Escalonamento em Arquiteturas Heterogêneas: Um Estudo Comparativo

Guilherme Andrade, Matheus Mendonça, Gabriel Ramos, Daniel Madeira, Rafael Sachetto, Leonardo Rocha
DCOMP/UFSJ - São João del-Rei, MG , Brasil

Resumo

A necessidade do aumento da capacidade de processamento, afim de tratar grandes volumes de dados em um tempo aceitável, vem impulsionando o surgimento de novas arquiteturas computacionais compostas por diferentes unidades de processamento (UPs). Ambientes de execução vem sendo propostos com o objetivo de explorar ao máximo esses recursos, os quais disponibilizam um variado conjunto de métodos capazes de escalonar tarefas entre as diferentes UPs. Entretanto, definir qual o melhor método para cada tipo de aplicação ainda é um desafio. Assim, apresentamos nesse trabalho uma avaliação do comportamento de diferentes escalonadores, utilizando um variado conjunto de aplicações, afim de determinar quais os melhores escalonadores para cada tipo de aplicação.[1]

Keywords:: Heterogeneous Architecture, Scheduler

Author's Contact:

{guilherme.neri, matheus.mrfm, gabriel.ramos,
dmadeira, sachetto, lcrocha}@ufsj.edu.br

1 Introdução

O desenvolvimento e a evolução de novas tecnologias vem marcando uma nova era caracterizada, dentre outros fatores, pela geração maciça de dados. O constante crescimento desse volume de dados, aliado à necessidade de processamentos mais rápidos e eficientes nas diversas áreas do conhecimento, vem impulsionando avanços significativos nas arquiteturas computacionais, refletindo em sistemas de armazenamento mais eficientes, bem como o uso de diferentes tipos de unidades de processamento (UPs), os chamados sistemas híbridos. Um exemplo dessas novas arquiteturas são os computadores com vários processadores (arquitetura multicore) e GPUs (plataforma CUDA [Fatica and Luebke 2007]).

Nesse contexto, torna-se necessário que aplicações de diferentes cenários sejam capazes de explorar de forma coordenada e eficiente todas as UPs disponibilizadas, aproveitando ao máximo o potencial dessas unidades. Dessa forma, diversas bibliotecas e ambientes de execução vêm sendo propostos [Augonnet et al. 2011; Ferreira et al. ; Diamos and Yalamanchili 2008] com o objetivo fornecer diferentes métodos que permitam a utilização das diferentes UPs por meio de um escalonamento eficiente de tarefas. Por escalonamento de tarefas, entende-se a capacidade de distribuir as diferentes tarefas que compõem uma determinada aplicação entre as UPs, visando a otimização de seu desempenho global, bem como sua corretude. Um bom escalonador deve não só conhecer muito bem as características distintas de cada uma das UPs, como também as características do conjunto de tarefas a ser escalonado e suas peculiaridades. Em muitos dos ambientes propostos, a escolha do escalonador a ser utilizado na execução de uma determinada aplicação é de responsabilidade do programador. Entretanto, em muitas das vezes, falta conhecimento ao programador para inferir qual a melhor opção de escalonamento para sua aplicação.

Neste trabalho, apresentamos uma avaliação do comportamento de diferentes políticas de escalonamento, utilizando um conjunto variado de aplicações, a fim de determinar quais são os melhores escalonadores para cada tipo de aplicação, de acordo com suas características. O objetivo é fazer um levantamento de insumos que possam ser utilizados como base no desenvolvimento de um meta escalonador que seja capaz de trabalhar na ponta de ambientes de execução. A ideia desse meta escalonador é analisar as características dos conjuntos de tarefas a serem escalonadas e definir qual política de escalonamento é o mais adequado para cada conjunto.

[1]Trabalho financiado por CNPq, CAPES, FINEP, Fapemig, e INWEB.

2 Trabalhos Relacionados

Com o surgimento das arquiteturas híbridas de computadores, compostas por diferentes UPs, diferentes ambientes de execução vem sendo desenvolvidos e aprimorados para que a capacidade de processamento dessas arquiteturas seja explorada ao máximo [Augonnet et al. 2011; Ferreira et al. ; Diamos and Yalamanchili 2008; Jimenez et al. 2009], proporcionando uma série de facilidades ao programador, tais como funções e diretivas, *debugging* e geração otimizada de código baixo nível. Esses ambientes objetivam a portabilidade e integração das várias unidades de processamentos, de forma clara em um nível mais abstrato para o programador. Distribuir as diferentes tarefas que compõem uma determinada aplicação entre as diferentes UPs disponíveis, por meio de políticas de escalonamento de tarefas, é um desafio para esses ambientes de execução.

O ambiente de execução *StarPU* [Augonnet et al. 2011] foi um dos primeiros a apresentar um estudo específico nesse aspecto, oferecendo uma plataforma portável e mais transparente ao programador no que se refere ao escalonamento de tarefas em máquinas híbridas, sem a necessidade de utilização de estruturas complexas de baixo nível. Nesse ambiente, diversas políticas de escalonamento estão implementados, utilizando diferentes abordagens. Além disso, o StarPU permite ao programador definir outras políticas de escalonamento, de acordo com sua necessidade. Em [Diamos and Yalamanchili 2008] os autores apresentam o *Harmony*, um ambiente de execução bastante semelhante ao StarPU, porém sem permitir que programadores implementem suas próprias estratégias de escalonamento. Em [Jimenez et al. 2009] os autores apresentam e avaliam um ambiente para escalonamento de tarefas em ambientes híbridos. Nesse ambiente, as tarefas são assinaladas as diferentes unidades de processamento de acordo com o *speedup* relativo observado durante execuções anteriores, independentemente do conjunto de dados. Por fim, em [Ferreira et al.] é apresentado um ambiente de execução para baseado na estratégia *filter-stream* para arquiteturas paralelas e distribuídas, também abordando o impacto de escalonamentos de tarefas em clusters equipados com GPUs.

Nosso objetivo nesse trabalho não é propor nenhum novo ambiente de execução e sim, conforme mencionado anteriormente, avaliar o comportamento de diferentes políticas de escalonamento, utilizando variados tipos de aplicações, no intuito de determinar a política mais adequada para cada aplicação, de acordo com suas características. Nesse sentido, em função das peculiaridades mencionadas no parágrafo anterior, adotamos o ambiente StarPU para realizar nossas avaliações. É importante mencionar que as conclusões obtidas a partir de nossas análises poderão ser utilizadas por qualquer ambiente de execução na proposta de novas técnicas de escalonamento, ou mesmo de um meta escalonador.

3 StarPU

O StarPU [Augonnet et al. 2011] é um ambiente de execução desenvolvido pelo grupo francês *RUNTIME Inria*, que provê suporte para arquiteturas multicore híbridas. O ambiente unifica os recursos de processamento presentes, como as CPUs multicore e os aceleradores gráficos (i.e. GPUs), disponibilizando mecanismos para escalonar tarefas na arquitetura híbrida, além prover de forma transparente e portável a transferência de dados entre as UPs.

O StarPU apresenta um modelo de execução o qual aborda as tarefas independente da arquitetura que se tem como base. São definidos os chamados *codelets*, uma abstração de uma tarefa, podendo ser executado em uma das UPs. As execuções dos *codelets* nas diferentes UPs são gerenciadas pelos chamados *workers*, que são *threads* atribuídas a cada uma das UPs. Os *codelets* são lançados de forma assíncrona aos *workers*, o que possibilita ao escalonador reordenar essas tarefas afim de melhor o desempenho.

3.1 Escalonadores

No StarPU já implementa diversas estratégias de escalonamento, as quais são apresentadas e detalhadas nos tópicos abaixo. Estas estratégias de escalonamento foram tomadas como base na avaliação de diferentes aplicações, detalhadas na Seção 4.

- **Random**: essa estratégia de escalonamento distribui as tarefas aleatoriamente entre as unidades de processamento disponíveis. Baseado na performance de cada UP é realizado um sorteio que definirá qual tarefa será atribuída a cada UP.

- **Eager**: esse método utiliza uma fila central de tarefas ordenada pela ordem de chegada das mesmas. Essas tarefas são distribuídas entre as UPs a medida que as mesma se tornam ociosas, seguindo a ordem da fila onde a primeira a ser inserida é a primeira a ser atendida (FIFO). Esse método permite atribuir prioridades distintas as tarefas, sendo que as de maior prioridade são inseridas no início da fila.

- **WS (Work Stealing)**: nessa estratégia são mantidas várias filas de tarefas, uma para para cada UP. A medida que surgem novas tarefas, as mesmas são distribuídas entre as filas seguindo uma estratégia gulosa de *round robin*. Quando uma UP está com sua fila vazia, ou seja ociosa, o escalonador retira uma ou mais tarefas da fila de outra UP (a que está mais sobrecarregada) e insere naquela que estava vazia.

- **DM (Deque Model)**: assim como no *WS*, a estratégia DM matém uma fila de tarefas para cada UP. A distribuição das tarefas entre as várias filas é determinada baseada na capacidade de processamento de cada unidade, baseado no tempo de execução de tarefas anteriores. Mais especificamente,o StarPU mantém um histórico dos tempos de execuções, e dessa forma atribui uma tarefa a UP que minimize o tempo de término da tarefa. A escolha é feita baseada na UP que minimize a seguinte fórmula:

  $$mim_{Pi}(Avail(Pi) + Est_{Pi}(T))$$

  Sendo Pi a unidade processamento que está sendo avaliada, $Avail(Pi)$ a quantidade de tempo em que essa unidade de processamento terminará todas as tarefas atribuídas a ela, e Est_{Pi} a estimativa de tempo que a unidade Pi demorará para realizar a tarefa T, caso a mesma fosse inserida.

- **DMDA(Deque Model Data Aware)**: bastante similar à estratégia *DM*, uma vez que também utiliza um modelo baseado no desempenho das UPs em tarefas anteriores. Além disso, a estratégia *DMDA* considera também o tempo de transferência de dados entre a memória principal e a memória das UPS. A escolha de qual UP executará uma determinada tarefa é feita baseada naquela que minimize a seguinte fórmula:

  $$mim_{Pi}(Avail(Pi) + Est_{Pi}(T) + \sum_{data} mim_{Pj}(\tau_{j \to i}(data)))$$

  Sendo a primeira parte da soma: $Avail(Pi) + Est_{Pi}(T)$ o modelo do escalonador *DM* e a segunda parte: $\sum_{data} mim_{Pj}(\tau_{j \to i}(data))$ a penalização pela transferência de dados.

- **DMDAR (Deque Model Data Aware Ready)**: essa estratégia é baseada na *DMDA*, entretanto nas filas individuas de cada UP as tarefas são ordenadas pelo número de *buffers* de dados disponíveis ou prontos. Em outras palavras, este escalonador organizará as filas de cada unidade de processamento, colocando na frente aquelas cujo os dados necessários para execução da tarefa já estejam prontos. Para que essa estratégia funcione corretamente, é necessário conhecer o modelo que representa as relações de dependência entre as tarefas.

- **DMDAS (Deque Model Data Aware Sorted)**: bastante similar ao escalonador *DMDA*, entretanto suporta valores arbitrários de prioridades entre as tarefas nas filas de cada uma das UPs. Para que a potencialidade dessa estratégia seja explorada ao máximo, é preciso conhecer o modelo que define as prioridades entre as tarefas.

- **HEFT (Heterogeneous Earliest Finish Time)** : essa estratégia também é baseado na estratégia *DM*. Entretanto, ao invés de definir qual UP executará cada tarefa, pacotes de tarefas são analisados em conjunto e atribuídos por completo a uma determinada UP.

4 Avaliação Experimental

Com o objetivo de avaliar as políticas de escalonamento apresentados na seção anterior, escolhemos diferentes aplicações, com características variadas, gerando diferentes cargas de trabalho. Foram selecionados 3 aplicações conhecidas: Gradiente Conjugado, Fatoração Cholesky e Decomposição LU. Além disso, implementamos uma aplicação artificial que simula o comportamento de um sistema de análise de imagens multirresolução de neuroblastomas [Teodoro et al. 2009]. A seguir apresentamos uma breve descrição das cargas e apresentamos os resultados alcançados com cada uma delas. Os tempos apresentados representam a média de 10 execuções de cada carga. A máquina em que os experimentos foram realizados é uma Intel Core i7-2600 (3.40GHz) com 16Gb de RAM, com uma placa aceleradora GeForce GT 520.

4.1 Gradiente Conjugado

O Gradiente Conjugado (GC) é um método para resolução de sistemas de equações lineares que possuem sua matriz associada simétrica e positiva. É um método iterativo utilizado como alternativa aos métodos diretos, como a fatoração Cholesky, para sistemas esparsos de alta dimensionalidade. Para essa aplicação foram 5 cargas distintas variando o tamanho das matrizes (1024, 2048, 3072, 4096 e 5120). O histograma apresentado na Figura 1 mostra, para cada tamanho de entrada, os tempos de execução obtidos.

Observamos que, para estas cargas, o escalonador *WS* obteve melhor desempenho, seguido do *Eager*. O de pior desempenho foi o *Random*, seguido por *DM* e *DMDA*. É importante citar que para o GC, o StarPU não possui um modelo de desempenho mais detalhado implementado, apenas um histórico de tempo de trabalho de cada UP em execuções anteriores.

O GC pode dividido em etapas, sendo que três se destacam das demais pela maior demanda de processamento: multiplicação matriz-vetor (GEMV), soma de vetores e redução. Analisando o *profiling* gerado por cada uma das políticas, observamos que a multiplicação matriz-vetor é a que mais demanda processamento, chegando a 95% do tempo total do processo. Assim, realizar uma distribuição eficiente das tarefas nesta etapa é fator determinante para o desempenho da aplicação. Observamos que o *DM* enviou a maioria das tarefas para o *GPU WORKER*, já que o tempo de realização das tarefas nesta UP é duas vezes mais rápido que nos *CPU WORKERS*. Assim, como utilizamos apenas um *GPU WORKER* e quatro *CPU WORKERS*, este escalonador fez escolhas ruins ao deliberar muitas tarefas à GPU, deixando as CPUs com pouco trabalho. O *DMDA* se sai um pouco melhor que o *DM*, uma vez que leva em consideração o tempo de transferência de dados, entretanto também atribuiu muitas tarefas à GPU deixando as CPUs ociosas.

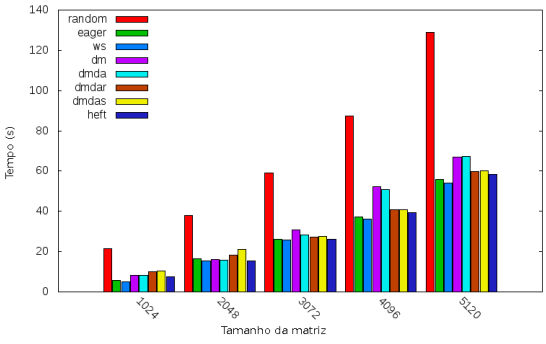

Figura 1: *Desempenho dos escalonadores para o Gradiente Conjugado*

O *DMDAR* e *DMDAS* escalonam as tarefas de forma que se os dados já estiverem disponíveis em uma UP, a tarefa é delegada à esta. Por este comportamento, diminui-se o tempo global em que as tarefas estão esperando por dados. Este comportamento foi observado nesta carga, com estes escalonadores demonstrando melhor desempenho que o *DMDA*. Por fim, o *Random*, por sua natureza aleatória, teve o pior desempenho entre os escalonadores. Por fim, analisando os dois melhores escalonadores (*Eager* e *WS*), observamos uma melhor distribuição de tarefas. Durante 80% do tempo total do processo, os *CPU WORKERS* estavam trabalhando, enquanto o *GPU WORKER* obteve 50%. Observa-se então que as CPUs sempre tiveram trabalhos delegados, mesmo com o desempenho superior da GPU. Destacamos também que o escalonador *WS* se sobressaiu por sua estratégia de desafogamento. Sempre que um *worker* fica sem trabalho, ele ”rouba”trabalho de outro.

4.2 Fatoração de Cholesky

A Fatoração de Cholesky (FG) utiliza do fato de ser possível decompor uma matriz simétrica e positiva em uma matriz triangular inferior e sua transposta. A matriz triangular é o triângulo de Cholesky da matriz original. É utilizada na resolução de problemas de ortogonalização de sinais, permitindo obter de forma matricial a ortonormalização de Gram-Schimdt. Geramos 5 cargas distintas variando o tamanho das matrizes (12288, 13312, 14336, 15360 e 16384). O desempenho de alcançado é apresentado na Figura 2.

Nesta aplicação observamos que estratégias de escalonamento baseadas em modelos (*DM*, *DMDAS*, *DMDAR* e *HEFT*) foram as mais eficientes. Este comportamento pode ser justificado uma vez que o StarPU possui modelos de desempenho específicos para o problema previamente implementados, ao contrário do GC, permitindo assim prever com maior precisão qual UP minimiza o tempo de processamento de cada tarefa dessa aplicação. As estratégias de escalonamento não baseadas em modelos de desempenho, como *RANDOM*, *EAGER* e *WS*, apresentaram um desempenho inferior aos demais escalonadores. Por meio da análise do *profilling*, observamos que as tarefas relacionadas a FG podem ser agrupadas em dois conjuntos distintos, um grupo que executar de forma mais rápida da GPU e outro grupo que executa mais rápido na CPU. Assim, os escalonadores que não conhecem essas peculiaridades apriori, apresentaram o pior desempenho, enquanto os que conhecem, obtiveram excelentes resultados.

Analisando as estratégias de melhor desempenho, temos que a *DM* considera apenas qual UP será capaz de minimizar o tempo de término da tarefa, sem se preocupar com a transferência nem dependência de dados. Assim, a unidade que minimiza o término de uma tarefa acaba sendo sobrecarregada, já que as tarefas possuem tempos subestimados. Em contrapartida, a estratégia *DMDAR* melhora o desempenho apresentado pelo escalonador *DM*, já que considera o tempo de transferência de dados entre as UPs. Com isso, a estimativa de tempo de término de cada tarefa é mais próxima da realidade, resultando em um melhor balanceamento na distribuição de tarefas, reduzindo o tempo de ociosidade das unidades menos utilizadas. Esta estimativa de tempo de disponibilização de um dado é também utilizada na estratégia *DMDAS*. O ganho em desempenho obtido por esta estratégia vem do fato de que as

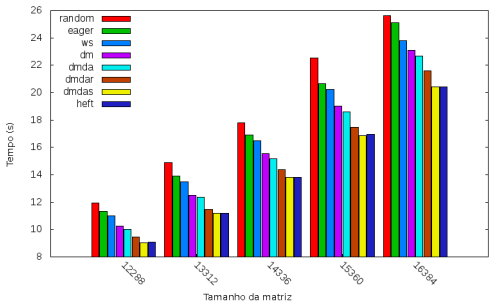

Figura 2: *Desempenho dos escalonadores para a Fatoração Cholesky*

tarefas com maior prioridade são as que mais criam dependências de dados. Executando estas tarefas primeiro, minimizando os casos de ociosidade por dependência de dados. A estratégia *HEFT* possui um bom desempenho (similar ao *DMDAS*) pois envia pacotes de tarefas, onde cada pacote possui as tarefas de maior prioridade juntamente com as tarefas que dependem desses dados. Sendo executadas em pacotes, a dependência de dados é também reduzida.

4.3 Decomposição LU

A Decomposição LU (DLU) é uma forma de fatoração de matrizes como o produto de uma matriz triangular inferior e uma matriz triangular superior. Para essa aplicação foram geradas 5 cargas distintas variando o tamanho das matrizes (1024, 2048, 3072, 4096 e 5120) e o desempenho de alcançado é apresentado na Figura 3.

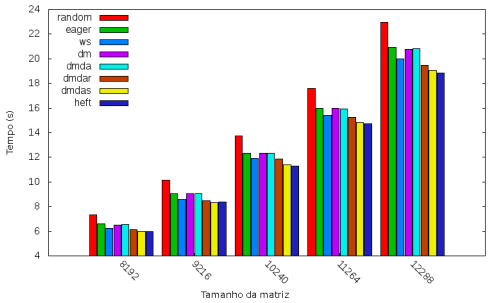

Figura 3: *Desempenho dos escalonadores para a Decomposição LU*

Podemos observar que a diferença entre os tempos de execução para cada política de escalonamento (com exceção do *RANDOM*) é pequena. Esse fato se justifica em função da alta dependência de dados entre tarefas que compõem essa aplicação. Assim como a aplicação FG, o StarPU também possui modelos de desempenho específicos para o problema previamente implementados, o que dá aos escalonadores baseados em modelo de desempenho uma ligeira vantagem na execução. Para o caso da DLU, a escolha equivocada de qual UP deve executar uma tarefa pode ainda ser mais desastroso em função da alta dependência de dados entre as tarefas.

O escalonador *Eager* delega tarefas em UPs cujo desempenho não é o melhor, assim como ocorre com a carga Cholesky. Por exemplo, ao enviar uma tarefa *GPU bound* para a CPU, esta terá seu término atrasado, provocando dependência de dados em outras tarefas. Essa dependência acaba por atrasar a execução final da aplicação. A estratégia *WS* não é capaz de minimizar consideravelmente o tempo total pelo mesmo motivo do que ocorre na carga Cholesky, onde a má escolha da unidade de processamento caracteriza o problema.

As estratégias baseadas em modelo apresentam melhores desempenhos, com exceção da *DM* e *DMDA*, que não visam resolver as dependências. Essa atitude acaba por causar problemas de dependência de dados e consequentemente espera ociosa pelo dado correto a ser processado, assim como ocorre no Cholesky. Esse problema é contornado pela estratégia *DMDAR*, que visa executar primeiro tarefas prontas, cujas dependências foram todas resolvidas, diminuindo o problema de espera ociosa. Já na estratégia *DMDAS* há prioridade maior para tarefas que geram mais dependências de dados, o que evita que novas tarefas com dependências sejam geradas. A estratégia *HEFT* possui um bom comportamento pelo mesmo motivo do que ocorre na aplicação Cholesky.

4.4 Aplicação Artificial

Implementamos uma aplicação artificial que simula o comportamento do Sistema de Análise de Imagens Multirresolução de Neuroblastomas [Teodoro et al. 2009]. A ideia geral desse problema é, a partir de pequenos *tiles* de uma imagem maior, aplicar a análise de neuroblastoma (NBIA). Quando não é possível detectar o tumor nos pequenos *tiles* da imagem, aumenta-se a resolução do *tile* (de 32 para 64, de 64 para 128, até 512) e a análise é executada novamente.

Nossa aplicação artificial, baseado nesse problema, simula cada *tile* como sendo uma matriz quadrada $n \times n$, sendo esse n variando entre 32, 64, 128, 256 e 512. Simulamos a análise da imagem percorrendo a matriz m vezes realizando um cálculo aleatório em cada posição, o que constitui a nossa tarefa. Ao fim de cada tarefa, definimos de forma aleatória se é necessário aumentar a resolução do *tile*, lançando ou não uma nova tarefa com as dimensões da matriz duplicadas. Por exemplo, se uma tarefa de tamanho 32×32, ao final de sua execução, sorteia um valor intervalo de probabilidade, uma nova tarefa de tamanho 64×64 é criada, e assim sucessivamente. Este processo simula o comportamento do processo de NBIA encontrar o tumor ou não em *tiles* multirresolução. Para esta aplicação, a carga foi definida inicialmente com 200000 matrizes de tamanho 32 e com probabilidade de gerar uma matriz maior em cada tarefa variando entre 0, 4, 8, 12, 16 e 20%. A Figura 4 apresenta o histograma que sumariza os resultados obtidos.

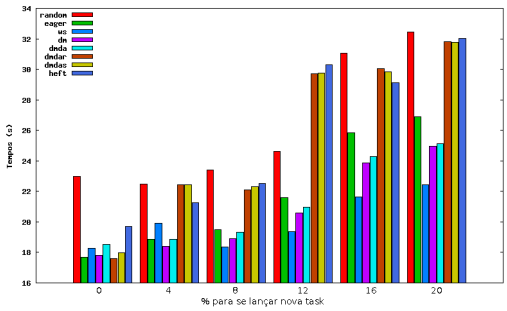

Figura 4: *Desempenho dos escalonadores para o Aplicação Artificial*

Nessa aplicação, quanto maior a probabilidade, maior a chance de criação de novas tarefas maiores. Com a probabilidade de 0%, só se executam as 200000 tarefas iniciais. Para estas tarefas, o desempenho da GPU e da CPU é muito parecido. Já em tarefas maiores, a GPU pode executar até 100 vezes mais rápido que a CPU. A partir dessa observação e analisando os resultados, temos que a estratégia de escalonamento *WS* se mostrou mais eficiente a partir da probabilidade 8%. Como a distribuição inicial das tarefa entre os *workers* é feita de forma igual (*round robin*), e pelo fato das tarefas maiores ocuparem uma CPU por muito tempo, temos que a GPU finaliza suas tarefas muito mais rápido que a CPU. Isso possibilita à GPU "roubar"as tarefas das CPUs (que estão ocupadas). Como mencionado anteriormente, a chance de criarmos tarefas maiores aumenta com a probabilidade, o que, consequentemente, explica o bom desempenho do *WS* a medida que essa probabilidade aumenta.

Para reforçar o argumento apresentado no parágrafo anterior, na Tabela 1 apresentamos como fica a distribuição de tarefas entre as diversas UPs disponíveis a partir da política de escalonamento *WS*. Nessa tabela apresentamos os resultados referentes a uma probabilidade de gerar uma matriz maior de 20%. Como podemos perceber, temos que as tarefas grandes (256 e 512) são muitas vezes executadas pela GPU e poucas ou nenhuma vez pela CPU. Utilizando porcentagens pequenas (0% e 4%), poucas tarefas maiores são geradas, com isso, apenas tarefas pequenas são transferidas para a GPU, o que não gera ganhos efetivos.

N	GPU	CPU 1	CPU 2	CPU 3
32	119344	26861	26908	19610
64	35499	936	897	925
128	7253	10	13	11
256	1385	0	1	1
512	254	0	0	0

Tabela 1: *Distribuição de Tarefas por tamanho entre as UPs (WS)*

Outra estratégia que obteve bons resultados foi o *DM*, que da mesma forma que o *WS* delegou menos tarefas grandes para a CPU, conforme mostra a tabela 2. Porém, é observado que as CPUs executaram mais tarefas grandes com o *DM* do que com o *WS*, o que explica o desempenho inferior em relação à estratégia *WS*.

Por fim, a estratégia *HEFT* não obteve um bom desempenho. Mesmo a GPU ficando com quase todas as tarefas grandes, ao atribuir pacotes de tarefas à CPU, o escalonador fez com que a CPU ficasse com muitas tarefas de tamanho 64 (quase 6x mais que nos outros escalonadores). Apesar de pequenas, o excesso destas tarefas impactou o tempo total do processo. A Tabela 3 exibe estes resultados.

N	GPU	CPU 1	CPU 2	CPU 3
32	141811	19322	19447	19420
64	34518	1219	1165	1165
128	6825	147	157	157
256	1262	49	39	39
512	196	20	22	22

Tabela 2: *Distribuição de Tarefas por tamanho entre as UPs (DM)*

N	GPU	CPU 1	CPU 2	CPU 3
32	141327	19498	19565	19610
64	20311	5928	5969	5999
128	7085	73	78	62
256	1412	1	0	1
512	265	0	0	0

Tabela 3: *Distribuição de Tarefas por tamanho entre as UPs (HEFT)*

5 Conclusão

O crescimento de computadores com unidades de processamento heterogêneas impulsionou o aparecimento de ambientes de execução capazes de escalonar tarefas eficientemente entre estas UPs. Analisamos neste trabalho um destes ambientes, o *StarPU*. Neste ambiente é necessária a escolha, pelo usuário, de um entre vários escalonadores de tarefas para uma aplicação. Na nossa análise, mostramos que, para uma mesma aplicação, a diferença de desempenho entre os escalonadores é grande de onde concluímos que uma má escolha do escalonador a ser utilizado pode impactar significativamente o tempo de execução de uma aplicação.

Como trabalho futuro, pretendemos propor um meta-escalonador, que será uma ferramenta de pré-processamento. Este meta-escalonador trabalhará na ponta do *StarPU*, analisando a aplicação de entrada e definindo o melhor escalonador para ela, tornando-se uma ferramenta atraente para o usuário, já que retira do mesmo a tarefa de definir o escalonador a ser utilizado.

Referências

AUGONNET, C., THIBAULT, S., NAMYST, R., AND WACRENIER, P. 2011. Starpu: A unified platform for task scheduling on heterogeneous multicore architectures. *Concurrency and Computation: Practice and Experience and Special Issue: Euro Par 2009 23* (Feb.), 187–198.

DIAMOS, G. F., AND YALAMANCHILI, S. 2008. Harmony: an execution model and runtime for heterogeneous many core systems. *HPDC 08: Proceedings of the 17th international symposium on High performance distributed computing*, 197–200.

FATICA, M., AND LUEBKE, D., 2007. High performance computing with CUDA. Supercomputing 2007 tutorial. In Supercomputing 2007 tutorial notes, November.

FERREIRA, R., MEIRA JR, W., GUEDES, D., DRUMMOND, L., COUTINHO, B., TEODORO, G., TAVARES, T., ARAUJO, R., AND FERREIRA, G. Anthill: A scalable run-time environment for data mining applications. In *SBAC-PAD 2005*, 159–166.

JIMENEZ, V. J., LLU'VILANOVA, GELADO, I., GIL, M., FURSIN, G., AND NAVARRO, N. 2009. Predictive runtime code scheduling for heterogeneous architectures. *HiPEAC*, 19–33.

TEODORO, G., SACHETTO, R., SERTEL, O., GURCAN, M., MEIRA, W., CATALYUREK, U., AND FERREIRA, R. 2009. Coordinating the use of gpu and cpu for improving performance of compute intensive applications. In *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*, IEEE, 1–10.