i, e);
  input [3:0] i;
  output [1:0] e;
  reg e;

  always @(i)
  begin
    if (i[0]) e = 2'b00;
    else if (i[1]) e = 2'b01;
    else if (i[2]) e = 2'b10;
    else if (i[3]) e = 2'b11;
    else e = 2'bxx;
  end
endmodule
```

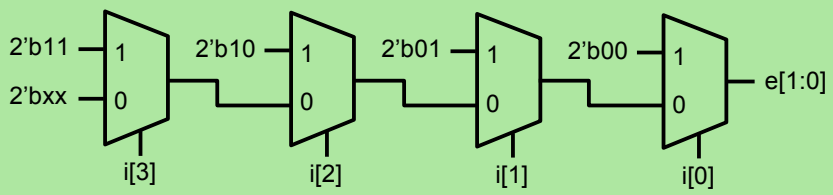
http://web.mit.edu/6.111/www/f2007/handouts/L04.pdf

Prioridades das atribuições

I_3 I_2 I_1 I_0	E_1 E_0
0 0 0 1	0 0
0 0 1 0	0 1
0 1 0 0	1 0
1 0 0 0	1 1
all others	X X

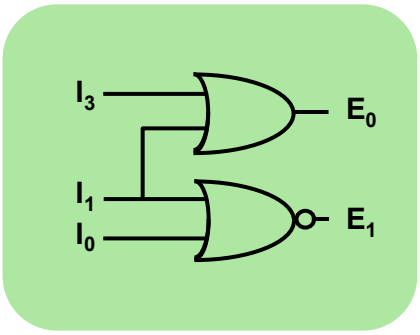
```
if (i[0]) e = 2'b00;
else if (i[1]) e = 2'b01;
else if (i[2]) e = 2'b10;
else if (i[3]) e = 2'b11;
else e = 2'bxx;
end
```

Inferred Result:



http://web.mit.edu/6.111/www/f2007/handouts/L04.pdf

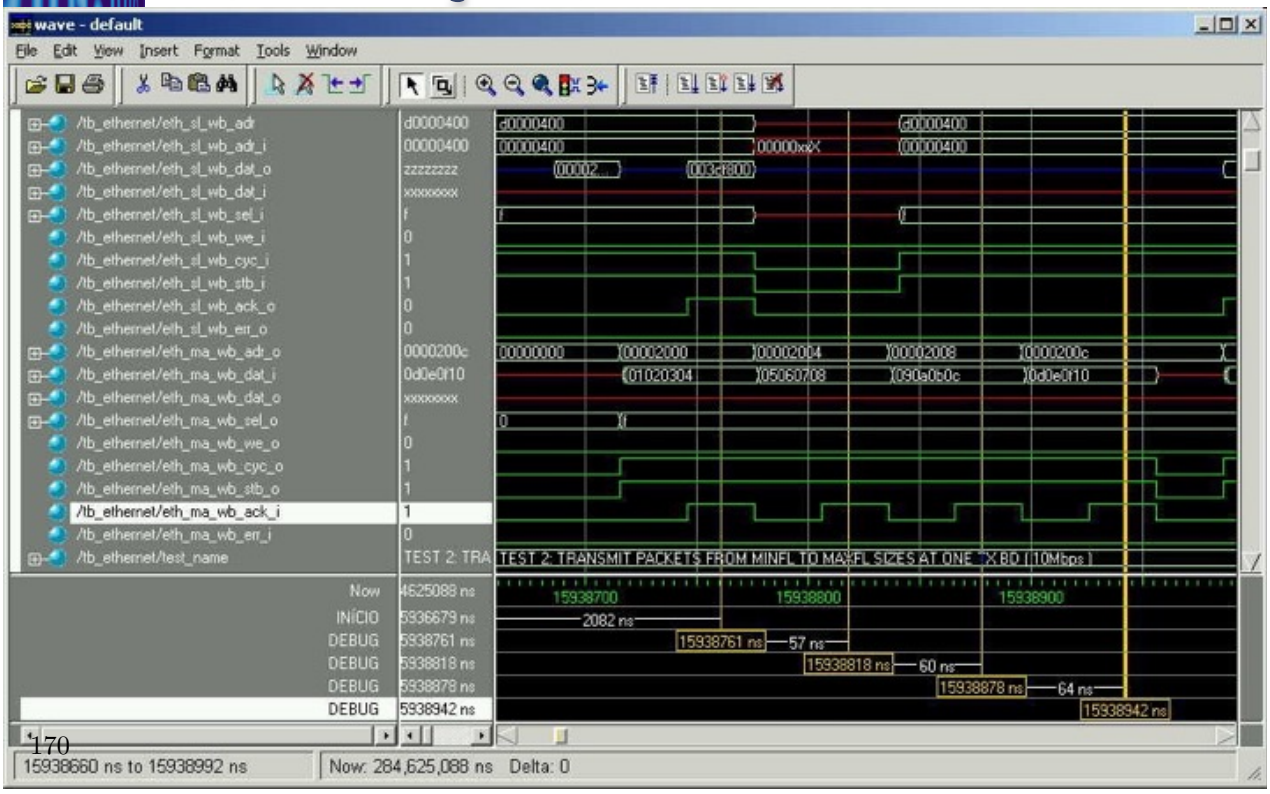
Prioridades das atribuições



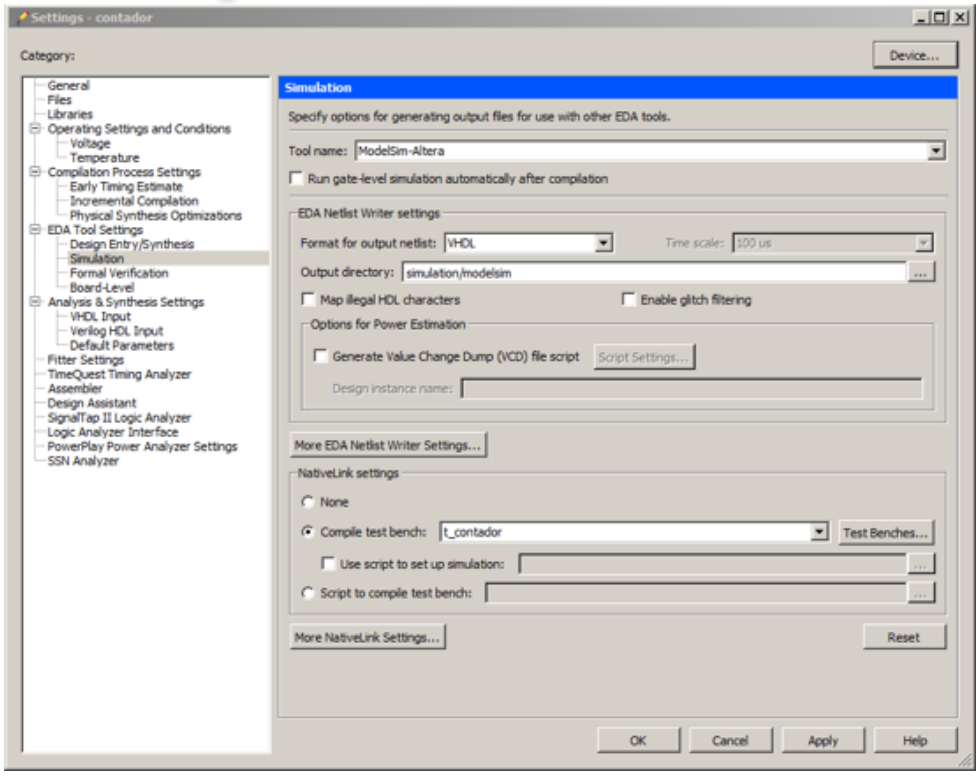

```
module binary_encoder(i, e);
  input [3:0] i;
  output [1:0] e;
  reg e;

  always @(i)
  begin
    if (i == 4'b0001) e = 2'b00;
    else if (i == 4'b0010) e = 2'b01;
    else if (i == 4'b0100) e = 2'b10;
    else if (i == 4'b1000) e = 2'b11;
    else e = 2'bxx;
  end
endmodule
```

http://web.mit.edu/6.111/www/f2007/handouts/L04.pdf

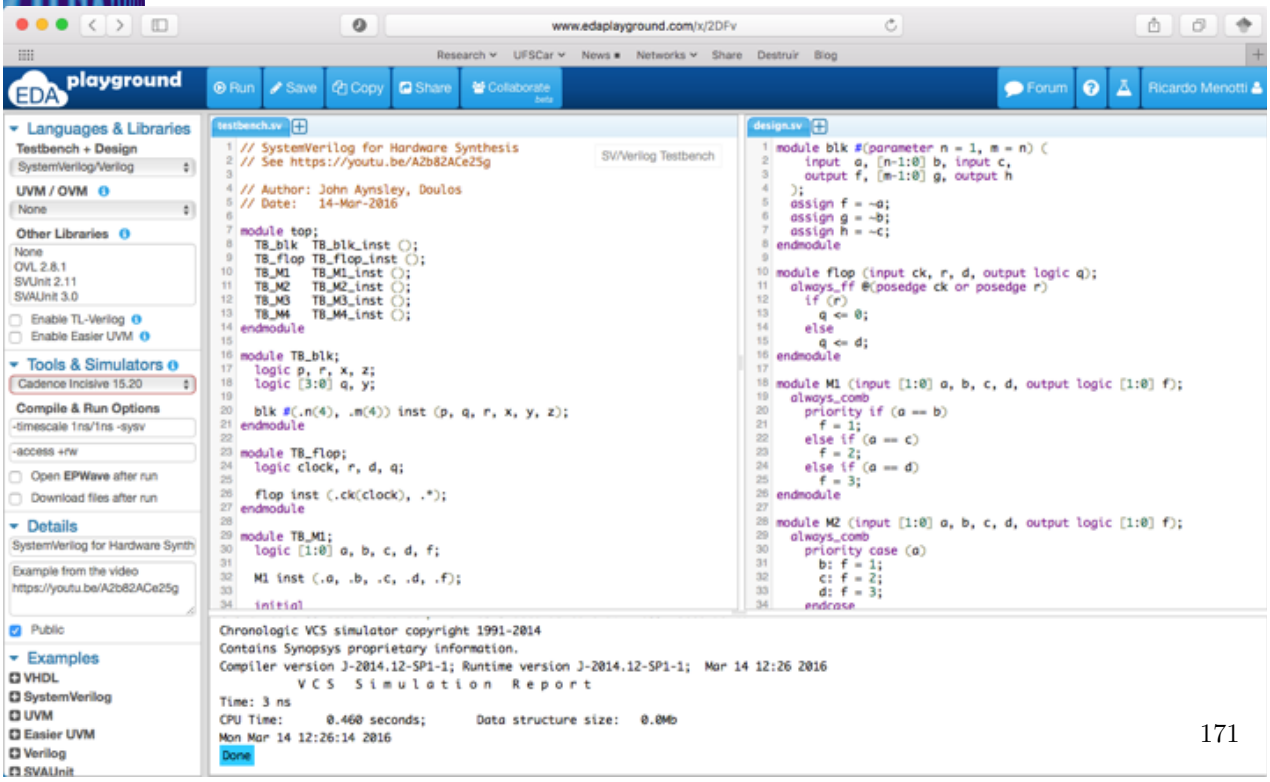
Simulações



Simulações



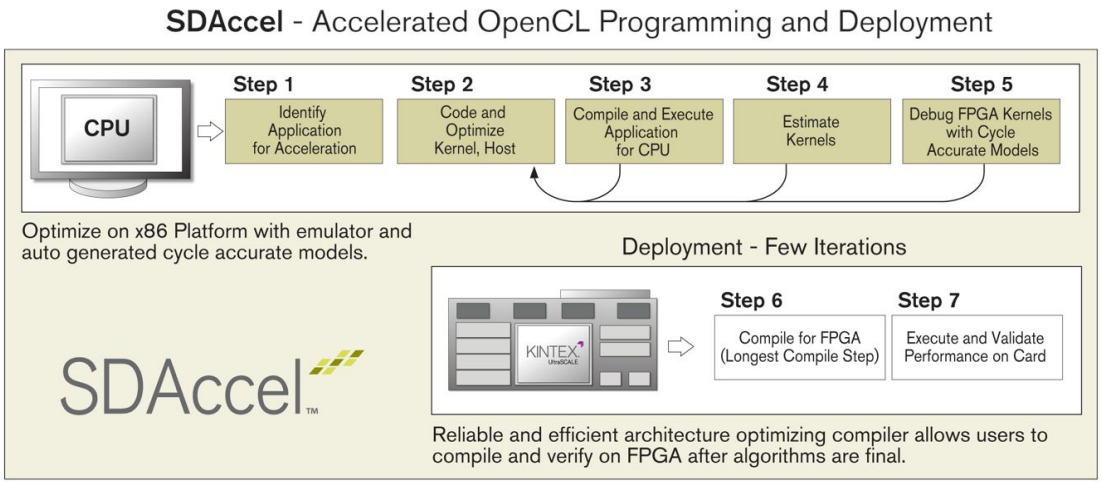
Ferramentas online



Field-Programmable Gate Array Como aceleradores

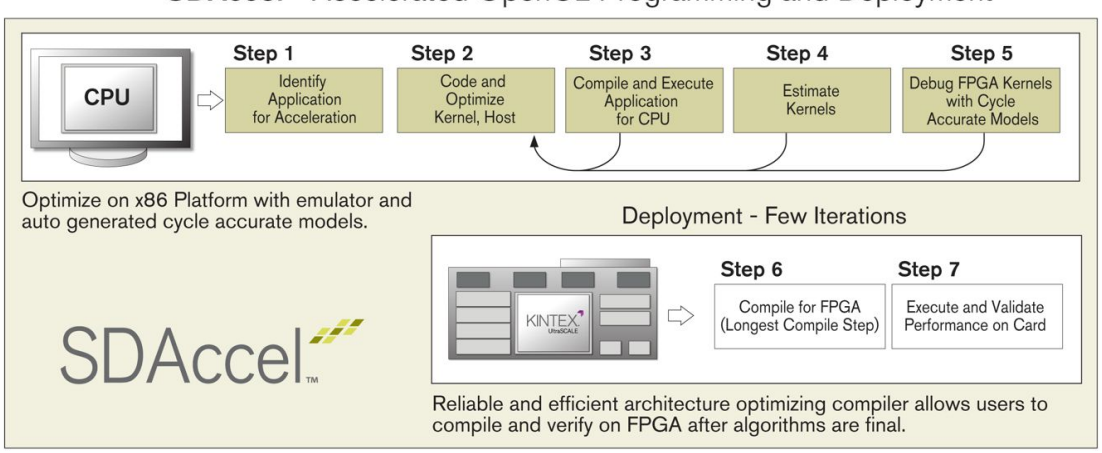
- CPU/GPU — arquitetura fixa
  - compilador mapeia os kernels
  - opcional — tempo de execução (jit)
- FPGA — arquitetura configurável
  - compilador gera arquiteturas correspondentes a cada kernel
  - offline — tempo de síntese proibitivo
- ↑ Complexidade — ↓ Abstração
- Compilação/otimizações mais complexas
- Informações adicionais no código

Xilinx HLS (SDAccel)



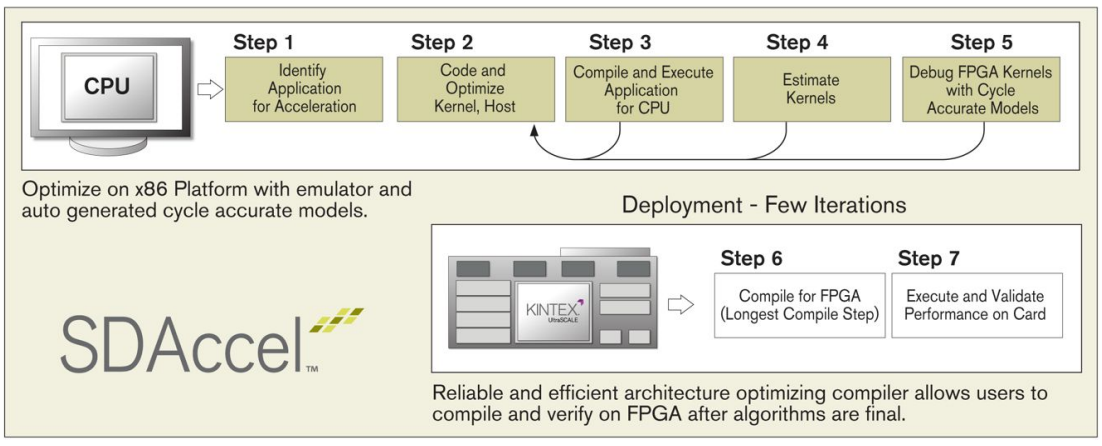
Xilinx HLS (SDAccel)

"…enables concurrent programming of the system processor and the FPGA logic and requires no RTL design experience"



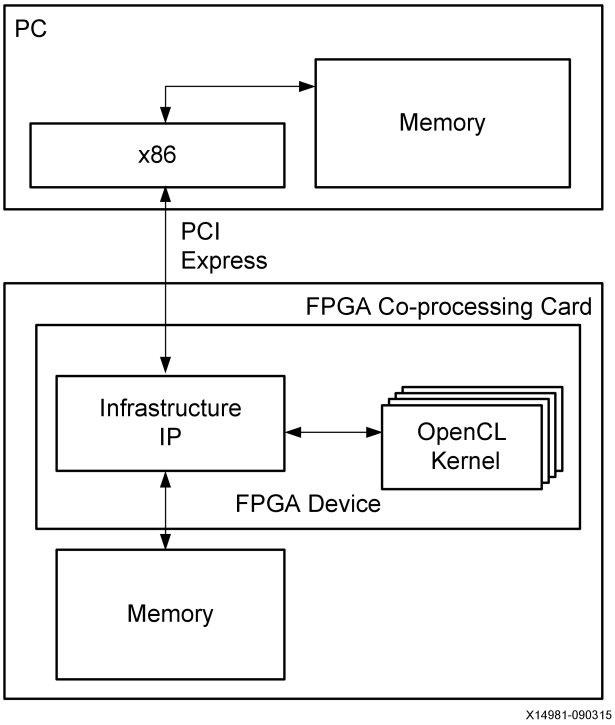
Xilinx HLS (SDAccel)

"…enables concurrent programming of the system processor and the FPGA logic and requires no RTL design experience"



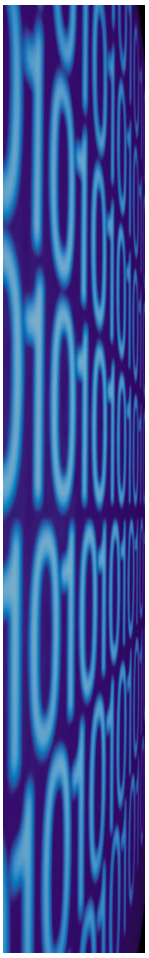
IMPORTANT: *The SDAccel environment has no concept of what a user application does or what constitutes correct behavior. The programmer must check application results for correctness.*

Xilinx HLS (SDAccel)



Xilinx HLS (SDAccel)

- Infra. IP cores requeridos (PCIe&DDR)
  - Comunicação limitada com o FPGA
- Modelo de memória:
  - Global & Cons. Mem. — DDR
  - Local Mem. — Block RAM
  - Priv. Mem. — Registradores

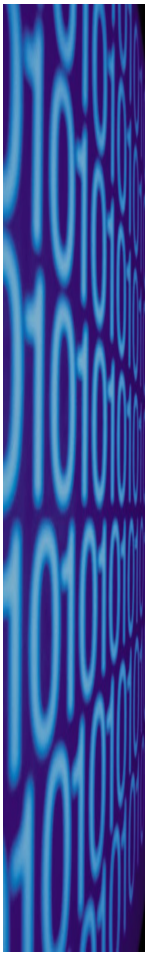


Xilinx HLS (SDAccel)

- Emulação CPU & Acc (precisão de ciclos)
- Smith-Waterman Algorithm
- 200 MHz
- Start Interval:
  - Best: 166.424
  - Worst: 202.124
  - Average: 187.843
- Resources:
  - FFs: 2095
  - LUTs: 3180
  - DSP: 1
  - BRAMs: 11

Após otimizações:

- Start Interval:
  - Fixed: 21.817
- Resources:
  - FFs: 2428
  - LUTs: 3430
  - DSP: 1
  - BRAMS: 11



Xilinx HLS (SDAccel)

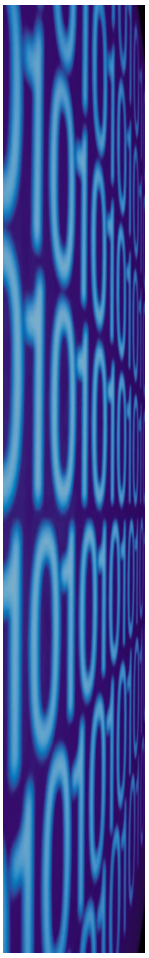
```
#ifdef __xilinx__
    __attribute__((xcl_pipeline_loop))
#endif
    for (int i = 1; i < N; i++) {
        localS1[i] = s1[i];
    }
#ifdef __xilinx__
    __attribute__((xcl_pipeline_loop))
#endif
    for (int i = 1; i < N; i++) {
        localS2[i] = s2[i];
    }
#ifdef __xilinx__
    __attribute__((xcl_pipeline_loop))
#endif
    for (int i = 0; i < N * N; i++) {
        localMatrix[i] = 0;
    }
...
```

Xilinx HLS (SDAccel)

```
...
#ifdef __xilinx__
    __attribute__((xcl_pipeline_loop))
#endif
    for (short index = N; index < N * N; index++)
    {
        short dir = CENTER;
        short val = 0;
        short j = index % N;
        if (j == 0) { // Skip the first column
            west = 0;
            northwest = 0;
            continue;
        }
        short i = index / N;
        short2 temp = localMatrix[index - N];
        north = temp.x;
        const short match = (localS1[j] == localS2[i]) ?
                                            MATCH : MISS_MATCH;
        short val1 = northwest + match;
        if (val1 > val) {
            val = val1;
            dir = NORTH_WEST;
        }
```

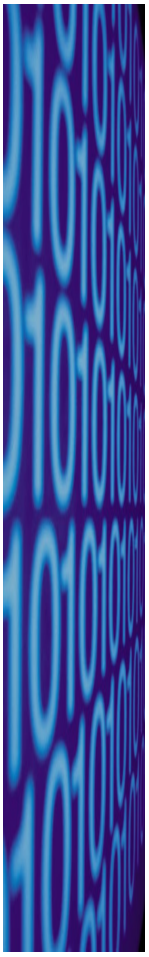

Altera HLS (AOCL)

```
[menotti@MyDocMyHell altera]$ aocl diagnose
aocl diagnose: Running diagnostic from /opt/altera/14.1/hld/board/s5phq/
linux64/libexec

Verified that the kernel mode driver is installed on the host machine.

Using board package from vendor: BittWare Inc
Querying information for all supported devices that are installed on the
host machine ...


Device Name   Status   Information

acl0          Passed   S5PHQ
                       PCIe dev_id = D800, bus:slot.func = 04:00.00, at
Gen 2 with 8 lanes
                       FPGA temperature = 48 degrees C.


Found 1 active device(s) installed on the host machine. To perform a
full diagnostic on a specific device, please run
      aocl diagnose <device_name>

DIAGNOSTIC_PASSED
```



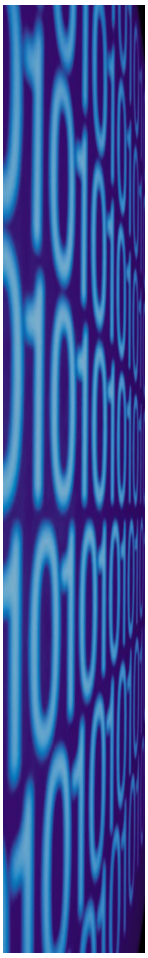
Altera HLS (AOCL)

```
[menotti@MyDocMyHell hello_world]$ aoc -march=emulator
-v --board s5phq_a7 device/hello_world.cl -o bin/
hello_world_emulation.aocx

aoc: Environment checks are completed successfully.
You are now compiling the full flow!!
aoc: Selected target board s5phq_a7
aoc: Running OpenCL parser....
aoc: OpenCL parser completed successfully.
aoc: Compiling for Emulation ....
aoc: Emulator Compilation completed successfully.
Emulator flow is successful.
To execute emulated kernel, invoke host with
   env CL_CONTEXT_EMULATOR_DEVICE_ALTERA=1
<host_program>
 For multi device emulations replace the 1 with the
number of devices you which to emulate
```

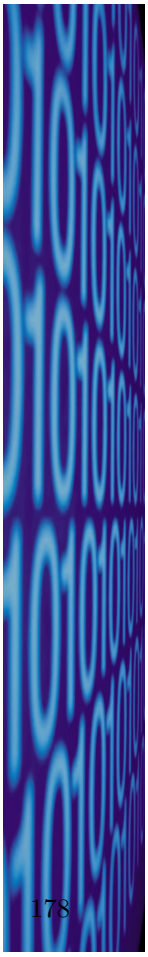
Altera HLS (AOCL)

```
[menotti@MyDocMyHell bin]$ env
CL_CONTEXT_EMULATOR_DEVICE_ALTERA=4 ./hello_world
Querying platform for info:
==========================
CL_PLATFORM_NAME    = Altera SDK for OpenCL
CL_PLATFORM_VENDOR  = Altera Corporation
CL_PLATFORM_VERSION = OpenCL 1.0 Altera SDK for
OpenCL, Version 15.1.2
```



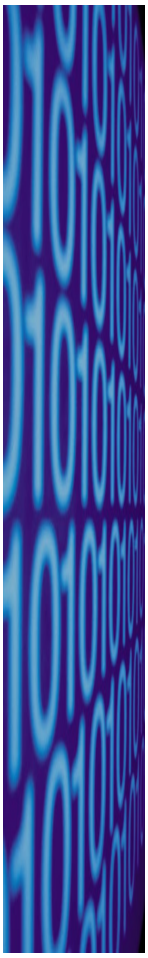
Altera HLS (AOCL)

```
Querying device for info:
========================
CL_DEVICE_NAME                           = EmulatorDevice : Emulated Device
CL_DEVICE_VENDOR                         = Altera Corporation
CL_DEVICE_VENDOR_ID                      = 4466
CL_DEVICE_VERSION                        = OpenCL 1.0 Altera SDK for OpenCL, Version 15.1.2
CL_DRIVER_VERSION                        = 15.1
CL_DEVICE_ADDRESS_BITS                   = 64
CL_DEVICE_AVAILABLE                      = true
CL_DEVICE_ENDIAN_LITTLE                  = true
CL_DEVICE_GLOBAL_MEM_CACHE_SIZE          = 32768
CL_DEVICE_GLOBAL_MEM_CACHELINE_SIZE      = 0
CL_DEVICE_GLOBAL_MEM_SIZE                = 8589934592
CL_DEVICE_IMAGE_SUPPORT                  = true
CL_DEVICE_LOCAL_MEM_SIZE                 = 16384
CL_DEVICE_MAX_CLOCK_FREQUENCY            = 1000
CL_DEVICE_MAX_COMPUTE_UNITS              = 1
CL_DEVICE_MAX_CONSTANT_ARGS              = 8
CL_DEVICE_MAX_CONSTANT_BUFFER_SIZE       = 2147483648
CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS       = 3
CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS       = 8192
CL_DEVICE_MIN_DATA_TYPE_ALIGN_SIZE       = 1024
CL_DEVICE_PREFERRED_VECTOR_WIDTH_CHAR    = 4
CL_DEVICE_PREFERRED_VECTOR_WIDTH_SHORT   = 2
CL_DEVICE_PREFERRED_VECTOR_WIDTH_INT     = 1
CL_DEVICE_PREFERRED_VECTOR_WIDTH_LONG    = 1
CL_DEVICE_PREFERRED_VECTOR_WIDTH_FLOAT   = 1
CL_DEVICE_PREFERRED_VECTOR_WIDTH_DOUBLE  = 0
Command queue out of order?              = false
Command queue profiling enabled?         = true
```

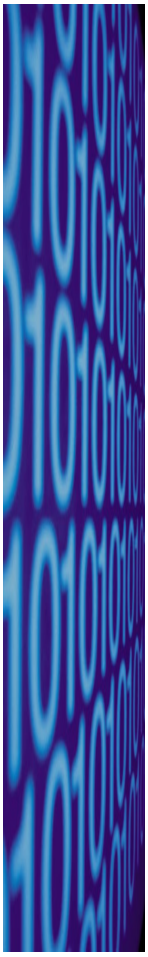
Altera HLS (AOCL)

```
Using AOCX: hello_world_emulation.aocx

Kernel initialization is complete.
Launching the kernel...

Thread #2: Hello from Altera's OpenCL Compiler!

Kernel execution is complete.
```



Altera HLS (AOCL)

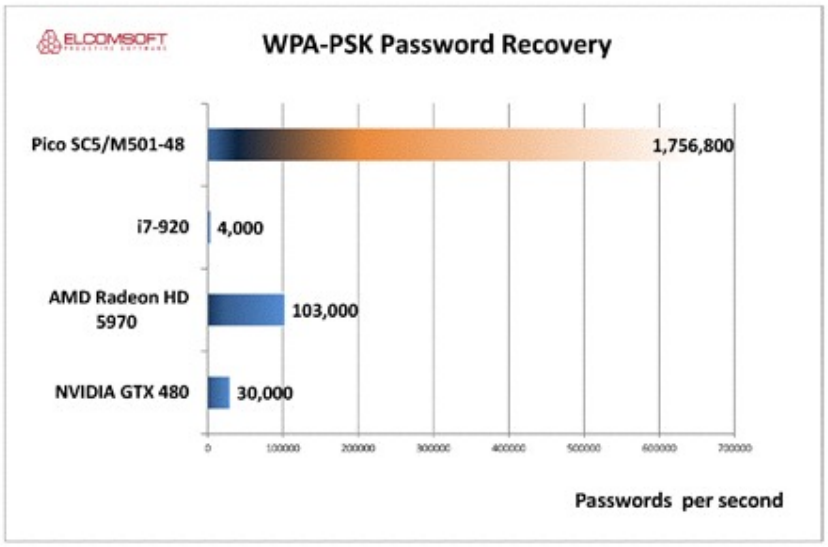
```
[menotti@MyDocMyHell hello_world]$ aoc -v --board
s5phq_a7 device/hello_world.cl -o bin/
hello_world.aocx
aoc: Environment checks are completed successfully.
You are now compiling the full flow!!
aoc: Selected target board s5phq_a7
aoc: Running OpenCL parser....
aoc: OpenCL parser completed successfully.
aoc: Compiling....
aoc: Linking with IP library ...
aoc: First stage compilation completed successfully.
aoc: Hardware generation completed successfully.
```

Altera HLS (AOCL)

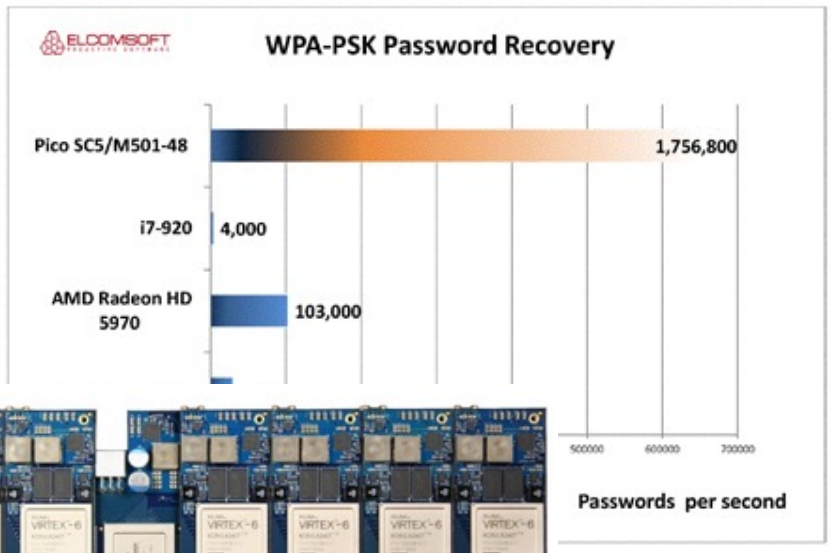
```
Using AOCX: hello_world.aocx
Reprogramming device with handle 1

Kernel initialization is complete.
Launching the kernel...

Thread #2: Hello from Altera's OpenCL Compiler!

Kernel execution is complete.
```

Password Crack



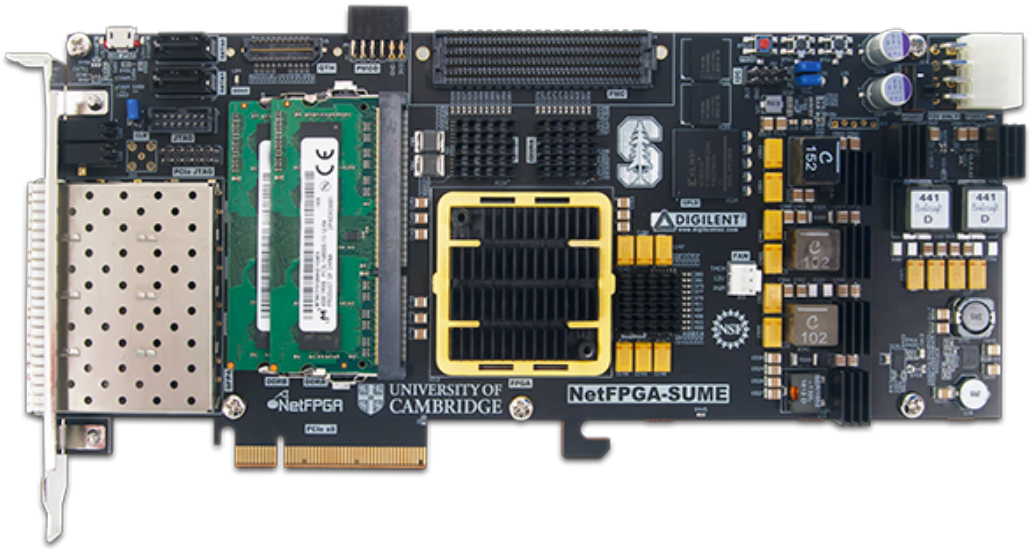
http://blog.crackpassword.com/2012/07/accelerating-password-recovery-the-addition-of-fpga/

Password Crack



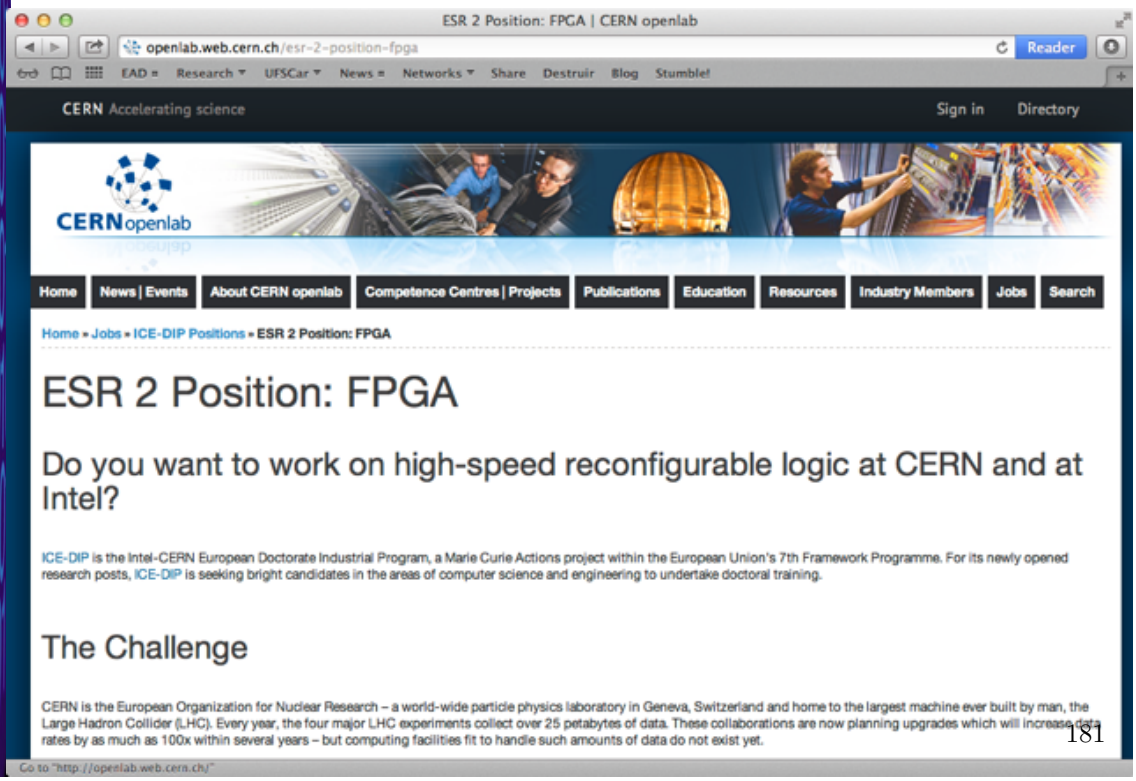
http://blog.crackpassword.com/2012/07/accelerating-password-recovery-the-addition-of-fpga/

NetFPGA



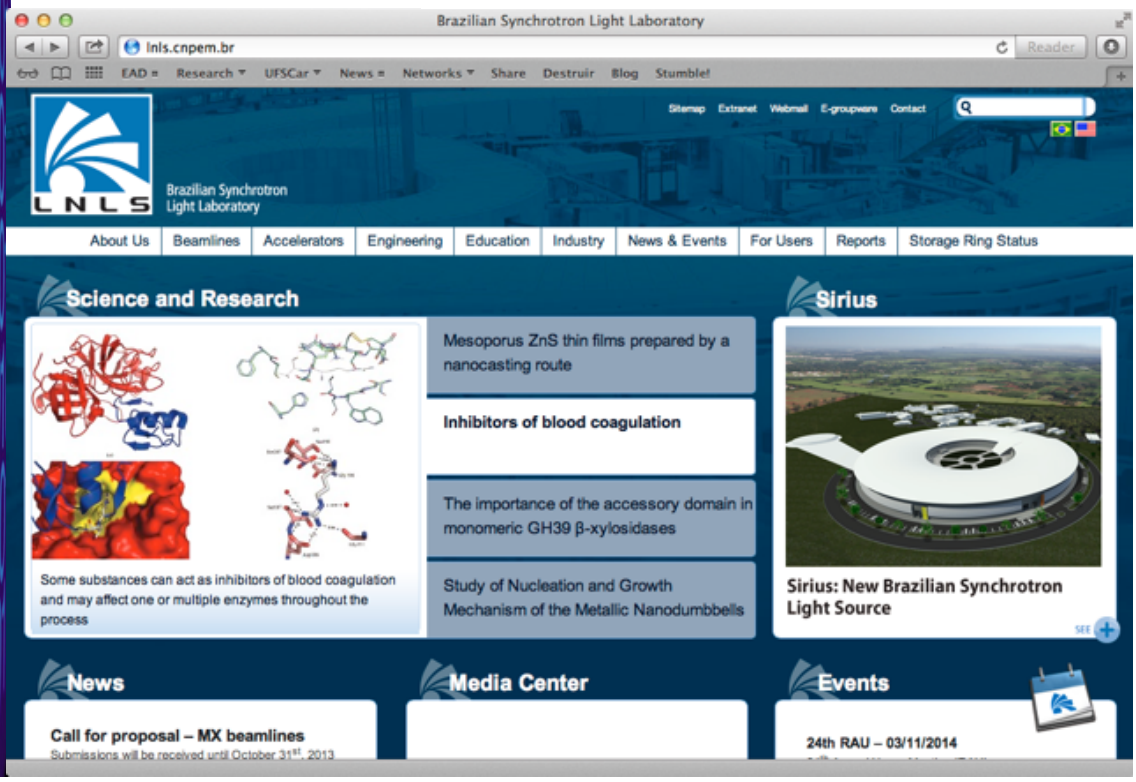
cisco Google DIGILENT BROADCOM. Synplicity Juniper NETWORKS
HUAWEI XILINX Agilent Technologies CYPRESS PERFORM Micron

CERN



LNLS

CNPEM
Centro Nacional de Pesquisa em Energia e Materiais



Catapult: Microsoft – ISCA2014

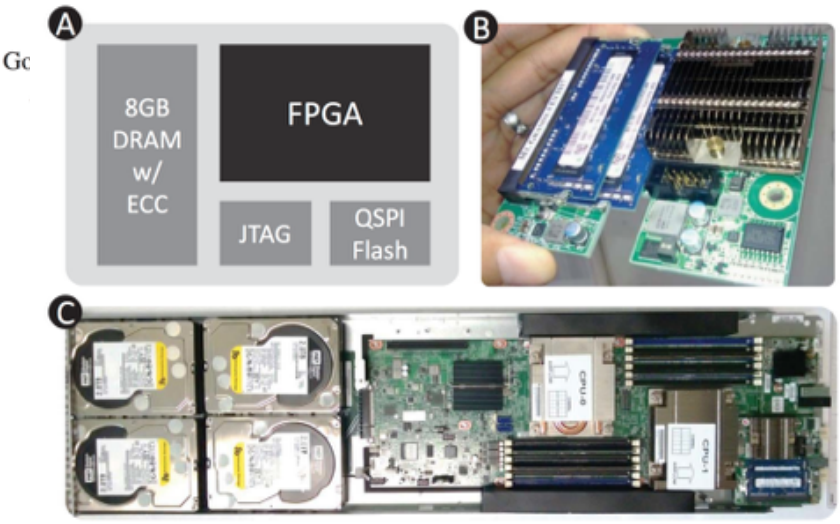
A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services

Andrew Putnam Adrian M. Caulfield Eric S. Chung Derek Chiou[1]
Kypros Constantinides[2] John Demme[3] Hadi Esmaeilzadeh[4] Jeremy Fowers
Gopi Prashanth Gopal Jan Gray Michael Haselman Scott Hauck[5] Stephen Heil
Amir Hormati[6] Joo-Young Kim Sitaram Lanka James Larus[7] Eric Peterson
Simon Pope Aaron Smith Jason Thong Phillip Yi Xiao Doug Burger

[1]Microsoft and University of Texas at Austin
[2]Amazon Web Services
[3]Columbia University
[4]Georgia Institute of Technology
[5]Microsoft and University of Washington
[6]Google, Inc.
[7]École Polytechnique Fédérale de Lausanne (EPFL)
All authors contributed to this work while employed by Microsoft.

Catapult: Microsoft – ISCA2014

A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services



[6]Google, Inc.
[7]École Polytechnique Fédérale de Lausanne (EPFL)
All authors contributed to this work while employed by Microsoft.

FPL 2016 (Keynotes)

Doug Burger

Microsoft, US

Configurable Clouds

Hyperscale clouds are already disrupting the industry in significant ways and are on their way to being a trillion-dollar market. Reconfigurable computing holds the promise of a transformation—not an augmentation—of cloud architecture. In this talk, I will describe the evolution of the Catapult cloud FPGA architecture developed at Microsoft, through its early prototypes, through previously published designs, to the current *v2* architecture that is successfully transforming our cloud architecture at large scale. I will show why we believe this particular design is disruptive, and will describe a few case studies about how Microsoft is using it to accelerate our both our services and our Azure cloud platform. I will conclude with some thoughts about how these *Configurable Clouds* may evolve and some of the future opportunities for large-scale reconfigurable computing in general.

FPL 2016 (Keynotes)

Gustavo Alonso

ETHZ, CH

Data Processing on the Fast Lane

Data processing is changing in radical ways. On the one had, data science and big data have brought an unprecedented growth and variety in data sizes, demanding workloads, data types, and applications. On the other hand, hardware is no longer a source of performance as it has been in the last decades. Instead, it has become a complex, fast evolving, highly specialized, and heterogeneous platform that requires considerable tuning and effort to use optimally. In this talk I will discuss the problem, arguing that there is an opportunity for specialized designs based on FPGAs and showing the challenges to data processing resulting from modern hardware. I will illustrate the points with examples from research and recent developments from industry to argue there is a significant opportunity for FPGAs in data centers if one focuses on the correct problems and finds the proper architecture for the complete system.

FPL 2016 (Keynotes)

Christoph Hagleitner

IBM, CH

Heterogeneous Computing Systems in Cloud Datacenters

Cloud computing is at the core of the ongoing revolution inside the IT industry and the cloud platform is a central strategic element in IBMs current transformation. Heterogeneous computing systems employing GPUs, FPGAs, and custom ASICs promise to address the performance and energy-efficiency bottlenecks of homogeneous, CPU-based *datacenter (DC)* infrastructures. FPGAs offer reconfigurability and superior energy-efficiency, but have only established themselves in niche markets like networking and storage appliances in current DCs.

FPL 2016 (Keynotes)

P. K. Gupta

Intel, US

Accelerating Datacenter Workloads

Developers are continually challenged with solving workloads that are increasing in diversity and complexity. Modern workloads include use cases for cloud, networking, HPC and storage which have different attributes. We will discuss strategies and tools to accelerate these workloads using FPGAs, including development environments that make it easier to deploy and manage FPGAs in the datacenter.

FPL 2016 (Keynotes)

Tomas Evensen

Xilinx, US

A Software Developer's Journey into a Deeply Heterogeneous Hardware World

Embedded software developers have always had to deal with more than just the code running on a processor. To be effective you had to understand the underlying hardware, including caches and interrupts. But with today's deeply heterogeneous architectures, including multiple types of microprocessors, FPGAs, DSPs, and GPUs, the software developer is being asked to partition the application to run efficiently across hardware that sometimes does not even remotely resemble the CPU you are used to. Using the Xilinx MPSoC as an example, this talk will cover what it takes to create an effective software environment that can handle the heterogeneous nature of modern embedded SoCs, including using multiple operating systems and putting some of your C code in the FPGA.

Estamos próximos

"My personal belief is that if reconfigurable computing is to be successful it must create a methodology to automatically map from standard programming languages to the hardware system."

Scott Hauck, 1998

78

Conclusões

79

Conclusões

- O que esperar do futuro?
  - Padrões Abertos

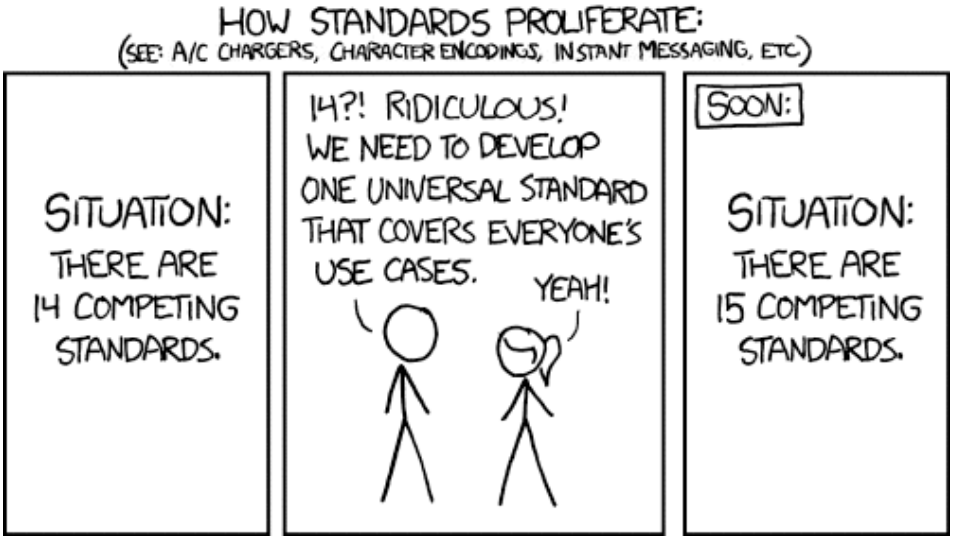
Conclusões

- O que esperar do futuro?
  - Padrões Abertos

"The nice thing about standards is that you have so many to choose from; furthermore, if you do not like any of them, you can just wait for next year's model." (Andrew S. Tanenbaum)

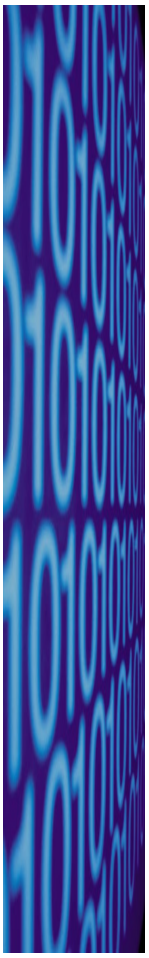
Conclusões

- O que esperar do futuro?
  - Padrões Abertos



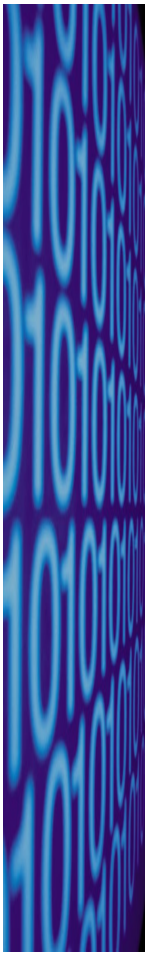
Conclusões

- O que esperar do futuro?
  - Padrões Abertos



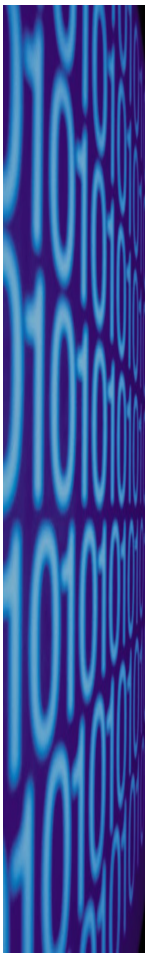
Conclusões

- O que esperar do futuro?
  - Padrões Abertos
  - Heterogêneo



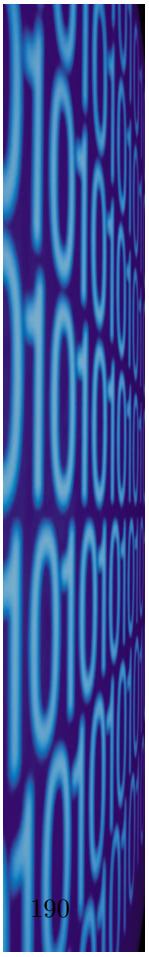
Conclusões

- O que esperar do futuro?
  - Padrões Abertos
  - Heterogêneo
  - Massivamente paralelo



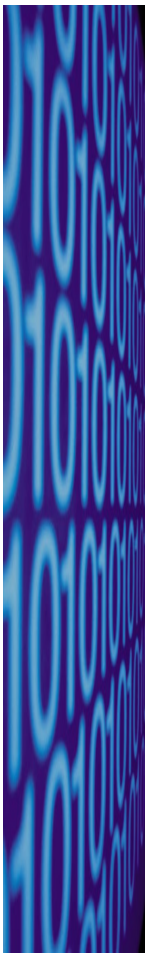
Conclusões

- O que esperar do futuro?
  - Padrões Abertos
  - Heterogêneo
  - Massivamente paralelo
  - Energeticamente eficiente



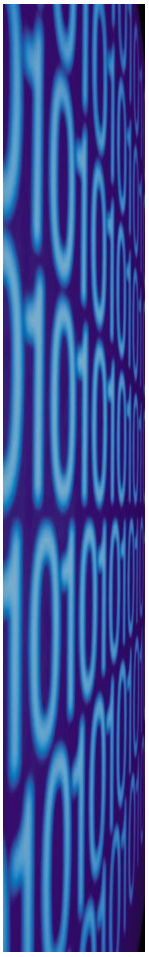
Conclusões

- Desafios
  - Remodelar os “computadores”
  - Reescrever bilhões de linhas de código
  - Retreinar milhões de programadores
  - Reescrever os currículos dos cursos



Conclusões

- Oportunidades
  - Arquiteturas
  - Linguagens
  - Metodologias
  - Técnicas de Compilação



Obrigado!

- Organização da ERAD-SP
  - pelo convite!
- Vanderlei Bonato (LCR/ICMC/USP)
  - pelo acesso ao equipamento!
- Leandro Rosa (LCR/ICMC/USP)
  - pelo setup da máquina e dicas!
- Todos
  - Pela audiência!

O primeiro passo para programar dez milhões de *cores*

Álvaro Fazenda
Denise Stringhini

ICT-UNIFESP
São José dos Campos

ERAD-SP 2016

Por que **dez milhões** de *co*

top500.org *(junho/2016)*

Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	National Supercomputing Center in Wuxi China	**Sunway TaihuLight** - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway NRCPC	10,649,600	93,014.6	125,435.9	15,371
2	National Super Computer Center in Guangzhou China	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3,120,000	33,862.7	54,902.4	17,808
3	DOE/SC/Oak Ridge National Laboratory United States	**Titan** - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560,640	17,590.0	27,112.5	8,209
4	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1,572,864	17,173.2	20,132.7	7,890

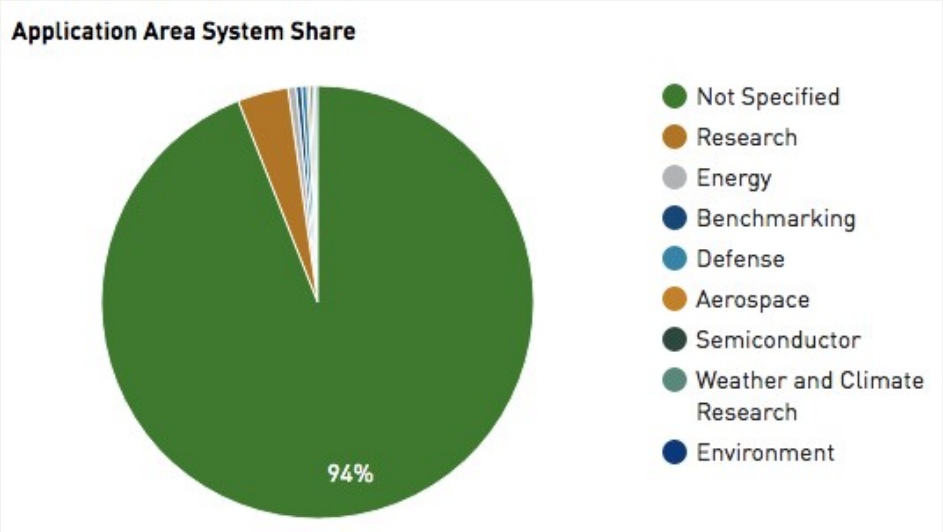
E o Brasil no Top 500?

Brasileiros no Top 500

Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
265	Laboratório Nacional de Computação Científica Brazil	**Santos Dumont GPU** - Bullx B710, Intel Xeon E5-2695v2 12C 2.4GHz, Infiniband FDR, Nvidia K40 Bull, Atos Group	10,692	456.8	657.5	
323	SENAI CIMATEC Brazil	**CIMATEC Yemoja** - SGI ICE X, Intel Xeon E5-2690v2 10C 3GHz, Infiniband FDR SGI	17,200	405.4	412.8	
364	Laboratório Nacional de Computação Científica Brazil	**Santos Dumont Hybrid** - Bullx B710, Intel Xeon E5-2695v2 12C 2.4GHz, Infiniband FDR, Intel Xeon Phi 7120P Bull, Atos Group	24,732	363.2	478.8	
433	Laboratório Nacional de Computação Científica Brazil	**Santos Dumont CPU** - Bullx B71x, Intel Xeon E5-2695v2 12C 2.4GHz, Infiniband FDR Bull, Atos Group	18,144	321.2	348.4	

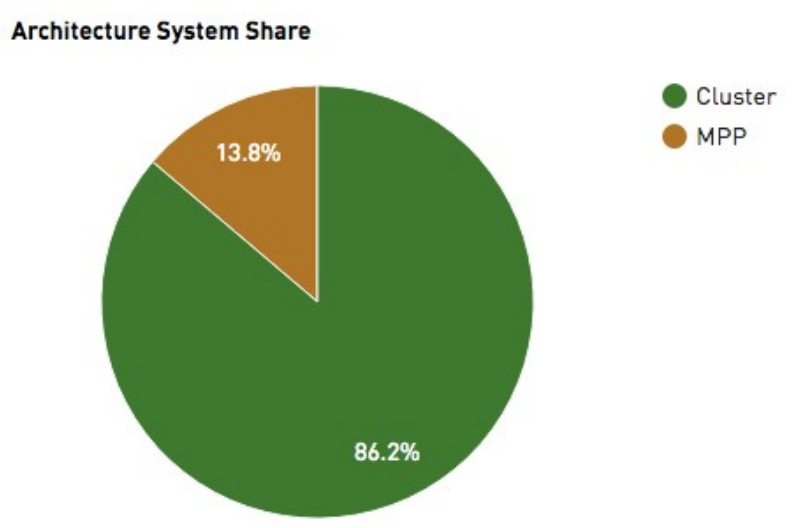
E quanto às aplicações?

Aplicações – Top 500



Arquitetura

Arquitetura – Top 500



Exemplo: IBM Blue Gene

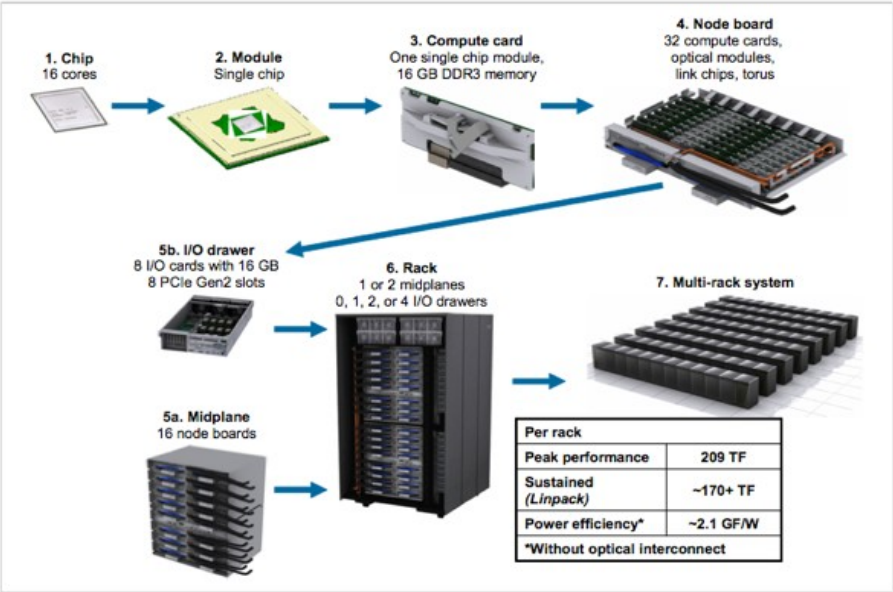
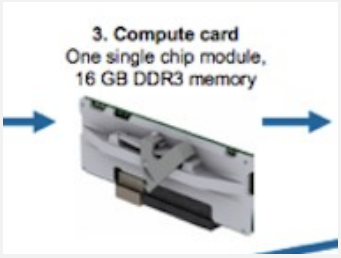


Figure 1-2 Blue Gene/Q hardware overview

Memória compartilhada

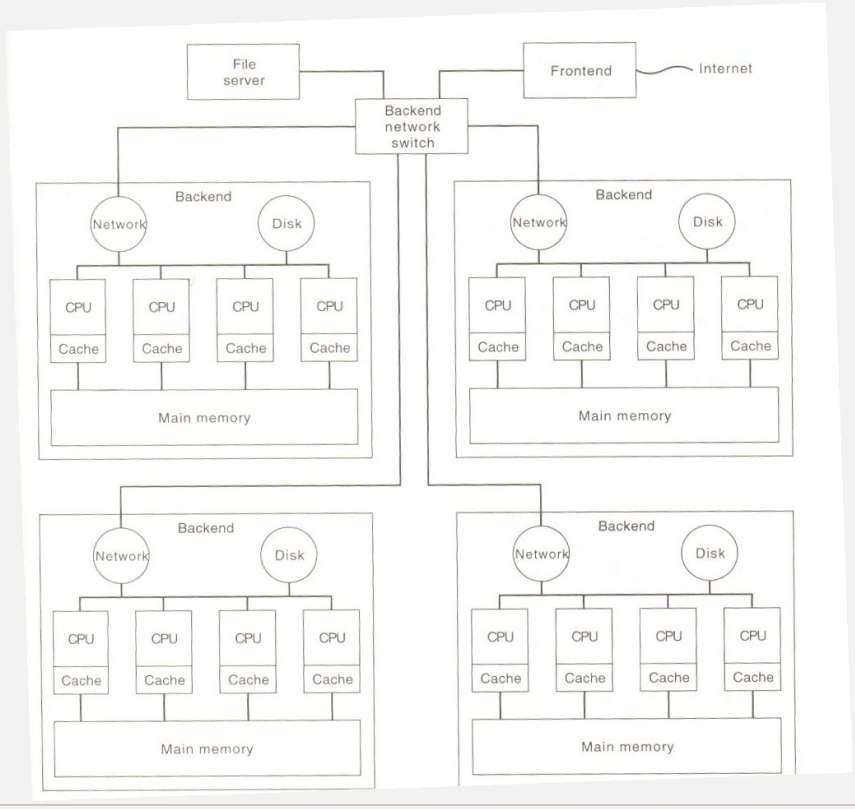
- Espaço de endereçamento compartilhado entre os núcleos (*cores*).
- Multicore
- Programação: variáveis compartilhadas entre *threads*.
  - OpenMP, Pthreads



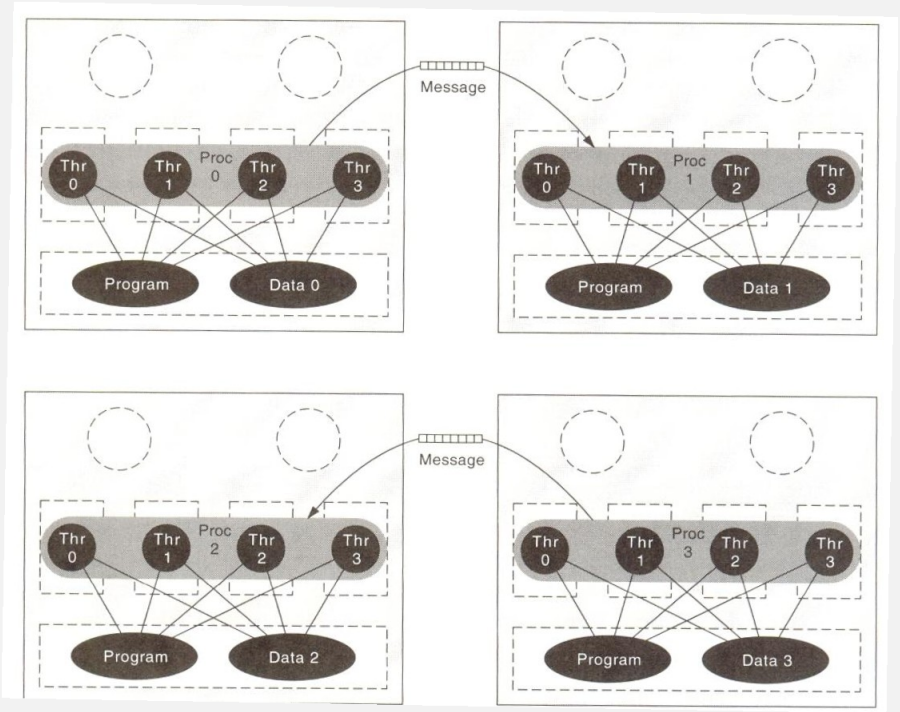
Memória distribuída

- Espaço de endereçamento não compartilhado entre nós de processamento.
- Cluster, MPP
- Programação: troca de mensagens entre processos.
  - **MPI**, PVM

MPP



Híbridos



Híbridos

Computação heterogênea

- Uso de aceleradores para a obtenção de maior desempenho.
  - Conexão PCIe.
  - Transferência de dados.
- Programação: bibliotecas específicas para cada tipo de acelerador.
  - CUDA, OpenCL, OpenMP 4, OpenAcc

E o desempenho?

High Performance Computing...

Características que afetam o desempenho

- Dois dos principais objetivos do projeto de aplicações paralelas consistem em obter-se:
  - **Desempenho**: a capacidade de reduzir o tempo de resolução do problema à medida que os recursos computacionais aumentam;
  - **Escalabilidade**: a capacidade de aumentar o desempenho à medida que a complexidade do problema aumenta.

Limites nos Algoritmos Paralelos

- Limites Arquiteturais
  - Latência e Largura de Banda
  - Capacidade de Memória
- Limites Algoritmicos
  - Falta de Paralelismo (fração sequencial/concorrente)
  - Frequência de Comunicação
  - Frequência de Sincronização

Lei de Amdahl

- Limitação teórica para os ganhos de desempenho

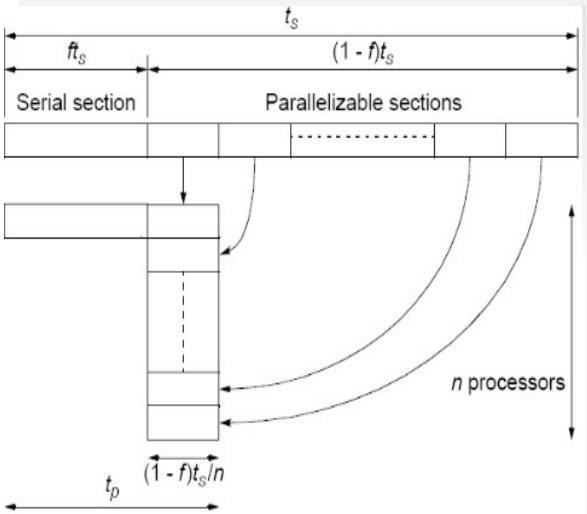
Programa serial: *Tserial = T0 = (s+q)T0*

- Onde:
  - *s+q = 1*
  - *s* corresponde a fração serial do código (impossível de ser paralelizada)
  - *q* corresponde a fração paralelizavel do código

Gene M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities", AFIPS spring joint computer conference, 1967

Lei de Amdahl

- Supondo paralelização ideal:
  - *Tpar = sT0 + (q/p)T0*
- *Speedup = Tserial/Tpar = (s+q)/(s+q/P) = 1 / (s+q/P)*
- *Speedup = 1 / [s+(1-s)/P]*



Comparação de desempenho

- Medida básica: **Tempo de Execução**
- O sistema *A* é ***n*** vezes mais rápido que o sistema *B* quando:
  - *Texec(A) / Texec(B) = n*
- Maior desempenho ↔ Menor tempo de execução

Speedup / Eficiência

- ***Speedup***
  - Medida de ganho em tempo

    Speedup(P) = Texec(1 proc) / Texec(P proc)
    - Onde *P* = número de processadores
      - *1 ≤ Speedup ≤ P*
- **Eficiência**
  - Medida de uso dos processadores

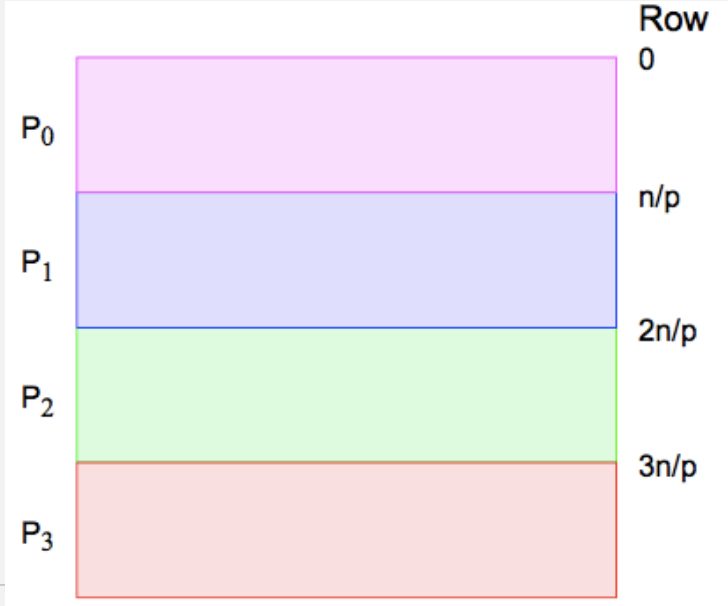
    Eficiência(P) = Speedup(P) / P
    - *0 < Eficiência ≤ 1*

Programação

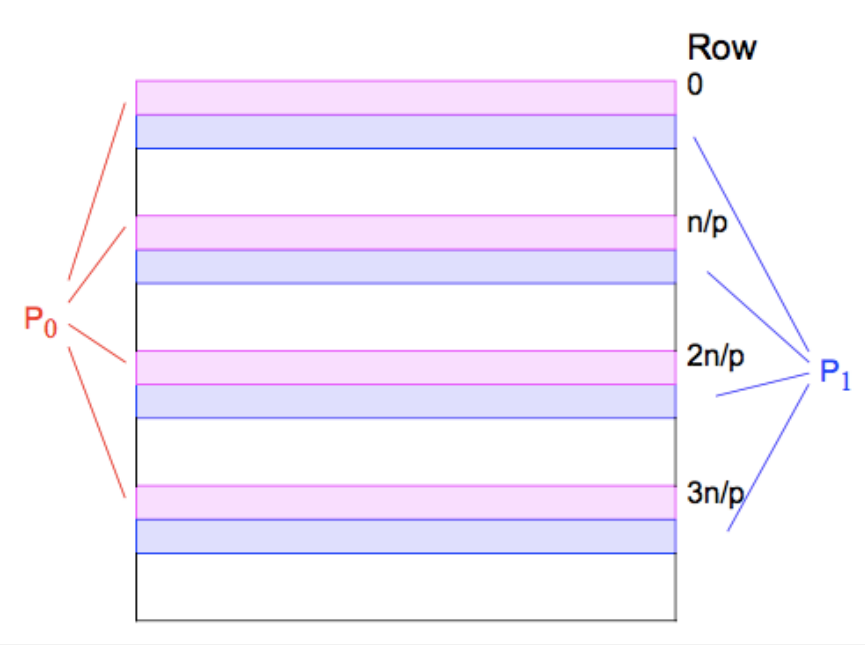
Sequencial x Concorrente

- Algoritmo Sequencial:
  - Sequência de passos para resolver um problema.
- Algoritmo Concorrente (definição aproximada): sequência de passos para resolver um problema **+**
  - **Decomposição em tarefas**
  - **Mapeamento de tarefas** (em processadores)
  - **Distribuição dos dados** (entrada, saída e intermediários)
  - **Sincronização e Comunicação**

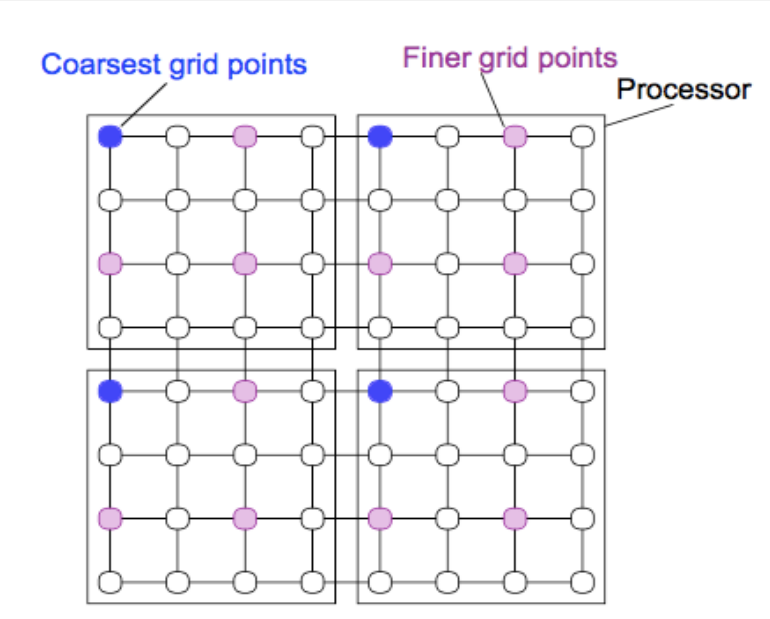
Particionamento 1: blocos de linhas



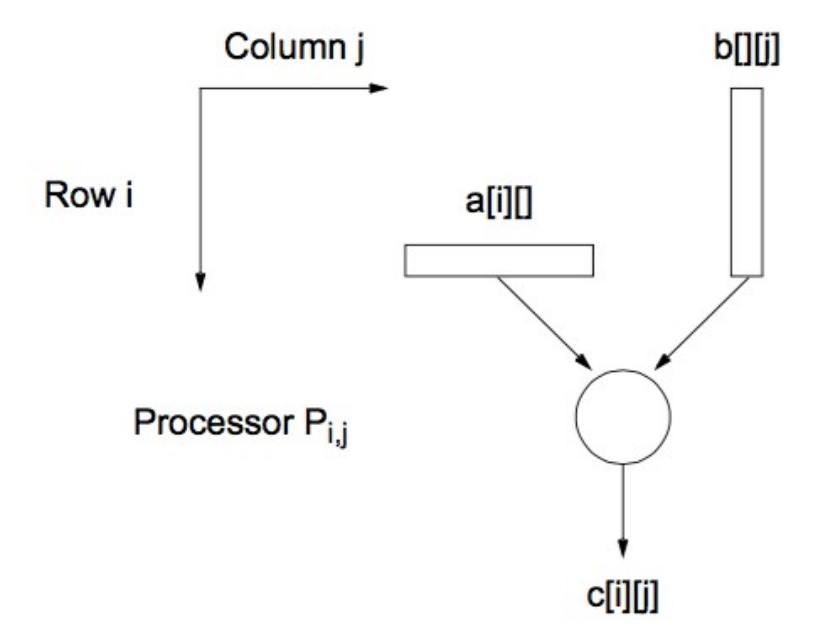
Particionamento 2: cíclico



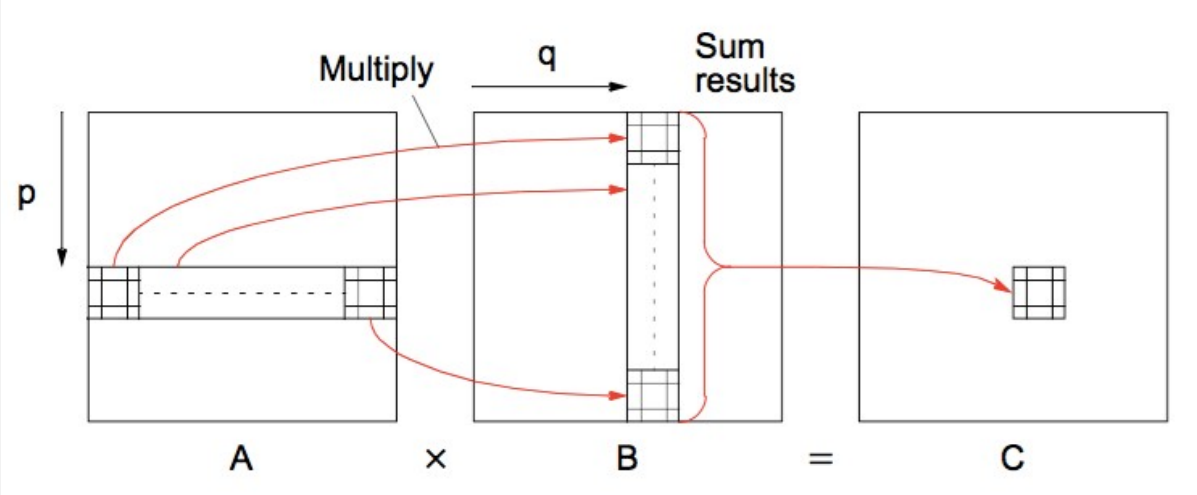
Particionamento 3: blocos



Exemplo 1: implementação direta (linha e coluna)



Exemplo 2: multiplicação de matrizes por blocos



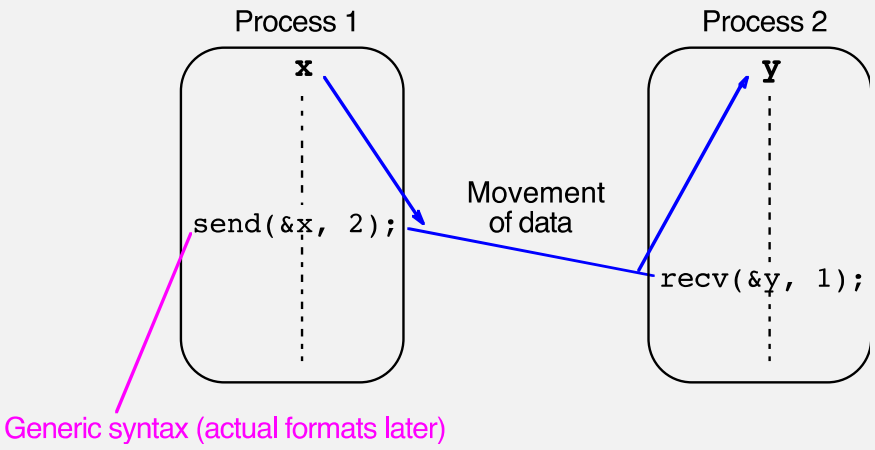
MPI

Message Passing Interface

Introdução

- Biblioteca padrão para computação por troca de mensagens (memória distribuída)
- Rotinas de comunicação ponto-a-ponto e coletiva
- Métodos para criação de processos remotos
- Várias implementações existentes (MPICH, OpenMPI)
- Linguagens: C/C++ e Fortran

Rotinas de comunicação ponto-a-ponto

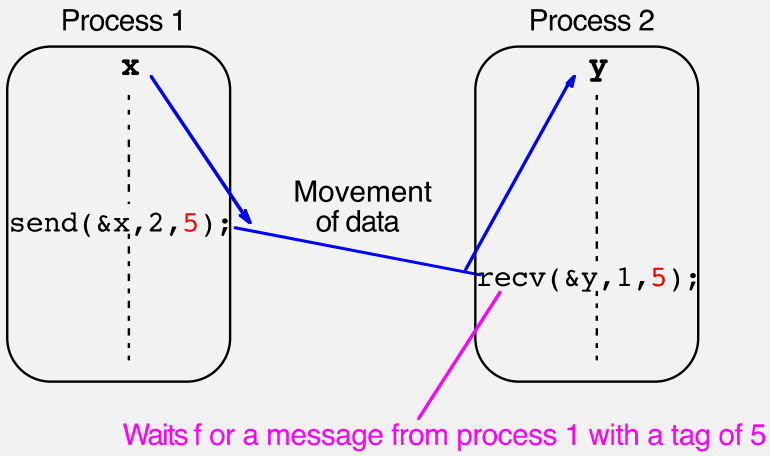


MPI_Send

```
MPI_Send(buf, count, datatype, dest, tag, comm)
```

- buf: Address of send buffer
- count: Number of items to send
- datatype: Datatype of each item
- dest: Rank of destination process
- tag: Message tag
- comm: Communicator

Message tag



Communicator

- Define o escopo de comunicação
- Dentro deste escopo, cada processo possui um identificador ou *rank*
- Existe um comunicador padrão (*default*) que engloba todos os processos iniciados em uma aplicação:

MPI_COMM_WORLD

- Usado em todas as rotinas de comunicação

Exemplo

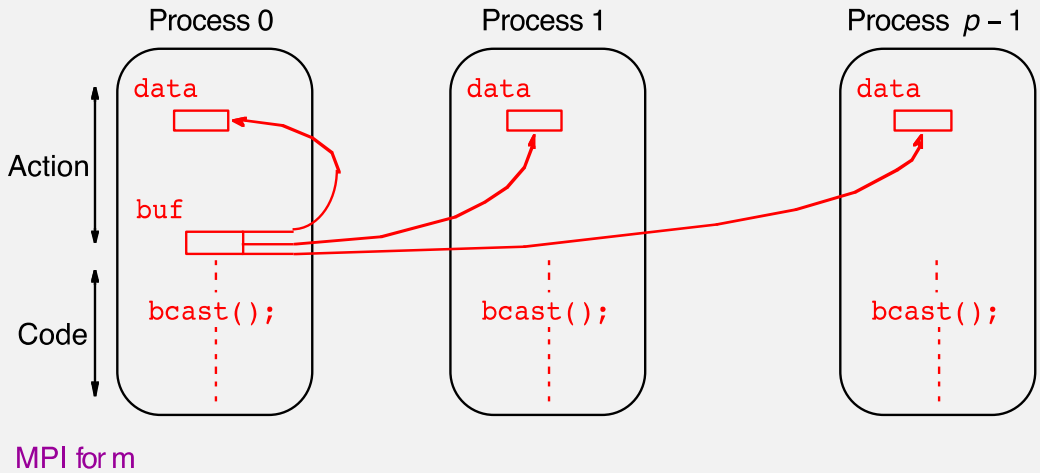
Enviar o inteiro x do processo 0 ao processo 1

```
MPI_Comm_rank(MPI_COMM_WORLD,&myrank); /* find rank */

if (myrank == 0) {
    int x;
    MPI_Send(&x, 1, MPI_INT, 1, msgtag, MPI_COMM_WORLD);
} else if (myrank == 1) {
    int x;
    MPI_Recv(&x, 1, MPI_INT,
    0,msgtag,MPI_COMM_WORLD,status);
}
```

Rotinas de comunicação coletiva

- Envolvem um conjunto de processos definidos pelo comunicador.
- Principais rotinas:
  - **`MPI_Bcast()`** - Envia do *root* para todos os outros
  - **`MPI_Gather()`** - Recolhe valores de um grupo
  - **`MPI_Scatter()`** - Distribui um buffer entre processos
  - **`MPI_Alltoall()`** - Envia de todos para todos
  - **`MPI_Reduce()`** - Combina valores de vários processos
  - **`MPI_Reduce_scatter()`** - Combina valores e distribui

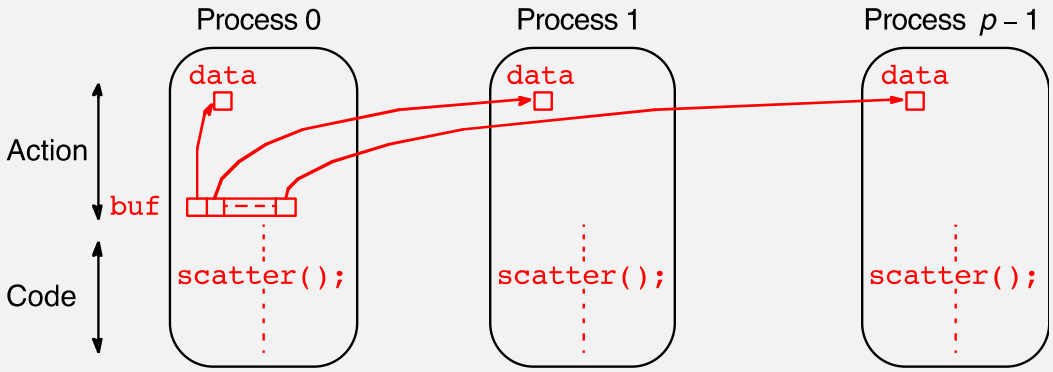
Broadcast



MPI for m

I_Bcast(void* buffer, int count, MPI_Datatype datatype, int root, MPI_Comm com

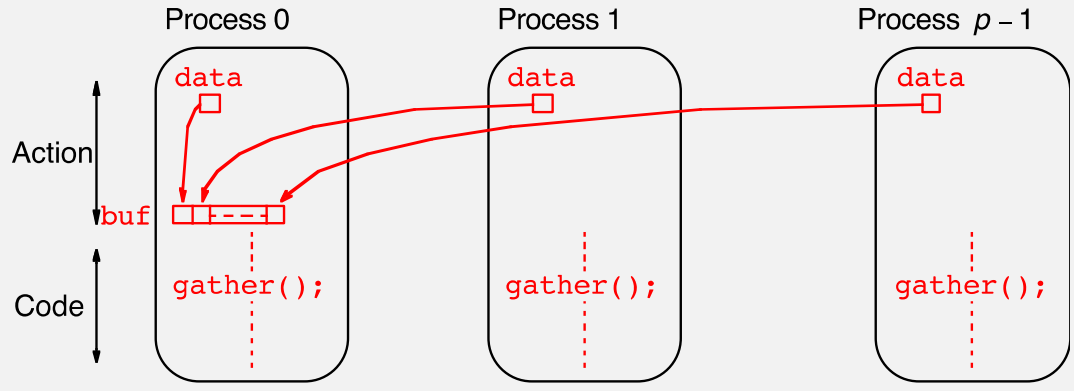
Scatter



MPI form

MPI_Scatter(void* sendbuf, int sendcount, MPI_Datatype sendtype, void* recvbuf,
int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

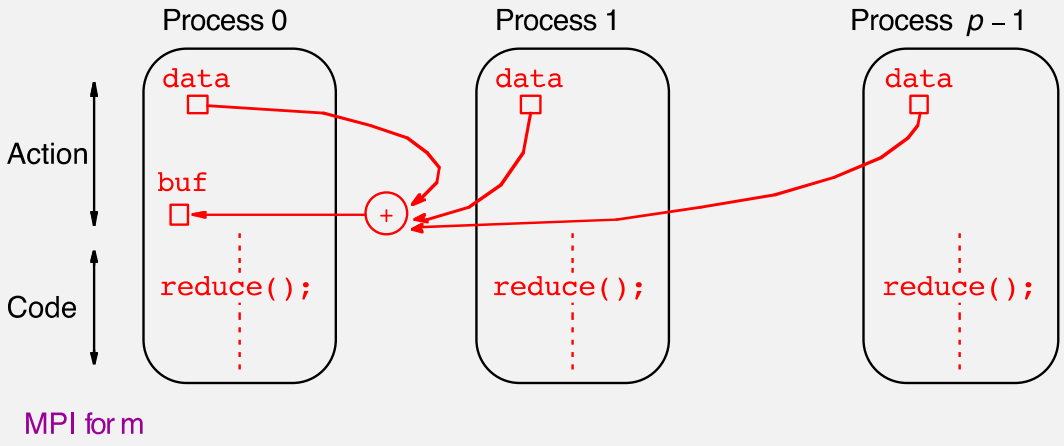
Gather



MPI form

MPI_Gather(void* sendbuf, int sendcount, MPI_Datatype sendtype, void* recvbuf,
int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

Reduce



MPI form

MPI_Reduce(void* sendbuf, void* recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)

Exemplo

- *Root* (processo 0) recebe items enviados de todos os demais processos e armazena em memória dinâmica.

```
int data[10];        /*data to be gathered from processes*/
MPI_Comm_rank(MPI_COMM_WORLD, &myrank); /* find rank */
if (myrank == 0) {
   MPI_Comm_size(MPI_COMM_WORLD, &grp_size); /*find group size*/
   buf = (int *)malloc(grp_size*10*sizeof (int)); /*allocate memory*/
}
MPI_Gather(data,10,MPI_INT,buf,grp_size*10,MPI_INT,0,
   MPI_COMM_WORLD) ;
```

- **`MPI_Gather()`** recebe de todos os processos, incluindo o *root*.

Exemplo de programa em MPI com C

Broadcast/Reduce

```
#include “mpi.h”
#include <stdio.h>
#include <math.h>
#define MAXSIZE 1000
void main(int argc, char *argv)
{
     int myid, numprocs;
     int data[MAXSIZE], i, chunksize, low, high, myresult=0, result;
     char fn[255];
     char *fp;
     MPI_Init(&argc,&argv);
     MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
     MPI_Comm_rank(MPI_COMM_WORLD,&myid);
     if (myid == 0) {  /* Open input file and initialize data */
          strcpy(fn,getenv(“HOME”));
          strcat(fn,”/MPI/rand_data.txt”);
          if ((fp = fopen(fn,”r”)) == NULL) {
               printf(“Can’t open the input file: %s\n\n”, fn);
               exit(1);
          }
          for(i = 0; i < MAXSIZE; i++) fscanf(fp,”%d”, &data[i]);
     }
     MPI_Bcast(data, MAXSIZE, MPI_INT, 0, MPI_COMM_WORLD); /* broadcast data */
     chunksize= MAXSIZE/numprocs; /* Add my portion Of data */
     low = myid * chunksize;
     high = low + chunksize;
     for(i = low; i < high; i++)
          myresult += data[i];
     printf(“I got %d from %d\n”, myresult, myid); /* Compute global sum */
     MPI_Reduce(&myresult, &result, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
     if (myid == 0) printf(“The sum is %d.\n”, result);
     MPI_Finalize();
}
```


Exemplo de programa em MPI com C

Broadcast/Reduce


```
#include “mpi.h”
#include <stdio.h>
#include <math.h>
#define MAXSIZE 1000
void main(int argc, char *argv)
{
     int myid, numprocs;
     int data[MAXSIZE], i, chunksize, low, high, myresult=0, result;
     char fn[255];
     char *fp;
     MPI_Init(&argc,&argv);
     MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
     MPI_Comm_rank(MPI_COMM_WORLD,&myid);
     if (myid == 0) {
          strcpy(fn,getenv(“HOME”));
          strcat(fn,”/MPI/rand_data.txt”);
          if ((fp = fopen(fn,”r”)) == NULL) {
               printf(“Can’t open the input file: %s\n\n”, fn);
               exit(1);
          }
          for(i = 0; i < MAXSIZE; i++) fscanf(fp,”%d”, &data[i]);
     }
     MPI_Bcast(data, MAXSIZE, MPI_INT, 0, MPI_COMM_WORLD); /* broadcast data */
     chunksize= MAXSIZE/numprocs; /* Add my portion Of data */
     low = myid * chunksize;
     high = low + chunksize;
     for(i = low; i < high; i++)
          myresult += data[i];
     printf(“I got %d from %d\n”, myresult, myid); /* Compute global sum */
     MPI_Reduce(&myresult, &result, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
     if (myid == 0) printf(“The sum is %d.\n”, result);
     MPI_Finalize();
}
```

Exemplo de programa em MPI com C

Broadcast/Reduce

```
#include "mpi.h"
#include <stdio.h>
#include <math.h>
#define MAXSIZE 1000
void main(int argc, char *argv)
{
     int myid, numprocs;
     int data[MAXSIZE], i, chunksize, low, high, myresult=0, result;
     char fn[255];
     char *fp;
     MPI_Init(&argc,&argv);
     MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
     MPI_Comm_rank(MPI_COMM_WORLD,&myid);
     if (myid == 0) {  /* Open input file and initialize data */
            strcpy(fn,getenv("HOME"));
            strcat(fn,"/MPI/rand_data.txt");
            if ((fp = fopen(fn,"r")) == NULL) {
                   printf("Can't open the input file: %s\n\n", fn);
                   exit(1);
            }
            for(i = 0; i < MAXSIZE; i++) fscanf(fp,"%d", &data[i]);
     }
     MPI_Bcast(data,MAXSIZE,MPI_INT, 0, MPI_COMM_WORLD);
     chunksize= MAXSIZE/numprocs;
     low = myid * chunksize;
     high = low + chunksize;
     for(i = low; i < high; i++)
        myresult += data[i];
     printf("I got %d from %d\n", myresult, myid); /* Compute global sum */
     MPI_Reduce(&myresult, &result, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
     if (myid == 0) printf("The sum is %d.\n", result);
     MPI_Finalize();
}
```

Exemplo de programa em MPI com C

Scatter/Reduce

```
#include "mpi.h"
#include <stdio.h>
#include <math.h>
#define MAXSIZE 1000
void main(int argc, char *argv)
{
     int myid, numprocs;
     int data[MAXSIZE], i, chunksize, low, high, myresult=0, result;
     char fn[255];
     char *fp;
     MPI_Init(&argc,&argv);
     MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
     MPI_Comm_rank(MPI_COMM_WORLD,&myid);
     if (myid == 0) {  /* Open input file and initialize data */
            strcpy(fn,getenv("HOME"));
            strcat(fn,"/MPI/rand_data.txt");
            if ((fp = fopen(fn,"r")) == NULL) {
                   printf("Can't open the input file: %s\n\n", fn);
                   exit(1);
            }
            for(i = 0; i < MAXSIZE; i++) fscanf(fp,"%d", &data[i]);
     }
     chunksize = MAXSIZE/numprocs;
     int rdata[chunkSize];
     MPI_Scatter(data, chunkSize, MPI_INT, rdata,
                 chunkSize, MPI_INT, 0, MPI_COMM_WORLD);
     for(i = 0; i < chunksize; i++)
        myresult += rdata[i];
     printf("I got %d from %d\n", myresult, myid); /* Compute global sum */
     MPI_Reduce(&myresult, &result, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
     if (myid == 0) printf("The sum is %d.\n", result);
     MPI_Finalize();
}
```

Exemplo de programa em MPI com C

Broadcast/Reduce

```
#include "mpi.h"
#include <stdio.h>
#include <math.h>
#define MAXSIZE 1000
void main(int argc, char *argv)
{
     int myid, numprocs;
     int data[MAXSIZE], i, chunksize, low, high, myresult=0, result;
     char fn[255];
     char *fp;
     MPI_Init(&argc,&argv);
     MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
     MPI_Comm_rank(MPI_COMM_WORLD,&myid);
     if (myid == 0) {  /* Open input file and initialize data */
          strcpy(fn,getenv("HOME"));
          strcat(fn,"/MPI/rand_data.txt");
          if ((fp = fopen(fn,"r")) == NULL) {
               printf("Can't open the input file: %s\n\n", fn);
               exit(1);
          }
          for(i = 0; i < MAXSIZE; i++) fscanf(fp,"%d", &data[i]);
     }
     MPI_Bcast(data, MAXSIZE, MPI_INT, 0, MPI_COMM_WORLD); /* broadcast data */
     chunksize= MAXSIZE/numprocs; /* Add my portion Of data */
     low = myid * chunksize;
     high = low + chunksize;
     for(i = low; i < high; i++)
          myresult += data[i];
     printf("I got %d from %d\n", myresult, myid); /* Compute global sum */
     MPI_Reduce(&myresult, &result, 1, MPI_INT,
                MPI_SUM, 0, MPI_COMM_WORLD);
     if (myid == 0)
        printf("The sum is %d.\n", result);
     MPI_Finalize();
}
```

Exemplo de programa em MPI com C

Send/Recv

```
#include "mpi.h"
#include <stdio.h>
#include <math.h>
#define MAXSIZE 1000
void main(int argc, char *argv)
{
     int myid, numprocs;
     int data[MAXSIZE], i, chunksize, low, high, myresult=0, result;
     char fn[255];
     char *fp;
     MPI_Init(&argc,&argv);
     MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
     MPI_Comm_rank(MPI_COMM_WORLD,&myid);
     if (myid == 0) {
          strcpy(fn,getenv("HOME"));
          strcat(fn,"/MPI/rand_data.txt");
          if ((fp = fopen(fn,"r")) == NULL) {
               printf("Can't open the input file: %s\n\n", fn);
               exit(1);
          }
          for(i = 0; i < MAXSIZE; i++) fscanf(fp,"%d", &data[i]);
          for(i = 0; i<numprocs; i++)
              MPI_Send(data,MAXSIZE,MPI_INT,i,
                       1,MPI_COMM_WORLD);
     } else{
          MPI_Recv(data,MAXSIZE,MPI_INT,0,
                   1,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
          chunksize= MAXSIZE/numprocs; /* Add my portion Of data */
          low = myid * chunksize;
          high = low + chunksize;
          for(i = low; i < high; i++)
               myresult += data[i];
          printf("I got %d from %d\n", myresult, myid); /* Compute global sum */
     }
     MPI_Reduce(&myresult, &result, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
     if (myid == 0) printf("The sum is %d.\n", result);
     MPI_Finalize();
}
```

Compilação e execução

- Compilação: **mpicc prog.c -o prog**
- Execução local: **mpirun -np 2 prog**
- Mais de uma máquina:
  - Necessário configurar o **ssh** para que realize a conexão sem exigir senha.
  - O arquivo executável deve estar disponível em todas as máquinas.
  - Criar um arquivo texto com os IPs das máquinas.
  - Fornecer arquivo ao **mpirun**.

Depuração e Visualização

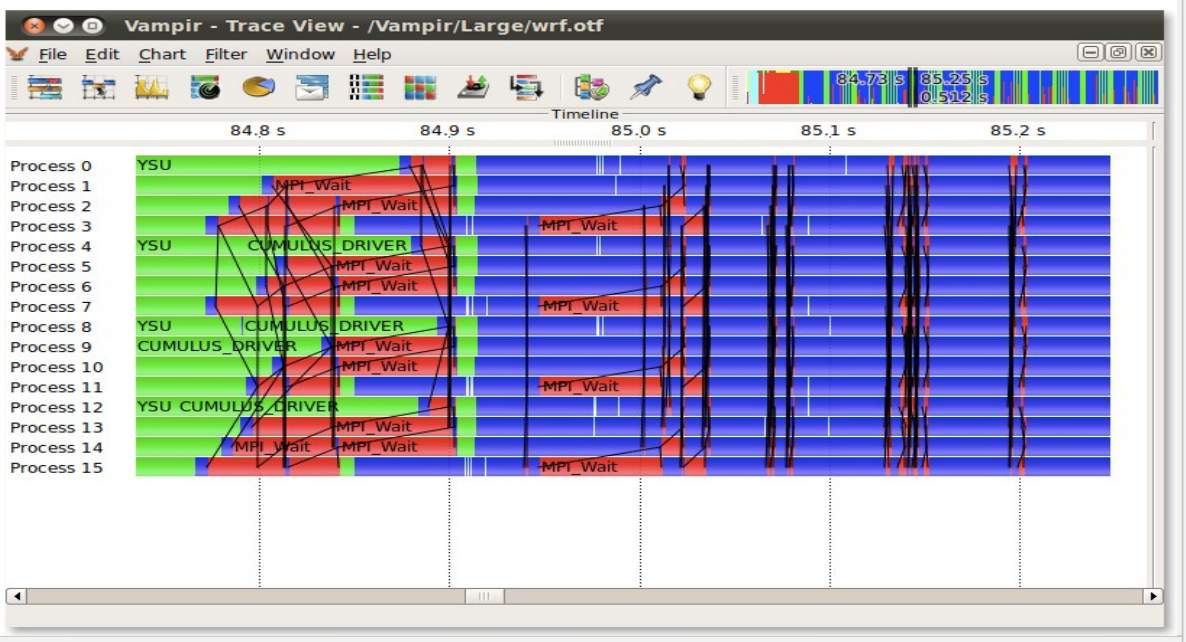
Depuração

- Ferramenta difícil de encontrar para MPI, normalmente são paga$.
  - Exemplo: TotalView, Intel Debugger
- Dificuldade: *breakpoints* distribuídos, parada em estado consistente
- Podem ser *online* ou *post mortem.*
- Ferramentas de visualização podem ser usadas para depuração

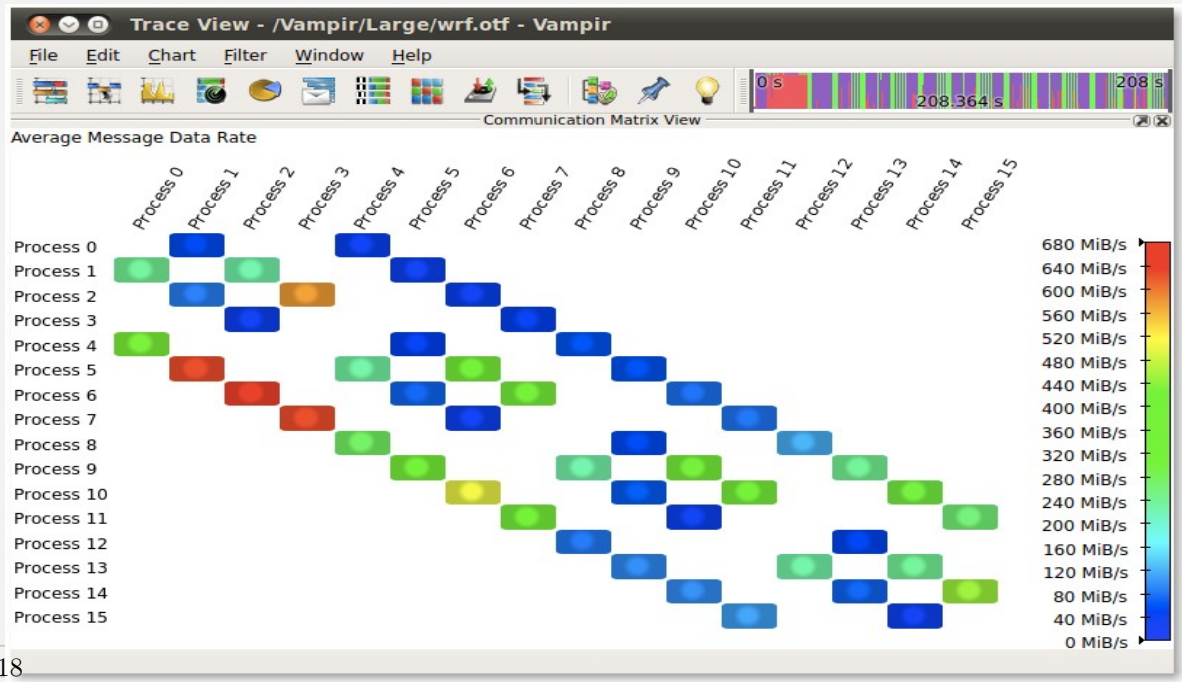
Visualização

- Permite que os desenvolvedores consigam visualizar o comportamento do programa em diferentes níveis de detalhes.
- Eventos são coletados durante a execução do programa e são posteriormente mostrados em diferentes tipos de janelas de visualização.

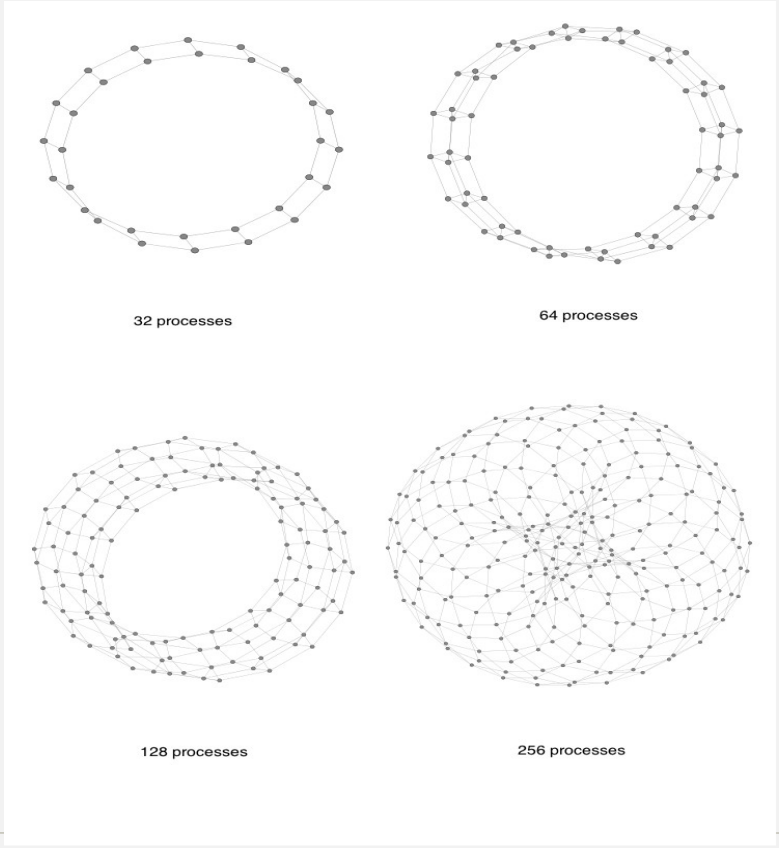
Exemplo: Vampir *space-time*



Exemplo: Vampir *communication matrix*



Visualização de grafos



Considerações finais

- Apenas primeiro passo...
- Para programar dez milhões de cores também é necessário:
  - OpenMP (incluindo aceleradores)
  - CUDA, OpenCL, OpenACC
  - outras (Charm++, OmpSS, etc)
  - Estudo de algoritmos e técnicas de programação concorrente.

Bibliografia

- Kaminsky, A. Building Parallel Programs: Smps, Clusters & Java. New York: Course Technology Ptr,2009.
- Wilkinson, B.; Allen, M. Parallel Programming Techniques & Applications Using Networked Workstations & Parallel Computers 2nd Edition, 2004

Um Estudo sobre a Expansão das Diretivas de Compilação para Interceptação de Código OpenMP

Rogério A. Gonçalves[1,2,3] e Alfredo Goldman[2]

[1]Universidade Tecnológica Federal do Paraná (UTFPR)
Departamento de Computação (DACOM)
Campo Mourão – PR – Brasil

[2]Universidade de São Paulo (USP)
Instituto de Matemática e Estatística (IME)
Centro de Competência em Software Livre (CCSL)
Laboratório de Sistemas de Software (LSS)
São Paulo – SP – Brasil

rogerioag@utfpr.edu.br, {rag, gold}@ime.usp.br

Agenda

Objetivos

- Apresentar alguns conceitos sobre a implementação das diretivas de compilação do `OpenMP`
- Que podem ser utilizados para o desenvolvimento de bibliotecas de interceptação.
- Técnica de hooking – `LD_PRELOAD`
- Alguns exemplos de aplicações

Padrão OpenMP I

- O `OpenMP`[1] é um padrão bem conhecido e amplamente utilizado em aplicações para plataformas *multicore*.
- O uso de *diretivas de compilação*.
- O `OpenMP` implementa o modelo `fork-join`.
- Múltiplas threads executam tarefas definidas implicitamente ou explicitamente.
- As diretivas funcionam como anotações.
- São implementadas usando-se as diretivas de pré-processamento `#pragma`, em `C/C++`, e sentinelas `!$`, no `Fortran`.

Padrão OpenMP II

- As diretivas são substituídas pelo seu formato de código expandido com as chamadas para o runtime do OpenMP.
- Nosso estudo foi baseado nas diretivas de compilação da biblioteca libgomp[2] do GCC.
- A motivação desse estudo foi a necessidade de interceptar código de aplicações OpenMP para fazer *offloading* de código para aceleradores.

[1] Dagum and Menon (1998); OpenMP-ARB (2011, 2013, 2015)
[2] GNU Libgomp (2015b,c, 2016a,b)

Implementações OpenMP I

- O OpenMP tem sido suportado por praticamente todos os compiladores atuais.
- Compiladores como GCC[3], Intel icc[4] e LLVM clang[5] tem implementações para OpenMP.
- Pelo menos duas implementações: GNU GCC libgomp[6] e a Intel libomp *(OpenMP* Runtime Library)* [7].
- A especificação do OpenMP atualmente cobre *offloading* de código para aceledores.
- A libgomp é capaz de fazer *offloading* usando o padrão OpenACC[8].

[3] GCC (2015); GNU Libgomp (2015a)
[4] Intel (2016b)
[5] Lattner and Adve (2004); LLVM Clang (2015); LLVM OpenMP (2015)
[6] GNU Libgomp (2015b,c, 2016a,b)
[7] Intel (2016a)
[8] OpenACC (2012, 2015b, 2011, 2013, 2015a)

Diretivas de Compilação e Código OpenMP Expandido

(Formato pós expansão das diretivas)

LibGOMP: GNU OpenMP Runtime Library
(GNU Offloading and Multi Processing Runtime Library)

Estudo das Diretivas de Compilação I

- Verificou-se o formato de código gerado pelo GCC + libgomp para os construtores de regiões paralelas *(parallel region)* e compartilhamento de trabalho *(parallel for)*.
- O código foi gerado nas versões do GCC (4.8, 4.9, 5.3, 6.1) para verificarmos o formato interceptável.
- Adotamos o GCC 4.8, porque esta versão apresentar um formato mais consistente e definido que possibilita a interceptação.

Regiões paralelas: construtor *parallel* I

- Esta é uma das mais importantes diretivas, pois ela é responsável pela demarcação de regiões paralelas.

Regiões paralelas são criadas usando-se o construtor:

```
#pragma omp parallel [clause[ [,] clause] ... ] new-line
{
   /* Bloco estruturado. */
}
```

- Quando uma região paralela é encontrada, é criado um time de threads para executar o código da região.
- Porém, esse construtor não divide o trabalho entre as threads.

Regiões paralelas: construtor *parallel* I

- Quando a diretiva de região paralela é utilizada:

```
1 #pragma omp parallel
2 {
3   // body;
4 }
```

- A diretiva *parallel* é implementada com a criação de uma nova função *(outlined function)* usando o código contido em `body`.
- A `libgomp` usa funções para delimitar a região. Chamadas a essas funções são colocadas no código para indicar o início e o fim de uma região paralela:

The `libgomp` ABI

```
void GOMP_parallel_start(void (*fn)(void *), void *data,
    unsigned num_threads)
void GOMP_parallel_end(void)
```

Regiões paralelas: construtor *parallel* II

- O código expandido gerado assume o formato:

```
1 void subfunction (void *data){
2   use data;
3   body;
4 }
5
6 // replace the annotated parallel region.
7 setup data;
8
9 GOMP_parallel_start(subfunction, &data, num threads);
10 subfunction(&data);
11 GOMP_parallel_end();
```


Regiões paralelas: construtor *parallel* III

- O código em GIMPLE, que é a representação intermediária do GCC:

```
1 main._omp_fn.0 (void * .omp_data_i) {
2   return;
3 }
4
5 main () {
6 int D.1803;
7
8 <bb 2>:
9   __builtin_GOMP_parallel_start (main._omp_fn.0, 0B,
        0);
10   main._omp_fn.0 (0B);
11   __builtin_GOMP_parallel_end ();
12   D.1803 = 0;
13
14 <L0>:
15   return D.1803;
16 }
```

Regiões paralelas: construtor *parallel* IV

- O código em *assembly*:

```
 1    .file "parallel-region.c"
 2    .text
 3    .globl main
 4    .type main, @function
 5 main:
 6 .LFB0:
 7    .cfi_startproc
 8    pushq %rbp
 9    .cfi_def_cfa_offset 16
10    .cfi_offset 6, -16
11    movq %rsp, %rbp
12    .cfi_def_cfa_register 6
13    movl $0, %edx
14    movl $0, %esi
15    movl $main._omp_fn.0, %edi
16    call GOMP_parallel_start
17    movl $0, %edi
18    call main._omp_fn.0
19    call GOMP_parallel_end
20    movl $0, %eax
21    popq %rbp
22    .cfi_def_cfa 7, 8
23    ret
24    .cfi_endproc
25 .LFE0:
26    .size main, .-main
27    .type main._omp_fn.0, @function
28
29 main._omp_fn.0:
30 .LFB1:
31    .cfi_startproc
32    pushq %rbp
33    .cfi_def_cfa_offset 16
34    .cfi_offset 6, -16
35    movq %rsp, %rbp
36    .cfi_def_cfa_register 6
37    movq %rdi, -8(%rbp)
38    popq %rbp
39    .cfi_def_cfa 7, 8
40    ret
41    .cfi_endproc
42 .LFE1:
43    .size main._omp_fn.0, .-
          main._omp_fn.0
44    .ident "GCC: (Debian 4.8.4-1) 4
          .8.4"
45    .section .note.GNU-stack,"",
          @progbits
```


Loops: Construtor `for` I

- Um time de threads é criado quando uma região paralela é alcançada.
- Mas é necessário compartilhar o trabalho e coordenar a execução paralela.
- O construtor `for` é usado para distribuir o trabalho entre as threads.

Construtor *for*:

```
#pragma omp parallel for num_threads (number_of_threads)
    schedule ({auto, static, dynamic, guided, runtime}, {variable
    /expression | numerical value/constant})
```

Loops: Construtor `for` II

- O Código com uma região paralela com um laço é equivalente ao que apresenta os construtores em modo combinado.

```
1 #pragma omp parallel
2 {
3   #pragma omp for
4   for (i = lb; i <= ub; i++){
5     body;
6   }
7 }
```

```
1 #pragma omp parallel for
2 for (i = lb; i <= ub; i++){
3   body;
4 }
```


Loops: Construtor `for` I

- Semelhante ao processamento do construtor *parallel*, a diretiva `parallel for` também é implementada com a criação de uma nova função *(outlined function)* com o código do *loop*.
- Quando não se especifica um escalonamento o código é equivalente ao gerado para *schedule(static)*.
- A `libgomp` utiliza as funções para delimitar a região paralela e na construção do formato do *loop*, com escalonamento estático:

ABI da `libgomp` – Funções usadas a diretiva *parallel for*

```
void GOMP_parallel_loop_static(void (*)(void *), void *,unsigned
    , long, long, long, long, unsigned)
bool GOMP_loop_static_next(long *, long *)
void GOMP_loop_end_nowait(void)
void GOMP_parallel_end(void)
```

Loops: Construtor for II

- O código expandido que substitui a declaração do *loop* paralelo é composto de uma função *outlined* e de chamadas para criar a região paralela. O *loop* original está nas *linhas 5* e *6*.

```
1 void subfunction (void *data) {
2   long _s0, _e0;
3   while (GOMP_loop_static_next (&_s0, &_e0)){
4     long _e1 = _e0, i;
5     for (i = _s0; i < _e1; i++)
6       body;
7   }
8   GOMP_loop_end_nowait ();
9 }
10
11 GOMP_parallel_loop_static (subfunction, NULL, 0, lb,
       ub+1, 1, 0);
12 subfunction (NULL);
13 GOMP_parallel_end ();
```


Loops: Construtor for III

- Os códigos diferem apenas na definição do limite superior dos laços.

```
1 #pragma omp parallel for schedule(
      static)
2 for (i = 0; i < 1024; i++){
3   // body.
4 }
```

```
1 n = 1024;
2 #pragma omp parallel for schedule(
      static)
3 for (i = 0; i < n; i++){
4   // body.
5 }
```

Loops: Construtor `for` IV

- O código em GIMPLE, que é a representação intermediária do `GCC`:

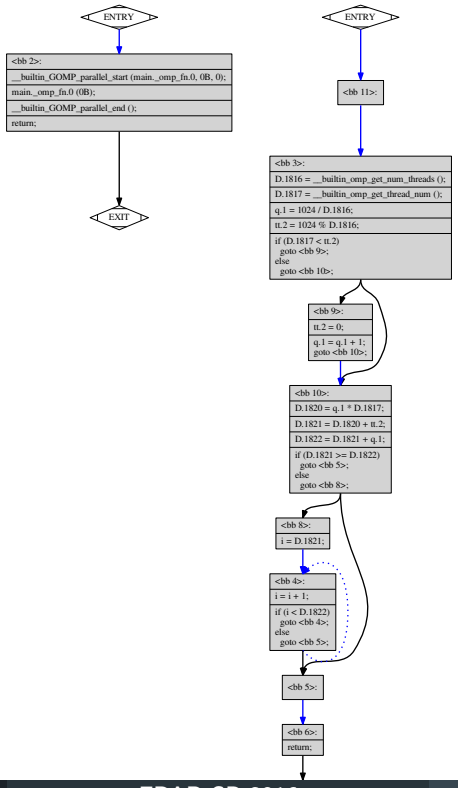
```
1  main () {
2
3  <bb 2>:
4    __builtin_GOMP_parallel_start (
         main._omp_fn.0, 0B, 0);
5    main._omp_fn.0 (0B);
6    __builtin_GOMP_parallel_end ();
7    return;
8  }
9
10 main._omp_fn.0(void* .omp_data_i){
11
12 <bb 11>:
13
14 <bb 3>:
15   D.1816 =
         __builtin_omp_get_num_threads
         ();
16   D.1817 =
         __builtin_omp_get_thread_num
         ();
17   q.1 = 1024 / D.1816;
18   tt.2 = 1024 % D.1816;
19   if (D.1817 < tt.2)
20     goto <bb 9>;
21   else
22     goto <bb 10>;
23
24 <bb 10>:
25   D.1820 = q.1 * D.1817;
26   D.1821 = D.1820 + tt.2;
27   D.1822 = D.1821 + q.1;
28   if (D.1821 >= D.1822)
29     goto <bb 5>;
30   else
31     goto <bb 8>;
32
33 <bb 8>:
34   i = D.1821;
35
36 <bb 4>:
37   i = i + 1;
38   if (i < D.1822)
39     goto <bb 4>;
40   else
41     goto <bb 5>;
42
43 <bb 5>:
44
45 <bb 6>:
46   return;
47
48 <bb 9>:
49   tt.2 = 0;
50   q.1 = q.1 + 1;
51   goto <bb 10>;
52 }
```


Loops: Construtor `for` V

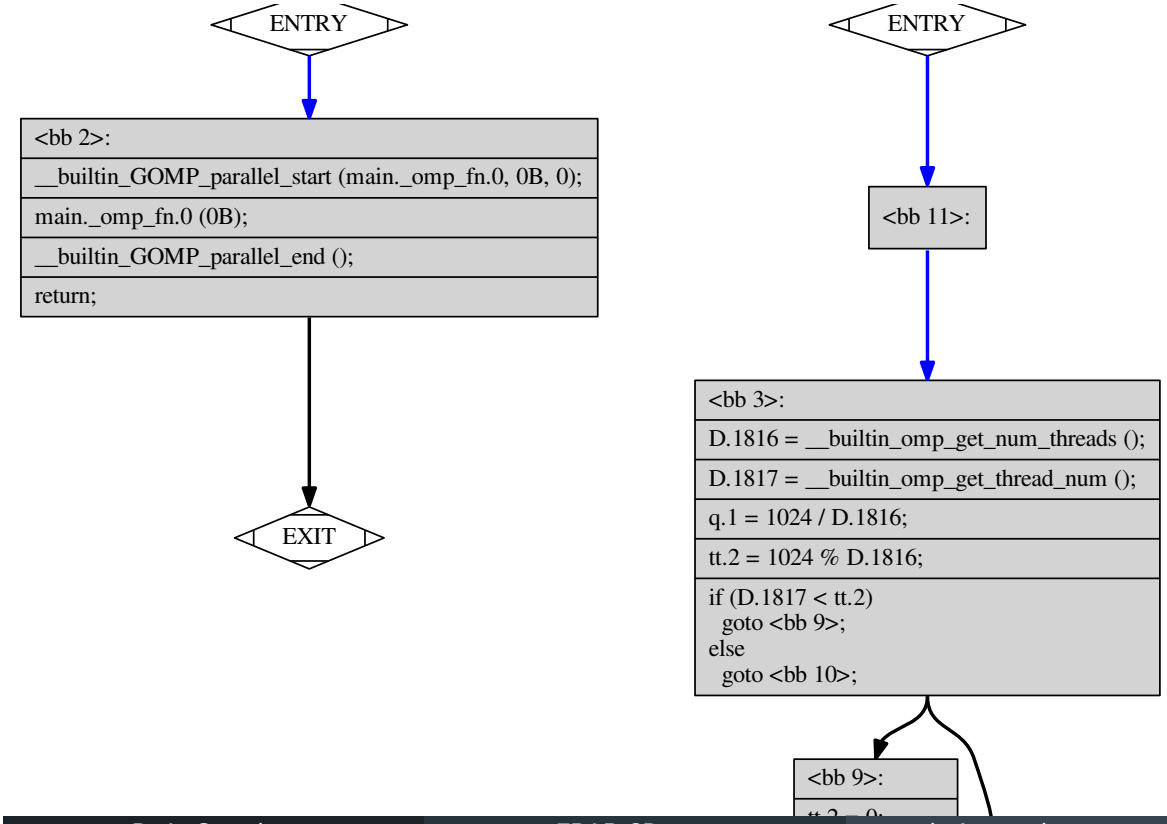
- O código em *assembly*:

```
1    .file "for-schedule-static-upper
         -bound-value.c"
2    .text
3    .globl main
4    .type main, @function
5  main:
6    pushq %rbp
7    movq %rsp, %rbp
8    movl $0, %edx
9    movl $0, %esi
10   movl $main._omp_fn.0, %edi
11   call GOMP_parallel_start
12   movl $0, %edi
13   call main._omp_fn.0
14   call GOMP_parallel_end
15   popq %rbp
16   ret
17   .size main, .-main
18   .type main._omp_fn.0, @function
19 main._omp_fn.0:
20   pushq %rbp
21   movq %rsp, %rbp
22   pushq %rbx
23   subq $40, %rsp
24   movq %rdi, -40(%rbp)
25   call omp_get_num_threads
26   movl %eax, %ebx
27   call omp_get_thread_num
28   movl %eax, %esi
29   movl $1024, %eax
30   cltd
34   cltd
35   idivl %ebx
36   movl %edx, %eax
37   cmpl %eax, %esi
38   jl .L3
39 .L6:
40   imull %ecx, %esi
41   movl %esi, %edx
42   addl %edx, %eax
43   leal (%rax,%rcx), %edx
44   cmpl %edx, %eax
45   jge .L2
46   movl %eax, -20(%rbp)
47 .L5:
48   addl $1, -20(%rbp)
49   cmpl %edx, -20(%rbp)
50   jl .L5
51   jmp .L2
52 .L3:
53   movl $0, %eax
54   addl $1, %ecx
55   jmp .L6
56 .L2:
57   addq $40, %rsp
58   popq %rbx
59   popq %rbp
60   ret
61   .size main._omp_fn.0, .-
         main._omp_fn.0
62   .ident "GCC: (Debian 4.8.4-1) 4
         .8.4"
63   .section .note.GNU-stack,"",
```

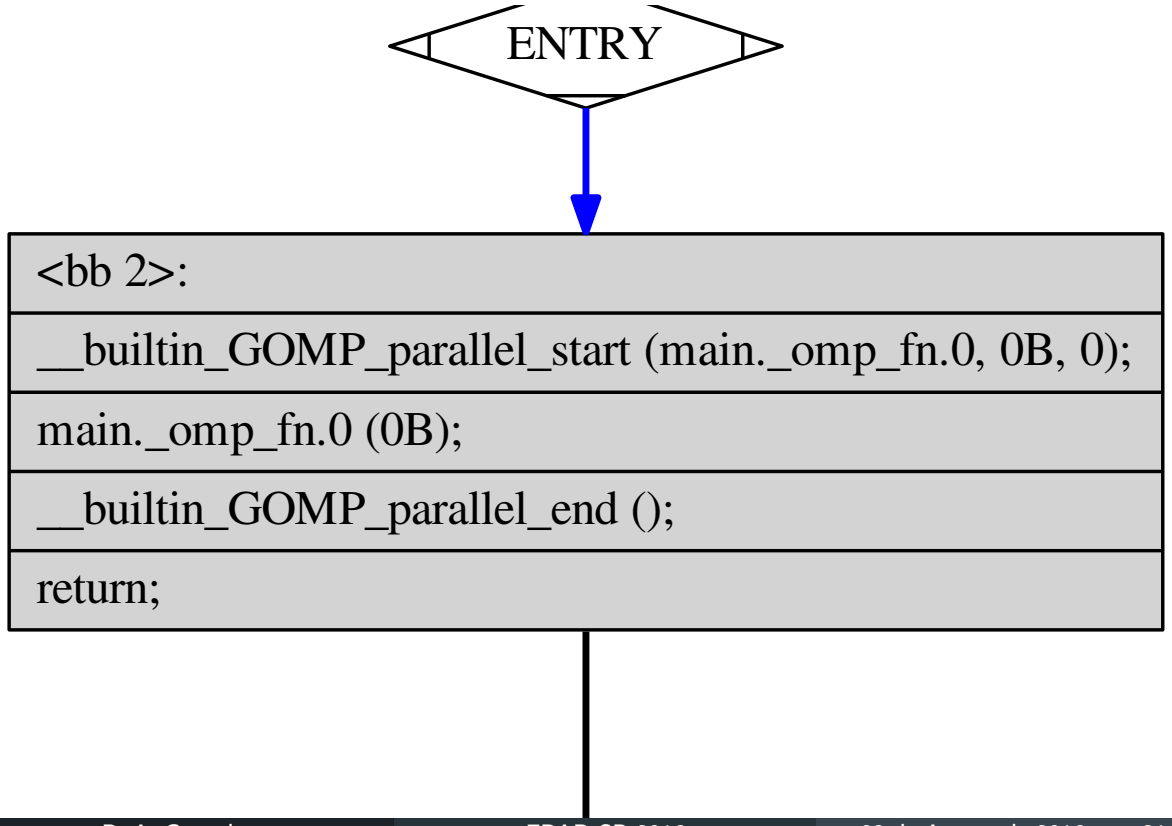
Loops: Construtor for



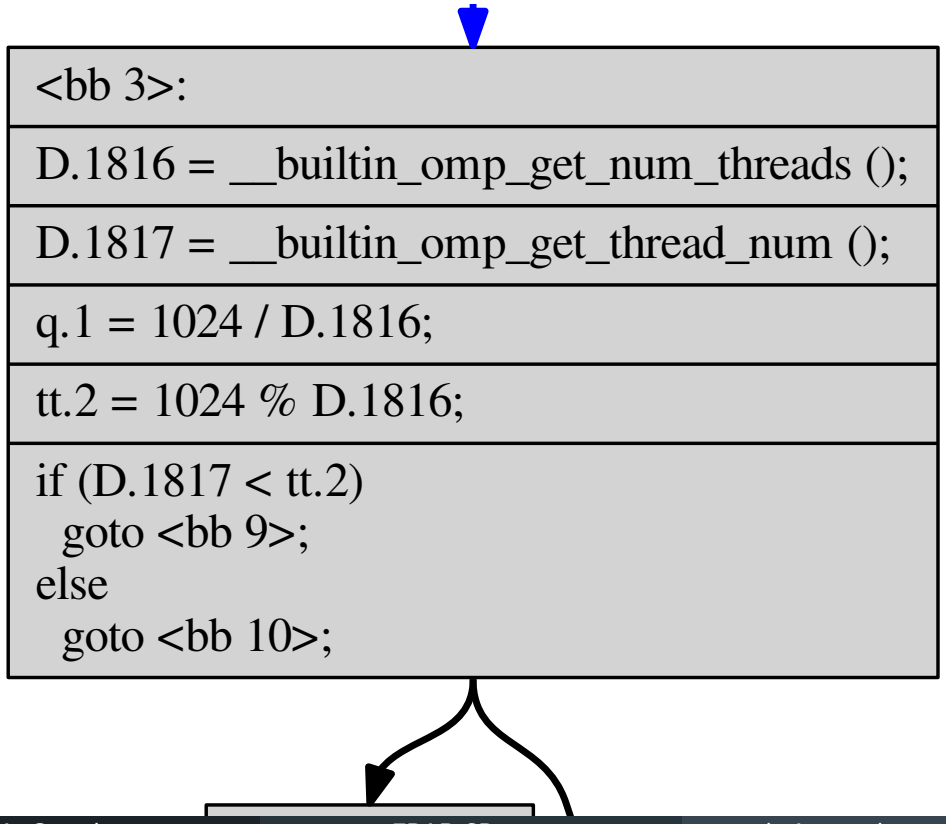
Loops: Construtor for



Loops: Construtor `for`



Loops: Construtor `for`



Loops: Construtor `for` – Primeiro formato I

- Para laços que utilizam escalonamentos dos tipos *dynamic, runtime* e *guided* – «schedule_type»:

```
1 #pragma omp parallel for schedule(dynamic|runtime|
      guided)
2 for (i = 0; i < 1024; i++) {
3   body;
4 }
```

- A `libgomp` usa as funções para delimitar a região paralela de código e criar o *primeiro* formato do *loop*:

ABI `libgomp` – funções usadas no primeiro formato da *parallel for*

```
void GOMP_parallel_loop_<<schedule_type>>_start (void (*fn) (
    void *), void *data, unsigned num_threads, long start, long
    end, long incr);
void GOMP_parallel_end (void);
bool GOMP_loop_<<schedule_type>>_next(long *istart, long *iend);
void GOMP_loop_end_nowait (void);
```

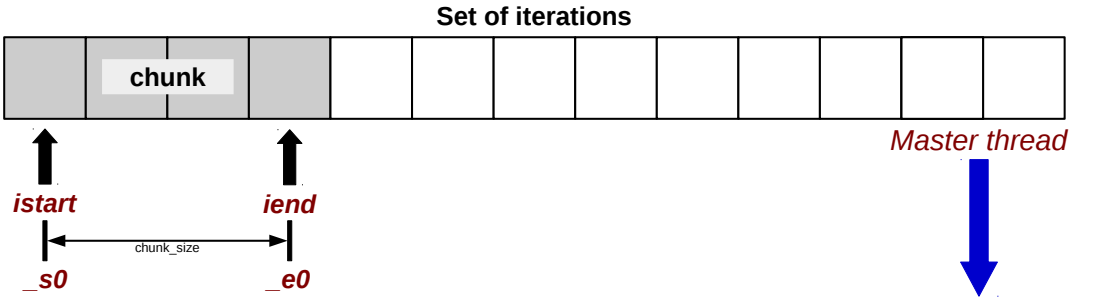

Loops: Construtor `for` – Primeiro formato II

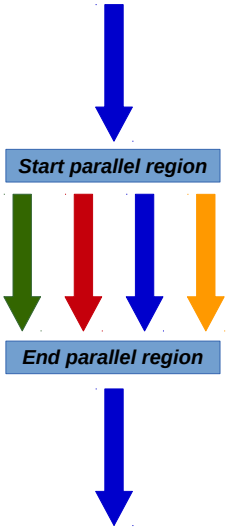
- O código expandido para o *primeiro formato*:

```
1 void subfunction (void *data){
2   long _s0, _e0;
3   while (GOMP_loop_<<schedule_type>>_next (&_s0, &_e0
        )){
4     long _e1 = _e0, i;
5     for (i = _s0; i < _e1; i++){
6       body;
7     }
8   }
9   GOMP_loop_end_nowait ();
10 }
11
12 GOMP_parallel_loop_<<schedule_type>>_start (
      subfunction, NULL, 0, lb, ub+1, 1, 0);
13 subfunction (NULL);
14 GOMP_parallel_end ();
```

Loops: Construtor `for` – Primeiro formato III

- Execução de Chunks pelas Threads






```
void subfunction (void *data){
  long _s0, _e0;
  while (GOMP_loop_runtime_next (&_s0, &_e0)){
    long _e1 = _e0, i;
    for (i = _s0; i < _e1; i++)
      body;
  }
  GOMP_loop_end_nowait ();
}

GOMP_parallel_loop_runtime_start (subfunction, NULL, 0, lb, ub+1, 1, 0);
subfunction (NULL);
GOMP_parallel_end ();
```

Original Loop

Loops: Construtor `for` – Primeiro formato IV

- O código `GIMPLE` para o primeiro formato:

```
1 main () {
2
3 <bb 2>:
4   __builtin_GOMP_parallel_loop_dynamic_start (main._omp_fn.0, 0B, 0, 0,
        1024, 1, 1);
5   main._omp_fn.0 (0B);
6   __builtin_GOMP_parallel_end ();
7   return;
8 }
9
10 main._omp_fn.0 (void * .omp_data_i) {
11
12 <bb 10>:
13
14 <bb 3>:
15   D.1818 = __builtin_GOMP_loop_dynamic_next (&.istart0.1, &.iend0.2);
16   if (D.1818 != 0)
17     goto <bb 8>;
18   else
19     goto <bb 5>;
20
21 <bb 8>:
22   .istart0.3 = .istart0.1;
23   i = (int) .istart0.3;
24   .iend0.4 = .iend0.2;
25   D.1822 = (int) .iend0.4;
```

Loops: Construtor `for` – Primeiro formato V

- O código `GIMPLE` para o primeiro formato:

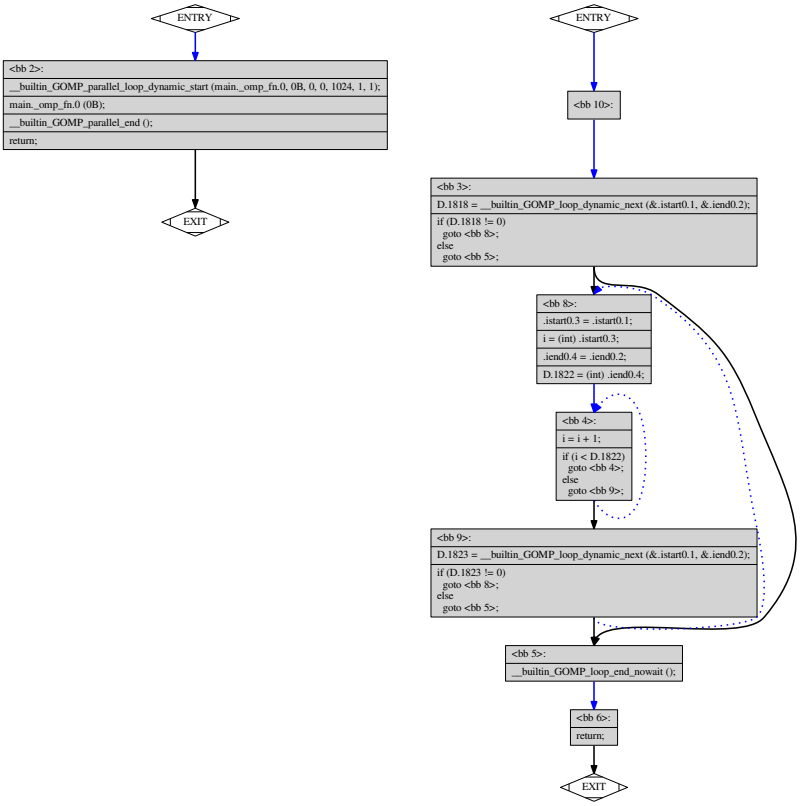
```
<bb 4>:
  i = i + 1;
  if (i < D.1822)
    goto <bb 4>;
  else
    goto <bb 9>;

<bb 9>:
  D.1823 = __builtin_GOMP_loop_dynamic_next (&.istart0.1, &.iend0.2);
  if (D.1823 != 0)
    goto <bb 8>;
  else
    goto <bb 5>;

<bb 5>:
  __builtin_GOMP_loop_end_nowait ();

<bb 6>:
  return;
}
```


Loops: Construtor `for` – Primeiro formato VI

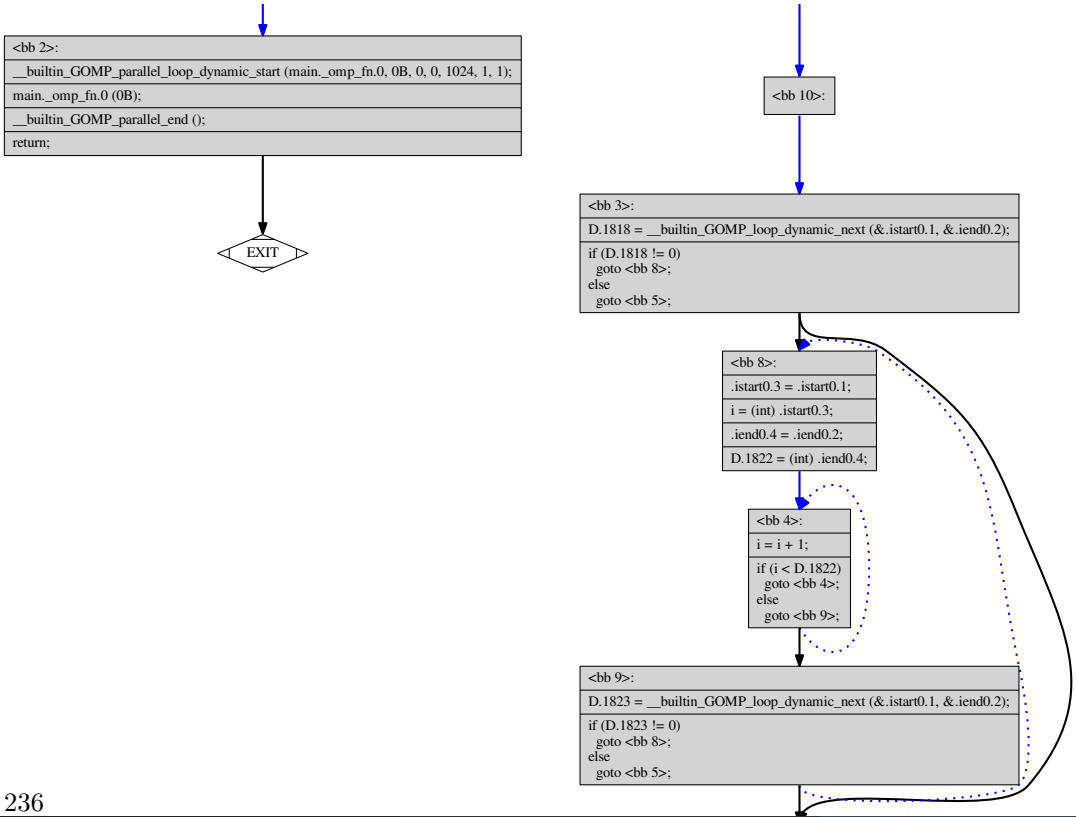
- O código em *assembly* para o primeiro formato:

```
  .file "for-schedule-dynamic-upper-bound-value.c"
  .text
  .globl main
  .type main, @function
main:
  pushq %rbp
  movq %rsp, %rbp
  subq $32, %rsp
  movq $1, (%rsp)
  movl $1, %r9d
  movl $1024, %r8d
  movl $0, %ecx
  movl $0, %edx
  movl $0, %esi
  movl $main._omp_fn.0, %edi
  call GOMP_parallel_loop_dynamic_start
  movl $0, %edi
  call main._omp_fn.0
  call GOMP_parallel_end
  leave
  ret
  .size main, .-main
  .type main._omp_fn.0, @function
main._omp_fn.0:
  pushq %rbp
  movq %rsp, %rbp
  subq $48, %rsp
  movq %rdi, -40(%rbp)
  leaq -24(%rbp), %rax
  movq %rdx, %rsi
  movq %rax, %rdi
  call GOMP_loop_dynamic_next
  testb %al, %al
  je .L3
.L5:
  movq -24(%rbp), %rax
  movl %eax, -4(%rbp)
  movq -16(%rbp), %rax
.L4:
  addl $1, -4(%rbp)
  cmpl %eax, -4(%rbp)
  jl .L4
  leaq -16(%rbp), %rdx
  leaq -24(%rbp), %rax
  movq %rdx, %rsi
  movq %rax, %rdi
  call GOMP_loop_dynamic_next
  testb %al, %al
  jne .L5
.L3:
  call GOMP_loop_end_nowait
  leave
  ret
  .size main._omp_fn.0, .-main._omp_fn.0
  .ident "GCC: (Debian 4.8.4-1) 4.8.4"
  .section .note.GNU-stack,"",@progbits
```

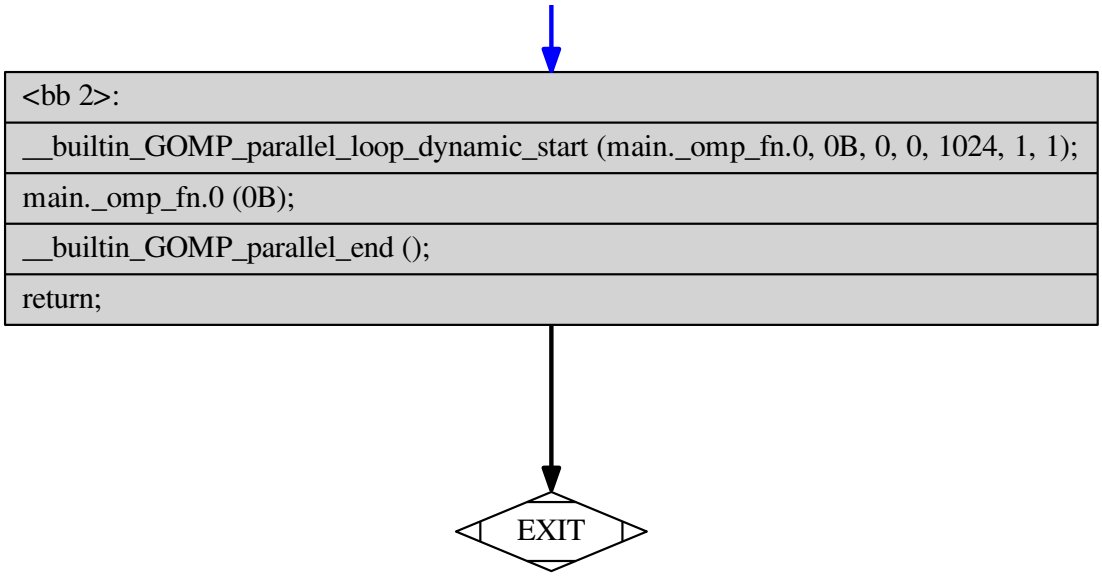
Loops: Construtor `for`



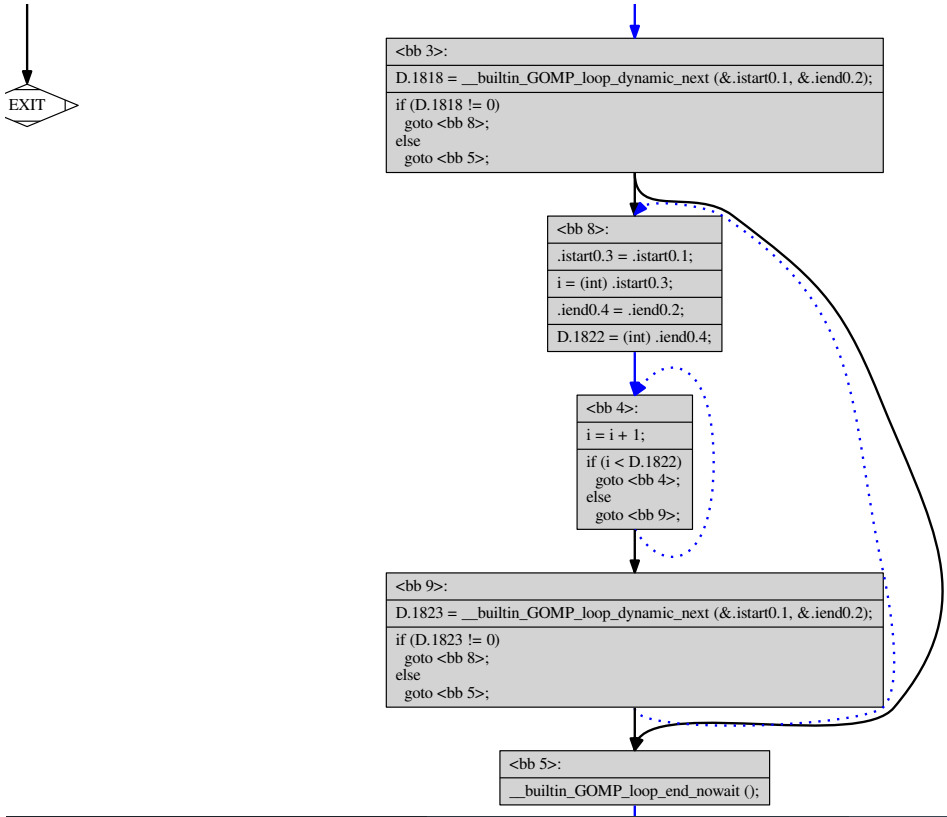
Loops: Construtor `for`



Loops: Construtor `for`



Loops: Construtor `for`



Loops: Construtor `for` – Segundo formato I

- Para laços que utilizam escalonamentos dos tipos *dynamic, runtime* e *guided* – «`schedule_type`»:

```
1 #pragma omp parallel for schedule(dynamic|runtime|
      guided)
2 n = 1024;
3 for (i = 0; i < n; i++) {
4   body;
5 }
```

- A `libgomp` usa as funções para delimitar a região paralela de código e criar o *segundo* formato do *loop*:

ABI `libgomp` – funções usadas no segundo formato da *parallel for*

```
void GOMP_parallel_start (void (*fn) (void *), void *data,
    unsigned num_threads);
void GOMP_parallel_end (void);
void GOMP_parallel_loop_<<schedule_type>>_start (void (*fn) (
    void *), void *data, unsigned num_threads, long start, long
    end, long incr);
bool GOMP_loop_<<schedule_type>>_next(long *istart, long *iend);
void GOMP_loop_end_nowait (void);
```


Loops: Construtor `for` – Segundo formato II

- O código expandido para o *segundo formato*:

```
1 void subfunction (void *data){
2   long i, _s0, _e0;
3   if (GOMP_loop_runtime_start (0, n, 1, &_s0, &_e0)){
4     do {
5       long _e1 = _e0;
6       for (i = _s0; i < _e0; i++) {
7         body;
8       }
9     } while (GOMP_loop_runtime_next (&_s0, &_e0));
10  }
11  GOMP_loop_end ();
12 }
13 /* The annoted loop is replaced. */
14 GOMP_parallel_loop_static (subfunction, NULL, 0, lb,
      ub+1, 1, 0);
15 subfunction (NULL);
16 GOMP_parallel_end ();
```

238

Loops: Construtor `for` – Segundo formato III

- O código `GIMPLE` para o segundo formato:

```
 1 main () {
 2   struct .omp_data_s.0 .omp_data_o.1;
 3
 4 <bb 2>:
 5   n = 1024;
 6   .omp_data_o.1.n = n;
 7   __builtin_GOMP_parallel_start (main._omp_fn.0, &.omp_data_o.1, 0);
 8   main._omp_fn.0 (&.omp_data_o.1);
 9   __builtin_GOMP_parallel_end ();
10   n = .omp_data_o.1.n;
11   return;
12 }
13
14 main._omp_fn.0 (struct .omp_data_s.0 * .omp_data_i) {
15
16 <bb 10>:
17
18 <bb 3>:
19   D.1822 = .omp_data_i->n;
20   D.1823 = (long int) D.1822;
21   D.1826 = __builtin_GOMP_loop_dynamic_start (0, D.1823, 1, 1, &.istart0.2,
         &.iend0.3);
22   if (D.1826 != 0)
23     goto <bb 8>;
24   else
25     goto <bb 5>;
```


Loops: Construtor `for` – Segundo formato IV

- O código `GIMPLE` para o segundo formato:

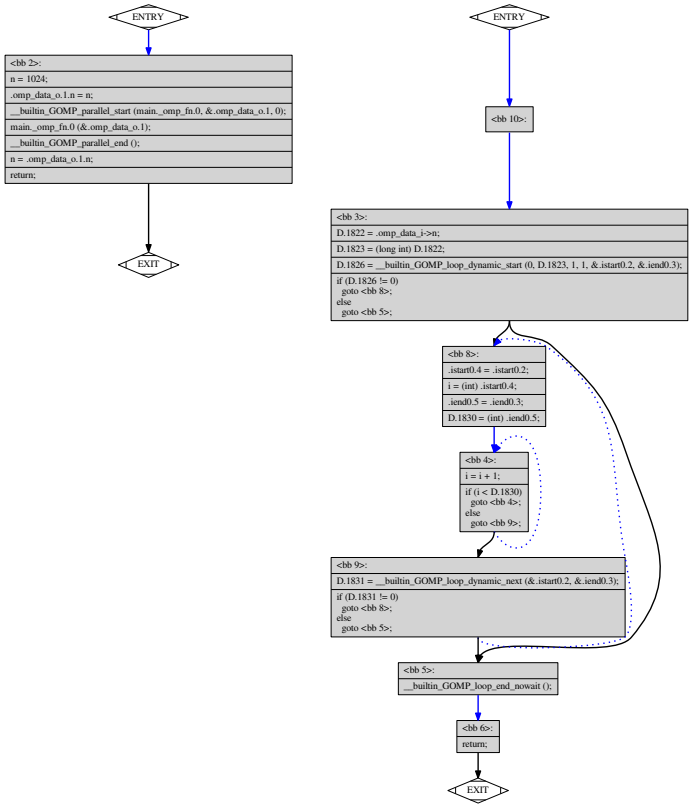
```
 1 <bb 8>:
 2   .istart0.4 = .istart0.2;
 3   i = (int) .istart0.4;
 4   .iend0.5 = .iend0.3;
 5   D.1830 = (int) .iend0.5;
 6
 7 <bb 4>:
 8   i = i + 1;
 9   if (i < D.1830)
10     goto <bb 4>;
11   else
12     goto <bb 9>;
13
14 <bb 9>:
15   D.1831 = __builtin_GOMP_loop_dynamic_next (&.istart0.2, &.iend0.3);
16   if (D.1831 != 0)
17     goto <bb 8>;
18 else
19     goto <bb 5>;
20
21 <bb 5>:
22   __builtin_GOMP_loop_end_nowait ();
23
24 <bb 6>:
25   return;
26 }
```

Loops: Construtor `for` – Segundo formato V

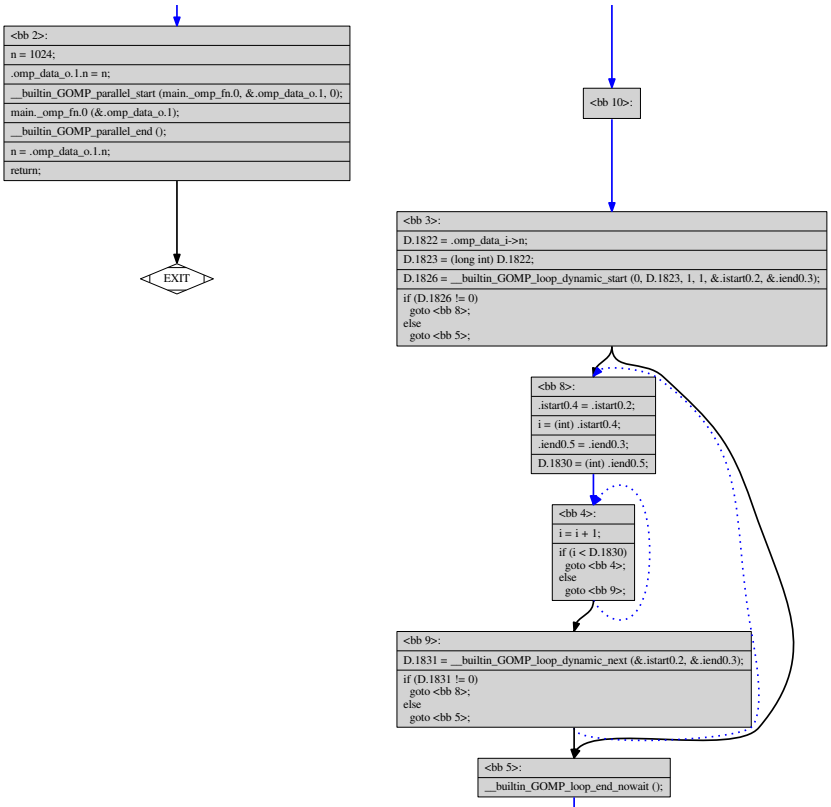
- O código em *assembly* para o segundo formato:

```
1     .file "for-schedule-dynamic-
          upper-bound-variable.c"
2     .text
3     .globl main
4     .type main, @function
5  main:
6     pushq %rbp
7     movq %rsp, %rbp
8     subq $32, %rsp
9     movl $1024, -4(%rbp)
10    movl -4(%rbp), %eax
11    movl %eax, -32(%rbp)
12    leaq -32(%rbp), %rax
13    movl $0, %edx
14    movq %rax, %rsi
15    movl $main._omp_fn.0, %edi
16    call GOMP_parallel_start
17    leaq -32(%rbp), %rax
18    movq %rax, %rdi
19    call main._omp_fn.0
20    call GOMP_parallel_end
21    movl -32(%rbp), %eax
22    movl %eax, -4(%rbp)
23    leave
24    ret
25    .size main, .-main
26    .type main._omp_fn.0, @function
27 main._omp_fn.0:
28    pushq %rbp
29    movq %rsp, %rbp
30    subq $48, %rsp
36    leaq -24(%rbp), %rdx
37    movq %rcx, %r9
38    movq %rdx, %r8
39    movl $1, %ecx
40    movl $1, %edx
41    movq %rax, %rsi
42    movl $0, %edi
43    call GOMP_loop_dynamic_start
44    testb %al, %al
45    je .L3
46 .L5:
47    movq -24(%rbp), %rax
48    movl %eax, -4(%rbp)
49    movq -16(%rbp), %rax
50 .L4:
51    addl $1, -4(%rbp)
52    cmpl %eax, -4(%rbp)
53    jl .L4
54    leaq -16(%rbp), %rdx
55    leaq -24(%rbp), %rax
56    movq %rdx, %rsi
57    movq %rax, %rdi
58    call GOMP_loop_dynamic_next
59    testb %al, %al
60    jne .L5
61 .L3:
62    call GOMP_loop_end_nowait
63    leave
64    ret
65    .size main._omp_fn.0, .-
          main._omp_fn.0
66    .ident "GCC: (Debian 4.8.4-1) 4
```

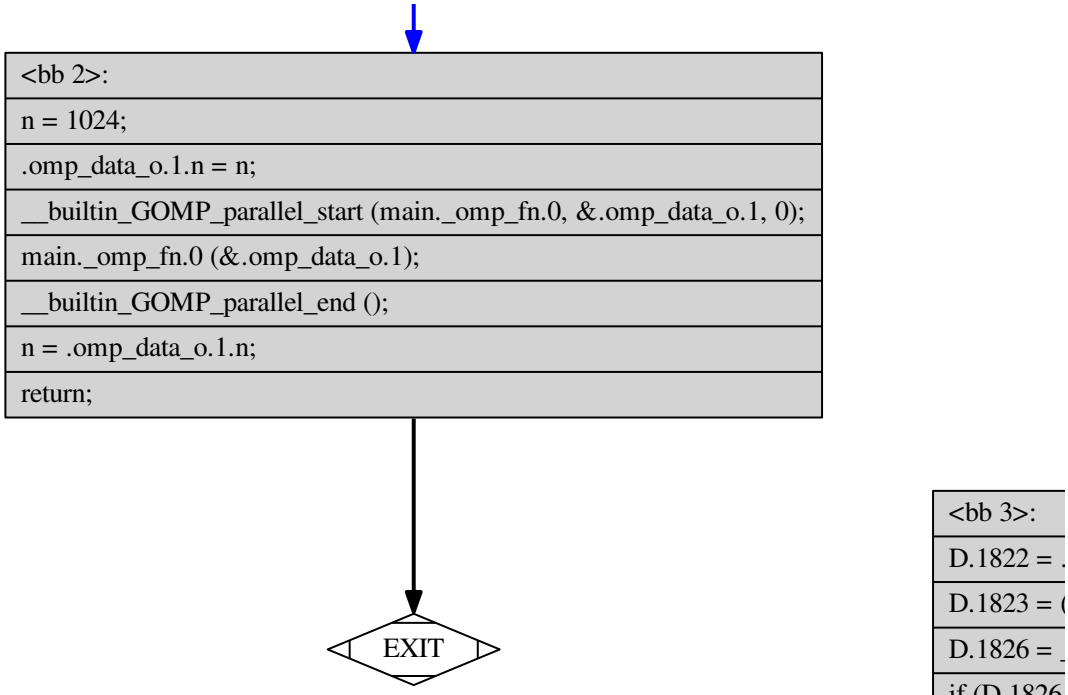

Loops: Construtor `for`



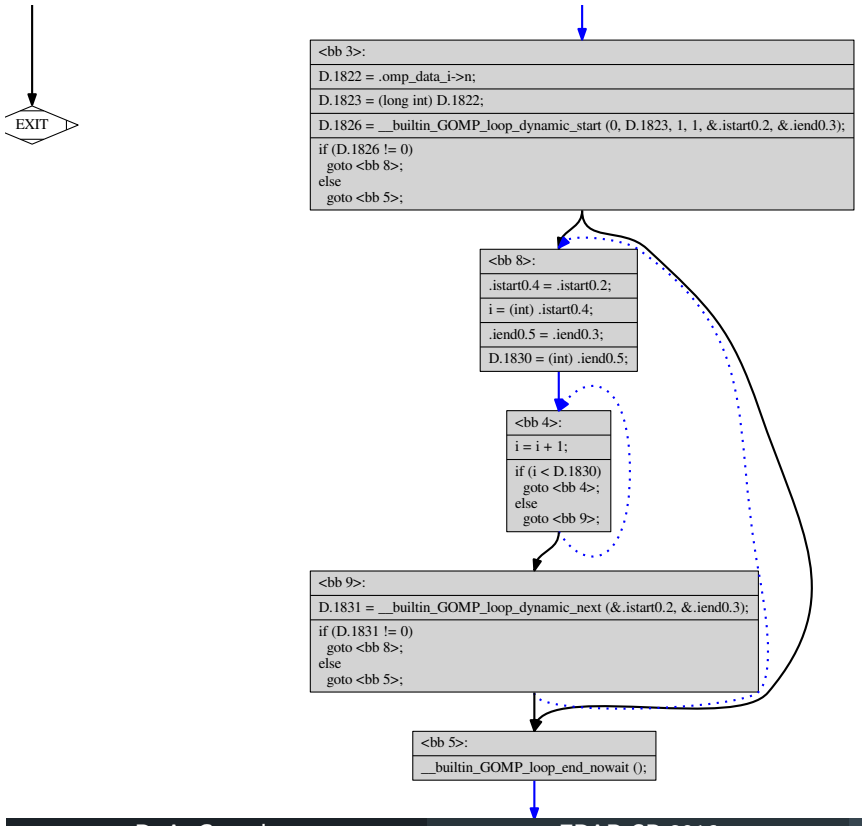
Loops: Construtor `for`



Loops: Construtor `for`



Loops: Construtor `for`



Loops: Construtor `task` I

- O construtor *task* permite a criação de tarefas explícitas.

Construtor *task*:

```
#pragma omp task
{
  // bloco de codigo.
}
```

- Quando uma thread encontra um construtor task, uma nova tarefa é gerada para executar o bloco associado à diretiva.

Loops: Construtor task I

- Para cada construtor *task* é criada uma nova função *(outlined function)* com o código do seu bloco.
- A `libgomp` utiliza as funções para a criação do formato de código para tasks.

ABI da `libgomp` – Funções usadas a diretiva *task*

```
void GOMP_parallel_start (void (*fn) (void *), void *data,
    unsigned num_threads);
void GOMP_parallel_end (void);

void GOMP_task (void (*fn) (void *), void *data, void (*cpyfn) (
    void *, void *),
long arg_size, long arg_align, bool if_clause, unsigned flags,
void **depend);
void GOMP_taskwait (void);
```


Loops: Construtor task II

- Neste exemplo duas *tasks* são criadas dentro de uma região paralela.
- A diretiva *single* é utilizada para garantir que o código seja executado por apenas uma das threads do time.
- Caso contário todas as threads criadas executariam o mesmo código criando cada uma delas duas *tasks*.

```
1  #pragma omp parallel num_threads(8)
2  {
3    #pragma omp single
4    {
5      printf("ESCOLA REGIONAL DE ");
6      #pragma omp task
7      {
8        printf("ALTO ");
9      }
10     #pragma omp task
11     {
12       printf("DESEMPENHO ");
13     }
14     #pragma omp taskwait
15
16     printf("DE SAO PAULO.\n");
17   }
18 }
```

Loops: Construtor task III

- O código em GIMPLE, que é a representação intermediária do GCC:

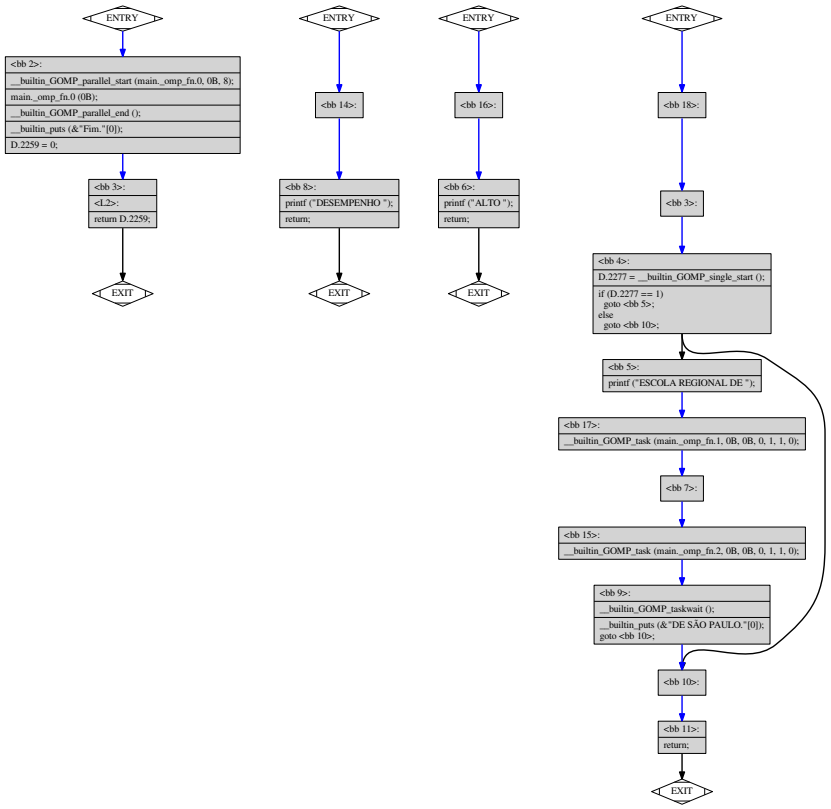
```
main (){

<bb 2>:
  __builtin_GOMP_parallel_start (
      main._omp_fn.0, 0B, 8);
  main._omp_fn.0 (0B);
  __builtin_GOMP_parallel_end ();
  D.2259 = 0;

<L2>:
  return D.2259;
}

main._omp_fn.0 (void * .omp_data_i
    ) {

<bb 18>:

<bb 3>:

<bb 4>:
  D.2277 =
      __builtin_GOMP_single_start
      ();
  if (D.2277 == 1)
    goto <bb 5>;
  else
    goto <bb 10>;

<bb 10>:

<bb 17>:
  __builtin_GOMP_task (main.
      _omp_fn.1, 0B, 0B, 0, 1, 1,
      0);

<bb 7>:

<bb 15>:
  __builtin_GOMP_task (main.
      _omp_fn.2, 0B, 0B, 0, 1, 1,
      0);

<bb 9>:
  __builtin_GOMP_taskwait ();
  __builtin_puts (&"DE SAO PAULO."
      [0]);
  goto <bb 10>;
}

main._omp_fn.2 (void * .omp_data_i
    ) {
<bb 14>:

<bb 8>:
  printf ("DESEMPENHO ");
  return;
}

main._omp_fn.1 (void * .omp_data_i
    ) {
<bb 16>:
```


Loops: Construtor task IV

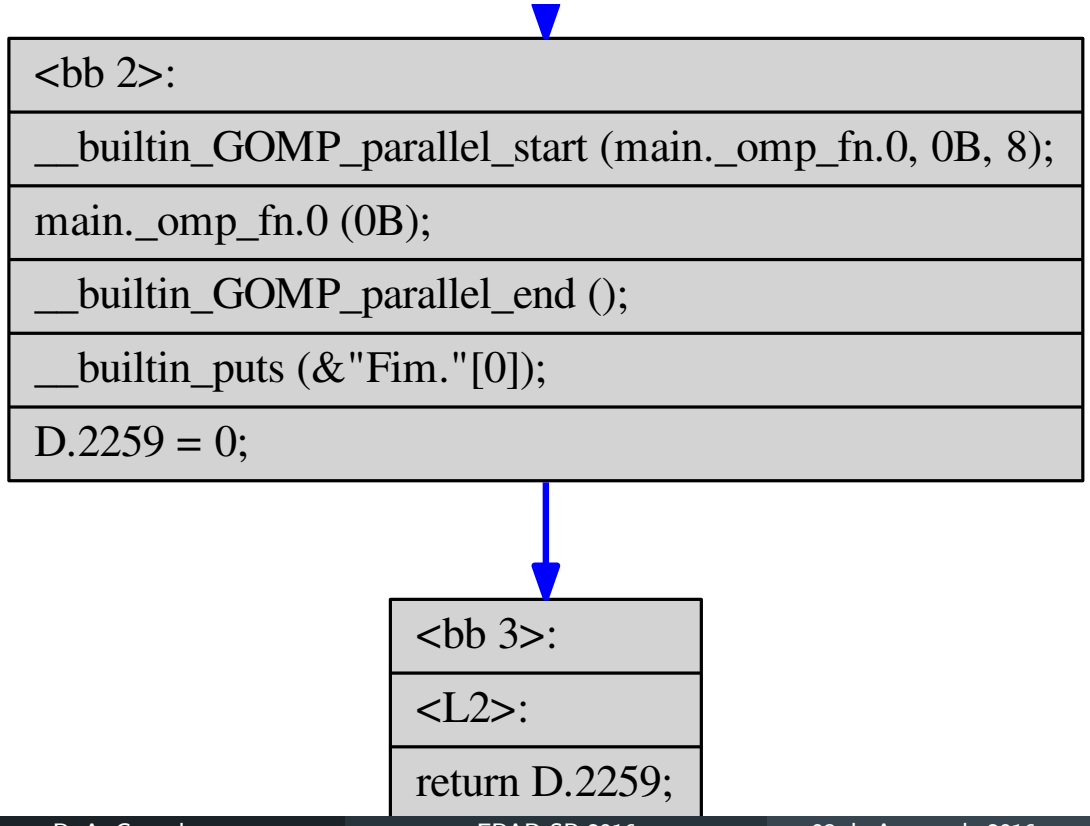
- O código em *assembly*:

```
  .file "task-01.c"
  .section .rodata
.LC0:
  .string "Fim."
  .text
  .globl main
  .type main, @function
main:
  pushq %rbp
  movq %rsp, %rbp
  subq $16, %rsp
  movl %edi, -4(%rbp)
  movq %rsi, -16(%rbp)
  movl $8, %edx
  movl $0, %esi
  movl $main._omp_fn.0, %edi
  call GOMP_parallel_start
  movl $0, %edi
  call main._omp_fn.0
  call GOMP_parallel_end
  movl $.LC0, %edi
  call puts
  movl $0, %eax
  leave
  ret
  .size main, .-main
  .section .rodata
  .LC1:
  .string "ESCOLA REGIONAL DE "
.LC2:
  .string "DE SAO PAULO."
  movl $1, %r8d
  movl $0, %ecx
  movl $0, %edx
  movl $0, %esi
  movl $main._omp_fn.2, %edi
  call GOMP_task
  call GOMP_taskwait
  movl $.LC2, %edi
  call puts
.L3:
  leave
  ret
  .size main._omp_fn.0, .-
      main._omp_fn.0
  .section .rodata
.LC3:
  .string "ALTO "
  .text
  .type main._omp_fn.1, @function
main._omp_fn.1:
  pushq %rbp
  movq %rsp, %rbp
  subq $16, %rsp
  movq %rdi, -8(%rbp)
  movl $.LC3, %edi
  movl $0, %eax
  call printf
  leave
  ret
  .size main._omp_fn.1, .-
      main._omp_fn.1
  .section .rodata
```

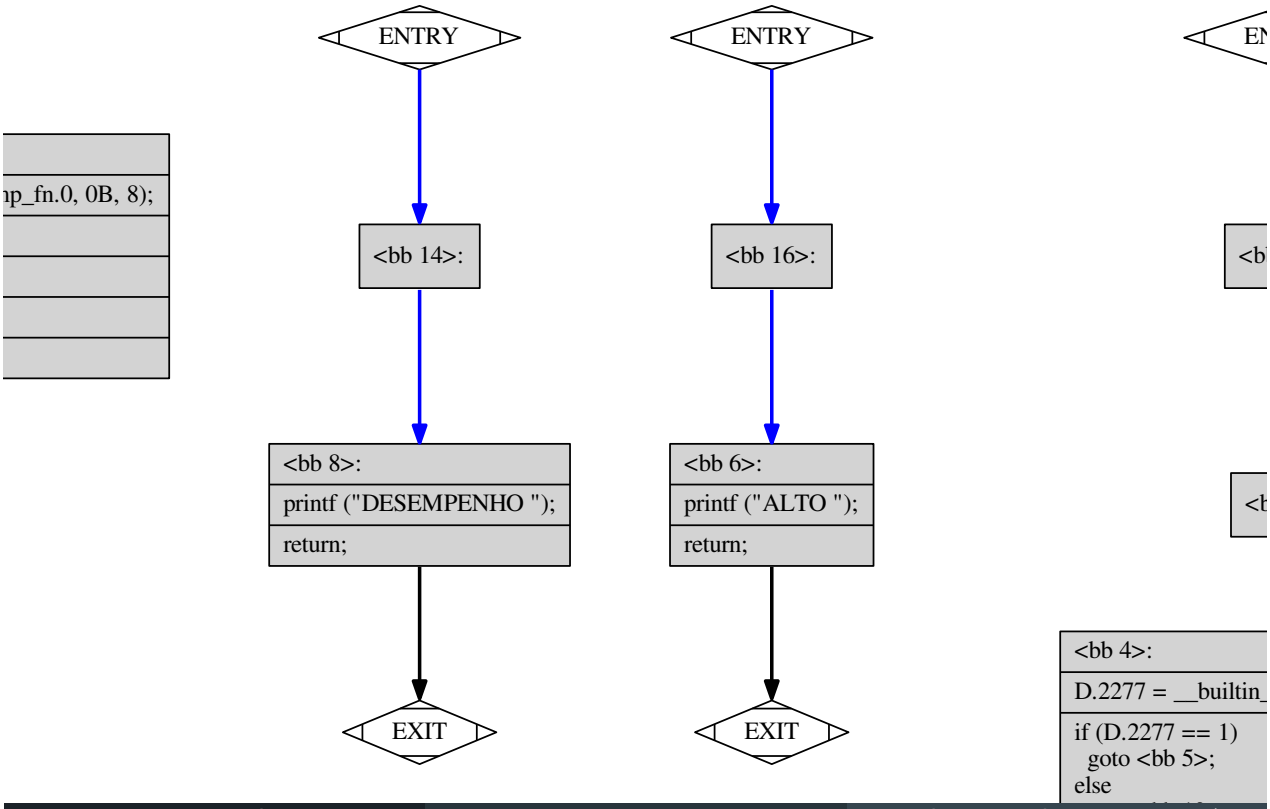
Loops: Construtor task



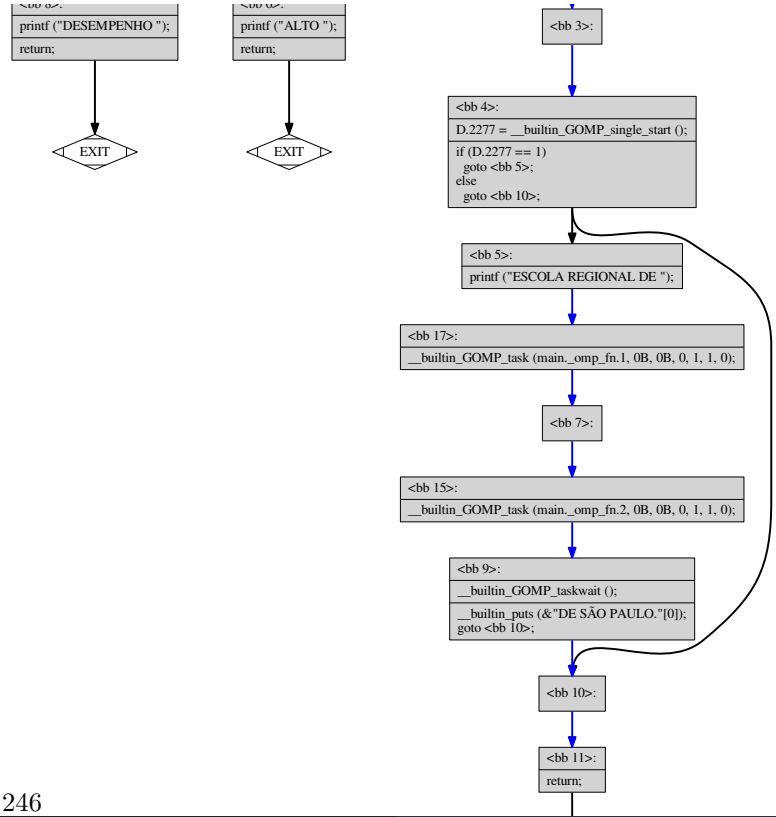
Loops: Construtor task



Loops: Construtor task



Loops: Construtor task



Formato de Código Interceptável

Formato de Código OpenMP Interceptável I

```
1 #pragma omp parallel for schedule(dynamic) chunk_size(N/num_threads)
2 for (i = 0; i < n; i++) {...}
```

Application	Call function/Operation	First format (chunk is a variable or one expression)
Parallel Region Start		`GOMP_parallel_start (main._omp_fn.0, &.omp_data_o.1, 4);`
		>>> Create and start the team. `gomp_team_start(...)`
Call function	***main._omp_fn.0 (&.omp_data_o.1)***	***Function Context***
Loop Start	***Initial Chunk***	`GOMP_loop_dynamic_start (0, 1025, 1, D.1753, &.istart0.2, &.iend0.3);`
		>>> Create and start work share: Loop initialization. ***>>> Get the first set of iterations.*** `gomp_loop_init(…), gomp_iter_dynamic_next_*(...)`
	Execution	`<bb 9>: .istart0.4 = .istart0.2; i = (int) .istart0.4; .iend0.5 = .iend0.3; D.1760 = (int) .iend0.5;` `<bb 4>: D.1761 = (long unsigned int) i; D.1762 = D.1761 * 4; D.1763 = .omp_data_i->a; D.1764 = D.1763 + D.1762; *D.1764 = 1; i = i + 1; if (i < D.1760) goto <bb 4>; else goto <bb 10>;`
	Next Chunk	`GOMP_loop_dynamic_next (&.istart0.2, &.iend0.3);`
		>>> Get the next set of iterations. `gomp_iter_dynamic_next_*(...)`
Loop End	***Return of Function Call***	`GOMP_loop_end_nowait ();`
		>>> Finish the work share. `gomp_work_share_end_nowait()`
Parallel Region End		`GOMP_parallel_end ();`
		>>> Finish the parallel region. `gomp_team_end ();`

Formato de Código OpenMP Interceptável II

```
1 #pragma omp parallel for schedule(dynamic) chunk_size(64)
2 for (i = 0; i < n; i++) {...}
```

Application	Call function/Operation	Second format (chunk is a value or a constant)
Parallel Region Start		`GOMP_parallel_loop_dynamic_start (main._omp_fn.0, &.omp_data_o.1, 4, 0, 1025, 1, 4);` ***>>> Create and start the team, with Initialization of loop.*** *gomp_new_team(…), gomp_loop_init(...), gomp_team_start(...)*
Call function	***main._omp_fn.0(&.omp_data_o.1)***	
Loop Start	*Initial Chunk*	`GOMP_loop_dynamic_next (&.istart0.2, &.iend0.3);` ***>>> Get the first set of iterations.*** *gomp_iter_dynamic_next_*(...)*

	Execution	`<bb 9>:`
		`.istart0.4 = .istart0.2;`
		`i = (int) .istart0.4;`
		`.iend0.5 = .iend0.3;`
		`D.1760 = (int) .iend0.5;`
		
		`<bb 4>:`
		`D.1761 = (long unsigned int) i;`
		`D.1762 = D.1761 * 4;`
		`D.1763 = .omp_data_i->a;`
		`D.1764 = D.1763 + D.1762;`
		`*D.1764 = 1;`
		`i = i + 1;`
		`if (i < D.1760)`
		`goto <bb 4>;`
		`else`
		`goto <bb 10>;`

	Next Chunk	`GOMP_loop_dynamic_next (&.istart0.2, &.iend0.3);` ***>>> Get the next set of iterations.*** *gomp_iter_dynamic_next_*(...)*
Loop End	*Return of Function Call*	`GOMP_loop_end_nowait ();` ***>>> Finish the work share.*** *gomp_work_share_end_nowait()*
Parallel Region End		`GOMP_parallel_end ();` ***>>> Finish the parallel region.*** *gomp_team_end ();*

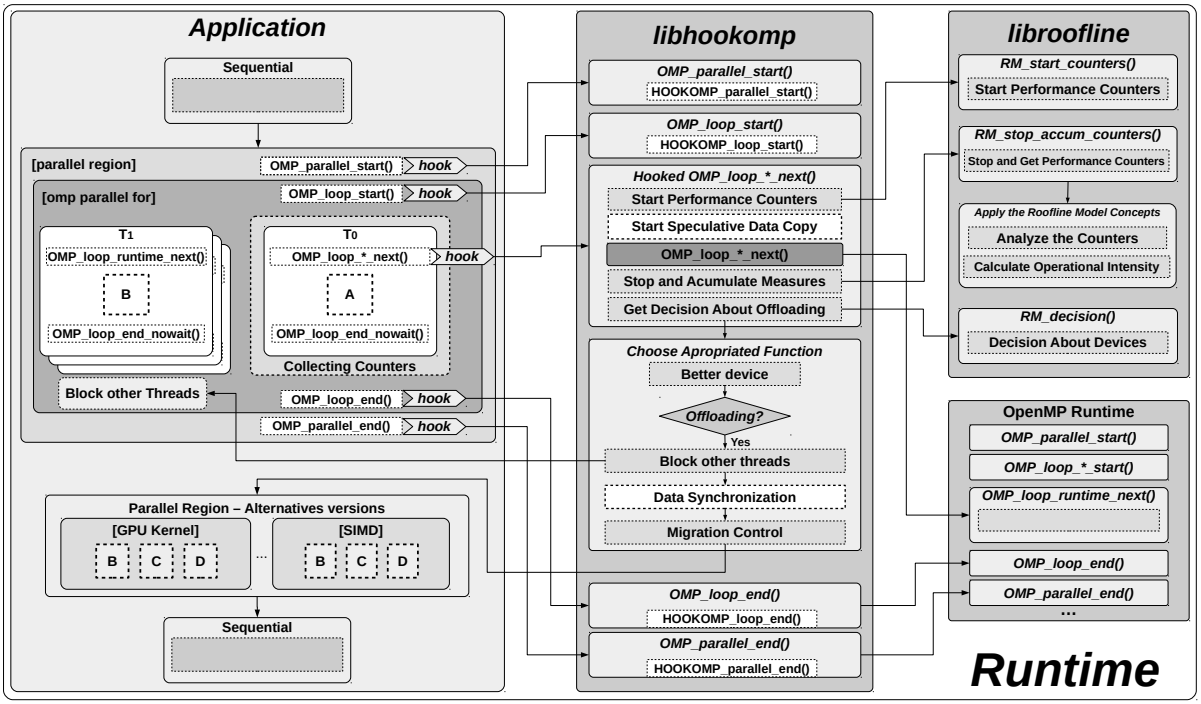
Hook para OpenMP

Hook para OpenMP I

- O conceito de interceptação por *hooking* pode ser utilizado no desenvolvimento de bibliotecas.
- Para criar um *hooks* para funções da `libgomp` é necessário criar uma biblioteca que tenha funções com o mesmo nome das funções disponibilizadas via ABI.
- Bibliotecas que podem ser pré-carregadas para alterarem o comportamento da execução de aplicações.
- Essa técnica pode ser utilizada para a execução de código pré ou pós chamada ao runtime `OpenMP`.
- O que pode cobrir desde *logging*, criação de *traces*[a] , monitoramento para avaliação de desempenho[b] ou *offloading* de código para dispositivos aceleradores.

[a] Trahay et al. (2011)
[b] Mohr et al. (2002)

Interação entre a Aplicação e as bibliotecas do runtime



Interceptando por hooking I

- Uma vez que a biblioteca de *hooking* seja carregada antes da biblioteca `libgomp`, os símbolos como as chamadas para as funções do *runtime* do OpenMP serão ligados aos símbolos da biblioteca de interceptação.
- A ideia é recuperar do *linker* via *dlsym* um ponteiro para a função original para que a chamada original possa ser feita de dentro da função *proxy*.

```
1 void GOMP_parallel_start (void (*fn) (void *), void *data,
      unsigned num_threads){
2   PRINT_FUNC_NAME;
3
4   /* Retrieve the OpenMP runtime function. */
5   typedef void (*func_t) (void (*fn) (void *), void *, unsigned
        );
6   func_t lib_GOMP_parallel_start = (func_t) dlsym(RTLD_NEXT, "
        GOMP_parallel_start");
7
8   lib_GOMP_parallel_start(fn, data, num_threads);
9 }
```


Interceptando por hooking II

- Uma macro pode ser definida para recuperar os ponteiros para as funções originais do runtime `OpenMP`:

```
1 #define GET_RUNTIME_FUNCTION(hook_func_pointer,func_name) \
2 do { \
3   if (hook_func_pointer) break; \
4   void *__handle = RTLD_NEXT; \
5   hook_func_pointer = (typeof(hook_func_pointer)) (uintptr_t)
        dlsym(__handle, func_name); \
6   PRINT_ERROR(); \
7 } while(0)
8
9 #if defined(VERBOSE) && VERBOSE > 0
10   #define PRINT_FUNC_NAME fprintf(stderr, "TRACE-FUNC-NAME:
        [%10s:%07d] Thread [%lu] is calling [%s()]\n",__FILE__,
        __LINE__, (long int) pthread_self(), __FUNCTION__)
11 #else
12   #define PRINT_FUNC_NAME (void) 0
13 #endif
```

Interceptando por hooking I

- Função proxy para a função original utilizando a macro.
- Chamadas de funções para executar algum código antes (`PRE_`) ou algum código depois (`POST_`).

```
1 void GOMP_parallel_start (void (*fn) (void *), void *data,
      unsigned num_threads){
2   PRINT_FUNC_NAME;
3
4   /* Recupera o ponteiro para a funcao OpenMP. */
5   GET_RUNTIME_FUNCTION(lib_GOMP_parallel_start, "
        GOMP_parallel_start");
6
7   /* Codigo a ser executado antes. */
8   PRE_GOMP_parallel_start();
9
10  /* Chamada a funcao original. */
11  lib_GOMP_parallel_start(fn, data, num_threads);
12
13  /* Codigo a ser executado depois. */
14  POST_GOMP_parallel_start();
15 }
```


Fim

Obrigado!

Referências I

Dagum, L. and Menon, R. (1998). OpenMP: An Industry-Standard API for Shared-Memory Programming. *IEEE Computational Science and Engineering*, 5(1):46–55.

GCC (2015). Gcc, the gnu compiler collection.

GNU Libgomp (2015a). GNU libgomp, GNU Offloading and Multi Processing Runtime Library documentation (Online manual).

GNU Libgomp (2015b). GNU Offloading and Multi Processing Runtime Library: The GNU OpenMP and OpenACC Implementation. Technical report, GNU.

GNU Libgomp (2015c). GNU Offloading and Multi Processing Runtime Library: The GNU OpenMP and OpenACC Implementation. Technical report, GNU libgomp.

GNU Libgomp (2016a). GNU Offloading and Multi Processing Runtime Library: The GNU OpenMP and OpenACC Implementation. Technical report, GNU libgomp.

GNU Libgomp (2016b). GNU Offloading and Multi Processing Runtime Library: The GNU OpenMP and OpenACC Implementation. Technical report, GNU libgomp.

Intel (2016a). Intel® OpenMP* Runtime Library Interface. Technical report, Intel. OpenMP* 4.5.

Intel (2016b). Openmp* support.

Referências II

Lattner, C. and Adve, V. (2004). LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proceedings of the International Symposium on Code Generation and Optimization*, number c in CGO '04, pages 75–86, Palo Alto, California. IEEE Computer Society.

LLVM Clang (2015). clang: a C language family frontend for llvm.

LLVM OpenMP (2015). OpenMP®: Support for the OpenMP language.

Mohr, B., Malony, A. D., Shende, S., and Wolf, F. (2002). Design and Prototype of a Performance Tool Interface for OpenMP. *The Journal of Supercomputing*, 23(1):105–128.

OpenACC (2011). OpenACC Application Programming Interface. Version 1.0.

OpenACC (2012). OpenACC Directives for Accelerators Site.

OpenACC (2013). OpenACC Application Programming Interface. Version 2.0.

OpenACC (2015a). OpenACC Application Programming Interface. Version 2.5.

OpenACC (2015b). OpenACC Directives for Accelerators.

OpenMP-ARB (2011). OpenMP Application Program Interface Version 3.1. Technical report, OpenMP Architecture Review Board (ARB).

OpenMP-ARB (2013). OpenMP Application Program Interface Version 4.0. Technical report, OpenMP Architecture Review Board (ARB).

OpenMP-ARB (2015). OpenMP Application Program Interface Version 4.5. Technical report, OpenMP Architecture Review Board (ARB). Version 4.5.

252

Referências III

Trahay, F., Rue, F., Faverge, M., Ishikawa, Y., Namyst, R., and Dongarra, J. (2011). EZTrace: a generic framework for performance analysis. In *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, Newport Beach, CA, United States. Poster Session.

Contatos

- Rogério Gonçalves: `rogerioag@utfpr.edu.br, rag@ime.usp.br`
- Alfredo Goldman: `gold@ime.usp.br`

Agradecimentos

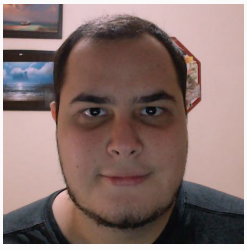
Os autores agradecem à CAPES (BEX 3401/15-4, CAPES-COFECUB Projeto No. 828/15, *Choosing: Cooperation on Hybrid Computing Clouds for Energy Saving*) pela bolsa e a Fundação Araucária, Departamento de Ciência, Tecnologia e de Ensino Superior do Estado do Paraná (SETI-PR) e o Governo do Estado do Paraná pelo financiamento que viabilizou o DINTER e a realização deste trabalho.

Introdução à Programação de GPUs com a Plataforma CUDA

Pedro Bruel
phrb@ime.usp.br
04 de Agosto de 2016

Instituto de Matemática e Estatística
Universidade de São Paulo

Sobre



Pedro Bruel

- phrb@ime.usp.br
- www.ime.usp.br/~phrb
- github.com/phrb

Alfredo Goldman

- gold@ime.usp.br
- www.ime.usp.br/~gold

Pesquisa

Meus interesses de pesquisa:

- *Autotuning*
- *Stochastic Local Search*
- *Model Based Search*

Pesquisa

Meus interesses de pesquisa:

- *Autotuning*
- *Stochastic Local Search*
- *Model Based Search*
- GPUs, FPGAs, *cloud*
- Julia Language

Pesquisa

Meus interesses de pesquisa:

- *Autotuning*
- *Stochastic Local Search*
- *Model Based Search*
- GPUs, FPGAs, *cloud*
- Julia Language
- Colaboração! (phrb@ime.usp.br)

Parte I

1. Introdução
2. Computação Heterogênea
3. GPUs
4. Plataforma CUDA
5. CUDA C

Parte II

Slides

O *pdf* com as aulas e todo o código fonte estão no GitHub:

- `github.com/phrb/intro-cuda`

Aceleração por *Hardware*

Uso de dispositivos (*devices*) para acelerar computações aplicadas a grandes conjuntos de dados:

Aceleração por *Hardware*

Uso de dispositivos (*devices*) para acelerar computações aplicadas a grandes conjuntos de dados:

- Associação a um processador hospedeiro (*host*)

Aceleração por *Hardware*

Uso de dispositivos (*devices*) para acelerar computações aplicadas a grandes conjuntos de dados:

- Associação a um processador hospedeiro (*host*)
- Controle e memória próprios

Aceleração por *Hardware*

Uso de dispositivos (*devices*) para acelerar computações aplicadas a grandes conjuntos de dados:

- Associação a um processador hospedeiro (*host*)
- Controle e memória próprios
- Diferem em especialização e configurabilidade

Aceleração por *Hardware*

Uso de dispositivos (*devices*) para acelerar computações aplicadas a grandes conjuntos de dados:

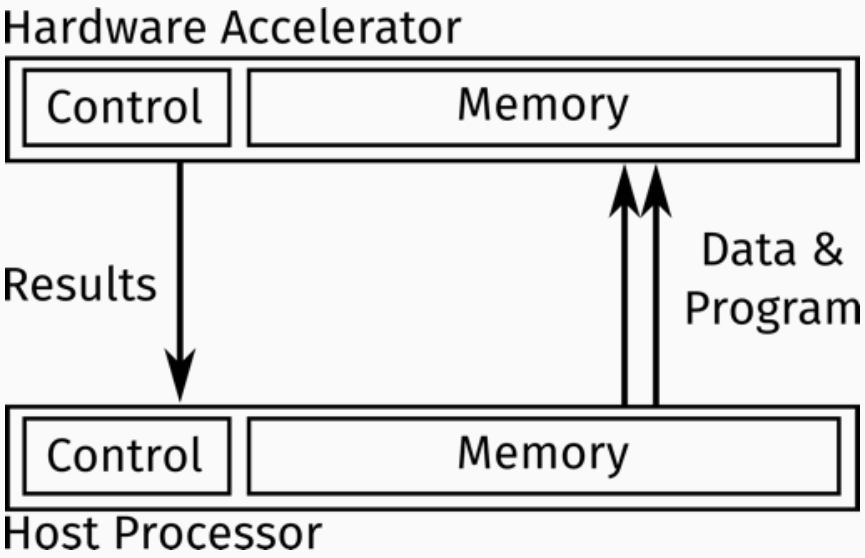
- Associação a um processador hospedeiro (*host*)
- Controle e memória próprios
- Diferem em especialização e configurabilidade
- GPUs, DSPs, FPGAs, ASICs

Aceleração por *Hardware*

Casos de uso:

- Aprendizagem Computacional
- Processamento Digital de Sinais e Imagens
- Bioinformática
- Criptografia
- Meteorologia
- Simulações
- ...

Aceleração por *Hardware*



Aceleração por *Hardware*

Porcentagem de sistemas com aceleradores na *Top500*:

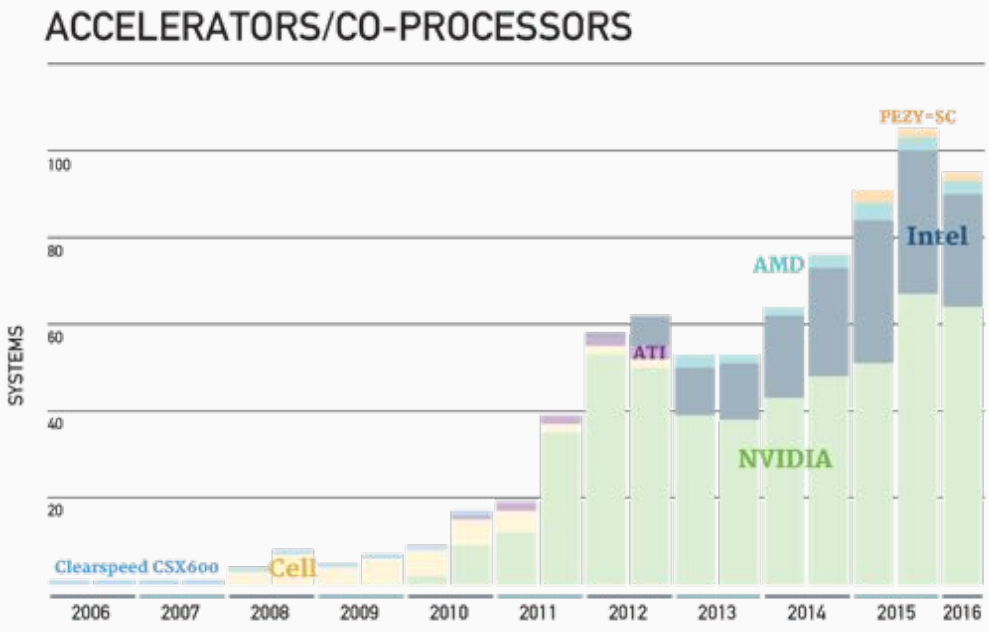


Imagem: `top500.org/lists/2016/06/download/TOP500_201606_Poster.pdf` [Acessado em 29/07/16]

Computação Heterogênea

Computação Heterogênea

Dados recursos computacionais heterogêneos, conjuntos de dados e computações, como distribuir computações e dados de forma a otimizar o uso dos recursos?

Imagem: `olcf.ornl.gov/titan` [Acessado em 29/07/16]

Computação Heterogênea

Recursos computacionais heterogêneos:

Computação Heterogênea

Recursos computacionais heterogêneos:

- Baixa latência: CPUs

Computação Heterogênea

Recursos computacionais heterogêneos:

- Baixa latência: CPUs
- Alta vazão: GPUs

Computação Heterogênea

Recursos computacionais heterogêneos:

- Baixa latência: CPUs
- Alta vazão: GPUs
- Reconfiguráveis: FPGAs

Computação Heterogênea

Recursos computacionais heterogêneos:

- Baixa latência: CPUs
- Alta vazão: GPUs
- Reconfiguráveis: FPGAs
- Especializados: ASICs, DSPs

Computação Heterogênea

Recursos computacionais heterogêneos:

- Baixa latência: CPUs
- Alta vazão: GPUs
- Reconfiguráveis: FPGAs
- Especializados: ASICs, DSPs
- Memória?

Computação Heterogênea

Conjuntos de dados:

- *Big Data*
- *Data Streams*
- ...

Computação Heterogênea

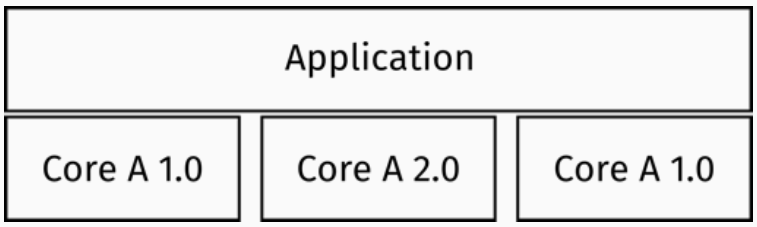
Conjuntos de dados:

- *Big Data*
- *Data Streams*
- ...

Modelos de programação (computações):

- *MapReduce*
- *Task Parallelism*
- ...

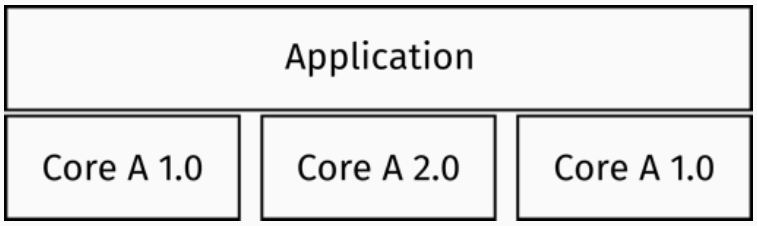
Computação Heterogênea: Escalabilidade



Escalabilidade significa não perder desempenho executando em:

- Múltiplos *cores* idênticos

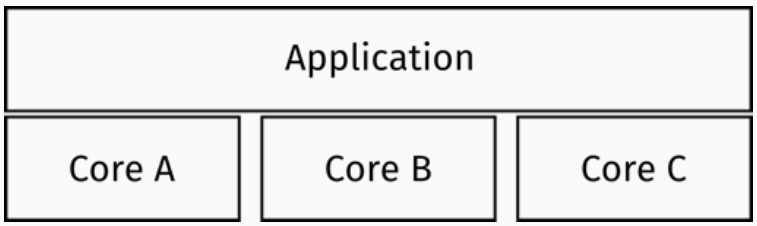
Computação Heterogênea: Escalabilidade



Escalabilidade significa não perder desempenho executando em:

- Múltiplos *cores* idênticos
- Novas versões do mesmo *core*

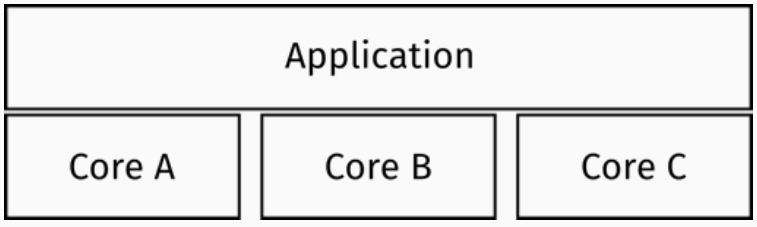
Computação Heterogênea: Portabilidade



Portabilidade significa não perder desempenho executando em diferentes:

- *Cores* ou aceleradores

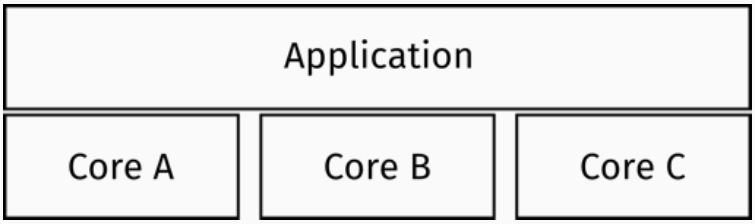
Computação Heterogênea: Portabilidade



Portabilidade significa não perder desempenho executando em diferentes:

- *Cores* ou aceleradores
- Modelos de memória

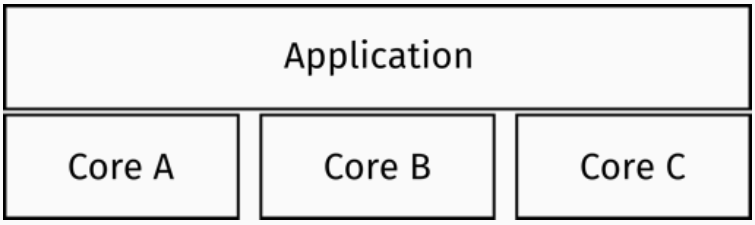
Computação Heterogênea: Portabilidade



Portabilidade significa não perder desempenho executando em diferentes:

- *Cores* ou aceleradores
- Modelos de memória
- Modelos de paralelismo

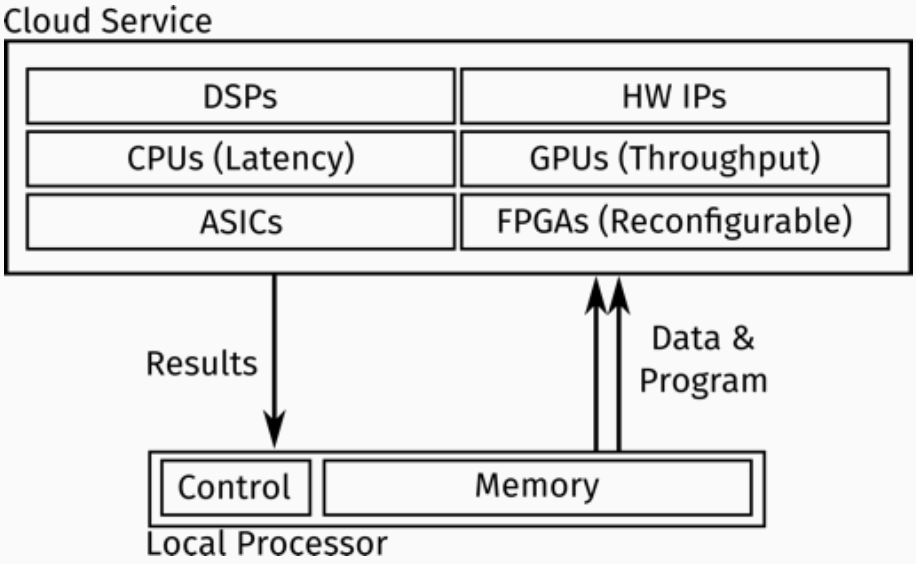
Computação Heterogênea: Portabilidade



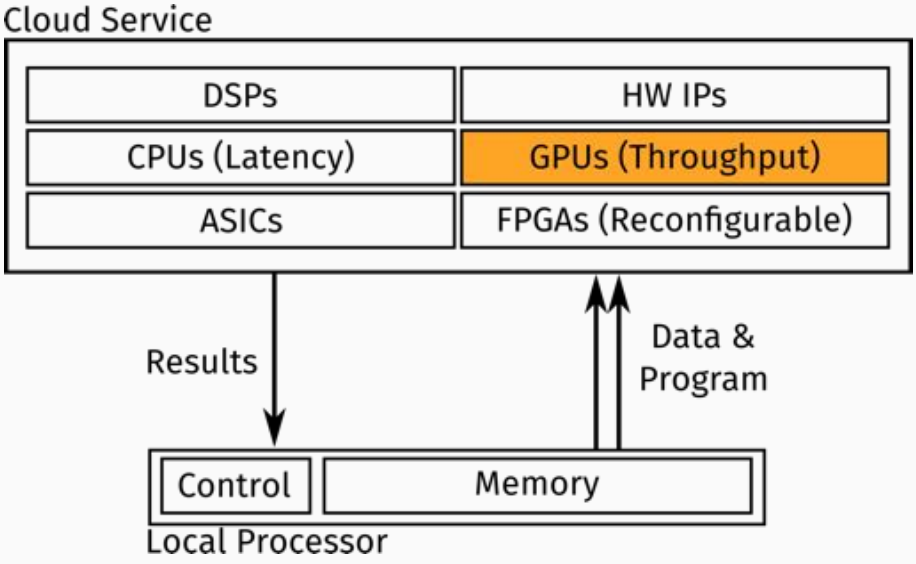
Portabilidade significa não perder desempenho executando em diferentes:

- *Cores* ou aceleradores
- Modelos de memória
- Modelos de paralelismo
- Conjuntos de instruções

Computação Heterogênea



Computação Heterogênea



Graphics Processing Units

Graphics Processing Units

Originalmente especializadas em processamento gráfico, trabalham com muitos dados e têm alta vazão:

Graphics Processing Units

Originalmente especializadas em processamento gráfico, trabalham com muitos dados e têm alta vazão:

- Caches pequenos (*kilobytes*)

Graphics Processing Units

Originalmente especializadas em processamento gráfico, trabalham com muitos dados e têm alta vazão:

- Caches pequenos (*kilobytes*)
- Sem branch prediction

Graphics Processing Units

Originalmente especializadas em processamento gráfico, trabalham com muitos dados e têm alta vazão:

- Caches pequenos (*kilobytes*)
- Sem branch prediction
- Milhares de ALUs de maior latência

Graphics Processing Units

Originalmente especializadas em processamento gráfico, trabalham com muitos dados e têm alta vazão:

- Caches pequenos (*kilobytes*)
- Sem branch prediction
- Milhares de ALUs de maior latência
- Pipelines de execução

Graphics Processing Units

Originalmente especializadas em processamento gráfico, trabalham com muitos dados e têm alta vazão:

- Caches pequenos (*kilobytes*)
- Sem branch prediction
- Milhares de ALUs de maior latência
- Pipelines de execução
- 114 688 threads concorrentes na arquitetura Pascal

Graphics Processing Units

Hoje são usadas em computação de propósito geral, podem ser chamadas de *General Purpose GPUs* (GPGPUs):

Graphics Processing Units

Hoje são usadas em computação de propósito geral, podem ser chamadas de *General Purpose GPUs* (GPGPUs):

- *OpenGL*: 1992
- *DirectX*: 1995
- CUDA: 2007
- *OpenCL*: 2009
- *Vulkan*: 2016

Modelo de *Hardware*

Taxonomia de Flynn:

- *Single Instruction Multiple Data* (SIMD)

Modelo de *Hardware*

Taxonomia de Flynn:

- *Single Instruction Multiple Data* (SIMD)
- *Single Instruction Multiple Thread* (SIMT)?

Modelo de *Hardware*

Taxonomia de Flynn:

- *Single Instruction Multiple Data* (SIMD)
- *Single Instruction Multiple Thread* (SIMT)?

Escalonamento e execução:

- *Streaming Multiprocessor* (SM)

Modelo de *Hardware*

Taxonomia de Flynn:

- *Single Instruction Multiple Data* (SIMD)
- *Single Instruction Multiple Thread* (SIMT)?

Escalonamento e execução:

- *Streaming Multiprocessor* (SM)
- *Warps*

Modelo de *Hardware*

Taxonomia de Flynn:

- *Single Instruction Multiple Data* (SIMD)
- *Single Instruction Multiple Thread* (SIMT)?

Escalonamento e execução:

- *Streaming Multiprocessor* (SM)
- *Warps*
- *Grids, blocks* e *threads*

Modelo de *Hardware*

Escalonamento de mais alto-nível:

- *Pipelines*
- *Texture Processing Cluster* (TPC)
- *Graphics Processing Cluster* (GPC)

Modelo de *Hardware*

Escalonamento de mais alto-nível:

- *Pipelines*
- *Texture Processing Cluster* (TPC)
- *Graphics Processing Cluster* (GPC)

Memória:

- Compartilhada: Cache L1, L2; *megabytes* (Nvidia Pascal)
- Global: volátil, GDDR5; *gigabytes*

Modelo de *Hardware*

Escalonamento de mais alto-nível:

- *Pipelines*
- *Texture Processing Cluster* (TPC)
- *Graphics Processing Cluster* (GPC)

Memória:

- Compartilhada: Cache L1, L2; *megabytes* (Nvidia Pascal)
- Global: volátil, GDDR5; *gigabytes*
- SSD: não-volátil; *terabytes* (AMD SSG Fiji, 2017)

AMD SSG: `anandtech.com/show/10518/amd-announces-radeon-pro-ssg-fiji-with-m2-ssds-onboard` [Acessado em 30/07/16]

Compute Capability e SIMT

A Compute Capability especifica características de arquiteturas *Single Instruction Multiple Thread* (SIMT):

Compute Capability e SIMT

A Compute Capability especifica características de arquiteturas *Single Instruction Multiple Thread* (SIMT):

- *Warp*: Grupo de *threads* executadas em paralelo
- *Streaming Multiprocessor* (SM): Cria, escalona e executa *warps*

Compute Capability e SIMT

A Compute Capability especifica características de arquiteturas *Single Instruction Multiple Thread* (SIMT):

- *Warp*: Grupo de *threads* executadas em paralelo
- *Streaming Multiprocessor* (SM): Cria, escalona e executa *warps*
- Múltiplas *warps* concorrentes no mesmo SM

Fonte: `docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#hardware-implementation` [Acessado em 29/07/16]

Compute Capability e SIMT

Warps:

Compute Capability e SIMT

Warps:

- Unidade de execução e escalonamento
- 32 *threads*

Compute Capability e SIMT

Warps:

- Unidade de execução e escalonamento
- 32 *threads*
- 1 instrução em comum por vez

Compute Capability e SIMT

Warps:

- Unidade de execução e escalonamento
- 32 *threads*
- 1 instrução em comum por vez
- *Threads ativas*: estão no *branch* atual de execução
- *Threads inativas*: não estão no *branch* atual de execução

Compute Capability e SIMT

Warps:

- Unidade de execução e escalonamento
- 32 *threads*
- 1 instrução em comum por vez
- *Threads ativas*: estão no *branch* atual de execução
- *Threads inativas*: não estão no *branch* atual de execução
- *Threads* fora do *branch* atual executadas sequencialmente

Compute Capability e SIMT

Warps:

- Unidade de execução e escalonamento
- 32 *threads*
- 1 instrução em comum por vez
- *Threads ativas*: estão no *branch* atual de execução
- *Threads inativas*: não estão no *branch* atual de execução
- *Threads* fora do *branch* atual executadas sequencialmente
- Eficiência: sem divergência de execução dentro de *Warps*

Fonte: `docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#hardware-implementation` [Acessado em 29/07/16]

Arquitetura Pascal GP100

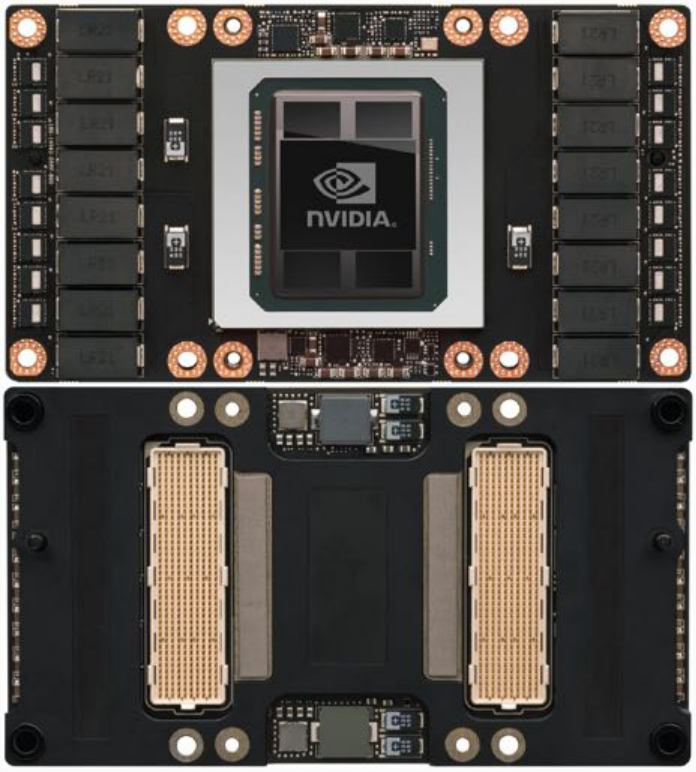


Imagem: images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf [Acessado em 29/07/16]

Arquitetura Pascal GP100: *Compute Capability 6.0*

GPU	Kepler GK110	Maxwell GM200	Pascal GP100
Compute Capability	3.5	5.2	6.0
Threads / Warp	32	32	32
Max Warps / Multiprocessor	64	64	64
Max Threads / Multiprocessor	2048	2048	2048
Max Thread Blocks / Multiprocessor	16	32	32
Max 32-bit Registers / SM	65536	65536	65536
Max Registers / Block	65536	32768	65536
Max Registers / Thread	255	255	255
Max Thread Block Size	1024	1024	1024
Shared Memory Size / SM	16 KB/32 KB/48 KB	96 KB	64 KB

Imagem: images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf [Acessado em 29/07/16]

Arquitetura Pascal GP100: SM

Imagem: `images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf` [Acessado em 29/07/16]

Arquitetura Pascal GP100

Tesla Products	Tesla K40	Tesla M40	Tesla P100
GPU	GK110 (Kepler)	GM200 (Maxwell)	GP100 (Pascal)
SMs	15	24	56
TPCs	15	24	28
FP32 CUDA Cores / SM	192	128	64
FP32 CUDA Cores / GPU	2880	3072	3584
FP64 CUDA Cores / SM	64	4	32
FP64 CUDA Cores / GPU	960	96	1792
Base Clock	745 MHz	948 MHz	1328 MHz
GPU Boost Clock	810/875 MHz	1114 MHz	1480 MHz
Peak FP32 GFLOPs[1]	5040	6840	10600
Peak FP64 GFLOPs[1]	1680	210	5300
Texture Units	240	192	224
Memory Interface	384-bit GDDR5	384-bit GDDR5	4096-bit HBM2
Memory Size	Up to 12 GB	Up to 24 GB	16 GB
L2 Cache Size	1536 KB	3072 KB	4096 KB
Register File Size / SM	256 KB	256 KB	256 KB
Register File Size / GPU	3840 KB	6144 KB	14336 KB
TDP	235 Watts	250 Watts	300 Watts
Transistors	7.1 billion	8 billion	15.3 billion
GPU Die Size	551 mm²	601 mm²	610 mm²
Manufacturing Process	28-nm	28-nm	16-nm FinFET

[1] The GFLOPS in this chart are based on GPU Boost Clocks.

Imagem: `images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf` [Acessado em 29/07/16]

Arquitetura Pascal GP100

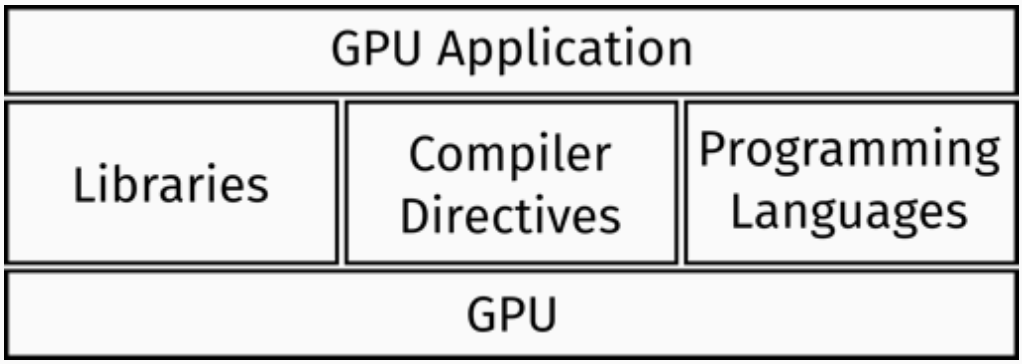
Imagem: images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf [Acessado em 29/07/16]

Plataforma CUDA

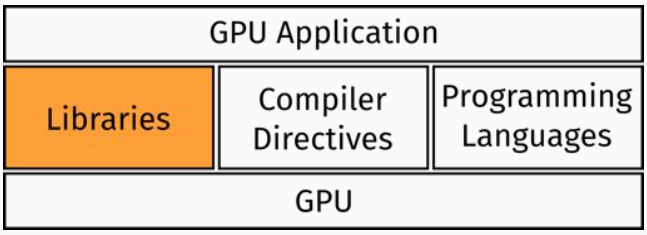
Plataforma CUDA

- Plataforma para computação paralela
- *Application Programming Interface* (API)
- CUDA *Toolkit*

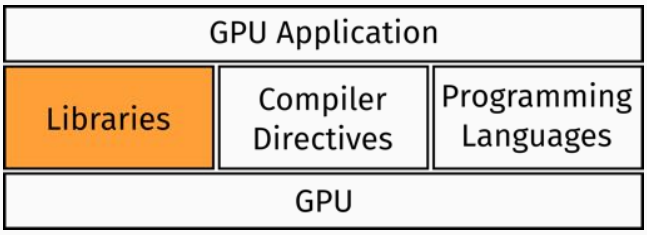
Aceleração por *Software*



Bibliotecas

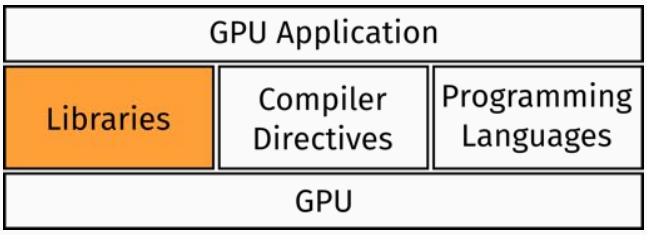


Bibliotecas



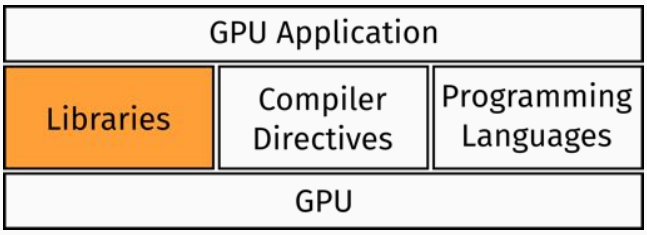
- Fáceis de usar

Bibliotecas



- Fáceis de usar
- Aceleração *Drop-in*

Bibliotecas



- Fáceis de usar
- Aceleração *Drop-in*
- Otimizadas por especialistas

Bibliotecas

Imagem: `developer.nvidia.com/gpu-accelerated-libraries` [Acessado em 29/07/16]

Bibliotecas

Soma de vetores com a biblioteca Thrust:

```
thrust::device_vector<float> device_input1(input_lenght);
thrust::device_vector<float> device_input2(input_lenght);
thrust::device_vector<float> device_output(input_lenght);
```

Bibliotecas

Soma de vetores com a biblioteca Thrust:

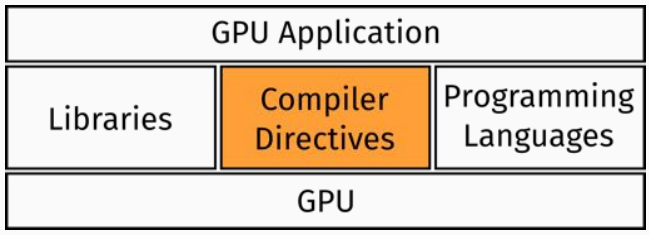
```
thrust::device_vector<float> device_input1(input_lenght);
thrust::device_vector<float> device_input2(input_lenght);
thrust::device_vector<float> device_output(input_lenght);

thrust::copy(host_input1, host_input1 + input_lenght,
    device_input1.begin());

thrust::copy(host_input2, host_input2 + input_lenght,
    device_input2.begin());
```


Bibliotecas

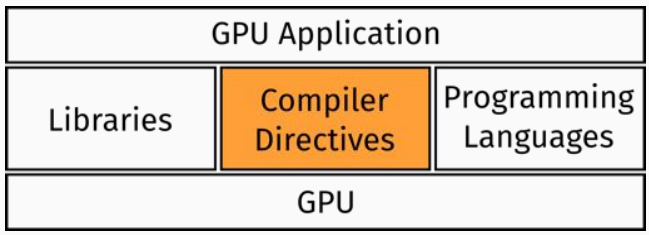
Soma de vetores com a biblioteca Thrust:

```
thrust::device_vector<float> device_input1(input_lenght);
thrust::device_vector<float> device_input2(input_lenght);
thrust::device_vector<float> device_output(input_lenght);

thrust::copy(host_input1, host_input1 + input_lenght,
    device_input1.begin());

thrust::copy(host_input2, host_input2 + input_lenght,
    device_input2.begin());

thrust::transform(device_input1.begin(),
    device_input1.end(), device_input2.begin(),
    device_output.begin(), thrust::plus<float>());
```

Diretivas de Compilação

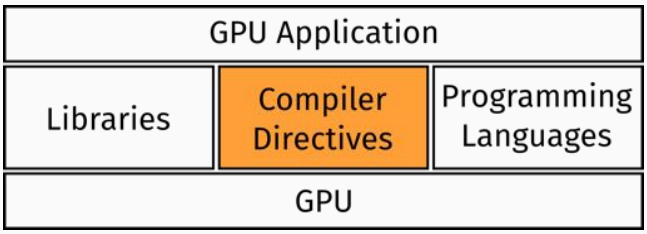


Diretivas de Compilação



- Fáceis de usar

Diretivas de Compilação



- Fáceis de usar
- Portáveis, mas desempenho depende do compilador

Diretivas de Compilação

Soma de vetores com OpenACC:

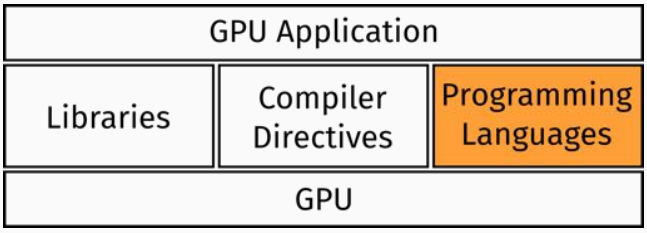
```
#pragma acc data
    copyin(input1[0:input_lenght],input2[0:input_lenght]),
    copyout(output[0:input_lenght])
```

Diretivas de Compilação

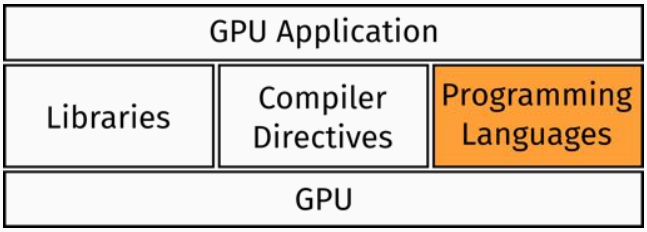
Soma de vetores com OpenACC:

```
#pragma acc data
    copyin(input1[0:input_lenght],input2[0:input_lenght]),
    copyout(output[0:input_lenght])

{
    #pragma acc kernels loop independent
    for(i = 0; i < input_lenght; i++) {
        output[i] = input1[i] + input2[i];
    }
}
```


Linguagens de Programação

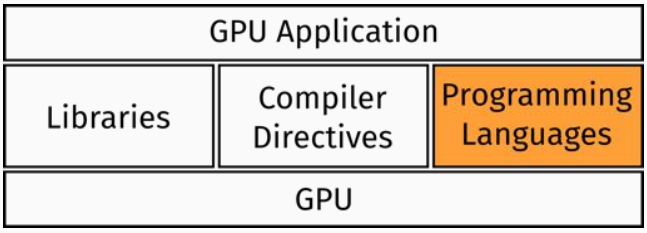


Linguagens de Programação



- Melhor desempenho, mas mais difíceis de usar

Linguagens de Programação



- Melhor desempenho, mas mais difíceis de usar
- Flexíveis

Linguagens de Programação

Implementações de CUDA em:

- C
- Fortran
- C++
- Python
- F#

CUDA C

CUDA C

Modelo de Programação:

CUDA C

Modelo de Programação:

- Kernels

CUDA C

Modelo de Programação:

- Kernels
- Hierarquia de Threads

CUDA C

Modelo de Programação:

- Kernels
- Hierarquia de Threads
- Hierarquia de Memória

CUDA C

Modelo de Programação:

- Kernels
- Hierarquia de Threads
- Hierarquia de Memória
- Programação Heterogênea

CUDA C: *Kernel*

Kernels:

- Kernels são funções C executadas por threads
- *N threads* executam *N kernels*

CUDA C: *Kernel*

Kernels:

- Kernels são funções C executadas por threads
- *N threads* executam *N kernels*
- Acessam ID de suas *threads* pela variável `threadIdx`

CUDA C: *Kernel*

Kernels:

- Kernels são funções C executadas por threads
- *N threads* executam *N kernels*
- Acessam ID de suas *threads* pela variável `threadIdx`
- Definidos usando a palavra-chave `__global__`
- Seu tipo deve ser `void`

Fonte: `docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#programming-model` [Acessado em 29/07/16]

CUDA C: *Kernel*

Kernels:

- Lançados e configurados usando a sintaxe:
  - `kernel_name<<<...>>>(...);`

CUDA C: *Kernel*

Kernels:

- Lançados e configurados usando a sintaxe:
  - `kernel_name<<<...>>>(...);`
- Idealmente, são executados em paralelo

CUDA C: *Kernel*

Kernels:

- Lançados e configurados usando a sintaxe:
  - `kernel_name<<<...>>>(...);`
- Idealmente, são executados em paralelo
- Na prática, o paralelismo depende:
  - Número de *threads* em relação a uma *warp*

CUDA C: *Kernel*

Kernels:

- Lançados e configurados usando a sintaxe:
  - `kernel_name<<<...>>>(...);`
- Idealmente, são executados em paralelo
- Na prática, o paralelismo depende:
  - Número de *threads* em relação a uma *warp*
  - Coerência entre ramos de execução

Fonte: `docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#programming-model` [Acessado em 29/07/16]

CUDA C: *Kernel*

Escrevendo e lançando (launching) um kernel:

```
#include <cuda_runtime.h>

__global__ void VecAdd(float* A, float* B, float* C) {
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}
```


CUDA C: *Kernel*

Escrevendo e lançando (launching) um kernel:

```
#include <cuda_runtime.h>

__global__ void VecAdd(float* A, float* B, float* C) {
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}

int main() {
    ...
    VecAdd<<<1, N>>>(A, B, C);
    ...
}
```

Fonte: `docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#programming-model` [Acessado em 29/07/16]

CUDA C: Hierarquia de *Threads*

Thread Block:

- Agrupamentos de *Threads*

CUDA C: Hierarquia de *Threads*

Thread Block:

- Agrupamentos de *Threads*
- Tridimensionais: `Db = (Dx, Dy, Dz)`

CUDA C: Hierarquia de *Threads*

Thread Block:

- Agrupamentos de *Threads*
- Tridimensionais: `Db = (Dx, Dy, Dz)`
- Tamanho máximo de 1024 *threads*

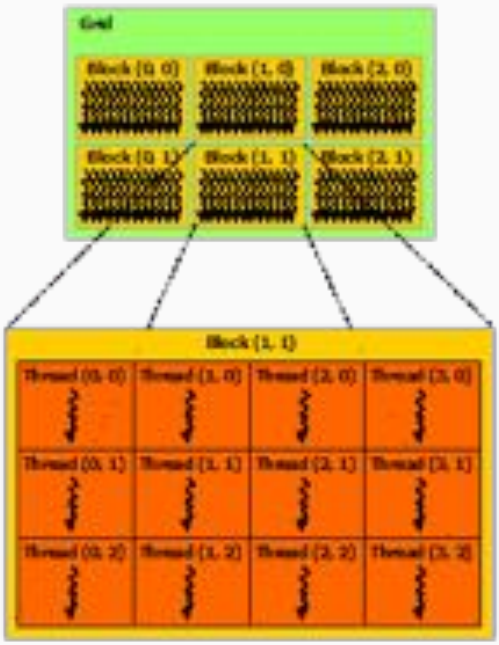
CUDA C: Hierarquia de *Threads*

Thread Block:

- Agrupamentos de *Threads*
- Tridimensionais: `Db = (Dx, Dy, Dz)`
- Tamanho máximo de 1024 *threads*
- Um Grid é um agrupamento tridimensional de *blocks*

Fonte: `docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#programming-model` [Acessado em 29/07/16]

CUDA C: Hierarquia de *Threads*



Fonte: `docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#programming-model` [Acessado em 29/07/16]

Relembrando: SM da arquitetura Pascal GP100

Imagem: `images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf` [Acessado em 29/07/16]

CUDA C: Hierarquia de *Threads*

```
__global__ void MatAdd(float A[N][N], float B[N][N], float
    C[N][N]) {
    int i = threadIdx.x;
    int j = threadIdx.y;
    C[i][j] = A[i][j] + B[i][j];
}
```


CUDA C: Hierarquia de *Threads*

```
__global__ void MatAdd(float A[N][N], float B[N][N], float
    C[N][N]) {
    int i = threadIdx.x;
    int j = threadIdx.y;
    C[i][j] = A[i][j] + B[i][j];
}

int main() {
    ...
    int numBlocks = 1;
    dim3 threadsPerBlock(N, N);
    MatAdd<<<numBlocks, threadsPerBlock>>>(A, B, C);
    ...
}
```

Fonte: `docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#programming-model` [Acessado em 29/07/16]

CUDA C: Hierarquia de *Threads*

```
__global__ void MatAdd(float A[N][N], float B[N][N], float
    C[N][N]) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;
    if (i < N && j < N) {
        C[i][j] = A[i][j] + B[i][j];
    }
}
```


CUDA C: Hierarquia de *Threads*

```
__global__ void MatAdd(float A[N][N], float B[N][N], float
    C[N][N]) {
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;
    if (i < N && j < N) {
        C[i][j] = A[i][j] + B[i][j];
    }
}

int main() {
    ...
    dim3 threadsPerBlock(16, 16);
    dim3 numBlocks(N / threadsPerBlock.x, N /
        threadsPerBlock.y);
    MatAdd<<<numBlocks, threadsPerBlock>>>(A, B, C);
    ...
}
```

Fonte: docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#programming-model [Acessado em 29/07/16]

CUDA C: Hierarquia de *Threads*

Sobre a sintaxe de lançamento e configuração `<<< Dg, Db, Ns, S >>>`:

- `dim3` **Dg** determina dimensão e tamanho do *grid*:

  `Dg.x * Dg.y * Dg.z = numBlocks`

CUDA C: Hierarquia de *Threads*

Sobre a sintaxe de lançamento e configuração `<<< Dg, Db, Ns, S >>>`:

- `dim3` **Dg** determina dimensão e tamanho do *grid*:

  `Dg.x * Dg.y * Dg.z = numBlocks`

- `dim3` **Db** determina dimensão e tamanho de cada *block*:

  `Db.x * Db.y * Db.z = threadsPerBlock`

CUDA C: Hierarquia de *Threads*

Sobre a sintaxe de lançamento e configuração `<<< Dg, Db, Ns, S >>>`:

- `dim3 Dg` determina dimensão e tamanho do *grid*:

  `Dg.x * Dg.y * Dg.z = numBlocks`

- `dim3 Db` determina dimensão e tamanho de cada *block*:

  `Db.x * Db.y * Db.z = threadsPerBlock`

- `size_t Ns = 0`: *Bytes* extras na memória compartilhada
- `cudaStream_t S = 0`: CUDA *stream*

Fonte: `docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#programming-model` [Acessado em 29/07/16]

CUDA C: Hierarquia de Memória

Threads acessam múltiplos espaços de memória durante a execução de um *kernel*:

- Local

CUDA C: Hierarquia de Memória

Threads acessam múltiplos espaços de memória durante a execução de um *kernel*:

- Local
- Compartilhada com o *block*

CUDA C: Hierarquia de Memória

Threads acessam múltiplos espaços de memória durante a execução de um *kernel*:

- Local
- Compartilhada com o *block*
- Global

CUDA C: Hierarquia de Memória

Threads acessam múltiplos espaços de memória durante a execução de um *kernel*:

- Local
- Compartilhada com o *block*
- Global: persistente entre *kernels* da mesma aplicação

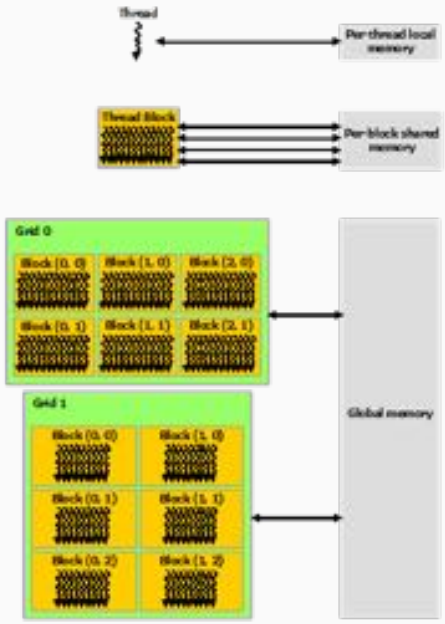
CUDA C: Hierarquia de Memória

Threads acessam múltiplos espaços de memória durante a execução de um *kernel*:

- Local
- Compartilhada com o *block*
- Global: persistente entre *kernels* da mesma aplicação
- *Read-only*

Fonte: `docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#programming-model` [Acessado em 29/07/16]

CUDA C: Hierarquia de Memória



Fonte: `docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#programming-model` [Acessado em 29/07/16]

CUDA C: Programação Heterogênea

Tarefas do host:

- Alocar memória e recursos do device
- Mover dados (gargalo!)

CUDA C: Programação Heterogênea

Tarefas do host:

- Alocar memória e recursos do device
- Mover dados (gargalo!)
- Lançar *kernel*

CUDA C: Programação Heterogênea

Tarefas do host:

- Alocar memória e recursos do device
- Mover dados (gargalo!)
- Lançar *kernel*
- Mover resultados (gargalo!)
- Liberar memória do device

CUDA C: Programação Heterogênea

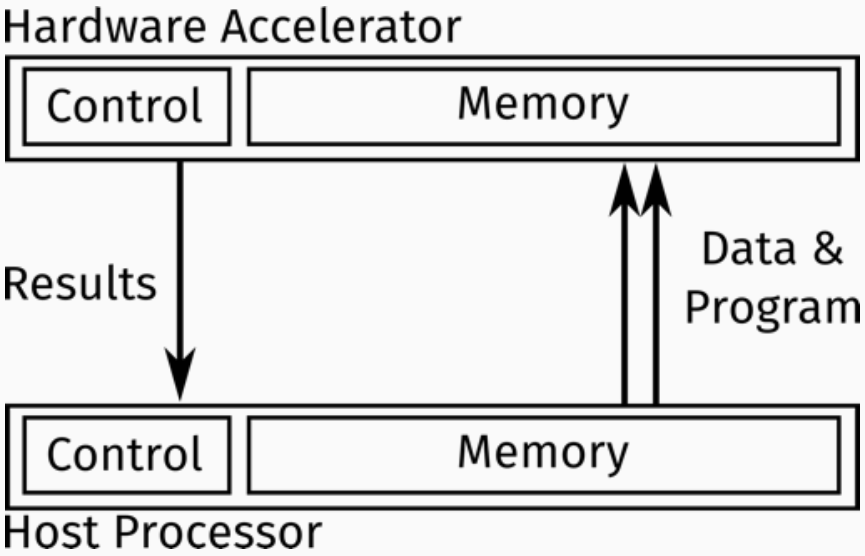
Tarefas do host:

- Alocar memória e recursos do device
- Mover dados (gargalo!)
- Lançar *kernel*
- Mover resultados (gargalo!)
- Liberar memória do device

Tarefas do device:

- Executar *kernel*

CUDA C: Programação Heterogênea



Exemplo: Adição de Vetores

```
__global__ void vectorAdd(const float *A, const float *B,
    float *C, int numElements) {
    int i = blockDim.x * blockIdx.x + threadIdx.x;

    if (i < numElements) {
        C[i] = A[i] + B[i];
    }
}
```

Fonte: `github.com/phrb/intro-cuda` [Acessado em 29/07/16]

Exemplo: Adição de Vetores

```
#include <cuda_runtime.h>
```

Exemplo: Adição de Vetores

```
#include <cuda_runtime.h>

float *h_A = (float *) malloc(size);
if (h_A == NULL) { ... };

float *d_A = NULL;
err = cudaMalloc((void **) &d_A, size);
err = cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
if (err != cudaSuccess) { ... };
```


Exemplo: Adição de Vetores

```
#include <cuda_runtime.h>

float *h_A = (float *) malloc(size);
if (h_A == NULL) { ... };

float *d_A = NULL;
err = cudaMalloc((void **) &d_A, size);
err = cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
if (err != cudaSuccess) { ... };

int threadsPerBlock = 256;
int blocksPerGrid = (numElements + threadsPerBlock - 1) /
    threadsPerBlock;
```

Exemplo: Adição de Vetores

```
#include <cuda_runtime.h>

float *h_A = (float *) malloc(size);
if (h_A == NULL) { ... };

float *d_A = NULL;
err = cudaMalloc((void **) &d_A, size);
err = cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
if (err != cudaSuccess) { ... };

int threadsPerBlock = 256;
int blocksPerGrid = (numElements + threadsPerBlock - 1) /
     threadsPerBlock;

vectorAdd<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B, d_C,
     numElements);

err = cudaGetLastError()
err = cudaDeviceSynchronize()
if (err != cudaSuccess) { ... };
```


Exemplo: Adição de Vetores

```
#include <cuda_runtime.h>

float *h_A = (float *) malloc(size);
if (h_A == NULL) { ... };

float *d_A = NULL;
err = cudaMalloc((void **) &d_A, size);
err = cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
if (err != cudaSuccess) { ... };

int threadsPerBlock = 256;
int blocksPerGrid = (numElements + threadsPerBlock - 1) /
     threadsPerBlock;

vectorAdd<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B, d_C,
     numElements);

err = cudaGetLastError()
err = cudaDeviceSynchronize()
if (err != cudaSuccess) { ... };
```

```
err = cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);
err = cudaFree(d_A);
if (err != cudaSuccess) { ... };
```


Fonte: github.com/phrb/intro-cuda [Acessado em 29/07/16]

Exemplo: Dica Prática

Sempre procure por erros!

```
float *d_A = NULL;
err = cudaMalloc((void **) &d_A, size);

err = cudaGetLastError()

err = cudaDeviceSynchronize()
```


Exemplo: Dica Prática

Sempre procure por erros!

```
float *d_A = NULL;
err = cudaMalloc((void **) &d_A, size);

err = cudaGetLastError()

err = cudaDeviceSynchronize()

if (err != cudaSuccess) {
    fprintf(stderr, "Failed to allocate device vector A
        (error code %s)!\n", cudaGetErrorString(err));
    exit(EXIT_FAILURE);
}
```

Fonte: `github.com/phrb/intro-cuda` [Acessado em 29/07/16]

Mão na massa!

Vamos executar alguns exemplos em CUDA C, disponíveis em `github.com/phrb/intro-cuda`:

- `src/cuda-samples/0_Simple/vectorAdd`

Mão na massa!

Vamos executar alguns exemplos em CUDA C, disponíveis em `github.com/phrb/intro-cuda`:

- `src/cuda-samples/0_Simple/vectorAdd`
- `src/cuda-samples/5_Simulations/fluidsGL`
- `src/cuda-samples/5_Simulations/oceanFFT`
- `src/cuda-samples/5_Simulations/smokeParticles`

Mão na massa!

Vamos executar alguns exemplos em CUDA C, disponíveis em `github.com/phrb/intro-cuda`:

- `src/cuda-samples/0_Simple/vectorAdd`
- `src/cuda-samples/5_Simulations/fluidsGL`
- `src/cuda-samples/5_Simulations/oceanFFT`
- `src/cuda-samples/5_Simulations/smokeParticles`
- `src/mandelbrot_numba`

Recursos

O *pdf* com as aulas e todo o código fonte estão no GitHub:

- `github.com/phrb/intro-cuda`

Outros recursos:

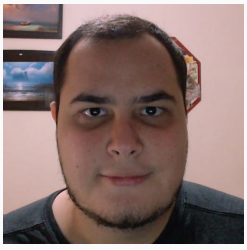
- CUDA C: `docs.nvidia.com/cuda/cuda-c-programming-guide`
- CUDA Toolkit: `developer.nvidia.com/cuda-toolkit`
- Guia de Boas Práticas:
  - `docs.nvidia.com/cuda/cuda-c-best-practices-guide`
- GPU Teaching Kit: `syllabus.gputeachingkit.com`
- iPython: `ipython.org/notebook.html`
- Anaconda: `continuum.io/downloads`

Introdução à Programação de GPUs com a Plataforma CUDA

Pedro Bruel
phrb@ime.usp.br
04 de Agosto de 2016

Instituto de Matemática e Estatística
Universidade de São Paulo

Sobre



Pedro Bruel

- phrb@ime.usp.br
- www.ime.usp.br/~phrb
- github.com/phrb

Alfredo Goldman

- gold@ime.usp.br
- www.ime.usp.br/~gold

Pesquisa

Meus interesses de pesquisa:

- *Autotuning*
- *Stochastic Local Search*
- *Model Based Search*
- GPUs, FPGAs, *cloud*
- Julia Language
- Colaboração! (phrb@ime.usp.br)

Parte I

1. Introdução
2. Computação Heterogênea
3. GPUs
4. Plataforma CUDA
5. CUDA C

Parte II

Slides

O *pdf* com as aulas e todo o código fonte estão no GitHub:

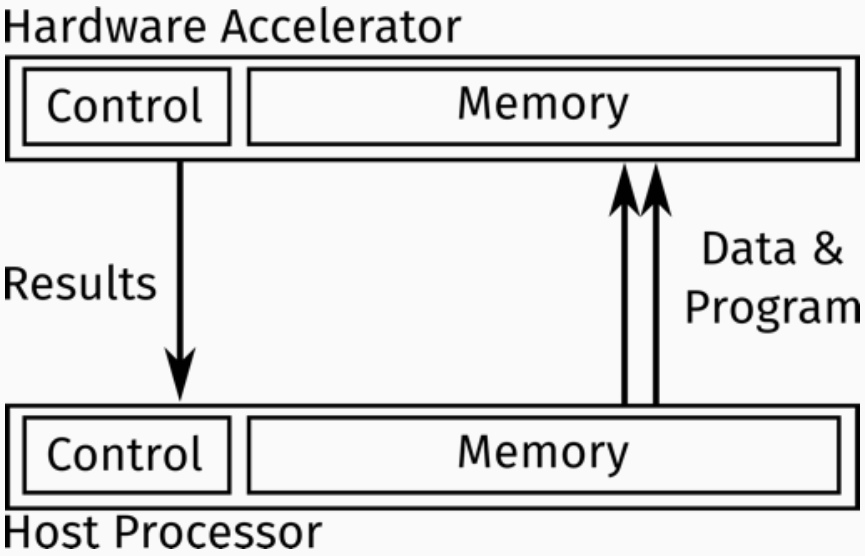
- `github.com/phrb/intro-cuda`

Aceleração por *Hardware*

Uso de dispositivos (devices) para acelerar computações aplicadas a grandes conjuntos de dados:

- Associação a um processador hospedeiro (host)
- Controle e memória próprios
- Diferem em especialização e configurabilidade
- GPUs, DSPs, FPGAs, ASICs

Aceleração por *Hardware*



Aceleração por *Hardware*

Porcentagem de sistemas com aceleradores na *Top500*:

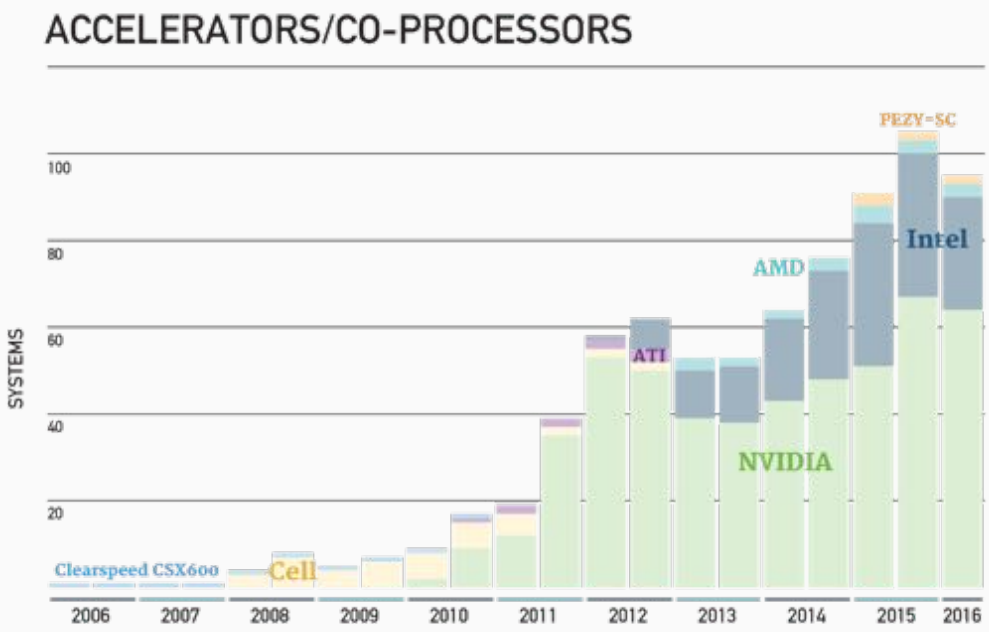


Imagem: `top500.org/lists/2016/06/download/TOP500_201606_Poster.pdf` [Acessado em 29/07/16]

Computação Heterogênea

Dados recursos computacionais heterogêneos, conjuntos de dados e computações, como distribuir computações e dados de forma a otimizar o uso dos recursos?

Imagem: `olcf.ornl.gov/titan` [Acessado em 29/07/16]

Graphics Processing Units

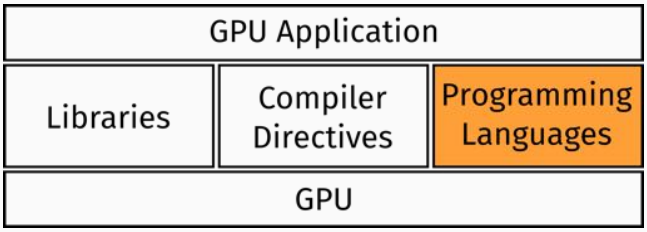
Originalmente especializadas em processamento gráfico, trabalham com muitos dados e têm alta vazão:

- Caches pequenos (*kilobytes*)
- Sem branch prediction
- Milhares de ALUs de maior latência
- Pipelines de execução
- 114 688 threads concorrentes na arquitetura Pascal

Plataforma CUDA

- Plataforma para computação paralela
- *Application Programming Interface* (API)
- CUDA *Toolkit*

Linguagens de Programação



- Melhor desempenho, mas mais difíceis de usar
- Flexíveis

CUDA C

Modelo de Programação:

- Kernels
- Hierarquia de Threads
- Hierarquia de Memória
- Programação Heterogênea

Exemplo: Adição de Vetores

```
#include <cuda_runtime.h>

float *h_A = (float *) malloc(size);
if (h_A == NULL) { ... };

float *d_A = NULL;
err = cudaMalloc((void **) &d_A, size);
err = cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
if (err != cudaSuccess) { ... };

int threadsPerBlock = 256;
int blocksPerGrid = (numElements + threadsPerBlock - 1) /
    threadsPerBlock;

vectorAdd<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B, d_C,
    numElements);

err = cudaGetLastError()
err = cudaDeviceSynchronize()
if (err != cudaSuccess) { ... };

err = cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);
err = cudaFree(d_A);
if (err != cudaSuccess) { ... };
```


Fonte: `github.com/phrb/intro-cuda` [Acessado em 29/07/16]

Mão na massa!

Vamos executar alguns exemplos em CUDA C, disponíveis em `github.com/phrb/intro-cuda`:

- `src/cuda-samples/0_Simple/vectorAdd`
- `src/cuda-samples/5_Simulations/fluidsGL`
- `src/cuda-samples/5_Simulations/oceanFFT`
- `src/cuda-samples/5_Simulations/smokeParticles`
- `src/mandelbrot_numba`

Dica Prática

Sempre procure por erros!

```
float *d_A = NULL;
err = cudaMalloc((void **) &d_A, size);

err = cudaGetLastError()

err = cudaDeviceSynchronize()

if (err != cudaSuccess) {
    fprintf(stderr, "Failed to allocate device vector A
        (error code %s)!\n", cudaGetErrorString(err));
    exit(EXIT_FAILURE);
}
```

Fonte: `github.com/phrb/intro-cuda` [Acessado em 29/07/16]

Nvidia CUDA Compiler (nvcc)

Nvidia CUDA Compiler (nvcc)

- Sistema proprietário

Nvidia CUDA Compiler (nvcc)

- Sistema proprietário
- Usa o compilador C++ do *host*

Nvidia CUDA Compiler (nvcc)

- Sistema proprietário
- Usa o compilador C++ do *host*
- Compila código CUDA para *Parallel Thread Execution* (PTX ISA)

Nvidia CUDA Compiler (nvcc)

- Sistema proprietário
- Usa o compilador C++ do *host*
- Compila código CUDA para *Parallel Thread Execution* (PTX ISA)
- Código do *host* + Código do *device* → fatbinary

nvcc: Modelo de programação CUDA

- SIMD, SIMT
- *Single Program Multiple Data* (SPMD)

nvcc: Modelo de programação CUDA

- SIMD, SIMT
- *Single Program Multiple Data* (SPMD)
- *Device*: threads paralelas e independentes
- *Host*: não interfere

nvcc: Trajetória de Compilação

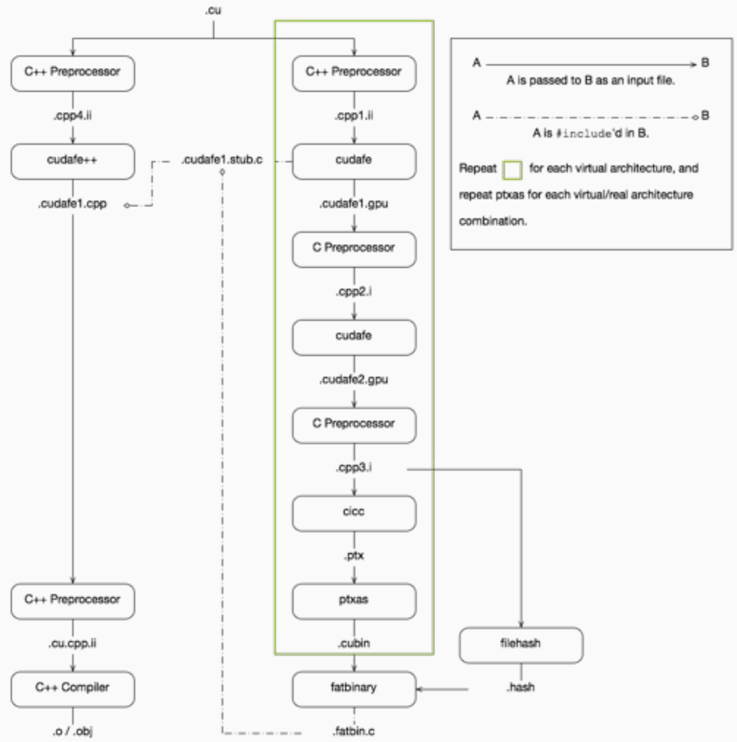
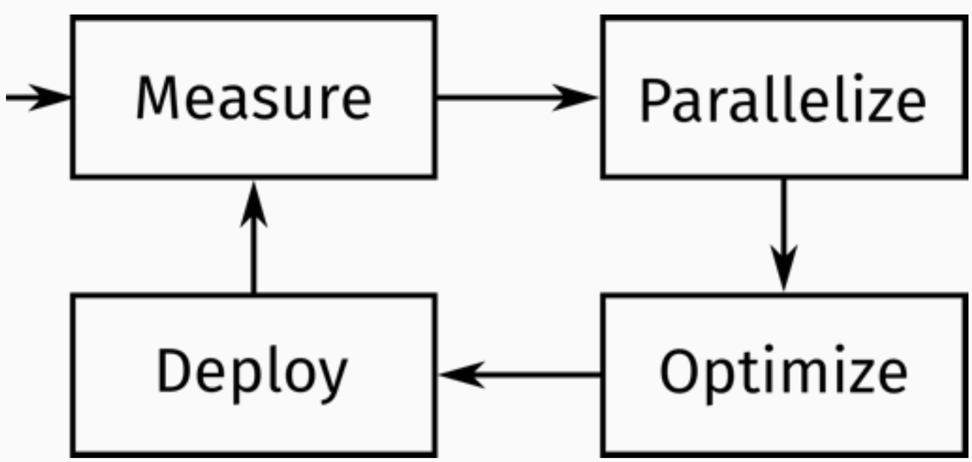


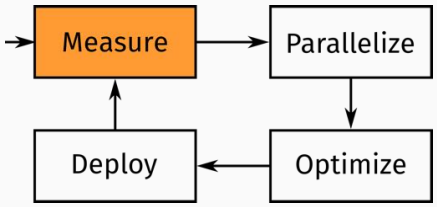
Imagem: `docs.nvidia.com/cuda/cuda-compiler-driver-nvcc` [Acessado em 29/07/16]

Measure, Parallelize, Optimize, Deploy

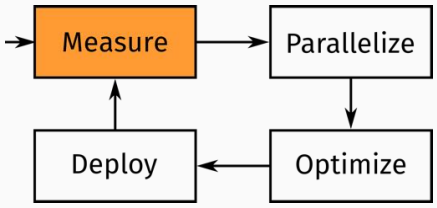


Fonte: `docs.nvidia.com/cuda/cuda-c-best-practices-guide` [Acessado em 29/07/16]

Measure



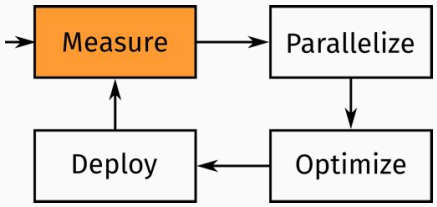
Measure



Dado um programa sequencial:

- Quais funções fazem o trabalho pesado (gargalos)?

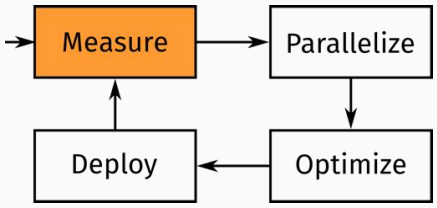
Measure



Dado um programa sequencial:

- Quais funções fazem o trabalho pesado (gargalos)?
- É possível paralelizá-las?

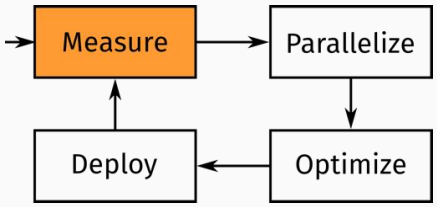
Measure



Dado um programa sequencial:

- Quais funções fazem o trabalho pesado (gargalos)?
- É possível paralelizá-las?
- Como o programa se comporta quando:
  - O trabalho é divido entre mais processos? (strong scaling)

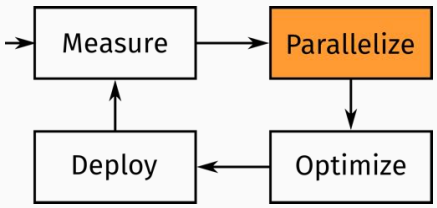
Measure



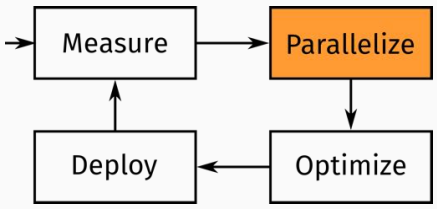
Dado um programa sequencial:

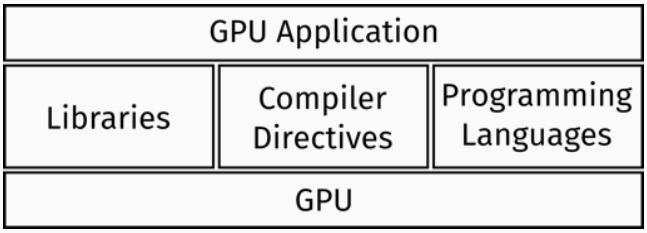
- Quais funções fazem o trabalho pesado (gargalos)?
- É possível paralelizá-las?
- Como o programa se comporta quando:
  - O trabalho é divido entre mais processos? (strong scaling)
  - Há mais trabalho a ser feito? (weak scaling)

Parallelize

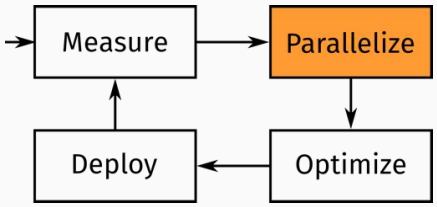


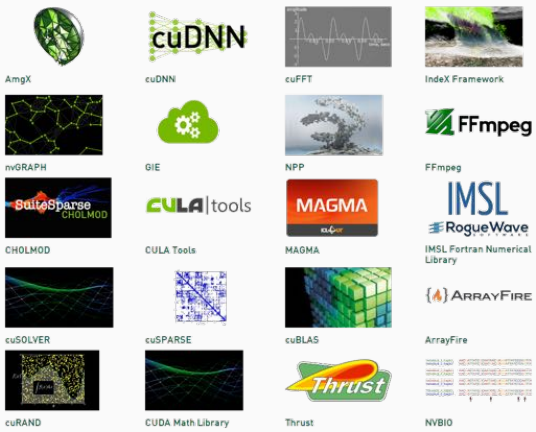
Parallelize



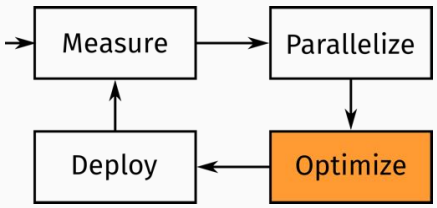


Parallelize

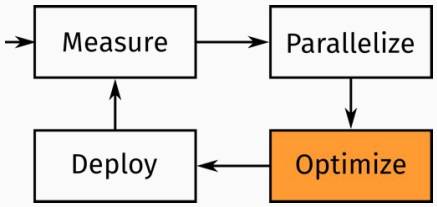




Optimize



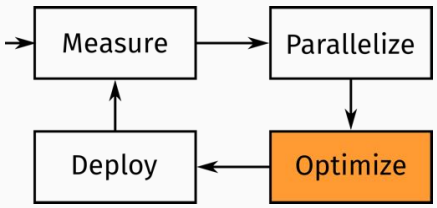
Optimize



Otimização:

- Processo cíclico

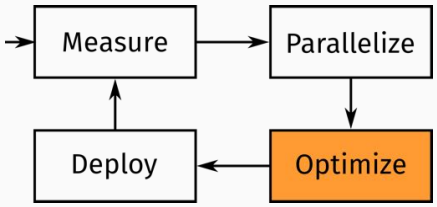
Optimize



Otimização:

- Processo cíclico e incremental

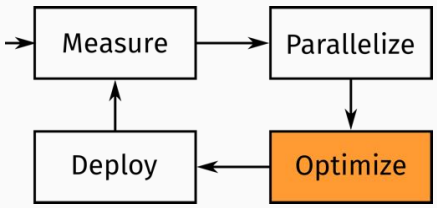
Optimize



Otimização:

- Processo cíclico e incremental
- Aplicável a diferentes níveis

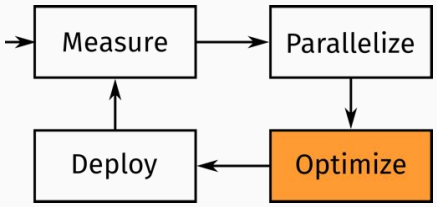
Optimize



Otimização:

- Processo cíclico e incremental
- Aplicável a diferentes níveis
- Ferramentas são muito importantes

Optimize

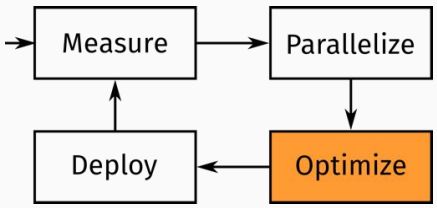


Otimização:

- Processo cíclico e incremental
- Aplicável a diferentes níveis
- Ferramentas são muito importantes

Eficiência com algoritmos,

Optimize

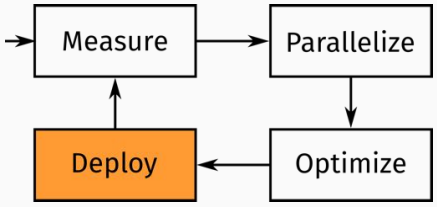


Otimização:

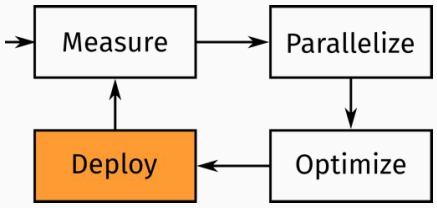
- Processo cíclico e incremental
- Aplicável a diferentes níveis
- Ferramentas são muito importantes

Eficiência com algoritmos, desempenho com estruturas de dados

Deploy

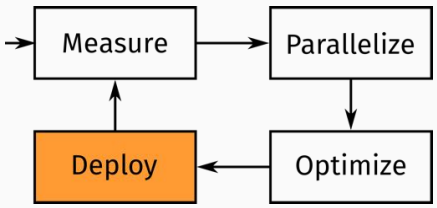


Deploy



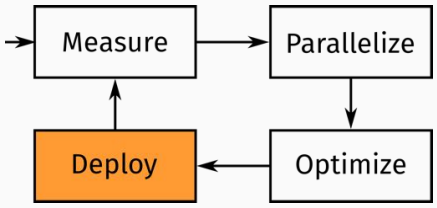
- Preparar a aplicação para a medição

Deploy



- Preparar a aplicação para a medição
- Otimizações e mudanças incrementais

Deploy



- Preparar a aplicação para a medição
- Otimizações e mudanças incrementais
- O quanto ainda é possível otimizar?

Análise de Aplicações CUDA

Computação paralela e distribuída é cada vez mais importante, mas código legado:

Análise de Aplicações CUDA

Computação paralela e distribuída é cada vez mais importante, mas código legado:

- Sequencial

Análise de Aplicações CUDA

Computação paralela e distribuída é cada vez mais importante, mas código legado:

- Sequencial
- Paralelismo de baixa granularidade

Análise de Aplicações CUDA

Computação paralela e distribuída é cada vez mais importante, mas código legado:

- Sequencial
- Paralelismo de baixa granularidade

Análise ou profiling:

Análise de Aplicações CUDA

Computação paralela e distribuída é cada vez mais importante, mas código legado:

- Sequencial
- Paralelismo de baixa granularidade

Análise ou profiling:

- Quais são os hotspots (*gargalos*, *bottlenecks*)?

Análise de Aplicações CUDA

Computação paralela e distribuída é cada vez mais importante, mas código legado:

- Sequencial
- Paralelismo de baixa granularidade

Análise ou profiling:

- Quais são os hotspots (*gargalos*, *bottlenecks*)?
- Podem ser paralelizados?

Análise de Aplicações CUDA

Computação paralela e distribuída é cada vez mais importante, mas código legado:

- Sequencial
- Paralelismo de baixa granularidade

Análise ou profiling:

- Quais são os hotspots (*gargalos*, *bottlenecks*)?
- Podem ser paralelizados?
- Quais workloads são relevantes?

Host e Device

Diferenças entre *host* e *device*:

- Modelo de *threading*

Host e Device

Diferenças entre *host* e *device*:

- Modelo de *threading*
- Memória

Host e Device

Diferenças entre *host* e *device*:

- Modelo de *threading*
- Memória

O que executar em GPUs?

Host e Device

Diferenças entre *host* e *device*:

- Modelo de *threading*
- Memória

O que executar em GPUs?

- Grande quantidade de dados

Host e Device

Diferenças entre *host* e *device*:

- Modelo de *threading*
- Memória

O que executar em GPUs?

- Grande quantidade de dados
- Operações independentes

Host e Device

Diferenças entre *host* e *device*:

- Modelo de *threading*
- Memória

O que executar em GPUs?

- Grande quantidade de dados
- Operações independentes

Qual o impacto das transferências de dados?

Profiling

Monitoramento de código em tempo de execução, obtendo informação para otimizar o código

Profiling

Monitoramento de código em tempo de execução, obtendo informação para otimizar o código

Instrumentação do Código:

- Captura de eventos

Profiling

Monitoramento de código em tempo de execução, obtendo informação para otimizar o código

Instrumentação do Código:

- Captura de eventos
- Geração de dados

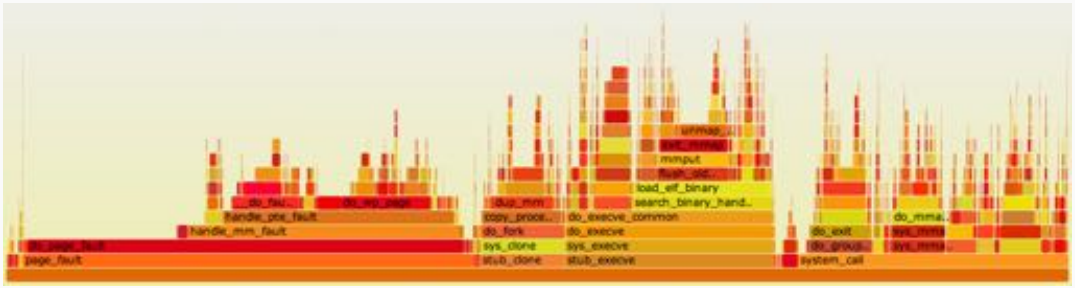
Profiling

Monitoramento de código em tempo de execução, obtendo informação para otimizar o código

Instrumentação do Código:

- Captura de eventos
- Geração de dados
- Ferramentas

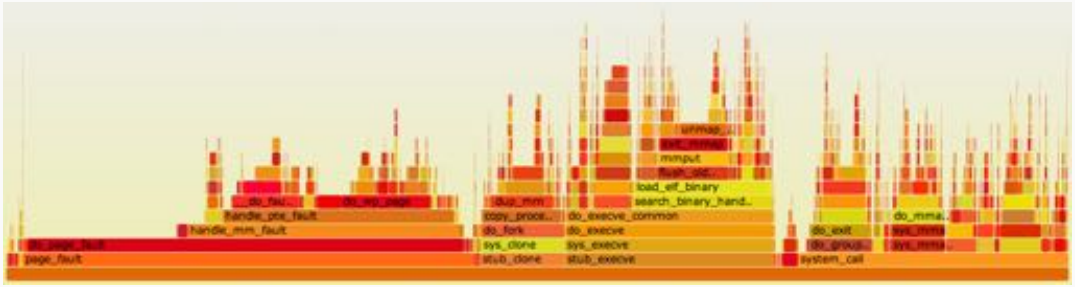
Profiling: Flamegraphs



Facilitam a análise de:

- *Bottlenecks*, *hotspots*, ou *gargalos* de desempenho

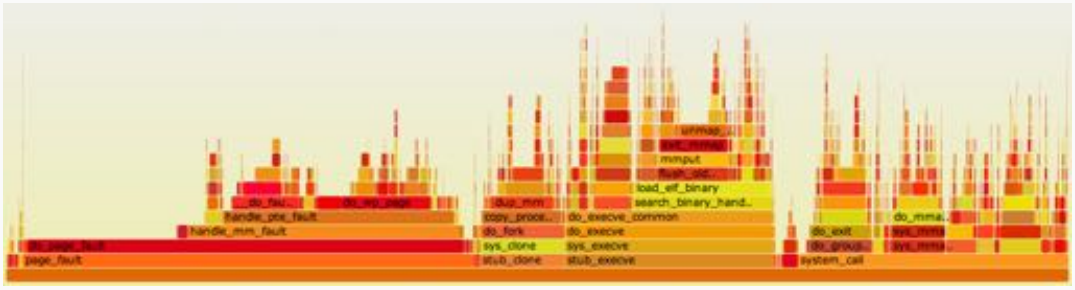
Profiling: Flamegraphs



Facilitam a análise de:

- *Bottlenecks*, *hotspots*, ou *gargalos* de desempenho
- Frequência e duração de chamadas de função: *On-CPU*

Profiling: Flamegraphs



Facilitam a análise de:

- *Bottlenecks*, *hotspots*, ou *gargalos* de desempenho
- Frequência e duração de chamadas de função: *On-CPU*
- Tempo ocioso (*espera*): *Off-CPU*

Profiling: nvprof

Profiler de linha de comando, permite a coleta de eventos relacionados a CUDA:

Profiling: nvprof

Profiler de linha de comando, permite a coleta de eventos relacionados a CUDA:

- CPU e GPU

Profiling: nvprof

Profiler de linha de comando, permite a coleta de eventos relacionados a CUDA:

- CPU e GPU
- Execução do *kernel*

85/96

Profiling: nvprof

Profiler de linha de comando, permite a coleta de eventos relacionados a CUDA:

- CPU e GPU
- Execução do *kernel*
- Uso e transferência de memória

Profiling: nvprof

Profiler de linha de comando, permite a coleta de eventos relacionados a CUDA:

- CPU e GPU
- Execução do *kernel*
- Uso e transferência de memória
- CUDA API

Profiling: nvprof

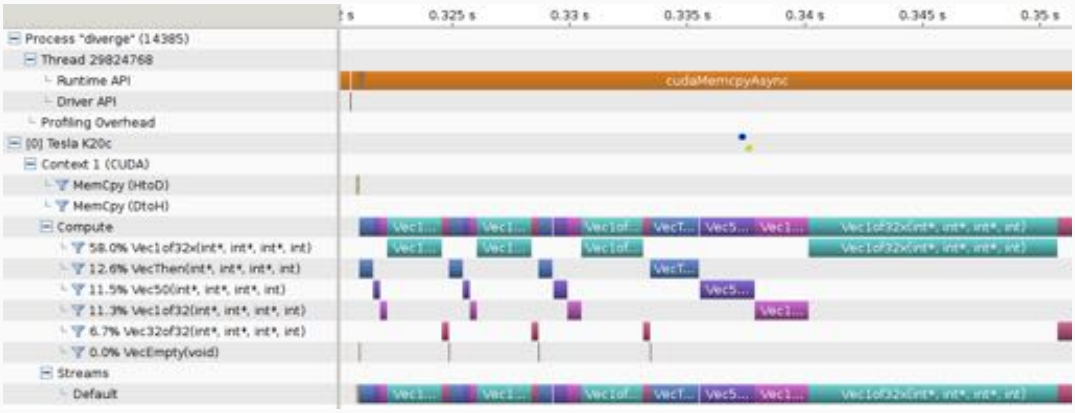
Profiler de linha de comando, permite a coleta de eventos relacionados a CUDA:

- CPU e GPU
- Execução do *kernel*
- Uso e transferência de memória
- CUDA API

Gera dados para posterior visualização

Fonte: `docs.nvidia.com/cuda/profiler-users-guide` [Acessado em 29/07/16]

Profiling: nvvp



Fonte: `docs.nvidia.com/cuda/profiler-users-guide` [Acessado em 29/07/16]

Profiling: Ferramentas

Fonte: `developer.nvidia.com/performance-analysis-tools` [Acessado em 29/07/16]

Debugging

Monitoramento de código em tempo de execução, obtendo informação para consertar erros (bugs)

Debugging

Monitoramento de código em tempo de execução, obtendo informação para consertar erros (bugs)

Instrumentação do Código:

- Execução passo-a-passo

Debugging

Monitoramento de código em tempo de execução, obtendo informação para consertar erros (bugs)

Instrumentação do Código:

- Execução passo-a-passo
- Breakpoints

Debugging

Monitoramento de código em tempo de execução, obtendo informação para consertar erros (bugs)

Instrumentação do Código:

- Execução passo-a-passo
- Breakpoints
- Diferentes threads

Debugging

Monitoramento de código em tempo de execução, obtendo informação para consertar erros (bugs)

Instrumentação do Código:

- Execução passo-a-passo
- Breakpoints
- Diferentes threads
- CUDA `gdb` e `memcheck`

Debugging

Monitoramento de código em tempo de execução, obtendo informação para consertar erros (bugs)

Instrumentação do Código:

- Execução passo-a-passo
- Breakpoints
- Diferentes threads
- CUDA `gdb` e `memcheck`

Facilitam a solução de:

- Vazamentos de memória

Debugging

Monitoramento de código em tempo de execução, obtendo informação para consertar erros (bugs)

Instrumentação do Código:

- Execução passo-a-passo
- Breakpoints
- Diferentes threads
- CUDA `gdb` e `memcheck`

Facilitam a solução de:

- Vazamentos de memória
- Problemas com fluxo de controle

Debugging

Monitoramento de código em tempo de execução, obtendo informação para consertar erros (bugs)

Instrumentação do Código:

- Execução passo-a-passo
- Breakpoints
- Diferentes threads
- CUDA `gdb` e `memcheck`

Facilitam a solução de:

- Vazamentos de memória
- Problemas com fluxo de controle
- ...

Otimização de Aplicações CUDA

Diferentes níveis de abstração e prioridade:

Otimização de Aplicações CUDA

Diferentes níveis de abstração e prioridade:

- Memória

Otimização de Aplicações CUDA

Diferentes níveis de abstração e prioridade:

- Memória
- Fluxo de controle

Otimização de Aplicações CUDA

Diferentes níveis de abstração e prioridade:

- Memória
- Fluxo de controle
- Configuração de execução

Otimização de Aplicações CUDA

Diferentes níveis de abstração e prioridade:

- Memória
- Fluxo de controle
- Configuração de execução
- Instruções

Otimizações de Memória

São as mais importantes para atingir alto desempenho

Otimizações de Memória

São as mais importantes para atingir alto desempenho

Objetivos:

- Maximizar largura de banda (*bandwidth*)

Otimizações de Memória

São as mais importantes para atingir alto desempenho

Objetivos:

- Maximizar largura de banda (*bandwidth*)
- Minimizar transferências entre *device* e *host*

Otimizações de Memória

São as mais importantes para atingir alto desempenho

Objetivos:

- Maximizar largura de banda (*bandwidth*)
- Minimizar transferências entre *device* e *host*

Boas práticas:

Otimizações de Memória

São as mais importantes para atingir alto desempenho

Objetivos:

- Maximizar largura de banda (*bandwidth*)
- Minimizar transferências entre *device* e *host*

Boas práticas:

- No *device*:

Otimizações de Memória

São as mais importantes para atingir alto desempenho

Objetivos:

- Maximizar largura de banda (*bandwidth*)
- Minimizar transferências entre *device* e *host*

Boas práticas:

- No *device*:
  - Priorizar execução de computações

Otimizações de Memória

São as mais importantes para atingir alto desempenho

Objetivos:

- Maximizar largura de banda (*bandwidth*)
- Minimizar transferências entre *device* e *host*

Boas práticas:

- No *device*:
  - Priorizar execução de computações
  - Criar e destruir estruturas de dados intermediárias

Otimizações de Memória

São as mais importantes para atingir alto desempenho

Objetivos:

- Maximizar largura de banda (*bandwidth*)
- Minimizar transferências entre *device* e *host*

Boas práticas:

- No *device*:
  - Priorizar execução de computações
  - Criar e destruir estruturas de dados intermediárias
  - Accesos agrupados (*coalescentes*) à memória

Otimizações de Memória

São as mais importantes para atingir alto desempenho

Objetivos:

- Maximizar largura de banda (*bandwidth*)
- Minimizar transferências entre *device* e *host*

Boas práticas:

- No *device*:
  - Priorizar execução de computações
  - Criar e destruir estruturas de dados intermediárias
  - Accesos agrupados (*coalescentes*) à memória
- No *host*:

Otimizações de Memória

São as mais importantes para atingir alto desempenho

Objetivos:

- Maximizar largura de banda (*bandwidth*)
- Minimizar transferências entre *device* e *host*

Boas práticas:

- No *device*:
  - Priorizar execução de computações
  - Criar e destruir estruturas de dados intermediárias
  - Accesos agrupados (*coalescentes*) à memória
- No *host*:
  - Delegar execução de computações

Otimizações de Memória

São as mais importantes para atingir alto desempenho

Objetivos:

- Maximizar largura de banda (*bandwidth*)
- Minimizar transferências entre *device* e *host*

Boas práticas:

- No *device*:
  - Priorizar execução de computações
  - Criar e destruir estruturas de dados intermediárias
  - Accesos agrupados (*coalescentes*) à memória
- No *host*:
  - Delegar execução de computações
  - Minimizar o número de transferências de dados

Otimizações de Fluxo de Controle

Alta prioridade para atingir alto desempenho

Otimizações de Fluxo de Controle

Alta prioridade para atingir alto desempenho

Objetivos:

- Evitar divergência de fluxo de controle dentro de *warps*

Otimizações de Fluxo de Controle

Alta prioridade para atingir alto desempenho

Objetivos:

- Evitar divergência de fluxo de controle dentro de *warps*
- `if`, `switch`, `do`, `for`, `while`

Otimizações de Fluxo de Controle

Alta prioridade para atingir alto desempenho

Objetivos:

- Evitar divergência de fluxo de controle dentro de *warps*
- `if`, `switch`, `do`, `for`, `while`
- Fazer bom uso do `threadIdx`

Otimizações de Configuração de Execução

Como aproveitar totalmente o potencial do device?

Otimizações de Configuração de Execução

Como aproveitar totalmente o potencial do device?

- Maximizar a ocupância

Otimizações de Configuração de Execução

Como aproveitar totalmente o potencial do device?

- Maximizar a ocupância
- Escolher bem os blocks e threads

Otimizações de Configuração de Execução

Como aproveitar totalmente o potencial do device?

- Maximizar a ocupância
- Escolher bem os blocks e threads
- Execução independente de kernels

Otimizações de Instruções

Baixa prioridade para atingir alto desempenho:

Otimizações de Instruções

Baixa prioridade para atingir alto desempenho:

- Baixo nível

Otimizações de Instruções

Baixa prioridade para atingir alto desempenho:

- Baixo nível
- Aplicadas a hotspots

Otimizações de Instruções

Baixa prioridade para atingir alto desempenho:

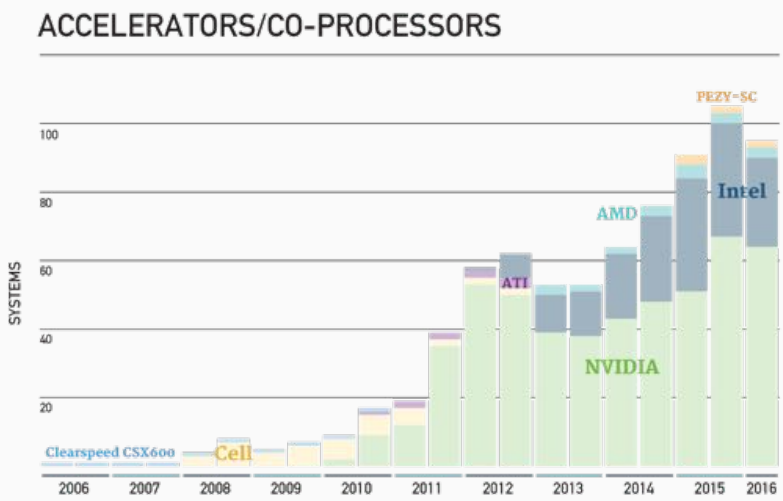
- Baixo nível
- Aplicadas a hotspots
- Feitas por experts
- Após exaurir outras possibilidades de otimização

Otimizações de Instruções

Baixa prioridade para atingir alto desempenho:

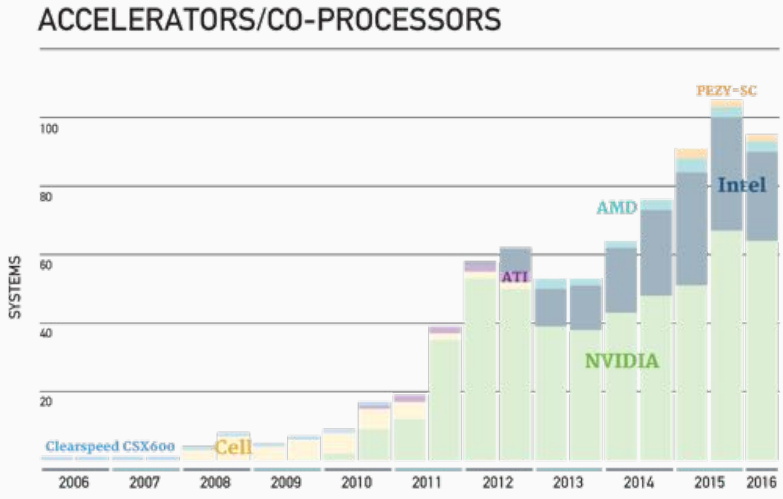
- Baixo nível
- Aplicadas a hotspots
- Feitas por experts
- Após exaurir outras possibilidades de otimização
- “Escovar *bits*” ☺

Conclusão



Atingir alto desempenho através do uso de aceleradores

Conclusão



Atingir alto desempenho através do uso de aceleradores (GPUs)

Conclusão

Conclusão

Custo de implementação, análise e otimização

Conclusão

Custo de implementação, análise e otimização

Ferramentas da plataforma CUDA

Recursos

O *pdf* com as aulas e todo o código fonte estão no GitHub:

- `github.com/phrb/intro-cuda`

Outros recursos:

- CUDA C: `docs.nvidia.com/cuda/cuda-c-programming-guide`
- CUDA Toolkit: `developer.nvidia.com/cuda-toolkit`
- Guia de Boas Práticas:
  - `docs.nvidia.com/cuda/cuda-c-best-practices-guide`
- GPU Teaching Kit: `syllabus.gputeachingkit.com`
- iPython: `ipython.org/notebook.html`
- Anaconda: `continuum.io/downloads`

Introdução à Programação de GPUs com a Plataforma CUDA

Pedro Bruel
phrb@ime.usp.br
04 de Agosto de 2016

Instituto de Matemática e Estatística
Universidade de São Paulo

Vectorization

Silvio Stanzani , Raphael Cóbe , Rogério Iope
Núcleo de Computação Científica – UNESP

ERAD-SP 2016 - Mackenzie
05 August 2016

Agenda

- Hybrid Parallel Architectures**;**
- Memory System and Vector Processing Units;
- Intel Architectures;
- Profiling;
- Optimizing Memory Access;
- Auto Vectorization;
- Guided Vectorization;
- Examples.

Agenda

- **Hybrid Parallel Architectures;**
- Memory System and Vector Processing Units;
- Intel Architectures;
- Profiling;
- Optimizing Memory Access;
- Auto Vectorization;
- Guided Vectorization;
- Examples.

Hybrid Parallel Architectures

- Heterogeneous computational systems:
  - Multicore processors;
  - Multi-level memory sub-system;
  - Input and Output sub-system;
- Multi-level parallelism:
  - Processing core;
  - Chip multiprocessor;
  - Computing node;
  - Computing cluster;
- Hybrid Parallel architectures
  - Coprocessors and accelerators;

Hybrid Parallel Architectures

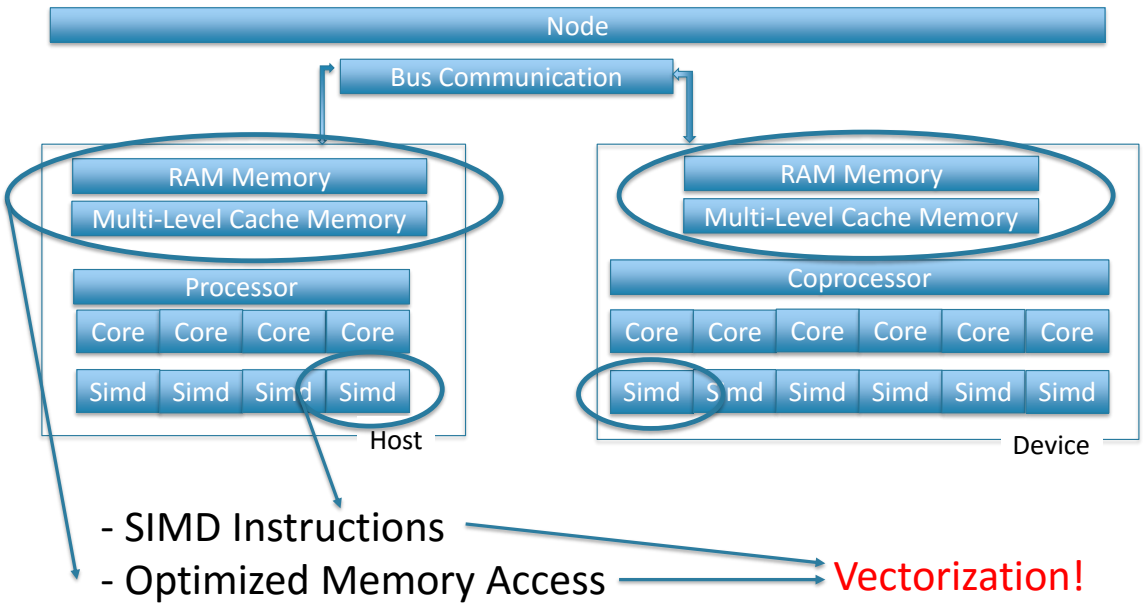
- Heterogeneous computational systems:
  - Scalar and Vector Instructions

Vector Instructions (SIMD)								Scalar Instructions
A7	A6	A5	A4	A3	A2	A1	A0	A
			+					+
B7	B6	B5	B4	B3	B2	B1	B0	B
			=					=
A7+B7	A6+B6	A5+B5	A4+B4	A3+B3	A2+B2	A1+B1	A0+B0	A+B

  - Multi-level memory
    - RAM Memory;
    - Multi-level Cache.

Processor 1			Processor 2		
Core 1	Core 2	Core N	Core 1	Core 2	Core N
L1	L1	L1	L1	L1	L1
L2	L2	L2	L2	L2	L2
L3			L3		
RAM					

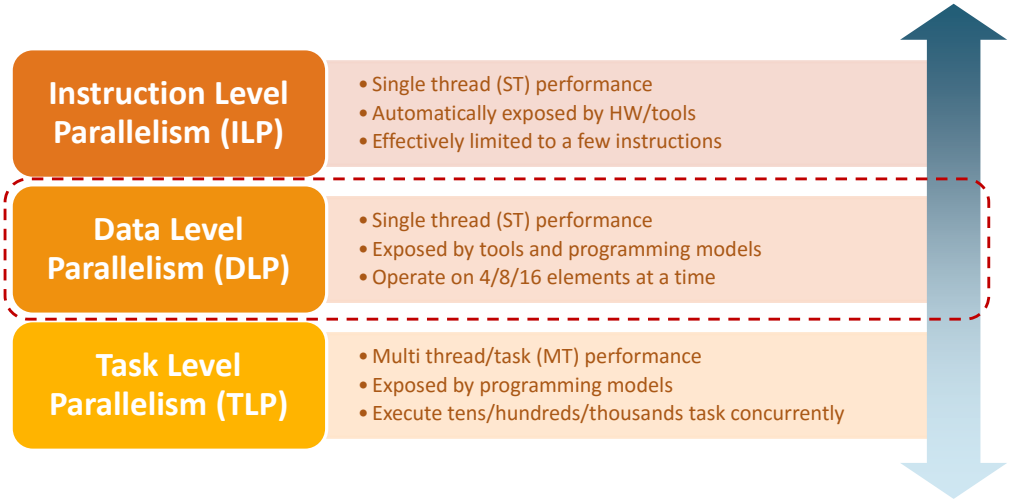
Hybrid Parallel Architectures



- SIMD Instructions
- Optimized Memory Access → Vectorization!

Don’t use a single thread or vector lane

Exploiting the parallel universe



Programmers' responsibility to expose DLP/TLP

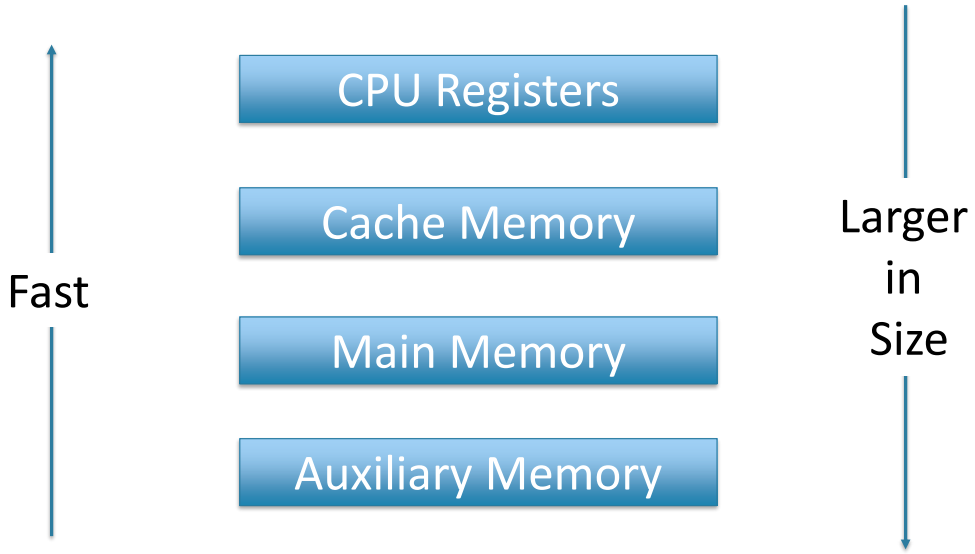
Agenda

- Hybrid Parallel Architectures;
- **Memory System and Vector Processing Units;**
- Intel Architectures;
- Profiling;
- Optimizing Memory Access;
- Auto Vectorization;
- Guided Vectorization;
- Examples.

Memory System

- CPU Register: internal Processor Memory. Stores data or instruction to be executed;
- Cache: stores segments of programs currently being executed in the CPU and temporary data frequently needed in the present calculations;
- Main memory: only program and data currently needed by the processor resides in main memory;
- Auxiliary memory: devices that provides backup storage.

Memory Hierarchy



Cache Memory

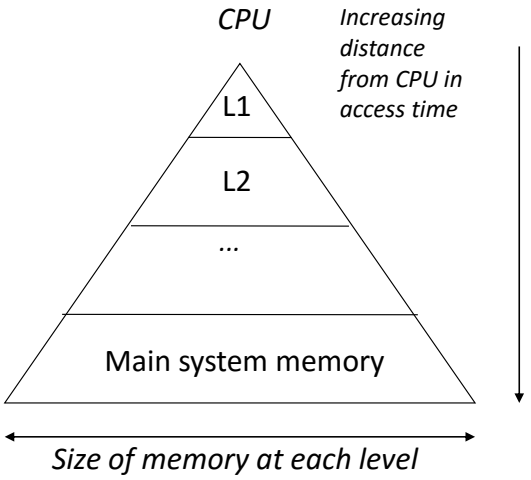
- Cache Memory is employed in computer systems to compensate for the difference in speed between main memory access time and processor logic.
- Operating System controls the load of Data to Cache; such load can be guided by the developer

Cache Memory

- The Performance of cache memory is frequently measured in terms of hit ratio.
  - When the CPU refers to memory and finds the word in cache, it is said to produce a **hit**.
  - If the word is not found in cache, it is in main memory and it counts as a **miss**

Locality

- Temporal locality: if an item was referenced, it will be referenced again soon (e.g. cyclical execution in loops);
- Spatial locality: if an item was referenced, items close to it will be referenced too (the very nature of every program – serial stream of instructions)

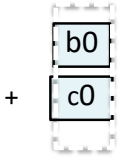


Vectorization

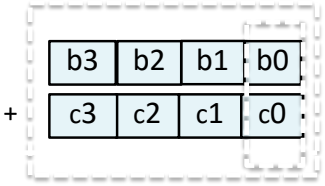
- Scalar Code computes this one-element at a time.
- Vector (or SIMD) Code computes more than one element at a time. SIMD stands for **S**ingle **I**nstruction **M**ultiple **D**ata.

```
float *A, *B, *C;
for(i=0;i<n;i++){
  A[i] = B[i] + C[i];
}
```

- Scalar



- SIMD



Vectorization

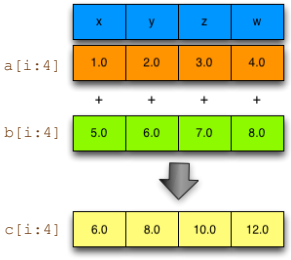
- Vectorization
  - Loading data into cache accordingly;
  - Store elements on SIMD registers or vectors;
  - Apply the same operation to a set of Data at the same time;
  - Iterations need to be independent;
  - Usually on inner loops.

Scalar loop

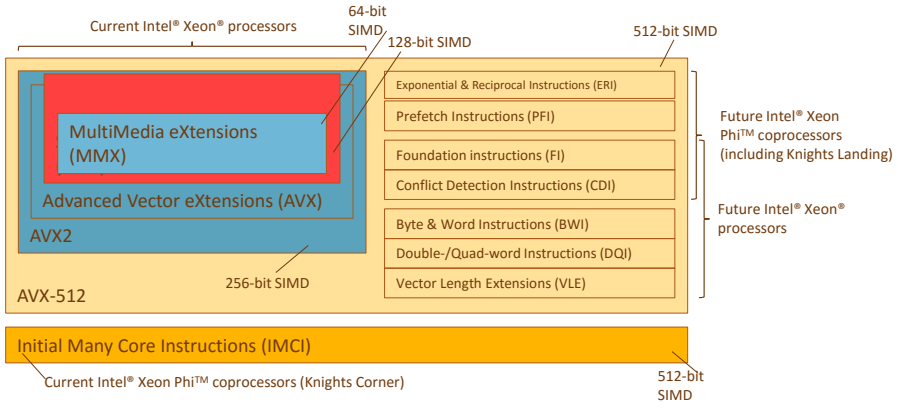
```
for (int i = 0; i < N; i++)
    c[i] = a[i] + b[i];
```

SIMD loop (4 elements)

```
for (int i = 0; i < N; i += 4)
    c[i:4] = a[i:4] + b[i:4];
```



Past, present, and future of Intel SIMD types



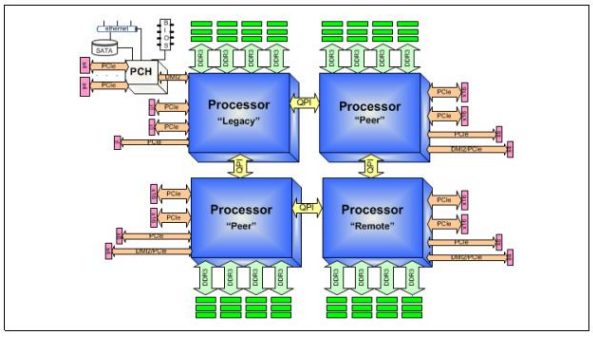
Intel® AVX2/IMCI/AVX-512 differences

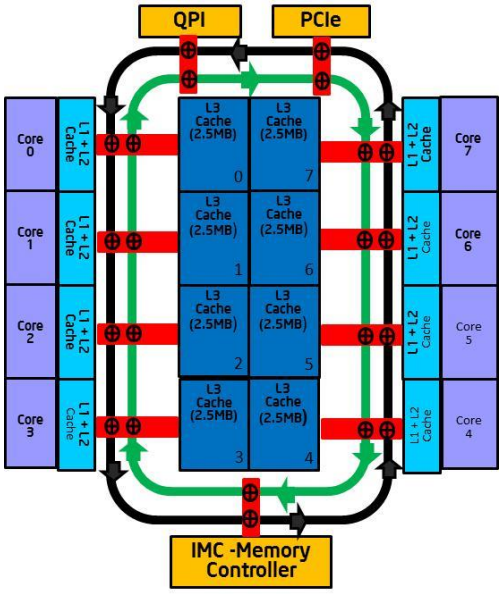
	Intel® Initial Many Core Instructions **IMCI**	Intel® Advanced Vector Extensions 2 **AVX2**	Intel® Advanced Vector Extensions 512 **AVX-512**
Introduction	2012	2013	2015
Products	Knights Corner	Haswell, Broadwell	Knights Landing, future Intel® Xeon® and Xeon® Phi™ products
Register file	SP/DP/int32/int64 data types 32 x 512-bit SIMD registers 8 x 16-bit mask registers	SP/DP/int32/int64 data types 16 x 256-bit SIMD registers No mask registers (instr. blending)	SP/DP/int32/int64 data types 32 x 512-bit SIMD registers 8 x (up to) 64-bit mask
ISA features	Not compatible with AVX*/SSE* No unaligned data support Embedded broadcast/cvt/swizzle MVEX encoding	Fully compatible with AVX/SSE* Unaligned data support (penalty) VEX encoding	Fully compatible with AVX*/SSE* Unaligned data support (penalty) Embedded broadcast/rounding EVEX encoding
Instruction features	Fused multiply-and-add (FMA) Partial gather/scatter Transcendental support	Fused multiply-and-add (FMA) Full gather	Fused multiply-and-add (FMA) Full gather/scatter Transcendental support (ERI only) Conflict detection instructions PFI/BWI/DQI/VLE (if applies)

Agenda

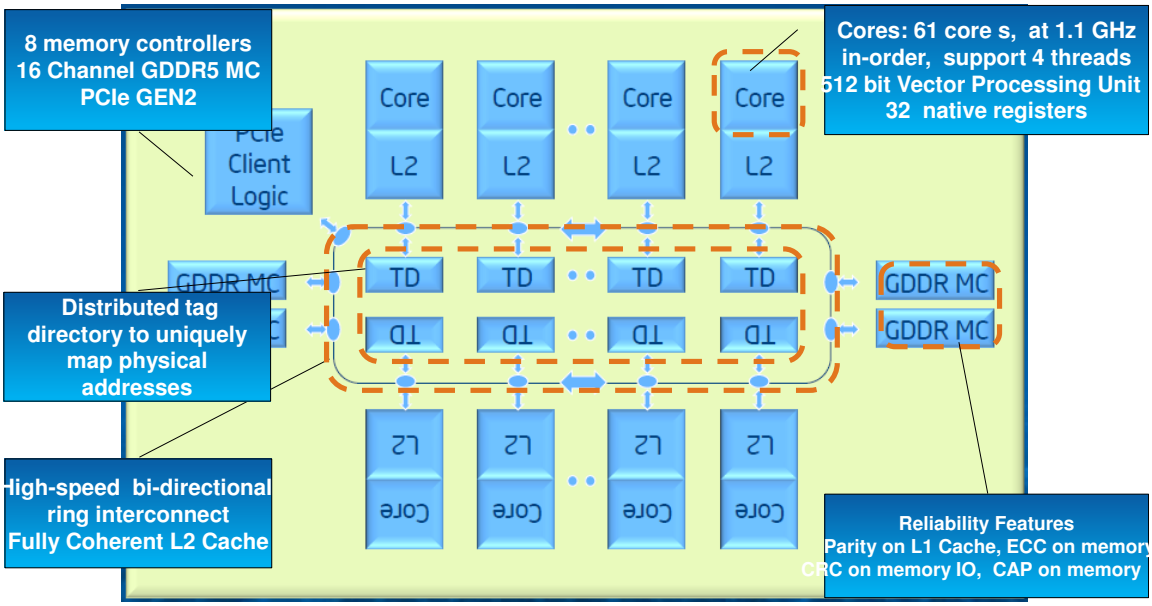
- Hybrid Parallel Architectures;
- Memory System and Vector Processing Units;
- **Intel Architectures;**
- Profiling;
- Optimizing Memory Access;
- Auto Vectorization;
- Guided Vectorization;
- Examples.

Intel Xeon Architecture Overview

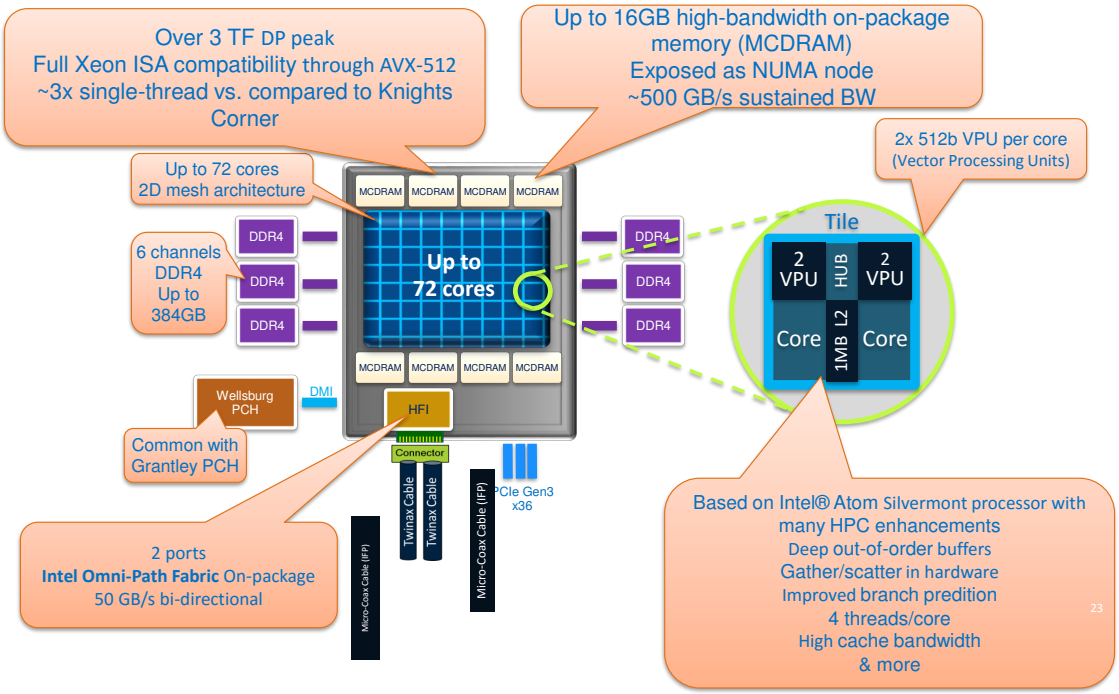




Intel® Xeon Phi™ Architecture Overview



Knights Landing (KNL)



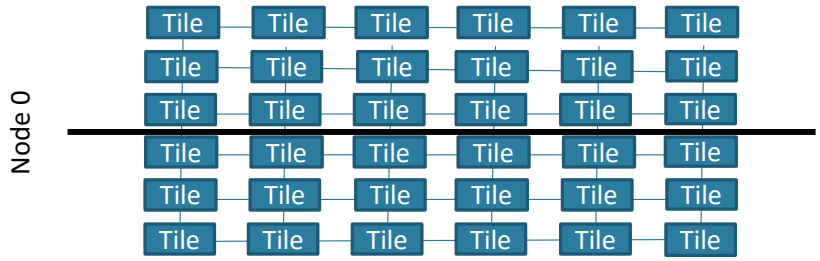
Knights Landing (KNL)

- KNL Knights Landing has 72 cores;
- Each one has an L1 cache;
- Pairs of cores are organized into tiles with a slice of the L2 cache symmetrically shared between the two cores;
- All caches are kept coherent;
- Two VPUs (Vector Processing Units) per core;

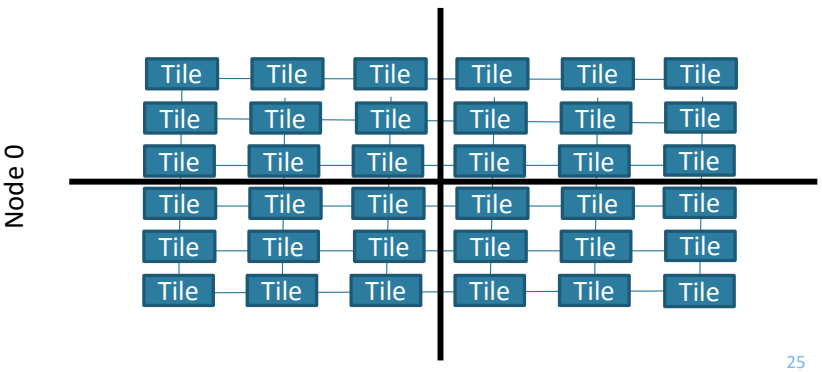
Cluster modes

One single space address

Hemisphere:
the tiles are divided into two parts called hemisphere



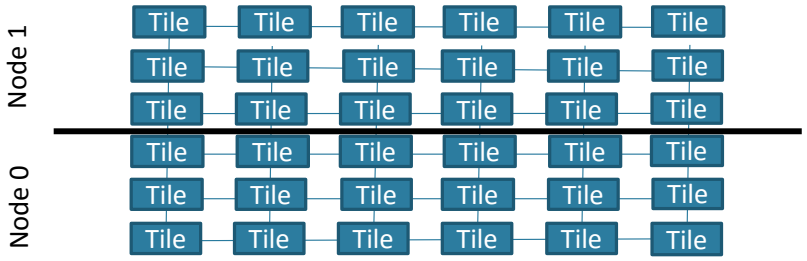
Quadrant:
tiles are divided into two parts called hemisphere or into four parts called qudrants



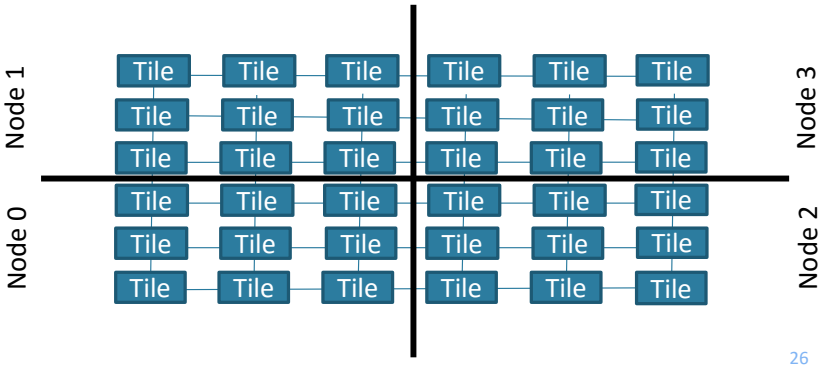
Cluster modes

Cache data are isolated in each sub numa domain

SNC-2:
the tiles are divided into two Numa Nodes

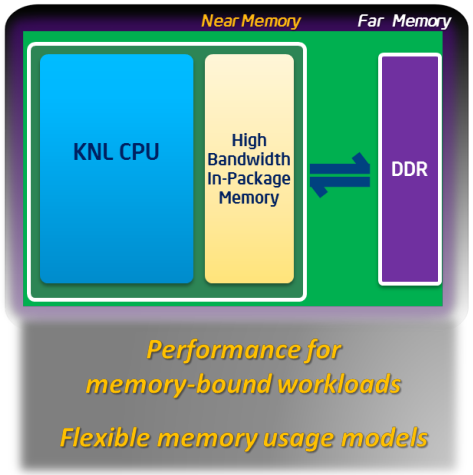


SNC-4:
the tiles are divided into two Numa Nodes



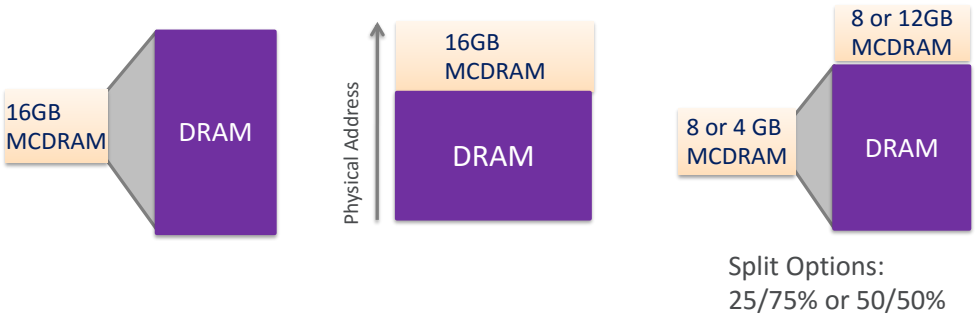
Knights Landing Integrated On-Package Memory

Multi-Channel DRAM (High-bandwidth memory)



Integrated On-Package Memory Usage Models

Model configurable at boot tiem and software exposed through NUMA



Split Options:
25/75% or 50/50%

Cache Model	Flat Model	Hybrid Model
Hardware automatically manages the MCDRAM as a "L3 cache" between CPU and ext DDR memory	Manually manage how the app uses the integrated on-package memory and external DDR for peak perf	Harness the benefits of both Cache and Flat models by segmenting the integrated on-package memory
▪ App and/or data set is very large and will not fit into MCDRAM ▪ Unknown or unstructured memory access behavior	▪ App or portion of an app or data set that can be, or is needed to be "locked" into MCDRAM so it doesn't get flushed out	▪ Need to "lock" in a relatively small portion of an app or data set via the Flat model ▪ Remaining MCDRAM can then be configured as Cache

Agenda

- Hybrid Parallel Architectures;
- Memory System and Vector Processing Units;
- Intel Architectures;
- **Profiling;**
- Optimizing Memory Access;
- Auto Vectorization;
- Guided Vectorization;
- Examples.

Intel Advisor

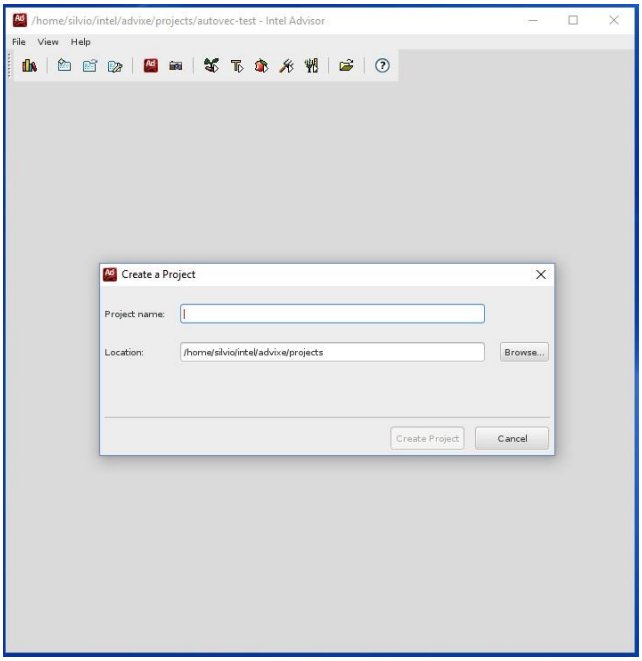
- Evaluate multi-threading parallelization

- Intel® Advisor XE
  - ❑ Performance modeling using several frameworks for multi-threading in processors and co-processors:
    - OpenMP, Intel® Cilk ™ Plus, Intel® Threading Bulding Blocks
    - C, C++, Fortran (OpenMP only) and C# (Microsoft TPL)
  - ❑ Identify parallel opportunities
    - Detailed information about vectorization;
    - Check loop dependencies;
  - ❑ Scalability prediction: amount of threads/performance gains
  - ❑ Correctness (deadlocks, race conditions)

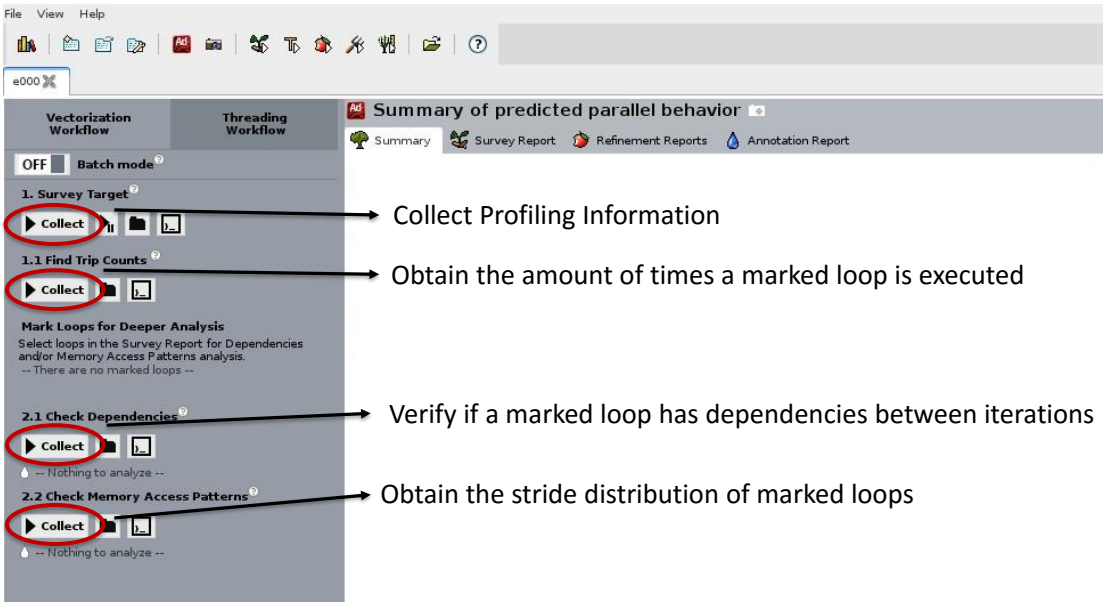
Intel Advisor

- Survey Target;
  - Vectorization of loops: detailed information about vectorization;
  - Total Time: elapsed time on each loop considering the time involved in internal loops;
  - Self Time: elapsed time on each loop not considering the time involved in internal loops;
- Find Trip Counts;
  - Analysis to identify how many time particular loops run;
- Check Dependencies;
  - Analysis it there are many loop-carried dependencies;
- Check Memory Access Patterns.
  - Analysis to identify how your code is iterating with memory.

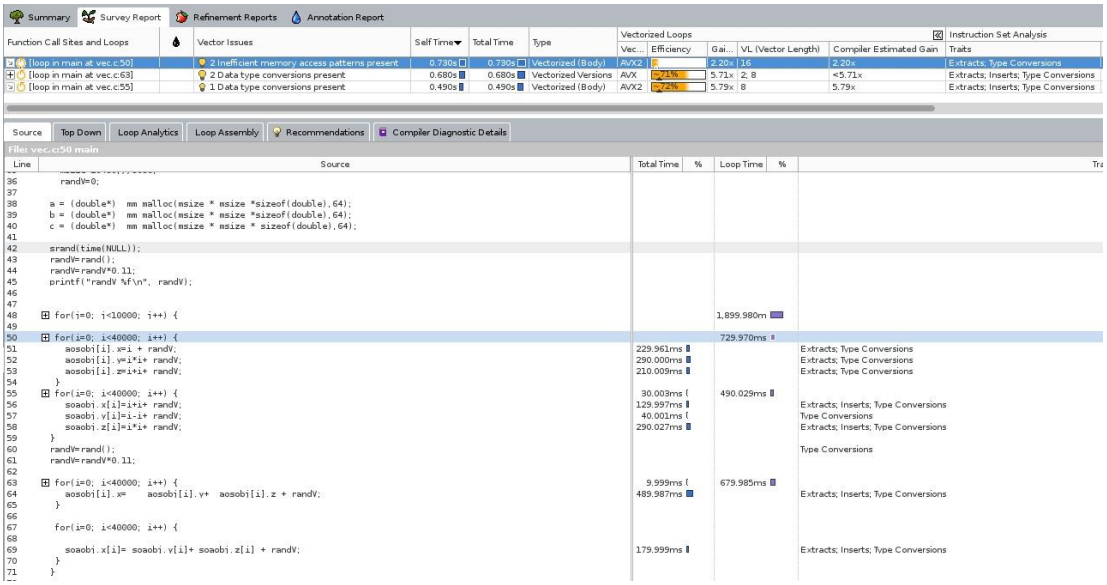
Advisor – New Project



Advisor - Analysis



Advisor – Survey Target



Advisor – Survey Target

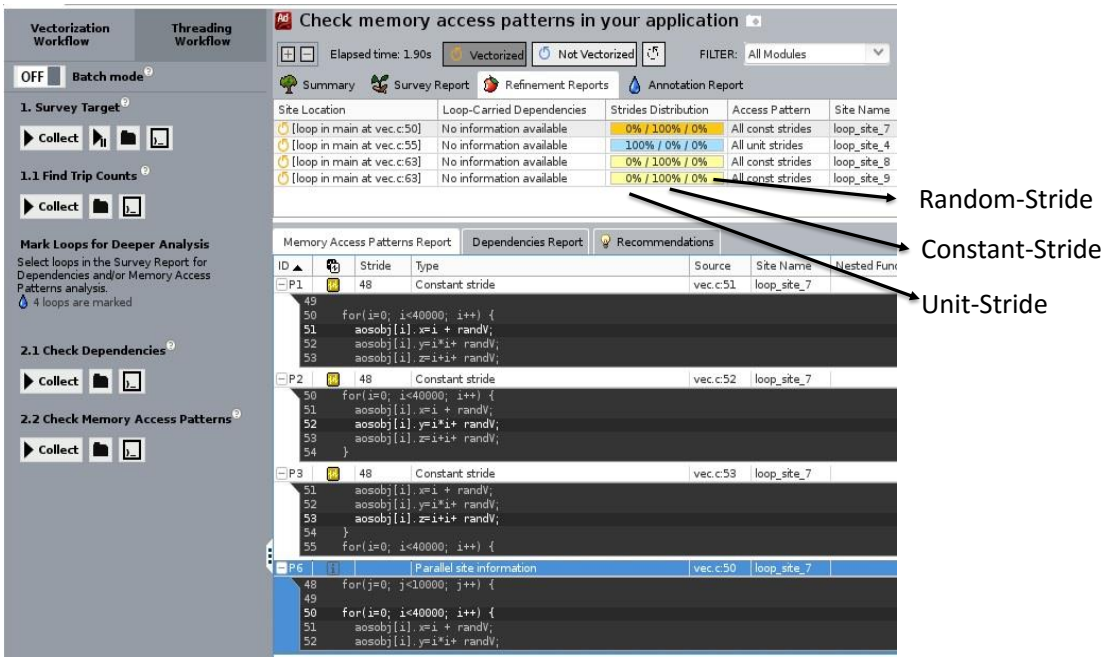
Instruction Set Analysis

Traits	Data Types	Number of Vector Registers	Vector Widths	Instruction Sets
Extracts; Type Conversions	Float32; Float64; Int32; UIn...	15	128/256	AVX; AVX2
Extracts; Inserts; Type Conversions	Float32; Float64	14; 15	128; 256	AVX
Extracts; Inserts; Type Conversions	Float32; Float64; Int32; UIn...	16	256	AVX; AVX2

Advanced

Transformations	Unroll Factor	Vectorization Details	Optimization Details	Location
				vec.c:50
Fused; Unrolled	4		LOOP WAS DISTRIBUTED, CHUNK 1; LOOP WAS DISTRIBUTED, C...	vec.c:63
				vec.c:55

Advisor – Memory Access Patterns



Agenda

- Hybrid Parallel Architectures;
- Memory System and Vector Processing Units;
- Intel Architectures;
- Profiling;
- **Optimizing Memory Access;**
- Auto Vectorization;
- Guided Vectorization;
- Examples.

Stride (array elements)

- Stride:
  - Step size between consecutive access of array elements;
- Strided access with stride k means touching every kth memory element
  - Unit Stride :
    - Sequential access (0, 1, 2, 3, 4, 5, 6, ...)
  - Non-unit stride
    - Constant Stride =
      - 2 is (0, 2, 4, 6, 8, ...)
    - k is (0, k, 2k, 3k, 4k, ...)
    - Random Access;
- Strides > 1 commonly found in multidimensional data
  - Row accesses (stride=N) & diagonal accesses (stride=N+1)
  - Scientific computing (e.g., matrix multiplication)

Padding

- Data structures may have members with different sizes.
- To maintain proper alignment the translator normally inserts additional unnamed data members so that each member is properly aligned.
- Example:

```
struct stu_a {
   int i;
   char c;
};
```

- Actual size 4+1 (5)

- After Padding size 4+4 (5)

- ...

Padding

- Vectorization more efficient with unit strides
  - Non-unit strides will generate gather/scatter
  - Unit strides also better for data locality

Demo: padd.c

Icc padd.c –o padd

./padd

Data layout

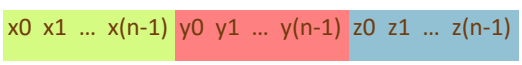
- AoS vs SoA (Array of Structures vs Structure of Arrays)
  - Layout your data as Structure of Arrays (SoA)

```
// Array of Structures (AoS)
struct coordinate {
    float x, y, z;
} crd[N];
    …
for (int i = 0; i < N; i++)
    … = … f(crd[i].x, crd[i],y,
crd[i].z);
```

Consecutive elements in memory


```
// Structure of Arrays (SoA)
struct coordinate {
    float x[N], y[N], z[N];
} crd;
    …
for (int i = 0; i < N; i++)
    … = … f(crd.x[i], crd.y[i],
crd.z[i]);
```

Consecutive elements in memory



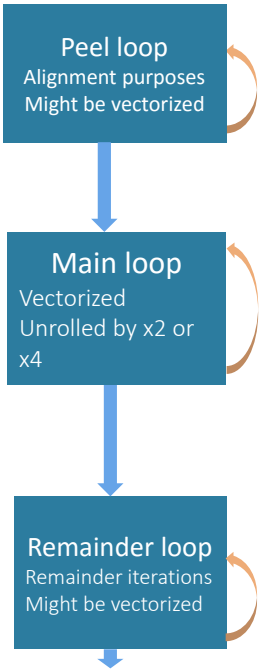
Data Alignment

How to…	Syntax	Semantics
…align data	`void* _mm_malloc(int size, int n)` `void* _mm_free(int size)`	Allocate memory on heap aligned to *n* byte boundary.
	`int posix_memalign (void **p, size_t n, size_t size)`	
	`__declspec(align(n)) array`	Alignment for variable declarations.
…tell the compiler about it	`#pragma vector aligned`	Vectorize assuming all array data accessed are aligned (may cause fault otherwise).
	`__assume_aligned(array, n)`	Compiler may assume array is aligned to *n* byte boundary.

Loop Splitting

- Loop Splitting
  - Set of techniques to breaking the loop into multiple loops which have the same body, but iterate over different contiguous portions of the index range.
    - Body
    - Peel Loop: beginning of loop
    - Remainder Loop: end of loop
- Loop Unrolling
  - Execute a set of iterations as a single iteration;

Vectorization with multi-version loops

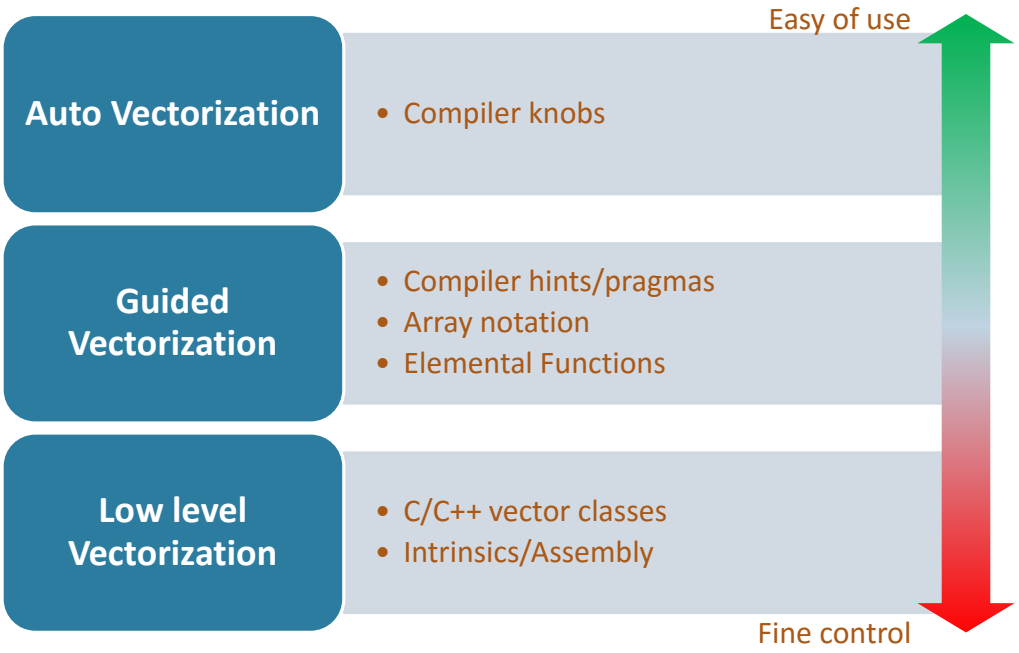



```
LOOP BEGIN at gas_dyn2.f90(2330,26)
<Peeled>
   remark #15389: vectorization support: reference AMAC1U has unaligned
access
   remark #15381: vectorization support: unaligned access used inside loop
body
   remark #15301: PEEL LOOP WAS VECTORIZED
LOOP END

LOOP BEGIN at gas_dyn2.f90(2330,26)
   remark #25084: Preprocess Loopnests: Moving Out Store
   remark #15388: vectorization support: reference AMAC1U has aligned access
   remark #15399: vectorization support: unroll factor set to 2
   remark #15300: LOOP WAS VECTORIZED
   remark #15475: --- begin vector loop cost summary ---
   remark #15476: scalar loop cost: 8
   remark #15477: vector loop cost: 0.620
   remark #15478: estimated potential speedup: 15.890
   remark #15479: lightweight vector operations: 5
   remark #15488: --- end vector loop cost summary ---
   remark #25018: Total number of lines prefetched=4
   remark #25019: Number of spatial prefetches=4, dist=8
   remark #25021: Number of initial-value prefetches=6
LOOP END

LOOP BEGIN at gas_dyn2.f90(2330,26)
<Remainder>
   remark #15388: vectorization support: reference AMAC1U has aligned access
   remark #15388: vectorization support: reference AMAC1U has aligned access
   remark #15301: REMAINDER LOOP WAS VECTORIZED
LOOP END
```


Agenda

- Hybrid Parallel Architectures;
- Memory System and Vector Processing Units;
- Intel Architectures;
- Profiling;
- Optimizing Memory Access;
- **Auto Vectorization;**
- Guided Vectorization;
- Examples.

Vectorization on Intel® compilers



Auto vectorization

- Relies on the compiler for vectorization
  - No source code changes
  - Enabled with `-vec` compiler knob (default in `-O2` and `-O3` modes)
- Compiler smart enough to apply loop transformations
  - It will allow to vectorize more loops

Option	Description
`-O0`	Disables all optimizations.
`-O1`	Enables optimizations for speed which are know to not cause code size increase.
`-O2/-O` (default)	Enables intra-file interprocedural optimizations for speed, including: • **Vectorization** • **Loop unrolling**
`-O3`	**Performs O2 optimizations and enables more aggressive loop transformations such as:** • **Loop fusion** • **Block unroll-and-jam** • **Collapsing IF statements** This option is recommended for applications that have loops that heavily use floating-point calculations and process large data sets. However, it might incur in slower code, numerical stability issues, and compilation time increase.

Vectorization: target architecture options

Option	Description
-mmic	Builds an application that runs natively on Intel® MIC Architecture.
-xfeature -xHost	Tells the compiler which processor features it may target, referring to which instruction sets and optimizations it may generate (not available for Intel® Xeon Phi™ architecture). Values for *feature* are: • COMMON-AVX512 (includes AVX512 FI and CDI instructions) • **MIC-AVX512 (includes AVX512 FI, CDI, PFI, and ERI instructions)** • **CORE-AVX512 (includes AVX512 FI, CDI, BWI, DQI, and VLE instructions)** • **CORE-AVX2** • CORE-AVX-I (including RDRND instruction) • AVX • SSE4.2, SSE4.1 • ATOM_SSE4.2, ATOM_SSSE3 (including MOVBE instruction) • SSSE3, SSE3, SSE2 **When using -xHost, the compiler will generate instructions for the highest instruction set available on the compilation host processor.**
-axfeature	Tells the compiler to generate multiple, feature-specific auto-dispatch code paths for Intel® processors if there is a performance benefit. Values for *feature* are the same described for -xfeature option. Multiple features/paths possible, e.g.: -axSSE2,AVX. It also generates a baseline code path for the default case.

Auto vectorization: not all loops will vectorize

- Data dependencies between iterations
  - Proven Read-after-Write data (i.e., loop carried) dependencies
  - Assumed data dependencies
    - ❑ Aggressive optimizations

RaW dependency

```
for (int i = 0; i < N; i++)
    a[i] = a[i-1] + b[i];
```

- Vectorization won't be efficient
  - Compiler estimates how better the vectorized version will be
  - Affected by data alignment, data layout, etc.

Inefficient vectorization

```
for (int i = 0; i < N; i++)
    a[c[i]] = b[d[i]];
```

- Unsupported loop structure
  - While-loop, for-loop with unknown number of iterations
  - Complex loops, unsupported data types, etc.
  - (Some) function calls within loop bodies

Function call within loop body

```
for (int i = 0; i < N; i++)
    a[i] = foo(b[i]);
```

Validating vectorization

- Generate compiler report about optimizations

  `-qopt-report[=n]` Generate report (level [1..6], default 2)

```
LOOP BEGIN at gas_dyn2.f90(193,11) inlined into gas_dyn2.f90(4326,31)
   remark #15300: LOOP WAS VECTORIZED
   remark #15448: unmasked aligned unit stride loads: 1
   remark #15450: unmasked unaligned unit stride loads: 1
   remark #15475: --- begin vector loop cost summary ---
   remark #15476: scalar loop cost: 53
   remark #15477: vector loop cost: 14.870
   remark #15478: estimated potential speedup: 2.520
   remark #15479: lightweight vector operations: 19
   remark #15481: heavy-overhead vector operations: 1
   remark #15488: --- end vector loop cost summary ---
   remark #25456: Number of Array Refs Scalar Replaced In Loop: 1
   remark #25015: Estimate of max trip count of loop=4
LOOP END
```

Vectorized loop

```
LOOP BEGIN at gas_dyn2.f90(2346,15)
   remark #15344: loop was not vectorized: vector dependence prevents vectorization
   remark #15346: vector dependence: assumed OUTPUT dependence between IOLD line 376 and IOLD line 354
   remark #25015: Estimate of max trip count of loop=3000001
LOOP END
```

Non-vectorized loop

Agenda

- Hybrid Parallel Architectures;
- Memory System and Vector Processing Units;
- Intel Architectures;
- Profiling;
- Optimizing Memory Access;
- Auto Vectorization;
- **Guided Vectorization;**
- Examples.

Intel® compiler directives for vectorization

Directive	Clause	Description
`ivdep`		Instructs the compiler to ignore assumed vector dependencies.
`vector`	`always`	Force vectorization even when it might be not efficient.
	`[un]aligned`	Use [un]aligned data movement instructions for all array vector references.
	`[non]temporal(var1[,…])`	Do or do not generate non-temporal (streaming) stores for the given array variables. On Intel® MIC architecture, generates a cache-line-evict instruction when the store is known to be aligned.
	`[no]vecreminder`	Do (not) vectorize the remainder loop when the main loop is vectorized.
	`[no]mask_readwrite`	Enables/disables memory speculation causing the generation of [non-]masked loads and stores within conditions.

52

Intel® compiler directives for vectorization

Directive	Clause	Description
`ivdep`		Instructs the compiler to ignore assumed vector dependencies.
`simd`	`vectorlength(n1[,…])` `vectorlengthfor(dtype)`	Assume safe vectorization for the given vector length values or data type.

	`private(var1[,…])`
	`firstprivate(var1[,…])`
	`lastprivate(var1[,…])`
		Which variables are private to each iteration; *firstprivate*, initial value is broadcasted to all private instances; *lastprivate*, last value is copied out from the last instance.

	`linear(var1:step1[,…])`	Letting know the compiler that *var1* is incremented by *step1* on every iteration of the original loop.
	`reduction(oper:var1[,…])`	Which variables are reduction variables with a given operator.
	`[no]assert`	Warning or error when vectorization fails.
	`[no]vecremainder`	Do (not) vectorize the remainder loop when the mail loop is vectorized.

53

Guided vectorization: disambiguation hints

- Assume function arguments won't be aliased
  - C/C++: Compile with `-fargument-noalias`
- C99 "restrict" keyword for pointers
  - Compile with `-restrict` otherwise

```
void v_add(float *restrict c,
           float *restrict a,
           float *restrict b)
{
    for (int i = 0; i < N; i++)
        c[i] = a[i] + b[i];
}
```

```
void v_add(float *c, float *a, float *b)
{
for (int i = 0; i < N; i++)
        c[i] = a[i] + b[i];
}
```

54

Guided vectorization:

- `#pragma simd or #pragma ivdep`
  - Force loop vectorization ignoring **all** dependencies
    - Additional clauses for specify reductions, etc.

```
void v_add(float *c, float *a, float *b)
{
#pragma simd
    for (int i = 0; i < N; i++)
        c[i] = a[i] + b[i];
}
```

SIMD loop

```
__declspec(vector)
void v_add(float c, float a, float b)
{
    c = a + b;
}
    …
for (int i = 0; i < N; i++)
    v_add(C[i], A[i], B[i]);
```

SIMD function

55

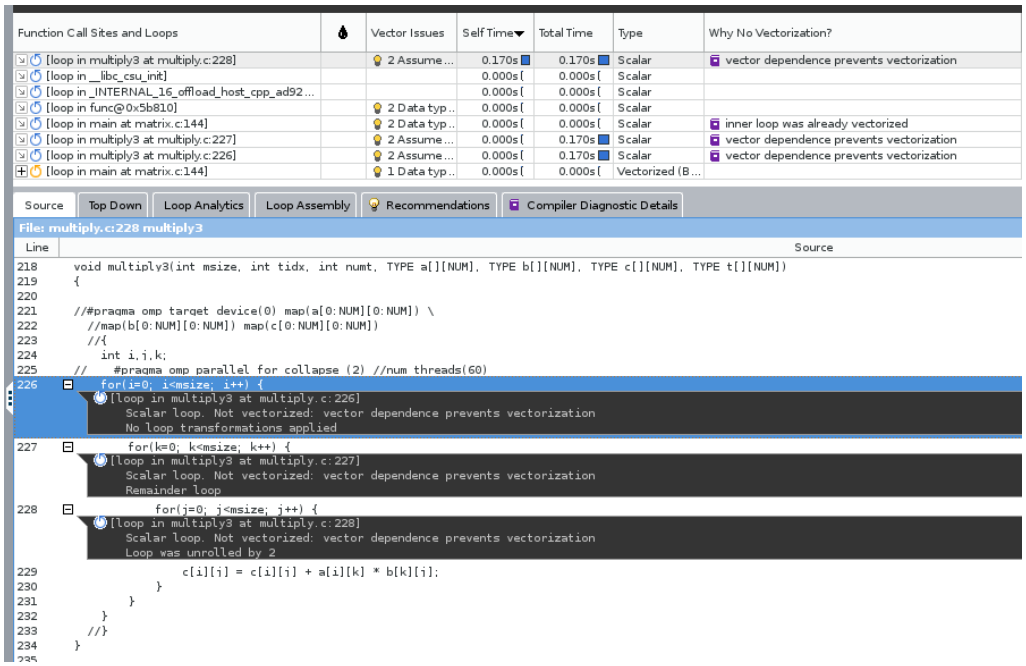
Agenda

- Hybrid Parallel Architectures;
- Memory System and Vector Processing Units;
- Intel Architectures;
- Profiling;
- Optimizing Memory Access;
- Auto Vectorization;
- Guided Vectorization;
- **Examples.**

Matrix Multiplication - Serial

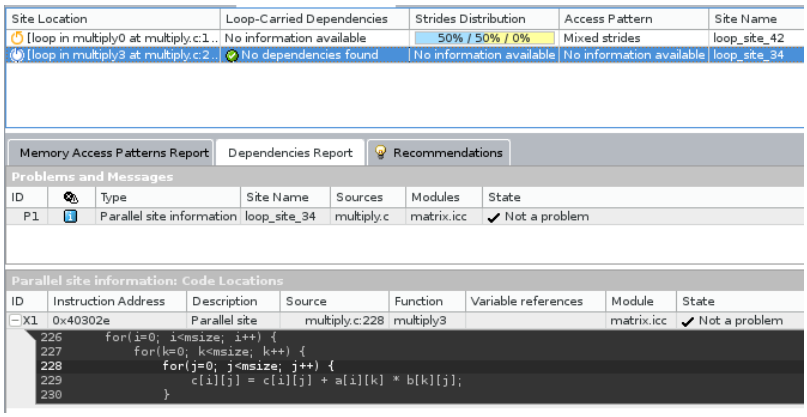
```
void multiply(int msize, int tidx, int numt, TYPE a[][NUM], TYPE
b[][NUM], TYPE c[][NUM], TYPE t[][NUM])
{

int i,j,k;
    for(i=0; i<msize; i++) {
        for(k=0; k<msize; k++) {
            for(j=0; j<msize; j++) {
                c[i][j] = c[i][j] + a[i][k] * b[k][j];
            }
        }
    }
}
```

Matrix Multiplication



Matrix Multiplication

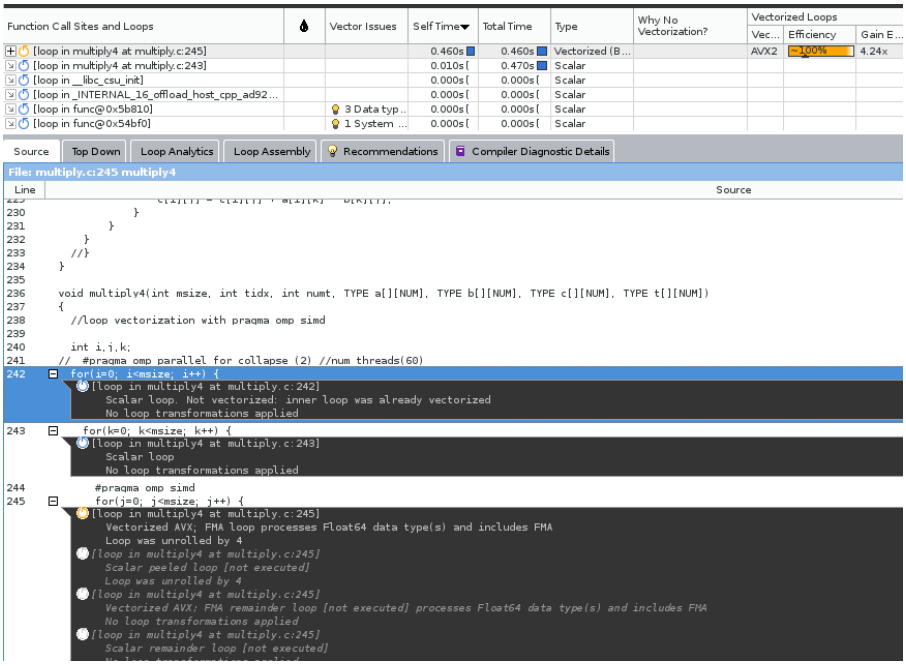
- Check dependency analysis shows that it is safe to enforce the vectorization of this loop



Matrix Multiplication - vectorized

```
void multiply(int msize, int tidx, int numt, TYPE a[][NUM], TYPE
b[][NUM], TYPE c[][NUM], TYPE t[][NUM])
{

int i,j,k;
    for(i=0; i<msize; i++) {
        for(k=0; k<msize; k++) {
            #pragma simd
            for(j=0; j<msize; j++) {
                c[i][j] = c[i][j] + a[i][k] * b[k][j];
            }
        }
    }
}
```


Matrix Multiplication



Example

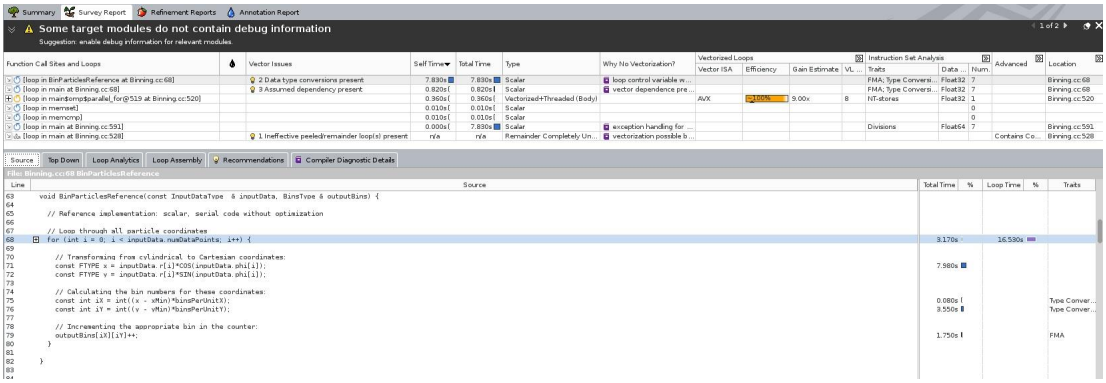
- Particle Binning Problem**[1]**
- Optimizations:
  - Automatic Vectorization
  - Data Alignment

[1] http://colfaxresearch.com/optimization-techniques-for-the-intel-mic-architecture-part-2-of-3-strip-mining-for-vectorization/

Particle Binning - Serial

```
for (int i = 0; i < inputData.numDataPoints; i++) {

    // Transforming from cylindrical to Cartesian coordinates:
    const FTYPE x = inputData.r[i]*COS(inputData.phi[i]);
    const FTYPE y = inputData.r[i]*SIN(inputData.phi[i]);

    // Calculating the bin numbers for these coordinates:
    const int iX = int((x - xMin)*binsPerUnitX);
    const int iY = int((y - yMin)*binsPerUnitY);

}
```

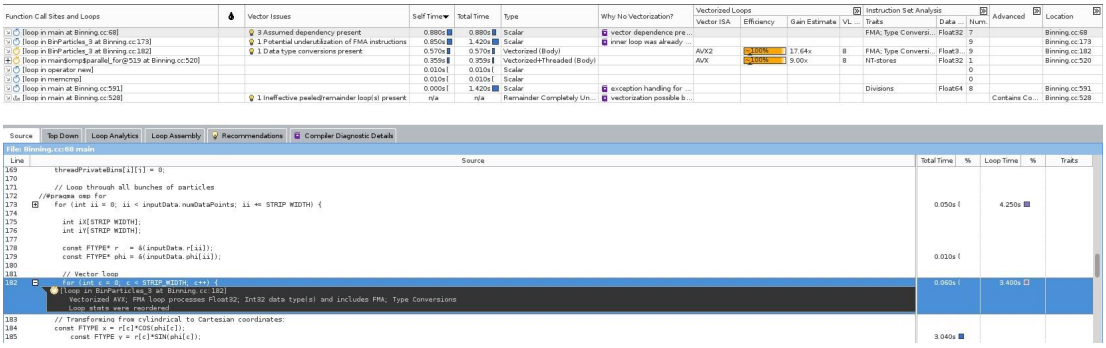
Particle Binning - Serial



Particle Binning - Vectorized

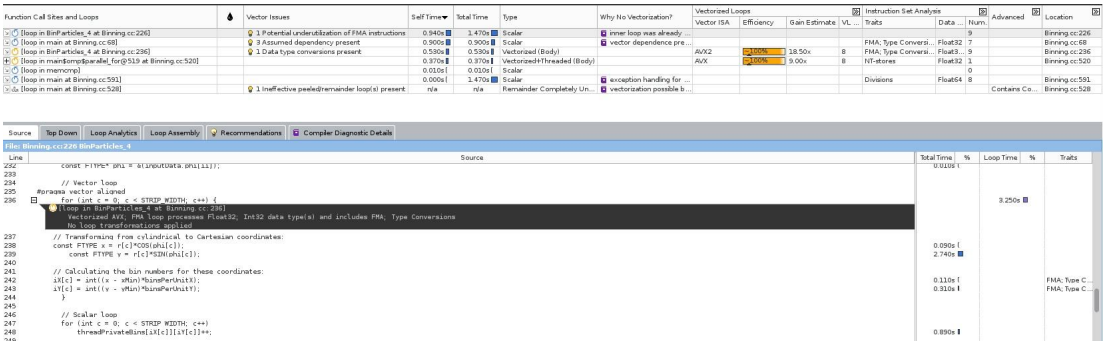
```
for (int ii = 0; ii < inputData.numDataPoints; ii += STRIP_WIDTH) {

    int iX[STRIP_WIDTH];
    int iY[STRIP_WIDTH];

    const FTYPE* r   = &(inputData.r[ii]);
    const FTYPE* phi = &(inputData.phi[ii]);

    // Vector loop

    for (int c = 0; c < STRIP_WIDTH; c++) {
       // Transforming from cylindrical to Cartesian coordinates:
       const FTYPE x = r[c]*COS(phi[c]);
      const FTYPE y = r[c]*SIN(phi[c]);

       // Calculating the bin numbers for these coordinates:
       iX[c] = int((x - xMin)*binsPerUnitX);
       iY[c] = int((y - yMin)*binsPerUnitY);
    }

}
```

Particle Binning - Vectorized



Particle Binning - Data Alignment

```
for (int ii = 0; ii < inputData.numDataPoints; ii += STRIP_WIDTH) {

    int iX[STRIP_WIDTH] __attribute__((aligned(64)));
    int iY[STRIP_WIDTH] __attribute__((aligned(64)));

    const FTYPE* r   = &(inputData.r[ii]);
    const FTYPE* phi = &(inputData.phi[ii]);

    // Vector loop
#pragma vector aligned
    for (int c = 0; c < STRIP_WIDTH; c++) {
       // Transforming from cylindrical to Cartesian coordinates:
       const FTYPE x = r[c]*COS(phi[c]);
      const FTYPE y = r[c]*SIN(phi[c]);

       // Calculating the bin numbers for these coordinates:
       iX[c] = int((x - xMin)*binsPerUnitX);
       iY[c] = int((y - yMin)*binsPerUnitY);
    }

}
```



Particle Binning - Data Alignment

Function Call Sites and Loops		Vector Issues	Self Time	Total Time	Type	Why No Vectorization?	Vectorized Loops				Instruction Set Analysis			Advanced	Location
							Vector ISA	Efficiency	Gain Estimate	VL ...	Traits	Data ...	Num.		
[loop in BinParticles_4 at Binning.cc:226]		1 Potential underutilization of FMA instructions	0.940s	1.470s	Scalar	inner loop was already ...							9		Binning.cc:226
[loop in main at Binning.cc:68]		3 Assumed dependency present	0.900s	0.900s	Scalar	vector dependence pre...					FMA; Type Conversi...	Float32	7		Binning.cc:68
[loop in BinParticles_4 at Binning.cc:236]		1 Data type conversions present	0.530s	0.530s	Vectorized (Body)		AVX2	~100%	18.50x	8	FMA; Type Conversi...	Float3...	9		Binning.cc:236
[loop in mainompparallel_for@519 at Binning.cc:520]			0.370s	0.370s	Vectorized+Threaded (Body)		AVX	~100%	9.00x	8	NT-stores	Float32	1		Binning.cc:520
[loop in memcmp]			0.010s	0.010s	Scalar								0		
[loop in main at Binning.cc:591]			0.000s	1.470s	Scalar	exception handling for ...					Divisions	Float64	8		Binning.cc:591
[loop in main at Binning.cc:528]		1 Ineffective peeled/remainder loop(s) present	n/a	n/a	Remainder Completely Un...	vectorization possible b...								Contains Co...	Binning.cc:528

File: Binning.cc:226 BinParticles_4

```
        const FTYPE* phi = &(inputData.phi[ii]);

        // Vector loop
  #pragma vector aligned
        for (int c = 0; c < STRIP_WIDTH; c++) {
     [loop in BinParticles_4 at Binning.cc:236]
       Vectorized AVX; FMA loop processes Float32; Int32 data type(s) and includes FMA; Type Conversions
       No loop transformations applied
      // Transforming from cylindrical to Cartesian coordinates:
      const FTYPE x = r[c]*COS(phi[c]);
          const FTYPE y = r[c]*SIN(phi[c]);

      // Calculating the bin numbers for these coordinates:
      iX[c] = int((x - xMin)*binsPerUnitX);
      iY[c] = int((y - yMin)*binsPerUnitY);
        }

        // Scalar loop
        for (int c = 0; c < STRIP_WIDTH; c++)
            threadPrivateBins[iX[c]][iY[c]]++;
```

Total Time	%	Loop Time	%	Traits
0.010s				
		3.250s		
0.090s				
2.740s				
0.110s				FMA; Type C...
0.310s				FMA; Type C...
0.890s				

Diffusion - Serial

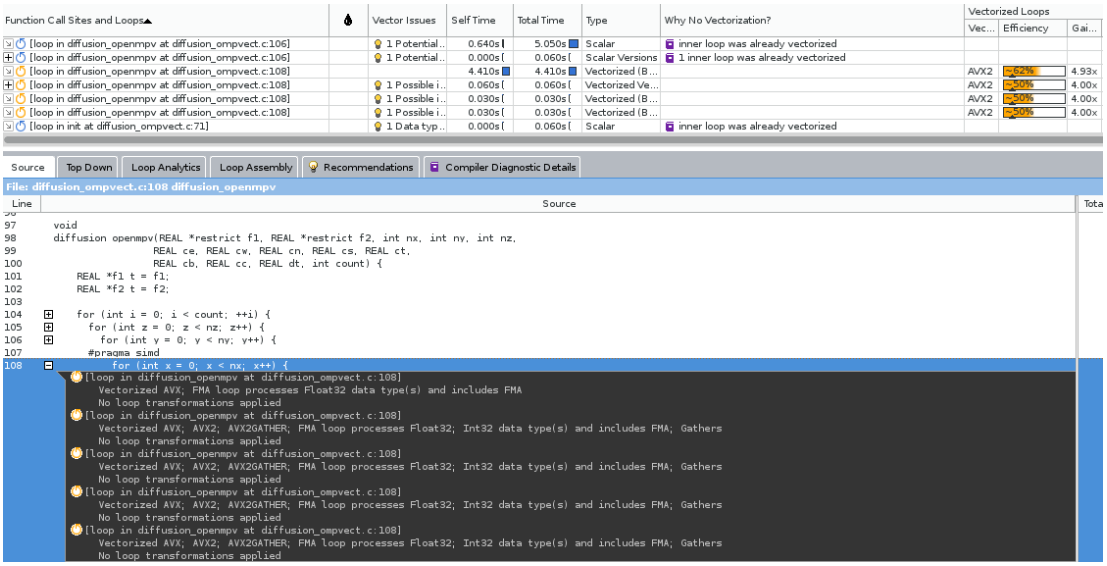
Function Call Sites and Loops		Vector Issues	Self Time	Total Time	Type	Why No Vectorization?
[loop in diffusion_baseline at diffusion_base.c:103]		1 Potential ..	0.000s	10.530s	Scalar	outer loop was not auto-vectorized: consider using SIMD ...
[loop in diffusion_baseline at diffusion_base.c:104]		1 Potential ..	0.010s	10.450s	Scalar	outer loop was not auto-vectorized: consider using SIMD ...
[loop in diffusion_baseline at diffusion_base.c:105]		2 Assume ...	0.130s	10.370s	Scalar	vector dependence: assumed dependence between lines
[loop in diffusion_baseline at diffusion_base.c:105]		2 Assume ...	0.000s	0.080s	Scalar Versions	1 vector dependence: assumed dependence between lines
[loop in diffusion_baseline at diffusion_base.c:107]		2 Assume ...	10.240s	10.240s	Scalar	vector dependence: assumed dependence between lines
[loop in diffusion_baseline at diffusion_base.c:107]		2 Assume ...	0.070s	0.070s	Scalar Versions	1 vector dependence: assumed dependence between lines
[loop in diffusion_baseline at diffusion_base.c:107]		2 Assume ...	0.060s	0.060s	Scalar	vector dependence: assumed dependence between lines

File: diffusion_base.c:107 diffusion_baseline

```
void
diffusion_baseline(REAL *f1, REAL *f2, int nx, int ny, int nz,
                   REAL ce, REAL cw, REAL cn, REAL cs, REAL ct,
                   REAL cb, REAL cc, REAL dt,
                   int count) {
  int i;
  for (i = 0; i < count; ++i) {
    for (int z = 0; z < nz; z++) {
      for (int y = 0; y < ny; y++) {
        //#pragma simd
        for (int x = 0; x < nx; x++) {
     [loop in diffusion_baseline at diffusion_base.c:107]
       Scalar loop. Vector dependence: assumed dependence between lines
       No loop transformations applied
     [loop in diffusion_baseline at diffusion_base.c:107]
       Scalar loop. Vector dependence: assumed dependence between lines
       No loop transformations applied
     [loop in diffusion_baseline at diffusion_base.c:107]
       Scalar loop. Vector dependence: assumed dependence between lines
       No loop transformations applied
     [loop in diffusion_baseline at diffusion_base.c:107]
       Scalar loop. Vector dependence: assumed dependence between lines
       No loop transformations applied
     [loop in diffusion_baseline at diffusion_base.c:107]
       Scalar loop. Vector dependence: assumed dependence between lines
       No loop transformations applied
```

Diffusion - Vectorized

Potential inefficient memory access;

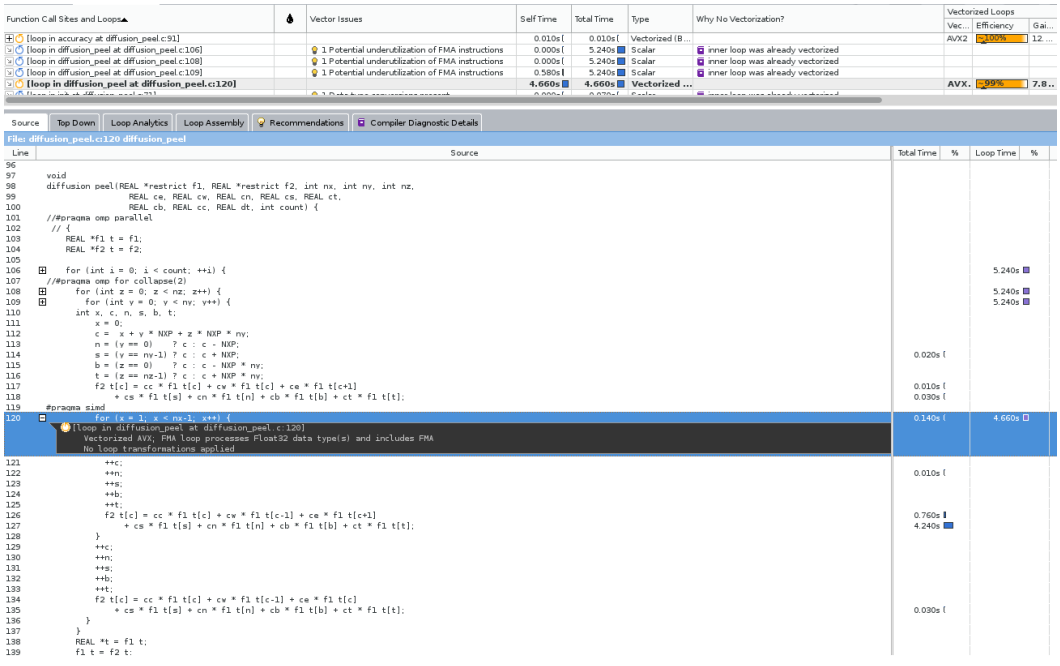


Function Call Sites and Loops		Vector Issues	Self Time	Total Time	Type	Why No Vectorization?	Vectorized Loops		
							Vec...	Efficiency	Gai...
[loop in diffusion_openmpv at diffusion_ompvect.c:106]		1 Potential..	0.640s	5.050s	Scalar	inner loop was already vectorized			
[loop in diffusion_openmpv at diffusion_ompvect.c:106]		1 Potential..	0.000s	0.060s	Scalar Versions	1 inner loop was already vectorized			
[loop in diffusion_openmpv at diffusion_ompvect.c:108]			4.410s	4.410s	Vectorized (B...		AVX2	~62%	4.93x
[loop in diffusion_openmpv at diffusion_ompvect.c:108]		1 Possible i..	0.060s	0.060s	Vectorized Ve...		AVX2	~50%	4.00x
[loop in diffusion_openmpv at diffusion_ompvect.c:108]		1 Possible i..	0.030s	0.030s	Vectorized (B...		AVX2	~50%	4.00x
[loop in diffusion_openmpv at diffusion_ompvect.c:108]		1 Possible i..	0.030s	0.030s	Vectorized (B...		AVX2	~50%	4.00x
[loop in init at diffusion_ompvect.c:71]		1 Data typ..	0.000s	0.060s	Scalar	inner loop was already vectorized			

File: diffusion_ompvect.c:108 diffusion_openmpv

```
Line  Source
97      void
98      diffusion openmpv(REAL *restrict f1, REAL *restrict f2, int nx, int ny, int nz,
99                        REAL ce, REAL cw, REAL cn, REAL cs, REAL ct,
100                       REAL cb, REAL cc, REAL dt, int count) {
101         REAL *f1 t = f1;
102         REAL *f2 t = f2;
103
104         for (int i = 0; i < count; ++i) {
105           for (int z = 0; z < nz; z++) {
106             for (int y = 0; y < ny; y++) {
107           #pragma simd
108               for (int x = 0; x < nx; x++) {
             [loop in diffusion_openmpv at diffusion_ompvect.c:108]
               Vectorized AVX; FMA loop processes Float32 data type(s) and includes FMA
               No loop transformations applied
             [loop in diffusion_openmpv at diffusion_ompvect.c:108]
               Vectorized AVX; AVX2; AVX2GATHER; FMA loop processes Float32; Int32 data type(s) and includes FMA; Gathers
               No loop transformations applied
             [loop in diffusion_openmpv at diffusion_ompvect.c:108]
               Vectorized AVX; AVX2; AVX2GATHER; FMA loop processes Float32; Int32 data type(s) and includes FMA; Gathers
               No loop transformations applied
             [loop in diffusion_openmpv at diffusion_ompvect.c:108]
               Vectorized AVX; AVX2; AVX2GATHER; FMA loop processes Float32; Int32 data type(s) and includes FMA; Gathers
               No loop transformations applied
             [loop in diffusion_openmpv at diffusion_ompvect.c:108]
               Vectorized AVX; AVX2; AVX2GATHER; FMA loop processes Float32; Int32 data type(s) and includes FMA; Gathers
               No loop transformations applied
```

Diffusion - alignment



Function Call Sites and Loops		Vector Issues	Self Time	Total Time	Type	Why No Vectorization?	Vectorized Loops		
							Vec...	Efficiency	Gai...
[loop in accuracy at diffusion_peel.c:91]			0.010s	0.010s	Vectorized (B...		AVX2	~100%	12. ...
[loop in diffusion_peel at diffusion_peel.c:106]		1 Potential underutilization of FMA instructions	0.000s	5.240s	Scalar	inner loop was already vectorized			
[loop in diffusion_peel at diffusion_peel.c:108]		1 Potential underutilization of FMA instructions	0.000s	5.240s	Scalar	inner loop was already vectorized			
[loop in diffusion_peel at diffusion_peel.c:109]		1 Potential underutilization of FMA instructions	0.580s	5.240s	Scalar	inner loop was already vectorized			
[loop in diffusion_peel at diffusion_peel.c:120]			**4.660s**	**4.660s**	**Vectorized ...**		**AVX.**	**~99%**	**7.8..**

File: diffusion_peel.c:120 diffusion_peel

```
Line  Source                                                                  Total Time  %  Loop Time  %
96
97      void
98      diffusion peel(REAL *restrict f1, REAL *restrict f2, int nx, int ny, int nz,
99                     REAL ce, REAL cw, REAL cn, REAL cs, REAL ct,
100                    REAL cb, REAL cc, REAL dt, int count) {
101     //#pragma omp parallel
102      // {
103         REAL *f1 t = f1;
104         REAL *f2 t = f2;
105
106         for (int i = 0; i < count; ++i) {                                                 5.240s
107     //#pragma omp for collapse(2)
108           for (int z = 0; z < nz; z++) {                                                  5.240s
109             for (int y = 0; y < ny; y++) {                                                5.240s
110           int x, c, n, s, b, t;
111               x = 0;
112               c =  x + y * NXP + z * NXP * ny;
113               n = (y == 0)    ? c : c - NXP;
114               s = (y == ny-1) ? c : c + NXP;                              0.020s
115               b = (z == 0)    ? c : c - NXP * ny;
116               t = (z == nz-1) ? c : c + NXP * ny;
117               f2 t[c] = cc * f1 t[c] + cw * f1 t[c] + ce * f1 t[c+1]      0.010s
118                   + cs * f1 t[s] + cn * f1 t[n] + cb * f1 t[b] + ct * f1 t[t];   0.030s
119     #pragma simd
120               for (x = 1; x < nx-1; x++) {                                0.140s            4.660s
          [loop in diffusion_peel at diffusion_peel.c:120]
            Vectorized AVX; FMA loop processes Float32 data type(s) and includes FMA
            No loop transformations applied
121                 ++c;
122                 ++n;                                                      0.010s
123                 ++s;
124                 ++b;
125                 ++t;
126                 f2 t[c] = cc * f1 t[c] + cw * f1 t[c-1] + ce * f1 t[c+1]  0.760s
127                     + cs * f1 t[s] + cn * f1 t[n] + cb * f1 t[b] + ct * f1 t[t];   4.240s
128               }
129               ++c;
130               ++n;
131               ++s;
132               ++b;
133               ++t;
134               f2 t[c] = cc * f1 t[c] + cw * f1 t[c-1] + ce * f1 t[c]
135                   + cs * f1 t[s] + cn * f1 t[n] + cb * f1 t[b] + ct * f1 t[t];   0.030s
136             }
137           }
138           REAL *t = f1 t;
139           f1 t = f2 t;
```

Interpolation

```
__declspec(vector)
int FindPosition(double x) {
   return (int)(log(exp(x*steps)));
}

__declspec(vector)
double Interpolate(double x, const point*
vals)
{
   int ind = FindPosition(x);
   ...

   return res;
}
```

```
int main ( int argc , char argv [] )
{
   . . .
  for ( i=0; i <ARRAY_SIZE;++ i ) {
      dst[i] = Interpolate( src[i], vals ) ;
   }
   . . .
}
```

George M. Raskulinec, Evgeny Fiksman "Chapter 22 - **SIMD functions via OpenMP**", In High Performance Parallelism Pearls, edited by James Reinders and Jim Jeffers, Morgan Kaufmann, Boston, 2015, Pages 171-190, ISBN 9780128038192

Vectorization report - Interpolate

```
Begin optimization report for: Interpolate..._simdsimd3__H2n_v1_s1.P(double, const point *)

   Report from: Vector optimizations [vec]

remark #15301: FUNCTION WAS VECTORIZED   [ main.c(74,48) ]
================================================================================

Begin optimization report for: Interpolate..._simdsimd3__H2m_v1_s1.P(double, const point *)

   Report from: Vector optimizations [vec]

remark #15301: FUNCTION WAS VECTORIZED   [ main.c(74,48) ]
================================================================================

Begin optimization report for: Interpolate..._simdsimd3__L4n_v1_s1.V(double, const point *)

   Report from: Vector optimizations [vec]

remark #15301: FUNCTION WAS VECTORIZED   [ main.c(74,48) ]
remark #15415: vectorization support: gather was generated for the variable pnt:  indirect access, 64bit indexed   [ main.c(78,26) ]
remark #15415: vectorization support: gather was generated for the variable pnt:  indirect access, 64bit indexed   [ main.c(78,36) ]
================================================================================

Begin optimization report for: Interpolate..._simdsimd3__L4m_v1_s1.V(double, const point *)

   Report from: Vector optimizations [vec]

remark #15301: FUNCTION WAS VECTORIZED   [ main.c(74,48) ]
remark #15415: vectorization support: gather was generated for the variable pnt: masked, indirect access, 64bit indexed   [ main.c(78,26) ]
remark #15415: vectorization support: gather was generated for the variable pnt: masked, indirect access, 64bit indexed   [ main.c(78,36) ]
```

Vectorization report - FindPosition

egin optimization report for: FindPosition.._simdsimd3__H2n_v1.P(double)

Report from: Vector optimizations [vec]

remark #15301: FUNCTION WAS VECTORIZED [main.c(70,28)]
==

Begin optimization report for: FindPosition.._simdsimd3__H2m_v1.P(double)

Report from: Vector optimizations [vec]

remark #15301: FUNCTION WAS VECTORIZED [main.c(70,28)]
==

Begin optimization report for: FindPosition.._simdsimd3__L4n_v1.V(double)

Report from: Vector optimizations [vec]

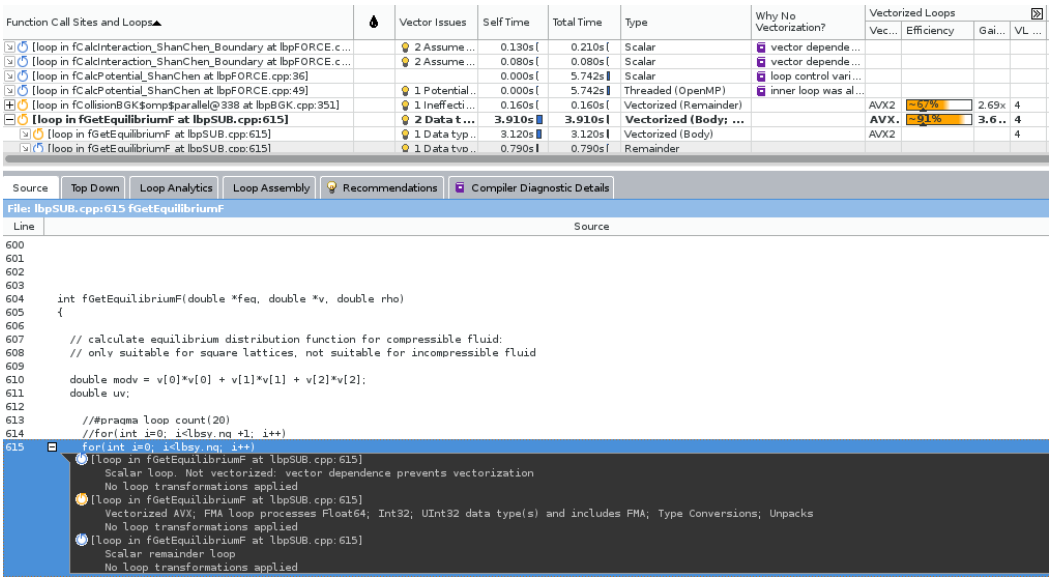
remark #15301: FUNCTION WAS VECTORIZED [main.c(70,28)]
==

Begin optimization report for: FindPosition.._simdsimd3__L4m_v1.V(double)

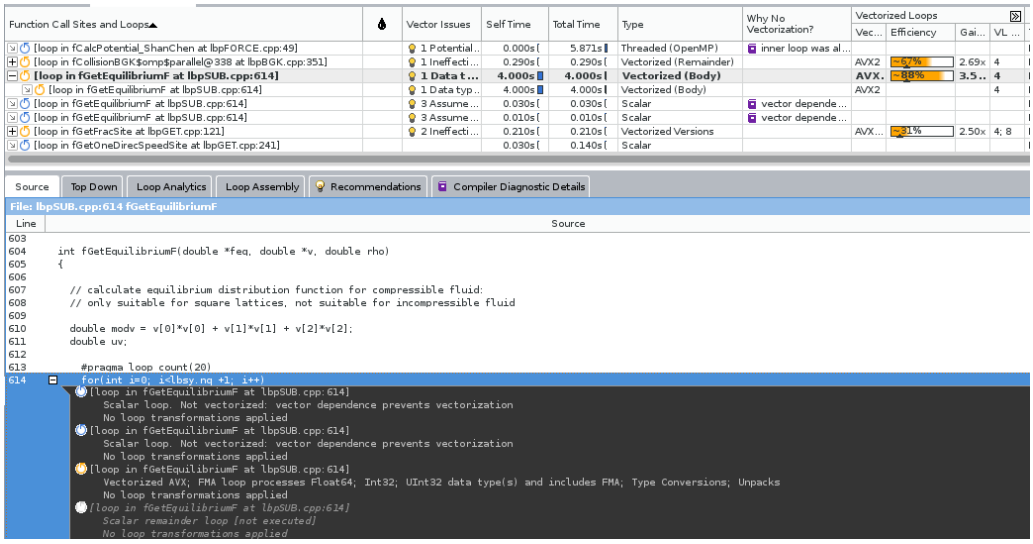
Report from: Vector optimizations [vec]

remark #15301: FUNCTION WAS VECTORIZED [main.c(70,28)]
==

Lattice Boltzmann



Lattice Boltzmann



unesp

intel Software

Questions?

Silvio Stanzani , Raphael Cóbe , Rogério Iope
Núcleo de Computação Científica – UNESP

silvio@ncc.unesp.br
rmcobe@ncc.unesp.br
rogerio@ncc.unesp.br

Pesquisa: http://bit.ly/ERAD_Survey

Efetue o cadastro no Community Server da DataBricks para o curso prático:
https://community.cloud.databricks.com

NoteBook para o curso Erad: http://bit.ly/2aMLqIp

Apresentação PPT:
https://drive.google.com/open?id=1rxHbrzEBCmiB0B_XaAr0GTpED94etgMSvKVJpY6XiWQ

Novo Notebook:
http://bit.ly/2aEgVVX

Links Interessantes:

Cursos Spark Gratuítos:
https://databricks.com/spark/training

Download Spark:
http://spark.apache.org/downloads.html

Documentação Spark:
http://spark.apache.org/docs/latest/

Introdução ao Apache Spark para Data Science

Thiago Baldim
Mastery Consulting

Instruções Minicurso

- http://bit.ly/ERAD-SPARK
- Preencham a Pesquisa!

Apache Spark a sua solução Big Data!

O que é Big Data?

Problemas do Big Data

- Crescimento de dados mais rápido que a capacidade de processamento
- Identificar qual informação é relevante
- Integrar informações de diferentes bases de dados.
- Armazenamento
- Acesso rápido
- Streaming de Dados


Spark


Apache Atlas


kafka


Spark


FLUME


hadoop


HIVE

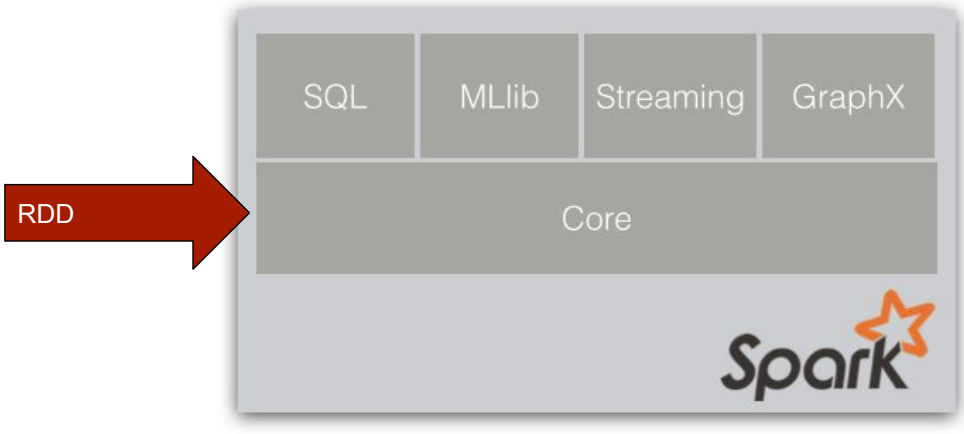
HCatalog


MESOS

O que o Spark faz?

- MapReduce
- DataFrames (SQL em arquivos não estruturados)
- Streaming
- Machine Learning
- Graph

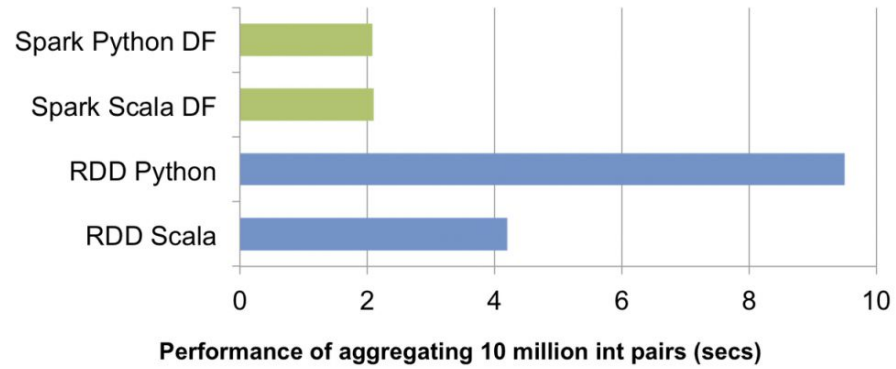
Resilient Distributed Datasets (RDDs)



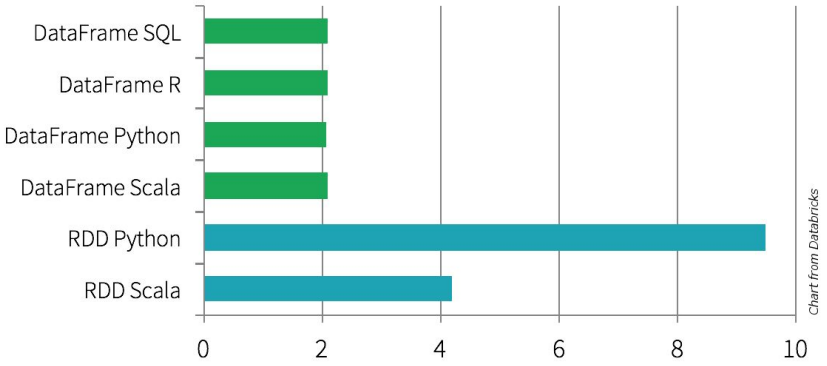
Exemplo de RDD Python

```
>>> text = ‘Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent gravida leo quis magna varius feugiat. Sed …’

>>> wordCounts = sc.parallelize(text)
                  .flatMap(lambda line: line.split())
                  .map(lambda word: (word, 1))
                  .reduceByKey(lambda a, b: a+b)

>>> wordCount.collect()

Out[1]: ('Vestibulum', 2), ('elit', 1), ('tortor', 2), ('justo', 1), ('ex.', 1), ('et.', 1), ('posuere', 2), ('fermentum', 1), ('consectetur', 1), ('interdum', 1), ('mauris.', 1), ('massa', 1), ('a', 2), ('leo,', 1), ('consequat', 1), ('Lorem', 1), ('ipsum.', 1), ('placerat.', 1), ('ipsum', 1), ('gravida', 1), ('accumsan', 1), ('arcu', 1), ('et', 1), ('ut', 1), ('amet', 1), ('ullamcorper', 1), ('ante,', 1), ('Duis', 2), ('at', 1)
```

Scala vs Python

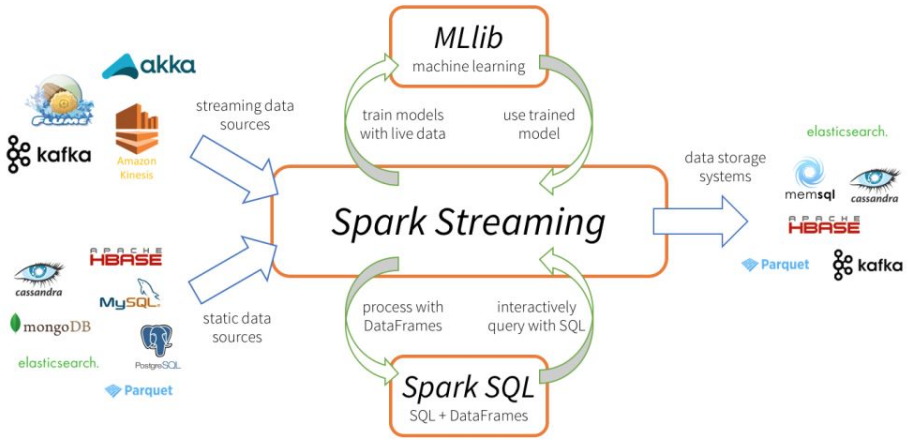


Dataframes

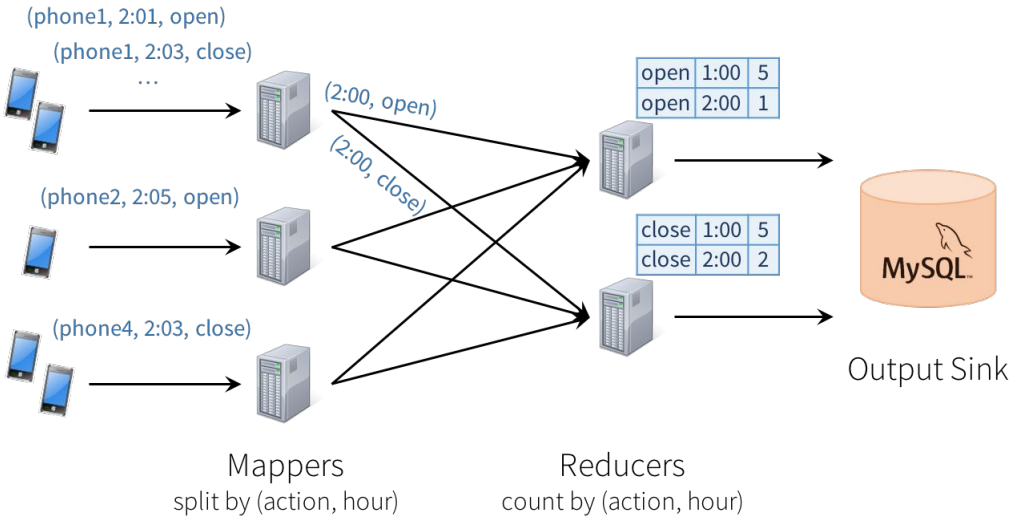


Time to aggregate 10 million integer pairs (in seconds)

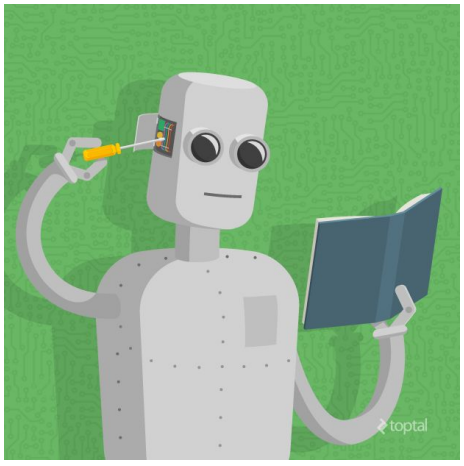
Spark Streaming



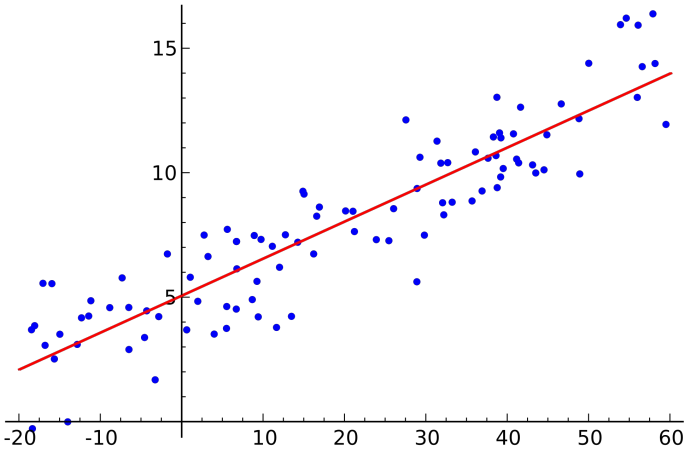
Problema do Streaming



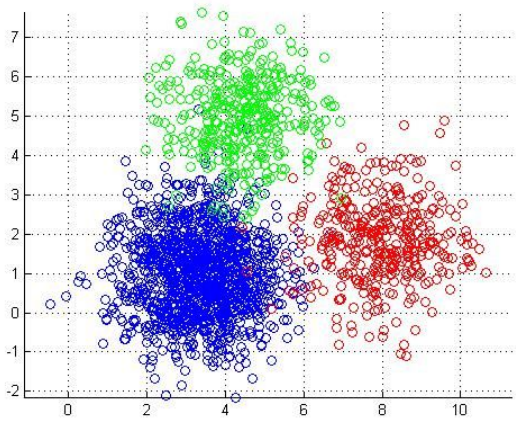
Machine Learning



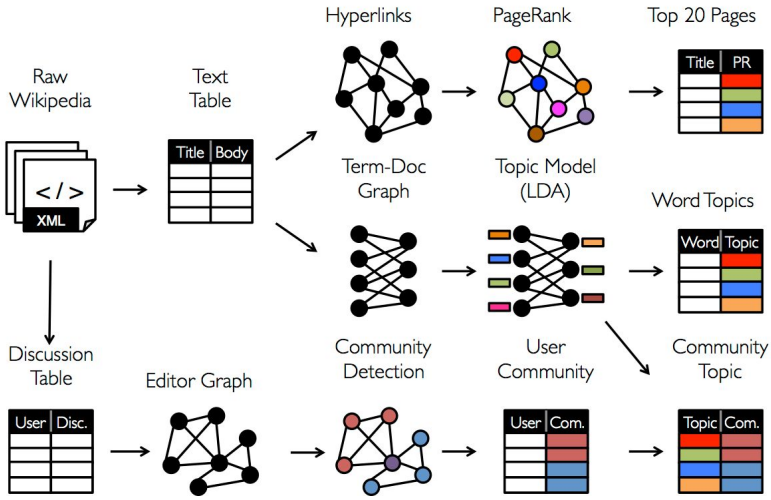
Modelos Lineares



Clusterização



GraphX



Qual o proximo passo para o Spark?

Material Prático!

NETFLIX

Filtro Colaborativo