Escalonamento Dinâmico Eficiente em Arquiteturas Híbridas

Guilherme Andrade and Renato Ferreira
Departament of Computer Science
Universidade Federal de Minas Gerais
{*gnandrade,renato*}*@dcc.ufmg.br*

Gabriel Ramos, Rafael Sachetto,
Daniel Madeira and Leonardo Rocha
Departament of Computer Science
Universidade Federal de São João Del Rei
{*gramos,sachetto,dmadeira,lcrocha*}*@ufsj.edu.br*

***Resumo*—**
Aplicações que lidam com grandes quantidades de dados em tempo aceitável vem impulsionando o desenvolvimento de novas arquiteturas computacionais, compostas por diferentes unidades de processamento (UP). Ambientes de execução vem sendo propostos para explorar esses recursos, oferecendo métodos capazes de escalonar tarefas entre diferentes UPs. Embora a maioria das aplicações sejam heterogêneas (tarefas com características distintas), as técnicas atuais focam nessas características de forma isolada, gerando execuções ineficientes. Neste trabalho apresentamos duas novas estratégias de escalonamento, combinando diferentes estratégias, capazes de generalizar em diferentes cenários, sendo até 20% mais eficientes que as técnicas atuais.

I. Introdução

O desenvolvimento 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 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 placas gráficas (GPUs [1] com arquitetura CUDA [2]).

Diante desse novo contexto, torna-se necessário que as aplicações de diferentes cenários sejam capazes de explorar de forma coordenada e eficiente todas as UPs disponibilizadas, aproveitando ao máximo suas capacidades de processamento. Dessa forma, diversas bibliotecas e ambientes de execução vem sendo propostos [3], [4], [5], [6], [7], [8] com o intuito de fornecer um conjunto de métodos capazes de tornar a utilização das diferentes UPs transparente aos desenvolvedores. Dentre os principais métodos que são fornecidos por esses ambientes, podemos destacar os chamados escalonadores de tarefa. Esses escalonadores visam distribuir, adequadamente, as diferentes tarefas que compõem uma determinada aplicação entre as UPs disponíveis.

Existem basicamente duas categorias de escalonadores de tarefa, os estáticos e os dinâmicos. Em ambas as categorias, é feita uma avaliação das características de cada tarefa e de cada UP. A partir dessas avaliações, o escalonador associa cada tarefa a UP que será capaz de executá-la no menor tempo possível. Os escalonadores estáticos [9], [10] realizam avaliações a partir de um pré-processamento, utilizando informações globais da aplicação, tais como a relação de dependência entre as tarefas que compõem um determinada aplicação. Por outro lado, os escalonadores dinâmicos realizam essas análises em tempo de execução, associando dinamicamente tarefas as UPs a partir de uma visão limitada da execução de toda aplicação, sendo assim, um cenário mais desafiador.

Existem na literatura diversas propostas de escalonadores dinâmicos, entretanto, focados em conjuntos específicos de aplicações. Enquanto algumas dessas propostas são voltadas para aplicações compostas de tarefas com intensa troca de dados entre a memória principal e a memória dos dispositivos aceleradores [11], outras conseguem trabalhar melhor com aplicações compostas por tarefas de intenso processamento e poucas transferências de dados [12], [13]. Entretanto, a grande maioria das aplicações são compostas por diferentes tipos tarefas, alternando entre momentos de intensa transferência de dados e intenso processamento, também denominadas de aplicações heterogêneas. Esse fato gera dois problemas distintos: (1) a escolha do escalonador fica sob a responsabilidade do desenvolvedor, que precisa analisar minunciosamente as características de sua aplicação, bem como as opções de escalonadores existentes; (2) ainda que o desenvolvedor conheça detalhadamente as tarefas que compõem sua aplicação, as propostas de escalonadores existentes não são gerais o suficientes para serem instanciadas de forma eficiente em cenários de aplicações distintos, ou seja, dependendo da aplicação pode não haver uma boa opção de escalonador.

Nesse trabalho, propomos duas novas estratégias de escalonamento, ambas construídas combinando diferentes abordagens de escalonadores dinâmicos já propostos na literatura. Basicamente, os escalonadores propostos combinam dois tipos de abordagens distintas. Um primeiro tipo, mais simples, em que a distribuição das tarefas entre as UPs é feita de forma aleatória, mas permitindo que uma determinada UP, quando ociosa, execute tarefas inicialmente assinaladas a outra UP [13]. Um segundo tipo, mais elaborado, onde modelos de predição de tempo de execução de uma tarefa, baseados no histórico de tarefas anteriores, são utilizados para definir de forma mais adequada qual UP ficará responsável por cada tarefa [14]. Para avaliar nossas estratégias, elaboramos diversos conjuntos de experimentos, compostos por aplicações sintéticas, nas quais as características das aplicações foram apropria-

damente variadas. Além disso, avaliamos também nossa proposta em cenários reais, compostos por aplicações relacionadas à álgebra linear (Fatoração de Cholesky, Gradiente Conjugado e Decomposição LU). Comparamos nossas propostas com os escalonadores nos quais elas foram baseadas, todos implementados no StarPU [3], um importante framework de execução paralela em arquiteturas híbridas. Analisando os resultados, temos que nossas propostas foram capazes de generalizar em diferentes cenários, alcançando sempre os melhores resultados. Para alguns cenários, o ganho de eficiência de nossas propostas foi de até 20% em relação às demais estratégias.

II. Trabalhos Relacionados

A partir do surgimento das arquiteturas híbridas de computadores, compostas por diferentes UPs, diferentes ambientes de execução foram sendo desenvolvidos e aprimorados para que a capacidade de processamento dessas arquiteturas pudessem ser exploradas ao máximo [3], [5], [4]. Esses ambientes proporcionam uma série de facilidades ao desenvolvedor, tais como funções e diretivas, *debugging*, geração otimizada de código baixo nível, entre outros. O objetivo desses ambientes é prover a portabilidade e integração das várias unidades de processamento, de forma clara em um nível mais abstrato para o desenvolvedor. Em [3] é apresentando o *StarPU*, um ambiente de execução que oferece uma uma plataforma portável e transparente ao desenvolvedor, onde o desenvolvimento das aplicações é feito sem a necessidade de utilização de estruturas complexas de baixo nível. Além disso, o StarPU permite ao desenvolvedores alterar os métodos existentes, bem como incorporar outros métodos no ambiente. Em [5] os autores apresentam o *Harmony*, um ambiente de execução bastante semelhante ao *StarPU*, porém sem permitir que os desenvolvedores incorporarem qualquer outro método no ambiente. Por fim, em [4] é apresentado um ambiente de execução baseado na estratégia *filter-stream* para arquiteturas paralelas e distribuídas.

Para todos esses ambientes, dentre os diversos métodos fornecidos, os que merecem destaques são aqueles relacionados à distribuição das tarefas que compõem uma determinada aplicação entre as diferentes UPs disponíveis, os chamados escalonadores dinâmicos. Conforme mencionado anteriormente, esses escalonadores realizam uma avaliação das características de cada tarefa e de cada UP, co-ocorrendo com a execução da aplicação, e a partir dessa avaliação, definem qual a UP mais adequada para executar cada tarefa. Esses escalonadores, mesmo com uma visão limitada do conjunto de tarefas que compõem uma aplicação, devem ser capaz de minimizar fatores que comprometem a boa distribuição das tarefas, tais como a sobrecarga de tarefas em uma UP, a má adequação de uma tarefa à uma UP, e até mesmo os efeitos da transferência de dados entre as memórias das UPs.

Existem na literatura diversas propostas de escalonadores dinâmicos ([13], [15], [14], [16], [8]), muitas delas utilizadas pelos ambientes de execução acima mencionados. Em [13] os autores focam em uma estratégia que visa minimizar a sobrecarga de trabalho entre as unidades de processamento. Nessa estratégia a distribuição das tarefas entre as UPs é feita de forma aleatória, entretanto quando uma determinada UP se torna ociosa, é permitido que a mesma execute tarefas inicialmente assinaladas a outra UP, por meio de uma abordagem de "roubo de tarefas". Em [15] os autores apresentam a estratégia de escalonamento em arquiteturas híbridas denominado Dynamic Weighted Round Robin (*DWRR*), a qual é focada em aplicações de intensa necessidade de processamento. A política *DWRR* atribui um peso de execução para cada tarefa em cada UP e esse peso é utilizado para ordenar a fila de execução das diferentes UPs. Em [14], os autores apresentam uma estratégia baseada em modelos de predição de tempo de execução. Nessa estratégia é feita uma previsão do tempo de execução para cada tarefa, bem como do tempo total da fila de execução de cada UP. A partir dessa previsão, as tarefas são associadas as UPs menos ocupadas, provendo dessa forma, um balanceamento de carga equilibrado. Por fim, em [16] é apresentada uma proposta semelhante, onde tais modelos de execução são baseados nos históricos de execução passadas. Apesar de todas essas estratégias apresentarem bons resultados nos cenários nos quais se comprometem a trabalhar, não são capazes de generalizar para diferentes cenários de aplicação.

Nesse trabalho apresentamos duas estratégias de escalonamento dinâmico que visam trabalhar de forma eficiente em diferentes cenários de aplicação. Essas estratégias foram desenvolvidas combinando as características de diferentes técnicas já existentes. O ambiente de execução escolhido para implementar e avaliar essas estratégias foi o StarPU, por dois motivos: (1) já possui implementado diversas das estratégias mencionadas anteriormente [11]; e (2) permite o desenvolvimento e a incorporação de outros métodos de escalonamento de tarefas.

III. Descrição da Proposta

Nessa seção apresentamos nossas duas propostas de escalonadores dinâmicos eficientes para arquiteturas híbridas. Conforme mencionado, essas propostas foram desenvolvidas combinando diferentes estratégias de escalonamento existentes. Dessa forma, dividimos essa seção em duas subseções. Na primeira detalhamos as estratégias utilizadas como base para o presente trabalho, e seguida, na outra subseção, descrevemos como essas estratégias foram combinadas em nossas abordagens.

A. Estratégias de Escalonamento Originais

Após um detalhado estudo preliminar das políticas de escalonamento implementadas no ambiente de execução StarPU [17], selecionamos as três estratégias que apresentaram os melhores resultados quando avaliadas em diferentes cenários de aplicação. São elas: *Work Stealing*, *Deque Model* e *Deque Model Data Aware*. Descrevemos nos tópicos a seguir cada uma delas.

- **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

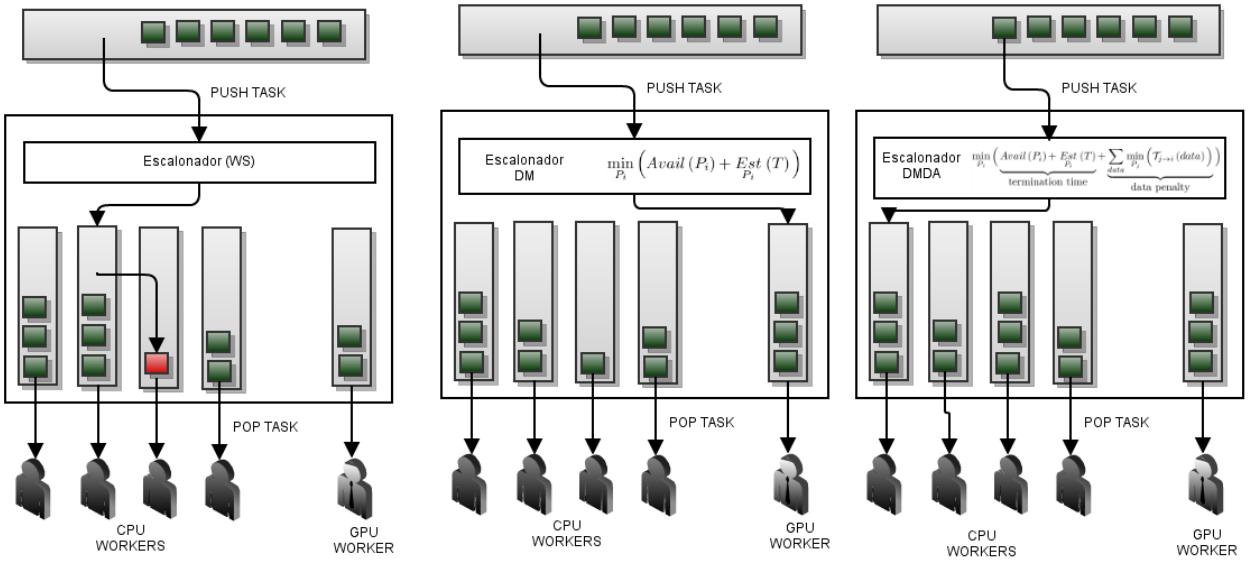


Figura 1: Estratégias de Escalonamento do StarPU: WS, DM e DMDA

distribuídas entre as filas seguindo uma estratégia gulosa de *round robin*. Quando uma UP está com sua fila vazia, ou seja ociosa, o escalonador verifica qual UP está mais sobrecarregada e retira uma ou mais tarefas da fila de dessa UP e insere na fila de execução da UP ociosa.

- **DM (Deque Model)**: assim como no *WS*, a estratégia DM mantém uma fila de tarefas para cada UP. A distribuição das tarefas entre as várias filas é determinada de acordo com a capacidade de processamento de cada unidade, baseado no tempo de execução de tarefas anteriores. Mais especificamente, o escalonador mantém um histórico dos tempos de execução, 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 a Equação 1

$$mim_{Pi}(Avail(Pi) + Est_{Pi}(T)) \quad (1)$$

  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 seja atribuída a ela.
- **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 Equação 2

$$min_{Pi}(Avail(Pi)+Est_{Pi}(T)+\sum_{data} min_{Pj}(\tau_{j\to i}(data))) \quad (2)$$

  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.

A estratégia *WS* é uma estratégia bastante simples mas que no entanto funciona muito bem em cenários em que o comportamento das tarefas que compõem a aplicação são bastante variados, impossibilitando de se realizar uma boa modelagem matemática de previsão do tempo de execução. Entretanto, essa estratégia pode fazer atribuições equivocadas de tarefas a UPs, penalizando significativamente o tempo total de execução. Por outro lado, as estratégias *DM* e *DMDA* são ótimas opções quando os comportamentos das tarefas de uma aplicação são bem conhecidos, por meio de execuções anteriores. Tal modelo tende a aproximar bem o tempo de execução da tarefa na UP através de execuções passadas, porém é uma estratégia que demanda uma calibragem durante a execução da aplicação, agregando um *overhead* que pode não ser compensado pela boa atribuição de uma tarefa em uma UP. Uma ilustração dessas estratégias é apresentada na figura 1

B. Novas Estratégias de Escalonamento

Baseados nos escalonadores acima descritos, nessa seção descrevemos nossas duas novas abordagens, as quais combinam os aspectos principais dos escalonadores de modelos *DM* e *DMDA*, com a característica de "roubo de tarefas"do escalonador *WS*. Nos tópicos abaixo descrevemos cada um dos escalonadores proposto.

- **Deque Model Work Stealing (DMWS):** Este escalonador combina as características dos escalonadores *DM* e *WS*. Basicamente o escalonador mantém uma fila de tarefas para cada UP e, similarmente ao *DM*, a distribuição das tarefas entre as várias filas é baseada na UP que minimize a Equação 1. Similarmente ao escalonador *WS*, quando uma UP está ociosa, o escalonador retira uma ou

mais tarefas da fila de outra UP (a que está mais sobrecarregada) e insere na UP ociosa.

- **Deque Model Data Aware Work Stealing (DMDAWS):** Essa política de escalonamento combina característica dos escalonadores *DMDA* e *WS*. Cada UP possui uma fila de tarefas e a distribuição das mesmas nessas filas se dá pela minimização da Equação 2, a qual agrega a informação de quantidade de dados que será transferido entre as UPs para a execução desta tarefa. Por fim, assim no *WS*, quando uma UP fica ociosa, o escalonador atribui a ela tarefas de outras UPs.

Dessa forma, os dois escalonadores propostos agregam características que tentam resolver os principais problemas do escalonamento dinâmico de tarefas em arquiteturas híbridas. Enquanto o escalonado *DMWS* se preocupa com a boa adequação das tarefas nas UPs e também com a sobrecarga de trabalho. O escalonador *DMDAWS* vai além e considera também a informação da quantidade de dados utilizados que deverão ser transferidos, evitando assim perdas de desempenho por conta do *overhead* da transferência. Na figura 2 ilustramos nossas estratégias.

Nossa expectativa é que, dessa forma, as nossas abordagens sejam capazes de generalizar melhor, se adaptando a cenários mais variados. Para comprovar nossas hipóteses, avaliamos nossas estratégias em diversos cenários de aplicação, contrastando os resultados obtidos por elas com os resultados das estratégias originais. Na próxima seção apresentamos, as cargas de trabalho utilizadas em nossas avaliações, e na seção seguinte os resultados obtidos nessas avaliações.

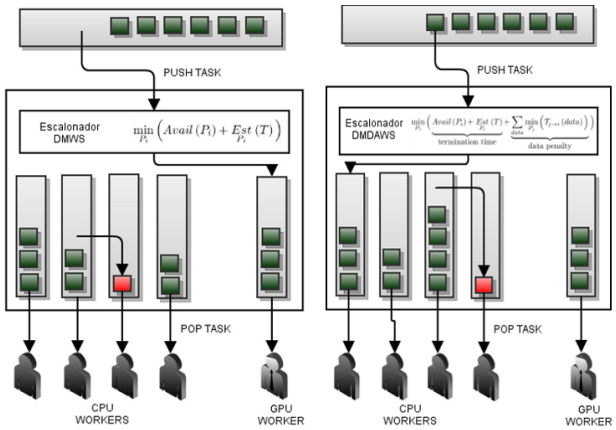


Figura 2: Estratégias de Escalonamento Propostas: DMWS e DMDAWS

IV. Cargas de Trabalho

Nessa seção descrevemos as cargas de trabalho que serão utilizadas para avaliarmos nossas propostas de escalonadores dinâmicos. Conforme veremos nas seções abaixo, essas cargas são compostas por aplicações sintéticas, nas quais variamos de forma controlada e diversificada as características das tarefas que são geradas, e também por aplicações reais, sem nenhum tipo de controle. O objetivo é alcançar um conjunto grande de diferentes aplicações, e assim analisar o comportamento dos escalonadores nos mais diferentes cenários.

A. Aplicações Sintéticas

Em uma arquitetura híbrida como a utilizada nesse trabalho, compostas por várias *CPUs* e *GPUS*, um fator determinante para a qualidade de um escalonador está associado à razão entre os tempos de execução de uma tarefa em *CPU* e *GPU*, que denominamos de *SpeedUp relativo*. Enquanto para algumas tarefas o tempo de execução em CPU é inferior ao tempo de execução em GPU, sendo portanto recomendadas a serem executadas em *CPU*, para outras, o tempo de GPU é que é inferior, sendo portanto recomendadas a serem executadas em *GPU*. Um escolha inadequada do escalonador pode impactar fortemente no tempo de execução total da aplicação. Outro fator importante de ser considerado pelos escalonadores é o tempo de transferência de dados entre memória principal e a memória da *GPU*. Normalmente, esse tempo é alto e aumenta à medida que a quantidade de dados a ser transferida aumenta. Ao associar uma tarefa que manipula uma grande quantidade de dados a *GPU*, é necessário que o escalonador avalie também a quantidade de operações que será aplicada a esses dados. Essa quantidade precisa ser grande o suficiente para compensar o tempo de transferência de dados, caso contrário o *overhead* causado pela transferência dos dados impactará no tempo total de execução da aplicação. Dessa forma, geramos diversos conjuntos de aplicações sintéticas, nas quais as característica acima mencionadas foram diversificadas.

Primeiramente, dividimos nossas cargas em três grupos principais:

- **Grupo 1:** as aplicações desse grupo são compostas, em sua grande maioria, por tarefas mais apropriadas de serem executadas em *CPU*.
- **Grupo 2:** as aplicações desse grupo são compostas, em sua grande maioria, por tarefas mais apropriadas de serem executadas em *GPU*.
- **Grupo 3:** as aplicações desse grupo são compostas, de forma equilibrada, por tarefas que são apropriadas tanto para CPU quanto para GPU.

Para uma tarefa ser considerada melhor na CPU do que na GPU, seu tempo de execução deve ser pelo menos 2 vezes menor que o da GPU (Grupo 1). De maneira análoga, para uma tarefa ser melhor em GPU (Grupo 2), seu tempo de execução deve ser pelo menos 2 vezes menor na GPU se comparada com o tempo na CPU. Para as tarefas do Grupo 3, essa diferença de tempo é pequena.

Para cada grupo de carga, variamos também as diferenças nos tempos de execução de cada tarefa entre CPU e GPU. Para isso, criamos 2 classes distintas:

- **Classe 1:** Compostas por tarefas cujo *SpeedUp relativo* é alto, superior a 5 vezes. São tarefas as quais é de extrema importância associar a UP adequada para a sua execução, uma vez que a execução na UP errada, fará com que ocorra uma significativa perda de desempenho em relação ao tempo total de execução da aplicação.

- **Classe 2:** Compostas por tarefas cujo *SpeedUp relativo* é baixo, inferior a 5 vezes. Tarefas desse tipo, mesmo que associadas à unidades de processamento inadequadas, poderão acarretar perdas menos significativas de desempenho no tempo total de execução da aplicação.

Combinando os três grupos e duas classes mencionadas acima, criamos seis cargas de trabalhos distintas, conforme apresentado na Tabela I.

Tabela I: Cargas de Trabalho.

Nome	Grupo	Classe	Nome	Grupo	Classe
Workload 1	1	1	Workload 4	1	2
Workload 2	2	1	Workload 5	2	2
Workload 3	3	1	Workload 6	3	2

Por fim, para cada uma das seis cargas de trabalho apresentadas na Tabela I, simulamos tarefas com diferentes quantidades de dados a serem manipulados. Mais especificamente, as tarefas geradas podem assumir quatro tipos distintos de tamanho de dados: sem transferência (i.e. 1KB), ou seja, tarefas cujo o custo de transferência de memória pode ser desprezado; pequenas (50MB); médias (150MB) e grandes (250MB).

B. Aplicações reais

Com o objetivo de avaliarmos ainda mais detalhadamente as estratégias de escalonamento propostas nesse trabalho, selecionamos três aplicações reais, todas relacionadas à álgebra linear, conforme apresentado abaixo:

- **Fatoração Cholesky:** A Fatoração de Cholesky (FC) utiliza do fato de ser possível decompor uma matriz simétrica e definida positiva em uma matriz triangular inferior e sua transposta. Essa matriz triangular é o triângulo de Cholesky da matriz original. Este método pode ser utilizado na resolução de problemas de ortogonalização de sinais, resolução de sistemas lineares, entre outros. Para a avaliação dos escalonadores, utilizamos 5 cargas distintas variando o tamanho das matrizes utilizadas (12.288 x 12.288, 13.312 x 13.312, 14.336 x 14.336, 15.360 x 15.360 e 16.384 x 16.384).
- **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, que também pode ser utilizada na solução de sistemas lineares. Para essa aplicação foram geradas 5 cargas distintas variando o tamanho das matrizes (8.192 x 8.192, 9.126 x 9.126,10.240 x 10.240,11.264 x 11.264 e 12.288 x 12.288).
- **Gradiente Conjugado:** O Gradiente Conjugado (GC) é um método iterativo para resolução de sistemas de equações lineares que possuem uma matriz associada simétrica e definida positiva. O CG é utilizado como alternativa aos métodos diretos, como a fatoração Cholesky, para resolução sistemas esparsos de alta dimensionalidade. Utilizamos para essa aplicação 5 cargas distintas, variando o tamanho da matriz associada ao sistema linear a ser resolvido (1.024 x 1.024, 2.048 x 2.048, 3.072 x 3.072, 4.096 x 4.096 e 5.120 x 5.120).

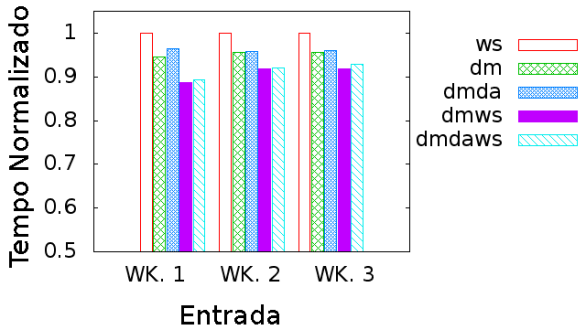
V. Avaliação Experimental

Nessa seção apresentamos os resultados das avaliação das estratégias de escalonamento propostas nesse trabalho (*DMWS* e *DMDAWS*), contrastando com os resultados obtidos por outras estratégias de escalonamento mais conhecidas (*WS*, *DM* e *DMDA*). Conforme mencionamos na seção III, todos esses escalonadores foram implementados dentro do ambiente de execução StarPU e as cargas utilizadas estão descritas na seção IV. Todos os experimentos foram realizados em uma máquina equipada com uma CPU Intel Core i7-2600 (3.40GHz), 16Gb de RAM e GPU GeForce GT520 com 1Gb de RAM. Para cada cenário de avaliação, foram geradas 500 tarefas e os tempos obtidos são referentes a uma média de 10 execuções e são apresentados de forma normalizada em função do pior tempo de execução dentre os escalonadores avaliados.

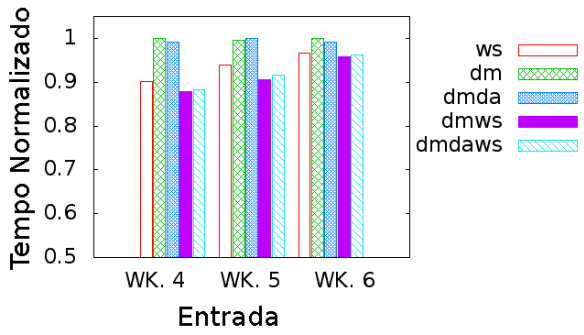
A. Cenários sem transferência

O primeiro conjunto de experimentos realizados são referentes a execução das seis cargas de trabalho sintéticas descritas na Tabela I, entretanto gerando apenas tarefas onde o tempo de transferência de dados entre memória principal e GPU são desprezíveis (i.e. tarefas que manipulam apenas 1KB de memória). Nesse primeiro conjunto de experimentos nosso objetivo é avaliar o comportamento dos escalonadores considerando apenas tarefas de diferentes *SpeedUp relativo*, isolando dos desafios relacionados à transferência de dados entre as memórias principais e de GPU. Os resultados alcançados nesse primeiro cenário são apresentados nos gráficos da Figura 3.

Observando os gráficos, vamos considerar, a princípio, apenas o comportamento dos escalonadores *WS*, *DM* e *DMDA* para ambos os cenários. Para o cenário onde há uma incidência maior de tarefas cujo o *SpeedUp relativo* é baixo, temos que o escalonador *WS* apresentou resultados superiores aos do *DM* e *DMDA*. Isso acontece pelo fato da política de "roubar" as tarefas de outras UPs, mesmo que não muito adequadas para UP de destino, se torna uma boa opção, uma vez que é pequena a diferença entre os tempos de execução das tarefa em CPU e GPU. Além disso, o *overhead* gerado para se executar os modelos de previsão para adequar uma tarefa à melhor UP não irá compensar a minimização do tempo de execução da tarefa. Por outro lado, no cenário em que há uma incidência maior de tarefas cujo o *SpeedUp relativo* é alto, os escalonadores baseados em modelos de previsão de tempo obtiveram os melhores resultados. Nesse caso, uma associação adequada de uma tarefa a uma UP é fundamental, uma vez que uma associação inadequada pode tornar o tempo total de execução muito alto. Sendo assim, o *overhead* gerado no processo de execução dos modelos de previsão são compensados pela atribuição correta de uma tarefa a uma UP. Por outro lado, a política de "roubo" de tarefas implementadas pela estratégia *WS* torna-se ineficiente.

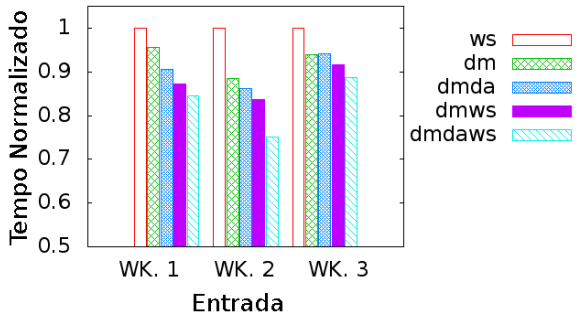


(a) SpeedUp Relativo Alto

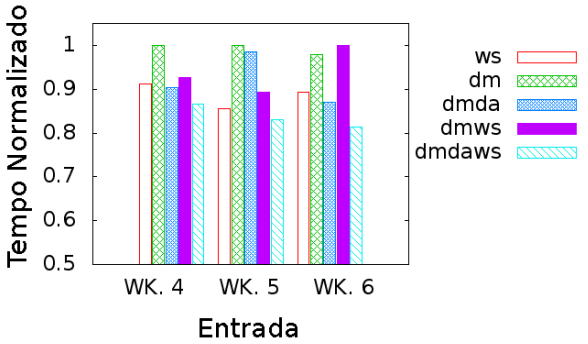


(b) SpeedUp Relativo Baixo

Figura 3: Avaliação dos Escalonadores em Cenários sem Transferência de Dados



(a) SpeedUp Relativo Alto



(b) SpeedUp Relativo Baixo

Figura 4: Avaliação dos Escalonadores em Cenários com Transferência de Dados

Observando o comportamento dos escalonadores *DMWS* e *DMDAWS* propostos nesse trabalho, temos que ambos alcançaram os melhores resultados em ambos cenários, sendo até 10% mais eficientes. Esse resultado mostra que, a combinação entre estratégias de escalonamento que utilizam modelos de predição com estratégias mais simples, como "rouba" de tarefas, pode ser bem vantajoso, se adaptando melhor a diferentes cenários de aplicações.

B. *Cenários com transferência*

Em nosso segundo conjunto de experimentos, consideramos novamente cada uma das seis cargas apresentadas na seção IV-A, entretanto gerando agora tarefas que manipulam diferentes quantidades de dados (50MB, 150MB e 250MB). Para esse cenário, é importante que o escalonador seja capaz de prever adequadamente o tempo necessário para realizar as transferências de dados entre a memória principal e a memória da GPU para verificar se realmente é vantajoso associar a tarefa a GPU. Os resultados alcançados nesse segundo cenário são apresentados nos gráficos da Figura 4.

Analisando os gráficos, observamos um comportamento muito parecido com aqueles apresentados na seção anterior. Novamente, temos que em cenários com maioria de tarefas com SpeedUp relativo baixo, o escalonador *WS* se apresenta como uma boa alternativa, enquanto que em cenários onde a maioria das tarefas possui SpeedUp relativo alto, os escalonadores baseados em modelos de predição são mais eficientes. Entretanto, diferentemente do cenário sem transferência de dados, nesse novo cenário observamos que os escalonadores baseados em modelos de predição que consideram o custo de transferência de dados apresentaram os melhores resultados. Nesse caso, o *overhead* gerado para analisar o custo das transferências de dados é compensado por uma associação correta de uma tarefa a uma UP adequada, alcançando bons resultados. Por fim, observamos que, mais uma vez, os escalonadores propostos nesse trabalho (o *DMWS* e o *DMDAWS*) apresentaram os melhores resultados em quase todos os cenários avaliados, com grande destaque ao *DMDAWS*, que em alguns casos foi até 20% mais eficiente.

C. *Cargas Reais*

Por fim, nosso último conjunto de experimentos diz respeito a avaliação dos escalonadores em cenários de

cargas reais apresentados na seção IV-B: Fatoração de Cholesky (FC), Decomposição LU (DLU) e Gradiente Conjugado (GC). O objetivo desse último conjunto de experimentos é avaliar o desempenho dos escalonadores propostos nesse trabalho em cenários reais, onde as características das tarefas geradas por cada aplicação não são conhecidos em detalhes como nas cargas sintéticas geradas. Nos gráficos da Figura 5 apresentamos os resultados alcançados por cada escalonador.

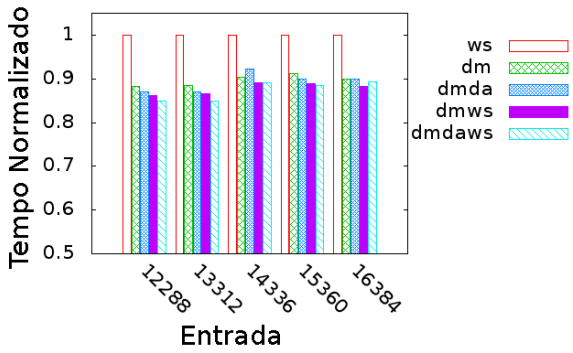
Observando os resultados, temos que o escalonador *WS* alcançou o melhor desempenho para a aplicação de Gradiente Conjugado. Por outro lado, o escalonador *DMDA* se mostrou a melhor estratégia para as aplicações Decomposição LU e Fatoração de Cholesky. Esse fato corrobora mais uma vez nossa hipótese de que, diferentes escalonadores podem apresentar diferentes desempenhos, dependendo das características do cenário de aplicação. Entretanto, em todos os cenários os escalonadores propostos nesse trabalho apresentaram os melhores resultados, mostrando mais uma vez que nossa abordagem de combinar diferentes estratégias de escalonamento é uma boa alternativa e genérica o suficiente para ser aplicada em diversos cenários.

VI. Conclusões e Trabalhos Futuros

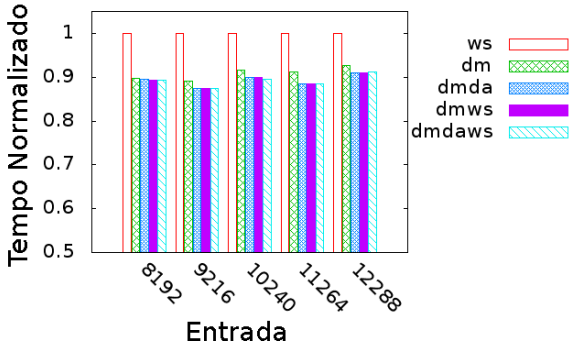
O aumento do número de computadores com unidades de processamento heterogêneas impulsionou o surgimento de diversos ambientes de execução que, cujo objetivo é explorar de forma coordenada e eficiente todas as UPs disponibilizadas e, aproveitando ao máximo o potencial dessas unidades. Uma das principais características desses ambientes é a disponibilização de métodos de escalonamento de tarefas, visando a utilização plena de todas as UPs disponíveis. Nesse trabalho, focamos no escalonamento dinâmico de tarefas, onde o escalonamento das tarefas coocorre com a submissão e execução das mesmas nas unidades de processamento disponíveis.

Apesar de existirem diversas propostas de escalonadores dinâmicos de tarefas, a grande maioria deles é focada em nichos específicos de aplicação, o que faz com que os mesmos não sejam gerais o suficientes para serem utilizados em diferentes cenários. Baseado nessa observação, propomos nesse trabalho duas novas estratégias de escalonamento dinâmico, construídas combinando as características de diferentes estratégias já existentes. Dessa forma, o objetivo é conseguir um melhor desempenho em uma gama maior de aplicações. Basicamente, nossas estratégias combinam características de escalonadores baseados em "roubo de tarefas"com estratégia baseadas em modelos de predição de tempo de execução, considerando (DMWS) ou não (DMDAWS) o tempo de transferência de dados entre memória principal e memória das UPs.

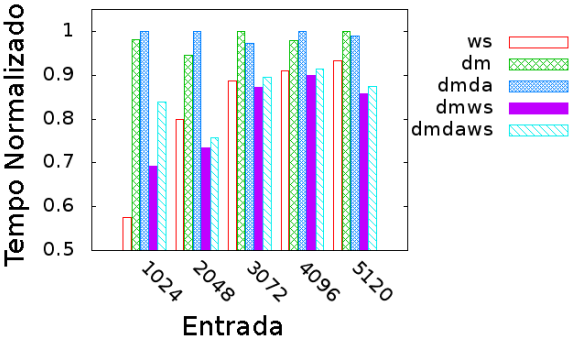
Avaliamos as políticas de escalonamento propostas em diferentes cenários. No primeiro, foram geradas diferentes cargas sintéticas, nas quais variamos as características das tarefas de cada aplicação, considerando os diversos aspectos relacionados a escalonadores de tarefas (transferência de dados, computacionalmente intensivas etc.).



(a) Fatoração de Cholesky



(b) Decomposição LU



(c) Gradiente Conjugado

Figura 5: Avaliação dos Escalonadores em Aplicações Reais

No segundo cenário consideramos três aplicações reais distintas, todas relacionadas a álgebra linear (Fatoração Cholesky, Gradiente Conjugado e Decomposição LU). Contrastamos os resultados obtidos por nossas abordagens com os diferentes escalonadores nos quais elas foram baseadas (*WS*, *DM* e *DMDA*). Nossos resultados mostram que, enquanto em alguns cenários as abordagens baseadas somente em modelos de predição se mostram mais adequa-

das, em outros cenários a abordagem baseada apenas no "roubo de tarefas"se mostrou mais apropriada. Entretanto, em todos os cenários avaliados nossas abordagens se mostraram a mais eficientes, ou seja, foram capazes de generalizar um bom funcionamento em diferentes cenários de aplicação. Em alguns cenários, nossas abordagens foram até 20% mais eficientes que as abordagens originais.

Dessa forma, mostramos nesse trabalho que nossas abordagens de combinar estratégias diferentes de escalonamento se mostrou uma boa opção, capazes de serem instanciadas em diferentes cenários de forma eficiente. Como trabalhos futuros, nosso objetivo é avaliar a combinação de outras estratégias de escalonamento existentes. Além disso, pretendemos avaliar estratégias de escalonamento em diferentes níveis, considerando aglomerados de computadores, sendo cada um deles composto por arquiteturas híbridas. O objetivo é escalonar, em um primeiro nível, uma aplicação inteira dentre os diversos computadores disponíveis, e em um segundo nível, escalonar as diferentes tarefas que compõem uma aplicação entre as unidades de processamento disponíveis em cada computador.

AGRADECIMENTO

Esse trabalho foi parcialmente financiado por CNPq, CAPES, FINEP, Fapemig, e INWEB.

REFERÊNCIAS

[1] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A. E. Lefohn, , and T. J. Purcell., "A survey of general-purpose computation on graphics hardware. computer graphics," vol. 26, pp. 80–113, Mar. 2007.

[2] M. Fatica and D. Luebke, "High performance computing with CUDA," Supercomputing 2007 tutorial. In Supercomputing 2007 tutorial notes, November 2007.

[3] C. Augonnet, S. Thibault, R. Namyst, and P. Wacrenier, "Starpu: A unified platform for task scheduling on heterogeneous multicore architectures," *Concurrency and Computation: Practice and Experience and Special Issue: Euro-Par 2009*, vol. 23, pp. 187–198.

[4] R. Ferreira, W. Meira Jr, D. Guedes, L. Drummond, B. Coutinho, G. Teodoro, T. Tavares, R. Araujo, and G. Ferreira, "Anthill: A scalable run-time environment for data mining applications," in *SBAC-PAD 2005*, pp. 159–166.

[5] G. F. Diamos and S. Yalamanchili, "Harmony: an execution model and runtime for heterogeneous many core systems," *HPDC 08: Proceedings of the 17th international symposium on High performance distributed computing*, pp. 197–200, 2008.

[6] P. Bellens, J. M. Perez, R. M. Badia, and J. Labarta, "Cellss: a programming model for the cell be architecture," *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, p. 86, 2006.

[7] C. Augonnet, S. Thibault, R. Namyst, and M. Nijhuis, "Exploiting theCell/BE architecture with the StarPU unified runtime system," Jul. 2009.

[8] G. Teodoro, T. D. R. Hartley, Ü. V. Çatalyürek, and R. Ferreira, "Optimizing dataflow applications on heterogeneous environments," *Cluster Computing*, vol. 15, no. 2, pp. 125–144, 2012.

[9] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Comput. Surv.*, vol. 31, no. 4, pp. 406–471, Dec. 1999.

[10] D. G. Amalarethinam and G. J. Mary, "Article: A new dag based dynamic task scheduling algorithm (dytas) for multiprocessor systems," *International Journal of Computer Applications*, vol. 19, no. 8, pp. 24–28, April 2011, published by Foundation of Computer Science.

[11] C. Augonnet, S. Thibault, and R. Namyst, "StarPU: a Runtime System for Scheduling Tasks over Accelerator-Based Multicore Machines," INRIA, Rapport de recherche RR-7240, Mar. 2010. [Online]. Available: http://hal.inria.fr/inria-00467677

[12] W. Wang and G. Zeng, "Trusted dynamic scheduling for large-scale parallel distributed systems," in *Parallel Processing Workshops (ICPPW), 2011 40th International Conference on*, sept. 2011, pp. 137 –144.

[13] R. D. Blumofe and C. E. Leiserson, "Scheduling multithreaded computations by work stealing," *J. ACM*, vol. 46, no. 5, pp. 720–748, Sep. 1999. [Online]. Available: http://doi.acm.org/10.1145/324133.324234

[14] W. Smith, V. Taylor, and I. Foster, "Using run-time predictions to estimate queue wait times and improve scheduler performance," in *Scheduling Strategies for Parallel Processing*. Springer-Verlag, 1999, pp. 202–219.

[15] G. Teodoro, R. Sachetto, O. Sertel, M. Gurcan, W. M. Jr., U. Catalyurek, , and R. Ferreira, "Coordinating the use of gpu and cpu for improving performance of compute intensive applications," *IEEE International Conference on Cluster Computing*, Sep. 2009.

[16] A. Jooya, A. Baniasadi, and M. Analoui, "History-aware, resource-based dynamic scheduling for heterogeneous multi-core processors," *Computers Digital Techniques, IET*, vol. 5, no. 4, pp. 254 –262, july 2011.

[17] G. Andrade., M. Mendonça, R. Sachetto., D. Madeira, and L. Rocha, "Escalonamento em arquiteturas heterogeneas: Um estudo comparativo," *XXXII Congresso da Sociedade Brasileira de Computação, Brasil*, 2012.