Uso da Classificação Dwarf Mine para a Avaliação Comparativa entre a Arquitetura CUDA e Multicores*

Laércio Lima Pilla[1], **Philippe Olivier Alexandre Navaux**[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brasil

{laercio.pilla,navaux}@inf.ufrgs.br

Abstract. *The use of graphic processors (GPUs) as accelerators for general purpose applications is becoming commonplace. Still, even with the success achieved on some ported applications, there is not a clear definition of the performance behavior to be expected with applications on these complex architectures. In this context, this paper presents a performance comparison between the CUDA GPU architecture and the multicores Nehalem and Core 2 Duo considering three applications representative of classes according to the Dwarf Mine classification. The results when comparing CUDA to a dual Nehalem system showed better or similar performances. In conclusion, these results indicate a successful mapping of three important classes of applications to the GPUs.*

Resumo. *O uso de processadores gráficos (GPUs) para a aceleração de aplicações de propósito geral vem ganhando popularidade. Porém, apesar dos ganhos obtidos com algumas aplicações portadas para GPUs, ainda não existe uma definição clara do comportamento a ser esperado nestas arquiteturas. Nesse contexto, este artigo apresenta uma comparação entre a arquitetura de GPUs CUDA e os multicore Nehalem e Core 2 Duo considerando três aplicações pertencentes a categorias da classificação Dwarf Mine. Os resultados quando comparando CUDA ao sistema com dois processadores Nehalem apresentaram desempenhos similares e um speedup de 4, o que indica um mapeamento positivo entre três importantes classes de aplicações para GPUs.*

1. Introdução

A indústria da computação segue uma mudança de sentido em direção à computação paralela, como consequência das limitações de potência e desempenho dos processadores sequenciais [Asanovic et al. 2006, Asanovic et al. 2009]. Uma plataforma alternativa que vem ganhando popularidade para computações de alto desempenho envolve o uso de processadores gráficos (*Graphics Processing Units*, ou GPUs) como aceleradores. A arquitetura CUDA [NVIDIA 2009a] é um notável exemplo disso. Essa arquitetura massivamente paralela tem provido aumentos de desempenho para várias aplicações, como engenharia de petróleo [Panetta et al. 2009], climatologia [Michalakes and Vachharajani 2008] e *ray tracing* [Hou et al. 2008].

Entretanto, o desenvolvimento de aplicações para esta arquitetura não é trivial. Isso acontece devido aos aspectos complexos das GPUs, como hierar-

*Projeto parcialmente apoiado pelo CNPq e pela empresa Microsoft.

quias de processamento e memória, assim como diferentes quantidades de componentes disponíveis para operações de ponto flutuante com precisão simples e dupla. Enquanto a maior parte dos estudos foca em um melhor uso da arquitetura [Nukada and Matsuoka 2009, Volkov and Demmel 2008], no desempenho obtido com aplicações específicas [Gulati and Khatri 2008, Tölke 2008] ou em facilitar a programação para GPUs [Hou et al. 2008, Lee et al. 2009], há uma menor quantidade de pesquisas disponíveis sobre quais categorias de aplicações podem realmente tirar proveito dessa arquitetura.

Uma melhor definição das características de aplicações compatíveis com a arquitetura CUDA poderia trazer benefícios aos usuários ao coibir implementações não produtivas, assim como investimentos perdidos em equipamentos. Além disso, tal definição poderia ser de auxílio ao precisar o comportamento esperado com aplicações em GPUs e a utilidade dessa arquitetura.

Neste contexto, este artigo apresenta uma avaliação comparativa da arquitetura CUDA utilizando três categorias representativas de aplicações da classificação *Dwarf Mine* de Berkeley [Asanovic et al. 2006]. Os resultados de desempenho da arquitetura são comparados aos obtidos com as arquiteturas de processadores *multicore* Nehalem [Barker et al. 2008] e Core 2 Duo, todas utilizando operações de ponto flutuante de precisão dupla.

A Seção 2 apresenta informações referentes à classificação *Dwarf Mine*. A Seção 3 trata de detalhes da arquitetura CUDA. As categorias de aplicações escolhidas são exibidas na Seção 4 e seus resultados experimentais são apresentados na Seção 5. A Seção 6 aborda trabalhos relacionados a esta pesquisa. Por fim, a Seção 7 encerra o artigo abordando suas principais contribuições e trabalhos futuros.

2. Classificação Dwarf Mine

A classificação *Dwarf Mine*- também conhecida como *13 Dwarfs* [Asanovic et al. 2006, Asanovic et al. 2009] - organiza métodos algorítmicos (*dwarfs*) através de seus comportamentos comuns de computação e comunicação. Ela foi primeiramente desenvolvida por Phillip Collela com sete categorias e, posteriormente, estendida por pesquisadores de Berkeley. Tal classificação foca nos padrões básicos que persistiram nas aplicações, sendo independentes de implementação. As treze categorias da classificação são listadas abaixo.

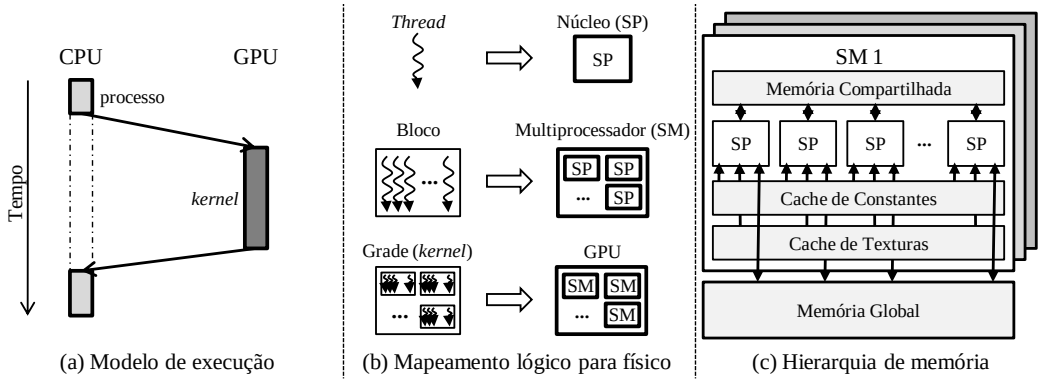
1. *Dense Linear Algebra*;
2. *Sparse Linear Algebra*;
3. *Spectral Methods*;
4. *N-Body Methods*;
5. *Structured Grids*;
6. *Unstructured Grids*;
7. *MapReduce*;
8. *Combinational Logic*;
9. *Graph Traversal*;
10. *Dynamic Programming*;
11. *Back-track and Branch+Bound*;
12. *Graphical Models*; e

13. *Finite State Machines*.

O presente artigo foca nas categorias *MapReduce*, *Spectral Methods* e *Sparse Linear Algebra*. Para a avaliação experimental, três perfis paralelos (*benchmarks*) do NAS [Bailey et al. 1994, Jin et al. 1999], mapeados para essas categorias, foram implementados para a arquitetura CUDA. Maiores detalhes sobre tais categorias e *benchmarks* são apresentados na Seção 4.

3. Arquitetura CUDA

CUDA (*Compute Unified Device Architecture* [NVIDIA 2009a] significa tanto uma arquitetura SIMD (*Single Instruction, Multiple Data*) [Duncan 1990] de GPUs quanto um modelo de programação que estende linguagens (como C) para utilizar essa arquitetura. A GPU funciona como um acelerador paralelo para a CPU (*Central Processing Unit*), como ilustrado na Figura 1(a). Uma *thread* ou processo em CPU executa uma chamada de função especial, nomeada *kernel*, a qual executa na GPU. Todos dados necessários para tal execução devem ser transferidos com antecedência para a placa gráfica, pois essa possui sua própria memória.



(a) Modelo de execução (b) Mapeamento lógico para físico (c) Hierarquia de memória

Figura 1. Características da arquitetura CUDA.

Uma GPU com arquitetura CUDA é composta por núcleos nomeados *Scalar Processors* (**SPs**). Cada SP pode executar instruções sobre inteiros e números de ponto flutuante de precisão simples. Os SPs são organizados em multiprocessadores nomeados *Streaming Multiprocessors* (**SMs**). Na atual geração da arquitetura, cada SM é composto de 8 SPs e uma Unidade de Instruções. Assim sendo, todos SPs em um mesmo SM executam a mesma instrução ao mesmo ciclo, o que pode ser visto como uma máquina vetorial [Volkov and Demmel 2008]. Em adição, há apenas uma unidade de precisão dupla por SM.

O modelo de programação de CUDA trabalha com a abstração de milhares de *threads* executando em paralelo. Essas *threads* são organizadas em blocos. *Threads* em um mesmo bloco podem ser facilmente sincronizadas com uma função de barreira. Múltiplos blocos são organizados em uma grade, o que compõe um *kernel*. Essa organização hierárquica lógica é mapeada para a hierarquia física conforme ilustrado na Figura 1(b). Cada bloco executa em um SM e cada uma de suas *threads* faz suas computações em um dos SPs. Um bloco pode possuir mais *threads* do que um SM pode computar em paralelo, assim como podem haver mais blocos do que SMs.

A arquitetura CUDA também possui uma hierarquia de memória, apresentada na Figura 1(c). Todas as memórias internas ao SM possuem pequeno tamanho e baixa latência. As caches aceleram acessos somente de leitura às texturas e constantes. Cada bloco de *threads* executando em um mesmo SM enxerga sua própria memória compartilhada. A memória global da GPU é de grande tamanho e alta latência (de 400 a 600 ciclos) [NVIDIA 2009a] e pode ser acessada por todas as *threads*. Para aumentar a eficiência no uso da largura de banda da memória, alguns padrões de acesso podem coalescer (agregar) diversas requisições em um única requisição acessando um trecho maior de memória.

Todas essas características podem influenciar o desempenho de aplicações de diferentes formas. Especialmente, o uso de ponto flutuante de precisão dupla incorre em conflitos em bancos na memória compartilhada, redução na quantidade de registradores disponíveis (pois cada variável acaba utilizando dois registradores), incapacidade de utilizar as caches de texturas e constantes, assim como resulta em um desempenho teórico máximo uma ordem de magnitude menor do que o obtido com precisão simples. Outrossim, operações com precisão dupla somente estão disponíveis nas placas gráficas mais modernas (com *Compute Capability* 1.3 ou acima) [NVIDIA 2009a].

4. Implementação de Dwarfs

Para os testes com as três categorias de problemas escolhidas, três *benchmarks* paralelos do NAS (*NAS Parallel Benchmarks*, ou NPB) [Bailey et al. 1994, Jin et al. 1999], originalmente implementados com a biblioteca OpenMP[1], foram portados para a GPU com CUDA[2].

O NPB é um conjunto gratuito de *benchmarks* para a avaliação de arquiteturas paralelas e é composto por aplicações representativas de dinâmica de fluídos. Estas aplicações podem ser facilmente mapeadas para a classificação Dwarf Mine, como visto em [Asanovic et al. 2006]. Cada *benchmark* contém seus próprios códigos de verificação de desempenho e corretude, além de instâncias de teste com diferentes tamanhos (em ordem crescente: W, A, B, C e D). Todas as operações de ponto flutuante nos NPB são executadas com precisão dupla.

As subseções seguintes apresentam as principais características das três categorias escolhidas e de seus respectivos *benchmarks*.

4.1. MapReduce - EP

A categoria *MapReduce* trata sobre aplicações com paralelismo massivo e poucas ou ausentes comunicações. Esse tipo de aplicação pode ser visto em 3 partes: (i) o mapeamento de tarefas independentes para múltiplos processadores; (ii) as computações paralelas; e (iii) a redução dos resultados independentes. A Figura 2 ilustra essa ideia. Esse comportamento pode ser visto em métodos de Monte Carlo [Asanovic et al. 2006], buscas de força bruta em criptografia, entre outros. Esse comportamento é tão importante que a ideia de *MapReduce* foi adicionada como um padrão estrutural para arquitetar aplicações paralelas [Asanovic et al. 2009].

[1]Código original em C+OpenMP disponível gratuitamente na página do Projeto Omni em *http://www.hpcs.cs.tsukuba.ac.jp/omni-openmp/*

[2]Código implementado em C+CUDA disponível gratuitamente na página do Projeto HPCGPU em *http://hpcgpu.codeplex.com*

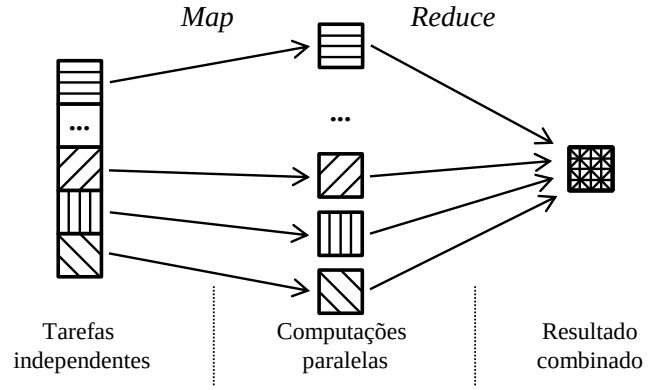


Figura 2. Comportamento da categoria *MapReduce*.

O *benchmark Embarrassingly Parallel* (EP) gera pares de números aleatórios. Ele é utilizado para estimar o máximo desempenho atingível com ponto flutuante de precisão dupla em arquiteturas. Esse perfil paralelo apresenta regularidade nas computações e acessos à memória, além de uma alta intensidade aritmética. A quantidade de memória necessária para a execução do *benchmark* é reduzida, pois pode ser reutilizada.

O laço paralelo principal do *benchmark* EP foi portado para a GPU como um *kernel*. Dentro dele, há mínima divergência entre as *threads*. Além disso, os acessos à memória global da GPU são coalescidos. Para a obtenção do coalescimento, os vetores independentes foram armazenados como uma matriz transposta [Lee et al. 2009]. Após o *kernel* principal, uma redução paralela é executada na GPU para agrupar os resultados.

4.2. Spectral Methods - FT

A categoria *Spectral Methods* foca em aplicações que trabalham com dados no domínio da frequência [Asanovic et al. 2006]. Essas aplicações combinam múltiplas etapas, cada qual com seu próprio padrão de acesso aos dados, como ilustrado na Figura 3. Isso leva a permutações de dados e comunicações todos-para-todos entre as etapas. Aplicações da categoria *Spectral Methods* apresentam uma intensidade aritmética reduzida (devido aos movimentos de dados) e usam operações como $multiply - add$.

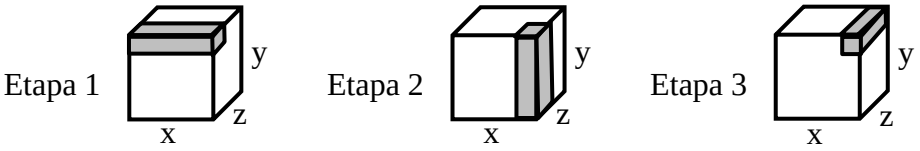


Figura 3. Diferentes padrões de acesso a dados da categoria *Spectral Methods*.

O *benchmark Fast Fourier Transform* (FT) resolve uma equação parcial tridimensional através de FFTs [Bailey et al. 1994]. Ele é utilizado para testar o desempenho de comunicações todos-para-todos regulares. Métodos de Transformada de Fourier são presentes em diversas áreas, como Física, Astronomia e Finanças Computacionais [Govindaraju et al. 2008]. Esses métodos são tão representativos que a própria categoria *Spectral Methods* também é chamada de FFT [Asanovic et al. 2009].

As diferentes etapas do *benchmark* FT foram implementadas como *kernels* para executarem na GPU, o que leva a uma sincronização automática dos dados entre as etapas. Os dados da matriz tridimensional são permutados entre etapas, o que é feito de uma maneira coalescida para diminuir sobrecustos ligados à memória. Experimentos iniciais [3] mostraram que é mais rápido executar um *kernel* com acessos não coalescidos à memória do que permutar os dados de forma não coalescida e executar o *kernel* com acessos coalescidos. Assim, uma etapa específica do *benchmark* segue a primeira abordagem. A biblioteca CUFFT [NVIDIA 2009a], a qual oferece funções de FFT com CUDA, não foi utilizada pois o *benchmark* necessita acessar dados internos à matriz para a verificação de corretude dos resultados.

4.3. Sparse Linear Algebra - CG

A categoria *Sparse Linear Algebra* compreende as características de algoritmos de matrizes esparsas. Devido a esse tipo de matriz conter grandes quantidades de valores iguais a zero, os seus dados são usualmente armazenados omitindo esses zeros de forma comprimida, como representado na Figura 4. Assim, os dados acabam sendo acessados através de requisições indexadas na memória, o que leva a acessos não organizados e dependências paralelas complexas. Este tipo de algoritmo é conhecido pelo seu relativamente baixo desempenho [Vuduc et al. 2005]. Este comportamento pode ser visto, por exemplo, em algoritmos de grafos esparsos [Harish and Narayanan 2007].

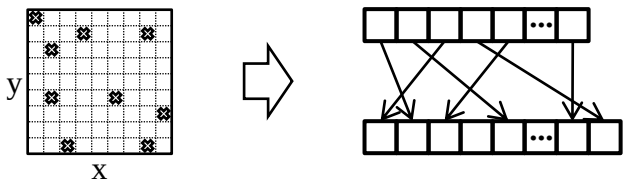


Figura 4. Características da categoria *Sparse Linear Algebra*.

O *benchmark Conjugate Gradient* (CG) computa uma aproximação do menor autovalor (*eigenvalue*) de uma grande, esparsa e simétrica matriz positiva definida [Bailey et al. 1994]. Ele testa o desempenho de comunicações irregulares e apresenta múltiplas reduções internas, o que resulta em diversas sincronizações. Os acessos esparsos à memória subutilizam a largura de banda da memória da GPU, pois cada operação de leitura ou escrita acaba sendo feita separadamente.

Para diminuir os sobrecustos ligados as diversas sincronizações na execução do *benchmark* na arquitetura CUDA, uma função de barreira paralela foi implementada baseada nas ideias apresentam em [Volkov and Demmel 2008]. Com isso, o número de chamadas de *kernel* acaba sendo reduzido, também evitando o armazenamento desnecessário de resultados parciais em memória global. Todos os dados são movidos para a GPU no início da execução da aplicação e copiados de volta ao fim do *kernel*. Nenhuma otimização específica foi feita com o intuito de melhorar o desempenho dos acesso esparsos aos dados.

[3] Experimentos iniciais podem ser encontrados na página do Projeto HPCGPU em *http://hpcgpu.codeplex.com*

5. Avaliação Experimental

Os experimentos comparam o desempenho das três categorias escolhidas da classificação *Dwarf Mine*, detalhadas na seção anterior, em três diferentes sistemas: um processador Core 2 Duo, uma GPU adicionada ao sistema do processador anterior, e um computador com 2 processadores modernos com a arquitetura Nehalem. Detalhes de configuração dos diferentes sistemas são apresentados na Tabela 1. As aplicações foram compiladas usando o compilador gcc $4.3.1$ para os sistemas *multicore* e o compilador nvcc $0.2.1221$ para os testes com a GPU. As versões paralelas nos sistemas *multicore* utilizam a biblioteca OpenMP. Todas versões foram compiladas com a opção $-O3$. O *driver* de CUDA utilizado foi a versão 2.3. Os experimentos foram executados sobre o sistema operacional Ubuntu Linux 9.04 32 bits (64 bits no sistema 2xNehalem).

	2x Nehalem	**Core 2 Duo**	**GTX280**
Modelo	Intel Xeon E5530	Intel Core 2 Duo E8500	NVIDIA GTX 280
Núcleos	2x Quad + HT	Dual	240 SP
Frequência	2.40GHz	3.16GHz	1.296GHz
Cache ou memória compartilhada	8x 256KB + 2x 8MB	6MB	30x 16KB
Memória	12GB	4GB	1GB

Tabela 1. Configuração dos sistemas paralelos.

Os resultados experimentais apresentam confiança de 99% com um erro máximo de 10% e um mínimo de 20 execuções. O desempenho dos sistemas foi medido utilizando as métricas do NPB: tempo em segundos e MOPS (milhões de operações por segundo), onde a natureza da operação considerada depende do *benchmark*. As Tabelas 2 e 3 apresentam o desempenho dos diferentes sistemas nas respectivas métricas. Os *speedups* apresentados nesta seção consideram o desempenho da execução sequencial das aplicações sobre o processador Core 2 Duo. O desempenho medido da GTX280 inclui os tempos gastos em alocações e transferências de memória para a GPU. As versões paralelas dos *benchmarks* executam sobre 2 *threads* no processador Core 2 Duo e em 16 *threads* no sistema 2xNehalem, pois essa arquitetura inclui *Hyper-Threading Technology*. O tamanho das instâncias de testes foram limitados pela quantidade de memória na GPU e pelo tempo de execução dos *kernels*, pois a GPU não teve uso dedicado (também utilizada para vídeo).

	EP W	**EP A**	**EP B**	**FT W**	**FT A**	**CG W**	**CG A**
2x Nehalem (sequencial)	4,62	36,75	147,58	0,29	5,39	0,42	2,06
2x Nehalem (16 *threads*)	0,60	3,82	14,39	0,17	1,78	0,62	1,14
Core 2 Duo (sequencial)	4,54	36,25	145,05	0,33	6,44	0,39	1,74
Core 2 Duo (2 *threads*)	2,27	18,08	72,23	0,38	6,89	0,27	1,53
GTX280	0,15	0,89	3,42	0,13	2,03	0,32	1,17

Tabela 2. Tempos de execução (em segundos) para os diferentes *benchmarks*.

5.1. MapReduce - EP

Os resultados dos diferentes sistemas em tempo e vazão para o *benchmark* EP são apresentados nas Tabelas 2 e 3. O tipo de operação considerado na métrica MOPS envolve

	EP W	EP A	EP B	FT W	FT A	CG W	CG A
2x Nehalem (sequencial)	14,53	14,62	14,89	1306,93	1323,27	1003,26	728,47
2x Nehalem (16 *threads*)	113,74	141,24	149,52	2235,45	4023,71	687,78	1323,47
Core 2 Duo (sequencial)	14,79	14,81	14,81	1122,48	1108,42	1063,30	861,05
Core 2 Duo (2 *threads*)	29,52	29,70	29,74	977,74	1034,28	1534,86	979,50
GTX280	452,09	605,07	627,52	2787,27	3523,51	1323,47	1277,28

Tabela 3. Vazão (em MOPS) para os diferentes *benchmarks*.

a quantidade de números aleatórios gerados por segundo. As instâncias de teste EP W, EP A e EP B tratam da geração de 2^{26}, 2^{29} e 2^{31} números aleatórios, respectivamente.

Como pode ser visto na Figura 5, onde os *speedups* são calculados considerando a execução sequencial do *benchmark* no processador Core 2 Duo, o melhor desempenho obtido para todas instâncias foi obtido com a GPU GTX280. As características do *benchmark* e de sua categoria - paralelismo massivo, alta intensidade aritmética e ausência de dependências - são as melhores mapeadas para a GPU. O aumento de desempenho de mais de 30% entre as instâncias EP W e EP A está ligado ao aumento de paralelismo e, consequentemente, a uma melhor utilização da GPU. Já a menor diferença entre os resultados obtidos com as instâncias EP A e EP B - com *speedups* de $40,7$ e $42,4$, respectivamente - acontece devido ao fato da GPU se encontrar completamente utilizada pela menor das duas instâncias. Assim, as diferenças em vazão estão ligadas ao aumento do tempo de processamento enquanto mantendo os mesmos custos iniciais (a mesma quantidade de memória é alocada e transferida para as duas instâncias).

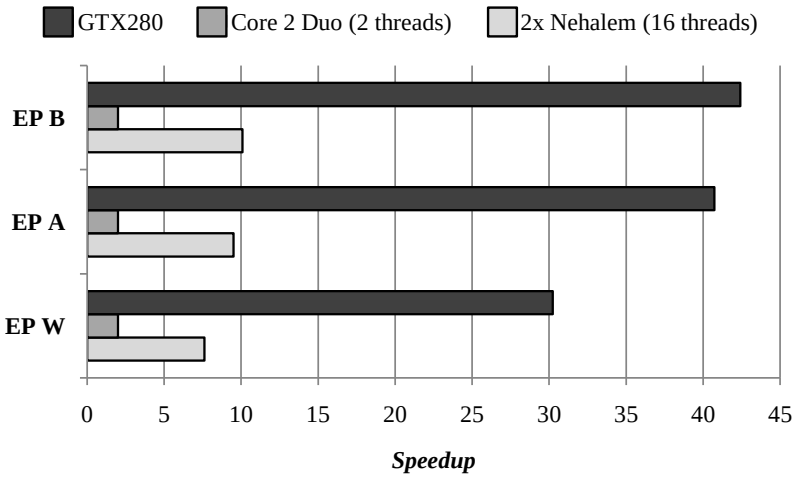


Figura 5. *Speedups* obtidos para o *benchmark* EP.

Os resultados de desempenho para o processador Core 2 Duo foram mantidos com uma vazão constante e um *speedup* de 2 ao utilizar ambos núcleos, o que indica que mesmo a menor instância de problema acabava por utilizar todos seus recursos. Isto acontece de forma diferente para o sistema 2xNehalem, o qual apresenta um aumento de desempenho com o aumento do tamanho das instâncias. Ainda assim, mesmo alcançando um *speedup* de 10, a GTX280 obteve um desempenho 4 vezes maior que o sistema 2xNehalem para todas as instâncias.

5.2. Spectral Methods - FT

As Tabelas 2 e 3 apresentam os resultados em tempo e vazão obtidos com as diferentes instâncias do *benchmark* FT. Diferentemente do *benchmark* EP, a métrica de vazão em MOPS considera operações de ponto flutuante de precisão dupla. As instâncias FT W e FT A computam 6 iterações de FFTs sobre matrizes de tamanho 128x128x32 e 256x256x128, respectivamente.

Para esse *benchmark*, o processador Core 2 Duo apresenta uma pequena diminuição de desempenho quando utilizando ambos núcleos, como visto na Figura 6. Isso é principalmente relacionado aos custos de memória e comunicação, pois esta categoria de problema apresenta diferentes padrões de acesso à memória, dependências de dados e uma intensidade aritmética reduzida. Entretanto, a diferença de desempenho se encontra dentro dos limites de 10% de erro relativo considerados.

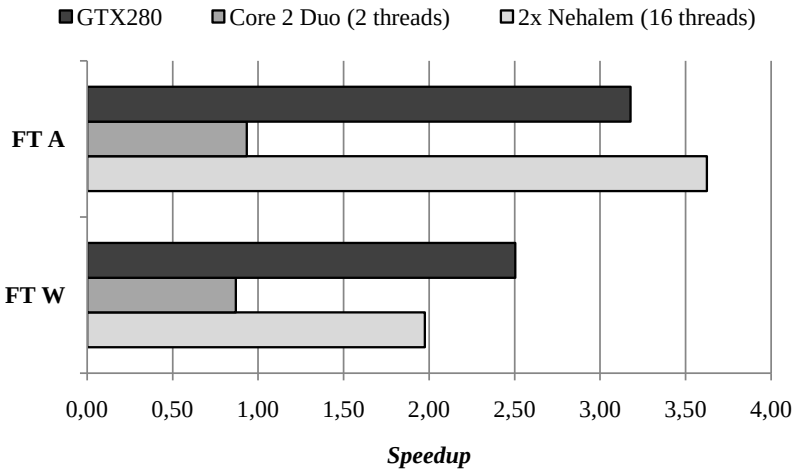


Figura 6. *Speedups* obtidos para o *benchmark* FT.

O melhor desempenho para a menor instância (FT W) foi obtido pela GTX280, com um *speedup* de $2,5$. Como a categoria de aplicação apresenta um comportamento regular, não há caminhos divergentes a serem computados sequencialmente na GPU. Mesmo com paralelismo e computações reduzidas, os ganhos de desempenho estão ligados à menor quantidade de dados da instância, a qual leva a uma quantidade reduzida de acessos não coalescidos à memória. Entretanto, este paralelismo reduzido limita o desempenho do sistema 2xNehalem, o qual obteve um *speedup* de 2.

Quando considerando a segunda instância (FT A), o melhor resultado (um *speedup* de $3,6$) foi obtido pelo sistema 2xNehalem - com sua grande memória cache compartilhada entre as *threads*- devido ao aumento de paralelismo. A GPU GTX280 apresentou um desempenho 13% menor que tal sistema para essa instância. O menor *speedup* para a GPU está ligado ao aumento na quantidade de dados, o que leva ao crescimento de requisições não coalescidas à memória e conflitos. Mesmo assim, a vazão obtida é 3 vezes maior do que a encontrada com o mesmo processador Core 2 Duo sem a utilização da GPU.

5.3. Sparse Linear Algebra - CG

Os resultados dos diferentes sistemas em tempo e vazão para o *benchmark* CG são apresentados nas Tabelas 2 e 3. Assim como o *benchmark* FT, a métrica de vazão em MOPS considera operações de ponto flutuante de precisão dupla. As instâncias CG W e CG A trabalham sobre matrizes de ordem 7000 com 637000 valores não nulos e ordem 14000 com 2198000 valores não nulos, respectivamente. Ambas executam 15 iterações do mesmo algoritmo.

Como pode ser visto na Figura 7, há uma diminuição no desempenho ao executar 16 *threads* no sistema 2xNehalem para a menor instância (CG W). Ainda, os *speedups* para os outros sistemas não ultrapassam $1,5$ com essa instância. Isto acontece principalmente devido ao pequeno paralelismo da instância, adicionado ao grande números de sincronizações existentes no *benchmark*. A GPU GTX280 ainda assim obteve um ganho de desempenho, mesmo sendo subutilizada. O melhor desempenho foi obtido com o processador Core 2 Duo, o qual possui a maior frequência de execução e menor número de *threads*.

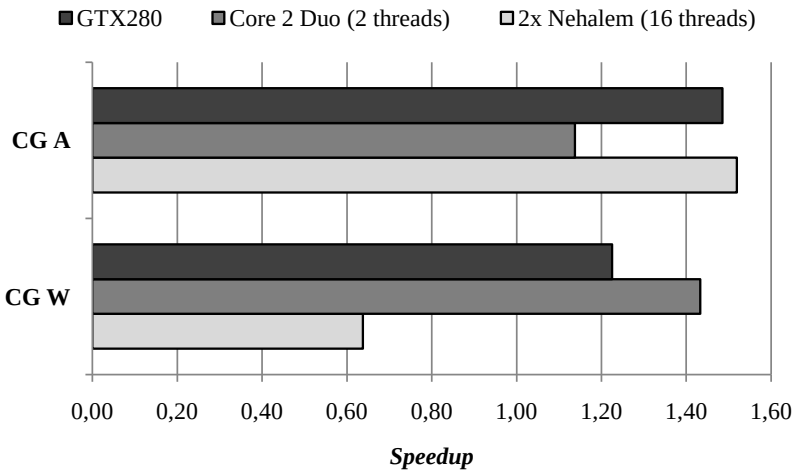


Figura 7. *Speedups* obtidos para o *benchmark* CG.

A Tabela 3 mostra uma diminuição em vazão com o aumento do tamanho da instância para a maior parte dos ambientes do experimento. Isso está ligado ao aumento na quantidade de dados, o que leva a uma maior quantidade de *cache misses* e acessos esparsos à memória. Isto acaba sendo muito custoso na GPU, pois resulta na subutilização da largura da banda da memória e em acessos separados. Ainda assim, o aumento no paralelismo fez com que a GTX280 mantivesse seu desempenho (com uma variação menor que 4%), enquanto trouxe um aumento de 2 vezes em vazão para o sistema 2xNehalem. Ambos sistemas apresentaram *speedups* similares de $1,5$.

6. Trabalhos Relacionados

O uso dos *NAS Parallel Benchmarks* pode ser visto como prática comum para a avaliação de arquiteturas e técnicas. Entretanto, todas publicações encontradas executando tais *benchmarks* ou algoritmos similares na arquitetura CUDA apresentam apenas resultados com ponto flutuante de precisão simples, devido a limitações das versões

iniciais da arquitetura [Cevahir et al. 2009, Govindaraju et al. 2008, Lee et al. 2009, Nukada and Matsuoka 2009].

Lee et al. utilizam os *benchmarks* EP e CG para avaliar seu sistema de tradução automática de OpenMP para CUDA [Lee et al. 2009]. Eles apresentam *speedups* de 300 e 4 para os *benchmarks* EP e CG, respectivamente. Cehavir et al. propuseram um algoritmo de *Conjugate Gradient* que obtém a exatidão de precisão dupla utilizando precisão simples nas GPUs combinado ao uso de precisão dupla em CPUs [Cevahir et al. 2009]. Entretanto, não há uma comparação clara entre o algoritmo proposto e os presentes algoritmos que executam em CPU.

Em [Govindaraju et al. 2008], um algoritmo para a computação de FFTs sobre múltiplas GPUs é apresentado. O proposto algoritmo alcança 300 GFlops (bilhões de operações de ponto flutuante por segundo) em GPU com precisão simples ($\approx 30\%$ do pico de desempenho teórico). Os resultados apresentaram um *speedup* de 40 sobre um algoritmo de FFT executando sobre uma CPU Quad Core moderna. Nukada e Matsuoka obtiveram resultados similares com sua biblioteca de FFT com *autotunning* [Nukada and Matsuoka 2009].

O trabalho mais próximo ao presente artigo foi desenvolvido por Dongarra et al. Tal pesquisa envolve questões iniciais sobre o uso de processadores gráficos e *Field Programmable Gate Arrays* (FPGAs) como aceleradores para aplicações de dinâmica de fluídos [Dongarra et al. 2008]. Eles apresentam resultados ligados as categorias *Dense Linear Algebra*, *Sparse Linear Algebra* e *Structured Grids* da classificação *Dwarf Mine*. Eles obtiveram um desempenho de 325 GFlops ao executar a Fatoração de Cholesky (uma aplicação da categoria *Dense Linear Algebra*) sobre uma GPU. Contudo, não foram apresentadas comparações diretas entre o uso dos diferentes aceleradores e *multicores*.

7. Conclusões

Este artigo apresentou uma avaliação da arquitetura CUDA considerando 3 categorias representativas da classificação *Dwarf Mine*. Para isso, 3 *benchmarks* paralelos do NAS foram portados para a GPU e seus desempenhos foram medidos e comparados aos obtidos com um processador Core 2 Duo e um sistema com 2 processadores modernos com a microarquitetura Nehalem.

A arquitetura CUDA mostrou seus melhores resultados com a categoria *MapReduce* (*benchmark* EP), com um *speedup* de 20 sobre a versão paralela da aplicação executando no processador Core 2 Duo e um *speedup* de 4 sobre o sistema 2xNehalem. A presença de paralelismo massivo, regularidade e independência de dados foram cruciais para a obtenção de tal desempenho.

Para a categoria *Spectral Methods* (*benchmark* FT), a GPU obteve *speedups* próximos a $2,5$ e $3,2$. Esse desempenho foi melhor do que o obtido pelo sistema 2xNehalem para a instância FT W mas 13% menor para a segunda instância. Isto aconteceu devido ao aumento na quantidade de dados, o que levou a um maior sobrecusto ligado aos acessos à memória não coalescidos.

A categoria *Sparse Linear Algebra* (*benchmark* CG) apresentou os menores ganhos. Apesar disso, o desempenho da arquitetura CUDA foi tão bom quanto o obtido pelo sistema com dois processadores Nehalem. Os custos de memória e a grande quan-

tidade de sincronizações foram responsáveis por um *speedup* máximo de $1,5$. Ademais, as limitações de desempenho dos algoritmos da categoria *Sparse Linear Algebra* são um problema conhecido [Vuduc et al. 2005].

Os resultados indicam que as três classes de aplicações consideradas podem ser mapeadas para a arquitetura CUDA com sucesso. Mesmo ao utilizar operações de ponto flutuante com precisão dupla, a inclusão da GPU no sistema trouxe aumentos de desempenho que podem ultrapassar os obtidos com arquiteturas *multicore* modernas, ou ao menos se igualar a estas. Na data da publicação deste artigo, isso resulta em uma economia de no mínimo 50% ao comparar os preços de um processador Core 2 Duo mais uma GPU moderna com o preço de um sistema com dois processadores Nehalem.

Trabalhos futuros incluem estudos sobre diferentes categorias da classificação e arquiteturas (como a nova arquitetura de GPU Fermi [NVIDIA 2009b]), uma análise energética da execução dos *dwarfs* em diferentes sistemas, e experimentos utilizando ambas CPU e GPU em paralelo. Resultados iniciais com as categorias *Dense Linear Algebra* e *Structured Grids* indicam um possível mapeamento positivo para a GPU [Tölke 2008, Volkov and Demmel 2008].

Referências

[Asanovic et al. 2009] Asanovic, K., Bodik, R., Demmel, J., Keaveny, T., Keutzer, K., Kubiatowicz, J., Morgan, N., Patterson, D., Sen, K., Wawrzynek, J., Wessel, D., and Yelick, K. (2009). A view of the parallel computing landscape. *Communications of the ACM*, 52(10):56–67.

[Asanovic et al. 2006] Asanovic, K., Catanzaro, B., Yelick, K., Bodik, R., Gebis, J., Husbands, P., Keutzer, K., Patterson, D., Plishker, W., Shalf, J., and Williams, S. (2006). The landscape of parallel computing research: A view from berkeley. *Electrical Engineering and Computer Sciences, University of California at Berkeley, Technical Report No. UCB/EECS-2006-183, December*, 18(2006-183).

[Bailey et al. 1994] Bailey, D., E., B., Barton, J., Browning, D., Carter, R., Dagum, L., Fatoohi, R., Fineberg, S., Frederickson, P., Lasinski, T., Schreiber, R., Simon, H., Venkatakrishnan, V., and Weeratunga, S. (1994). The NAS parallel benchmarks. *NASA Ames Research Center, RNR Technical Report RNR-94-007*.

[Barker et al. 2008] Barker, K. J., Davis, K., Hoisie, A., and Kerbyson, D. J. (2008). Performance Evaluation of the Nehalem Quad-Core Processor for Scientific Computing. *Parallel Processing Letters*, 18(4):453–469.

[Cevahir et al. 2009] Cevahir, A., Nukada, A., and Matsuoka, S. (2009). Fast Conjugate Gradients with Multiple GPUs. In *Proceedings of the 9th International Conference on Computational Science: Part I*, pages 893–903. Springer.

[Dongarra et al. 2008] Dongarra, J., Moore, S., Peterson, G., Tomov, S., Allred, J., Natoli, V., and Richie, D. (2008). Exploring new architectures in accelerating CFD for Air Force applications. In *Proceedings of HPCMP Users Group Conference*, pages 14–17. Citeseer.

[Duncan 1990] Duncan, R. (1990). A Survey of Parallel Computing Architectures. *Computer*, 23(2):5–16.

[Govindaraju et al. 2008] Govindaraju, N. K., Lloyd, B., Dotsenko, Y., Smith, B., and Manferdelli, J. (2008). High performance discrete Fourier transforms on graphics processors. In *2008 SC - International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12. Ieee.

[Gulati and Khatri 2008] Gulati, K. and Khatri, S. P. (2008). Towards acceleration of fault simulation using graphics processing units. *Proceedings of the 45th annual conference on Design automation - DAC '08*, pages 822–827.

[Harish and Narayanan 2007] Harish, P. and Narayanan, P. (2007). Accelerating large graph algorithms on the GPU using CUDA. *Lecture Notes in Computer Science*, 4873:197–208.

[Hou et al. 2008] Hou, Q., Zhou, K., and Guo, B. (2008). BSGP: bulk-synchronous GPU programming. In *ACM SIGGRAPH 2008 papers*, pages 1–12. ACM.

[Jin et al. 1999] Jin, H., Frumkin, M., and Yan, J. (1999). The OpenMP implementation of NAS parallel benchmarks and its performance. *NASA Ames Research Center, Technical Report NAS-99-011*.

[Lee et al. 2009] Lee, S., Min, S., and Eigenmann, R. (2009). OpenMP to GPGPU: a compiler framework for automatic translation and optimization. In *Proceedings of the 14th ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 101–110. ACM.

[Michalakes and Vachharajani 2008] Michalakes, J. and Vachharajani, M. (2008). Gpu Acceleration of Numerical Weather Prediction. *Parallel Processing Letters*, 18(04):1–8.

[Nukada and Matsuoka 2009] Nukada, A. and Matsuoka, S. (2009). Auto-tuning 3-D FFT library for CUDA GPUs. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 1–10. ACM.

[NVIDIA 2009a] NVIDIA (2009a). NVIDIA CUDA Compute Unified Device Architecture Programming Guide.

[NVIDIA 2009b] NVIDIA (2009b). NVIDIA's Next Generation CUDA Compute Architecture: Fermi.

[Panetta et al. 2009] Panetta, J., Teixeira, T., de Souza Filho, P. R., da Cunha Finho, C. A., Sotelo, D., da Motta, F. M. R., Pinheiro, S. S., Junior, I. P., Rosa, A. L. R., Monnerat, L. R., Carneiro, L. T., and de Albrecht, C. H. (2009). Accelerating Kirchhoff Migration by CPU and GPU Cooperation. *21st International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2009)*, pages 26–32.

[Tölke 2008] Tölke, J. (2008). Implementation of a Lattice Boltzmann kernel using the Compute Unified Device Architecture developed by nVIDIA. *Computing and Visualization in Science*, 13(1):29–39.

[Volkov and Demmel 2008] Volkov, V. and Demmel, J. (2008). Benchmarking GPUs to tune dense linear algebra. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, number November, pages 1–11. IEEE Press.

[Vuduc et al. 2005] Vuduc, R., Demmel, J., and Yelick, K. (2005). OSKI: A library of automatically tuned sparse matrix kernels. *Journal of Physics: Conference Series*, 16(i):521–530.