ESCALONAMENTO ADAPTATIVO DE TAREFAS EM ARQUITETURAS HÍBRIDAS

GUILHERME NERI ANDRADE

ESCALONAMENTO ADAPTATIVO DE TAREFAS EM ARQUITETURAS HÍBRIDAS

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Renato Antônio Celso Ferreira
Coorientador: Leonardo Chaves Dutra da Rocha

Belo Horizonte
Outubro de 2014

Andrade, Guilherme Neri
A553e Escalonamento adaptativo de tarefas em arquiteturas híbridas / Guilherme Neri Andrade. — Belo Horizonte, 2014
xxii, 84 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais. Departamento Ciência da Computação.

Orientador: Renato Antônio Celso Ferreira.
Coorientador: Leonardo Chaves Dutra da Rocha.

1. Computação - Teses. 2. Programação paralela (Computação) - Teses. 3. Computação paralela.
4. Escalonamento de processos - Tese.
I Orientador. II Coorientador. Título.

CDU 519.6*31(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

ESCALONAMENTO ADAPTATIVO DE TAREFAS EM ARQUITETURAS HÍBRIDAS

GUILHERME NERI ANDRADE

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. RENATO ANTÔNIO CELSO FERREIRA - Orientador
Departamento de Ciência da Computação - UFMG

PROF. LEONARDO CHAVES DUTRA DA ROCHA - COORIENTADOR
Departamento de Ciência da Computação - UFSJ

PROF. GEORGE LUIZ MEDEIROS TEODORO
Departamento de Ciência da Computação - UnB

PROF. WAGNER MEIRA JÚNIOR
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 03 de outubro de 2014.

À minha mãe Laisa
Ao meu pai Hébert
À toda minha família e amigos.

Agradecimentos

Interessante notar que as expectativas que eu tinha antes de iniciar o mestrado foram superadas, que pessoas se fizeram mais presentes e outras simplesmente surgiram em minha vida, que meu crescimento como pessoa e profissional é motivo de orgulho para mim mesmo e que acima de qualquer coisa hoje estou muito bem e feliz com os passos e escolhas feitas. E essa felicididade, que é evidente nesse pequeno texto que escrevo, devo à muitas pessoas que serei sempre grato.

Agradeço à Deus (ou o nome que você achar melhor para àquela energia positiva que de uma forma ou de outra conduz bons acontecimentos em nossas vidas), por em momentos de extremo estresse, impaciencia e ansiedade, ter acalmado meus ânimos, seja através de uma musica agradável ou por palavras de pessoas amigas ou por qualquer outra coisa que tenha me proporcionado paciência e calma.

Agradeço também a toda minha familia. Em especial à minha mãe, Laísa, por ser um exemplo de determinação, de força e sabedoria, por incentivar sempre os meus estudos, me motivando em cada escolha e acima de qualquer coisa me dando apoio e suporte. Ao meu pai, Hébert, além de todo o carinho e dedicação, por ser meu exemplo de caráter e por me guiar no caminho do bem. Meus pais são pessoas determinantes em toda minha caminhada, mesmo que em algumas vezes distantes, se fizeram presentes em todos os momentos.

Aos meu amigos agradeço imensamente pelos momentos de descanso e divertimento, sem vocês a trajetória seria com toda certeza mais árdua. Aos amigos de república, agradeço especialmente pelo convivio, pelo respeito e amizade.

Agradeço também a minha namorada Kerlei pelo companheirismo, amizade, carinho, dedição e pelo amor. Com ela pude compartilhar minhas ansiedades, aflições e medos, e em reposta obtive o mais sincero aconchego de um abraço. Agradeço por estar comigo nesses momentos difíceis e por estar comigo agora em uma conquista!

Por fim, agradeço ao meu orientador, Renato, e ao meu co-orientador e amigo, Leo, por contribuirem imensamente com meu crescimento como profissional.

"Só se pode alcançar um grande êxito quando nos mantemos fiéis a nós mesmos."
(Friedrich Nietzsche)

Resumo

O constante crescimento do volume de dados aliado à necessidade de processamentos mais eficientes nas diversas áreas do conhecimento, vem impulsionando avanços significativos nas arquiteturas computacionais. Presenciamos a popularização do uso de eficientes coprocessadores, como por exemplo GPUs e MICs, em máquinas equipadas com poderosos processadores *multicore*, constituindo assim arquiteturas híbridas. Diante desse novo contexto, torna-se necessário que aplicações de diferentes cenários sejam capazes de explorar de forma coordenada e eficiente todas as unidades de processamento (processadores e coprocessadores) disponibilizadas em uma arquitetura híbrida, aproveitando ao máximo sua capacidade de processamento. Ambientes de execução vem sendo propostos para explorar eficientemente esses recursos, oferecendo métodos capazes de escalonar tarefas entre diferentes unidades de processamento (UPs). Esses escalonadores determinam qual UP é a mais adequada para executar uma determinada tarefa, por meio da análise de características das tarefas e dos recursos disponíveis. Em nosso trabalho, portanto, apresentamos um estudo sobre escalonamento dinâmico de tarefas em arquiteturas híbridas, propondo novas estratégias de escalonamento eficientes e explorando diferentes cenários de aplicações e sistemas híbridos com variadas unidades de processamento.

Palavras-chave: Computação paralela, Arquiteturas híbridas, Escalonamento de Tarefas.

Abstract

The constant growing volume of data coupled with the need for more processing power in differents areas of knowledge, has stimulated the emergence of new computing architectures. We witnessed the popularization of the use of efficient coprocessors, such as GPUs and MICs for machines equipped with powerful multicore processors, constituting hybrid architectures. In this new context, it becomes necessary applications of differents scenarios are able to explore coordinated and efficient all processing units (processors and coprocessors) available in a hybrid architecture, taking advantage the most of their processing capabilities. Runtime environments have been proposed in order to exploit these resource as much as possible by offering a variety of methods for dynamically scheduling tasks on different process units (PUs). These schedulers determine which PU is better suited for executing each task, based on caracteristics of tasks and avaiable resources. In our study, therefore, present a study on dynamic task scheduling in hybrid architectures, proposing new strategies for efficient scheduling and exploring different application scenarios and hybrid systems with various processing units.

Keywords: Parallel Computing, Hybrid Architectures, Task Scheduling.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

Um conjunto de aplicações que surgiram nos últimos anos, denominado Ciência de Dados, caracterizam-se por extrair conhecimento significativo a partir de grandes conjuntos de dados coletados e armazenados em diversas áreas da ciência e engenharia. Nesse contexto, aplicações consistem de vários estágios de extração de dados e transformações através de uma larga variedade de algoritmos complexos, os quais, associados ao grande volume de dados a serem tratados, necessitam de uma alta demanda de processamento.

Análises de imagens é uma aplicação que ilustra bem esse contexto. Mais especificamente podemos citar o estudo de imagens de microscopia, como por exemplo análise de patologias em imagens, usado para investigar a morfologia de câncer cerebral. Nessa aplicação um conjunto de imagens de dimensões expressivas, $120Kx120K$ pixels, são processados em várias etapas, como por exemplo normalizações, segmentações, extração de características (*feature computation*) e classificações, com o objetivo de se obter alguma observação em cima do dado inicial. Nesses estudos, várias execuções de diferentes operações são fundamentais, varia-se os parâmetros e algoritmos, para explorar a diversidade de aspectos do dado e do processamento. Porém, como dito, para se atingir um tempo de execução computacionalmente viável, é necessário um alto poder de processamento.

Dessa forma, o constante crescimento do volume de dados, aliado à necessidade de processamentos mais eficientes nas diversas áreas do conhecimento, vem impulsionando avanços significativos nas arquiteturas computacionais. Grandes redes de computadores, com máquinas de *hardware* diversificado e em sua maioria heterogêneo (diferentes unidades de processamento para computação de alto desempenho), possuindo processadores de computação intensiva e coprocessadores (www.top500.org) associados a computação massivamente paralela de dados, são utilizados com frequên-

cia. Tais coprocessadores consistem de arquiteturas massivamente paralelas, focando o alto desempenho de operações aritiméticas e de controle de fluxos. A conclusão é que esses coprocessadores podem proporcionar considerável poder computacional se determinados critérios são obedecidos pela aplicação e seus dados. Podemos destacar dois coprocessadores comumente utilizados: placas gráficas (GPUs [Owens et al., 2007] por meio da interface de programação CUDA [NVIDIA, 2011; Fatica & Luebke, 2007]) e o Intel® Xeon Phi(MIC).

Diante desse novo contexto, torna-se necessário que as aplicações de diferentes cenários e com diferentes organizações de dados, sejam capazes de explorar de forma coordenada e eficiente todas as unidades de processamento (processadores e coprocessadores) disponibilizadas, aproveitando ao máximo suas capacidades de processamento. Sendo assim, a pesquisa no campo de sistemas de computadores vem ganhando considerável incentivo, na qual, ambientes de execução (*runtime systems*) e arcabouços de programação (*frameworks*) [He et al., 2008; Linderman et al., 2008; Luk et al., 2009; Rossbach et al., 2011; Augonnet et al., 2012; Bosilca et al., 2011; Gautier et al., 2013; Bueno et al., 2012] vem sendo propostos, com o intuito de fornecer um conjunto de métodos capazes de tornar a utilização das diferentes unidades de processamento transparente aos desenvolvedores. Tais arcabouços de programação tornam possível a execução de aplicações em *hardware* heterogêneo. Dentre esses ambientes de execução podemos destacar o StarPU [Augonnet et al., 2009] e o Extreme DataCutter [Teodoro et al., 2013]. O primeiro, é utilizado para o processamento de aplicações em apenas uma máquina híbrida, através da distribuição de tarefas entre as unidades de processamento disponíveis, Por outro lado, no Extreme DataCutter as aplicações são representadas por fluxos de operações hierárquicos (*dataflows* hierárquicos) e cada estágio dessa aplicação pode ser replicado e assinalado para a execução em diferentes máquinhas de um sistema híbrido distribuído.

Nesses ambientes de execução, algumas ferramentas para a execução coordenada de tarefas são fornecidos para explorar de forma eficiente todos os recursos de uma máquina híbrida. Dentre essas ferramentas, podemos destacar os chamados escalonadores de tarefa. Esses escalonadores visam distribuir adequadamente as diferentes tarefas que compõem uma determinada aplicação entre as unidades de processamento disponíveis, visando a otimização de seu desempenho global, garantindo também sua corretude. Em muitos casos, o escalonamento de tarefas é considerado um problema NP-Completo [Ullman, 1975].

De forma geral, o problema do escalonamento de tarefas pode ser organizado em grupos, os quais os escalonadores são separados de acordo com suas características de tomada de decisões, características da aplicação e até mesmo a organização dos

recursos disponíveis para a execução. No contexto desse trabalho, escalonamento de tarefas de aplicações em uma única máquina equipada com diferentes unidades de processamento, podemos ter escalonadores estáticos e escalonadores dinâmicos [Casavant & Kuhl, 1988]. Em ambas as categorias, utiliza-se diferentes estratégias para tentar melhor adequar uma tarefa à unidade de processamento que será capaz de executá-la satisfazendo uma determinada função objetivo, como por exemplo a minimização do tempo de execução ou até mesmo a redução do consumo energético de um recurso. Os escalonadores estáticos [Kwok & Ahmad, 1999; Amalarethinam & Mary, 2011] realizam avaliações e associações de tarefas antes da execução da aplicação, utilizando informações globais, 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 unidades de processamento a partir de uma visão limitada da execução de toda aplicação.

Dessa forma, nesse trabalho será abordado o problema de escalonamento dinâmico de tarefas em arquiteturas híbridas. Nesse contexto, podemos classificar esse nicho de escalonadores observando a forma com que ocorre a distribuição e organização das tarefas entre os recursos disponíveis, tendo assim, escalonadores dinâmicos não-adaptativos e escalonadores dinâmicos adaptativos. O foco adaptativo de um escalonador dinâmico é caracterizado pela realocação das tarefas entre as unidades de processamento após uma associação inicial. Basicamente, nesse grupo temos escalonadores que realizam uma associação inicial de tarefas bem simples, como por exemplo uma distribuição igualitária circular (nomeada *round robin*) , e adaptativamente reorganiza essas tarefas entre as filas das unidades de processamento, por meio de “roubo" [Blumofe & Leiserson, 1999] ou replicação. Por sua vez, escalonadores dinâmicos não-adaptativos, se beneficiam de estratégias diversas, como predição de tempo ou desempenho (*Speedup*), para realizarem uma boa adequação inicial e definitiva das tarefas entre os recursos disponíveis. Nesse grupo de escalonadores não existe realocação de tarefas entre os recursos após a associação inicial.

Existem na literatura diversas propostas de escalonadores dinâmicos, entretanto, apresentando características puramente adaptativas, ou não-adaptativas. Em diversos contextos existe uma variação do desempenho desses escalonadores já que, para determinados conjuntos de trabalhos, características específicas das estratégias de escalonamento sobressaem e realizam melhor a associação das tarefas entre as unidades de processamento. 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.

As características de tomada de decisões dos escalonadores dinâmicos se baseiam, contudo, na variação do desempenho das tarefas entre as diferentes unidades de processamento. Comumente utiliza-se uma predição de tempo de execução ou predição do desempenho, por meio de históricos de execuções passadas, para que ocorra uma correta adequação das tarefas aos recursos disponíveis. Adicionalmente, essas estratégias de predição são suscetíveis à presença de erros, tornando-as estratégias pouco acuradas dependendo da métrica utilizada ou da forma com que a predição é realizada. É possível também encontrarmos propostas de escalonadores que inserem outros componentes nessa tomada de decisão, como por exemplo a quantidade de dados manipulados pela tarefa [Augonnet et al., 2009] e até mesmo o consumo energético [Li et al., 2013] da execução em um determinado recurso. Essas características, bem como a presença de variadas unidades de processamento com diferentes arquiteturas, impactam diretamente no desempenho dos escalonadores em diversos contextos de aplicações, criando-se assim um conjunto de fatores a serem estudados e explorados, visando a compreensão e possivelmente a proposta de novas estratégias de escalonamento dinâmico para arquiteturas híbridas.

1.1 Objetivo

Tendo em vista as características apontadas anteriormente, o objetivo deste trabalho é apresentar um estudo sobre escalonamento dinâmico de tarefas em arquiteturas híbridas, propondo novas estratégias de escalonamento eficientes e explorando diferentes cenários de aplicações e sistemas híbridos com variadas unidades de processamento.

1.2 Descrição

A princípio, exploramos as principais características desse nicho de escalonadores, analisando o comportamento de estratégias tradicionais, que são adaptativas ou não-adaptativas, presentes na literatura, em diversos conjuntos de aplicações sintéticas e reais. Por meio das observações extraídas dessa análise propomos duas novas estratégias de escalonamento dinâmicos de tarefas em arquiteturas híbridas, ambas construídas combinando diferentes abordagens de escalonadores dinâmicos já propostos na literatura. Basicamente, os escalonadores propostos combinam dois tipos de abor-

dagens distintas. Um primeiro, adaptativo, em que a distribuição das tarefas entre as unidades de processamento é feita de forma aleatória, mas permitindo que uma determinada unidade de processamento, quando ociosa, execute tarefas inicialmente assinaladas a outra UP [Blumofe & Leiserson, 1999]. 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 unidade de processamento ficará responsável por cada tarefa [Smith et al., 1999]. Para avaliar as estratégias, elaboramos diversos conjuntos de experimentos, compostos por aplicações sintéticas, nas quais as características das aplicações foram devidamente variadas. Além disso, a proposta foi avaliada também em cenários reais, compostos por aplicações relacionadas à álgebra linear (Fatoração de Cholesky, Gradiente Conjugado e Decomposição LU). Mostramos que nos cenários avaliados, as estratégias propostas conseguem melhor desempenho quando comparadas com as estratégias tradicionais, ao combinar características adaptativas com características não-adaptativas.

Adicionalmente à esses estudo inicial, apresentamos uma abordagem do escalonamento dinâmico de tarefas em uma arquitetura que não só é equipada CPU e GPU, mas como também o coprocessador MIC. Nessa nova abordagem, outras características do escalonado dinâmico de tarefas são avaliadas. Apresentamos outras duas novas estratégias de escalonamento que exploram o uso da predição de desempenho (*Speedup*) de tarefas dado as unidades de processamento, diferentemente da predição de tempo utilizada nos escalonadores apresentados anteriormente, e também a utilização de replicação de tarefas. Nesse contexto, os escalonadores propostos são avaliados por meio de uma aplicação de análise de imagens, a qual envolve a geração de uma grande quantidade de tarefas à serem escalonadas. Mostramos que os escalonadores propostos atingem bom desempenho no cenário avaliado, sendo que um deles supera as abordagens tradicionais. Complementarmente mostramos que a métrica de desempenho é menos sensível a presença de erro do que a métrica tempo de execução, o que torna nossas propostas mais acuradas.

1.3 Organização do texto

O restante deste trabalho está dividido em 7 capítulos, organizados da seguinte forma:

- **Capítulo 2 [Contextualização]**: aborda conceitos e tecnologias que foram explorados durante o desenvolvimento deste trabalho.
- **Capítulo 3 [Escalonamento de tarefas]**: apresenta um estudo sobre o problema de escalonamento de conjuntos de trabalho. Descreve os principais nichos

de escalonadores e suas características, apresentando com mais detalhes o problema do escalonamento dinâmico de tarefas, foco deste trabalho. Apresenta característica de escalonadores dinâmicos presentes na literatura, avaliando-os em diferentes cenários de aplicações.

- **Capítulo 4 [Escalonamento Adaptativo]**: propõe duas novas políticas de escalonamento dinâmico por meio da combinação de características adaptativas e não-adaptativas.

- **Capítulo 5 [Escalonamento Genérico em Arquiteturas Heterogêneas]**: apresenta uma nova abordagem para o problema de escalonamento dinâmico de tarefas com a inserção do coprocessador MIC. Adicionalmente, propõe outras duas novas políticas de escalonamento que se beneficiam de predição de desempenho e replicação de tarefas.

- **Capítulo 6 [Trabalhos Relacionados]**: descreve estudos similares ao que este trabalho aborda. Apresenta diferentes escalonadores dinâmicos para arquiteturas híbridas presentes na literatura, apontando as diferenças entre esses e as propostas deste trabalho.

- **Capítulo 7 [Conclusões e Trabalhos Futuros]**: discute alguns dos principais resultados deste trabalho e sugere direções para trabalhos futuros.

Capítulo 2

Contextutalização

Nesse capítulo apresentamos tecnologias e ferramentas utilizadas em nosso trabalho. Em um primeiro momento, discutimos arquiteturas heterogêneas de computadores para o processamento eficiente e massivamente paralelo de aplicações. Nesse contexto, apresentamos brevemente alguns coprocessadores, e com mais profundidade mostramos detalhes dos coprocessadores utilizados durante o trabalho: GPUs através da interface CUDA e Intel Xeon Phi (MIC). Em sequencia, motivado pelo uso coordenado desses coprocessadores, apresentamos *frameworks* e ambientes de execução para a exploração eficiente e coordenada de um sistema híbrido. Novamente apresentamos brevemente algumas dessas ferramentas presentes na literatura, e mais detalhadamente discutimos os ambientes de execução abordados nesse trabalho: StarPU e Extreme DataCutter.

2.1 Arquiteturas Emergentes

O estudo e aprimoramento de unidades de processamento (UPs) presentes em arquiteturas híbridas de computadores se tornou tópico de destaque no cenário de computação de alto desempenho devido à necessidade de processamento mais rápidos e eficientes em diversas áreas do conhecimento. Existe grande número de aplicações que podem ser executadas nessas UPs, a fim de conseguir extrair o máximo do processamento disponível. Porém a escrita de código para estas não é um tarefa trivial. O programador deve conhecer detalhadamente a arquitetura da UP, além de ser capaz de realizar operações complexas em baixo nível. Dessa forma, muitos estudos envolvem esforços para a escrita mais eficiente e ágil de código para as UPs, além de proporcionar uma interface mais alto nível para a implementação.

Com esse propósito as **GPGPUs** [Luebke et al., 2006] (*General-purpose computing on graphics processing units*) são tradicionalmente programadas usando

APIs padrões, como por exemplo **NVIDIA's CUDA** [NVIDIA, 2011; Fatica & Luebke, 2007] que é a forma mais comun de programação para GPUs atualmente. Similarmente, **Cell**[Williams et al., 2006] é usualmente programada diretamente por sua interface de baixo nível *LibSPE*. A FPGA *Clearspeed* e todas as outras placas aceleradoras (que são UPs), também precisam das suas respectivas interfaces comerciais para a programação. Um modelo genérico de programação para diferentes unidades de processamento é o OpenCL [Kim et al., 2012]. OpenCL (Open Computing Language) é uma arquitetura para escrever programas que funcionam em plataformas heterogêneas, consistindo em CPUs, GPUs e outros processadores. Inclui uma linguagem para escrever núcleos (funções executadas em dispositivos OpenCL), além de APIs que são usadas para definir e depois controlar as plataformas heterogêneas.

Similarmente as GPUs e FPGAs, atualmente podemos destacar o crescimento significativo de um coprocessador da Intel, o Intel's Xeon Phi. O que torna o uso desse coprocessador de alto desempenho interessante, além do seu expressivo poder de processamento, é a facilidade de implementação de aplicações para sua arquitetura. Basicamente, os núcles de processamento que esse coprocessador apresenta são arquiteturas x86, tornando a implementação similar à implementação em uma CPU tradicional, a qual utiliza-se de APIs padrões de processamento paralelo, como por exemplo OpenMP.

O estudo do desempenho de aplicações no coprocessador Intel's Xeon Phi é um tópico de pesquisas crescente atualmente [Joó et al., 2013; Heinecke et al., 2013; Cramer et al., 2012; Eisenlohr et al., 2012]. Algoritmos de álgebra linear vem sendo implementados para esse coprocessador [Heinecke et al., 2013; Eisenlohr et al., 2012], bem como métodos cromodinâmicos quânticos (Lattice Quantum Chromodynamics methods, LQCD) [Joó et al., 2013]. No trabalho [Hamidouche et al., 2013], os autores avaliam um processamento compartilhado entre várias máquinas equipadas com coprocessadores Intel Phi. Uma detalhada avaliação de prós e contras ao se usar OpenMP para a paralelização de operações no Intel Phi é outro recente tópico discutido no trabalho [Cramer et al., 2012].

O contexto de nosso trabalho aborda o uso coordenado de um sistema equipado com CPUs *multicores* e coprocessadores. Utilizaremos em nossas discussões e análises dois coprocessadores: (1) GPUs por meio da interface de programação CUDA; (2) Intel Xeon Phi. Tais coprocessadores se destacam pela pela eficiência no processamento massivo de dados e pela popularidade em pesquisas no contexto de computação paralela de alto desempenho.

Nas subseções seguintes serão apresentados os coprocessadores utilizados neste trabalho, características relevantes e informações sobre GPUs por meio da interface CUDA e Intel Xeon Phi.

2.1.1 GPUs (CUDA)

Com a popularização da utilização de placas gráficas (GPUs) para a computação de propósito geral (GPGPU), a empresa NVidia [1] desenvolveu uma interface de programação para facilitar a implementação de aplicações gerais para as suas GPUs [NVIDIA, 2011]. A popularização dessa interface de programação baseia-se no fato de ser uma extensão da linguagem de programação C, incluindo diretivas necessárias para controle e exploração eficiente das placas gráficas.

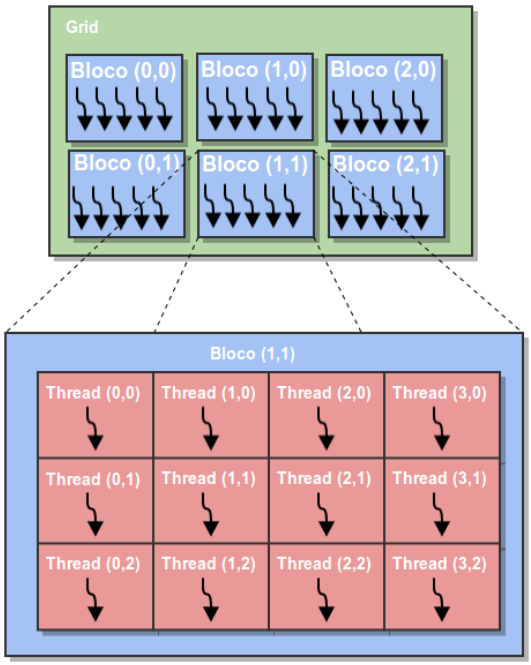


Figura 2.1: Arquitetura de uma GPU

O alto poder de processamento de placas de video NVidia se dá pela presença de um conjunto de multiprocessadores com múltiplos núcleos de processamento. Atualmente existe uma arquitetura de GPUs NVidia de alto poder de processamento massivamente paralelo para programação de propósito geral, NVidia Kepler. Nesse conjunto de GPUs, a quantidade de núcleos de processamento pode chegar a 2880, conseguindo processar operações de ponto flutuando com precisão simples em até $4.29Tflops$, e 1.43 Tflops para precisão dupla. Cada multiprocessador possui uma quantidade limitada e bem pequena de memória compartilhada, e em cada um dos núcleos são executadas sempre a mesma instrução, visto o paradigma SIMT (Single Instruction Multiple Thread) adotado pelo processamento GPU. Os diferentes multiprocessadores compartilham, por sua vez, uma memória global, expressivamente maior que a memória compartilhada internamente no multiprocessador. Existem diferentes custos de

[1] http://www.nvidia.com.br

acessos à esses níveis de memória, sendo que é de responsabilidade do programador a gerência e controle desses acessos.A Figura 2.1 expressa essa organização.

As GPUs por meio da interface CUDA são utilizadas para processamento massivamente paralelo de dados, executando concorrentemente um conjunto expressivo de threads na aplicação. De forma geral, a organização das *threads* em CUDA é realizado por meio da abstração de blocos e grids. As *threads* são agrupadas em conjuntos blocos, que por sua vez são organizados em conjuntos de grids, conforme ilustrado na Figura 2.2. Essa abstração facilita o controle e organização das *threads* durante o desenvolvimento da aplicação em GPU. Adicionalmente, o compartilhamento de memória entre as *threads* é limitado. *Threads* agrupadas em um mesmo bloco compartilham dados através da memória compartilhada do bloco, que é bastante pequena e o acesso é extremamente rápido. *Threads* de blocos diferentes compartilham dados através da memória global, expressivamente maior que a memória compartilhada, porém com acesso mais lento. Todo esse compartilhamento pode ser sincronizado através de diretivas fornecidas pela interface CUDA.

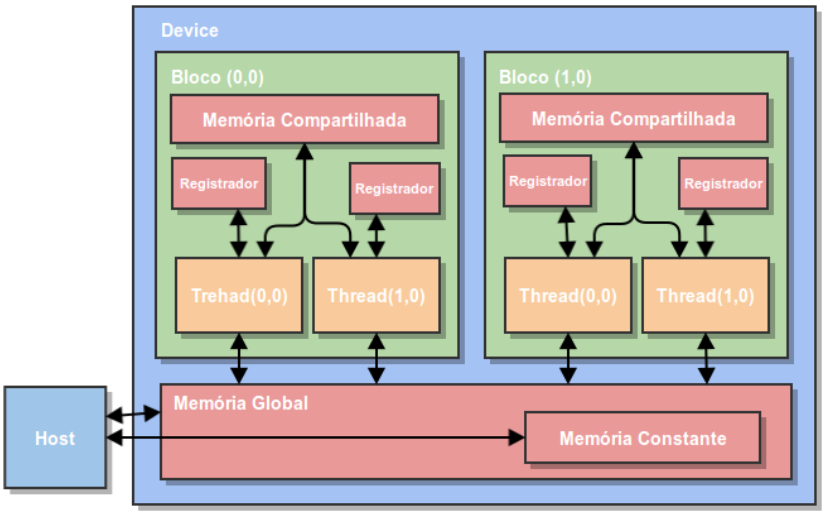


Figura 2.2: Organizações das *threads* pela interface CUDA

2.1.2 Intel Xeon Phi (MIC)

Intel Many Integrated Core Architecture (MIC) são coprocessadores adicionais que trabalham cooperativamente com as CPUs padrões de uma máquina. Comercializados através do nome Intel® Xeon Phi™, esse coprocessador tem formato similar ao de uma placa de vídeo PCIe x16. Pode ser considerado como um pequeno "supercomputador em uma placa"já que possui 61 núcleos de processamento, 8GB memória DDR5 e é abstraído pelo sistema hospedeiro (*host*) como sendo um outro computador. A sua

capacidade de processamento alcança cerca de 1.2 Tflops em operações de precisão dupla e cerca de 2 Tflops em operações de precisão simples.

Além do alto poder de processamento paralelo desse coprocessador, o qual consegue ter, através da tecnologia *Hyper-Threading* da Intel, 244 threads trabalhando concorrentemente, é a programação para esse coprocessador é através de conhecidas linguagens e frameworks de programação. Portanto, não há necessidade de aprender novas linguagens ou ferramentas.

Diferentemente de uma GPU, esse coprocessador pode hospedar um sistema operacional, ter IP totalmente endereçável e oferecer suporte a padrões como MPI. Além disso, outra característica extremamente interessante desse coprocessador, é que ele pode operar em diferentes modos de execução. Em um modo “simétrico"tarefas de carga de trabalho são compartilhadas entre o processador host e o coprocessador. No modo “Nativo"todo o trabalho é realizado no coprocessador, que por sua vez age como se fosse um nodo de computação separado. E por fim, o modo “Descarga"em que o coprocessador auxilia o processamento do host em parte da carga de trabalho conforme necessário.

A integração de 61 cores em uma placa exigiu que os circuitos desses processadores fossem construídos de forma simplificada. Cada um desses cores de processamento são uma versão de Pentium, com clock de apenas 1,053 GHz, porém para serem integrados em um mesmo chip de forma a reduzir o consumo energético, utilizam tecnologia de 22 nm da Intel.

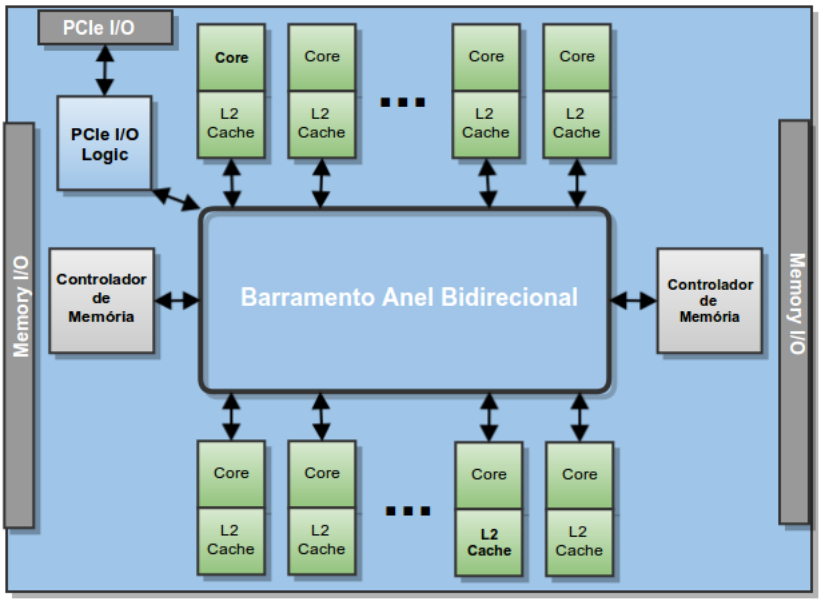


Figura 2.3: Arquitetura Intel Xeon Phi (MIC)

Em cada um dos cores de processamento existe uma uma memória cache L2

de 512k, adicionalmente os 8GB de memória DD5 são implementados através de 16 canais de memória de 5.5 Gbit cada. A organização desses componentes é realizada de forma anel bidirecional (Ring Interconnect), e está ilustrada na Figura 2.3. Essa estrutura permite algumas vantagens, como por exemplo o “cache miss"de um core ser atendido pela cache L2 de outro core através de um barramento interno. Além disso, em cada um dos cores são implementados uma unidade escalar e uma unidade vetorial, implementando a estratégia Fused Multiply-Add (FMA), a qual possibilita realizar duas operações simultâneas nos registros, sendo uma multiplicação e uma adição. Com registradores de 512 bits, esse coprocessador realiza 32 operações de ponto flutuante precisão simples de uma só vez, e 16 operações com precisão dupla.

2.2 Frameworks

A execução eficiente em sistemas híbridos equipados com CPUs e coprocessadores é um desafiador problema que requer uma implementação otimizada das operações de uma determinada aplicação para múltiplos processadores e escalonamento de trabalho entre dispositivos heterogêneos. Recentemente, uma quantidade significativa de ambientes de execução (runtime systems) [He et al., 2008; Linderman et al., 2008; Luk et al., 2009; Rossbach et al., 2011; Augonnet et al., 2012; Bosilca et al., 2011; Teodoro et al., 2010, 2009; Bueno et al., 2012; Ravi et al., 2010; Lima et al., 2013], técnicas de compilação [Ravi et al., 2010], e bibliotecas para contextos específicos [Bradski, 2000], vem sendo propostos para reduzir o esforço de programação e a complexidade envolvida na portabilidade de aplicações para esses sistemas híbridos.

Executar em uma plataforma CPU-GPU distribuída vem sendo o foco de vários projetos atualmente [Augonnet et al., 2012; Bosilca et al., 2011; Teodoro et al., 2010, 2009; Bueno et al., 2012; Ravi et al., 2010; Lima et al., 2013; Teodoro et al., 2012]. Podemos destacar o ambiente de execução DaGuE [Bosilca et al., 2011], o qual expressa as operações para serem computadas através de uma Directed Acyclic Graph (DAG), e foi avaliado através de aplicações de algebra linear. OmpSs [Bueno et al., 2012], também executa aplicações de fluxos de dados, as quais são paralelizadas em tempo de compilação, através de anotações realizadas no código pelo programador. Outro exemplo é o XKaapi [Gautier et al., 2013] que também suporta a execução cooperativa em máquinas híbridas CPU-GPU utilizando um esquema de multi versões, no qual operações possuem múltiplas implementações focando diferentes dispositivos de computação. Outro ambiente de execução que recebe grande destaque, é o StarPU [Augonnet et al., 2009]. Tal ambiente provê uma framework

para desenvolvimento de aplicações com alto nível de abstração. A transferência de dados é realizada de forma transparente ao programador e similarmente ao XKaapi, as operações possuem múltiplas implementações para os recursos de computação disponíveis. Esse ambiente se destaca por apresentar um consolidado framework para o desenvolvimento de escalonadores de operações nos sistemas híbridos.

Considerando não só GPUs como dispositivos de coprocessamento, pode-se citar Charm++ [Kale & Krishnan, 1993] que, atualmente, oferece suporte para Cell [Kunzman, 2006] além de GPUs, e o Extreme DataCutter [Teodoro et al., 2013] o qual insere o MIC como dispositivo de coprocessamento. Esse último ambiente de execução suporta a implementação de aplicações dataflow e o desenvolvimento de escalonadores de operações para o sistema híbrido.

Em nosso trabalho focamos a distribuição eficiente de tarefas entre as unidades de processamento disponíveis, sendo que utilizaremos coprocessadores GPUs e MICs em nossas discussões. Portanto, utilizaremos os ambientes StarPU e Extreme DataCutter para as implementações e avaliações de estratégias de escalonamento dinâmico ao longo do trabalho. Nas subseções seguintes detalhes desses ambientes serão apresentados.

2.2.1 StarPU

O *StarPU* [Augonnet et al., 2009] é um ambiente de execução que provê suporte para arquiteturas *multicore* híbridas, conforme apresentado na Figura 2.4. O ambiente unifica os recursos de processamento presentes, como as CPUs *multicore* e os aceleradores gráficos (GPUs), disponibilizando mecanismos para escalonar tarefas entre os recursos disponíveis, realizando de forma transparente e portável a transferência de dados.

O acesso a memória dos diferentes coprocessadores, bem como a transferência de dados entre as unidades de processamento, tradicionalmente é realizado de forma explícita através das respectivas interfaces de programação. Porém, o ambiente de execução StarPU fornece uma biblioteca para facilitar e automatizar essa movimentação e acesso de dados. Esse mecanismo implementa uma memória virtual compartilhada e visível por todas as unidades de processamento, utilizando técnicas de consistência de memória e recursos de replicação de dados. Além disso, todas as tarefas possuem referências explícitas aos seus dados e a localizações dos mesmo, o que facilita ao escalonador buscar e endereçar qualquer informação antes da execução da operação.

Além da transferência transparente de dados, o *StarPU* apresenta um modelo de execução o qual aborda as tarefas a serem executadas independente da arquitetura que se tem como base. São definidos os chamados *codelets*, uma abstração de tarefa, podendo ser executado em qualquer unidade de processamento. Cada *codelet* apre-

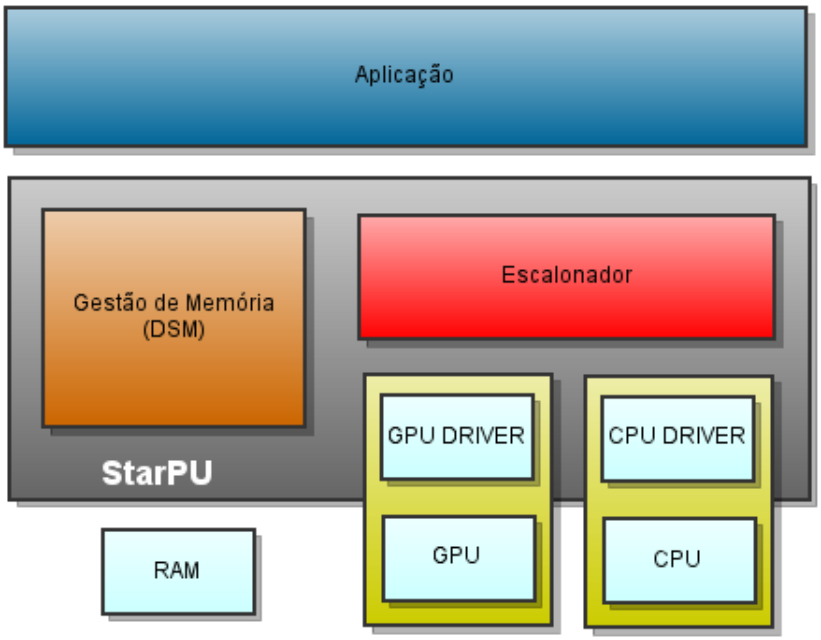


Figura 2.4: Modelo abstrato do *StarPU*

senta um conjunto de funções núcleo, implementadas para cada um dos dispositivos disponíveis, de forma que cada implementação utiliza as interfaces de programação ou bibliotecas específicas para a arquitetura em questão. Adicionalmente, em cada *codelet* é apresentado uma descrição dos dados e o tipo de acesso a eles, seja apenas leitura, ou apenas escrita, ou ambas. Dessa forma o ambiente de execução pode reordenar as tarefas durante a execução seguindo a política de escalonamento especificada, garantindo a semântica de dependência de dados definida pelo desenvolvedor. O código 2.1 exemplifica a implementação de um *codelet*. No código está representado uma função que será executada pela CPU e outra que será executada na GPU. A estrutura do *codelet* indica onde ele será executado, através das *flags* $STARPU\_CPU$ e $STARPU\_GPU$, bem como são definidos quais as funções responsáveis por cada execução, em CPU e GPU..

```
1  void funcao_gpu(void *buffers[], void *cl_arg){
2    /* Implementacao em CUDA */
3    ...
4  }
5  void funcao_cpu(void *buffers[], void *cl_arg){
6    /* Implementacao em CPU */
7    ...
8  }
9  static starpu_codelet funcao_cl = {
10   .where = STARPU_CPU | STARPU_GPU,
11   .cpu_func = funcao_cpu,
12   .gpu_func = funcao_gpu
```

```
13 }
```

Código 2.1: Estrutura básica de um Codelet

As execuções dos *codelets* nas diferentes unidades de processamento são gerenciadas por *threads* controladores atribuídas a cada um dos recursos disponíveis. O ambiente StarPU, utiliza a estratégia de assinalar tarefas para a execução em uma unidade de processamento, quando está estiver ociosa, em outras palavras, a *thread* responsável por gerenciar determinado recurso vai recuperar uma tarefa para a execução sob demanda. Os *codelets* são lançados de forma assíncrona para serem executados, o que possibilita ao escalonador reordenar as tarefas da melhor maneira possível a fim de alcançar um bom desempenho..

Basicamente, o ambiente de execução StarPU está organizado em três componentes principais: **Gerenciamento de Dados**, o qual consiste em uma biblioteca de alto nível para se automatizar de forma eficiente as transferências de dados entre os *workers* da arquitetura híbrida em questão. Um **Modelo de Execução Unificado**, ou seja, uma abordagem uniforme para paralelismo de dados e tarefas em plataformas híbridas. E, por fim, **Políticas de Escalonamento**, onde o StarPU disponibiliza um framework que permite, além de a utilização de diferentes escalonadores, a implementação de políticas de escalonamento projetadas pelos usuários. O diagrama da figura 2.5 apresenta o encadeamento das chamadas de funções de um programa simples implementado, utilizando o ambiente StarPU. A maioria das chamadas apresentadas não são visíveis ao programador, já que fazem parte da do funcionamento do ambiente. As cores diferenciam os componentes citados anteriormente: Em amarelo estão chamadas de funções pertencentes ao **Modelo de Execução Unificado**, em vermelho, está o **Gerenciamento de Dados e Memória**, e por fim, em azul funções pertencentes ao arcabouço de implementação (*framework*) de **Políticas de Escalonamento**.

O ambiente de execução StarPU, além de disponibilizar alguns escalonadores já implementados, permite ao usuário desenvolver suas próprias estratégias de escalonamento. O *framework* de implementação de novas políticas de escalonamento envolvem a elaboração de apenas quatro rotinas:

- **Inicialização de Estruturas**: Rotina responsável por inicializar as estruturas de dados e variáveis que serão utilizadas durante a execução do escalonador. Geralmente nessa rotina inicializam-se as listas de tarefas, as estruturas de controle de seções críticas (*mutex, semáforos*), bem como qualquer outro tipo abstrato de dados que será utilizado.

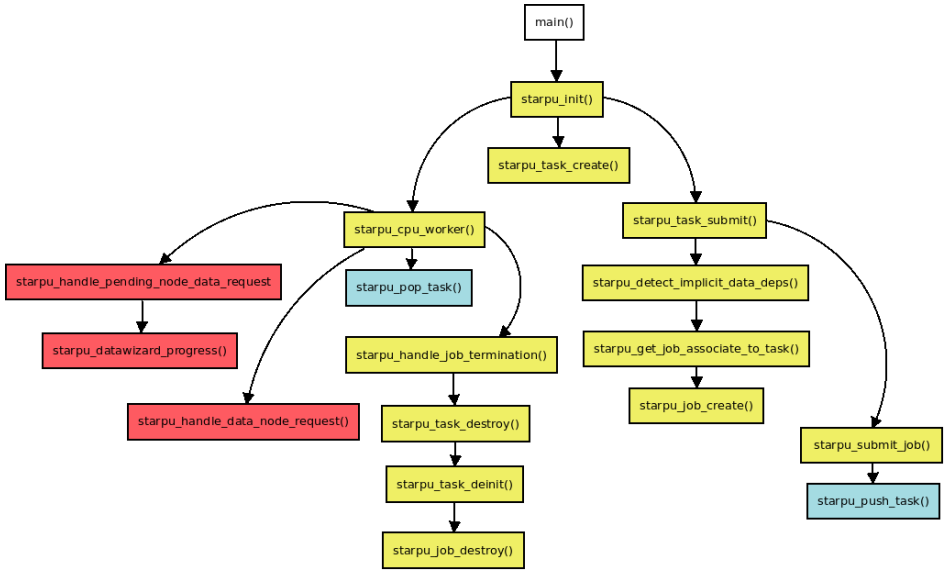


Figura 2.5: Diagrama de fluxo do *StarPU*

- **Término do Escalonador**: Responsável por finalizar as estruturas inicializadas e, dessa forma, terminar a execução do escalonador.

- **Push Task**: Função responsável por receber as tarefas após elas terem sido submetidas na aplicação principal pela rotina *starpu_ submit_ task()*. Nessa função, é implementada uma parte da lógica do escalonador, onde são estabelecidas para qual estrutura de dados as tarefas vão ser alocadas, bem como a ordem em que elas são guardadas até que uma *thread* gerenciadora solicite uma para a execução.

- **Pop Task**: Rotina responsável por retornar uma tarefa, assim que uma *thread* gerenciadora solicitar. Essa função é chamada constantemente por cada *thread* gerenciadora, e nela está a outra parte da lógica da política de escalonamento. Nela será definido como e de onde cada *thread* gerenciadora recuperará uma tarefa para executar.

2.2.2 Extreme DataCutter

O Extreme DataCutter [Teodoro et al., 2013] é um ambiente de execução para sistema heterogêneos distribuídos, que suporta a execução de aplicações com estrutura baseada em fluxo de dados (*dataflows*). Similarmente ao DataCutter [Beynon et al., 2001], cada estágio da aplicação pode ser replicado e associado a diferentes nodos de um sistema distribuído de máquinas.

Mais especificamente, no Extreme DataCutter, aplicações podem ser expressadas através de um *dataflow* hierárquico. Essa estrutura permite que os estágios mais

complexos da aplicação (estágios grão-grosso) sejam implementados através de outro dataflow constituído de operações indivisíveis (operações grão-fino). De forma que, essas tarefas indivisíveis, ou mesmo operações grão-fino, serão executadas nas unidades de processamento presentes nos nodos do sistema distribuído.

O ambiente executa aplicações utilizando uma estratégia de “bag-of-tasks"para assinalar instâncias de estágios grão-grosso para máquinas do sistema distribuído. Em cada nodo, o ambiente executa um esquema de resolução de dependências de tarefas para assegurar que esses estágios sejam assinaladas para a execução apenas quando todas as dependências estejam resolvidas.

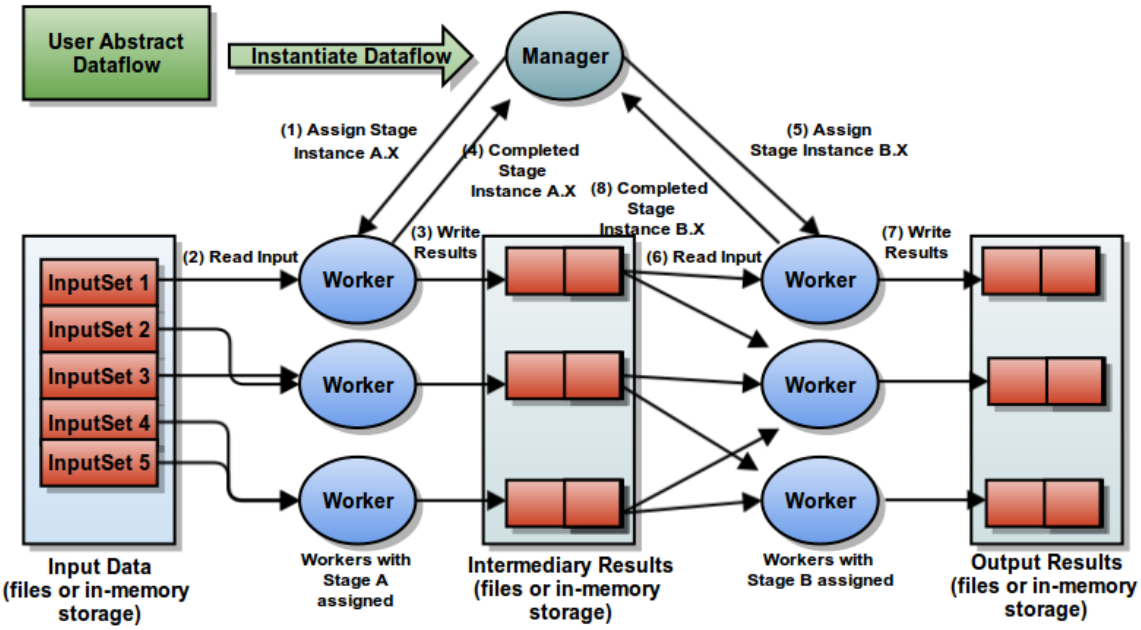


Figura 2.6: Visão Geral da Arquitetura do Extreme DataCutter.

Para escalonar e executar estágios grão-grosso da aplicação entre os nodos de processamento do sistema híbrido distribuído, o Extreme DataCutter utiliza um modelo Manager-Worker, como apresentado pela Figura 2.6. Através desse modelo, o desenvolvedor é responsável por implementar uma parte do código do processo Manager. Essa parte do código é responsável por instanciar estágios grão-grosso e definir dependências entre eles.

Cada um dos nodos do ambiente de execução, irá executar uma instância do processo Worker, que sob demanda irá receber e executar estágios grão-grosso da aplicação. Em outras palavras, o processo Manager instancia e define dependências dos estágios grão-grosso da aplicação com arquitetura dataflow hierárquico. Esses estágios são assinalados à nodos de processamento que executam instâncias de processos Workers,

até que não haja mais nenhum estágio grão-grosso para ser executado. A comunicação Manager-Worker é implementada através de Message Passing Interface (MPI).

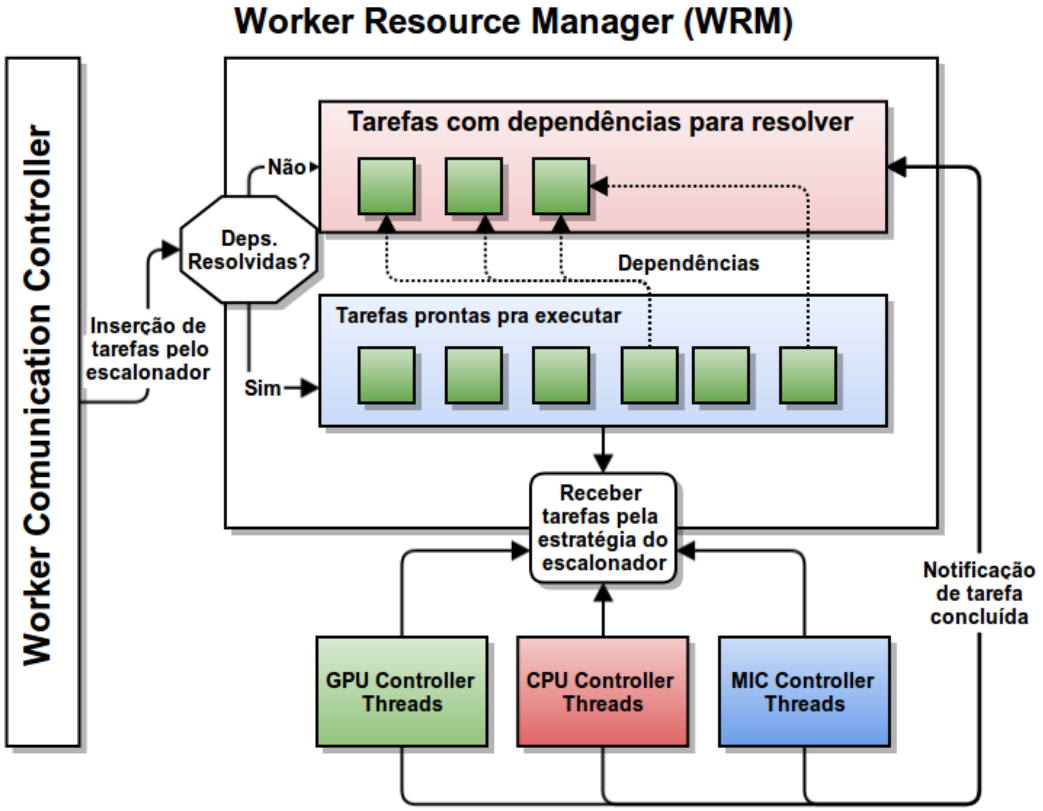


Figura 2.7: Ambiente de Execução de um Worker.

No nível mais baixo da organização do Extreme DataCutter, os processos Workers são capazes de usar todos os recursos em um nodo de processamento do sistema distribuído (CPU cores, GPUs and MICs). Múltiplas instâncias de estágios grão-grosso podem ser assinalados para a execução para o mesmo Worker concorrentemente, de forma a utilizar completamente os dispositivos disponíveis, sendo que a quantidade máxima de instâncias ativas em um worker é um do ambiente parâmetro chamado Worker Request Window Size.

Como apresentado pela Figura 2.7, um Worker é implementado por diversos componentes. O Worker Communication Controller (WCC) é responsável pela comunicação com o processo Maneger. A administração dos dispositivos de computação é executada por um Worker Resource Manager (WRM).

Uma vez que uma instancia de um estágio grão-grosso é recebida para a execução, ela deve instanciar um dataflow de operações grão-fino para executarem e concluírem o estágio grão-grosso. Essas operações, ou tarefas, são enviadas para a execução pelo WRM, o qual vai associar tarefas para serem executadas nos dispositivos disponíveis.

O módulo WRM cria uma *thread* para cada CPU core, GPU e MIC disponíveis. A thread controladora do dispositivo notifica ao WRM uma vez que está disponível, e assim o escalonador presente no WRM seleciona uma das tarefas (operações grão-fino) com dependências resolvidas para a execução. Após todos as operações grão-fino enviadas pela instancia do estágio de grão-grosso tenham sido finalizadas, o módulo WCC notifica ao Maneger, o qual irá liberar as dependências desse estágio concluído.

Da mesma forma que no ambiente StarPU, no Extreme DataCutter os escalonadores são implementados basicamente através de três funções principais: (1) Inicialização das estruturas utilizadas pelos algoritmos: listas, filas, sincronizadores. (2) Rotina para receber e organizar em uma das estruturas do escalonador, uma operação grão-fino quando essa é submetida para a execução. (3) Rotina para retornar uma operação grão-fino quando a thread gerenciadora sinaliza ao WRM que está ociosa.

2.3 Sumário

Neste capítulo apresentamos um pouco sobre tecnologias e ferramentas que utilizamos em nosso trabalho. A princípio, foram discutidas coprocessadores e interfaces de programação para arquiteturas heterogêneas, focando o processamento massivamente paralelo de aplicação. Mais detalhadamente foi apresentados detalhes dos coprocessadores utilizados neste trabalho: GPUs através da interface de programação CUDA e Intel Xeon Phi (MIC). Posteriormente foram discutidos alguns ambientes de execução presentes na literatura, utilizados para o uso coordenado e eficiente de sistemas híbridos. Novamente, de forma mais detalhada foram apresentadas as características dos dois ambientes de execução utilizados nesse trabalho: StarPU e Extreme DataCutter.

O uso coordenado dos recursos computacionais presentes em uma arquitetura híbrida através dos ambientes de execução, se dá, entre outras técnicas, pelo escalonamento eficiente de tarefas da aplicação. Por tanto no capítulo seguinte será apresentado uma completa discussão sobre escalonamento de tarefas, expondo características e avaliações de escalonadores tradicionais presentes na literatura.

Capítulo 3

Escalonamento de tarefas

Nesse capítulo apresentamos um estudo sobre o problema de escalonamento de tarefas. O objetivo do capítulo é apresentar as diferentes classificações dos escalonadores, que são organizadas por suas características de tomada de decisões ou pela características das tarefas que estão sendo escalonadas. Mais detalhadamente expomos o problema do escalonamento dinâmico de tarefas em arquiteturas híbridas, foco deste trabalho. Apresentamos sub divisões desse problema e características fundamentais que devem ser tratadas nesse contexto. Algumas definições descritas ao longo da classificação dos escalonadores foram transcritas do trabalho [RODAMILANS, 2009], tomado como base para essa análise.

Adicionalmente realizamos uma análise experimental do escalonamento dinâmico de tarefas em arquiteturas híbridas. Na literatura são apresentados alguns escalonadores desse grupo, tradicionais e consolidados [Augonnet et al., 2009; Smith et al., 1999; Blumofe & Leiserson, 1999], os quais implementamos em um ambiente de execução com suporte à arquiteturas híbridas e apresentamos uma avaliação do comportamento dessas políticas de escalonamento, utilizando um conjunto variado de aplicações. Foram selecionados três aplicações conhecidas: Gradiente Conjugado, Fatoração Cholesky e Decomposição LU. Além disso, implementamos aplicações sintéticas genéricas, as quais controlamos as características das tarefas, de formas a termos cenários sintéticos completamente controlados. Discutimos o desempenho de cada escalonador associando os resultados às características de tomada de decisão de cada escalonador, evidenciando pontos que tornam determinada política de escalonamento melhor ao distribuir as tarefas.

Por meio desse estudo mostramos que tais escalonadores possuem desempenhos variados em diferentes cenários de aplicações. Em outras palavras, em nosso estudo destacamos que não existe um escalonador ideal para qualquer tipo de aplicação.

3.1 Escalonamento

Escalonamento de forma geral, é o processo de associar componentes de trabalho, como por exemplo tarefas de uma aplicação, à recursos de processamento disponíveis, visando atingir uma determinada função objetivo. Podemos ter como metas, a minimização do tempo de término do conjunto de componentes de trabalho, a maximização da utilização dos recursos disponíveis para processamento, a maximização do número de tarefas processadas por unidade de tempo, ou até mesmo a minimização do consumo energético dos recursos do sistema. O estudo de escalonamento é tópico de pesquisa à vários anos em diversos cenários, como por exemplo em sistemas distribuídos [Feitelson et al., 1997; Ahmad, 1995], e mais recentemente vem ganhando destaque no cenário de computação paralela em arquiteturas híbridas compostas por diferentes processadores e coprocessadores, foco deste trabalho.

Em [Casavant & Kuhl, 1988] é proposta uma taxonomia para o problema de escalonamento. Por meio dessa classificação de escalonadores ilustrada na figura 3.1 será possível identificarmos o conjunto de estratégias e características que compõe a motivação deste trabalho.

Segundo essa organização de escalonamento, no topo temos duas vertentes diferentes e primárias: Escalonamento Local e Escalonamento Global. Basicamente o objetivo de escalonadores locais é atribuir intervalos de tempo de apenas um processador. Sendo que dessa forma os componentes de um conjunto de trabalho não são executados paralelamente, e sim um de cada vez seguindo uma determinada ordem estipulada pelo escalonador. Por sua vez, escalonadores globais definem em qual processador um determinado processo ou tarefa será executado. Em nosso trabalho, visamos a execução eficiente e paralela de tarefas de uma aplicação, portanto estudaremos apenas a classe de escalonadores globais.

Na categoria de escalonadores globais, novamente temos uma divisão em dois tipos distintos de estratégias de escalonamento: Escalonadores Estáticos e Escalonadores Dinâmicos. Podemos definir que no escalonamento estático, as decisões são tomadas previamente. Em outras palavras, antes da execução da aplicação, o escalonador tem conhecimento de todas as tarefas da aplicação e as reorganiza da melhor forma entre os processadores disponíveis. Por sua vez, no escalonamento dinâmico, o comportamento dinâmico dos recursos e das tarefas são considerados, sendo assim, as decisões de escalonamento não acontecem previamente, como no escalonamento estático, e sim concorrentemente com a execução da aplicação.

O escalonamento dinâmico é um cenário um pouco mais desafiador que o escalonamento estático, visto que as tarefas chegam dinamicamente ao escalonador, que

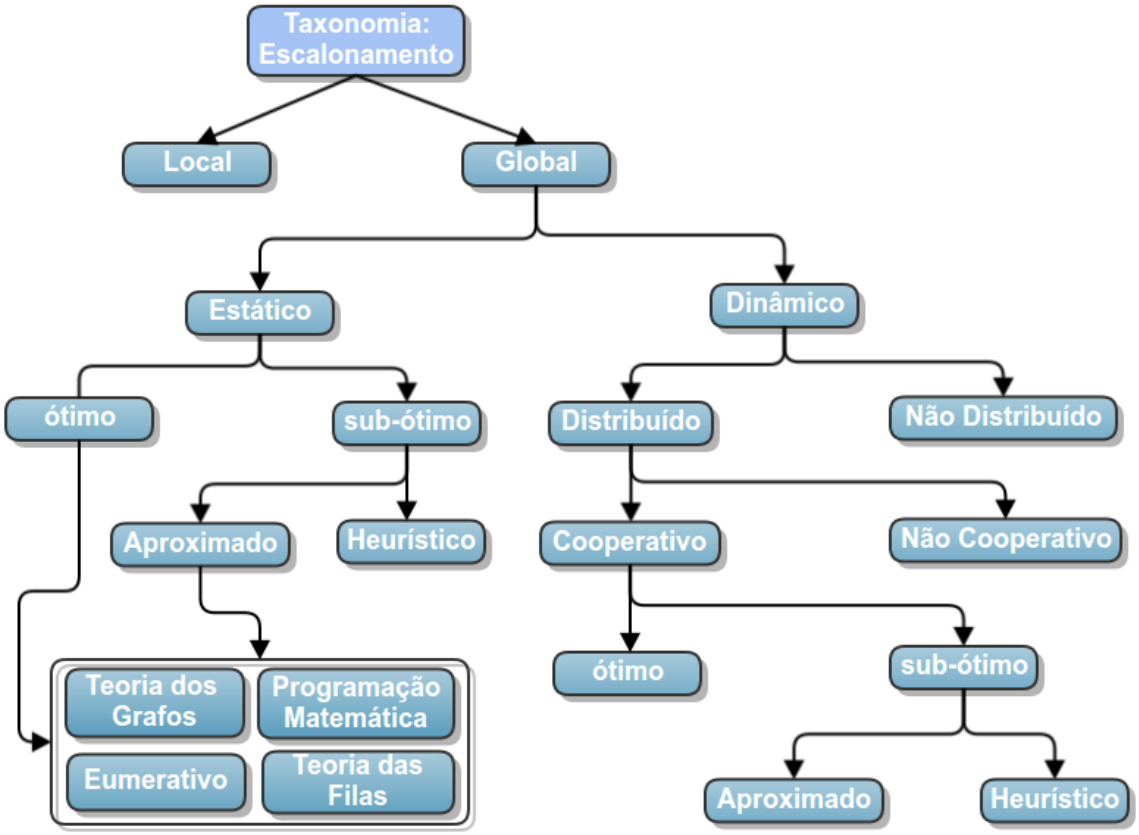


Figura 3.1: Classificação de Escalonamento apresentada em [RODAMILANS, 2009] tendo como base o trabalho [Casavant & Kuhl, 1988].

por sua vez deve coletar informações dos recursos (processadores e coprocessadores, por exemplo) disponíveis além de analisar características da tarefa que chegou para ser escalonada. Só depois de coletadas informações do sistema como um todo, recursos e tarefa, é feito o processo de decisão e associação da tarefa à unidade de processamento adequada.

Seguindo a organização taxonômica dos escalonadores, temos que os escalonadores estáticos podem ser ótimos ou sub-ótimos. Por ser, em muitos casos, um problema NP-Completo, só teremos um escalonamento estático ótimo quando os recursos de processamento disponíveis e a aplicação a ser executada, bem como todas as tarefas que a compõe, são muito bem conhecidos e estáveis. Apenas quando se tem total domínio e conhecimento do comportamento do sistema, recursos e aplicação, e quando é computacionalmente possível descobrir a melhor organização das tarefas, teremos um escalonamento ótimo. Porém, na maioria dos casos de escalonamento, é computacionalmente inviável selecionar a melhor organização das tarefas de uma aplicação de forma a termos um escalonamento ótimo, já que esse é um processo enumerativo. Dessa forma aplica-se o escalonamento sub-ótimo.

Os escalonadores sub-ótimos podem ser divididos em: Escalonadores Sub-ótimos aproximados e Escalonadores Sub-ótimos heurísticos. Sendo que o escalonamento aproximado acontece quando encontra-se uma boa solução para o problema do escalonamento, utilizando uma determinada métrica que dirá quando a solução é aceitável ou não. Em contrapartida, no escalonamento heurístico, não existe uma métrica ou resultado objetivo, a definição de uma boa solução é dada baseada em observações prévias e no conhecimento das características do sistema.

Pelo fato de escalonadores estáticos terem uma visão global das tarefas da aplicação, dependências e prioridades, diversas técnicas são utilizadas para a tomada de decisão. Comumente as tarefas e suas dependências são modeladas, nesses casos, como um grafo direcionado acíclico (DAG), e técnicas presentas na teoria dos grafos são empregadas para se encontrar a melhor forma de percorrer esse grafo e assim encontrar uma organização satisfatória das tarefas [Bosilca et al., 2011]. Como o escalonamento tem por finalidade maximizar ou minimizar um determinado objetivo, também no escalonamento estático é possível a aplicação de técnicas de programação matemático [Augonnet et al., 2009] e teoria das filas [Terekhov et al., 2013] para encontrar a melhor distribuição prévia das tarefas.

O escalonamento dinâmico também apresenta duas sub divisões: Escalonamento dinâmico distribuído e Escalonamento dinâmico não-distribuídos. No caso de escalonamento dinâmico distribuído, as tarefas serão distribuídas para máquinas em um agrupamento de computadores. Mais especificamente, os recursos de processamentos são nós de uma rede de computação distribuída. Esse subtipo de escalonamento dinâmico, ainda pode ser dividido em outros dois outros: Escalonamento dinâmico distribuído cooperativo e Escalonamento dinâmico distribuído não-cooperativo. O escalonamento é dito cooperativo quando os componentes de processamento distribuídos colaboram entre si, com resultados parciais e informações de estados dos recursos. Por outro lado, o escalonamento é não-cooperativos quando os processadores tomada decisões independentemente dos resultados dos outros processadores envolvidos. Descendo mais o nível na taxonomia apresentada, os escalonadores cooperativos podem ser sub-ótimos ou ótimos (Aproximados ou Heurísticos), que tiveram suas características mencionadas em parágrafos anteriores.

Por sua vez, escalonamento dinâmico não-distribuído acontece em apenas uma máquina, a qual aproveita-se os recursos computacionais presentes em sua arquitetura para o processamento das tarefas. Atualmente, as arquiteturas computacionais são compostas de processadores com múltiplos cores (processadores *multicores*) auxiliados por coprocessadores de alto desempenho. Sendo assim, o escalonamento de tarefas dinâmico não-distribuído se dá nessas arquiteturas híbridas, com o conjunto de tarefas

de uma determinada aplicação sendo escalonado para as unidades de processamento de uma única máquina.

O escalonamento dinâmico de tarefas não possui o benefício de utilizar técnicas de programação matemática ou teoria dos grafos para auxiliar na tomada de decisões, como é feito no escalonamento estático, já que as tarefas vão surgindo dinamicamente e as decisões de escalonamento coocorrem com a execução. Sendo assim, para esse tipo de escalonamento, técnicas de predição e estimativa de desempenho de uma tarefa dado um recurso de processamento, são comumente aplicadas.

3.1.1 Dependências de tarefas da aplicação

Uma importante consideração que se deve fazer no projeto do algoritmo de escalonamento é em relação à dependência ou independência entre as tarefas que compõe a aplicação. Basicamente tarefas com dependência de uma aplicação são aquelas que necessitam esperar o término de uma ou mais tarefas (chamadas de tarefas pai) para iniciarem sua execução. Essa dependência entre tarefas cria uma determinada ordem de execução que deve ser mantida para coerência do resultado final da aplicação, sendo que tarefas dependentes não podem ser executadas concorrentemente com as suas respectivas tarefas pai. Por outro lado, tarefas independentes são aquelas que não necessitam esperar a finalização de nenhuma outra tarefa, podendo essas ser executadas em paralelo com qualquer outra.

3.1.1.1 Escalonamento de tarefas Independentes

Como discutido anteriormente, tarefas independentes são aquelas que podem ser executadas paralelamente a qualquer outra tarefa da aplicação, sem a necessidade de esperar o término de alguma delas. Sendo apresentado por [Dong & Akl, 2006] os escalonadores de tarefas independentes podem ser não-adaptativos ou adaptativos.

- ***Escalonadores não-adaptativos:*** Conforme [Dong & Akl, 2006] os escalonadores não-adaptativos de tarefas independentes podem estimar a desempenho de uma determinada tarefa dada as unidade de processamento, de forma que essa estimativa auxiliará no tomada de decisão do escalonador. Podemos ter alguns tipos de estimativa: *Minimum Execution Time* (MET) e *Minimun Completion Time* (MCT). Os algoritmos de escalonamento que utilizam a heurística MET estimam qual a unidade de processamento que irá minimizar o tempo de execução dessa tarefa. Por outro lado os algoritmos de escalonamento que utilizam a heurística MCT estimam qual a unidade de processamento que irá minimizar o

tempo de término da tarefa. Por meio dessas duas heurísticas para estimativa do desempenho dos escalonadores são derivados alguns outros algoritmos como por exemplo o algoritmo MaxMin [Dong & Akl, 2006; Casanova et al., 2000] que é uma estratégia de escalonamento que irá alocar tarefas definidas como mais "pesadas"à unidades de processamento definidas como maior capacidade de processamento. E o algoritmo XSufferage [Casanova et al., 2000; dos Santos Neto, 2004; Dong & Akl, 2006] que estima um valor prejuízo para caso uma determinada tarefa não for escalonada para a a unidade de processamento que a executará mais rápido. Essa ultima estratégia de escalonamento utiliza a transferência de dados necessários para a execução da tarefa como base para a estimativa do tempo de execução.

- ***Escalonadores adaptativos:*** Tais escalonadores, diferentemente dos não-adaptativos, não se beneficiam de uma estimativa de tempo de execução ou término de tarefas. Tais algoritmos dinamicamente avaliam o recurso que melhor executará uma determinada tarefa. Temos por exemplo o algoritmo *Workqueue with Replication* (WQR) [Da Silva et al., 2003; Dong & Akl, 2006] que não necessita de informação das unidades de processamento disponíveis para se escalonar tarefas independentes. Esse algoritmo tem como princípio básico a utilização de réplicas de uma tarefa. Quando uma unidade de processamento está ociosa, e não existem mais tarefas a serem escalonadas, esta recebe réplicas de tarefas que ainda serão executadas por outras unidades de processamento. Caso a tarefa original termine a sua execução, todas as réplicas são deletadas. Por sua vez, caso uma réplica termine antes, todas as outras réplicas e a tarefa original são deletadas. É fácil percebermos que a avaliação das unidades de processamento é feita de forma dinâmica visto que, tal análise coocorre com a execução das tarefas.

3.1.1.2 Escalonamento de tarefas Dependentes

Como visto na definição de tarefas dependentes, temos que para iniciar a execução de uma tarefa dependente de outra, essa necessita esperar o término da execução da tarefa pai. Tradicionalmente essa relação de dependência entre tarefas é representada por uma DAG (Direct Acyclic Graph). Tal representação facilita determinar a ordem e relações entre as tarefas que compõe uma aplicação. Outro benefício dessa representação como apresentado em [Dong & Akl, 2006] é possível representarmos a importância de cada tarefa por meio da inserção de pesos nos nodos do grafo, e é possível representarmos a transferência de dados entre tarefas, por meio do ponderamento de arestas. Nomeamos

workflow a ordem e organização de precedências entre tarefas dependentes de uma aplicação.

Dentre os escalonadores de tarefas dependentes temos escalonadores não-adaptativos e escalonadores adaptativos.

- **Escalonadores não-adaptativos:** Para o escalonamento de tarefas dependentes, o escalonamento não-adaptativo ainda é subdividido em escalonadores baseados em algoritmos de lista, agrupamento e algoritmos baseados em duplicação.
  - Nos algoritmos de lista cada tarefa possui uma prioridade ou peso determinados pela própria estratégia de escalonamento. As tarefas são ordenadas na lista por meio da importância de sua prioridade ou peso. Um exemplo de algoritmo de escalonamento estático de tarefas dependentes é o *Fast Critical Path* (FCP) [Dong & Akl, 2006].
  - Algoritmos baseados em duplicação basicamente visão reduzir o tempo de execução da aplicação, minimizando o tempo de término das tarefas, por meio da duplicação das tarefas em diferentes unidades de processamento. Como exemplo desse tipo de algoritmo de escalonamento, podemos citar as estratégias de escalonamento em ambientes distribuídos: *Task Duplication-Based* para ambientes com recursos homogêneos e o *Task Duplication-Based scheduling Algorithm for Network of Heterogeneous System* para ambientes com recursos heterogêneos, conforme apresentado por [Dong & Akl, 2006].
  - Algoritmos de agrupamento visão diminuir o custo de transferência de dados ou comunicação entre tarefas, dado um DAG. Basicamente esses algoritmos em uma primeira etapa agrupa tarefas com transferência de dados ou comunicação intensivas entre si, e em uma segunda etapa associa essas tarefas agrupadas em uma mesma unidade de processamento. Como exemplo desses tipos de algoritmos, podemos citar *Dominant Sequence Clustering* (DSC) e CASS-II [Dong & Akl, 2006] para o agrupamento de tarefas com intensa comunicação entre si e balanceamento de cargas, minimização do tráfego de comunicação e aleatório (RAND), como estratégias de associação das tarefas agrupadas.
- **Escalonadores adaptativos:** Características dinâmicas de um sistema de escalonamento podem influenciar na tomada de decisão online do escalonador. Por tanto, estratégias de escalonamento que estão atentas ao comportamento dos recursos disponíveis e do dinamismo da aplicação, adaptando-se as mudanças ocorridas durante a execução são chamados de escalonadores dinâmicos. Para tarefas

dependentes escalonadores adaptativos como por exemplo pM-S e *Dynamic Level Scheduling* (DLS) [Dong & Akl, 2006] são utilizados.

3.1.2 Função Objetivo

Como mencionado em parágrafos anteriores, o problema de escalonamento tem como motivação organizar tarefas de uma aplicação em recursos de processamento disponíveis, de forma a atingir uma meta. Dependendo do contexto da aplicação, a estratégia de escalonamento visa alcançar diferentes metas, como por exemplo a minimização do tempo de termino do conjunto de componentes de trabalho, a maximização da utilização dos recursos disponíveis para processamento, maximização do número de tarefas processadas por unidade de tempo, ou até mesmo a minimização do consumo energético das unidades de processamento. Tais metas são chamadas função objetivo.

De forma bastante geral, as funções objetivos dos escalonadores são dividas em: Funções objetivo centradas na aplicação (application centric) e Funções objetivo centradas nos recursos (resource centric).

Estratégias de escalonamento com funções objetivo centradas na aplicação visam otimizar o desempenho da aplicação como um todo. Algumas métricas são utilizadas por funções objetivos dessa classe, como por exemplo *makespan* e custo econômico. A métrica *makespan* é basicamente o tempo gastro entre o inicio da primeira tarefa e o término da última tarefa executada de uma aplicação. Geralmente, a função objetivo centrada na aplicação mais comum visa reduzir o *makespan*. O Custo econômico é uma métrica baseada em modelos econômicos, utilizando tempo e custo financeira como base para os cálculos.

Em contrapartida, temos estratégias de escalonamento com funções objetivo centradas nos recursos. Estratégias que adotam esse tipo de função objetivo, visam otimizar a eficiências dos recursos disponíveis ao escalonador. Geralmente essas funções estão relacionadas a taxa de utilização dos recurso disponíveis. Um exemplo que tem bastante destaque no cenário atual, é a questão da redução do consumo energético. Nesse contexto, uma função objetivo centrada em recurso, pode ter como meta a redução ou controle do consumo energético por máquinas e processadores no sistema.

3.2 Escalonamento Dinâmico de tarefas em Sistemas Híbridos

Considerando o problema foco de nosso trabalho: Dada uma aplicação composta por um conjunto de tarefas, independentes, em que durante sua execução essas tarefas vão sendo assinaladas dinamicamente para processamento, encontrar a melhor distribuição dessas tarefas entre um conjunto de processadores (CPU *multicores*) e coprocessadores (GPUs, MICs, FPGAs, etc..) presentes em uma máquina de computação de alto desempenho híbrida. Conforme a taxonomia apresentada na seção anterior, temos um problema de escalonamento: (1) **Global**, já que tarefas são assinaladas a diferentes unidades de processamento. (2) **Dinâmico**, pelo fato do escalonamento de tarefas coocorrer com a execução da aplicação. (3) **Não-distribuído**, visto que o escalonamento é feito em apenas uma máquina com arquitetura heterogênea. (4) **Heurístico**, pois o problema apresentado é NP-Completo, e não possuímos uma métrica para definirmos uma boa aproximação da solução ótima. Adotaremos Escalonamento Dinâmico de tarefas em Sistemas Híbridos para referir à classe de escalonadores globais dinâmicos não-distribuídos heurísticos.

O escalonamento dinâmico de tarefas em sistemas híbridoss tem por finalidade explorar de maneira eficiente os recursos computacionais de máquinas com processadores e coprocessadores heterogêneos. Em tal contexto, o escalonador objetiva atribuir da melhor maneira possível sub tarefas de uma aplicação entre as unidades de processamento disponíveis, a fim de minimizar o tempo final de execução da aplicação.

Nesse cenário consideramos que tarefas distintas de uma aplicação, são computacionalmente adequadas para serem executadas em determinadas arquiteturas, em outras palavras, tarefas distintas possuem requisitos de execuções diferentes. Não só os requisitos de execução tornam-se um fator de decisão na hora do escalonamento, mas como também o custo de transferência de dados entre as unidades de processamento disponíveis e a dependência de dados entre as tarefas.

Portanto, alguns fatores devem ser considerados pelas políticas de escalonamento dinâmicos de tarefas em sistemas híbridos, para que o escalonador consiga explorar da melhor maneira possível os recursos disponíveis:

- **Boa adequação das tarefas:** Tendo em vista que tarefas distintas são computacionalmente adequadas para executar em arquiteturas específicas, é fundamental que o escalonador consiga identificar qual é a arquitetura que minimiza o tempo de execução de cada tarefa. Quando uma tarefa é associada à uma arquitetura inadequada para sua execução, o desempenho da aplicação é diretamente com-

prometido, já que essa tarefa poderia ter seu tempo de execução minimizado se associada a arquitetura adequada. Em outras palavras, quando há boa adequação das tarefas às unidades de processamento, evitamos trabalhos excessivo e consequentemente o tempo de término da aplicação é minimizado.

- **Balanceamento de Carga:** Conforme mencionado em [Casavant & Kuhl, 1988] esse fator é fundamental à todas as classes de escalonadores. É fundamental manter todos as unidades de processamento constantemente em execução, de forma justa e balanceada. Basicamente é necessário que todas as unidades de processamento tenham as mesmas taxas de execução, evitando sobrecarga de trabalho ou ociosidade em um determinado processador. Esse fator também impacta diretamente no desempenho final da aplicação, já que com uma distribuição de trabalho homogênea entre os processadores maximiza a utilização dos recursos presentes na arquitetura híbrida.

- **Transferência de dados entre processadores:** A associação de uma tarefa a um coprocessador exige a transferência de dados necessários ao processamento para a arquitetura que encarregada da execução. Essa movimentação de dados, quando manipulado uma quantidade relativamente grande de informação, gera um tempo extra de transferência, *overhead*, o qual em alguns casos pode ser suficientemente grande para prejudicar o tempo total da aplicação. Portanto, considerar o volume de dados que será transferido ao coprocessador é muito importante na tomada de decisão do escalonador, já que pode haver situações em que o *speedup* alcançado pela arquitetura não compensa o *overhead* de transferência de dados realizado.

Escalonadores dinâmicos de tarefas, ao tentar lidar com os fatores mencionados, criam estratégias variadas que os diferenciam em escalonadores adaptativos e escalonadores não-adaptativos, conforme apresentado no trabalho [Casavant & Kuhl, 1988]. Um escalonador dinâmico adaptativo apresenta um comportamento variável de sua política de decisão de acordo com o comportamento dinâmico do sistema. Mais especificamente, escalonadores dinâmicos adaptativos, analisam o comportamento dos recursos computacionais e tarefas, dinamicamente durante a execução da aplicação, realizando alterações nas associações de tarefas aos recursos disponíveis, beneficiando seu escalonamento das tarefas. Em contrapartida, os escalonamento não-adaptativo mantém a associação das tarefas independente do comportamento dinâmico do sistema.

3.2.1 Algoritmos de Escalonamento Dinâmico Não-Adaptativos

Como já mencionado, algoritmos de escalonamento dinâmico não-adaptativos, não altera dinamicamente a associação das tarefas às unidades de processamento disponíveis. Com a chegada de tarefas para serem escalonadas, a política não-adaptativa analisa os recursos de processamento disponíveis e cada tarefa individualmente, realizando a associação da forma mais adequada possível.

Podemos citar, estratégias simples não-adaptativos: First-Come First-Served (FCFS), uma estratégia aleatória (Random) ou e uma estratégia Round-Robin. A estratégia FCFS utiliza uma estrutura global de lista sendo que cada unidade de processamento recebe a próxima tarefa da cabeça dessa lista. Toda associação realizada é mantida, não havendo troca de tarefas entre unidades de processamento após a remoção da lista global. O mesmo acontece com uma estratégia aleatória e Round-Robin, a qual não apresenta critério nenhum para associação.

Visando a boa adequação das tarefas, estratégias mais elaboradas de associação de tarefas à unidades de processamento são utilizadas em algoritmos de escalonamento mais inteligentes. Nesses casos heurísticas de estimativa e predição de desempenho de uma tarefa dada a unidade de processamento auxiliam na tomada de decisão dos escalonadores.

O algoritmo HEFT (Heterogeneous Earliest Finish Time) [Augonnet et al., 2009] procura associa uma tarefa à unidade de processamento que minimiza seu tempo de término, ou seja, a unidade que apresenta o menor MCT (Minimun Completion Time) para uma determinada tarefa. Similarmente como já citado, o algoritmo MaxMin [Dong & Akl, 2006; Casanova et al., 2000] aloca tarefas definidas como mais “pesadas"à unidades de processamento definidas como maior capacidade de processamento. E o algoritmo XSufferage [Casanova et al., 2000; dos Santos Neto, 2004; Dong & Akl, 2006] que estima um valor prejuízo para caso uma determinada tarefa não for escalonada para a a unidade de processamento que a executará mais rápido.

Vemos que escalonadores não-adaptativos utilizando em sua grande maioria as métricas Minimum Execution Time (MET) e Minimun Completion Time (MCT). Estimar o tempo de execução de uma tarefa em uma determinada arquitetura é um desafio bastante estudado no contexto de escalonamento dinâmico de tarefas e arquiteturas híbridas [Smith et al., 1999].

Existem estratégias de predição baseadas em pré-processamento (“offline"), e estratégias de predição dinâmicas (“online"). Estimativas “offline", se baseiam na realização de um pré-processamento da aplicação, ou de aplicações similares, que será

executada, construindo um perfil (*profiling*) do conjunto de tarefas que as compõe. Esse *profiling* guarda informações do comportamento e características das tarefas executadas nas diferentes arquiteturas dos recursos disponíveis. Basicamente nesse *profiling* sao extraídos parâmetros da tarefa, quantidade de dados manipuladas, e os tempos de execução em cada arquiteturas de processamento e constrói-se uma base de informações, onde associa-se características de tarefas preprocessadas e os tempos de execução das mesmas nos processadores. Por sua vez, estratégias de predição "online", a base de informações vai sendo construído dinamicamente durante a execução das tarefas da aplicação. Essas estratégias são chamadas de escalonamento baseado em histórico de execução, podendo ser preprocessado ou dinâmico.

Com esse histórico de execuções passadas, o escalonador consegue predizer ou estimar o desempenho de uma tarefa. Essa estimativa, tradicionalmente se dá por similaridade entre as característica da tarefa e características armazenadas no histórico. Existem escalonadores que utilizam classificadores automáticos, como por exemplo o KNN (K-Nearest Neighbours) para encontrar qual a tarefa do histórico mais similar à que está sendo escalonada e assim estimar a unidade de processamento mais adequada a ela [Teodoro et al., 2009]. Existem outros escalonadores que utilizam regressão linear para tanto [Augonnet et al., 2009].

3.2.2 Escalonamento Dinâmico Adaptativos

Essa classe de algoritmos dinâmicos não mantém a associação inicial das tarefas. É considerado adaptativo o escalonador que durante a execução da aplicação movimenta, replica e transfere tarefas entre as unidades de processamento.

Como já mencionado o algoritmo Workqueue with Replication (WQR) [Da Silva et al., 2003; Dong & Akl, 2006] é um bom exemplo de escalonador adaptativo. O WQR é uma versão aprimorada do WQ [Da Silva et al., 2003] com a inclusão da utilização de réplicas em unidades de processamento ociosas. Quando uma unidade de processamento está ociosa, e não existem mais tarefas a serem escalonadas, esta recebe réplicas de tarefas que ainda serão executadas por outras unidades de processamento. Caso a tarefa original termine a sua execução, todas as réplicas são deletadas. Por sua vez, caso uma réplica termine antes, todas as outras réplicas e a tarefa original são deletadas.

Outro exemplo é o escalonador *Work Stealing* (WS) [Blumofe & Leiserson, 1999]. Seguindo a mesma ideia do escalonador WQR, quando uma unidade de processamento está ociosa, o algoritmo analisa qual outra unidade de processamento possui trabalho excessivo. Após identificar essa unidade de processamento, o escalonador WS

“rouba"uma tarefa que está associada à unidade sobrecarregada, e atribui à unidade ociosa.

Por meio dos dois exemplos citados, notamos que a avaliação das unidades de processamento é feita de forma dinâmica, já que, tal análise coocorre com a execução das tarefas. Além disso, a associação inicial das tarefas, que pode ser feita de forma Round-Robin ou aleatório, não é mantida durante a execução da aplicação, acontecendo movimentação de tarefas entre as unidades de processamento.

3.3 Avaliação Experimental de Escalonadores Dinâmicos

Nessa seção visamos analisar experimentalmente o comportamento de estratégias de escalonamento dinâmico adaptativas e não-adaptativas, a fim de identificarmos características interessantes em cada um desses conjuntos de escalonadores dinâmicos e verificarmos os cenários que cada uma estratégia alcança bom desempenho. Todos os escalonadores foram implementados dentro do ambiente de execução StarPU, detalhado na subseção 2.2.1 considerando como recursos de processamento CPUs e GPUs. Os algoritmos implementados serão descritos na subseção 3.3.1 e as cargas utilizadas serão descritas na subseção 3.3.2.

3.3.1 Algoritmos Implementados

Para a analise experimental de algoritmos de escalonamento dinâmico de tarefas, foram implementadas quatro estratégias distintas: FCFS, HEFT, HEFT *data-aware* e *Work Stealing*. Sendo (1) **FCFS** uma estratégia não-adaptativa sem uma política de predição de performance. (2) **HEFT** [Augonnet et al., 2009] uma estratégia não-adaptativa que se beneficia de uma política de predição de tempo baseada em histórico. (3) **HEFT data-aware** [Augonnet et al., 2009] uma estratégia não-adaptativa que pondera sua política de predição com o tempo de transferência de dados entre unidades de processamento. (4) ***Work Stealing*** [Blumofe & Leiserson, 1999] uma estratégia adaptativa baseada em roubo de tarefas por unidades de processamento ociosas.

- **HEFT (Heterogeneous Earliest-Finish Time)**: A divisão 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 durante o processamento da aplicação, e dessa forma atribui uma tarefa a

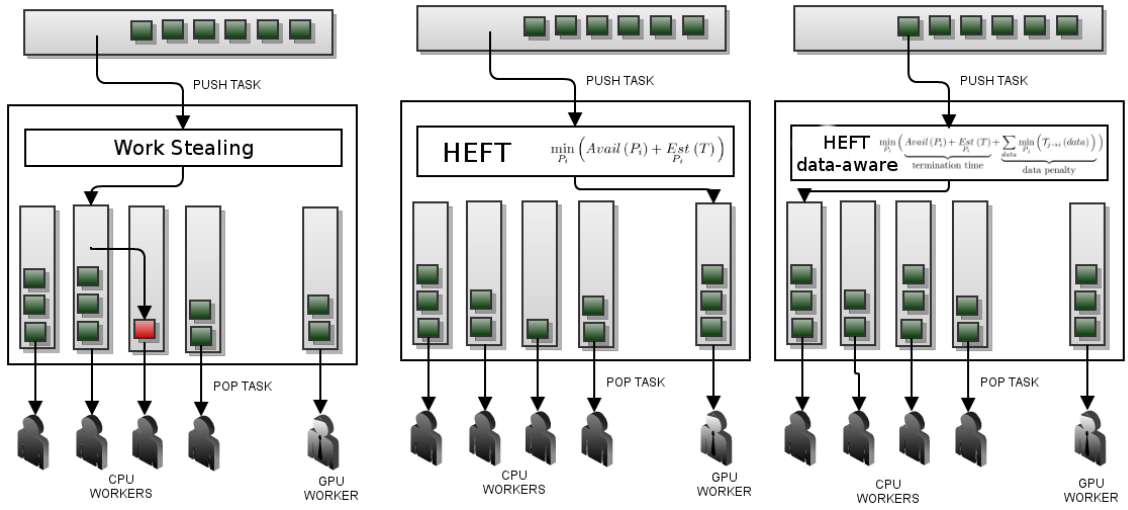


Figura 3.2: Estratégias de Escalonamento Dinâmico: HEFT, HEFT *data-aware* e *Work Stealing*

unidade de processamento que minimize o tempo de término da tarefa (MCT). O tempo de execução de uma tarefa é encontrado no histórico de execuções por meio da similaridade entre os parâmetros. A escolha é feita baseada na UP que minimize a seguinte a Equação 4.1

$$mim_{Pi}(Avail(Pi) + Est_{Pi}(T)) \tag{3.1}$$

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.

É um escalonador não-adaptativo, já que não realiza movimentações de tarefas após a associação nas unidades de processamento, e sua principal característica é lidar com a boa adequação das tarefas baseada na métrica Minimun Completion Time (MCT).

- **HEFT *data-aware* (*Heterogeneous Earliest-Finish Time Data Aware*)**: bastante similar à estratégia *HEFT*, uma vez que também utiliza um modelo baseado no desempenho das unidades de processamento em tarefas anteriores, mantendo o mesmo histórico de execução. Além disso, a estratégia *HEFT data-awere* considera também o tempo de transferência de dados entre a memória principal e a memória das UPs. A escolha de qual unidade de processamento executará uma determinada tarefa é feita baseada naquela que minimize a Equação 3.2

$$min_{Pi}(Avail(Pi) + Est_{Pi}(T) + \sum_{data} min_{Pj}(\tau_{j \to i}(data))) \quad (3.2)$$

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

É um escalonador não-adaptativo, já que não realiza movimentações de tarefas após a associação nas unidades de processamento, e sua principal característica é lidar com a boa adequação das tarefas baseada na métrica Minimun Completion Time (MCT) e na ponderação pelo volume de dados utilizados na transferência entre as unidades de processamento.

- **WS (*Work Stealing*)**: nessa estratégia são mantidas várias filas de tarefas, uma para para cada UP. A medida que surgem novas tarefas dinamicamente, as mesmas são distribuídas entre as filas seguindo uma estratégia gulosa de *round robin*. Quando uma unidade de processamento está com sua fila vazia, ou seja ociosa, o escalonador verifica qual recurso está mais sobrecarregada e retira uma ou mais tarefas da fila de dessa unidade de processamento e insere na fila de execução da UP ociosa, constituindo seu perfil adaptativo, no qual existe uma movimentação de tarefas após a associação delas a um recurso de processamento. A principal característica desse escalonador é lidar com balanceamento de carga, ao realizar as movimentações de tarefas para as unidades ociosas.

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 de previsão do tempo de execução, ou seja, de se construir um histórico de execução, seja “online"ou “offline", bem representativo e acurado. Além disso, por meio do roubo de tarefas, essa estratégia consegue lidar muito bem com o balanceamento de cargas, visto que sempre uma unidade de processamento estará executando alguma tarefa. Entretanto, essa estratégia pode fazer atribuições equivocadas de tarefas a unidades de processamento, visto que a associação inicial não utiliza um método de predição de performance, penalizando significativamente o tempo total de execução. Por outro lado, as estratégias *HEFT* e *HEFT data-aware* são ótimas opções quando os comportamentos das tarefas de uma aplicação são bem conhecidos, por meio de execuções anteriores, sendo que criar um histórico, “online"ou “offline", de execuções bem representativo e acurado é possível. Tal histórico tende a aproximar bem o tempo

de execução da tarefa na unidade de processamento por meio de execuções passadas, porém é uma estratégia que demanda uma calibragem (construção e preenchimento do histórico) durante a execução (ou antes dela) da aplicação, agregando um *overhead* que pode não ser compensado pela boa atribuição de uma tarefa em uma UP, visto que um histórico de execuções passadas pode não ser acurado e representativo o suficiente. Uma outra questão dessa estratégias que se beneficiam de um modelo de predição é, em muitos casos, não lidar diretamente com o balanceamento de cargas, podendo assim sobrecarregar uma unidade de processamento com sucessivas associações de tarefas.

3.3.2 Cenários de testes

Nessa subseção descrevemos as cargas de trabalho que serão utilizadas para avaliarmos o comportamento dos escalonadores dinâmicos implementados. Conforme veremos nos tópicos 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.

3.3.2.1 Aplicações Sintéticas

Em uma arquitetura híbrida como a utilizada nesse trabalho, compostas por várias *CPUs* e *GPUS*, um fator determinante para a auxiliar na tomada de decisão 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:

- **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 3.1.

Tabela 3.1: Cargas de Trabalho.

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

Por fim, para cada uma das seis cargas de trabalho apresentadas na Tabela 3.1, 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).

3.3.2.2 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).

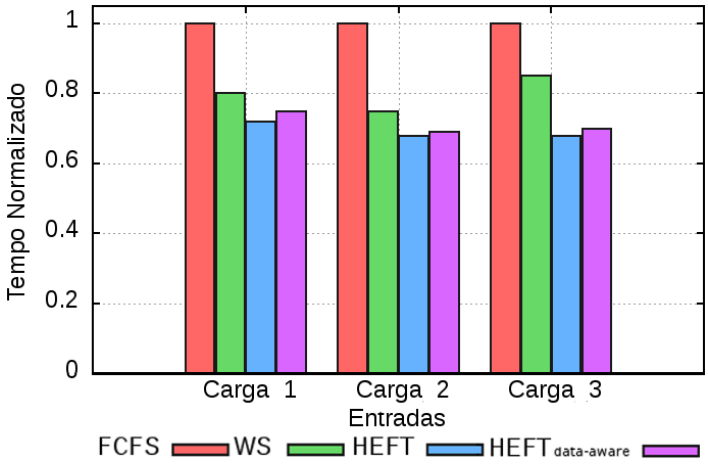
3.3.3 Resultados e discussões

Nessa subseção apresentaremos e discutiremos os resultados das avaliação de estratégias de escalonamento dinâmico tradicionais apresentadas. 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

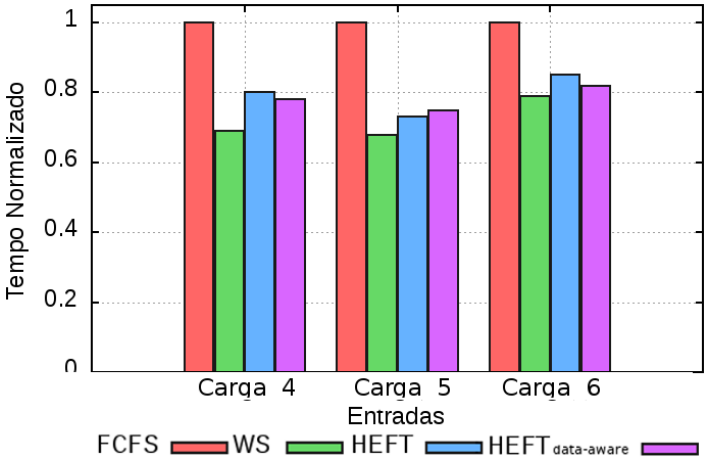
cenário de avaliação, foram geradas 500 tarefas independentes e os tempos obtidos são referentes a uma média de 10 execuções.

3.3.3.1 Cenários sem transferência

O primeiro conjunto de experimentos realizados são referentes a execução das seis cargas de trabalho sintéticas descritas na Tabela 3.1, 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 4.2.



(a) SpeedUp Relativo Alto



(b) SpeedUp Relativo Baixo

Figura 3.3: Avaliação dos Escalonadores em Cenários sem Transferência de Dados

Observando os gráficos, vamos considerar, apenas o comportamento dos escalonadores *WS*, *HEFT* e *HEFT data-aware* para ambos os cenários, já que o escalonador FCFS por ser um escalonador de natureza simples e aleatória não obteve bons resultados em nenhum cenário.

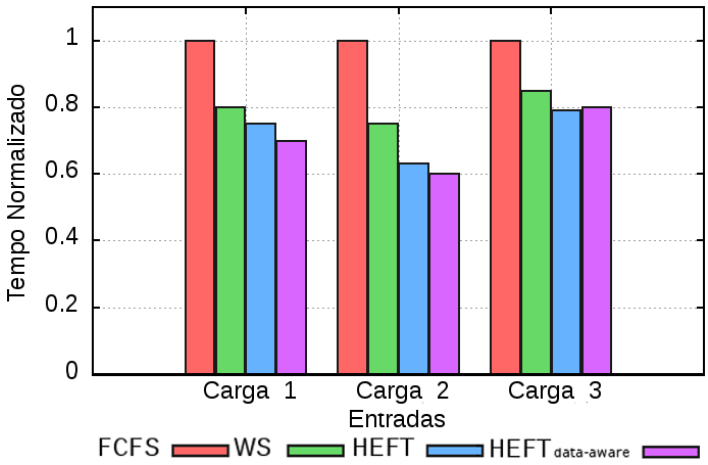
Para o cenário onde há uma incidência maior de tarefas cujo o *SpeedUp relativo* é baixo, temos que o escalonador *WS* apresentou resultados superiores aos do *HEFT* e *HEFT data-aware*. Nesse contexto, os tempos de execuções das tarefas em CPU e GPU apresentam uma diferença muito pequena, sendo que caso o escalonador realize um associação inadequada da tarefa à unidade de processamento, não ocorrerá perda significativa no desempenho da aplicação.

Dessa forma, tendo tarefas que a boa adequação não é um requisito crucial ao desempenho da aplicação, o balanceamento de cargas torna-se um fator determinante para a melhor organização das tarefas entre os recursos disponíveis. Portanto a estratégia de "roubar" as tarefas de outras UPs, mesmo que não muito adequadas para UP de destino, se torna uma boa opção, e apresenta melhores resultados.

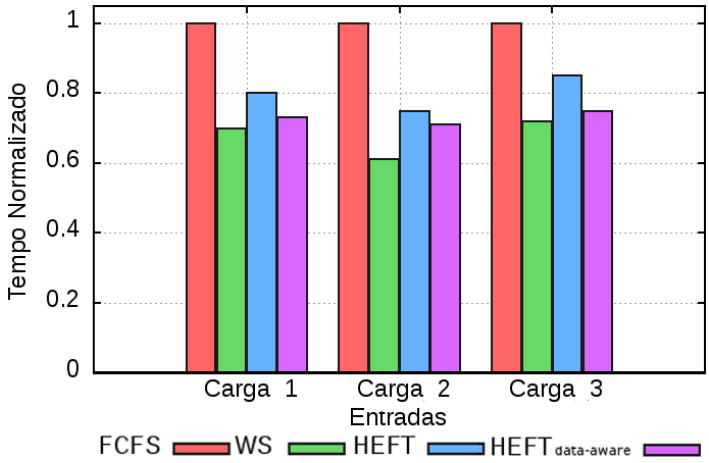
Além disso, o *overhead* gerado para se estimar os tempos de execuções para adequar uma tarefa à melhor UP não irá compensar a minimização do tempo de execução da tarefa. Por outro lado, no cenário em que há uma incidência maior de tarefas cujo o *SpeedUp relativo* é alto, os escalonadores baseados em 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 das estimativas dos tempos de execuções das tarefas 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, já que não adianta as cargas estarem balanceadas, se não estiverem bem adequadas nas unidades de processamento que minimizem os tempos de execuções das tarefas

3.3.3.2 Cenários com transferência

Em nosso segundo conjunto de experimentos, consideramos novamente cada uma das seis cargas apresentadas na seção 3.3.2.1, 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 4.3.



(a) SpeedUp Relativo Alto



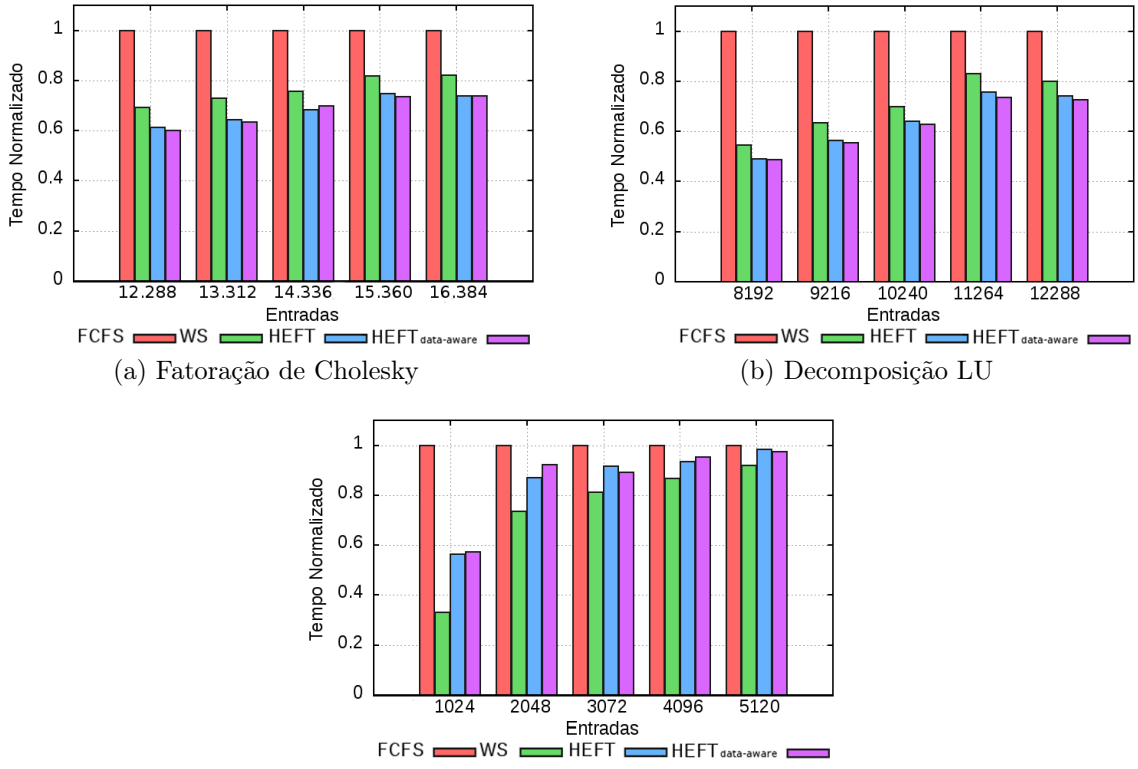
(b) SpeedUp Relativo Baixo

Figura 3.4: Avaliação dos Escalonadores em Cenários com Transferência de Dados

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.

3.3.3.3 Cargas Reais

Por fim, nosso último conjunto de experimentos diz respeito a avaliação dos escalonadores em cenários de cargas reais apresentados na seção 3.3.2.2: 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 4.4 apresentamos os resultados alcançados por cada escalonador.



(a) Fatoração de Cholesky

(b) Decomposição LU

(c) Gradiente Conjugado

Figura 3.5: Avaliação dos Escalonadores em Aplicações Reais

Observando os resultados, temos que o escalonador *WS* alcançou o melhor desempenho para a aplicação de Gradiente Conjugado. Por outro lado, o escalonador *HEFT data-aware* 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.

3.4 Sumário

Nessa seção foi apresentado um detalhado estudo sobre estratégias de escalonamento. Mostramos diferentes grupos de algoritmos, organizados pela forma com que tomam decisões e pelas características das tarefas que estão sendo escalonadas. Mais especificamente apresentamos o problema de escalonamento de tarefas em arquiteturas híbridas, foco deste trabalho. Foram expostas divisões desse grupo de escalonadores, adaptativos e não-adaptativos, e uma analise experimental foi realizada para compreendermos o comportamento desse tipo de escalonamento.

Foram implementadas no ambiente de execução StarPU estratégias de escalonamento dinâmico não-adaptativas: FCFS, HEFT e HEFT *data-aware*, e uma estratégia de escalonamento dinâmico adaptativa: *Work Stealing*. Por meio de uma análise experimental utilizando um cenário controlado de aplicações e um cenário com aplicações reais, algumas conclusões podem ser extraídas:

1. Politicas de escalonamento que utilizam estratégias de predição de tempo de execução, visando a boa adequação das tarefas nas unidades de processamento disponíveis, como no caso dos escalonadores HEFT e HEFT *data-aware*, tendem a ter melhor desempenho em aplicações que possuem tarefas com grande diferença de performance nas arquiteturas de processamento disponíveis. Nesse contexto, a boa adequação de tarefas é fundamental para que não se tenha excesso de trabalho. Porém, quando temos aplicações em que suas tarefas possuem desempenhos bastante similares nas arquiteturas presentes para processamento, a predição de tempo para boa adequação de tarefas, torna-se um trabalho não tão fundamental, gerando apenas *overhead* de processamento, prejudicando o desempenho da aplicação.

2. Situações em que uma unidade de processamento é eficiente o suficiente para executar mais rápido a maioria das tarefas de uma aplicação, pode resultar em várias associações de tarefas à essa unidade de processamento, por meio da predição de tempos de execução, como utilizadas pelo HEFT e HEFT *data-aware*. Em consequência, essa unidade de processamento fica sobrecarregada e outras ociosas, o que impacta diretamente na queda do desempenho da aplicação.

3. A politica de escalonamento *Work Stealing*, é uma estratégia interessante quando a diferença de performance das tarefas nas unidades de processamento disponíveis é bem pequena. Como o WS é uma estratégia focada no balanceamento de cargas, nesse contexto uma associação inadequada de tarefas à unidade de processamento não gera perda de desempenho expressivo. Sendo que a boa divisão

dessas tarefas entre os processadores disponíveis, uma característica fundamental para se manter a taxa de processamento similar entre os processadores, aumentará o paralelismo da aplicação e consequentemente reduzirá o tempo de execução final. Porém, nos cenários em que é expressiva a diferença de desempenho de uma tarefa nas unidades de processamento, a boa adequação das tarefas é um passo muito importante, e dessa forma a estratégia *Work Stealing* não alcança bons resultados.

4. A consideração da transferência de dados na predição do tempo de execução das tarefas, como no caso do escalonador HEFT *data-aware*, é um fator impactante no desempenho do escalonador, quando temos cenários de aplicações que manipulam quantidades expressivas de dados.

Por meio dessas observações, podemos concluir que nos cenários em que o *Work Stealing* é ineficiente, se houvesse uma predição de tempo de execução inicial, como feita pelo HEFT e HEFT *data-aware*, o desempenho do escalonamento poderia ser superior. Da mesma forma, no contexto em que HEFT e HEFT *data-aware* são ineficientes, se houvesse o balanceamento de carga adaptativo, como o realizado pelo *Work Stealing*, o desempenho do escalonamento também seria superior. Dessa forma, no próximo capítulo será apresentado duas novas estratégias de escalonamento, ambas construídas combinando diferentes abordagens de escalonadores dinâmicos, não-adaptativos e adaptativos, tendo como base os escalonadores tradicionais estudados: HEFT, HEFT *data-aware* e *Work Stealing*. Essas novas estratégias de escalonamento visam, por meio da combinação de características, apresentar um melhor desempenho todos os contextos de aplicações apresentados, comparado aos escalonadores base.

Capítulo 4

Escalonamento Adaptativo

No capítulo anterior apresentamos uma avaliação experimental na qual comparamos o comportamento de algumas estratégias de escalonamento dinâmico não-adaptativa de tarefas em arquiteturas híbridas: FCFS, HEFT e HEFT *data-aware*, e uma estratégia de escalonamento dinâmico adaptativa: *Work Stealing*. Conforme as estratégias de escalonamento tendem a ter diferentes desempenhos. Basicamente extraímos as seguintes conclusões: (1) Para aplicações com tarefas apresentando expressiva diferença de desempenho entre as unidades de processamento, escalonadores que se beneficiam de estratégias de predião de tempo de execução, visando a boa adequação das tarefas nos recursos; (2) existe sobrecarga de trabalho para unidades de processamento suficientemente rápidas quando utilizamos escalonadores baseados em predição de tempo; (3) quando as tarefas de uma aplicação apresentam uma pequena diferença de desempenho entre as unidades de processamento, a estratégia de roubo de tarefas é mais interessante visto o balanceamento de cargas realizado; (4) considerar a transferência de dados na predição do tempo de execução impacta na melhora do desempenho da aplicação em cenários o qual a quantidade de dados manipulados é expressiva.

Por meio dessas observações, nesse capítulo propomos duas novas estratégias de escalonamento, ambas construídas combinando diferentes abordagens de escalonadores dinâmicos, não-adaptativos e adaptativos, estudados na análise experimental apresentada na seção 3.3. Basicamente, os escalonadores propostos combinam essas abordagens : predição do tempo de execução de tarefas nas arquiteturas disponíveis (estratégia não-adaptativa) e “roubo"de tarefas (estratégia adaptativa), constituindo um escalonador dinâmico adaptativo com bom desempenho em todos os contextos de aplicações apresentados.

Para avaliar nossas propostas, utilizamos os mesmos cenários apresentados na subseção 3.3.2, comparamos os resultados alcançados com os escalonadores nos quais

elas foram baseadas, todos implementados no StarPU [Augonnet et al., 2009]. 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.

4.1 Escalonadores Adaptativos Propostos

Baseados nos escalonadores WS, HEFT e HEFT data-aware descritos na subseção 3.3.1, nessa seção descrevemos nossas duas novas abordagens, as quais combinam os aspectos principais dos escalonadores de modelos *HEFT* e *HEFT data-aware*, com a característica de "roubo de tarefas"do escalonador *WS*. Nos tópicos abaixo descrevemos cada um dos escalonadores proposto.

- ***Adaptative HEFT (A-HEFT):*** Este escalonador combina as características dos escalonadores *HEFT* e *WS*. Basicamente o escalonador mantém uma fila de tarefas para cada unidade de processamento e, similarmente ao *HEFT*, a distribuição das tarefas entre as várias filas é baseada no recurso que minimize a Equação 4.1

$$mim_{Pi}(Avail(Pi) + Est_{Pi}(T)) \tag{4.1}$$

  Baseando-se no escalonador *WS*, quando uma unidade de processamento está ociosa, o escalonador retira uma ou mais tarefas da fila de outro processador (a que está mais sobrecarregada) e insere na fila do processador ocioso.

- ***Adaptative HEFT data-aware (A-HEFT data-aware):*** Essa politica de escalonamento combina característica dos escalonadores *HEFT data-aware* e *WS*. Cada UP possui uma fila de tarefas e a distribuição das mesmas nessas filas se dá pela minimização da Equação 4.2, 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.

$$min_{Pi}(Avail(Pi) + Est_{Pi}(T) + \sum_{data} min_{Pj}(\tau_{j \to i}(data))) \tag{4.2}$$

Dessa forma, os dois escalonadores propostos agregam características, não-adaptativas e adaptativas, que tentam resolver os principais problemas do escalona-

mento dinâmico de tarefas em arquiteturas híbridas: A boa adequação de tarefas à unidade de processamento que minimize seu tempo de execução e o balanceamento de cargas. Enquanto o escalonado *A-HEFT* se preocupa com a boa adequação das tarefas nas unidades de processamento e também com a sobrecarga de trabalho, o escalonador Â-HEFT data-aware vai além, considerando também a informação da quantidade de dados utilizados que serão transferidos, evitando assim perdas de desempenho por conta do *overhead* da transferência.

A intuição por trás dos escalonadores propostos é iniciar a associação das tarefas às unidades de processamento da melhor forma possível, por meio da estimativa dos tempos de execuções, e de forma adaptativa, resolver o balanceamento de carga entre as unidades de processamento disponíveis, por meio do "roubo de tarefas". Além de que, outro problema de sobrecarga de tarefas gerado por várias associações à uma unidade de processamento eficiente, apresentado no segundo item das considerações iniciais deste capítulo, pode ser amenizado pelo "roubo de tarefas", o qual adaptativamente redistribui tarefas entre os processadores disponíveis, de forma a balancear a carga e consequentemente aumentar o desempenho final da aplicação.

Nossa expectativa é que, dessa forma, as nossas abordagens sejam capazes de generalizar melhor, se adaptando a cenários mais variados. Para comprovar nossas hipóteses, na seção seguinte avaliamos nossas estratégias nos cenários de aplicação apresentados na subseção 3.3.2 do capítulo 3, contrastando os resultados obtidos por elas com os resultados das estratégias originais. A figura 4.1 ilustra estas politica de escalonamento propostas.

4.2 Avaliação Experimental

Para avaliar nossas propostas, utilizamos os mesmos cenários apresentados na subseção 3.3.2, comparamos os resultados alcançados com os escalonadores nos quais elas foram baseadas, todos implementados no StarPU [Augonnet et al., 2009], ambiente de execução detalhado detalhado na subseção 2.2.1, considerando como recursos de processamento CPUs e GPUs.

4.2.1 Resultados e discussões

Nessa subseção apresentaremos e discutiremos os resultados das avaliação das estratégias de escalonamento dinâmico propostas contrastadas com as estratégias tradicionais apresentadas.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

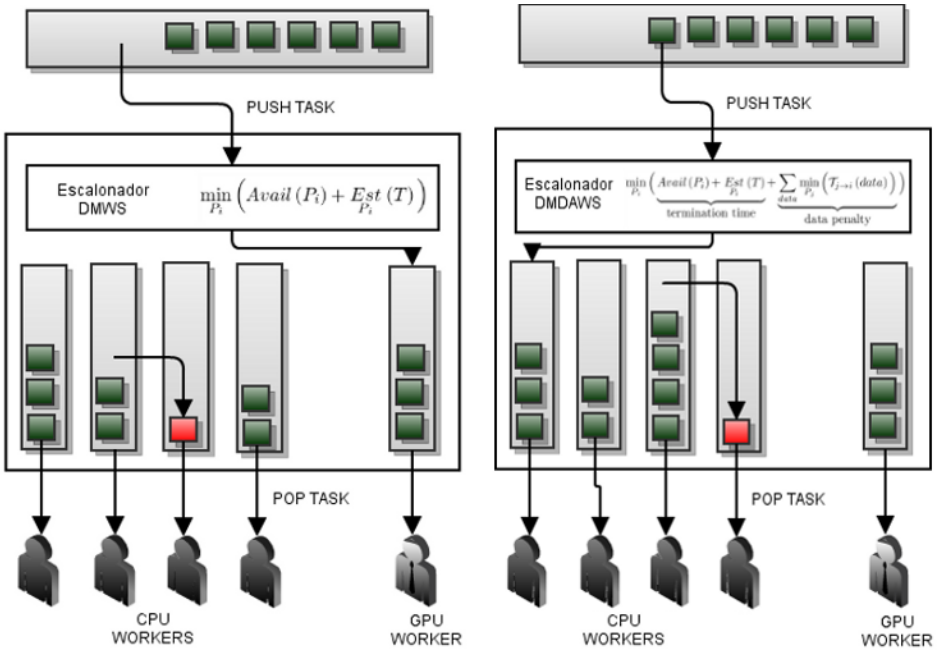


Figura 4.1: Estratégias de Escalonamento Propostas: A-HEFT e A-HEFT data-aware

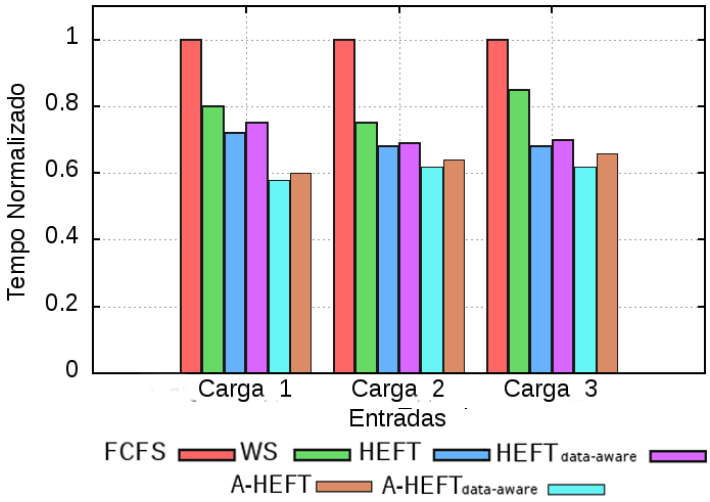
1Gb de RAM. Para cada cenário de avaliação, foram geradas 500 tarefas independentes e os tempos obtidos são referentes a uma média de 10 execuções e são apresentados de forma normalizada em função do pior tempo de execução dentre os escalonadores avaliados.

4.2.1.1 Cenários sem transferência

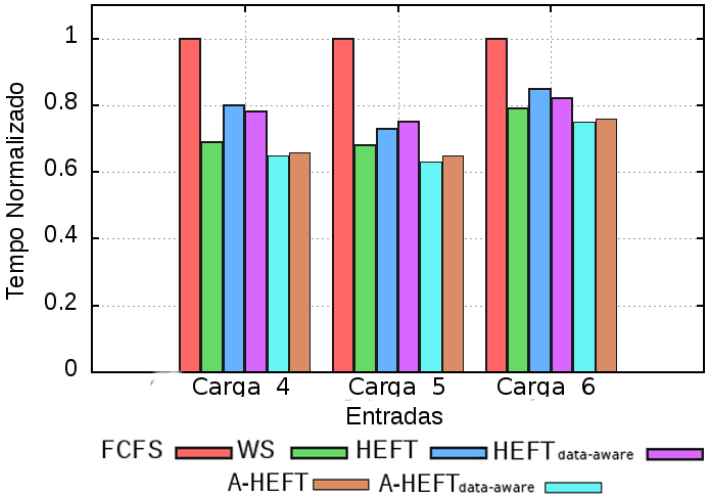
Observando o comportamento dos escalonadores *A-HEFT* e *A-HEFT data-aware* propostos nesse trabalho, temos que ambos alcançaram os melhores resultados nos dois cenários, sendo até 10% mais eficientes.

Esse resultado mostra que, no cenário em que a combinação entre estratégias de escalonamento que utilizam a predição do tempo de execução com estratégias mais simples, como "rouba" de tarefas, pode ser bem vantajoso, se adaptando melhor a diferentes cenários de aplicações.

Podemos afirmar que no contexto em que o a boa adequação é necessário, ou seja o *SpeedUP Relativo* é alto, a característica de predição do tempo de execução prevalece como fundamental, aliada ao balanceamento de carga executado pelo Work Stealing, proporciona maior desempenho aos escalonadores propostos. Da mesma forma, no contexto em que a boa adequação não é prioridade, ou seja o *SpeedUP Relativo* é baixo, o balanceamento de carga realizado pelo "roubo de tarefas", aliado a adequação mais consistente das tarefas realizada pela predição do tempo de execução, também dá



(a) SpeedUp Relativo Alto



(b) SpeedUp Relativo Baixo

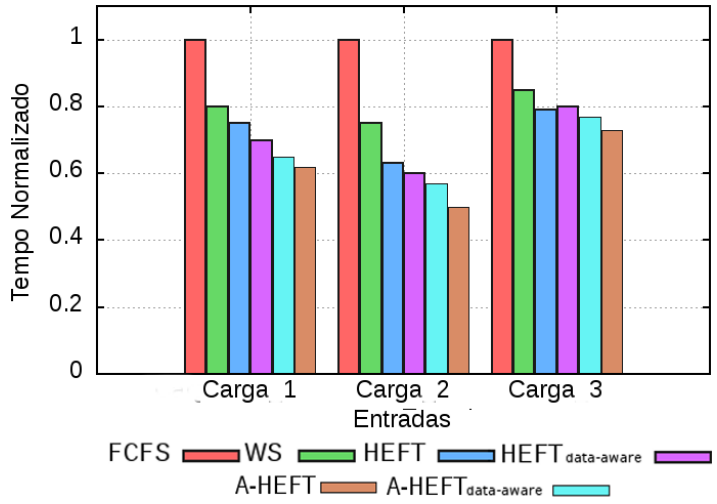
Figura 4.2: Avaliação dos Escalonadores em Cenários sem Transferência de Dados

às nossas propostas o melhor desempenho.

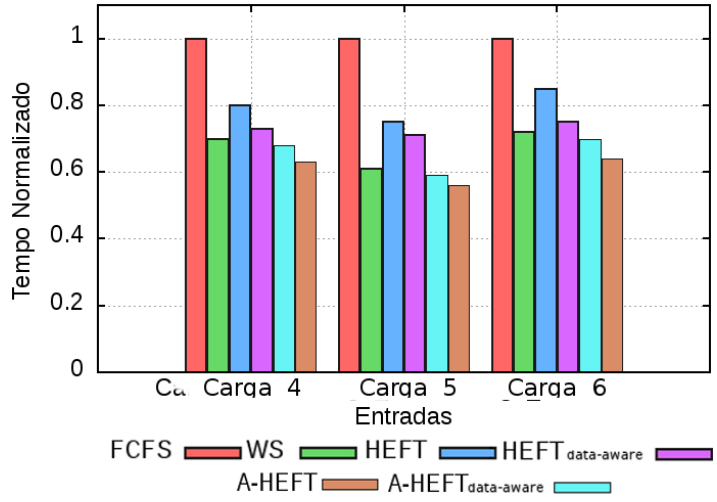
Sendo assim, a combinação de características de diferentes estratégias reduz os problemas individuais de cada uma delas nos cenários apresentados, tornando as políticas propostas com bons desempenhos constantes nos cenários de aplicações apresentados.

4.2.1.2 Cenários com transferência

Observamos que, mais uma vez, os escalonadores propostos nesse trabalho (o *A-HEFT* e o *A-HEFT data-aware*) apresentaram os melhores resultados em quase todos os cenários avaliados, com grande destaque ao *A-HEFT data-aware*, que em alguns casos foi até 20% mais eficiente, visto que considera a transferência de dados, fator que para



(a) SpeedUp Relativo Alto



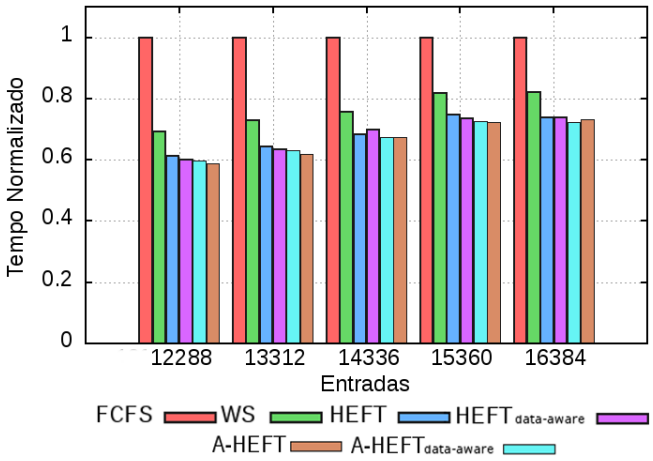
(b) SpeedUp Relativo Baixo

Figura 4.3: Avaliação dos Escalonadores em Cenários com Transferência de Dados

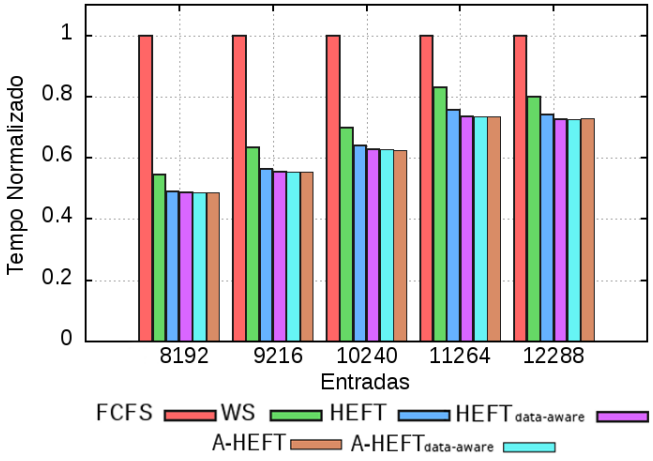
esse cenário é impactante.

4.2.1.3 Cargas Reais

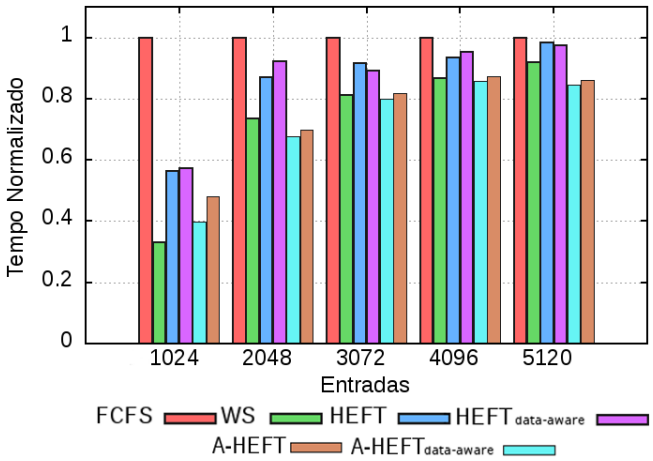
Por fim, para as cargas reais, aplicações não controladas, os escalonadores propostos nesse trabalho apresentaram os melhores resultados, mostrando mais uma vez que nossa abordagem de combinar diferentes estratégias de escalonamento é uma boa alternativa e genérica o suficiente para ser aplicada em diversos cenários.



(a) Fatoração de Cholesky



(b) Decomposição LU



(c) Gradiente Conjugado

Figura 4.4: Avaliação dos Escalonadores em Aplicações Reais

4.3 Sumário

Nesse capítulo propomos duas novas estratégias de escalonamento dinâmico, construídas combinando as características de diferentes estratégias já existentes. Dessa forma, o objetivo foi 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 (A-HEFT) ou não (A-HEFT data-aware) o tempo de transferência de dados entre memória principal e memória das unidades de processamento.

Avaliamos as políticas de escalonamento propostas em diferentes cenários, sintéticos e reais. Nossos resultados mostram que, enquanto em alguns cenários as abordagens baseadas somente em modelos de predição se mostram mais adequadas, 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 as 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.

Nesse capítulo as discussões e propostas sobre escalonamento dinâmico de tarefas em arquiteturas híbridas, tiveram como base uma arquitetura composta apenas por CPU *multicore* e coprocessadores GPU. Entretanto, nas arquiteturas híbridas atuais, não só GPUs tem se popularizado no contexto de coprocessamento massivamente paralelo, mas como também outros coprocessadores vem se destacando como interessantes alternativas, um exemplo é o coprocessador Intel Xeon Phi (MIC). Dessa forma, no próximo capítulo será apresentado um novo estudo sobre estratégias de escalonamento em arquiteturas híbridas, considerando mais um novo recurso de processamento além da CPU e da GPU: o Intel Xeon Phi.

Além da inserção de mais um recurso de processamento, no próximo capítulo avaliaremos a utilização de uma abordagem baseada em performance (Speedup) para a tomada de decisão dos escalonadores propostos e a utilização de réplicas de tarefas. Diferentemente dos escalonadores não-adaptativos apresentados nesse e no capítulo anterior, os quais utilizam a predição do desempenho de uma tarefa em uma determinada arquitetura por meio da estimativa do tempo de execução, no próximo capítulo iremos utilizar a estimativa do speedup da tarefa nos recursos disponíveis para adequá-las corretamente. Avaliaremos assim, qual dessas abordagens atinge melhor desempenho e qual delas é mais sensível à presença de erros nas estimativas. Adicionalmente, no

próximo capítulo, também será utilizada e avaliada uma nova abordagem adaptativa, a qual diferente do roubo de tarefas apresentada nos capítulos anteriores, se baseia na replicação de tarefas entre os recursos disponíveis.

Capítulo 5

Escalonamento Genérico em Arquiteturas Heterogêneas

Nos capítulos anteriores apresentamos uma extensa discussão sobre escalonamento dinâmico de tarefas em arquiteturas híbridas. Avaliamos o comportamento de diferentes escalonadores dinâmicos adaptativos e não-adaptativos presentes na litetura por meio de diferentes cenários de aplicações. Beseados nessas observações, no capítulo anterior propomos duas novas estratégias de escalonamento dinâmico de tarefas combinando características adaptativas e não-adaptativas dos escalonadores tradicionais, alcançando políticas de escalonamento que se tornam favoráveis à um leque mais completo de aplicações.

Todas as discussões e propostas apresentadas nos capítulos anteriores sobre escalonamento dinâmico de tarefas em arquiteturas híbridas, tiveram como base uma arquitetura composta apenas por CPU multicore e coprocessadores GPU. Entretanto, nas arquiteturas híbridas atuais, não só GPUs tem se popularizado no contexto de coprocessamento massivamente paralelo, mas como também outros coprocessadores vem se destacando como interessantes alternativas, um exemplo é o coprocessador Intel Xeon Phi (MIC). Dessa forma, nesse capitulo apresentamos um novo estudo sobre estratégias de escalonamento em arquiteturas híbridas, considerando mais um novo recurso de processamento além da CPU e da GPU: o Intel Xeon Phi (MIC).

Essa nossa nova abordagem para escalonamento dinâmico de tarefas em arquiteturas híbridas visa avaliar e propor escalonadores dinâmicos, adaptativos e não-adaptativos, para a utilização eficiente de uma arquitetura moderna, composta por CPU multicores e coprocessadores diferentes: GPU e MIC.

Nesse capítulo avaliamos não só a inserção de mais uma unidade de processamento, mas como também avaliamos a utilização de uma abordagem baseada em perfor-

mance (Speedup) para a tomada de decisão dos escalonadores propostos e a utilização de réplicas de tarefas. Diferentemente dos escalonadores não-adaptativos apresentados nos capítulos 3 e 4, os quais utilizam a predição do desempenho de uma tarefa em uma determinada arquitetura por meio da estimativa do tempo de execução, nesse capítulo utilizamos a estimativa do speedup da tarefa nos recursos disponíveis para adequá-las corretamente. Avaliamos qual dessas abordagens atinge melhor desempenho e qual delas é mais sensível à presença de erros nas estimativas. Por fim, também utilizamos uma nova abordagem adaptativa, a qual diferente do roubo de tarefas apresentada nos capítulos anteriores, se baseia na replicação de tarefas entre os recursos disponíveis.

Para a implementação e avaliação dos escalonadores propostos utilizaremos o ambiente Extreme DataCutter, detalhado na subseção 2.1.2, o qual apresenta suporte para a execução de aplicações *dataflow* em arquiteturas híbridas composta por CPUs, GPUs e MICs.

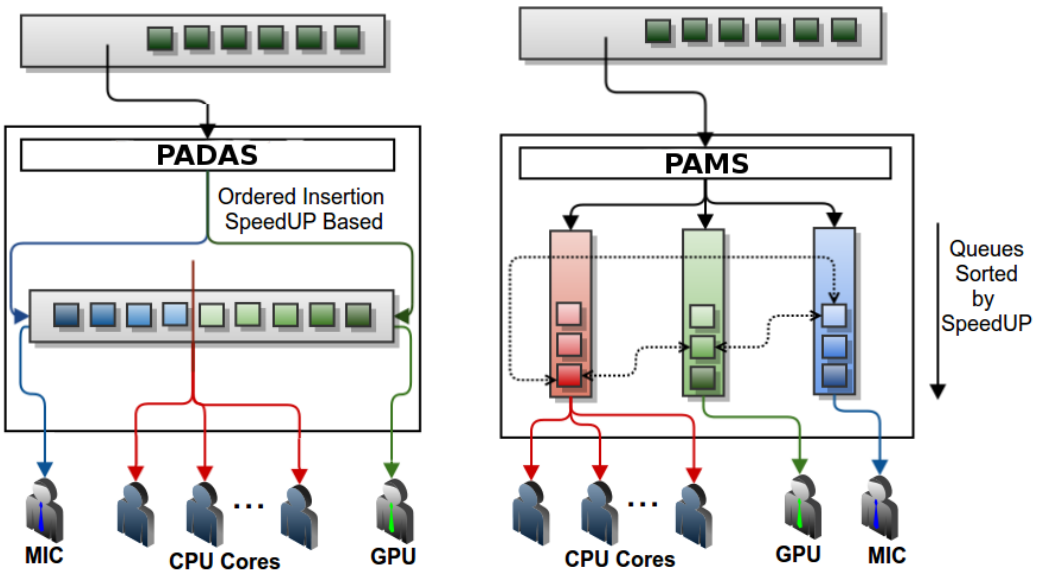
5.1 Escalonamento Dinâmico Adaptativo baseados em Performance

Nessa seção apresentaremos nossas políticas de escalonamento dinâmico para a execução eficiente em sistemas híbridos com múltiplos aceleradores. Como já mencionado nesse trabalho, as diferentes tarefas que compõe uma aplicação podem atingir desempenhos variados de acordo com a arquitetura de processamento usada. Dessa forma, utilizamos abordagens de predição de desempenho de uma tarefa dada uma unidade de processamento, para melhor utilizar um sistema híbrido com múltiplos processadores.

Nos capítulos 3 e 4, apresentamos estratégias não-adaptativas e adaptativas de escalonamento dinâmico de tarefas, que utilizam uma abordagem de estimativa do tempo de execução de tarefas, para predição do desempenho em uma determinada unidade de processamento. Nessa seção apresentaremos uma outra abordagem de predição de desempenho, a qual utiliza a estimativa da performance (Speedup) de uma tarefa nos recursos disponíveis.

Da mesma forma que no Capítulo 4, visamos combinar a boa adequação das tarefas às unidades de processamento e alcançar bom balanceamento de carga. Para tanto, em um dos nossos escalonadores propostos utilizamos, cooperativamente com a predição de desempenho baseada no speedup, uma estratégia adaptativa baseada em replicações de tarefas, a qual, similarmente ao roubo de tarefas apresentado nos capítulos 3 e 4, visa deixar sempre as unidades de processamento trabalhando, evitando a ociosidade de processadores.

Os escalonadores propostos, descritos em sequência, foram implementados no módulo WRM do Extreme DataCutter, e usado para escalonar operações de grão-fino do *dataflow*, criadas em cada estágio para diferentes processadores: CPUs, GPUs e MICs.



(a) PADAS (b) PAMS

Figura 5.1: Estratégias de Escalonamento Propostas.

- ***Performance-Aware Dequeue Adapted Scheduler* (PADAS):** Primeira estratégia de escalonamento proposta possui característica unicamente não-adaptativa. Foi implementada utilizando uma fila global de tarefas ordenadas pelos seus speedups esperados nas unidades de processamento presentes. Se o speedup esperado no MIC é maior que o speedup esperado em GPU, a tarefa é inserida de forma ordenada na "cabeça"da fila global, sendo alocada exatamente na frente da primeira tarefa encontrada com speedup esperado menor. A mesma abordagem é usada quando o speedup na GPU é maior que o speedup do MIC. Nesse caso, a inserção inicia-se na outra extremidade da fila global, a calda, e a tarefa é inserida exatamente na frente da tarefa com menor speedup na GPU.

  Essa estratégia resumidamente organiza as tarefas em uma única fila, de forma que tarefas mais apropriadas à serem executadas em MIC estejam em uma extremidade da fila, enquanto tarefas mais apropriadas à serem executadas em GPU estejam na outra extremidade. Da mesma forma, que os escalonadores apresentados nesse trabalho, a associação de uma tarefa à uma unidade de processamento é feita de forma sob demanda, sendo que quando uma unidade de processamento se torna disponível, a *thread* que administra esse recurso, busca a próxima tarefa

a ser executada. Se um dispositivo MIC está disponível, a primeira tarefa da cabeça da fila global é selecionada e associada pra execução no MIC. Seguindo o mesmo raciocínio, quando um dispositivo GPU está disponível, uma tarefa da calda da fila global é selecionada e associada para a execução. Quando o recurso disponível para a execução é um núvleo CPU, a tarefa que está exatamente no meio da fila é selecionada para a execução. O ponto exato do meio da fila indica o limite entre as tarefas inseridas para MIC e para GPU (ver Figura 5.1). Essa abordagem visa assinalar tarefas que apresentam baixo desempenho nos acelerados presentes, para os núcleos CPU.

- ***Performance-Aware Multiqueue Scheduler* (PAMS):** Esse escalonador proposto é uma variação da primeira estratégia, generalizando a organização para vários acelerados, não só GPU e MIC, possui uma característica adaptativa ao utilizar a replicação de tarefas. Basicamente cria-se uma fila ordenada para cada tipo de dispositivo presente. Por exemplo, se um processador CPU *multicore* e aceleradores GPU e MIC estão sendo usados cooperativamente em uma execução, três filas diferentes são instanciadas. Além disso, cada tarefa é inserida em todas as filas, e ordenadas de forma decrescente de acordo com o speedup esperado no dispositivo que a fila está representando. A fila que representa a CPU é ordenada de acordo com o maior valor de speedup da tarefa nos acelerados, seguindo a fórmula: $1/max(GPU\ Speedup, MIC\ Speedup)$. O objetivo, novamente, é usar os núcleos CPU para executar tarefas que tenham baixo desempenho em todos os outros dispositivos. Por fim, quando uma tarefa é requisitada, pelas unidades de processamento, a tarefa no início da fila relativa ao dispositivo (maior speedup para essa unidade de processamento) é recuperada, e essa tarefa é então removida de todas as outras filas. Para rapidamente acessar e remover tarefas de outras filas, são mantidos ponteiros referenciando a localização dessas tarefas nas filas vizinhas (ver Figura 5.1).

A estratégia de organizar tarefas em filas ordenadas pelo speedup nas unidades de processamento, procura adequar tarefas nos recursos que as execute de forma mais eficiente. Adicionalmente, o escalonador PAMS apresenta uma característica adaptativa que reduz o problema de ociosidade das unidades de processamento, mantendo cópias de uma tarefa entre várias filas. É uma característica adaptativa, já que dinamicamente apaga-se réplicas quando a execução da tarefa foi finalizada mais rapidamente.

As políticas de escalonamento propostas utilizam como métrica de predição de desempenho o speedup esperado. Assim, se a estimativa do speedup for acurada o

suficiente para manter a ordem das tarefas nas filas, o desempenho do escalonador não é afetado.

5.2 Avaliação Experimental

Nessa seção apresentamos a avaliação experimental dos escalonadores e estratégias propostas. Primeiramente será apresentado detalhadamente a aplicação base para todas as análises experimentais e em sequência os resultados e discussões de cada experimento realizado.

Para uma avaliação completa, foram implementados no Extreme DataCutter outras políticas de escalonamento tradicionais para a comparação com as nossas propostas baseadas em performance: First Come, First Served (FCFS) e Heterogeneous Earliest Finish Time (HEFT), ambas descritas na subseção 3.3.1. Para o HEFT, foi mantido um histórico de execução “offline", resultado de um profile feito previamente da aplicação.

5.2.1 Cenário de Teste

A aplicação utilizada para avaliarmos nossas estratégias de escalonamento é o estudo simulado de cancer cerebral(*in silico studies of brain cancer*) [Cooper et al., 2010]. Esse estudo objetiva encontrar a melhor classificação de tumores cerebrais usando a análise de dados complementares tais como imagens em alta resolução de fatias de um tecido cerebral completo (*Whole Tissue Slide Images*,WSIs), dados de expressão gênica, informações clínicas, e imagens de radiologia. As WSIs podem apresentar diversas camadas de 120K×120K pixels.

De forma geral, o fluxo de trabalho (*workflow* dessa aplicação apresenta os seguintes estágios: (1) preprocessamento por meio de normalização de cores, (2) segmentação de células e núcleos (objetos), (3) extração de conjuntos de formas e características de texturas por objetos, e (4) classificação baseada nas características dos objetos. Os estágios que demandam maior esforço computacional são a (2) segmentação e (3) extração de características. Portanto, em nosso trabalho utilizaremos esses estágios como base para a execução eficiente em sistemas híbridos, e para a execução e avaliação dos escalonadores propostos.

O estágio de segmentação detecta células e núcleos, delineando-os. Consistem em um conjunto de operações executadas em uma determinada ordem, as quais formam um grafo representando um fluxo de dados (*dataflow*), como apresentado na Figura 5.2. As operações de segmentação incluem reconstrução morfológica para identificar objetos

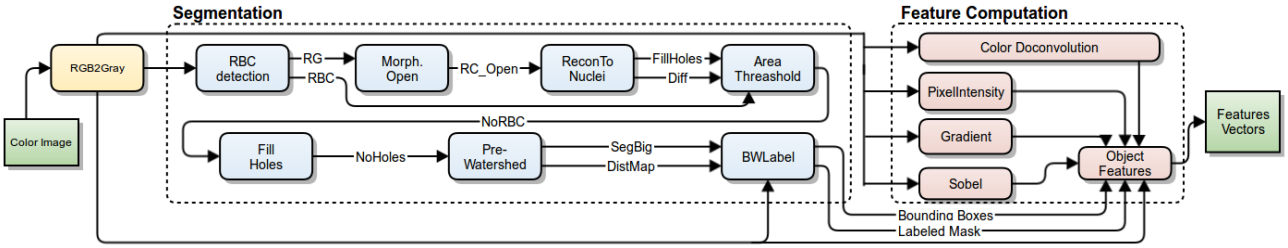


Figura 5.2: Fluxo de dados e operações da aplicação de analise de patologias em imagens. Os estágios de segmentação e extração de características (tarefas grão-grosso) da aplicação são decompostos em outros fluxos de dados eoperações de operações grão-fino, as quais são assinaladas para a execução em máquinas híbridos.

de interesse, segmentação em cascata para separar objetos sobrepostos, e filtragem para eliminar candidatos que são improváveis de serem núcleos.

O estágio de extração de características deriva atributos quantitativos, armazenando-os na forma de um vetor de características, para cada objeto encontrado na etapa de segmentação. Os tipos de características incluem estatísticas de pixel, informações de gradientes, detecção de bordas, e morfometria. A maioria das características podem ser computadas independentemente.

O trabalho [Teodoro et al., 2014] apresenta um estudo detalhado dessas operações. Nesse estudo cada operação teve seu desempenho avaliado em CPUs *multicore*, GPUs e no acelerador Intel Xeon Phi (MIC). Essas operações foram implementadas em todos os dispositivos utilizando as mesmas estratégias de paralelização e níveis de otimização. Os códigos em CPU foram implementados em C++, os quais foram adaptados em OpenMP para execuções em MIC. As versões em GPU foram implementadas em CUDA. Os speedups alcançados pelas operações nos aceleradores, GPU e MIC, usando um único núcleo CPU como base, são apresentados na Figura 5.3.

As operações podem ser classificados em três grupos diferentes com computações e padrões de acessos à memória similares:

1. Acesso de dados regular: RGB2Gray, Morphological Open, Color Deconvolution, Pixel Stats, Gradient Stats, e Sobel Edge;

2. Acesso de dados irregular: Morphological Reconstruction, FillHoles, e Distance Transform;

3. Operações que dependem de instruções atômicas: Area Threshold and Connected Component Labeling (CCL).

Essas características juntamente com a intensidade de processamento das operações foram correlacionadas com as especificações dos processadores e *benchmarks*

para explicar os referentes desempenhos. Todos os detalhes e explicações em relação ao desempenho individual das operações em cada dispositivo, são apresentados detalhadamente no trabalho [Teodoro et al., 2014].

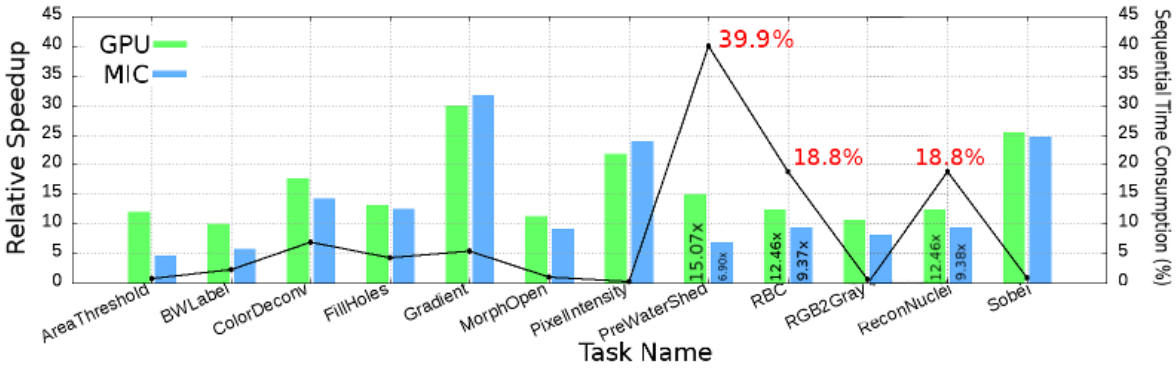


Figura 5.3: Speedups das operações grão-fino em cada um dos coprocessadores utilizados nesse trabalho, tendo como base um único núcleo CPU. A porcentagem do tempo de execução de uma operação em relação ao tempo total de execução da aplicação (peso de uma determinada operação) também é apresetnado.

Algumas características dessa aplicação podem ser observadas por meio da Figura 5.3: (1) Existe uma alta variabilidade dos speedups alcançados pelo mesmo processador dada diferentes operações; (2) A performance relativa entre processadores (CPU, GPU e MIC) varia de acordo com a operação executada. Esses fatos sugerem que diferentes processadores são mais eficientes para a execução de determinadas operações (ou acesso aos dados ou padrões de computação). Além disso, um escalonamento eficiente dessas operações em um sistema híbrido, considerando CPUs e acelerados além dessa variabilidade de desempenho, é fundamental para se extrair o máximo do poder de processamento de uma máquina heterogênea.

5.2.1.1 Desenvolvimento da Aplicação Dataflow no Extreme DataCutter

A fim de criar uma versão de alto desempenho da aplicação, implementamos várias versões das operações de grão-fino presentes nas etapas de segmentação e extração de características (Figura 5.2): C++ para CPUs, CUDA para GPUs e uma implementação C++ em OpenMP para a Intel Xeon Phi, como apresentado no trabalho trabalho [Teodoro et al., 2014]. A aplicação foi então desenvolvida usando o Extreme DataCutter para execução paralela em uma máquina de memória distribuída. Foram usadas as etapas de segmentação e extração de características em um primeiro nível (estágios grão-grosso), sendo que cada uma dessas duas etapas são compostas por um *dataflow* de operações grão-fino. Essas operações grão-fino são então escalonadas, por

meio de nossos escalonadores propostos, para diferentes unidades de processamento presentes em um computador híbrido.

A aplicação particiona as imagens de entrada em várias fatias que são analisadas de forma independente pelas etapas grão-grosso (segmentação e extração de características), permitindo criar instâncias separadas desses estágios para cada fatia da imagem de entrada.

Fizemos previamente um perfil da aplicação para armazenarmos os tempos de execução de cada operação nos dispositivos de processamento disponíveis, criando-se assim um histórico de execuções “offline". Os escalonadores propostos utilizam esse histórico para criar estimativas de speedup durante o processo de tomada de decisão.

5.2.2 Resultados e Discussões

A avaliação experimental foi realizada usando um conjunto de máquinas (*cluster*) em memória distribuída nomeado *Stampede*. Cada máquina de processamento nesse *cluster*, está equipado com dual socket Intel Xeon E5-2680 CPU, 32GB de memória principal, e um acelerador Intel Xeon Phi SE10P. Além de incluir 128 nodos com uma Intel Phi e uma placa NVIDIA K20 (GPU), a qual é usada principalmente para visualização. As máquinas desse *cluster* estão conectadas por uma rede Mellanox FDR Infiniband. Os códigos usados em nossa avaliação são compilados usando Intel Compiler 13.1 e com uma flag de otimização: -O3". Codigos baseados em Intel Phi são executado em modo “descarga"(conforme explicado na subseção 2.1.2). O conjunto de imagens usados como entrada foram todos coletados por meio do estudo de tumores cerebrais [Cooper et al., 2010], os quais são particionados em fatias de $4Kx4K$ pixel para a execução em paralelo.

5.2.2.1 Avaliação da Performance dos escalonadores

Nosso primeiro conjunto de experimentos avalia o desempenho dos escalonadores ao distribuir as tarefas entre os recursos disponíveis: CPUs *multicore*, GPU e MIC. A aplicação foi executada usando 1 GPU, 1 MIC, e 14 núcleos CPU: os outros 2 núcleos CPU foram usadas para a gerência dos aceleradores GPU e MIC. A carga da aplicação consistem em 800 fatias da imagem de entrada, que geram 10.800 operaçãos grão-fino para serem executadas. Adicionalmente, variamos a quantidade de estágios grão-grosso ativos concorrentemente em um Extreme DataCutter Worker (Worker Request Window Size), objetivando avaliar o impacto do espaço de decisão no desempenho dos escalonadores.

Variamos a *window size* iniciando no tamanho 16 (referente ao número de processadores usados para execução: CPU cores + aceleradores) até um determinado

tamanho em que a variação não impactasse mais no desempenho dos escalonadores. Speedups e tempos de execuções usados como entrada para os escalonadores foram coletados previamente, por meio da execução de um subconjunto da carga de entrada.

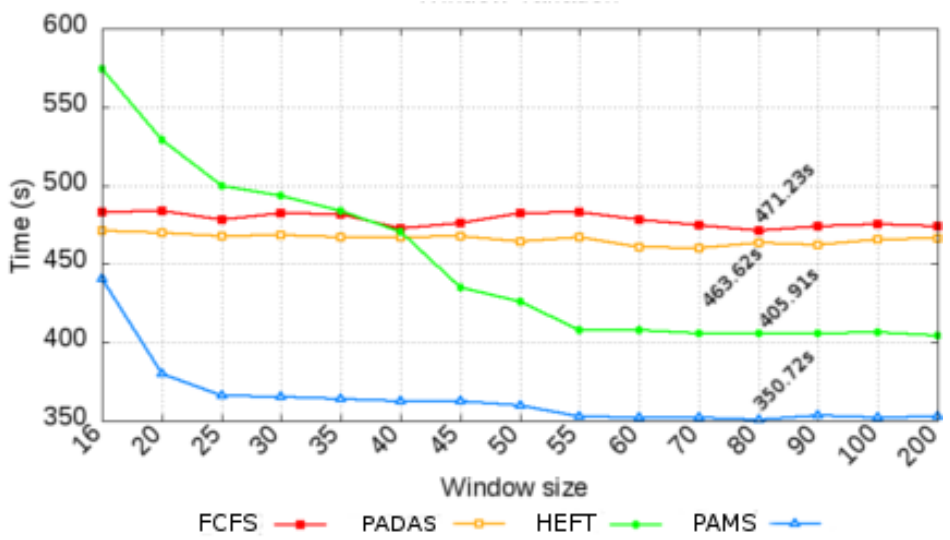


Figura 5.4: Desempenho dos escalonadores por meio da variação do número de instâncias de estágios grão-grosso ativos em um Worker (Worker Request Window Size).

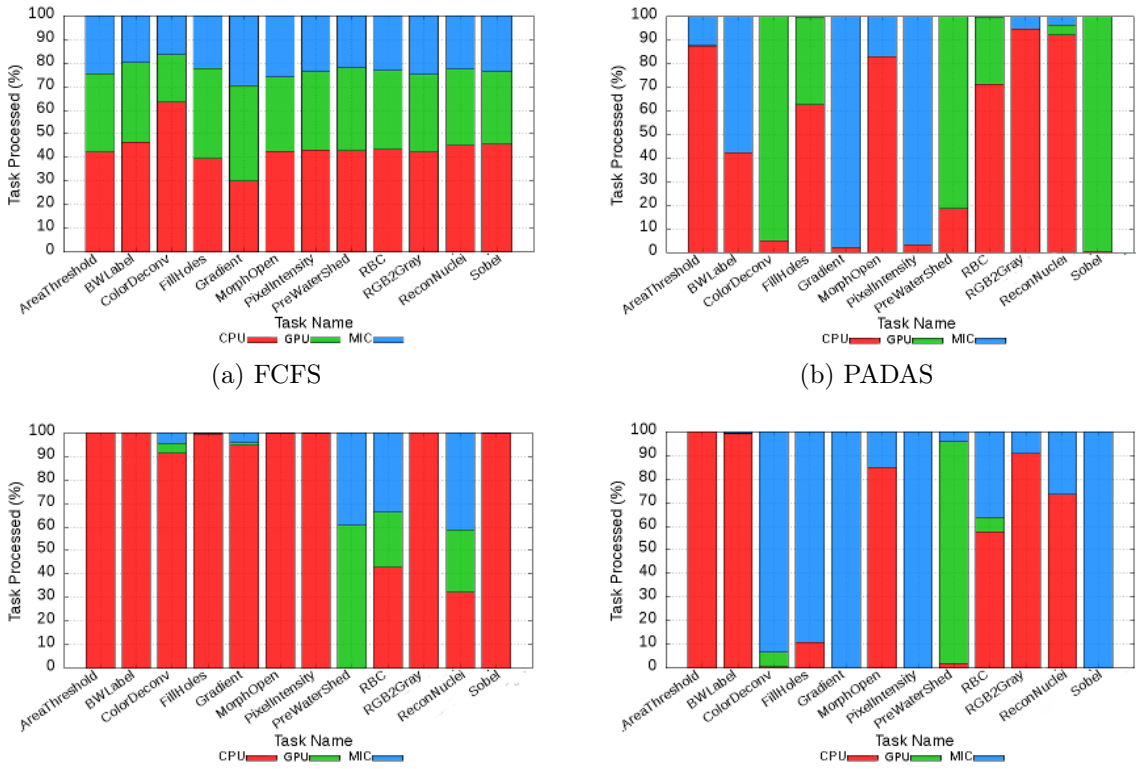
O desemepnho da aplicação para os escalonadores FCFS, HEFT, PADAS, e PAMS é apresentado na Figura 5.4. Como mostrado, as estratégias FCFS e PADAS alcançaram tempos de execução similares, sendo que a nossa proposta, PADAS, foi ligeiramente mais eficiente que o escalonador tradicional FCFS. Adicionalmente, vemos que a variação do *window size* teve pouco impacto nesses dois escalonadores: FCFS e PADAS. Além disso, a estratégia HEFT alcançou um desempenho ruim com o valor de *window size* pequeno, porém obteve melhores resultados que os escalonadores FCFS e PADAS para valores de *window size* superiores a 45. Um largo espaço de decisão (*window size*) é muito importante para o escalonador HEFT, visto que permite à essa estratégia melhor distribuição de trabalho entre os processadores, reduzindo a má adequação de tarefas entre os recursos.

Por fim, o escalonador proposto PAMS alcançou os menores tempos de execução entre as demais políticas de escalonamento em todos os tamanhos de *window size* observados. Além disso, na configuração onde o valor da *window size* é 80, PAMS é em torno de 1.16× mais rápido que o escalonador tradicional HEFT. Também é possível notarmos que a estratégia PAMS é a menos sensível à variação da *window size* ao compararmos com os outros escalonadores avaliados.

Na próxima subseção, apresentaremos um estudo detalhado do comportamento das políticas de escalonamento avaliadas.

5.2.2.2 Entendendo o comportamento dos escalonadores

Para compreendermos melhor o desempenho de cada um dos escalonadores, estudamos detalhadamente o processo de associação de tarefas realizado por cada escalonador correlacionando com a performance da tarefa no dispositivo. O desempenho em cada coprocessadores e o peso dessas operações no tempo total da aplicação são apresentados na Figura 5.3. É importante lembrarmos que: (1) existe uma alta variabilidade de performance entre as operações, visto que elas apresentam diferentes intensidades de computação e padrões de acesso aos dados, tornando algumas operações mais adequadas à serem executadas em diferentes processadores, e (2) operações possuem diferentes pesos de execução, de forma que uma associação inadequada de operações em unidades de processamento pode prejudicar o desempenho da aplicação.



(a) FCFS (b) PADAS

(c) HEFT (d) PAMS

Figura 5.5: Perfil da distribuição de tarefas entre os dispositivos realizado por cada política de escalonamento. Foi fixada o valor 80 para a *window size*.

O perfil de distribuição de tarefas/operações em cada unidade de processamento disponível (CPU, GPU e MIC) de acordo com a política de escalonamento usada: FCFS, PADAS, HEFT, e PAMS, para o valor 80 de *window size* é apresentado na Figura 5.5. Primeiramente, mostramos que a associação realizada de tarefas pelo FCFS

é praticamente a mesma para todas as operações, sendo que, essa estratégia tradicional não é capaz de tirar proveito de informações sobre a variação de performance das operações entre os recursos disponíveis. A política PADAS, apesar de levar em conta a diferença de performance das tarefas entre as unidades de processamento, alcança apenas uma ligeira melhora em relação ao escalonador FCFS. A análise da distribuição das operações com elevados tempos de execução (PreWatershed, RBC e ReconNuclei) mostra que o algoritmo PADAS associa corretamente a operação PreWatershed para a GPU, porém uma quantidade significativa dessa operação é associada a CPU, que é extremamente ineficiente para essa tarefa. Além disso, PADAS associa a maioria das tarefas RBC também para a CPU, sendo que essa é melhor para ser executada nos aceleradores: GPU ou MIC. A distribuição da operação de Sobel também deixa claro que o escalonador PADAS não foi capaz de realizar uma boa associação das tarefas, visto que associa a maior parte dessa operação à GPU, sendo que a execução da ReconNuclei é mais eficiente no coprocessador MIC.

O perfil de distribuiçãi de tarefas realizado pelo escalonador HEFT é apresentado na Figura 5.5c. Inicialmente notamos que o HEFT associou tarefas com os menores tempos de execução aos núcleos CPU, independentemente do speedup nos dispositivos disponíveis. Além disso, HEFT associou, exclusivamente, tarefas PreWatershed para os acelerados. Entretanto, o coprocessador GPU é duas vezes mais rápida que o MIC para a execução dessa operação e, como consequência, PreWatershed deveria ser executada em maior porcentagem na GPU. Essa operação, representa em torno de 40% do tempo de execução da aplicação, e sua correta associação é de maior importância para se obter uma boa performance. Por fim, o perfil do escalonador PAMS mostrou que nossa proposta: (1) Associou corretamente a maior parte das operações PreWatershed à GPU; (2); Associou corretamente operações adequadas ao MIC, como por exemplo ColorDevonv, FillHoles, Gradient, PixelIntensity, e Sobel; e (3) a CPU foi usada para operações que não são eficientemente executadas pelos aceleradores, incluindo AreaThreshold and BWLabel.

5.2.2.3 Variando a configuração dos processadores

Nessa subseção avaliamos os escalonadores com diferentes configurações de processadores: CPU-GPU (15 CPU cores e 1 GPU), CPU-MIC (15 CPU cores e 1 MIC) e CPU-GPU-MIC (14 CPU cores , 1 GPU, e 1 MIC). A carga de trabalho é a mesma usada nas seções anteriores.

Os tempos de execução dos escalonadores são apresentados na Figura 5.6. Como mostrado, o escalonador FCFS alcançou o pior desempenho para todos as configu-

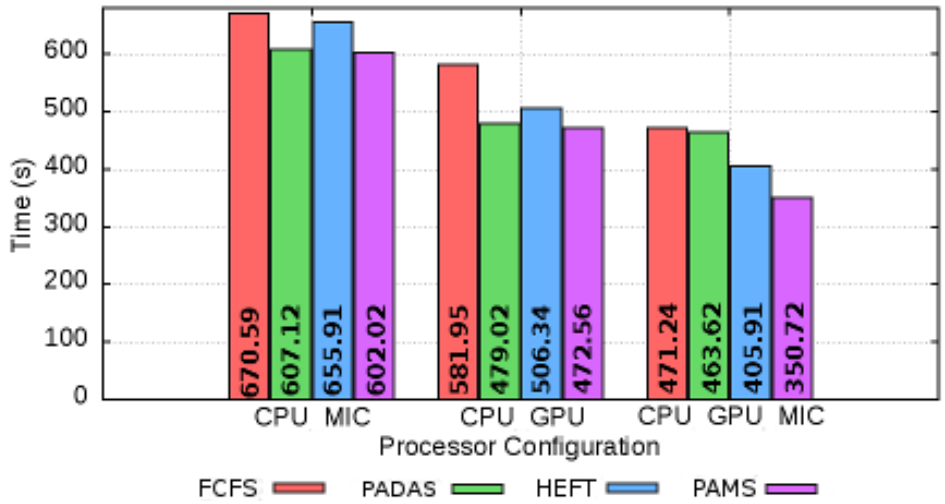


Figura 5.6: Desempenho dos escalonadores em execuções cooperativas usando diferentes tipos de processadores.

rações. Nas execuções cooperativas usando dois tipos de processadores, CPU-MIC ou CPU-GPU, as estratégias propostas PADAS e PAMS alcançaram tempos de execuções similares. Nessas configurações, ambos escalonadores basicamente executam o mesmo escalonamento usando diferentes estruturas: uma única lista ordenada pelo speedup e duas listas ordenadas pelos speedups. Adicionalmente, PADAS e PAMS alcançaram melhor performance do que o escalonador HEFT nas mesmas configurações. Além disso, na configuração com CPU-GPU-MIC, PAMS superou todos os outros escalonadores conforme discutidos na subseção anterior. É importante destacar que o uso da GPU sempre impacta no aumento de desempenho, por exemplo, o melhor tempo de execução na configuração CPU-GPU é 1.27× mais rápido que o melhor tempo de execução na configuração CPU-MIC.

5.2.2.4 Sensibilidade na presença de erros

Nessa subseção avaliamos os escalonadores com relação ao impacto da acurácia das métricas de entrada (tempo de execução e speedup) no desempenho dos escalonadores. Para essa avaliação, conduzimos experimentos nos quais erros foram inseridos de forma controlada nos tempos de execução esperados das operações. Para os escalonadores que se baseiam no speedup, esse valor foi derivado dos tempos de execução com os erros. Variamos os erros inseridos em 0%, 10%, 25%, 50%, 75%, e 100% nos tempos de execução esperados, com chances iguais da variação (erro) incrementar ou decrescer o valor do tempo de execução.

A performance dos escalonadores são apresentadas na Figura 5.7. Como

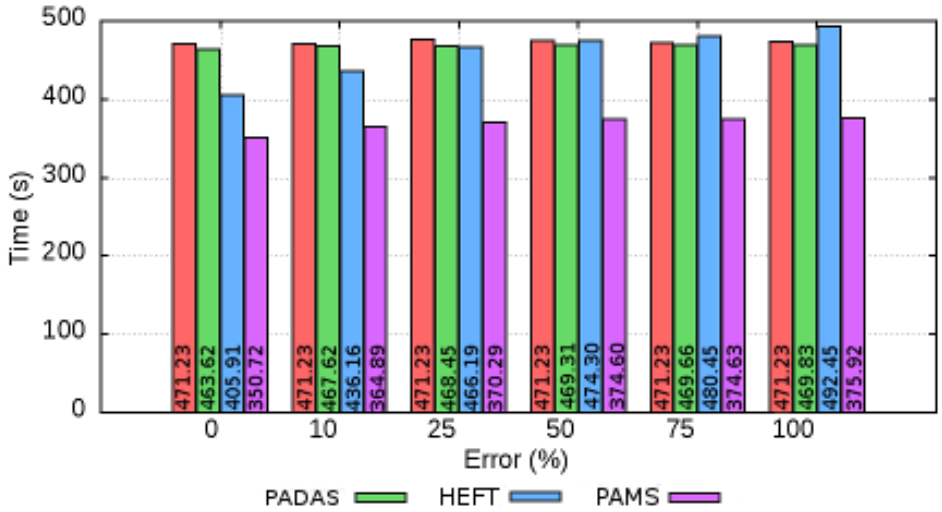


Figura 5.7: Desempenho das políticas de escalonamento na presença de diferentes níveis de erros inseridos nas estimativas dos tempos de execuções de cada operação.

mostrado, os escalonadores com base na métrica de speedup, PADAS e PAMS, são ligeiramente impactados de acordo com o crescimento dos erros. Em contrapartida, o escalonador baseado em tempo de execução HEFT sofre drasticamente com o aumento das taxas de erros. HEFT é ainda mais lento que FCFS para taxas de erros superiores a 50%. Esses resultados são importantes para mostrar que estratégias de escalonamento baseadas em performance utilizando o speedup são bastante insensíveis à presença de erros, e continuam funcionando bem com altas taxas de erros nas estimativas do tempo de execução. O escalonador PAMS com 0% de taxa de erro é apenas 1.07× mais rápido que o escalonador PAMS com 100% de taxa de erro.

5.2.2.5 Escalabilidade da Aplicação

Nessa subseção avaliamos a aplicação usando um *cluster* de máquinas em memória distribuída. A aplicação exemplo, descrita na subseção 5.2.1, é estruturada por meio de *dataflow* hierárquico, sendo o primeiro nível composto pelos estágios de segmentação e extração de características, os quais são implementados em um outro *dataflow* de operações grão-fino. Avaliamos a aplicação utilizando apenas todos os 16 CPU cores em cada nodo de processamento, e também avaliamos a aplicação utilizando CPUs, GPUs, e MICs com as diferentes estratégias de escalonamento.

A avaliação da intensa escalabilidade é apresentada na Figura 5.8. Os experimentos usaram 6,379 4K×4K tiles de imagens como entrada. Todas as versões da aplicação escalaram muito bem, e a melhor execução cooperativa utilizando o escalonador PAMS é em torno de 2.2× mais rápido que a execução da aplicação

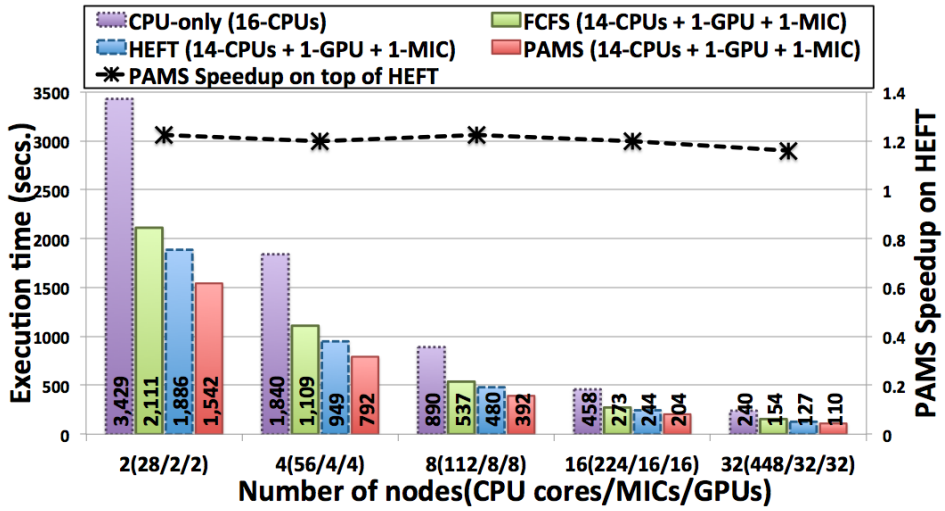


Figura 5.8: Avaliação de escalabilidade em diferentes quantidades de nodos de processamento.

apenas em CPU para todos as quantidades de nodos. Adicionalmente, como mostrado, PAMS é em torno de 1.2× mais rápido que HEFT, e essa observação se mantém de acordo com o crescimento do número de máquinas no *cluster*.

5.3 Sumário

Nesse capítulo apresentamos um estudo sobre escalonamento dinâmico de tarefas em arquiteturas híbridas composta por CPU *multicore* e diferentes coprocessadores. Foi adicionado, além da GPU, o coprocessador MIC, visto a sua popularização em sistemas híbridos, Dessa forma, visando a utilização eficiente dos recursos disponíveis, apresentamos duas propostas de escalonamento que diferentemente das abordagens baseadas na estimativa dos tempos de execução das tarefas, utilizam a performance (speedup) para a predição do desempenho de tarefas nos recursos disponíveis.

Conjuntamente com a predição do desempenho por meio do speedup das tarefas nos recursos disponíveis, foi inserido, no escalonador PAMS, uma característica adaptativa: a replicação de tarefas entre as filas das unidades de processamento. Tal estratégia, similarmente ao roubo de tarefas, visa eliminar o problema da ociosidade dos processadores disponíveis, aumentando a utilização do sistema.

Para a avaliação dos escalonadores propostos e discussão das características inseridas, a aplicação utilizada é a análise de patologias em imagens [Cooper et al., 2010]. A aplicação foi implementada no ambiente Extreme DataCutter, que possui suporte a implementação de aplicações com estrutura de *dataflow* hierárquico e também provê suporte ao coprocessador MIC.

Avaliamos experimentalmente as abordagens propostas, utilizando 800 fatias de imagens, o que gera cerca de 10.800 tarefas grão-fino para a execução. Comparamos nossas propostas com dois escalonadores dinâmicos tradicionais: FCFS e HEFT. Nossos resultados mostram que o escalonador PAMS alcançou os menores tempos de execução comparado às demais políticas, sendo 1.16× mais rápido que a política tradicional HEFT, a qual apresentou o segundo menor tempo de execução. Adicionalmente, também mostramos, que nossas estratégias, PADAS e PAMS, são menos sensíveis a presença de erros na estimativa de desempenho utilizada do que a estratégia HEFT, a qual sofre drasticamente quando taxas de erros na estimativa dos tempos de execução crescem. Por fim, nossos experimentos mostraram que a aplicação escala bem com o Extreme DataCutter e os melhores tempos de execução cooperativa, utilizando a estratégia proposta PAMS, são em torno de 2.2× mais rápido que a versão apenas em CPU para qualquer quantidade de nodos.

Capítulo 6

Trabalhos Relacionados

A partir do surgimento das arquiteturas híbridas de computadores, compostas por diferentes unidades de processamento, vários formas de se executar aplicações em plataformas híbridas vem sendo estudadas. Nesse contexto vários ambientes de execução foram sendo desenvolvidos e aprimorados para que a capacidade de processamento dessas arquiteturas pudessem ser exploradas ao máximo [He et al., 2008; Linderman et al., 2008; Luk et al., 2009; Rossbach et al., 2011; Augonnet et al., 2012; Bosilca et al., 2011; Teodoro et al., 2010, 2009; Bueno et al., 2012; Ravi et al., 2010; Lima et al., 2013]. Esses ambientes proporcionam uma série de facilidades ao desenvolvedor, tais como funções e diretivas, *debugging*, geração otimizada de código baixo nível, entre outros. De forma geral, o objetivo desses ambientes é prover a portabilidade e integração das várias unidades de processamento, de forma clara e em um nível mais abstrato para o desenvolvedor.

Podemos citar o *StarPU* [Augonnet et al., 2009], 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. Nesse ambiente é possível representar tarefas e suas dependências. Além de quê, o StarPU permite alterar os métodos de escalonamento de tarefas existentes, bem como incorporar outros no ambiente. Na mesma linha em [Diamos & Yalamanchili, 2008] os autores apresentam o *Harmony*, um ambiente de execução bastante semelhante ao *StarPU*, porém sem permitir que os desenvolvedores incorporarem qualquer outra estratégia de escalonamento no ambiente. Ainda no contexto de ambientes de execução para plataformas híbridas, DaGuE [Bosilca et al., 2011] é um fremeworks que suportam a execução de aplicações regulares de algebra linear em máquinas CPU-GPU. Esse sistema representa uma aplicação como um DAG de operações e garantem que as dependências são respeitadas. Oferece diversos estratégias de escalonamento, incluindo aquelas que priorizam a computação de caminhos críticos através do grafo de dependências,

para garantir o máximo de paralelismo. Suporta execução multi-nodo e assume que o grafo de dependências da aplicação é estático e conhecido antes da execução. Similarmente, OmpSs [Duran et al., 2011] estaticamente associa uma tarefa à uma unidade de processamento, oferece suporte a execução eficiente e assíncrona de aplicação paralelizadas em tempo de compilação através de anotações nos códigos. Por fim, em um contexto mais geral, o Extreme DataCutter [Teodoro et al., 2013] permite ao desenvolvedor implementar aplicações dataflow hierárquico e executá-las em um ambiente heterogêneo e distribuído com a presença de coprocessadores GPUs e MICs, além de possibilitar a inserção de escalonadores para tarefas de grão-fino presentes na estrutura da aplicação.

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 unidades de processamento disponíveis, os chamados escalonadores de tarefas. Diferentemente dos métodos fornecidos e implementados nos ambientes OmpSs e DaGuE, em nosso trabalho lidamos com aplicações, em que, não é conhecido a priori o conjunto de trabalho e dessa forma a associação das tarefas aos recursos de processamento é realizado de forma dinâmica. Portanto lidamos com o problema de escalonar tarefas dinamicamente entre unidades de processamento e sendo assim, 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 unidades de processamento. [Augonnet et al., 2009].

Existem na literatura diversas propostas de escalonadores dinâmicos ([Blumofe & Leiserson, 1999; Teodoro et al., 2009; Smith et al., 1999; Jooya et al., 2011]), muitas delas utilizadas pelos ambientes de execução acima mencionados. Em [Blumofe & Leiserson, 1999] 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". Por ajustar o balanceamento de cargas durante a execução da aplicação, realocando tarefas já associadas entre os recursos, esse tipo de abordagem é conhecida como escalonamento dinâmico adaptativo [Casavant & Kuhl, 1988], sendo que a replicações de tarefas também pode ser utilizada reduzir a ociosidades dos processadores.

Em contrapartida, quando não existe esse ajuste de tarefas já associadas à uma unidades de processamento, como feito no escalonamento adaptativo, de forma que a primeira associação realizada pelo escalonador é mantida, temos o escalonamento dinâ-

mico não-adaptativo [Casavant & Kuhl, 1988]. Exemplos tradicionais dessa abordagem, como First Come First Served e Heterogeneous Earliest Finish Time (HEFT) são apresentados nos trabalhos [Augonnet et al., 2009]. Estratégias de escalonamento dinâmica não-adaptativa mais elaboradas são apresentadas nos trabalhos [Smith et al., 1999; Teodoro et al., 2009; Jooya et al., 2011; Iverson et al., 1996, 1999; Breß et al., 2012]. Em [Teodoro et al., 2009] 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. Nesse caso, a preocupação na boa adequação das tarefas entre as unidades de processamento é feita através da ordenação das tarefas na fila global, porém outras técnicas aplicadas em tempo de execução incluem abordagens de predições de tempo de execução [Smith et al., 1999; Iverson et al., 1996, 1999], como feito pela estratégia tradicional HEFT.

Nesse contexto, em [Smith et al., 1999], 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 unidades de processamento menos ocupadas, provendo dessa forma, um balanceamento de carga equilibrado. Adicionalmente, em [Jooya et al., 2011] é apresentada uma proposta semelhante, onde tais modelos de execução são baseados nos históricos de execução passadas. Esse histórico de execuções passadas, como mostrado em [Jooya et al., 2011], são criados através de um pré-processamento de tarefas semelhantes, os quais são armazenados os tempos de execuções dessas tarefas em todas as unidades de processamento disponíveis. Essa estratégia, conhecida como predição estática, é também empregada nas estratégias de escalonamento apresentadas nos trabalhos [Iverson et al., 1996, 1999]. Diferentemente dessa predição estática, esse histórico de execução pode ser construído ou atualizado durante a execução das tarefas da aplicação, caracterizando assim um auto calibragem do histórico de execuções. No trabalho [Breß et al., 2012] temos esse tipo de estratégia, e também em [Augonnet et al., 2009] os autores mostram que as estratégias propostas, baseadas na estimativa do tempo de execução por histórico de execuções, podem se beneficiar de um histórico construído através de pré-processamento, ou construído e atualizado ao longo da execução da aplicação. Ainda em [Augonnet et al., 2009] também é apresentada uma forma de predição de tempo de execução baseada em estratégia de regressão linear. Não só a estimativa do tempo de execução das tarefas é usado como ferramenta para a boa adequação de conjuntos de trabalho entre as unidades de processamento, mas como também uma

estimativa de desempenho (speedup) é usualmente empregada. No trabalho [Teodoro et al., 2009] o speedup é utilizado como métrica para a boa adequação.

Por fim, devemos destacar que no escalonamento dinâmico, outras características das tarefas da aplicação, além de tempo de execução e desempenho, podem ser tomadas como diretivas para a boa distribuição das mesmas visando um determinado objetivo. No trabalho [Augonnet et al., 2009], é apresentada um escalonador "data-aware"o qual considera a quantidade de dados manipulados pela operação, de forma a evitar que o *overhead* de transferência de dados não compense a minimização do tempo de execução da tarefa no recurso destino. Outro fator que vem sendo foco de muitas investigações é a redução energética [Li et al., 2013], o qual caracteriza um escalonamento ecologicamente consciente.

Os escalonadores dinâmicos citados anteriormente possuem exclusivamente características: (1) adaptativas, como roubo de tarefas ou replicação; (2) não adaptativas, como por exemplo a adequação de tarefas pela estimativa do tempo de execução ou Speedup. Porém, ambas características são importantes em determinados contextos, tornando o desempenho desses escalonadores, apenas adaptativos ou não-adaptativos, instável quando varia-se características das aplicação. Dessa forma, propomos neste trabalho, escalonadores dinâmicos de tarefas combinando características adaptativas e não adaptativas. Em outras palavras, propomos unir a boa adequação de tarefas através de predições de tempo e desempenho, com o balanceamento de carga adaptativo realizado por exemplo pelo roubo ou replicação de tarefas entre as unidades de processamento. Nosso objetivo é apresentar novas estratégias de escalonamento, que combinando características adaptativos e não-adaptativas, tenham bons desempenhos em uma gama de aplicações genéricas.

Adicionalmente, os escalonadores apresentados pela literatura são implementadas e avaliadas em ambientes híbridos CPU-GPU. Com a popularização de outros coprocessadores, além da GPU, é necessário compreendermos o comportamento de escalonadores dinâmicos, adaptativos e não-adaptativos, na presença de uma variedade de recursos de processamento. Dessa forma, em nosso trabalho, apresentamos um estudo, proposta e avaliação de estratégias de escalonamento para arquiteturas híbridas com CPU multicores, e coprocessadores GPU e MIC. Para esse cenário, não foi encontrado abordagem similar em trabalhos anteriores da literatura.

Capítulo 7

Conclusões e Trabalhos Futuros

Nesse trabalho apresentamos um estudo sobre escalonamento dinâmico de tarefas em sistemas híbridos, propondo e avaliando diferentes políticas em cenários diversificados e sistemas equipados com coprocessadores GPU e MIC.

Em um primeiro momento apresentamos uma visão geral sobre o problema de escalonamento de tarefas. Durante essa apresentação, foram expostos os diferentes grupos de escalonadores, destacando as suas principais características e aplicações. Mais especificamente, estudamos o escalonamento dinâmico de tarefas. Nesse contexto, foram discutidos importantes fatores a cerca desse problema, apresentando duas subdivisões desse grupo: escalonadores dinâmicos adaptativos e escalonadores dinâmicos não-adaptativos. Dando sequencia a esse estudo, apresentamos e implementados escalonadores dinâmicos tradicionais presentes na literatura, sendo elas três politias de escalonamento dinâmico não-adaptativos: FCFS, HEFT e HEFT *data-aware*, e uma política de escalonamento dinâmico adaptativa: Work Stealing. Uma avaliação experimental desses escalonadores, em cenários sintéticos e reais, mostrou que existe uma variação de desempenho entre eles e que não existe um escalonador ideal para qualquer tipo de cenário de aplicação. Adicionalmente, foi concluído que determinadas características, adaptativas ou não-adaptativas, possuem melhor desempenho em determinados contextos, evidenciando que uma possível combinação de estratégias alcançaria melhores resultados que as estratégias originais. Resumidamente, esse estudo inicial acerca de escalonadores dinâmicos nos proporcional retirar as seguintes conclusões: (1) Para aplicações com tarefas apresentando expressiva diferença de desempenho entre as unidades de processamento, escalonadores que se beneficiam de estratégias de predião de tempo de execução, visando a boa adequação das tarefas nos recursos; (2) existe sobrecarga de trabalho para unidades de processamento suficientemente rápidas quando utilizamos escalonadores baseados em predição de

tempo; (3) quando as tarefas de uma aplicação apresentam uma pequena diferença de desempenho entre as unidades de processamento, a estratégia de roubo de tarefas é mais interessante visto o balanceamento de cargas realizado; (4) considerar a transferência de dados na predição do tempo de execução impacta na melhora do desempenho da aplicação em cenários o qual a quantidade de dados manipulados é expressiva.

Por meio das conclusões obtidas durante a análise dos escalonadores dinâmicos tradicionais, promomos duas novas estratégias. Tais estratégias basicamente combinam características de escalonadores baseados em "roubo de tarefas"com estratégia baseadas em modelos de predição de tempo de execução, considerando (A-HEFT) ou não (A-HEFT *data-aware*) o tempo de transferência de dados entre memória principal e memória das unidades de processamento. Avaliamos as políticas de escalonamento propostas em diferentes cenários, sintéticos e reais. Nossos resultados mostram que, enquanto em alguns cenários as abordagens baseadas somente em modelos de predição se mostram mais adequadas, 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 as 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.

Em sequência, motivados pelo fato de que nas arquiteturas híbridas atuais, não só GPUs tem se popularizado no contexto de coprocessamento massivamente paralelo, mas como também outros coprocessadores vem se destacando como interessantes alternativas, como por exemplo o coprocessador Intel Xeon Phi (MIC), apresentamos uma nova abordagem para o estudo do escalonamento dinâmico de tarefas em arquiteturas híbridas. Inseriomos o coprocessador MIC como recurso de processamento, conjuntamente com CPU multicore e GPUs em uma máquina híbrida. Para explorar de forma eficiente esse novo contexto foram proposto dois novos escalonadores de tarefas. Essas novas políticas, diferentemente das abordagens baseadas na estimativa dos tempos de execução das tarefas, utilizam a performance (speedup) para a predição do desempenho de tarefas nos recursos disponíveis. Conjuntamente com a predição do desempenho através da performance das tarefas nos recursos disponíveis, inserimos no escalonador PAMS, uma característica adaptativa: a replicação de tarefas entre as filas das unidades de processamento. Avaliamos experimentalmente as abordagens propostas, através de uma aplicação real de análise de imagens, com estrutura *dataflow* hierárquico. Comparamos nossas propostas com dois escalonadores dinâmicos tradi-

cionais: FCFS e HEFT. Os resultados mostram que o escalonador PAMS alcançou os menores tempos de execução comparado às demais políticas, sendo 1.16× mais rápido que a política tradicional HEFT, a qual apresentou o segundo menor tempo de execução. Adicionalmente, também mostramos, que nossas estratégias, PADAS e PAMS, são menos sensíveis a presença de erros na estimativa de desempenho utilizada do que a estratégia HEFT, a qual sofre drasticamente quando taxas de erros na estimativa dos tempos de execução crescem. Por fim, nossos experimentos mostraram que a aplicação escala bem com o Extreme DataCutter e os melhores tempos de execução cooperativa, utilizando a estratégia proposta PAMS, são em torno de 2.2× mais rápido que a versão apenas em CPU para qualquer quantidade de máquinas no *cluster*.

Como extensão do trabalho apresentado é possível realizarmos o mesmo conjunto de avaliações e análises dos escalonadores propostos em outros contextos de aplicações e ambientes de execuções para sistemas híbridos. Adicionalmente, faz-se necessário um estudo mais detalhado explicando teoricamente o por quê da predição de tempo ser mais sensível a presença de erros do que a predição de speedup. Por fim, outras formas de escalonamento podem ser exploradas no ambiente de execução Extreme DataCutter. Sua estrutura Maneger-Worker, permite o estudo de dois níveis de escalonamento: (1) escalonamento dinâmico distribuído de tarefas grão-grosso entre o manager e workers; (2) escalonamento dinâmico não distribuído de tarefas grão-fino entre as unidades de processamento presentes nas máquinas Workers. O segundo nível de escalonamento foi explorado em nosso trabalho, porém é possível estudarmos o primeiro nível, distribuído, de forma cooperativa com o segundo nível. Não só a cooperação entre os níveis de escalonamento pode ser explorada, mas como também a utilização de estratégias para lidar com um sistema de computação completamente distribuído e heterogêneo, no qual cada nodo de processamento apresenta diferentes unidades de processamento e diferentes capacidades dos recursos.

Referências Bibliográficas

Ahmad, I. (1995). Editorial: Resource management of parallel and distributed systems with static scheduling: Challenges, solutions and new problems. *Concurrency - Practice and Experience*, 7(5):339–347.

Amalarethinam, D. G. & Mary, G. J. (2011). Article: A new dag based dynamic task scheduling algorithm (dytas) for multiprocessor systems. *International Journal of Computer Applications*, 19(8):24–28. Published by Foundation of Computer Science.

Augonnet, C.; Aumage, O.; Furmento, N.; Namyst, R. & Thibault, S. (2012). StarPU-MPI: Task Programming over Clusters of Machines Enhanced with Accelerators. Em *The 19th European MPI Users' Group Meeting (EuroMPI 2012)*.

Augonnet, C.; Thibault, S.; Namyst, R. & Wacrenier, P.-A. (2009). Starpu: A unified platform for task scheduling on heterogeneous multicore architectures. Em *Euro-Par '09: Proceedings of the 15th International Euro-Par Conference on Parallel Processing*, pp. 863--874.

Beynon, M. D.; Kurc, T.; Catalyurek, U.; Chang, C.; Sussman, A. & Saltz, J. (2001). Distributed processing of very large datasets with DataCutter. *Parallel Comput.*, 27(11):1457--1478.

Blumofe, R. D. & Leiserson, C. E. (1999). Scheduling multithreaded computations by work stealing. *J. ACM*, 46(5):720--748.

Bosilca, G.; Bouteiller, A.; Herault, T.; Lemarinier, P.; Saengpatsa, N.; Tomov, S. & Dongarra, J. (2011). Performance Portability of a GPU Enabled Factorization with the DAGuE Framework. Em *IEEE Int. Conf. on Cluster Comp.*, pp. 395–402.

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

Breß, S.; Mohammad, S. & Schallehn, E. (2012). Self-tuning distribution of db-operations on hybrid cpu/gpu platforms. Em *Grundlagen von Datenbanken*, pp. 89--94.

Bueno, J.; Planas, J.; Duran, A.; Badia, R.; Martorell, X.; Ayguade, E. & Labarta, J. (2012). Productive Programming of GPU Clusters with OmpSs. Em *2012 IEEE 26th Int. Parallel Distributed Processing Symp. (IPDPS)*, pp. 557–568.

Casanova, H.; Legrand, A.; Zagorodnov, D. & Berman, F. (2000). Heuristics for scheduling parameter sweep applications in grid environments. Em *Heterogeneous Computing Workshop, 2000.(HCW 2000) Proceedings. 9th*, pp. 349--363. IEEE.

Casavant, T. & Kuhl, J. (1988). A taxonomy of scheduling in general-purpose distributed computing systems. *Software Engineering, IEEE Transactions on*, 14(2):141–154.

Cooper, L.; Kong, J.; Gutman, D.; Wang, F. & et al. (2010). An integrative approach for in silico glioma research. *IEEE Trans Biomed Eng.*, 57(10):2617--2621.

Cramer, T.; Schmidl, D.; Klemm, M. & an Mey., D. (2012). OpenMP Programming on Intel Xeon Phi Coprocessors: An Early Performance Comparison. Em *Many-core Applications Research Community Symposium*.

Da Silva, D. P.; Cirne, W. & Brasileiro, F. V. (2003). Trading cycles for information: Using replication to schedule bag-of-tasks applications on computational grids. Em *Euro-Par 2003 Parallel Processing*, pp. 169--180. Springer.

Diamos, G. F. & 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*, pp. 197--200.

Dong, F. & Akl, S. G. (2006). Scheduling algorithms for grid computing: State of the art and open problems. Relatório técnico, Technical report.

dos Santos Neto, E. L. (2004). *Escalonamento de Aplicações que Processam Grandes Quantidades de Dados em Grids Computacionais*. Tese de doutorado, Master's thesis, Coordenaçao de Pós-Graduaçao em Informática-Universidade Federal de Campina Grande.

Duran, A.; Ayguadé, E.; Badia, R. M.; Labarta, J.; Martinell, L.; Martorell, X. & Planas, J. (2011). Ompss: a proposal for programming heterogeneous multi-core architectures. *Parallel Processing Letters*, 21(02):173--193.

Eisenlohr, J.; Hudak, D. E.; Tomko, K. & Prince, T. C. (2012). Dense Linear Algebra Factorization in OpenMP and Cilk Plus on Intel's MIC Architecture: Development Experiences and Performance Analysis. Em *TACC-Intel Highly Parallel Computing Symp.*

Fatica, M. & Luebke, D. (2007). High performance computing with CUDA. Supercomputing 2007 tutorial. In Supercomputing 2007 tutorial notes.

Feitelson, D. G.; Rudolph, L.; Schwiegelshohn, U.; Sevcik, K. C. & Wong, P. (1997). Theory and practice in parallel job scheduling. Em *Proceedings of the Job Scheduling Strategies for Parallel Processing*, IPPS '97, pp. 1--34, London, UK, UK. Springer-Verlag.

Gautier, T.; Lima, J. V. F.; Maillard, N. & Raffin, B. (2013). Xkaapi: A runtime system for data-flow task programming on heterogeneous architectures. Em *IPDPS '13: Proceedings of the 2013 IEEE International Symposium on Parallel and Distributed Processing*, pp. 1299–1308. IEEE Computer Society.

Hamidouche, K.; Potluri, S.; Subramoni, H.; Kandalla, K. & Panda, D. K. (2013). MIC-RO: enabling efficient remote offload on heterogeneous many integrated core (MIC) clusters with InfiniBand. Em *27th Int. ACM International Conference on Supercomputing*, ICS '13, pp. 399--408.

He, B.; Fang, W.; Luo, Q.; Govindaraju, N. K. & Wang, T. (2008). Mars: A MapReduce Framework on Graphics Processors. Em *Parallel Architectures and Compilation Techniques*.

Heinecke, A.; Vaidyanathan, K.; Smelyanskiy, M. & et al. (2013). Design and Implementation of the Linpack Benchmark for Single and Multi-node Systems Based on Intel Xeon Phi Coprocessor. Em *The 27th IEEE International Symposium on Parallel Distributed Processing*.

Iverson, M. A.; Ozguner, F. & Follen, G. J. (1996). Run-time statistical estimation of task execution times for heterogeneous distributed computing. Em *High Performance Distributed Computing, 1996., Proceedings of 5th IEEE International Symposium on*, pp. 263--270. IEEE.

Iverson, M. A.; Ozguner, F. & Potter, L. C. (1999). Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment. Em *Heterogeneous Computing Workshop, 1999.(HCW'99) Proceedings. Eighth*, pp. 99--111. IEEE.

Jooya, A.; Baniasadi, A. & Analoui, M. (2011). History-aware, resource-based dynamic scheduling for heterogeneous multi-core processors. *Computers Digital Techniques, IET*, 5(4):254 –262.

Joó, B.; Kalamkar, D.; Vaidyanathan, K.; Smelyanskiy, M.; Pamnany, K.; Lee, V.; Dubey, P. & Watson, William, I. (2013). Lattice QCD on Intel Xeon Phi Coprocessors. Em *Supercomputing*, volume 7905 of *LNCS*, pp. 40–54.

Kale, L. V. & Krishnan, S. (1993). *CHARM++: a portable concurrent object oriented system based on C++*, volume 28. ACM.

Kim, J.; Seo, S.; Lee, J.; Nah, J.; Jo, G. & Lee, J. (2012). Opencl as a unified programming model for heterogeneous cpu/gpu clusters. *SIGPLAN Not.*, 47(8):299--300.

Kunzman, D. (2006). Charm++ on the cell processor.

Kwok, Y.-K. & Ahmad, I. (1999). Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Comput. Surv.*, 31(4):406--471.

Li, K.; Tang, X. & Li, K. (2013). Energy-efficient stochastic task scheduling on heterogeneous computing systems. *IEEE Transactions on Parallel and Distributed Systems*, 99(PrePrints):1.

Lima, J. V. F.; Broquedis, F.; Gautier, T. & Raffin, B. (2013). Preliminary experiments with xkaapi on intel xeon phi coprocessor. Em *25th International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD*, pp. 105–112.

Linderman, M. D.; Collins, J. D.; Wang, H. & Meng, T. H. (2008). Merge: a programming model for heterogeneous multi-core systems. *SIGPLAN Not.*, 43(3):287--296.

Luebke, D.; Harris, M.; Govindaraju, N.; Lefohn, A.; Houston, M.; Owens, J.; Segal, M.; Papakipos, M. & Buck, I. (2006). Gpgpu: general-purpose computation on graphics hardware. Em *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, p. 208. ACM.

Luk, C.-K.; Hong, S. & Kim, H. (2009). Qilin: Exploiting parallelism on heterogeneous multiprocessors with adaptive mapping. Em *42nd Int. Symp. on Microarchitecture*.

NVIDIA (2011). NVIDIA CUDA SDK.

Owens, J. D.; Luebke, D.; Govindaraju, N.; Harris, M.; Kruger, J.; Lefohn, A. E.; & Purcell., T. J. (2007). A survey of general-purpose computation on graphics hardware. computer graphics. 26:80--113.

Ravi, V.; Ma, W.; Chiu, D. & Agrawal, G. (2010). Compiler and runtime support for enabling generalized reduction computations on heterogeneous parallel configurations. Em *The 24th ACM Int. Conf. on Supercomputing*, pp. 137--146.

RODAMILANS, C. B. (2009). AnÁlise de desempenho de algoritmos de escalonamento de tarefas em grids computacionais usando simuladores. Em *Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Mestrado em Engenharia.*

Rossbach, C. J.; Currey, J.; Silberstein, M.; Ray, B. & Witchel, E. (2011). PTask: operating system abstractions to manage GPUs as compute devices. Em *The 23rd ACM Symposium on Operating Systems Principles*, pp. 233--248.

Smith, W.; Taylor, V. & Foster, I. (1999). Using run-time predictions to estimate queue wait times and improve scheduler performance. Em *Scheduling Strategies for Parallel Processing*, pp. 202--219. Springer-Verlag.

Teodoro, G.; Hartley, T.; Catalyurek, U. & Ferreira, R. (2010). Run-time optimizations for replicated dataflows on heterogeneous environments. Em *The 19th ACM International Symposium on High Performance Distributed Computing*, pp. 13–24.

Teodoro, G.; Hartley, T.; Catalyurek, U. & Ferreira, R. (2012). Optimizing dataflow applications on heterogeneous environments. *Cluster Computing*, 15:125–144.

Teodoro, G.; Kurc, T.; Kong, J.; Cooper, L. & Saltz, J. (2014). Comparative Performance Analysis of Intel Xeon Phi, GPU, and CPU: A Case Study from Microscopy Image Analysis. Em *28th IEEE Int. Parallel and Distributed Processing Symp.*

Teodoro, G.; Pan, T.; Kurc, T.; Kong, J.; Cooper, L.; Podhorszki, N.; Klasky, S. & Saltz, J. (2013). High-throughput analysis of large microscopy image datasets on cpu-gpu cluster platforms. Em *27th IEEE Int. Parallel and Distributed Processing Symp.*, pp. 103–114.

Teodoro, G.; Sachetto, R.; Sertel, O.; Gurcan, M.; Jr., W. M.; Catalyurek, U. & Ferreira, R. (2009). Coordinating the use of GPU and CPU for improving performance of compute intensive applications. Em *IEEE Cluster*, pp. 1–10.

Terekhov, D.; Tran, T. T.; Down, D. G. & Beck, J. C. (2013). Integrating queueing theory and scheduling for dynamic scheduling problems. *Submitted for publication.*

Ullman, J. D. (1975). Np-complete scheduling problems. *J. Comput. Syst. Sci.*, 10(3):384--393.

Williams, S.; Shalf, J.; Oliker, L.; Kamil, S.; Husbands, P. & Yelick, K. (2006). The potential of the cell processor for scientific computing. Em *Proceedings of the 3rd conference on Computing frontiers*, pp. 9--20. ACM.