UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

Julio Costella Vicenzi

EXPLORAÇÃO DE MÚLTIPLOS ALGORITMOS DE ESCALONAMENTO DE NÚCLEOS OPENCL PARA SISTEMAS DE PROCESSAMENTO EM NUVEM MULTI-INQUILINO BASEADOS EM CPUS E GPUS

Santa Maria, RS
2021

Julio Costella Vicenzi

EXPLORAÇÃO DE MÚLTIPLOS ALGORITMOS DE ESCALONAMENTO DE NÚCLEOS OPENCL PARA SISTEMAS DE PROCESSAMENTO EM NUVEM MULTI-INQUILINO BASEADOS EM CPUS E GPUS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **graudado em Engenharia de Computação**. Defesa realizada por videoconferência.

ORIENTADOR: Prof. Mateus Beck Rutzig

Santa Maria, RS
2021

Julio Costella Vicenzi

EXPLORAÇÃO DE MÚLTIPLOS ALGORITMOS DE ESCALONAMENTO DE NÚCLEOS OPENCL PARA SISTEMAS DE PROCESSAMENTO EM NUVEM MULTI-INQUILINO BASEADOS EM CPUS E GPUS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **graudado em Engenharia de Computação**.

Aprovado em 5 de janeiro de 2021:

_____

Mateus Beck Rutzig, Dr. (UFSM)
(Presidente/Orientador)

_____

Michael Guilherme Jordan, Me. (UFRGS) (videoconferência)

_____

Tiago Knorst, Eng. (UFRGS) (videoconferência)

Santa Maria, RS
2021

AGRADECIMENTOS

A realização deste trabalho representa a conclusão da etapa de graduação, a qual demandou a dedicação, auxílio e compreensão de diversas pessoas. De maneira especial agradeço:

- à Universidade Federal de Santa Maria, que dispôs infraestrutura e corpo docente capacitado a oferecer educação de qualidade durante meu período de graduação.

- aos meus pais, Nivaldo Luís Vicenzi e Marinez Costella, pelo amor, carinho e compreensão durante toda a minha jornada.

- aos meus tios, Juarez Rode e Elenice Rode pelo apoio e incentivo durante minha estadia em Santa Maria.

- ao meu orientador de iniciação científica e deste trabalho Mateus Beck Rutzig pelos conhecimentos, conselhos, paciência e direcionamentos acadêmicos e profissionais oferecidos.

- aos colegas de laboratório Michael Jordan, Tiago Knorst, Victor Oliveira Costa e Carlos Ghewer, pela cooperação em todos os trabalhos de pesquisa realizados juntos durante minha inciação científica.

- aos meus amigos de longo prazo Bruno Bertoldi, Gert Folz, Bárbara Donida, Priscila Poletti e Alexia Mainardi por sempre estarem ao meu lado.

- aos colegas de gradwuação pela amizade e colaboração nos estudos e trabalhos realizados.

A Divlab, administração da Divisão do Labor, tentava manter os casais juntos, ou então reuni-los o mais rápido possível quando o solicitavam. Mas nem sempre isso podia ser feito, especialmente durante recrutamentos especiais e urgentes, e ninguém esperava que a Divlab refizesse listas inteiras ou reprogramasse computadores para poder atendê-lo. Para sobreviver, para levar a vida adiante, um anarresti sabia que tinha de estar pronto para ir aonde ele fosse necessário, ou a fazer um trabalho que precisasse ser feito.

(Os despossuídos, Ursula Kroeber Le Guin)

RESUMO

EXPLORAÇÃO DE MÚLTIPLOS ALGORITMOS DE ESCALONAMENTO DE NÚCLEOS OPENCL PARA SISTEMAS DE PROCESSAMENTO EM NUVEM MULTI-INQUILINO BASEADOS EM CPUS E GPUS

AUTOR: Julio Costella Vicenzi
ORIENTADOR: Mateus Beck Rutzig

Computação em nuvem provê serviços de compartilhamento de recusos computacionais para a execução de aplicações dos inquilinos atráves de *computation offloading*. Para lidar com a carga computacional os nós da nuvem podem integrar unidades de processamento multinúcleos com unidades de processamento gráfico. Esta heterogeneidade do sistema abrange aplicações com massivo paralelismo ao nível de instruções, *threads* e dados, que podem ser exploradas através do modelo de programação OpenCL, levando a diminuição do tempo de execução das aplicações dos inquilinos e redução da energia consumida pelo provedor de serviços. Este trabalho propõe um *framework* para a execução de aplicações OpenCL em um ambiente de nuvem multi-inquilino formado por CPU e GPU, capaz de escalonar e escolher o melhor dispositivo de maneira transparente, sem modificação do código-fonte , com a finalidade de melhor utilizar os recursos computacionais, reduzir o tempo de execução e energia. Cinco algoritmos de escalonamento são explorados: First Come First Served, Weighted Round Robin, Max Min, Min Min e First Fit. A execução de aplicações do conjunto de benchmarks Polybench é utilizado para a avaliação do desempenho do *framework* em configurações de hardware formadas por um processador Intel Xeon Haswell combinado a NVIDIA Tesla K20m, NVIDIA Tesla K80 ou NVIDIA GeForce 1080 Ti. Dentre todas as configurações, First Fit é o algoritmo que se destaca com aceleração média de $3,04$ até $3,80$ vezes, com reduções de energia de 33% em média quando comparado a execução exclusiva da GPU.

Palavras-chave: OpenCL. Computação em nuvem. Sistemas heterogêneos. Programação paralela.

ABSTRACT

EXPLORATION OF MULTIPLE SCHEDULING ALGORITHMS FOR OPENCL KERNELS ON MULTI-TENANT CLOUD SYSTEMS BASED ON CPUS AND GPUS

AUTHOR: Julio Costella Vicenzi
ADVISOR: Mateus Beck Rutzig

Cloud computing provides services of shared computational resources for the execution of applications by tenants through computation offloading. To deal with the computational load, the cloud's nodes can integrate multi-core processing units with graphics processing units. The system's heterogeneity covers applications with massive instruction, thread and data parallelism, which can be explored through the OpenCL programming model, leading to reductions in the application's execution time for the tenants and a reduction of energy consumed to the cloud provider. This work proposes a framework for the execution of OpenCL applications on a multi-tenant cloud environment composed of a CPU and a GPU, scheduling the applications to the best device transparently, with the objective of best utilizing the computing resources, reducing execution time and energy consumed. Five scheduling algorithms are explored: First Come First Served, Weighted Round Robin, Max Min, Min Min e First Fit. The execution of the PolyBench benchmark suite is used to evaluate the performance of the framework in hardware configurations composed of an Intel Xeon Haswell processor combined with an NVIDIA Tesla K20m, NVIDIA Tesla K80 or NVIDIA GeForce 1080 Ti. For all of these configurations, First Fit is the highlighted algorithm, achieving an average acceleration of $3,04$ up to $3,80$ times and an average energy reduction of 33% when compared to GPU exclusive execution.

Keywords: OpenCL. Cloud computing. Heterogeneous Systems. Parallel programming.

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CPU	Central Processing Unit
GPU	Graphical Processing Unit
$FPGA$	Field Programmable Gate Array
DSP	Digital Signal Processor
API	Application Programming Interface
$FCFS$	First Come First Served
FF	First Fit
RR	Round Robin
WRR	Weighted Round Robin

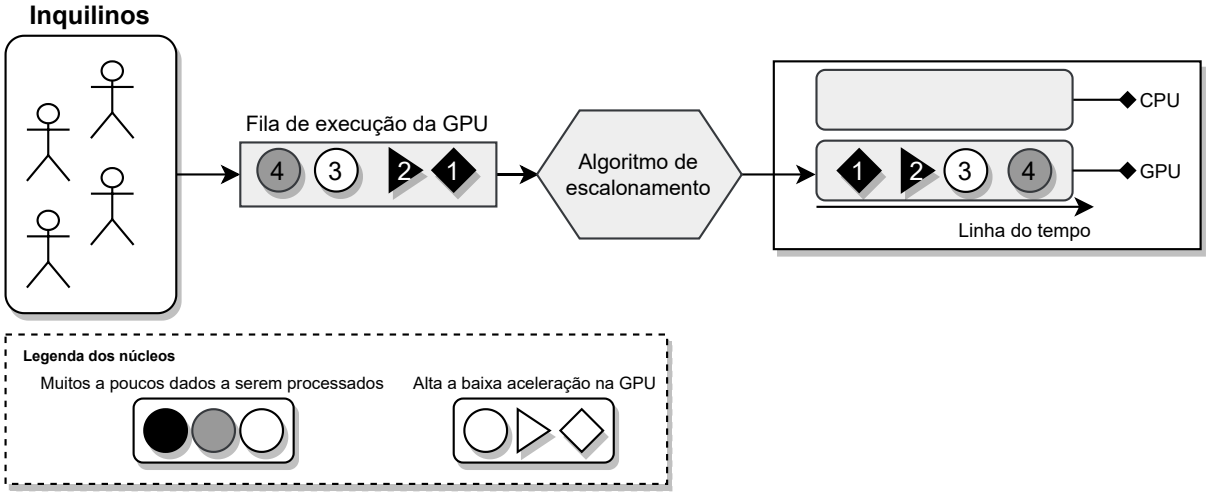
SUMÁRIO

1 INTRODUÇÃO

Computação em nuvem provê serviços de compartilhamento de recursos computacionais para executar as aplicações dos inquilinos, conhecido como processo de *computation offloading*. Para lidar com a alta carga computacional de aplicações dos inquilinos (*tenants*), cada nó da nuvem pode integrar unidades de processamento (CPU) multinúcleos e aceleradores tais como: unidades de processamento gráfico (GPUs), *Field Programmable Gate Arrays* (FPGAs) e *Digital Signal Processors* (DSPs). Esta heterogeneidade de dispositivos abrange a execução de aplicações com diferentes tipos de paralelismo. Por exemplo, aplicações com massivo paralelismo ao nível de instruções e *threads* podem ser alocadas nas CPUs enquanto aquelas com massivo paralelismo ao nível de dados podem ser escalonadas para GPUs. Para o lado do inquilino, a heterogeneidade pode acarretar benefícios no *makespan*, visto que suas aplicações podem ser aceleradas em dispositivos específicos. Em relação ao provedor de serviços (*service provider*), a variedade de dispositivos pode trazer benefícios em energia, através da maximização de quantidade de computação por watt consumido (CAMPA et al., 2014; RAVI et al., 2013).

A programação de aplicações para dispositivos heterogêneos na nuvem traz diversos desafios devido à variedade de arquiteturas e organizações envolvidas (BOOB; GONZáLEZ-VéLEZ; POPESCU, 2014). O programador deve conhecer as características de cada dispositivo e também é responsável por analisar o paralelismo inerente de cada parte de código e distribuí-las para os dispositivos mais adequados.

Figura 1.1 – Nó de nuvem com múltiplos inquilinos e um único dispositivo acelerador GPU.



Fonte: Autor.

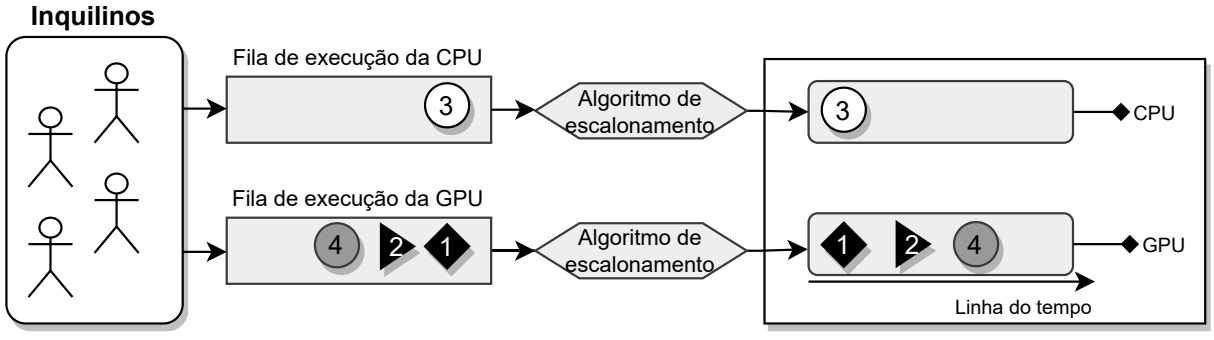
Com o intuito de suportar execução de aplicações em ambientes com dispositivos heterogêneos, a API unificada **OpenCL** provê suporte para execução em CPU, GPU, FPGA, DSP e ASIC (Khronos® OpenCL Working Group, 2021). As partes do código da

aplicação a serem aceleradas, denominadas núcleos, são definidas em tempo de programação por funções escritas na OpenCL C. Estes núcleos são compilados em tempo de execução para os dispositivos alvos. Entretanto, a responsabilidade da escolha do dispositivo alvo ainda continua sendo do programador.

A Figura 1.1 apresenta um exemplo de execução de núcleos OpenCL em um ambiente de nuvem que compartilha um nó entre vários inquilinos (sistema multi-inquilino). Nota-se que os núcleos têm oportunidades de aceleração na GPU e quantidade de dados a serem transferidas distintas, como mostra a legenda na Figura. O processo de execução inicia com os inquilinos requisitando execução na CPU e GPU. Nesse exemplo, todos os núcleos fazem requisição para executar na GPU, enfileirados e executadas por ordem de chegada. Desta forma, o núcleo 3, que tem grande oportunidade de aceleração na GPU e processa poucos dados é atrasada, por ser enfileirada depois de 1, que processa muitos dados, mas não provê grande aceleração quando executado na GPU. Também, neste cenário, como os programadores requisitaram execução na GPU, todos os núcleos são presos a este dispositivo, não explorando o completo poder computacional do nó pela inutilização da CPU.

A Figura 1.2 apresenta o mesmo sistema de CPU e GPU da Figura 1.1. Entretanto, nesse cenário os inquilinos fazem requisições de execuções para ambos dispositivos. Os núcleos são processados em paralelo, permitindo que 1 e 3 utilizem a CPU e GPU simultaneamente, maximizando o uso dos do hardware disponível. Entretanto, nesse sistema, o programador fica com a responsabilidade de escolher o melhor dispositivo para cada núcleo e cada dispositivo tem sua fila individual para a execução e escalonamento em tempo de execução. Isso respeita a decisão do programador, que não tem acesso a informações de tempo de execução para prever o estado do sistema e disponibilidade dos recursos.

Figura 1.2 – Nó de nuvem com múltiplos inquilinos e uma CPU multinúcleo e uma GPU, com uma fila para cada dispositivo.



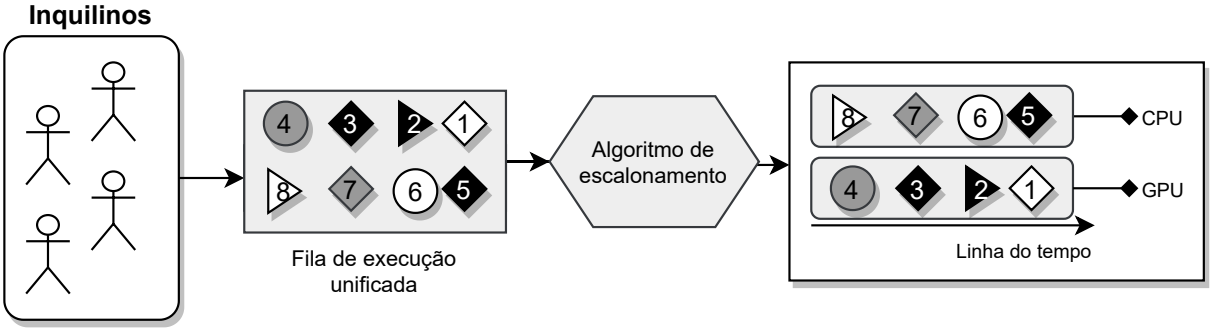
Fonte: Autor.

Os sistemas em nuvem estão em constante evolução, assim diferentes arquiteturas de GPUs em diferentes nós é natural nos atuais sistemas. Como esta expansão de GPUs ocorre durante anos, é comum que se crie nós com dispositivos com diferentes capacidades de processamento. Neste cenário surge uma nova variável para o escalonamento,

além de escalonar aplicações para diferentes dispositivos (CPU ou GPU), o escalonador deve se adaptar a diferença de capacidade computacional entre os dispositivos.

A Figura 1.3 ilustra o cenário supracitado. Nesta Figura, a CPU é utilizada em conjunto com uma GPU e uma única fila de execução, aumentando a utilização do hardware.

Figura 1.3 – Nó da nuvem multi-inquilino com CPU e GPU. Um único algorítimo de escalonado seleciona o dispositivo em que cada núcleo é executado.



Fonte: Autor.

Como notado na discussão acima, uso de um escalonador ideal para o sistema é fundamental para obter-se o melhor desempenho do sistema de nuvem multi-inquilino. Desta maneira, este trabalho propõe um *framework* transparente para o escalonamento de núcleos OpenCL em um nó de nuvem multi-inquilino, explorando diferentes algoritmos de escalonamento e avaliando o desempenho de tempo de execução e consumo de energia em diferentes configurações de CPU e GPU.

2 TRABALHOS RELACIONADOS

A execução conjunta de aplicações em CPU e GPU tem sido explorada de maneiras mais disruptivas do modelo de programação das aplicações, através de modificações ao código-fonte e uso de bibliotecas que encapsulam as funções OpenCL diretamente, assim como pelo uso de *frameworks* mais robustos ao nível de sistema, de maneira transparente ao modelo de programação.

Visando a otimização de aplicações individuais, EngineCL é uma API C++ proposta por Guzmán et al. (2019) focada em usabilidade e desempenho para sistemas heterogêneos, incluindo CPU, GPU e FPGA. A biblioteca explora três escalonadores diferentes (estático, dinâmico e HGuided) com o intuito de balancear a carga de trabalho entre os dispositivos e, consequentemente, fornecer ganhos de *Energy Delay Product* (EDP) quando comparado a execução em um único dispositivo. Entretanto, o programador ainda é responsável por decidir qual o melhor dispositivo deve executar cada parte da aplicação.

Visando o escalonamento de tarefas de maneira automática, Sun et al. (2012) propõe um escalonador baseado em *work-pools* e *task queuing*. A API proposta automatiza o uso de todos os dispositivos do sistema, implementando um escalonador estático baseado em particionamento de tarefas, selecionando o melhor dispositivo.

Usando um modelo preditivo e explorando as características da aplicação, Grewe e O'Boyle (2011) desenvolve um esquema de particionamento estático de programas OpenCL para CPU e GPU. O sistema usa uma máquina de vetores de suporte (*support vector machine*) e análise do código de núcleos usando CLANG. A carga computacional é particionada em incrementos de 10%, permitindo a execução exclusiva em CPU ou GPU.

Para soluções ao nível de sistema, Wen, Wang e O'Boyle (2014) utiliza um escalonador dinâmico para determinar o dispositivo que terá melhor aceleração para o núcleo, focando em execução de múltiplas aplicações que utilizam CPU e GPU. O escalonador seleciona o melhor dispositivo baseado em uma máquina de vetores de suporte e dados gerados durante a execução. Os resultados mostram um aumento no *throughput* e redução no *turnaround* para sistemas com dispositivos NVIDIA e AMD.

Riebler et al. (2019) apresenta HTrOP, uma infraestrutura completa para a detecção de laços paralelizáveis no código da aplicação, geração de código OpenCL automático e decisão de escalonamento para CPU e GPU. O escalonador é implementado no sistema *runtime* OpenCL, sendo capaz de fazer escolhas baseadas na disponibilidade dos aceleradores.

Diversos trabalhos já consideraram escalonamento de aplicações em múltiplos dispositivos heterogêneos. Kim et al. (2011) cria uma abstração de uma única imagem de dispositivo, fazendo a distribuição de núcleos OpenCL, em tempo de execução, para diversas GPUs. Este processo depende de modificações no código-fonte da aplicação, mas

garante que uma aplicação codificada para uma única GPU tenha portabilidade para execução em ambientes com múltiplas GPUs.

Taylor, Marco e Wang (2017) mapeia núcleos OpenCL para processadores heterogêneos em uma plataforma ARM big.LITTLE. O mapeamento é decidido por uma heurística de otimização treinada em tempo de projeto.

Massari et al. (2014) explora a execução de aplicações em OpenCL em sistemas com CPU e múltiplas GPUs. RTLib é proposto como uma extensão da biblioteca de *runtime* do OpenCL, com o *profiling* da aplicação e escalonamento heterogêneo. Essas características permitem a seleção automática de dispositivos heterogêneos. Isto é feito de maneira completamente transparente a aplicação OpenCL, mantendo sua portabilidade.

Este projeto propõe um *framework* capaz de fazer o escalonamento de núcleos OpenCL em um sistema de computação em nuvem multi-inquilino com CPU-GPU. Diferente de:

- Guzmán et al. (2019), Grewe e O'Boyle (2011), Kim et al. (2011), Sun et al. (2012), o *framework* proposto não requer a modificação ou recompilação do código-fonte das aplicações;

- Massari et al. (2014), Riebler et al. (2019), o *framework* proposto provê diversos algoritmos de escalonamento, explorando as características fundamentais dos núcleos com o intuito de maximizar a utilização do hardware e minimizar o tempo de execução das aplicações dos inquilinos;

3 FRAMEWORK

3.1 MODELO DE PROGRAMAÇÃO OPENCL

OpenCL é um padrão para a programação de programas capazes de serem utilizados em plataformas heterogêneas consistindo de CPUs, GPUs, FPGAs, DSPs ou ASICs, criado em 2009 pelo consórcio industrial Khronos Group, sendo um padrão aberto definido em Khronos® OpenCL Working Group (2021).

A visão do sistema do OpenCL consiste em aceleradores de hardware conectados a um processador hospedeiro. Funções que são executadas nos dispositivos são chamados **núcleos** (*kernels*), e programados em uma linguagem baseada em C. Para a execução de núcleos, a aplicação executada no hospedeiro faz uso da API, disponível em C e C++, para comunicação com a biblioteca de *runtime* OpenCL. Cada vendedor implementa separadamente um *driver* para o funcionamento do dispositivo. Desta maneira, dispositivos do mesmo vendedor são agrupados em plataformas distintas.

Um *context* é a abstração de um contêiner no hóspede, com informações para a coordenação entre dispositivos. Deve-se definir a plataforma e um conjunto de dispositivos que serão utilizados, e através de um *context*, define-se *command_queue* para cada dispositivo separadamente, permitindo o enfileiramento de comandos de leitura e escrita de memória e execução de núcleos. Esses comandos podem ser associadas a *events* que provêm informações de *profiling* e de estado enfileirados.

Para a execução, em tempo de execução o código de núcleos escrito em OpenCL C é compilado em um *program*. Para a execução dos núcleos, são passados argumentos e transferido dados para a memória do dispositivo através de seu respectivo *command_queue* e o enfileiramento do núcleo. Dado que a execução foi completa, os resultados são então lidos pelo hóspede.

A unidade de execução em OpenCL é *work-item*, agrupados em *work-groups* que são processados de maneira conjunta. Assim, mapeia-se cada elemento a ser processado a um *work-item*, que precisa ser definido ao enfileirar o núcleo. Um número maior de *work-items* implica em uma carga computacional maior.

3.2 VISÃO GERAL

O *framework* proposto neste trabalho é baseado em ambientes de execução em nuvem multi-inquilino, composta por nós de computação com uma CPU e uma GPU para a

aceleração de aplicações. Neste contexto, múltiplos inquilinos podem solicitar a execução de um núcleo OpenCL de maneira assíncrona e concorrente. Assim, os recursos computacionais do nó da nuvem são compartilhados temporalmente. Para melhor distribuição dos recursos em tempo de execução, focando em melhora de tempo de *makespan* e menor consumo de energia, o *framework* é capaz de:

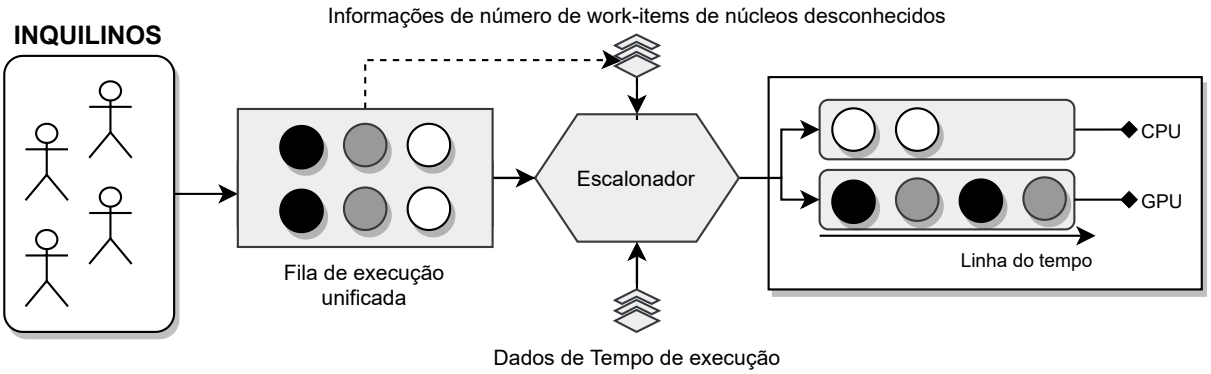
- Suplantar as chamadas a biblioteca de tempo de execução OpenCL das aplicações dos inquilinos, sem a necessidade de mudanças ou recompilação de código-fonte;
- Agregar as requisições de execução de núcleos em uma única fila unificada;
- Implementar um conjunto diverso de algoritmos de escalonamento;
- Selecionar o melhor dispositivo (CPU ou GPU) para cada núcleo a ser executado;
- Selecionar a ordem de execução dos núcleos em cada dispositivo;
- Manter informações de tempo de execução de núcleos para serem usadas pelos algoritmos de escalonamento para perfeicionar a decisão de escolha de dispositivos;

Como visto na Figura 3.1, núcleos OpenCL são enfileirados pelos múltiplos inquilinos da nuvem em uma fila unificada de execução. Como a aplicação original usando API OpenCL deve fazer chamadas de seleção de plataforma e dispositivo, e a criação de uma fila de execução, estas são ignoradas. O *framework* modifica a chamada de compilação de um núcleo para um dispositivo, gerando código para todos os dispositivos do sistema. As chamadas de escrita, leitura e execução em um dispositivo são então empacotadas como uma requisição de execução de núcleo, direcionadas a fila unificada de execução, tornando responsabilidade do algoritmo de escalonamento a seleção do dispositivo apropriado.

Um conjunto de núcleos da fila de execução é transformado em um lote. Os núcleos do lote são então escalonados para os dispositivos pelo algoritmo de escalonamento. Assim que este lote foi processado, os novos núcleos enfileirados são de mesma forma escalonados. A execução de múltiplos lotes em sequência não são abordados nos experimentos deste trabalho.

No momento em que um núcleo é executado, um *event* OpenCL é utilizado para obter-se o seu tempo de execução no dispositivo. Este dado é utilizado para considerar-se a aceleração da execução do núcleo na GPU sobre a CPU utilizada por alguns dos algoritmos de escalonamento. Para a primeira execução de um novo núcleo desconhecido, a aceleração é estimada baseando-se no número de *work-items* do núcleo. Quanto maior o número de *work-items*, maior a aceleração inicialmente prevista. A exploração do impacto de núcleos desconhecidos está fora do escopo deste trabalho.

Figura 3.1 – Visão geral do *framework*. Os círculos representam diferentes núcleos OpenCL



Fonte: Autor.

3.3 ESCALONADORES

Esta seção inclui o conjunto de algoritmos de escalonamento utilizados para avaliar a funcionalidade e desempenho do *framework* apresentado.

3.3.1 First Come First Served

First Come First Served (FCFS) é um algoritmo que segue o ordenamento dos núcleos do lote, dando prioridade a execução a GPU, como visto no Algoritmo 1. O primeiro núcleo é sempre alocado a GPU, e o seguinte a CPU. O algoritmo aguarda então algum dos dispositivos acabar a execução do núcleo e imediatamente escalona o próximo núcleo a este dispositivo.

Diferente dos algoritmos que serão apresentados a seguir, FCFS não tem uma visão dos núcleos no lote, nem da capacidade de processamento dos dispositivos. A única função é garantir que ambos os dispositivos estejam ocupados a qualquer instante. Embora isso garanta que o hardware seja utilizado enquanto há dados a serem processados, nem sempre o uso de ambos dispositivos é benéfico para o tempo de execução. Por exemplo, um lote formado apenas de núcleos com alta aceleração na GPU pode ser melhor escalonado ignorando a CPU como dispositivo.

Algoritmo 1: First Come First Served.

Entrada: lote

Dados: cpuFila, gpuFila

para $nucleo \in lote$ **faça**
    Espere um dos dispositivos ficar ocioso.;
    se *gpuFila.vazio()* ***AND*** *cpuFila.vazio()* **então**
        $gpuFila \leftarrow nucleo$;
    senão se *cpuFila.vazio()* **então**
        $cpuFila \leftarrow nucleo$;
    senão
        $gpuFila \leftarrow nucleo$;
fim

3.3.2 Weighted Round Robin

Weighted Round Robin (WRR) é um algoritmo que enfileira os primeiros $gpuPeso$ núcleos a GPU e o próximo à CPU, se repetindo até todos os núcleos do lote estarem enfileirados, como visto no algoritmo 2. O valor de $gpuPeso$ se refere a razão de núcleos que serão alocados a GPU a CPU. Quando $gpuPeso = 1$, os núcleos serão divididos igualmente entre a CPU e a GPU, denominado apenas Round Robin.

Usando o parâmetro $gpuPeso$ é possível dar prioridade de execução para o dispositivo GPU, balanceando as diferenças de desempenho entre os dispositivos. Por exemplo, se na média a GPU pode acelerar a execução em 2 vezes, $gpuPeso$ deve seguir este valor para balancear a carga. Isto pode ser um problema, entretanto. Nem todos os núcleos terão tempo de execução reduzido. A razão exata entre núcleos que precisam ser escalonados para a GPU em relação à CPU é então altamente dependente dos núcleos encontrados no lote.

Algoritmo 2: Weighted Round Robin.

Entrada: lote, gpuPeso

Dados: cpuFila, gpuFila

para $c \in 0$ **até** $tamanho(lote)$ **faça**
    se $c \mod (gpuPeso + 1) \neq 0$ **então**
        $gpuFila \leftarrow nucleo$;
    senão
        $cpuFila \leftarrow nucleo$;
    fim
fim

3.3.3 Max Min e Min Min

O Algoritmo 3 mostra Max Min, que primeiramente ordena o lote baseado na aceleração da GPU decrescente, e então enfileira os primeiros n núcleos com maior aceleração para a GPU. Os núcleos remanescentes são enfileirados para a CPU. Isto garante que os núcleos com maior aceleração serão executados nos dispositivos que diminuem o tempo de execução, mantendo uma distribuição dos núcleos entre os dispositivos, evitando a ociosidade e priorizando *makespan*.

Algoritmo 3: Max Min.

Entrada: lote, n

Dados: cpuFila, gpuFila

1 *Ordenar lote por aceleração da GPU decrescente.*;

2 $gpuFila \leftarrow lote[0$ **até** $n]$;

`// Os primeiros n núcleos são enfileirados na GPU.`

3 $cpuFila \leftarrow lote[n$ **até** $fim]$;

`// O resto dos núcleos são enfileirados na CPU.`

De maneira inversa, Min Min primeiramente ordena o lote por aceleração da GPU crescente, e então enfileira os primeiros n núcleos com menor aceleração a CPU, como visto no Algoritmo 4. Os núcleos remanescentes são enfileirados a GPU. Esta abordagem garante que núcleos sem aceleração na GPU utilizem a CPU, e ainda mantendo um número mínimo de núcleos a serem executados na GPU e priorizando menor tempo de espera.

Para ambos os algoritmos, o parâmetro n é fundamental para um escalonamento justo, visto que muitos núcleos podem ser escalonados para um dispositivo, causando um desbalanço da carga computacional. De maneira inversa, eles garantem que os núcleos com maior aceleração se beneficiem do dispositivo apropriado e ambos os dispositivos serão utilizados de maneira paralela. Em casos onde todos os núcleos têm grande aceleração na GPU, por exemplo, o algoritmo pode acabar enfileirando a CPU de maneira a atrasar o tempo de execução do lote todo.

Algoritmo 4: Min Min.

Entrada: lote, n

Dados: cpuFila, gpuFila

1 *Ordenar lote por aceleração da GPU crescente*;

2 $cpuFila \leftarrow lote[0$ **até** $n]$;

`// Os primeiros n núcleos são enfileirados para a CPU.`

3 $gpuFila \leftarrow lote[n$ **até** $fim]$;

`// O resto dos núcleos são enfileirados na GPU.`

3.3.4 First Fit

First Fit (FF) é um algoritmo que escalona os núcleos unicamente pelo parâmetro $minAcel$, que deve ser um valor referente a mínima aceleração do tempo de execução da GPU sobre a CPU, como visto no Algoritmo 1. Este valor determina então qual dispositivo levará mais tempo para processar o núcleo, com valores entre $0 < x < 1$ representando aceleração na CPU e valores $x > 1$ representando aceleração na GPU. Assim, todos os núcleos que tiverem aceleração $x \geq minAcel$ serão enfileirados para a GPU e os restantes para a CPU.

Este algoritmo não garante que ambos os dispositivos serão utilizados, visto que um lote pode conter apenas algoritmos com pouca aceleração na GPU, deixando o dispositivo ocioso. Entretanto, ele garante que a aceleração não vai ser desperdiçada pela execução em um dispositivo que aumente o tempo de processamento do núcleo.

Algoritmo 5: First Fit.

Entrada: lote, minAcel
Dados: cpuFila, gpuFila

```
1 para nucleo ∈ lote faça
2     se nucleo.gpuSpeedup ≥ minAcel então
3         gpuFila ← nucleo ;
4     senão
5         cpuFila ← nucleo ;
6     fim
7 fim
```

4 METODOLOGIA

4.1 BENCHMARKS

O conjunto de *benchmarks* **PolyBench** foi utilizado para a avaliação de desempenho (GRAUER-GRAY et al., 2012). Este conjunto é bastante diversificado, contendo núcleos de álgebra linear, mineração de dados e estênceis. Embora os núcleos são programados e otimizados para a execução em GPU, neste trabalho os dois dispositivos usam o mesmo código OpenCL, usando a sua interface transparente para diferentes plataformas. Todos os arquivos executáveis foram compilados usando **GCC versão 10.2.0** com otimizações de nível O3 para o código C (STALLMAN; DEVELOPERCOMMUNITY, 2009), enquanto os núcleos OpenCL foram compilados em tempo de execução sem nenhuma opção de compilação adicional. O número de *work-items* em cada núcleo de aplicação não é alterado.

4.2 AMBIENTE DE EXECUÇÃO

O ambiente utilizado para a execução das *benchmarks* é composto por um processador **Intel Xeon E5-2650 v3 Haswell**, conforme mostrado na tabela 4.1. Três diferentes unidades de processamento gráfico foram utilizadas para a avaliação do desempenho, como visto na tabela 4.2. Tanto **NVIDIA Tesla K80** e **NVIDIA Tesla K20m** tem características similares, embora a K20m tenha uma frequência de relógio menor e menos VRAM. Importante notar que embora K80 apresente dois dispositivos OpenCL na mesma placa, apenas um destes dispositivos é usado para o processamento, devido ao fato do escopo do trabalho se limitar a configurações com uma CPU e uma GPU. A **NVIDIA GeForce GTX 1080 Ti** é significativamente mais poderosa que as outras duas, usando o dobro de frequência de relógio, e com um aumento no número de *Cuda Cores*, que ditam a capacidade de processamento paralelo do dispositivo. O sistema operacional utilizado é **Ubuntu 18.04.4 LTS**.

Tabela 4.1 – Especificações da Unidade Central de Processamento utilizada para avaliação.

Especificações de CPU	
Modelo	Intel Xeon E5-2650 v3 Haswell
Frequência de relógio	2.3 GHZ
Memória Cache	25 MiB
RAM	128 GiB
TDP	105 W
Versão OpenCL	1.2
Compute Units	40
Núcleos	10
Threads	20

Tabela 4.2 – Especificações das Unidades de Computação Gráfica Utilizadas para avaliação.

Especificações de GPUs			
Modelo	NVIDIA Tesla K80	NVIDIA Tesla K20m	NVIDIA GeForce GTX 1080 Ti
Frequência de relógio	823 MHz	705 MHz	1670 MHz
Memória Local	48 KiB	48 KiB	48 KiB
Memória Global	208 KiB	208 KiB	1,312 MiB
VRAM	12 GiB	4,63 GiB	10,91 GiB
Máxima potência	149 W	225 W	250 W
Versão de Driver	455.32	440.33.01	465.19.01
Versão OpenCL	1.2	1.2	1.2
Compute Units	13	13	28
Cuda Cores	2496	2496	3584

Para a avaliação de informações do tempo de execução utilizou-se a ferramenta **perf versão 4.19.171** em conjunto com a biblioteca **Intel RAPL** para a avaliação da energia gasta pela CPU. A ferramenta **NVIDIA-SMI** foi usada para a extração de amostras de potência a cada $200ms$ das GPUs. A energia total do sistema foi aproximada pela Equação 4.1, onde P_i é a i-ésima amostra de potência e Δt o tempo entre cada amostra (HERBET, 1985).

$$E = \sum_{i=0}^{n} P_i \, \Delta t \tag{4.1}$$

Em ambos os casos, toda a energia utilizada pelo pacote do circuito integrado é considerada para uma comparação justa do hardware.

4.3 CENÁRIOS DE EXECUÇÃO

O *framework* utiliza lotes de núcleos enfileirados pelos inquilinos e para avaliar o seu desempenho de maneira justa diferentes conjuntos de núcleos foram criados para representar possíveis cenários de execução. A intenção é cobrir o maior número de casos específicos, buscando entender como cada algoritmo de escalonamento se comporta.

Após uma avaliação das características das *benchmarks* escolhidas, três características foram consideradas para a separação de cenários:

- Número de dados transferidos para a memória do dispositivo;
- Aceleração da execução na GPU sobre a CPU;
- Tempo de execução do núcleo na CPU.

A transferência de dados para a memória do dispositivo afeta desproporcionalmente a execução em GPU pelo tempo da transferência dos dados da memória RAM do sistema para a VRAM do dispositivo, adicionando um atraso inicial antes da computação. A aceleração das execuções nos diferentes dispositivos são casos muito importantes, pois nem todos os núcleos têm aceleração quando executados na GPU, e isto é um fator decisivo para muitos algoritmos de escalonamento. Finalmente, o tempo de execução do núcleo na CPU indica, de maneira geral, o tempo de execução para os outros dispositivos. Núcleos com maior tempo de execução podem causar problemas diferentes de núcleos muito curtos.

Para cada uma destas características, foi criado lotes com os núcleos que apresentavam os maiores valores e lotes com os menores valores. A ordenação dos lotes é dada em ordem decrescente para lotes de valores altos, e crescente para lotes de valores baixos. Estes lotes tem então características muito homogêneas. Para adicionar-se cenários de execução, lotes mistos com parte dos núcleos de maior e menor valor. Finalmente, um lote contendo todos os núcleos do conjunto de *benchmarks* também foi criado, para avaliar uma execução sem nenhuma característica específica pré determinada.

A tabela 4.3 apresenta os núcleos de cada lote executado, assim como abreviações que serão utilizadas no texto para se referir aos diferentes cenários durante o resto do texto.

Tabela 4.3 – Cenários propostos para avaliação do *framework*

Características	Abreviação	Núcleos ordenados
Todos os núcleos	Todos	gemver, bicg, lu, 3mm, 2mm, 2DConvolution, gesumv, syr2k, fdtd2d, doitgen, covariance, mvt, atax, correlation, jacobi2D, syrk , adi, gemm
Maior transferência de memória	M+	jacobi2D, mvt, gemver, gesummv, 2Dconvolution, doitgen, bicg, atax
Mista transferência de memória	M/	covariance, correlation, syr2k, jacobi2D, mvt, gemver, gesummv
Menor transferência de memória	M-	covariance, correlation, syr2k, lu, 2mm, 3mm
Maior tempo de execução na CPU	T+	gemm, gemver, adi, correlation, covariance, mvt, 2mm, doitgen, 2mm
Misto tempo de execução na CPU	T/	gemm, gemver, adi, correlation, 2Dconvolution, syr2k, gesummv, jacobi2D, fdtd2d
Menor tempo de execução na CPU	T-	2Dconvolution, syr2k, gesummv, jacobi2D, fdtd2d, lu
Maior aceleração na GPU	A+	3mm, gemm, 2mm, correlation, covariance
Mista aceleração na GPU	A/	syr2k, syrk, gesummv, 3mm, gemm, 2mm, correlation
Menor aceleração na GPU	A-	syr2k, syrk, gesummv, 2Dconvolution, doitgen

Fonte: Autor

4.4 EXPERIMENTOS

Para cada GPU mostrada na Tabela 4.2 em conjunto com o CPU mostrado na Tabela 4.1, foi executado o escalonamento dos cenários mostrados na Tabela 4.3, para cada um dos algoritmos de escalonamento discutidos na seção 3.3, assim como a execução do lote inteiramente na CPU e inteiramente na GPU para se verificar a eficácia da execução distribuída entre os dispositivos.

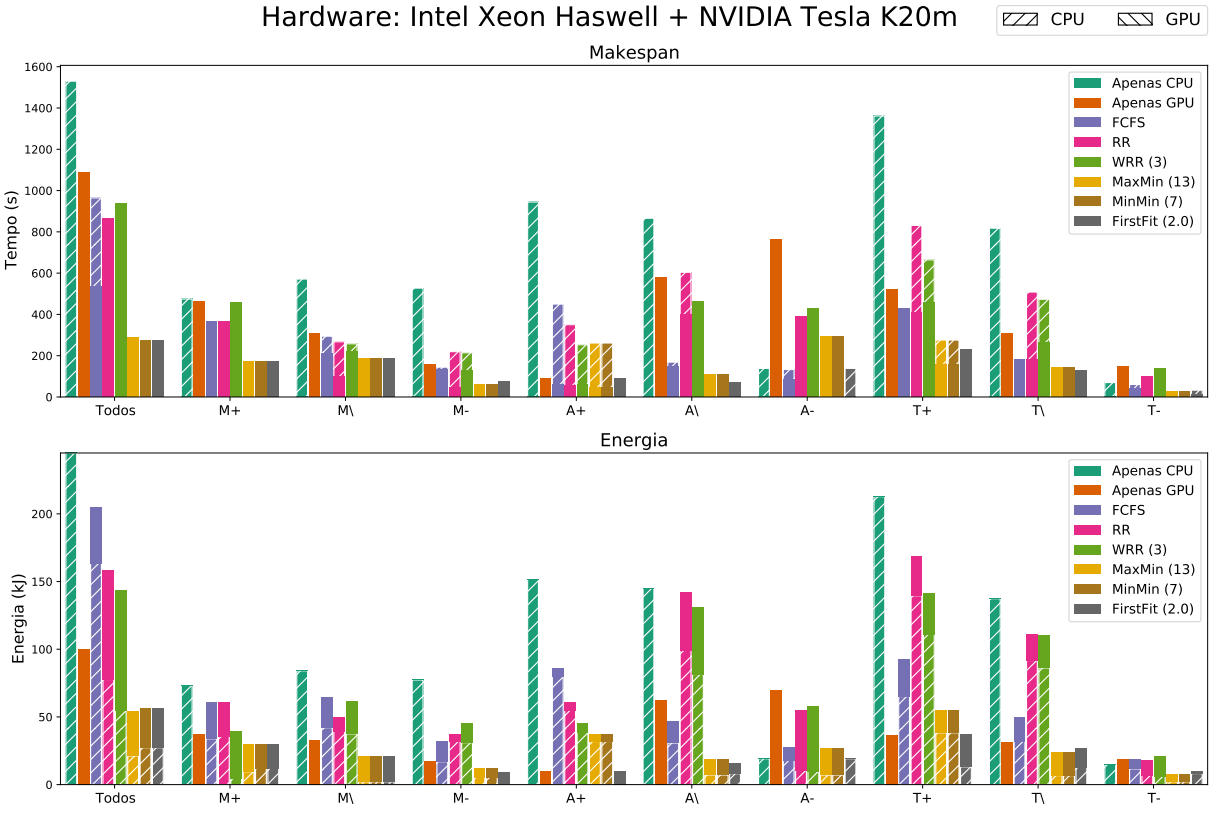
A execução é feita de modo a simular o funcionamento do *framework*. Isto é possível utilizando dados obtidos da execução de cada *benchmark* em todos os dispositivos individualmente. Assim, com o conjunto de dados de execução é então feito o escalonamento dos núcleos.

Os parâmetros dos algoritmos foram escolhidos de modo a tentar ser o mais justo possível e obter melhor desempenho sem uma exploração exaustiva de cada caso. Para Weighted Round Robin, uma versão com peso 3 e uma versão de Round Robin (peso 1). Max Min e Min Min tem como parâmetro para a alocação de $3/4$ e $2/5$ dos núcleos do lote para a GPU e CPU, respectivamente. First Fit usa um valor de aceleração mínimo fixado em 2.

5 RESULTADOS E DISCUSSÃO

Os Gráficos 5.1, 5.2 e 5.3 apresentam os dados de execução dos diferentes cenários propostos na seção 4.4. Para as informações de tempo, as barras representando a CPU e GPU estão sobrepostas, mostrando o valor do dispositivo mais demorado na execução dos núcleos enfileirados do lote, distinguidos pelas hachuras. Para gráficos de energia as barras estão empilhadas, mostrando a soma das energias de cada dispositivo utilizado, resultando na energia total do sistema para a execução do lote. As hachuras das barras representam a contribuição de cada dispositivo. Para os algoritmos de alocação que dependem de parâmetros adicionais, estes são apresentados entre parenteses após seu nome na legenda.

Gráfico 5.1 – Tempo de execução e energia do sistema para a configuração Intel Xeon Haswell + NVIDIA Tesla K20m.



Fonte: Autor.

Comparando as execuções dos lotes apenas na CPU e apenas na GPU para a configuração Intel Xeon Haswell + NVIDIA Tesla K20m na Figura 5.1, a execução exclusiva em GPU sempre é mais eficiente, tanto em tempo de execução e energia. Isto é esperado, visto que mesmo que a potência utilizada pela NVIDIA Tesla K20m seja $2,1$ vezes da CPU, a redução no tempo de execução de $1,4$ vezes leva a uma reduções de energia de $58,8\%$, considerando o cenário com todos os núcleos. Como o conjunto de *benchmarks* enfoca a

programação para GPU, os dados de aceleração e energia neste capítulo fazem comparação a execução exclusiva na GPU, quando não informados. O único cenário em que a execução exclusiva na GPU não redução de tempo e energia é para o lote A-, onde todos os núcleos têm maior aceleração sendo executadas na CPU, com redução de $5,4$ vezes de tempo e $72,2\%$ de energia. Assim, fica claro que embora o conjunto de *benchmarks* ser programado com a intenção de execução em um determinado hardware de GPU, para este conjunto de núcleos nas condições testadas, a CPU tem tempo de execução reduzido utilizando o mesmo código de núcleo OpenCL. A exploração da capacidade de reutilização do código é parte fundamental do *framework* proposto, que considera a capacidade da execução em plataformas heterogêneas com uso de OpenCL.

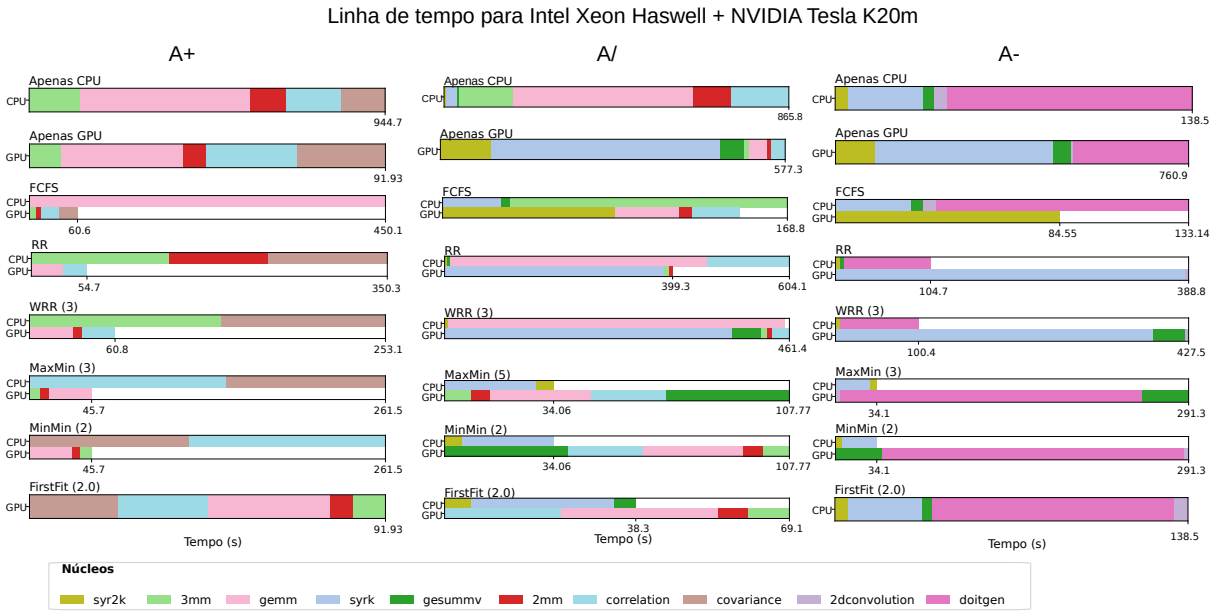
Quando se considera o lote A+, a utilização exclusiva da GPU sempre tem o menor tempo de execução, pois a execução na GPU é $10,3$ vezes mais rápida do que na CPU, fazendo que qualquer núcleo executado no dispositivo atrase todo o lote. Como visto na Figura 5.1, nenhum núcleo executado na CPU tem tempo de execução menor que a soma de todos os núcleos executados exclusivamente na GPU, assim o escalonamento de um único núcleo ao dispositivo causa um atraso em todo o lote. De maneira oposta, no lote A- o algoritmo FCFS é capaz de executar o núcleo syr2k na GPU sem atrasar o lote, porém aumentando a energia utilizada quando comparada a execução apenas CPU em $29,5\%$, pelo fato de ambos dispositivos serem utilizados para uma aceleração de apenas $1,04$ vezes.

A execução exclusiva não é ideal, levando a ociosidade e o uso de dispositivos menos otimizados para o núcleo processado. O *framework* proposto utiliza todos os recursos do sistema e é capaz de diminuir os tempos de execução em até $8,35$ vezes. Dentre os algoritmos explorados, RR e WRR não obtiveram melhora quando comparado a execução apenas na GPU, tendo em média uma desaceleração de $0,72$ e $0,76$ vezes. FCFS consegue em média uma aceleração de $1,23$ em tempo de execução, tendo desaceleração no cenário A+. Max Min e Min Min tem aceleração de $2,90$ e $2,88$ vezes, respectivamente, também tendo atrasos no mesmo cenário. O único algoritmo com aceleração em todos os cenários é FF, sendo em média $3,45$ vezes mais rápido em comparação a GPU e $5,69$ vezes em comparação a CPU. Mesmo nos cenário A+ e A-, ele é capaz de alcançar o tempo de execução idêntico ao cenário com apenas a GPU e CPU. Isso é possível pela estratégia baseada na análise de apenas a aceleração dos núcleos, sem restrições quanto a distribuição igualitária entre os dispositivos, diferente de todos os outros algoritmos.

Considerando a energia consumida por cada algoritmo, RR, WRR e FCFS aumentam a energia utilizada pelo sistema em média em 43%, 42% e 28%. Max Min e Min Min obtém uma redução de 24% em média, e FF tem uma melhora de 36%. Quando se considera a execução conjunta, o consumo de dois dispositivos simultaneamente pode levar ao aumento do consumo total, mas esse efeito é amortecido pela aceleração considerável em comparação a execução em um único dispositivo. Então, a escolha do melhor dispositivo

para a execução baseado no tempo de execução um fator essencial, favorecendo Min Min, Max Min e First Fit.

Figura 5.1 – Linha de tempo para os cenários A+, A/ e A-.



Fonte: Autor.

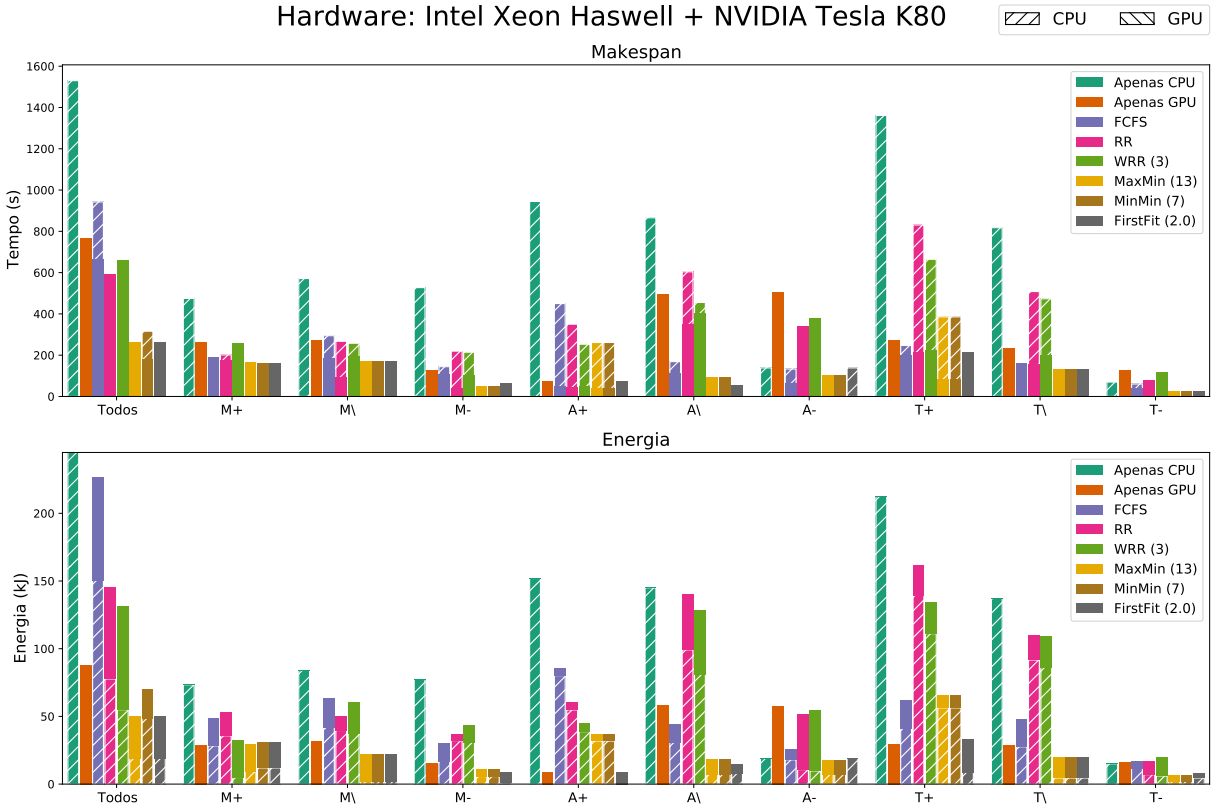
Round Robin e Weighted Round Robin têm os maiores tempos de execução quando comparado aos outros algoritmos, não trazendo diminuição no tempo de execução através da distribuição da execução entre os dispositivos. Mesmo com pesos diferentes dando prioridade para a GPU, os maiores ganhos de tempo de execução acontecem para o lote A- com aceleração de $1,96$ vezes em comparação a GPU, mas uma desaceleração de $0,32$ vezes quando comparado a CPU. Isso é devido ao fato do algoritmo assumir aceleração da GPU e não ter contexto para a disponibilidade dos dispositivos, fazendo apenas uma distribuição sem critérios específicos aos núcleos. Em muitos casos, o algoritmo é pior que a execução exclusiva dos dispositivos, fazendo a execução paralela irrelevante.

Para First Come First Served, mesmo sem informações de tempo de execução dos núcleos, é possível se obter bons resultados em alguns casos, como nos lotes A/, T-, aceleração de $3,42$, e $2,46$, onde o algoritmo mantém a carga computacional dos dispositivos balanceados, como visto na Figura 5.1. De qualquer maneira, situações como A+ leva ao pior desempenho entre os algoritmos, sendo 5 vezes mais demorado. Para os demais cenários, há pouca melhora no tempo de execução, levando a um aumento na energia utilizada pelo sistema de até 89% no cenário A+. A falta de informações de tempo de execução prejudica em casos onde a aceleração dos núcleos é muito grande, levando o algoritmo a escolher dispositivos que levam muito tempo para a execução do núcleo, atrasando todo o sistema, como visto na Figura 5.1 no cenário A+, onde o gemm toma todo o tempo da CPU enquanto a GPU termina o processamento do restante dos núcleos em uma fração desse tempo.

Max Min e Min Min tem desempenho semelhante, em algumas situações levando a mesma alocação entre os dispositivos, diferindo apenas em ordenação. Para o cenário com todos os núcleos, os algoritmos atingem aceleração de $3,96$ vezes. Entretanto, mesmo considerando a aceleração dos núcleos, os cenários A+ e A- e causam desaceleração de até $0,62$ vezes, pelo fato deles sempre alocarem um número mínimo de núcleos em cada dispositivo.

Para todos os casos, First Fit é o melhor ou o segundo melhor algoritmo para a alocação, tendo uma aceleração de $3,96$ no cenário com todos os núcleos, e se igualando a execução apenas na GPU para o lote A+. Seu maior ganho é para o cenário A/, com uma aceleração de $8,35$ vezes e redução de 75% de energia. A capacidade de adaptação em situações em que a execução conjunta não é benéfica permite que mesmo em cenários como A+ e A-, o algoritmo evite o consumo ineficaz de energia utilizando dispositivos inapropriados. É importante notar que esses são casos especiais, e para lotes mistos, o seu desempenho acaba sendo similar a Max Min e Min Min, tornando First Fit o melhor algoritmo de escalonamento para a configuração Intel Xeon Haswell + NVIDIA Telsa K20m.

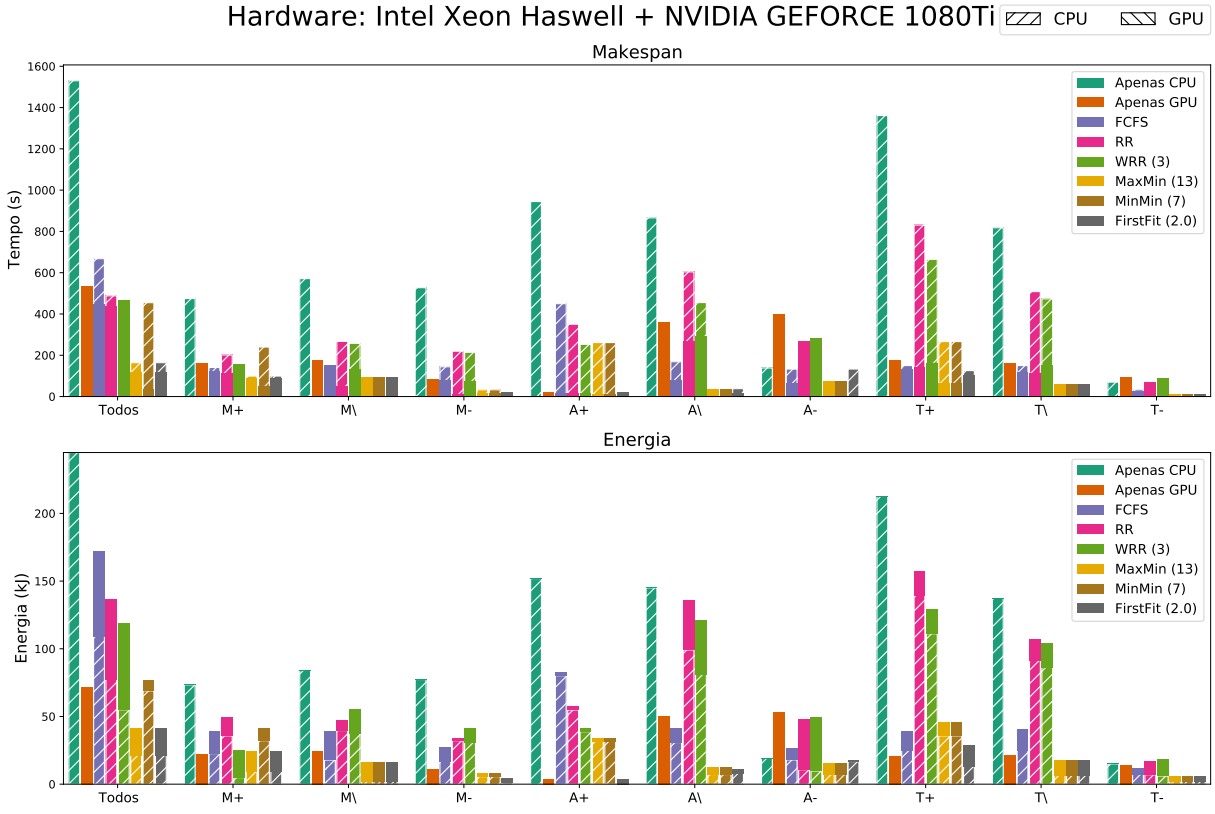
Gráfico 5.2 – Tempo de execução e energia do sistema para a configuração Intel Xeon Haswell + NVIDIA Tesla K80.



Fonte: Autor.

Os Gráficos 5.2 e 5.3 apresentam informações para as configurações Intel Xeon Haswell + NVIDIA Tesla K80 e Intel Xeon Haswell + NVIDIA GEFORCE 1080Ti. Considerando o tempo de execução e energia utilizado para a execução do cenário com todos os

Gráfico 5.3 – Tempo de execução e energia do sistema para a configuração Intel Xeon Haswell + NVIDIA GEFORCE 1080Ti.



Fonte: Autor.

núcleos em relação a NVIDIA Tesla K20m, as GPUs são $1,4$ e $2,0$ vezes mais rápidas e com uma redução de 12% e 22% de energia, respectivamente.

Para a configuração Intel Xeon Haswell + NVIDIA Tesla K80, a aceleração média dos algoritmos diminui, devido à redução de tempo de execução quando os cenários são executados exclusivamente na GPU. Para os algoritmos FCFS, RR e WRR, a execução conjunta continua causando desaceleração na maioria dos cenários, enquanto Max Min, Min Min, e FF obtém aceleração de $2,68$, $2,63$ e $3,04$ vezes na média e reduções de 19%, 17% e 33% de energia. FF continua sendo o único algoritmo capaz de competir com a execução em um único dispositivo em todos os cenários.

O aumento do número de CUDA cores e frequência de relógio na NVIDIA GEFORCE 1080 Ti com a mesma potência máxima de 250 W aumenta a diferença de capacidade computacional entre a CPU e a GPU nesta configuração, favorecendo a execução em GPU. Ainda assim, o benefício da execução conjunta dos lotes é evidente, sendo que Max Min, Min Min e FF têm aceleração média de $3,80$, $3,50$ e $3,80$, respectivamente. FF obtém até $15,13$ vezes de aceleração quando comparado a execução exclusiva na CPU. Porém, para o lote A+, os algoritmos Max Min e Min Min tem atrasos ainda maiores, devido ao fato da execução ainda mais curta na GPU em comparação a CPU, aumentando o tempo de execução em 13 vezes. Da mesma maneira, os problemas de escalonamento de FCFS

e WRR são exacerbados nessa configuração, levando a uma média de desaceleração de até $0,56$ vezes.

Assim, First Fit é o algoritmo ideal para todas as configurações de hardware e cenários explorados, devido a sua consideração única da métrica de aceleração da GPU em relação à CPU dos núcleos, essencial para encontrar-se o melhor escalonamento do lote.

6 CONCLUSÃO

Este trabalho propôs um *framework* para o escalonamento e seleção de dispositivo OpenCL de maneira transparente em um nó de nuvem CPU+GPU multi-inquilino, visando o melhor uso dos recursos do sistema e diminuição do makespan e energia. Cinco algoritmos de escalonamento são explorados: FCFS, RR, WRR, Max Min, Min Min e FF.

A execução do conjunto de *benchmarks* PolyBench nos diferentes dispositivos revela que, apesar de sua programação visada a determinada GPU, é possível se obter desempenho igual, ou melhor, fazendo uso da CPU como dispositivo OpenCL para algumas das aplicações. Isso evidencia a importância da capacidade do *framework* na seleção de dispositivo em tempo de execução, assim como o conhecimento geral dos recursos computacionais do sistema, sem alterações no código-fonte original da aplicação.

Dentre os algoritmos avaliados, First Fit se destaca com aceleração média de $3,04$ até $3,80$ vezes nas configurações de hardware utilizadas, com reduções de energia de 33% em média. O uso exclusivo do parâmetro de aceleração da GPU em relação à CPU é a característica que estabelece o desempenho acima dos outros algoritmos.

Como trabalhos futuros, podem ser explorados a estimação de aceleração para núcleos desconhecidos ou com cargas computacionais variantes para o funcionamento de algoritmos dependente de informações de aceleração. Além disso, a exploração de implementação de um modo de operação não baseado em lotes, fazendo análise de requisições em tempo real e execução *online* para a refinação dos dados em comparação ao funcionamento de uma nuvem real.

REFERÊNCIAS BIBLIOGRÁFICAS

BOOB, S.; GONZáLEZ-VéLEZ, H.; POPESCU, A. M. Automated instantiation of heterogeneous fast flow CPU/GPU parallel pattern applications in clouds. In: **2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing**. [S.l.: s.n.], 2014. p. 162–169. ISSN: 2377-5750.

CAMPA, S. et al. Parallel patterns for heterogeneous CPU/GPU architectures: Structured parallelism from cluster to cloud. v. 37, p. 354–366, 2014. ISSN 0167739X. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S0167739X14000041>.

GRAUER-GRAY, S. et al. Auto-tuning a high-level language targeted to GPU codes. In: **2012 Innovative Parallel Computing (InPar)**. IEEE, 2012. p. 1–10. ISBN 978-1-4673-2633-9 978-1-4673-2632-2 978-1-4673-2631-5. Disponível em: <http://ieeexplore.ieee.org/document/6339595/>.

GREWE, D.; O'BOYLE, M. F. P. A static task partitioning approach for heterogeneous systems using OpenCL. In: KNOOP, J. (Ed.). **Compiler Construction**. Springer Berlin Heidelberg, 2011. v. 6601, p. 286–305. ISBN 978-3-642-19860-1 978-3-642-19861-8. Series Title: Lecture Notes in Computer Science. Disponível em: <http://link.springer.com/10.1007/978-3-642-19861-8_16>.

GUZMáN, M. A. D. et al. Cooperative CPU, GPU, and FPGA heterogeneous execution with EngineCL. v. 75, n. 3, p. 1732–1746, 2019. ISSN 0920-8542, 1573-0484. Disponível em: <http://link.springer.com/10.1007/s11227-019-02768-y>.

HERBET, B. C. **General Principles of Classical Thermodynamics**. New Jersey, CA, USA: Wiley & Sons, 1985. ISBN 978-0471862567.

Khronos® OpenCL Working Group. **The OpenCL™ C Specification**. [S.l.], 2021. 252 p.

KIM, J. et al. Achieving a single compute device image in OpenCL for multiple GPUs. In: **PPoPP '11: Proceedings of the 16th ACM symposium on Principles and practice of parallel programming**. [S.l.: s.n.], 2011. p. 11.

MASSARI, G. et al. Extending a run-time resource management framework to support OpenCL and heterogeneous systems. In: **Proceedings of Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms - PARMA-DITAM '14**. ACM Press, 2014. p. 21–26. ISBN 978-1-4503-2607-0. Disponível em: <http://dl.acm.org/citation.cfm?doid=2556863.2556868>.

RAVI, V. T. et al. Scheduling concurrent applications on a cluster of CPU–GPU nodes. v. 29, n. 8, p. 2262–2271, 2013. ISSN 0167739X. Disponível em: <https://linkinghub.elsevier.com/retrieve/pii/S0167739X13001167>.

RIEBLER, H. et al. Transparent acceleration for heterogeneous platforms with compilation to OpenCL. v. 16, n. 2, p. 1–26, 2019. ISSN 1544-3566, 1544-3973. Disponível em: <https://dl.acm.org/doi/10.1145/3319423>.

STALLMAN, R. M.; DEVELOPERCOMMUNITY, G. **Using The Gnu Compiler Collection: A Gnu Manual For Gcc Version 4.3.3**. Scotts Valley, CA: CreateSpace, 2009.

SUN, E. et al. Enabling task-level scheduling on heterogeneous platforms. In: **Proceedings of the 5th Annual Workshop on General Purpose Processing with Graphics Proces-**

sing Units - GPGPU-5. ACM Press, 2012. p. 84–93. ISBN 978-1-4503-1233-2. Disponível em: <http://dl.acm.org/citation.cfm?doid=2159430.2159440>.

TAYLOR, B.; MARCO, V. S.; WANG, Z. Adaptive optimization for OpenCL programs on embedded heterogeneous systems. In: **Proceedings of the 18th ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems**. ACM, 2017. p. 11–20. ISBN 978-1-4503-5030-3. Disponível em: <https://dl.acm.org/doi/10.1145/3078633.3081040>.

WEN, Y.; WANG, Z.; O'BOYLE, M. F. P. Smart multi-task scheduling for OpenCL programs on CPU/GPU heterogeneous platforms. In: **2014 21st International Conference on High Performance Computing (HiPC)**. IEEE, 2014. p. 1–10. ISBN 978-1-4799-5976-1. Disponível em: <http://ieeexplore.ieee.org/document/7116910/>.