Exploração do Espaço de Projetos de Sistemas GP-GPU ciente de Dark Silicon

Rhayssa de Almeida Sonohata

Exploração do Espaço de Projetos de Sistemas GP-GPU ciente de Dark Silicon[1]

Rhayssa de Almeida Sonohata

Orientadora: *Profª Drª Liana D. Duenha*
Co-orientador: *Prof. Dr. Eraldo Luiz Rezende Fernandes*

Dissertação em Ciência da Computação apresentada ao Programa de Pós-Graduação da Faculdade de Computação (FACOM) da Universidade Federal de Mato Grosso do Sul (UFMS), como parte dos requisitos necessários à obtenção para do título de Mestre em Ciência da Computação.

UFMS - Campo Grande (MS)
Junho/2022

[1]Trabalho Realizado com Auxílio da CAPES Proc. No: 51001012

Aos meus pais,
Zenir e Roberto.

À minha vó,
Maria Aparecida,

Agradecimentos

Inicialmente, gostaria de agradecer a Deus, por ter sido meu amparo nessa jornada que foi o mestrado e ter me ajudado a passar por todo esse período de pandemia, sem ter me deixado desistir, por ter colocado excelentes pessoas em meu caminho e por ter sido minha companhia nos momentos mais difíceis.

Agradeço aos meus pais, Roberto Sonohata e Zenir de A. P. Sonohata, à minha irmã Laryssa e a minha vó, Maria Aparecida, por terem acreditado em mim, mais do que eu mesma acreditei, por terem feito o possível e o impossível para que eu pudesse alcançar os meus objetivos, por ter cuidado de mim com infinito amor e zelo.

Agradeço à minha orientadora, Liana Duenha, e ao meu coorientador, Eraldo Luiz, por não terem medido esforços para me ajudar na conclusão deste trabalho, por terem me ajudado a conseguir os computadores para simulação, por terem me orientado e ensinado, com muita paciência, mais do que estava no escopo de suas disciplinas e trabalhos durante esse período, pelos conselhos que levarei pela vida, e por serem exemplos de dedicação e trabalho a serem seguidos.

Agradeço ao professor Ricardo, por ter se feito disponível em todos os momentos necessários, sanando as dúvidas e pelas conversas no laboratório.

Agradeço ao meu namorado Wilton, que tem sido uma de minhas maiores redes de suporte, por ter me ajudado a usar o LaTeX, e por ter sido minha companhia para todas as horas.

Agradeço à minha amiga Emilly, pelas horas de conversa a distância, agradeço também aos meus amigos de laboratório, Casio e Gregório, que me acompanharam durante bons momentos da graduação e me ajudaram a realizar alguns experimentos.

Abstract

The high potential for parallelism and bandwidth offered by GPUs and the popularization of the CUDA and OpenCL programming languages made GPUs useful in different applications from those for which they were originally designed to. Then, these facts consolidate the concept of *GP-GPU* or *Graphics Processing Units for General-Purpose computing*. With the use of systems that join CPUs and GPUs for collaborative processing, tools were developed to explore the performance and consumption of the various architectural parameters in heterogeneous computing designs. However, these tools are scarce, limited, computationally expensive, and need architectural parameters that are difficult to obtain.This work proposes the design of performance prediction models for GP-GPU systems from Machine Learning techniques. We evaluate the predictors in a design space exploration tool. MultiExplorer meets pre-defined goals such as performance maximization and dark-silicon area minimization, subject to constrains as circuit area and energy consumption bounds. Depending on the design space, this tool evaluates hundreds of thousands of architectural alternatives and, therefore, performance predictors with low delay and high accuracy are essential.

Resumo

O alto potencial de paralelismo e de largura de banda oferecidos pelas GPUs, aliados à popularização das linguagens de programação CUDA e OpenCL, fizeram com que as unidades de processamento gráfico fossem utilizadas em aplicações distintas daquelas para as quais foram originalmente criadas, consolidando, assim, o conceito de *GP-GPU* ou *Unidades de Processamento Gráfico para computação de propósito geral*. A partir do crescente uso de sistemas que unem CPUs e GPUs para processamento de forma colaborativa, foram desenvolvidas ferramentas para explorar o desempenho e o consumo dos diversos parâmetros arquiteturais dos projetos de computação heterogênea. Entretanto, essas ferramentas são escassas, limitadas, computacionalmente custosas e precisam de parâmetros arquiteturais de difícil obtenção. Isso posto, este trabalho propõe o desenvolvimento e avaliação de modelos de predição de desempenho de sistemas heterogêneos GP-GPU usando técnicas de aprendizado de máquina, com objetivo de alcançar alta acurácia e substituir o custoso processo de simulação. Tais preditores foram validados a partir da integração a uma ferramenta de exploração de espaço de projeto ciente de *dark-silicon*, denominada *MultiExplorer*, que realiza a avaliação de alternativas arquiteturais para um projeto-base, buscando alcançar objetivos pré-estabelecidos como maximização de desempenho e minimização de área em *dark-silicon*, obedecendo restrições de área e consumo energético. Dependendo do espaço de projeto, tal ferramenta avalia centenas de milhares de alternativas arquiteturais e, portanto, os preditores de desempenho com baixo custo computacional e acurácia são essenciais.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

AD Árvore de Decisão

AM Aprendizado de Máquina

IA Inteligência Artificial

GPU *Graphics Processing Unit*

SVR *Support Vector Regression*

SVM Support Vector Machines

RBF *Radial Basis Function*

SIMD Single Instruction Multiple Data

SIMT Single Instruction Multiple Thread

CPU *Central Processing Unit*

SM *Streaming Multiprocessors*

AM *Aprendizado de máquina*

SP *Streaming Processors*

UC *Unidades de Computação*

SFU *Special Function Units*

DRAM *Dynamic Random-Access Memory*

SASS *Shader Assembly*

GP-GPU *General Purpose Graphics Processing Unit*

DS-DSE *Domain-specific Design Space Exploration*

DSE *Design Space Exploration*

PTX *Parallel Thread Execution*

ISA *Instruction Set Architecture*

PISA *Portable Instruction Set Architecture*

CUDA SDK *Software Development Kit for GPU Computing*

API *Application Programming Interface*

asyncAPI *Simple Reference*

kNN *K-nearest Neighbors*

Backprop *Back Propagation*

NW *Needleman-Wunsch*

dwt2d *Discrete Wavelet Transform*

BFS *Breadth-First Search*

IPC *Instructions per Cycle*

TSVM *Máquinas de Vetores de Suporte Transdutivas*

PCA *Análise de Componentes Principais*

Overfitting *Sobre-ajuste*

MLP *Perceptron Multicamada*

CTA *Cooperative Thread Array*

DS *Dark Silicon*

MIPS *Milhões de Instruções por Segundo*

Runtime *Tempo de execução aproximado*

MAPE *Mean Average Percentage Error*

CAPÍTULO

1

Introdução

Guiados pela Lei de Moore [6] e escala de Dennard [7], o aumento exponencial da quantidade de transistores nos chips e o consequente ganho de desempenho têm encontrado fortes limitações energéticas, tais como o aumento da corrente de fuga, potência dissipada e densidade de potência. Com isso, parte do chip deve ser desligado ou deve funcionar com frequência abaixo do limite máximo, o que é chamada de área em *dark silicon*. Mapear essa parte do chip e propor soluções arquiteturais que possam ocupá-la adequadamente, obedecendo a limitações de densidade de potência e área, tem sido uma solução alternativa ao problema [8].

A partir das limitações energéticas, escalar a frequência de um único *core* deixou de alcançar os benefícios em desempenho que, anteriormente, eram alcançáveis. Além disso, com os avanços nos processos tecnológicos, a mesma escalabilidade tornou-se inviável. Assim, os fabricantes buscaram como alternativa o aumento do paralelismo tanto em *hardware* quanto em *software*. O paralelismo em nível de hardware introduziu a era dos sistemas *multicore* e o uso de aceleradores como as GPUs [9].

As GPUs são unidades de processamento gráfico paralelas desenvolvidas para acelerar aplicações gráficas 3D. Elas têm como principais características a alta taxa de transferência, alta largura de banda e grande capacidade de realizar cálculos em paralelo [10]. Estes dispositivos tornaram-se programáveis à medida em que foram evoluindo. Esse processo foi facilitado com a introdução de linguagens de programação baseadas e C e C++, como o CUDA [11] e OpenCL [12], fornecendo o ferramental necessário para exploração do paralelismo em GPUs. Consequentemente, as aplicações de propósito geral passaram a ter ganho de desempenho quando executadas em GPUs, o que

resultou no termo *GP-GPU* ou *Unidades de Processamento Gráfico para (computação) de propósito geral*.

A vasta disseminação das aplicações GP-GPU leva a necessidade de ferramentas para exploração das novas arquiteturas. Entretanto, pode-se afirmar diversos desafios relacionados a isso:

- não existem muitas ferramentas para a simulação de desempenho de GPUs;
- os simuladores atualmente disponíveis precisam de várias informações arquiteturais, as quais não são facilmente disponibilizadas pelos fabricantes;
- o tempo de resposta das simulações é alto, pois elas demandam muitos recursos computacionais;
- alguns dos simuladores também dependem de GPUs reais para realizar a simulação/emulação e gerar estimativas de desempenho;
- praticamente não há ferramentas que permitem modelagem e simulação de sistemas heterogêneos.

Diante de tais limitações, propõe-se o desenvolvimento de modelos de aprendizado de máquina capazes de predizer o desempenho de GPUs, com base em parâmetros arquiteturais do sistema. Esses preditores devem ter acurácia mínima garantida e baixo custo computacional, atributos desejáveis para o uso em exploração do espaço de projetos de hardware heterogêneo.

Como contribuição adicional, esse trabalho integra os melhores modelos encontrados para predição de desempenho de sistemas baseados em GPUs a uma metodologia de exploração do espaço de projetos ciente de dark silicon (DS-DSE) proposta por Santos et al. [13]. Tal método escolhe, dentre uma grande quantidade soluções alternativas, qual é a mais indicada para um projeto em particular. Esse processo é dependente da avaliação das soluções alternativas, de acordo com restrições e objetivos predefinidos. Pelo alto custo computacional decorrente desse processo, DS-DSE torna-se parte crítica do ciclo de um projeto e gera a necessidade de encontrar alternativas que explorem o *trade-off* entre precisão e custo computacional, melhorando sua aplicabilidade. Desta forma, considera-se que a predição de desempenho de sistemas heterogêneos contendo GPUs, com acurácia e baixo custo computacional, aplicada à exploração do espaço de projetos ciente de *dark silicon* é um processo desafiador e uma contribuição significativa.

Sumarizando, este trabalho apresenta como principais contribuições:

- Adaptação da metodologia de estimativa de dark silicon proposta por Santos et al. [8] para GPUs.
- Desenvolvimento e avaliação de um preditor de desempenho para arquiteturas heterogêneas, contendo modelos de GPUs distintas, usando modelos de aprendizagem de máquina e redes neurais.
- Aplicação desse preditor em uma ferramenta de exploração do espaço de projeto, denominada MultiExplorer. [13].

Esta dissertação está organizada da seguinte forma: o Capítulo 2 apresenta os conceitos, simuladores e benchmarks para sistemas GPUs e descreve o fluxo da ferramenta MultiExplorer. O Capítulo 3 descreve algoritmos e métricas de avaliação de modelos de aprendizado de máquina que serão utilizados no desenvolvimento deste trabalho. O Capítulo 4 mostra os trabalhos publicados anteriormente, que possuem intersecção com nossa proposta. O Capítulo 5 descreve a metodologia adotada para o desenvolvimento e avaliação dos preditores de desempenho de GPUs. O Capítulo 6 descreve a integração do preditor mais adequado ao MultiExplorer para realização de exploração do espaço de projetos de sistemas GP-GPU. O Capítulo 7 apresenta as considerações finais.

CAPÍTULO

2

Referencial Teórico sobre sistemas GP-GPU

O objetivo deste Capítulo é introduzir os conceitos, simuladores e textitbenchmarks a respeito de GPUs. Os conceitos iniciais e o modelo de paralelismo adotados para esses dispositivos são introduzidos na Seção 2.1; os principais simuladores de sistemas baseados em GPUs e seus *benchmarks* são apresentados na Seção 2.2; os conceitos e a metodologia de estimativa de *Dark-Silicon* são introduzidos na Seção 2.3. Por fim, a infraestrutura da ferramenta MultiExplorer, que será estendida com infraestrutura para exploração de projetos com GPUS, é descrita na Seção 2.4. A Seção 2.5 conclui o capítulo.

2.1 As Unidades de Processamento Gráfico (GPUs)

A classificação SIMD (do inglês, *single instruction multiple data*) foi proposta na Taxonomia de Flynn [14] em 1966 e define um modelo de computação em que uma mesma instrução da aplicação é executada sobre um diverso conjunto de dados em paralelo. Esse modelo introduziu o *paralelismo em nível de dados* que tem o potencial de aumentar o desempenho de aplicações adequadas ao SIMD e de ser energeticamente mais eficiente. Desde então, diversos modelos arquiteturais de computação SIMD foram propostos, os quais são descritos a seguir:

- **Arquiteturas vetoriais:** nestas arquiteturas, uma única instrução opera sobre vetores de dados, que resulta em uma grande quantidade de operações sobre dados independentes. Esse tipo de processamento aproveita a

largura de banda da memória para mitigar a alta latência das operações de transferências de dados em lotes.

- **GPUs**: são unidades de processamento gráfico massivamente paralelas que surgiram prioritariamente para melhorar o desempenho de aplicações gráficas, mas que hoje são utilizadas também para computação de propósito geral. As GPUs não têm um ancestral comum às CPUs na genealogia das arquiteturas de computadores, pois elas foram criadas com propósitos específicos de serem aceleradores gráficos. O modelo de programação utilizando GPUs é heterogêneo pois considera elementos de processamento de distintas características. A CPU é considerada o *host* e a GPU é considerada o *device*. Para que um programa seja executado neste ambiente heterogêneo, é necessário definir quais trechos do programa serão executados como *threads* na GPU. Desta forma, é comum definir o modelo de programação para GPU como SIMT ou "Single Instruction Multiple Thread", que é uma especialização do modelo SIMD previsto por Flynn [14] em 1966.

Na última década, as GPUs se tornaram mais acessíveis para usuários de computadores pessoais e, além disso, cresceu também o uso de GPUs para processamento de propósito geral, aproveitando o alto poder computacional destes componentes e expondo, assim, o potencial para produção de sistemas heterogêneos CPU+GPUs para processamento de propósito geral. Estes aceleradores tornaram-se, então, uma alternativa para melhorar o desempenho de aplicações com intensiva demanda computacional, mantendo limites de consumo energético.

Muitas aplicações científicas, de aprendizado de máquina, mineração de dados, textos e moedas virtuais, e de computação de alto desempenho fazem uso do potencial paralelismo das GPUs para tirar proveito da abundância de recursos computacionais de processamento destes sistemas. Desta forma, dizemos que as GPUs passaram a ser utilizadas também para computação de propósito geral.

2.1.1 Desempenho das GPUs

A *latência* de uma unidade de processamento é grandeza que nos permite avaliar a variação de tempo entre o momento em que uma operação inicia e o momento em que o usuário é capaz de perceber o resultado. A *vazão* ou *throughput* pode ser medida pela quantidade de trabalho realizado por unidade de tempo. As CPUs são unidades de processamento projetadas para um modelo de paralelismo em nível de tarefas que tem como característica a baixa latência e baixa vazão [15]. Em contrapartida, as GPUs tem alta latência, mas

compensam essa indesejada característica realizando processamento com altíssima vazão.

As GPUs são projetadas para o modelo de computação paralela em nível de dados; conseguem realizar muitos cálculos em paralelo por terem muito mais unidades funcionais e com isso conseguem realizar mais operações por unidade de tempo quando comparados a núcleos de processamento convencionais. Além disso, as GPUs são projetadas para permitir a execução de dezenas de milhares de *threads* independentes umas das outras, minimizando a sincronização e *overhead* para escalonamento, já que as *threads* são gerenciadas e escalonadas pelo hardware. A Figura 2.1 ilustra a comparação de desempenho em GBytes/segundo entre modelos de GPUs GeForce e Tesla, e arquiteturas de núcleos de CPUs.

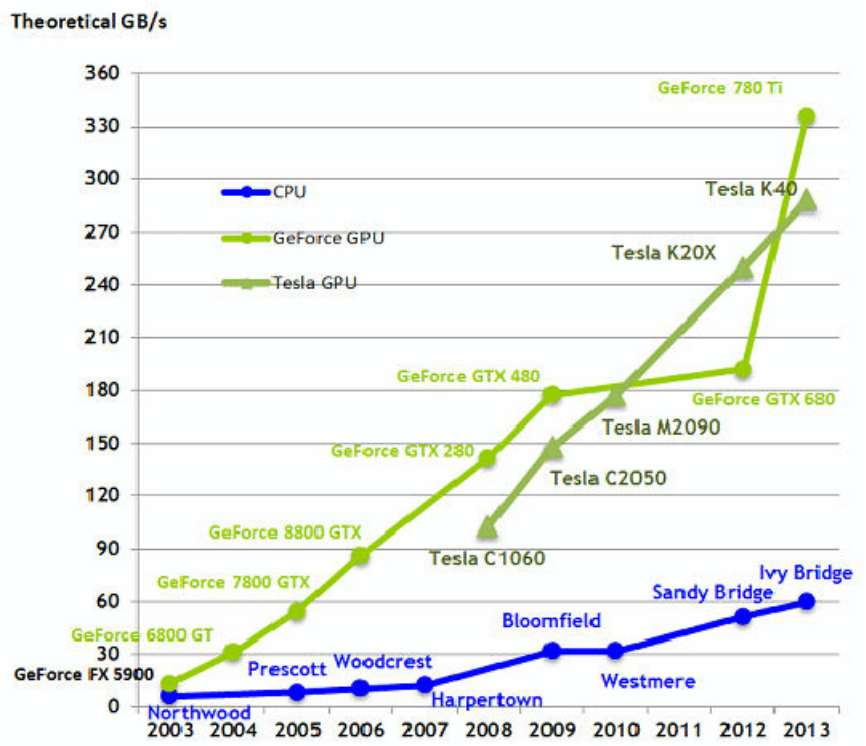


Figura 2.1: Comparação de desempenho em GB/s (gigabytes por segundo) entre modelos de GPUs e CPUs. Fonte: [1].

2.1.2 Arquitetura básica de uma GPU

A regularidade arquitetural que encontramos nos processadores (CPUs) convencionais, mesmo que de fabricantes distintos, embora desejada, não pode ser exigida para um projeto de GPU. Como o domínio para o qual os projetos de GPU foram definidos é muito específico, os fabricantes buscam especializar ao máximo a placa para alcançar desempenho para suas aplicações-alvo.

No ano de 2006 foi lançada no mercado a arquitetura Tesla implementada

na placa GeForce 8800 da NVIDIA, com 128 *cores* contendo, cada um, centenas de *threads*. Com isso, surgiu o primeiro pipeline unificado em que o processamento era realizado dentro dos *cores* representando um processamento escalar, dispensando o processamento vetorial anteriormente utilizado. Essa mudança fez com que a GPU tivesse compatibilidade com linguagens como C e com modelos de programação paralela [16].

Posteriormente, foi lançada a arquitetura Fermi com 16 multiprocessadores de *streaming* (ou do inglês, *Streaming Multiprocessors* (SM)) que foram denominados como UCs no texto, cada um com 32 CUDA cores ou processadores de *streaming* (ou do inglês, *Streaming processors* (SP)). Cada CUDA core possui uma Unidade Lógico-Aritmética e uma Unidade de Ponto-Flutuante de 32 bits. Uma UC definida na arquitetura Fermi mostrada Figura 2.2 inclui 4 unidades de ponto-flutuante (SFU) que realiza cálculos como raiz quadrada, seno e cosseno, além de ter 16 unidades para tratar instruções *LD/ST*, que buscam e armazenam dados na cache ou DRAM.

Os elementos computacionais dos algoritmos paralelos para execução em GPU são conhecidos como *kernels*. Uma vez compilados, consistem em múltiplas *threads* que executam o mesmo código em paralelo. Várias *threads* são agrupadas em blocos que são executados em uma UC, sequencialmente ou em paralelo, fazendo uso de uma memória compartilhada. Cada bloco é dividido em *warps* de 32 threads, o qual é a unidade básica de despacho para uma UC. Na Fermi, dois warps de diferentes blocos podem ser despachadas e executadas concorrentemente, melhorando a taxa de utilização e eficiência energética do hardware.

A Fermi possui dois escalonadores de *warps* e unidades de despacho, responsáveis por despachar e executar os *warps* de forma concorrente. A memória compartilhada e cache L1 de 64 KB permite a interação entre as *threads* de mesmo bloco e pode ser dividida de duas formas: 48 KB para memória compartilhada e 16 KB para cache L1 ou vice-versa [2].

2.2 Simuladores de sistemas baseados em GPUs

O processo de simulação de sistemas computacionais tem sido amplamente utilizado por projetistas e pesquisadores da área para realizar a avaliação de modelos com diversas combinações de parâmetros arquiteturais como variações de cache, frequência, quantidade de elementos de processamento, entre outros. Ao realizar a simulação, o usuário acumula conhecimento sobre o domínio do projeto e pode adotar um comportamento preditivo sobre a influência dos seus parâmetros no desempenho e consumo energético, evitando erros que podem acarretar muitas perdas para os fabricantes.

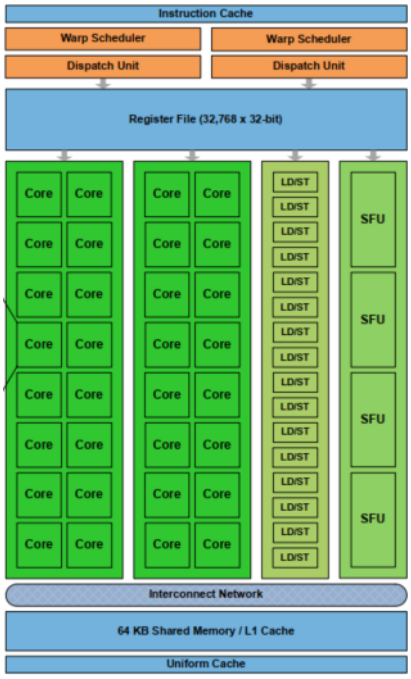


Figura 2.2: UC na arquitetura Fermi [2]

.

Como exemplo de simuladores de GPUs, podemos citar o framework Barra [17], que simula de modo paralelo programas CUDA baseados em arquitetura de GPU Tesla. Essa ferramenta é modular e tem como entrada executáveis de programas escritos em CUDA e com isso é capaz de avaliar parâmetros micro-arquiteturais como registradores, memória, unidades computacionais, mecanismos de interconexão, etc.

O Multi2Sim [18] é uma ferramenta que simula com precisão de ciclos as GPUs com base na arquitetura Kepler da NVIDIA. A ferramenta permite a modelagem do hardware das UCs, memória cache, estágios do pipeline, entre outros componentes. O Multi2Sim trabalha com uma linguagem da Nvidia de baixo nível denominada SASS (do inglês, *shader assembly*) e é potencialmente preparado para simular sistemas heterogêneos, já que sua versão original era voltada para sistemas *multicore*.

A ferramenta escolhida para a realização dos experimentos preliminares desse trabalho é o GPGPU-Sim [19, 20] desenvolvida para estimar desempenho de GPUs baseadas na arquitetura Fermi e modeladas com cores SIMT com precisão de ciclos [21].

Na primeira versão do simulador não havia suporte para instruções PTX CUDA, mas apenas para ISA PISA e a simulação era realizada por alternância (enquanto a GPU está executando, o *core* (host) espera e quando a GPU termina sua execução, o controle retorna ao *core*), até que aplicação termine [20]. Posteriormente, houve uma alteração no fluxo de execução da ferramenta para o suporte de programas escritos em CUDA e OpenCL. Subsequentemente, foi

realizada uma acoplação do GPGPU-sim com o GPUWattch [22], que viabilizou além de resultados de desempenho, a estimativa do consumo energético dos componentes arquiteturais.

2.3 Dark-silicon

A Lei de Moore [6, 23] afirma que a quantidade de transistores em um chip deveria duplicar a cada 18 meses. Juntamente com a escala Dennard [24], a Lei de Moore guiou o mercado tecnológico durante anos, resultando em uma miniaturização dos chips e consequente aumento de desempenho.

A diminuição da litografia dos chips encontrou, como barreira, o aumento da corrente de fuga, criando um problema conhecido na indústria como *dark-silicon*. A porção do chip que deve ser desligada ou funcionar com frequência aquém do esperado é denominada área de *dark-silicon* do projeto. Na tentativa de mitigar o problema, projetos com múltiplos processadores ou *multicore* foram adotados e já fazem parte do estado da arte há vários anos. Além disso, a indústria também busca mitigar o efeito *dark-silicon* por meio da integração de GPUs, FPGAs e componentes de processamento aproximados aos núcleos de processadores convencionais. Contudo, o problema de gerar *dark-silicon* nos projeto modernos ainda continua vigente.

No contexto deste trabalho, deseja-se poder estimar a porção de dark-silicon em tempo de projeto. Em outras palavras, dado um novo projeto contendo GPUs, ainda não implementado fisicamente, estimar qual a potencial área de dark-silicon do projeto e propor alternativas arquiteturais para reduzir ou mitigar o efeito dark-silicon.

Uma abordagem para estimativa de dark-silicon de sistemas com múltiplos processadores foi apresentada por Santos et al. [8] e baseia-se na densidade de potência dos componentes envolvidos. Neste trabalho, realizou-se a adequação desta metodologia para GPUs e sua integração à ferramenta MultiExplorer para exploração do espaço de projetos de sistemas GP-GPUs. A ferramenta MultiExplorer será apresentada a seguir e a metodologia de estimativa de dark-silicon, adequada para GPUs, será descrita no Capítulo 6, juntamente com os resultados experimentais.

2.4 MultiExplorer

O processo de exploração do espaço de projeto consiste em explorar um vasto conjunto de possibilidades de prototipação de um sistema antes da implementação, de otimização de projetos por comparação com outros modelos e de integração de forma a realizar a exploração ciente de restrições e objetivos

definidos pelo projetista [25].

O MultiExplorer é uma ferramenta de exploração do espaço de projetos apresentada em [13] e explorada em diversos outros trabalhos posteriores [26, 27, 28, 29]. Esta ferramenta recebe, como entrada, a descrição da arquitetura de uma plataforma multiprocessada que tem parte da sua área em dark-silicon. Em seguida, aplica algoritmos para explorar o espaço de projetos e fornecer, como saída, alternativas arquiteturais que mitiguem o dark-silicon. Em outras palavras, a ferramenta propõe as soluções arquiteturais mais eficientes livres de *dark-silicon*.

Na Figura 2.3 é possível visualizar o fluxo de execução da exploração do espaço de projetos do MultiExplorer. Na etapa de exploração de desempenho, o usuário fornece a descrição da plataforma, seleciona o simulador funcional, juntamente com a aplicação e/ou um conjunto de aplicações (benchmark) que deseja utilizar. Dentre os simuladores de desempenho, pode-se escolher o Sniper [30], para aplicações e sistemas paralelos em ambiente multithreaded e multicore ou o GPGPU-Sim [19, 20], para sistemas GP-GPU. Para sistemas multicore, são previstas aplicações dos benchmarks PARSEC [31] e SPLASH-2 [32]. Para os sistemas GP-GPU são utilizadas aplicações dos benchmaks NVIDIA CUDA SDK [33], disponível no GPGPU-Sim e Rodinia [34].

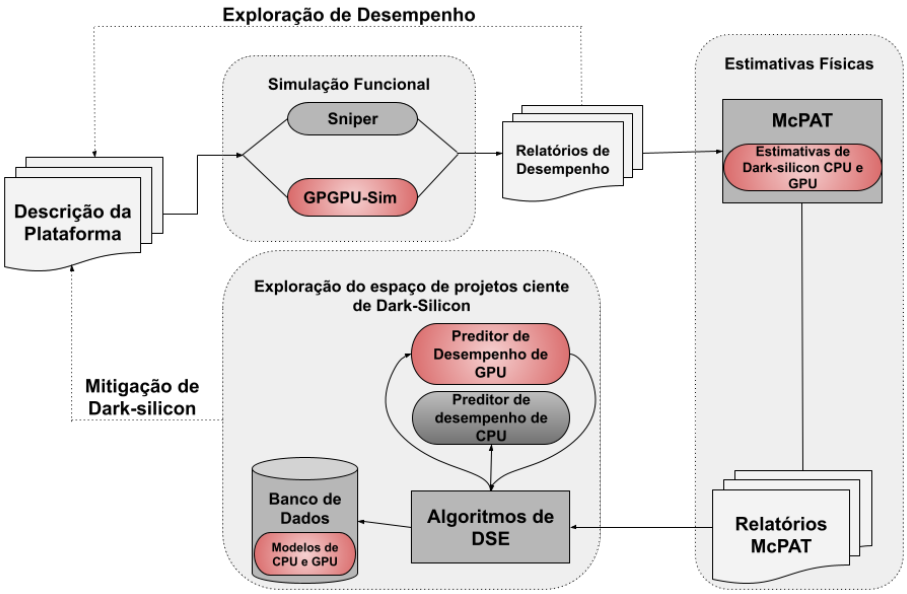


Figura 2.3: Fluxo do MultiExplorer. Elaborada pelo autor, com base em [3].

Após simulação funcional, a ferramenta reporta quantidade de ciclos, tempo de execução, IPC, desempenho das caches e outras estatísticas de desempenho. Logo após o resultado do relatório de desempenho, essa saída é adaptada para servir como entrada do processo de exploração física, que é efetuado pela ferramenta McPAT [35]. Esta ferramenta estima parâmetros como potência e área dos componentes do hardware. Nessa etapa é estimada área do chip em

dark-silicon, através de parâmetros como a densidade de potência, conforme apresentado na Seção 2.3

Posteriormente ao resultado da Exploração física ocorre o processo de exploração do espaço de projeto, que tem como entrada os resultados do relatório de desempenho, o relatório do McPAT, e o banco de dados de cores. A exploração pode ser realizada de maneira exata por um algoritmo de força-bruta ou por um algoritmo genético denominado NSGA-II, que gera novos indivíduos e descarta aqueles que estão fora dos limitantes impostos pelo usuário. Em ambas as abordagens, é necessário estimar o desempenho de cada solução alternativa heterogênea.

Na versão original da ferramenta, o desempenho do sistema heterogêneo era calculado como a soma dos desempenhos dos núcleos da plataforma. Este método é bastante otimista, pois dificilmente a plataforma multicore alcançaria tal desempenho. Com o objetivo de deixar mais acurado esse processo, foi desenvolvido e acoplado à ferramenta, um um preditor de desempenho de sistemas heterogêneos [3], com base em técnicas de aprendizado de máquina. Tal preditor tornou as estimativas de desempenho mais justas, independentes de simulação e com baixo custo computacional. O algoritmo de aprendizado de máquina escolhido foi o SVR com kernel RBF, que obteve acurácia de 97%, erro médio de 20,34%.

Os componentes destacados em cores na Figura 2.3 mostram todas as extensões que foram feitas no contexto deste trabalho, para dar suporte à exploração de sistemas GP-GPU. Além desta frente de trabalho, há outras pesquisas em andamento para extensão do módulo de DS-DSE com componentes de computação aproximada, com objetivo de melhorar eficiência energética, enquanto mantém sob controle os erros de aproximação. Ademais, também há uma frente de trabalho para utilização de técnicas de exploração de espaço de projeto do MultiExplorer para selecionar soluções arquiteturais adequadas para uma demanda específica usando recursos em nuvem.

2.5 Considerações Finais

Nesse capítulo, foram descritos os conceitos necessários sobre GP-GPU. Dentre os tópicos abordados, destacam-se as definições das arquiteturas de GPUs, métricas de desempenho, ferramentas de simulação, *benchmarks* e a ferramenta MultiExplorer para exploração de espaço de projetos ciente de dark-silicon.

CAPÍTULO

3

Referencial Teórico sobre Aprendizado de Máquina

Este capítulo aborda conceitos iniciais, modelos de aprendizado de máquina, técnicas e métricas para avaliação, os quais foram utilizados para implementação e avaliação dos modelos de predição de desempenho de GPUs.

3.1 Conceitos iniciais e modelos de aprendizado de máquina

Aprendizado de máquina (AM) é uma subárea da inteligência artificial que emprega técnicas estatísticas e encontra padrões nos dados avaliados com base em um conjunto de entrada visando a solução de problemas não-determinísticos. A qualidade da solução é dependente de características dos dados de entrada, como tamanho e complexidade da amostra [4, 36].

Esse processo de encontrar padrões faz uso da generalização que pode ser descrita como: encontrar uma função consistente para os dados de forma que essa mesma função possa ser aplicada a dados que ainda não foram vistos. Essa generalização é crucial para o aprendizado, que pode ser separado em alguns grupos como aprendizado supervisionado, aprendizado não supervisionado, aprendizado semissupervisionado e aprendizado por reforço, que serão brevemente definidos a seguir.

- ***Aprendizado supervisionado:*** É realizado com base em um conjunto de dados totalmente rotulados, tornando necessária uma forte intervenção humana na preparação do conjunto de treino. Destacam-se os modelos

kNN (k-vizinhos mais próximos ou *k-Nearest Neighbors*), Máquinas de Vetores de Suporte (SVM) e suas adaptações para problemas de regressão (SVR) e *Naïve Bayes*[37].

- ***Aprendizado Não-Supervisionado:*** Os algoritmos de aprendizado de máquina não-supervisionado utilizam um conjunto de dados sem nenhum tipo de rotulação e têm como objetivo descobrir semelhanças entre os objetos analisados, agrupando-os de acordo com suas similaridades. Modelos de aprendizado não-supervisionado são comumente divididos em algoritmos de Agrupamento (*Clustering*), Associação e Sumarização, dentre os quais destacam-se os modelos de agrupamento como *K-Means* e Agrupamentos Hierárquicos [38, 39].

- ***Aprendizado Semissupervisionado:*** Nos modelos de aprendizado de máquina semissupervisionado uma parte de dados são rotulados (supervisão humana) e o restante dos dados de treinamento são não-rotulados. A rotulação parcial dos dados cria um espaço entre esses dois modelos anteriormente descritos, no qual os modelos semissupervisionados atuam. Pode-se citar como exemplos de algoritmos semissupervisionados: Máquinas de Vetores de Suporte Transdutivas (TSVM), Algoritmos baseados em grafos e Algoritmos *Multiview* [40].

- ***Aprendizado por reforço:*** O objetivo dos modelos de aprendizado por esforço é aprender a partir de uma série de bonificações ou punições que auxiliem no aprendizado futuro ([39]). Este tipo de aprendizado é muito utilizado em jogos e na área de Robótica.

Neste projeto serão empregados modelos de aprendizado de máquina supervisionados.

O processo de aprendizado supervisionado passa por várias etapas, e antes de descrevê-las, é importante fazer uma breve descrição sobre os termos que virão a ser utilizados no decorrer desse trabalho [4].

- ***Conjunto de Dados:*** $D = \{(x, y)\}$, onde $x = (x_1, x_2, \ldots, x_m)$ é um vetor com os m atributos de entrada do exemplo (variáveis independentes), $x_j \in \mathbb{R}$ para $j = 1, 2, \ldots, m$ é um atributo de entrada, e $y \in \mathbb{R}$ é o atributo de saída (variável dependente).

- ***Conjunto de Treinamento:*** Pode ser descrito como uma amostra dentro do conjunto de dados que é separada para treinar o modelo.

- ***Conjunto de Validação:*** É uma amostra dentro do conjunto de dados que é separada para aprimorar o processo de escolhas de parâmetros automatizados do modelo.

- ***Conjunto de Teste:*** Amostra dentro do conjunto de dados que será utilizada na etapa de validação e não pode ser utilizada no processo de treinamento do modelo.

Para fazer com que um modelo aprenda de forma supervisionada, o conjunto de dados $D = \{(x, y)\}$, que é composto por um vetor de atributos de entrada x, e seus atributos de saída ou classificações y, é dividido entre conjuntos de treino, validação e teste. Em seguida, é criado um modelo com o algoritmo escolhido e ele é treinado com o conjunto de treino; em seguida, o modelo é aprimorado através do conjunto de validação e posteriormente é testado para verificar a acurácia do mesmo [36].

Atualmente, problemas como classificação de documentos, processamento de linguagem natural, processamento de fala, visão computacional e biologia computacional podem ser resolvidos através de técnicas de aprendizado de máquina. Além desses, as técnicas de aprendizado de máquina não são aplicadas também no desenvolvimento de sistemas especialistas que é aplicado no desenvolvimento desse trabalho visto que ele é eficaz para tomar decisões próximas a de um profissional da área. Ademais, podemos organizar os problemas de aprendizado de máquina em algumas classes [4]:

- ***Classificação:*** Consiste em atribuir um rótulo a cada item a ser classificado. O item pode ser um texto, uma imagem, dentre outros. Textos, por exemplo, podem ser classificados por tópico ou por gênero. [41]

- ***Redução de Dimensionalidade:*** Consiste em diminuir a dimensão dos dados sem alterar totalmente as suas características [4]. Para reduzir uma matriz são utilizadas técnicas como análise fatorial, análise de componentes principais (PCA), entre outras. Esses problemas são aplicados geralmente em aprendizado não supervisionado [42]

- ***Agrupamento:*** Também aplicado em aprendizado não supervisionado é um processo de agrupar objetos similares e separar objetos não similares. Também conhecido como *clustering*.

- ***Regressão:*** A regressão tem função preditiva, e é responsável por ajustar uma função a um conjunto de entrada buscando predizer a saída [5] e pode ser definida pela equação 3.1 em que $r(x)$ representa a função regressora.

$$\hat{y} = r(x) \tag{3.1}$$

  Após a definição da função de regressão, nos deparamos com um segundo problema de descobrir qual a melhor combinação de parâmetros

afim de fazer com que a função melhor se ajuste ao problema. Para isso, é comumente utilizado o critério de mínimos quadrados, que minimiza a soma do erro quadrático médio em um conjunto de dados anotados $D = \{(x, y)\}$:

$$e = \frac{\sum_{(x,y)\in D} (y_i - \hat{y}_i)^2}{|D|}. \tag{3.2}$$

Os modelos gerados pelo algoritmo podem ser mais especializados ou mais gerais [4]. Podemos ver na Figura 3.1 que o modelo da esquerda é mais complexo e mais especializado, já o da direita é mais geral. Entretanto, é muito comum achar que o primeiro é melhor, o que não necessariamente é verdade uma vez que o algoritmo pode ter decorado os dados de treinamento gerando um problema denominado *sobre-ajuste* (*overfitting*). Essa generalização é de extrema importância porque explica se o modelo que funciona bem em conjuntos de treinamento também tem eficácia em conjunto de teste. [36]

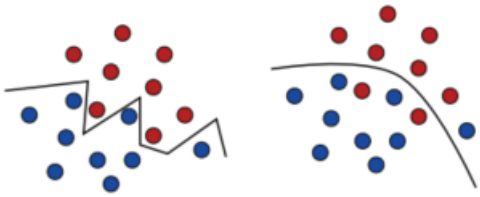


Figura 3.1: Modelos de separação sendo o primeiro mais complexo do que o segundo. Fonte: [4]

A partir daqui, serão apresentados alguns dos modelos de aprendizado de máquina para problemas de regressão.

3.2 Regressão Linear e Polinomial

A técnica de regressão é muito usada na pesquisa de mercado. Ela permite mapear, através de uma função, a relação entre variáveis independentes (x) que são aquelas que são adquiridas por algum processo de medição ou registro, e variáveis dependentes ($\hat{y}$), que são obtidas por certa combinação entre as variáveis independentes [43, 4]

Esta técnica é útil para a implementação de preditores. De forma mais resumida, poderíamos dizer que a técnica de regressão linear é empregada para aproximar os dados a uma reta de forma a minimizar uma dada função de erro. Na equação abaixo, representamos um modelo de regressão linear para uma única variável x.

$$\hat{y} = \alpha + \beta x \tag{3.3}$$

Na equação 3.3, temos a variável dependente $\hat{y}$ e a variável independente x, o coeficiente angular β, que é capaz de determinar a inclinação da reta e o coeficiente linear α, que representa o valor de intersecção de y quando $x = 0$. Além disso, para encontrar a função melhor ajustada é necessário utilizar a equação 3.2 e encontrar os melhores valores para α e β que minimize o erro.

Apesar da simplicidade do modelo de regressão linear, os problemas reais são muito mais complexos e, na maioria das vezes, é bastante incomum achar problemas em que a estratégia linear possa ser aplicada de forma eficiente. Por isso, surgiu o modelo regressão polinomial, em que a máquina tenta modelar os dados como um polinômio. A Equação 3.4 [44] rege essa regressão. Nela, g é o grau do polinômio.

$$\hat{y} = \beta_0 + \beta_1 x + \beta_2 x^2 + ... + \beta_g x^g \tag{3.4}$$

Além das Equações 3.1 e 3.4, que são simplificadas para um único valor de x, também existe a possibilidade de x corresponder a um vetor com m atributos de entrada (Equação 3.5). Cada elemento desse vetor corresponde a uma característica do problema que influencia na valor da variável de saída.

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_m x_m \tag{3.5}$$

3.3 Árvore de Decisão

Árvores de decisão são representações visuais do caminho que o simples processo de tomada de decisões pode levar. Esse modelo pode ser usado tanto para classificação, quanto para regressão e sua estratégia consiste em dividir um problema em subproblemas menores que possam ser resolvidos de forma recursiva.

Um dos objetivos da árvore de decisão é buscar características comuns e estudar o relacionamento entre classes ou rótulos, a fim de identificar padrões no conjunto de dados, que são úteis para gerar modelos de predição e ter maior conhecimento sobre os dados.

As árvores de decisão são montadas a partir de estruturas hierarquizadas compostas por nós, que podem ser raiz, ramos ou folhas. O modelo das árvores pode ser representado computacionalmente por um conjunto de estruturas condicionais *if/else*. O fluxo inicia-se pelo nó raiz e as possibilidades pelos seus nós ramos. A medida que as possibilidades vão diminuindo, chegamos perto do resultado, que é caracterizado por um nó folha.

Como exemplo, pode-se utilizar a Figura 3.2, que ilustra uma árvore de decisão com nós internos representando atributos (no exemplo, UCs ou quantidade de unidades de computação, e Frequência de Memória). A partir do nó

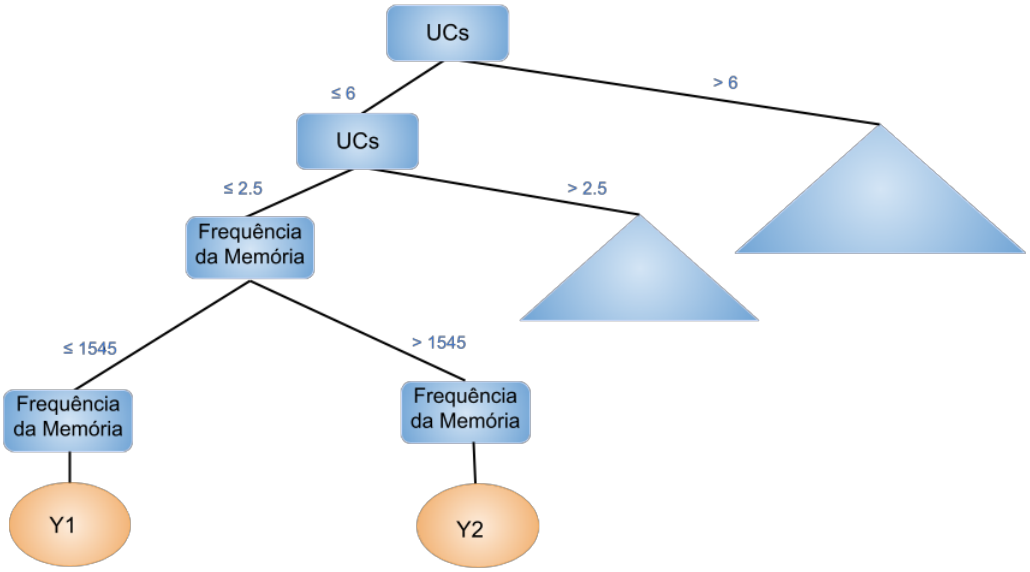


Figura 3.2: Árvore de Decisão

raiz (UCs), dois fluxos são viáveis: o da direita, quando a quantidade de UCs é maior do que 6, ou o da esquerda, quando a quantidade de UCS é menor ou igual a 6. Iniciando pelo fluxo da esquerda, quando a quantidade de UCs é menor ou igual a 6, e a quantidade de UCs é menor ou igual a 2.5, teríamos 2 soluções. Caso a frequência de memória seja menor ou igual a 1545, o resultado predito seria Y1; em contrapartida, se a frequência da memória fosse maior do que 1545 o resultado predito seria Y2.

3.4 Random Forest

O algoritmo *Random Forest*, introduzido por [45], usa uma técnica em que vários preditores isolados são agrupados, formando uma floresta, visando obter um melhor resultado de predição. Entretanto, cada árvore de decisão deve possuir uma parte aleatória do vetor de dados, de forma que as árvores não sejam treinadas com o mesmo vetor. Além disso, o conjunto de dados deve possuir características distintas.

A técnica é bastante conveniente, já que pode ser empregada tanto em problemas de classificação quanto de regressão [46]. Para que o *Random Forest* seja empregado, com boa acurácia, em casos de regressão, é necessário que as árvores geradas tenham baixo índice de correlação entre elas. No final, os resultados serão combinados por meio de uma média.

3.5 K-Nearest Neighbours

O *K-Nearest Neighbours* (kNN) é um algoritmo que pode ser usado tanto para classificação quanto regressão. Este algoritmo é baseado na distância

para os exemplos de treinamento, isto é, um elemento a ser classificado recebe uma classificação com base nos exemplos mais próximos a ele. A função de distância varia de acordo com a aplicação, porém, a distância Euclidiana é a mais utilizada [5]. Dados dois exemplos $x^{(i)} = (x_1^{(i)}, \ldots, x_m^{(i)})$ e $x^{(j)} = (x_1^{(j)}, \ldots, x_m^{(j)})$, a distância Euclidiana entre eles é dada por:

$$d = \sqrt{\sum_{k=1}^{m} (x_k^{(i)} - x_k^{(j)})^2}. \tag{3.6}$$

O problema consiste em, dado um ponto $x^{(i)}$ a ser classificado, são adquiridas as classificações dos k elementos mais próximos a ele no conjunto de treinamento. Posteriormente, as classificações são agrupadas de tal forma que, se o problema for de regressão, $\hat{y}^{(i)}$ recebe a média entre as classes. Quando o problema é de classificação, o resultado é a classe que mais se repete dentre os k vizinhos mais próximos [5].

A constante k é um parâmetro de entrada do algoritmo e reflete diretamente na qualidade do resultado. Na literatura, tem-se convenções para esse valor. Uma alternativa comum consiste em atribuir um valor ímpar a k, uma vez que, se k for par, em um problema de classificação, pode ocorrer um empate. E ainda, como alternativa, pode ser utilizada a validação cruzada, técnica descrita na Seção 3.8.2, para a escolha do melhor valor do parâmetro k para um determinado conjunto de dados [5].

Assim como o método do kNN traz muitas vantagens aos usuários, ele também apresenta algumas desvantagens. Dentre estas, vale ressaltar o fato de que, se algum rótulo é classificado de forma errônea, então todos os rótulos próximos a ele também estarão errados [47]. Além disso, se o conjunto de dados de teste for muito grande, o tempo de resposta tende a ser grande [5].

3.6 Regressão de Vetores de Suporte

O modelo de Máquinas de Vetores de Suporte (SVM) é baseado em aprendizado supervisionado e faz o uso de técnicas de otimização para resolver equações que envolvem uma série de restrições. As SVMs têm sido muito utilizadas por pesquisadores tanto da área da computação quanto por profissionais de outras áreas devido ao fato de que ele é robusto e os resultados alcançados são fortes, mesmo sem muito esforço de modelagem [48].

As SVMs são popularmente aplicadas em problemas de classificação; entretanto, elas também podem ser usadas em problemas de regressão e, nesse caso, são denominadas *Regressão de Vetores de Suporte* (SVR). Assim como nas SVMs, as SVRs incluem o conceito de margem que representa a menor distância entre um separador e um ponto no espaço. Com a introdução desse

conceito é possível modelar um dado problema.

Como pode ser visto na Figura 3.3, as SVRs buscam limitar o erro ϵ em determinado intervalo de valores e minimizar as margens, representadas na Figura 3.3 pelas linhas pretas. Com a introdução do erro ϵ, todos os pontos no espaço devem estar dentro dessa margem e deve ser encontrada uma reta, retratada pela linha vermelha, que melhor se adeque a esse problema. Entretanto, podem existir pontos fora da curva ou ruídos, como os pontos amarelos da Figura 3.3.

$$\min \frac{1}{2}||\beta||^2 + C\sum_{i=1}^{n}|S_i| \tag{3.7}$$

$$|y_i - \beta x_i| \leq \epsilon + |S_i| \tag{3.8}$$

Esses pontos poderiam ser ajustados se a margem fosse maximizada, o que causaria uma piora na acurácia. Em contrapartida, uma nova variável de folga S foi introduzida consoante à Equação 3.7. Ademais, juntamente a ela também houve o aparecimento de uma constante C que regulariza esta equação atribuindo peso a essa folga. Com isso, a Equação 3.8 mostra a restrição imposta ao separador, de que o módulo dele deve ser menor ou igual a soma do erro e a folga.

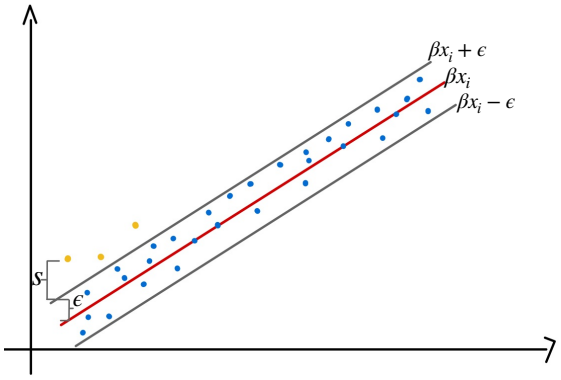


Figura 3.3: SVRs com as folgas.

Muitas vezes os dados não podem ser separados de forma linear, então eles são projetados em um espaço de maior dimensão por meio do *truque do kernel* (*kernel trick*). Se esse espaço é grande o suficiente e a transformação é não-linear, segundo o teorema de Cover [49], as amostras podem ser linearmente mapeadas em suas respectivas classes. Esta função de kernel é representada pela equação 3.9 e é responsável por realizar a transformação de um espaço de dimensão menor em dimensão maior [50].

$$K(x_1, x_2) = \Phi(x_1).\Phi(x_2) \tag{3.9}$$

A função 3.9 recebe, como parâmetros, dois pontos x_1 e x_2 no espaço. Os kernels utilizados nesse trabalho são os da Tabela 3.1 [5]. Nesta tabela, os símbolos $\delta, \kappa, d, \sigma$ são parâmetros das funções.

Tabela 3.1: Tipos de kernels e suas funções [5].

Tipo de kernel	**Função**
Linear	$\langle x_1, x_2 \rangle$
Polinomial	$(\delta \langle x_1, x_2 \rangle + \kappa)^d$
Função de Base Radial (RBF)	$\exp(-\sigma \|x_1 - x_2\|^2)$
Sigmoid	$\tanh(\delta \langle x_1, x_2 \rangle + \kappa)$

3.7 Perceptron Multicamada(MLP)

O Perceptron Multicamada (MLP) é uma rede neural, ou seja, um conjunto de neurônios que são capazes transmitir e responder a estímulos. Esses estímulos são somados e processados pelo corpo celular(que faz parte do neurônio), o resultado desse processamento gera um impulso que pode ser transmitido através da comunicação entre os neurônios também denominada sinapse [38] .

As redes neurais foram inspiradas no funcionamento do cérebro humano, visto que é evidente a sua capacidade de aprendizado. O neurônio artificial imita o comportamento de um neurônio biológico, inicialmente é atribuído um valor x_i e peso w_i para cada entrada do neurônio, tal qual a Equação 3.10. Esses valores são combinados por uma função f, que tem o comportamento similar ao corpo celular. Com isso, é gerado uma saída, conforme mostrado na Equação 3.11.

$$z = \sum_{i=0}^{n} w_i x_i \tag{3.10}$$

$$y = f(z) \tag{3.11}$$

Essas funções, também denominadas de *funções de ativação* podem ser: Identidade, Logística (que implementa a função sigmoid), Tangente Hiperbólica, ou a Relu, cuja função assume o valor de x para intervalos positivos ou zero para intervalos negativos, consoante demonstrado pela Tabela 3.2.

Tabela 3.2: Tipos de função de ativação

Nome	**Função**
Identidade	x
Logística	$1/(1 + \exp(-x))$
Tangente Hiperbólica	$\tanh(x)$
Relu	$max(0, x)$

Os neurônios da MLP são organizados, respectivamente, em camadas: uma camada de entrada, algumas camadas escondidas e uma camada de saída. O processo de aprendizagem tem o objetivo de diminuir o erro. Para isso, pesos randômicos são atribuídos aos neurônios das camadas intermediárias e é calculada a diferença entre saída dessa camada e o resultado esperado, se conhecido. Se esse erro for maior do que a taxa máxima, novos pesos são atribuídos até que esse objetivo seja atingido ou tenha completado a quantidade máxima de ciclos [51].

3.8 Técnicas para Parametrização e Amostragem

3.8.1 Grid Search

A maioria dos modelos de predição usados neste trabalho contém uma vasta gama de parâmetros. Como exemplos, pode-se citar o MLP, que contém a função de ativação, o *solver* utilizado, a quantidade máxima de iterações, entre outros. Além desse, o modelo SVR também possui o parâmetro C, o tipo do *kernel*, o grau e o *gamma*. Assim, técnicas como o *Grid Search* tornaram-se populares para otimizar a parametrização desses modelos.

O *Grid Search* é um método para encontrar a combinação de valores para os parâmetros de um modelo que minimiza o erro de treinamento. Se o tamanho do espaço de busca é muito grande, o *Grid Search* torna-se lento. Supondo que haja n parâmetros com k possíveis valores para cada parâmetro, a complexidade do algoritmo cresce exponencialmente a uma taxa de $O(k^n)$[52].

A fim de buscar soluções ótimas globais, o algoritmo precisa que o espaço de busca definido pelo usuário seja amplo e que haja experimentos anteriores com bom desempenho. Este passo deve ser contínuo [52], até que o erro da solução seja adequado ao desejado.

O problema do aumento exponencial na complexidade do algoritmo e, consequentemente, do tempo de execução, pode ser minimizado usando recursos de paralelização, visto que geralmente as execuções não são dependentes umas das outras[53].

3.8.2 Validação Cruzada

O desenvolvimento de modelos de aprendizado de máquina acarreta alguns desafios. Dentre eles, pode-se ressaltar a habilidade de ajustar o algoritmo para que ele tenha uma boa generalização para muitos casos, mas não sofra de *overfitting* ou *underfitting*. Com o objetivo de contornar esse obstáculo, técnicas de amostragem como validação cruzada, são muito utilizadas [54].

A *validação cruzada* (do inglês, *Cross-Validation*) é o método mais utilizado

para definir o desempenho do preditor. Esta técnica divide o *dataset* em vários subconjuntos de treino e teste e, iterativamente, aplica o modelo em cada um desses subconjuntos, assegurando que o algoritmo seja treinado com diversas combinações. A aplicação do modelo em cada subconjunto gera um valor de performance e, então, o desempenho final é estimado pela média dos valores obtidos [55, 56]. O método de validação cruzada possui inúmeras variantes. A variante utilizada nesse trabalho é a *Cross-Validation k-Fold*, descrita na seção seguinte.

3.8.3 Validação Cruzada k-Fold

Esta técnica consiste em dividir o conjunto de treinamento em k partes, de forma que esses conjuntos tenham tamanhos iguais (ou muito próximos) e os conjuntos sejam disjuntos entre si [54]. Inicialmente, separa-se $k-1$ conjuntos de treino e o restante é utilizado para validação. Esse processo ocorre repetidamente e a parada acontece quando os k conjuntos já foram usados como validação. Cada uma dessas iterações tem um valor de desempenho associado e, ao final, calcula-se a média dos desempenhos alcançados.

Geralmente, essa metodologia é aplicada juntamente com a estratificação, que é capaz de garantir, dentro de cada subconjunto, a mesma proporção da população inicial.

Por exemplo, considerando uma população de 100 GPUs de características distintas. Dada essa população, sabe-se que 30% dessas GPUs são do tipo TITAN X e 70% são do tipo GTX 480, portanto; a estratificação 10-*folds* assegura que dentro de cada subconjunto de treinamento tenha 7 GTX 480 e 3 TITAN X.

3.9 Considerações Finais

Nesse capítulo, foram apresentados os conceitos aprendizado de máquina, modelos para problemas de regressão, técnicas de amostragem, parametrização e as métricas para avaliação. Todas estas técnicas foram utilizadas no decorrer desse trabalho durante o desenvolvimento e avaliação de modelos de predição de sistemas GP-GPU.

CAPÍTULO

4

Trabalhos Relacionados

Neste capítulo serão mostrados os principais trabalhos que apresentam alguma intersecção com nosso projeto, incluindo trabalhos prévios de caracterização e modelagem de desempenho de GPUs utilizado simuladores e/ou sistemas nativos ou utilizando aprendizado de máquina. A Seção 4.3 expõe uma breve comparação entre os trabalhos descritos e o nosso projeto.

4.1 Exploração de Desempenho em GPUs Usando Simuladores

Bakhoda et at. [19] caracterizam várias aplicações de propósito geral escritas em CUDA da NVIDIA no GPGPU-Sim, um simulador funcional de sistemas baseados em GPUs, que executa *threads* paralelas sobre o conjunto de instruções PTX. Para tal, os autores selecionaram doze aplicações CUDA não triviais, demonstrando melhoria de desempenho da execução sobre GPU (versus uma versão sequencial apenas de CPU). Os modelos de hardware de GPUs adotados na simulação são compatíveis às placas gráficas contemporâneas.

Foram explorados parâmetros de sistema de uma arquitetura de GPU como a topologia de interconexão, caches, o projeto do controlador de memória, mecanismos de distribuição da carga de trabalho e coalescência de memória. Ao explorar a interconexão foi levado em consideração o custo e o desempenho. O custo depende basicamente da quantidade e da complexidade dos roteadores; o desempenho depende da latência, largura de banda e quantidade de roteadores pelos quais uma mensagem pode passar desde sua a origem até o seu destino na rede.

O *kernel* de um programa CUDA é composto por grades (ou *grids*) de *threads*. Dentro de uma grade, as *threads* são agrupadas em blocos que são comumente chamados de *vetores de threads cooperativas* (CTA). As *threads* dentro de um CTA têm acesso a uma memória compartilhada, cuja sincronização de seus acessos é realizada por meio de barreiras. As GPUs podem explorar o paralelismo em nível de dados presente nas aplicações e mitigar a latência de acesso à memória por meio da intercalação de *warps* que representam um conjunto de *threads*. Os autores propõem uma distribuição de carga de trabalho o mais uniforme possível, despachando CTAs para *SMs* que possuem o menor número de CTAs em execução. O modelo de caches adotado é uma cache de primeiro nível por *core* (L1) e uma cache de segundo nível (L2) compartilhada por meio da rede.

Após a modelagem desses parâmetros foi alterado o fluxo de execução do simulador de forma a suportar a execução de *threads* CUDA em paralelo, visando avaliar o desempenho das aplicações. Os autores analisaram o comportamento de cada uma das 12 aplicações e resumiram suas principais conclusões: (a) Aplicações não-gráficas tendem a ser mais sensíveis à largura de banda que à latência do mecanismo de interconexão; (b) acessos à cache ou à memória local podem causar queda no desempenho quando não obedecem os princípios de localidade; (c) Para algumas aplicações, diminuir a quantidade de *threads* concorrentes pode aumentar o desempenho pelo diminuição da concorrência por recursos dentro do chip; (d) A exploração de coalescência de memória pode aumentar o desempenho em até 41%.

Zhang e Owens [57] desenvolveram um modelo de desempenho para o modelo de GPU da NVIDIA da série GeForce 200, com o objetivo de detectar gargalos do programa e analisar o desempenho das aplicações, permitindo predizer os benefícios de potenciais otimizações e melhorias arquiteturais. A abordagem é baseada em um *microbenchmark* que explora os três parâmetros mais significativos durante a execução de aplicações sobre GPUs: (1) Pipeline de Instruções; (2) Acesso a Memória Compartilhada; (3) Acesso a Memória Global.

Um dos objetivos do trabalho é ajudar os programadores mostrando como fazer o uso desse *microbenchmark* para otimizar aplicações reais que sofrem com a influência desses três parâmetros. Além do mais, o modelo por eles proposto é capaz de analisar a performance das aplicações e sugerir mudanças arquiteturais visando melhorar a alocação de recursos de hardware, escalonamento de blocos, conflitos de *bank*, granularidade de transações de memória [58]. O modelo proposto é construído sobre um conjunto de instruções nativas de GPU com Kernel OpenCL em GPUs, ao invés de utilizar instruções

assembly do conjunto PTX ou linguagens de mais alto nível. Os resultados estatísticos de desempenho dos programas são obtidos dinamicamente pelo uso do simulador Barra [17]. As principais limitações desse trabalho citadas pelos autores são a falta de um modelo de cache no sistema de memória do simulador e a falta de um modelo que explore os impactos das barreiras de sincronização no desempenho.

Baghsorkhi et al. [59] apresenta um modelo analítico para prever o desempenho de aplicações de propósito geral em uma arquitetura de GPU. O modelo é projetado para fornecer informações de desempenho para um compilador poder realizar ajustes para geração de código mais eficiente ou também pode ser incorporado a uma ferramenta para ajudar os programadores a avaliar melhor os gargalos de desempenho em seus códigos.

O modelo identifica como cada *kernel* utiliza os principais recursos da microarquitetura da GPU. Com base em um gráfico de fluxo de trabalho criado pelo modelo, é possível identificar os gargalos de desempenho com precisão e estimar o tempo de execução de um *kernel de GPU*. O modelo foi validado com aplicações CUDA em GPUs da NVIDIA. As aplicações utilizadas foram multiplicação de matrízes, soma de prefixos, e uma coleção de aplicações sobre matrizes e vetores esparsos, por apresentarem padrões desafiadores de acessos à memória.

Os experimentos comparam os desempenhos obtidos pela aplicação do modelo e os previstos e também mostram que o modelo captura o efeito de todos os principais fatores de desempenho para a arquitetura de GPU.

Para identificar os gargalos de desempenho de aplicações paralelas em arquiteturas de GPU, Hong e Kim [60] propõem um modelo que estima o tempo de execução de programas maciçamente paralelos. O principal parâmetro do modelo é o número de requisições de memória compartilhada, considerando o número de *threads* em execução e a largura de banda da memória. Com base no grau de paralelismo da memória, o modelo estima o custo das solicitações de memória, estimando assim o tempo de execução geral de um programa. Comparações entre o resultado do modelo e o tempo de execução real em várias GPUs mostram que a média geométrica do erro absoluto do modelo usando microbenchmarks é de 5,4% e usando aplicações CUDA é de 13,3%.

4.2 Modelos de Desempenho para GPUs Utilizando Aprendizado de Máquina

Ter uma ferramenta para estimar o desempenho de um trecho de código de uma aplicação antes de escrever o seu código para a GPU é altamente desejável. Para este fim, Ardalani et al. [61] propuseram *Cross-Architecture Performance Prediction (XAPP)*, uma técnica baseada em aprendizado de máquina que usa apenas implementação sequencial em CPU para prever o potencial desempenho de sua execução em GPU.

Os autores citaram como suas principais motivações para o estudo os seguintes argumentos: o tempo de execução na GPU é uma função das propriedades do programa e das características de hardware. Ao examinar um vasta gama de códigos de GPU e seus equivalentes para CPU, pode-se usar técnicas de aprendizado de máquina para aprender essa correlação. Por isso foi usado um modelo adaptativo de dois níveis . Os resultados mostraram que a ferramenta alcançou 26,9% de erro médio em um conjunto de 24 *kernels* do mundo real.

Com base nos dados resultantes da instrumentação binária do código para CPU e no tempo de execução do código equivalente executado em GPU, o modelo de aprendizado de máquina foi aplicado para aprender a relação da execução do programa na CPU e o seu tempo de execução em GPU.

O trabalho apresentado por Wu et al. [62] fornece um modelo capaz de predizer o desempenho e eficiência energética de uma GPU usando técnicas de aprendizado de máquina e redes neurais. Esse modelo é treinado a partir de dados obtidos da execução de um conjunto de aplicações executadas com diversas configurações de hardware. A partir do desempenho e dados de consumo energético obtidos, o modelo aprende como as aplicações escalam a medida que a configuração da GPU é alterada. Os valores dos contadores do hardware alimentam uma rede neural que prediz a curva que melhor representa os dados de treinamento. Tal modelo matemático é usado, a posteriori, para predizer o comportamento (desempenho e consumo energético) de uma nova aplicação e um novo hardware.

Esse modelo foi capaz de estimar a performance da GPU com uma média de erro de 15%. A estimativa de consumo energético tem erro médio de 10 %.

Karami et al. [63] propõem um modelo estatístico para o desempenho de aplicações paralelas para GPUs NVIDIA que auxilia os programadores a encontrar gargalos e mostra as relações entre eles (o que não é possível utilizando apenas um sistema de *profiling*). Foram escolhidas aplicações OpenCL

para fornecer os dados de entrada para o modelo de desempenho, as quais foram executadas em GPU nativas para coleta de valores dos contadores do hardware. Em seguida, os resultados foram usados para criar um modelo de regressão que correlacionam parâmetros e expõe a significância de cada um deles nos resultados de desempenho.

Os resultados apresentados pelos autores mostram que o modelo proposto prevê o comportamento de aplicações paralelas OpenCL em modelos de GPUs com precisão de 91%. Além disso, o modelo proposto é capaz de caracterizar aplicações desconhecidas com base em suas semelhanças com entradas de um banco de dados de referência, o que permite que se possa prever os potenciais gargalos destas novas aplicações.

De acordo com a análise realizada por Jia et al. [64], modestas alterações em um software ou em alguns parâmetros do hardware de GPUs podem levar a grande e imprevisíveis mudanças no desempenho dos programas. Em resposta a esses desafios, os autores propuseram o *Stargazer*, um *framework* para exploração automatizada de desempenho de GPUs baseado em modelos de regressão. A ferramenta escolhe randomicamente amostras de parâmetros de GPUs de um vasto espaço de projeto e fornece tais entradas para um simulador. Em seguida, os valores obtidos a partir das simulações são usados para a construção de um estimador de desempenho que identifica quais são os parâmetros arquiteturais mais significativos e suas correlações. Por trabalhar com um conjunto pequeno de amostras, o uso do *framework* reduz significativamente o tempo de simulação em exploração arquitetural para GPUs. Como resultados de exploração de um espaço de projeto de quase 1 milhão de pontos, usando 300 amostras e 11 aplicações de GPUs, foi possível estimar o tempo de execução das aplicações com apenas 1,1% de erro.

4.3 Síntese dos Trabalhos Relacionados

Como observado nesse capítulo, com a crescente utilização de aceleradores como GPUs para computação de propóstico geral, medir ou estimar o desempenho ou o consumo energético durante ou antes da execução de aplicações paralelas em GPUs tem sido alvo de diversos trabalhos de pesquisa.

Neste capítulo, comentamos alguns dos trabalhos relacionados a este projeto em desenvolvimento. Para facilitar a análise, oferecemos uma reflexão sobre os seguintes critérios, que são considerados significativos no contexto deste projeto e que nortearam a organização da Tabela 4.1.

- Qual a linguagem usada nas aplicações paralelas adotadas para o estudo (Coluna **Linguagem**)?

- Os trabalhos em questão têm ou não foco em facilitar a exploração do espaço de projetos baseados em GPUs (Coluna **DSE**)?
- Os trabalhos em questão levaram em consideração algum tipo de heterogeneidade nos projetos (Coluna **Het**)?
- Os trabalhos realizaram medições por meio de simulação ou predições por meio de técnicas de aprendizado de máquina (Coluna **Sim/Pred**)?
- Os trabalhos em questão apresentaram algum estudo relacionado ao aparecimento de dark-silicon nos projetos a medida que escalam os parâmetros arquiteturais (Coluna **DS**)?

Tabela 4.1: Síntese dos trabalhos relacionados

Trabalhos relacionados	**Hetero-geneidade**	**Lingua-gem**	**Sim/ Pred**	**DSE**	**DS**
Bakhoda [19]	Não	CUDA	Simulação	Não	Não
Zhang [58]	Não	CUDA	Simulação	Não	Não
Baghsorkhi [59]	Não	CUDA	Simulação	Não	Não
Ardalani [61]	Não	OpenCL	Predição	Não	Não
Wu [62]	Não	OpenCL	Predição	Sim	Não
Karami [63]	Não	OpenCL	Predição	Sim	Não
Stargazer [65]	Não	CUDA	Predição	Sim	Não
Hong [60]	Não	CUDA	Simulação	Não	Não
Nossa proposta	**Sim**	**CUDA**	**Predição**	**Sim**	**Sim**

Por meio da Tabela 4.1 é possível visualizar algumas semelhanças e divergências entre os trabalhos apresentados. Nenhum dos artigos possuem exploração de sistemas com alguma heterogeneidade (CPU/GPU ou GPU/GPU); Apenas três dos trabalhos citados tem a preocupação de realizar uma exploração do espaço de projetos com objetivo de buscar o melhor *hardware* para um determinado conjunto de aplicações. Embora [64] tenha mencionado o uso de amostras de um vasto espaço de projetos, a escolha randômica e de um conjunto muito pequeno de amostras descaracteriza esse trabalho como parte do contexto de exploração do espaço de projetos. A coluna de simulação/predição permite que caracterizemos os trabalhos que realizaram medições por meio de simuladores ou sistemas nativos e aqueles que utilizaram a abordagem de aprendizado de máquina para automatizar o processo e reduzir o tempo de simulação em um vasto espaço de possibilidades. Na última coluna é interessante a informação que de todos os trabalhos apresentados nenhum é ciente de *Dark Silicon*, ou seja, por mais que alguns tenham realizado exploração arquitetural ou estimativas de desempenho ao escalar parâmetros das

GPUs visando melhorar o desempenho de um determinado conjunto de aplicações, nenhum deles se preocupou com a viabilidade física de tal projeto, ou seja, se o projeto ideal extrapola algum limite de área, densidade de potência ou consumo energético que venha a comprometer sua implementação real.

CAPÍTULO

5

Desenvolvimento e Avaliação do Preditores de Desempenho de GPUs

O objetivo deste trabalho é implementar um preditor de desempenho de sistemas heterogêneos GP-GPU e, sem seguida, adicioná-lo ao fluxo de exploração do espaço de projetos ciente de dark-silicon do MultiExplorer.

Trabalhos anteriores contribuíram com preditores de desempenho de sistemas *multicore* heterogêneos [26], tornando natural a aplicação de uma metodologia similar para predizer o desempenho de GPUs também heterogêneas. O nível de heterogeneidade que se deseja, no contexto de GPUs, corresponde ao uso de unidades de computação (UCs) de distintas arquiteturas de GPUs em um mesmo sistema.

O objetivo deste capítulo é descrever a metodologia de desenvolvimento e de avaliação dos preditores de desempenho de GPUs heterogêneas.

5.1 Conjunto de Treinamento

O conjunto de treinamento foi obtido por meio de dados de simulação oriundos do simulador GPGPU-Sim, bem como das informações arquiteturais das GPUs disponibilizadas pela empresa fabricante (NVIDIA [66, 67, 68, 69, 70]). Para realizar as estimativas físicas dos projetos foi utilizado o McPat [71], estendido com código para a realização da estimativa de *dark-silicon*. A Tabela 5.1 contém os principais atributos dos modelos de GPUs adotados neste trabalho.

Foram modeladas seis GPUs, as quais cobrem cinco arquiteturas diferentes: Fermi, Pascal, Turing, Kepler e Volta. Os modelos originais estão disponí-

veis em https://github.com/gpgpu-sim/gpgpu-sim_distribution. Com isso, foram detalhados, na Tabela 5.1, alguns dos parâmetros que influenciam no tempo de execução: a frequência de operação da placa; o número de UCs, que são escaláveis de acordo com os modelos disponíveis para cada arquitetura; a capacidade de memória compartilhada; o número de blocos por UC; o número de *threads* por UC e o número de registradores por core.

As aplicações escolhidas neste trabalho pertencem ao *benchmark* NVIDIA CUDA SDK [33], disponível no GPGPU-Sim e ao *benchmark* Rodinia [34] que possibilita a simulação de computação heterogênea utilizando CPU e GPU(API CUDA) a fim de explorar o paralelismo das aplicações. As aplicações utilizadas são as seguintes:

- **Simple Reference(asyncAPI):** Esta aplicação simula chamada de eventos CUDA assíncronas enquanto ocorre a sobreposição da execução de CPU e GPU;
- **clock:** É uma aplicação que avalia performance de um bloco de threads por uma função de clock;
- **vectorAdd:** É uma aplicação que realiza soma de vetores;
- **K-nearest Neighbors(NN)** É uma aplicação que calcula a distância Euclidiana de forma paralela, dividindo a computação entre múltiplas threads e aplica o algoritmo kNN;
- **Back Propagation (backprop):** é um algoritmo muito usado em redes neurais para treinamento e adequação de pesos entre os nós. Esse algoritmo contém duas fases: a de avanço em que passa da camada de entrada a camada de saída; e fase de retrocesso, que o erro é passado para trás a partir da camada de saída buscando corrigir os pesos;
- **Needleman-Wunsch(NW):** é um algoritmo de programação dinâmica popularmente usado na bioinformática para otimização de sequências de DNA;
- **Discrete Wavelet Transform (dwt2d):** é um algoritmo de processamento de sinal digital ;
- **Breadth-First Search (BFS):** é um algoritmo busca em largura em grafos;
- **Hotspot (hotspot):** é uma ferramenta capaz de simular medidas de potência, e temperatura de processadores através da resolução de equações diferenciais [72];

Foi realizado um *profilling* dessas nove aplicações usando NVIDIA Visual Profiler [33] e, a partir disso, foram obtidos o número de instruções, tempo de execução no *kernel* e o tempo gasto para transferir dados de/para a memória, os quais estão disponíveis na Tabela 5.2.

Tabela 5.1: Modelos de GPUs

Arquitetura de GPU:	Fermi	Pascal	Turing	Volta	Volta	Kepler
Modelo de GPU	GTX480	TITAN X	RTX2060	QV100	TITAN V	GK110
Frequência (Mhz):	700	1417	1365	1132	1200	837
Número de UCs:	4, 8, 15, 32, 64	8, 16, 28	4, 8, 16, 30	4, 8, 16, 40	4, 8, 16, 40	4, 8, 14, 32
Memória Compartilhada (KB):	8192, 16384, 32768, 98304, 100352 (todos os modelos)					
Blocos por UCs:	2, 8, 16, 32 (todos os modelos)					
Threads por UC:	512, 2048, 1024 (todos os modelos)					
Registradores por core:	4096, 16384, 32768, 65536 (todos os modelos)					

Tabela 5.2: Características das Aplicações

Aplicação	**Pacote de Benchmark**	**# Instruções**	**% *kernel***	**% Memória**
asyncAPI	CUDA Library	218.103.808	1,69%	98,31%
clock	CUDA Library	1.950.272	67,74%	32,26%
vectorAdd	CUDA Library	1.051.936	3,09%	96,91%
nn	Rodinia	1.201.052	3,91%	96,09%
backprop	Rodinia	134.545.760	6,48%	93,51%
needle	Rodinia	102.311.808	12,63%	87,36%
dwt2d	Rodinia	185.349.936	7,77%	92,23%
bfs	Rodinia	551.590.436	28,01%	71,99%
hotspot	Rodinia	87.962.832	12,02%	87,98%

A combinação de todos esses parâmetros arquiteturais resultam em um conjunto de 5.760 plataformas distintas. A simulação das nove aplicações em cada plataforma geraria 51.750 experimentos. Alguns dos experimentos levam poucos minutos, mas outros levam algumas horas. Em média, considerando essa infraestrutura computacional, simular todas as plataformas e aplicações iria consumir aproximadamente 2.160 dias de computação ininterruptos. Para tornar esse processo viável, foram selecionadas aleatoriamente 10.613 combinações, dentre as 51.750 possíveis. Estas simulações foram executadas paralelamente em quatro computadores, consumindo um total de 280 horas de simulação.

O cálculo de desempenho por aplicação Y é dado em *milhões de instruções por segundo* (MIPS). Primeiramente, é realizada a multiplicação da quantidade de ciclos (C) pelo inverso da frequência de operação (f); a partir disso, calcula-

se o tempo de execução aproximado $Runtime$, conforme a Equação 5.1. Em seguida, o desempenho final Y, em MIPS, é dado pela divisão entre a quantidade de instruções da aplicação (Inst) pelo tempo de execução estimado ($Runtime$), consoante a Equação 5.2.

$$Runtime = C \times \frac{1}{f} \tag{5.1}$$

$$Y = \frac{Inst}{Runtime} \tag{5.2}$$

O *dataset* foi definido como o conjunto de pares $T = \{(X, Y)\}$, em que $X = (x_1, \ldots, x_m) \in \mathbb{R}^m$ é um vetor composto por m atributos de entrada (no nosso caso, $m = 8$) que representam a entrada (plataformas e aplicações). Mais especificamente, os seguintes atributos são incluídos no *dataset*:

- Quantidade de unidades de computação (UCs ou SMs) na plataforma;
- Frequência da UC;
- Tamanho da memória compartilhada;
- Quantidade de blocos por UC;
- Quantidade de threads por UC;
- Quantidade de registradores por core;
- Quantidade de instruções do *kernel*;
- % Tempo de *kernel*.

O valor $Y \in \mathbb{R}$ corresponde ao desempenho da plataforma executando a aplicação em questão, o qual é definido pela Equação 5.2, a partir de dados resultantes da simulação no GPGPU-Sim.

Uma observação a se fazer é que os primeiros seis atributos são relacionados à arquitetura de GPU e os últimos dois são dependentes do *profilling* da aplicação. Já que o simulador é determinístico, cada experimento foi executado apenas uma única vez. Além disso, foi possível distribuir a simulação em quatro computadores com as seguintes configurações: dois computadores com processador Intel Core i7, 16 GB RAM, e uma NVIDIA GeForce GT730, um 8-core Intel i7, 8MB cache, 15GB RAM e NVIDIA GeForce GTX 745 com 284 cores; um Intel(R) Xeon(R), 64GB, e 4x NVIDIA GeForce GTX 1080 Ti 4x.

5.1.1 Análise de correlação de features

Para entender melhor os atributos de entrada e a relação entre eles, exibe-se, na Figura 5.1, um mapa de calor com os valores absolutos da correlação

de Pearson [73], considerando-se os atributos numéricos do *dataset*. Esse coeficiente expressa, utilizando valores entre -1 e 1, a forma como as variáveis se relacionam, duas a duas. Se o coeficiente for igual ou muito próximo a 1, indica que uma variável cresce na medida em que a outra também cresce. Em contrapartida, quando esse valor se aproxima de -1, implica que as variáveis se relacionam de maneira inversamente proporcional. No caso em que o coeficiente é igual ou muito próximo de zero, o crescimento ou decréscimo de uma variável não afeta a outra. A Figura 5.1 exibe o valor absoluto desse coeficiente, portanto temos valores entre 0 e 1. Esta análise indica que os atributos que mais têm impacto no desempenho são: a quantidade de UCs da plataforma, a quantidade de instruções do *kernel* da aplicação e a frequência de operação da plataforma.

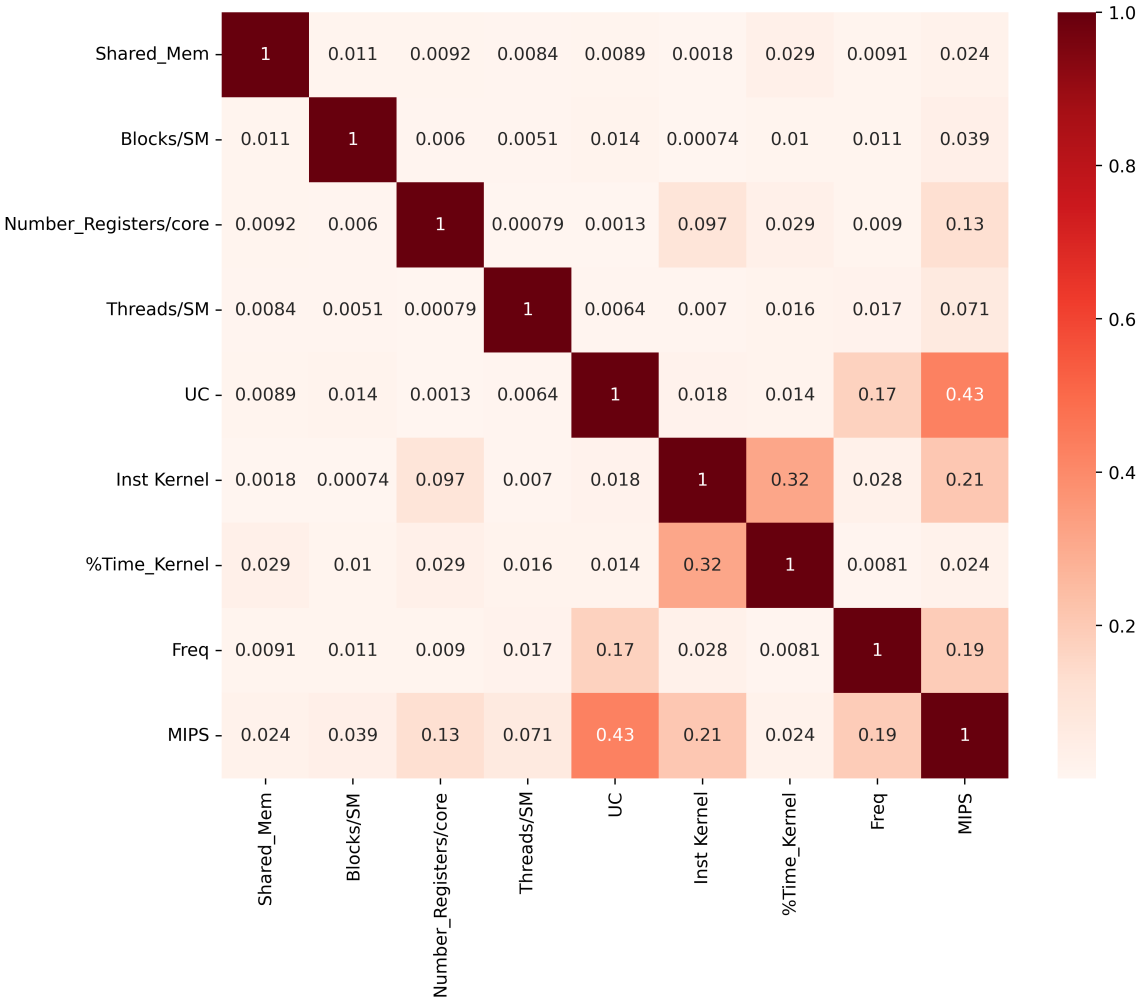
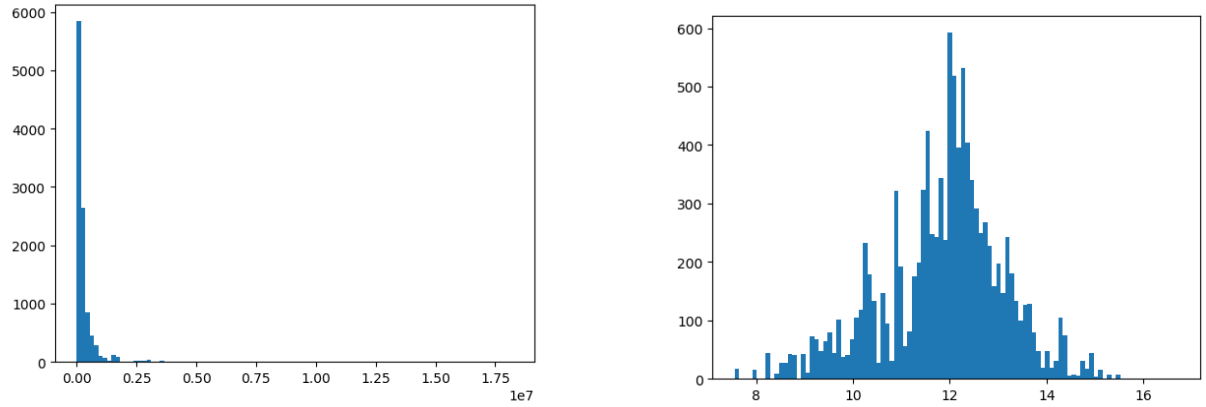


Figura 5.1: Valor absoluto da correlação de Pearson entre os atributos (tanto de entrada quanto de saída) do *dataset*.



(a) (b)

Figura 5.2: (a) Distribuição da variável Y original. (b) Distribuição da variável Y após aplicação da função logarítmica.

5.1.2 Pré-Processamento dos Dados

Após experimentos preliminares, foi observado um erro muito alto dos modelos de predição. Observando a distribuição, ilustrada na Figura 5.2a, pode-se notar que a variável Y apresenta um decaimento (aparentemente) exponencial. Uma variável com uma distribuição deste tipo traz problemas para a maioria dos algoritmos de aprendizado de máquina.

Foi, então, aplicada a função logarítmica na variável a ser predita Y. Com isto, a distribuição desta variável transformada ficou similar a uma distribuição normal, como é demonstrado na Figura 5.2b, que é mais adequada para os algoritmos de aprendizado de máquina.

5.2 Avaliação dos Preditores

Os algoritmos e métricas de avaliação utilizados estão disponíveis na biblioteca *scikit-learn* [74]. Os preditores foram desenvolvidos utilizando os seguintes algoritmos de aprendizado de máquina: Regressão linear, kNN, Árvores de Decisão, Random Forest, SVMs com seus diversos *kernels*, MLP, Ada.

A métrica utilizada para avaliação dos modelos foi o *erro percentual médio* (MAPE, do inglês, *mean average percentage error*). Esta métrica indica a diferença entre o valor de desempenho esperado (Y^i) e o valor predito ($\hat{Y}^i$) para n exemplos ($i = 1, \ldots, n$), como ilustrado na Equação 5.3.

$$\text{MAPE} = \frac{1}{n} \sum_{i=1}^{n} \frac{\left| Y^i - \hat{Y}^i \right|}{Y^i} \tag{5.3}$$

O método *grid search* [53] com validação cruzada 10-fold com dados agru-

pados e não agrupados por UCs foram usados para escolher os parâmetros mostrados nas Tabelas 5.5 e 5.3. Os melhores parâmetros ajustados em cada um dos algoritmos foram:

- Random Forest: quantidade de árvores na floresta (`n_trees` $= 10, 50, 100$); profundidade máxima da árvore (`max_depth` $= 2, 4, 8, 16$,None); o número mínimo de amostras necessárias para dividir um nó interno (`min_split` $= 2, 4, 8, 16$); e o número mínimo de amostras em um nó folha (`min_leaf` $= 1, 2, 4, 8$).

- kNN: o número de vizinhos (`n_neighbors` $= 1, 3, 5, 7$).

- Árvores de Decisão: profundidade máxima da árvores (`max_depth=2,4,8,16,None`); o número mínimo de amostras necessárias para dividir um nó interno (`min_split` $= 2, 4, 8, 16$); e o número mínimo de amostras em um nó folha (`min_leaf` $= 1, 2, 4, 8$).

- SVR: O ajuste para o parâmetro C nos modelos baseados em SVR foi realizado por meio do treinamento com quatro *kernels* (linear, polinomial, RBF e sigmoid) variando C de 1 a 10^3 e graus variando de 2 a 7, no caso do *kernel* polinomial.

- MLP: O número máximo de iterações(max_iter= 200,500,1000); a função de ativação para as camadas escondidas(activation= relu, logistic); o solver para otimização de pesos(solver=lbfgs, sgd, adam); número randômico de gerações para peso e o viés de inicialização(random_state= 1, None).

Para os demais métodos e parâmetros, utilizamos os valores padrão da biblioteca *scikit-learn*.

A eficiência também é essencial nessa análise, e por isso faz-se necessário avaliar os preditores com relação ao tempo de resposta para realizar uma predição (latência) ou à vazão ou *throughput* (quantidade de predições por unidade de tempo). Nas Tabelas 5.4 e 5.6 são apresentados o tempo médio de resposta de cada preditor para executar 1000 predições (*Latência*), o desvio padrão desse valor (*Desvio Padrão*) e a quantidade de predições por segundo (*Vazão*). A média e o desvio padrão calculados são medidos a partir de mil execuções do preditor e para mil exemplos em cada execução.

5.2.1 Resultados com Agrupamento por Quantidade de UCs

No contexto do MultiExplorer, é necessário que o modelo preditor generalize para diferentes quantidades de UCs, isto é, o modelo precisa ter um bom desempenho, mesmo quando aplicado a uma instância cuja quantidade de UCs

não exista no conjunto de treinamento. Desta forma, a validação cruzada dos modelos precisa ser agrupada por quantidade de UCs. No caso da validação k-fold, isto significa que dois folds diferentes não podem conter instâncias com quantidade de UCs iguais.

Na Tabela 5.3, é mostrado o erro percentual médio (MAPE) e o desvio padrão obtidos a partir da validação cruzada 10-folds, agrupada por quantidade de UC, e também o melhor conjunto de parâmetros para cada modelo visando predizer o desempenho.

A Figura 5.3 ilustra a acurácia dos modelos utilizando a métrica MAPE e o throughput, cujos valores foram mostrados na Tabela 5.3. É possível verificar que, quando consideramos apenas o MAPE do conjunto de validação, o melhor modelo é o MLP, com erro de 11.6% e desvio padrão de 3.4%; o segundo melhor modelo, considerando a mesma métrica, é o SVR com *kernel* RBF, com 14% de erro e desvio padrão de 3%.

Quando avalia-se estes mesmos modelos, considerando-se também o throughput, observa-se que o SVR com *kernel* RBF apresenta baixa vazão e que os métodos com alta vazão, como a regressão linear ou o Decision Tree, apresentam MAPE baixo. Assim, conclui-se que o MLP apresenta as melhores condições de desempenho dentre os modelos avaliados, com relação a estes dois critérios.

Tabela 5.3: Desempenho dos modelos de regressão avaliados por meio de validação cruzada 10-fold e agrupamento por UCs.

Modelo	**MAPE**	**Desvio Padrão**	**Parâmetros**
MLP	11.6 %	3.4 %	random_state= 1, activation=relu, max_iter= 1000, solver= lbfgs
SVR (*kernel* RBF)	14%	3%	C=100, *kernel*=rbf, gamma= auto, degree=1
SVR (*kernel* Polinomial)	16 %	4 %	C=100, *kernel*=poly, gamma=auto, degree=3
Regressão Linear	35 %	10 %	–
SVR (*kernel* Linear)	38%	12%	C=100 , *kernel*= linear, gamma=auto
Random Forest	44%	59.2 %	n_trees=50, max_depth=None, min_split4=, min_leaf=1
Árvores de decisão	55%	56%	min_split=8, min_leaf=1
SVR (*kernel* Sigmoid)	57%	36 %	C=1, *kernel*=sigmoid, gamma=auto
kNN	71.9 %	92%	n_neighbors=3
Ada	73%	30 %	n_estimators= 100, loss=square, learning_rate=1.0, random_state=2

Tabela 5.4: Avaliação dos modelos agrupados por UC com relação à latência e vazão (os modelos estão ordenados da maior para a menor vazão alcançada).

Modelo de Aprendizado de Máquina	**Latência** ($\times 10^{-6} seg$)	**Desvio Padrão** ($\times 10^{-6} seg$)	**Vazão** ($\times 10^{6}$ pred/seg.)
Linear	0.0765	0.0001	15.419
Árvore de Decisão	0.110	0.0001	9.122
MLP	0.445	0.0003	2.249
Random Forest	4.779	0.0002	0.209
Ada	6.400	0.0005	0.156
SVR (*kernel* Polinomial)	37.079	0.0009	0.027
kNN	43.725	0.0018	0.023
SVR (*kernel* RBF)	46.567	0.0022	0.021
SVR (*kernel* Linear)	48.870	0.0019	0.020
SVR (*kernel* Sigmoid)	111.569	0.0016	0.009

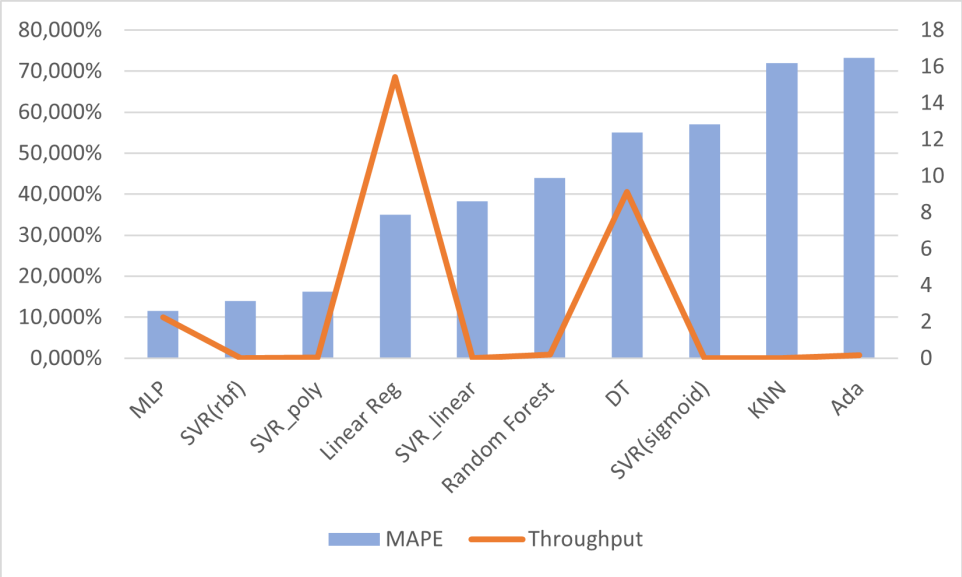


Figura 5.3: Os modelos agrupados pela UC estão ordenados pelo MAPE (barras), em contraste com os valores de vazão ou throughput (linha) em predições por segundo

5.2.2 Resultados Sem Agrupamento

É possível ver na Tabela 5.5, que os resultados de MAPE do conjunto de validação para experimentos que desconsideram o agrupamento, são completamente diferentes. A partir destes dados, vê-se que que o melhor modelo, dada a métrica escolhida, seria a Árvore de Decisão, com 2% de erro e desvio padrão de 0.7%. Esses resultados foram obtidos após a otimização de parâmetros com o *Grid Search* e validação cruzada 10-fold. Já o segundo melhor modelo seria o Random Forest com 2.5%. Por outro lado, dando importância ao *throughput* ou à vazão, a árvore de decisão ainda é o melhor modelo, já que realiza maior quantidade de predições por segundo. Em segundo lugar estaria o modelo de regressão linear e em terceiro lugar está o modelo MLP.

Apesar dos modelos apresentados terem bons resultados, o modelo de ár-

vore de decisão é muito ruim no contexto do problema de exploração do espaço de projetos, uma vez que é necessário generalizar o desempenho, escalando a quantidade de UCs, ao passo de 1 em 1, tornando inviável sua aplicação para um vasto espaço de projeto. Dada essa limitação, optou-se por treinar e utilizar modelos de predição agrupando os atributos pela UCs.

Tabela 5.5: Desempenho dos modelos de regressão avaliados por meio de validação cruzada 10-fold. Reportamos a média e o desvio padrão do MAPE para 10 folds (ordenados pelo MAPE).

Modelo	**MAPE**	**Desvio Padrão**	**Parâmetros**
Árvores de decisão	2 %	0.7 %	min_split=2, min_leaf=1
Random Forest	2.5%	0.2 %	n_trees=50, max_depth=None, min_split=2, min_leaf=1
MLP	6.3 %	0.2 %	random_state=1 , activation=relu, max_iter=1000 , solver=lbfgs
SVR (*kernel* RBF)	10 %	0.25%	C=100, *kernel*=rbf, gamma= auto, degree=1
SVR (*kernel* Polinomial)	11%	0.5 %	C=100, *kernel*=poly, gamma=auto, degree=3
Regressão Linear	31 %	1 %	–
SVR (*kernel* Linear)	32%	1%	C=100 , *kernel*= linear, gamma=auto
kNN	33%	2 %	n_neighbors= 3
Ada	35%	1 %	n_estimators= 200, loss=square, learning_rate=1.0, random_state=2
SVR (*kernel* Sigmoid)	43%	1 %	C=1, *kernel*=sigmoid, gamma=auto

Tabela 5.6: Avaliação dos modelos com relação à latência e vazão (os modelos estão ordenados da maior para a menor vazão alcançada).

Modelo de Aprendizado de Máquina	**Latência** ($\times 10^{-6} seg$)	**Desvio Padrão** ($\times 10^{-6} seg$)	**Vazão** ($\times 10^{6}$ pred/seg.)
Árvore de Decisão	0.10	0.0001	10.15
Linear	0.19	0.0011	5.19
MLP	0.75	0.0012	1.33
Random Forest	5.67	0.0006	0.18
Ada	13.68	0.0004	0.07
SVR (*kernel* Polinomial)	40.00	0.0017	0.02
kNN	46.74	0.0013	0.02
SVR (*kernel* RBF)	48.82	0.0014	0.02
SVR (*kernel* Linear)	52.96	0.0018	0.02
SVR (*kernel* Sigmoid)	120.71	0.0023	0.01

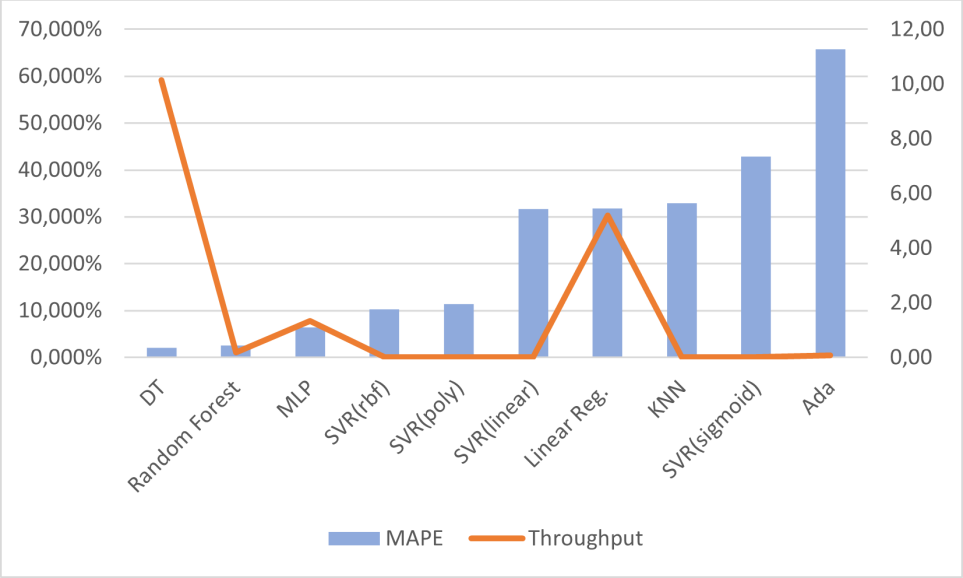


Figura 5.4: Os modelos estão ordenados pelo MAPE (barras), em contraste com os valores de vazão ou throughput (linha) em predições por segundo

5.3 Considerações Finais

Nesse Capítulo foi descrito o conjunto de treinamento, construído com base nas plataformas de GPUs e suas arquiteturas, as variáveis utilizadas para descrever as plataformas e a correlação entre elas. Também foi definida a métrica baseada em MIPS (milhões de instruções por segundo) para caracterizar o desempenho das plataformas e as métricas para avaliação dos modelos de predição.

Ademais, na Seção 5.2 foram evidenciados os modelos de aprendizado de máquina e a aplicação de GridSearch para escolha dos melhores parâmetros para cada um deles. Com isso, verificou-se que o modelo com menor erro é o de árvores de decisão, com MAPE de 2%. Entretanto, observou-se que esse modelo não faz boa generalização à medida que se escala a quantidade de UCs. Esse atributo é importante, não somente no contexto do uso do preditor na ferramenta MultiExplorer, mas também pelo fato de que pela análise de correlação de Pearson, esse parâmetro apresentou maior correlação com o desempenho alcançado pelas plataformas.

Isso posto, concluiu-se que o modelo o MLP, com MAPE de 11.6%, é o mais adequado para integração à ferramenta MultiExplorer, já que faz uma melhor generalização com relação à quantidade de UCs, além conter uma boa vazão.

CAPÍTULO

6

Integração dos Preditores ao MultiExplorer

O objetivo deste Capítulo é explicar a metodologia adotada na ferramenta MultiExplorer para realizar exploração de espaço de projetos (DSE) de sistemas multiprocessadores e, então, mostrar a inclusão do preditor de desempenho de GPUs no módulo de DSE da ferramenta e demais extensões para viabilizar o suporte à DSE com GPUs heterogêneas.

6.1 Integração ao Multiexplorer

Inicialmente, o Multiexplorer fazia estimativas de desempenho, área e exploração de espaço de projetos, apenas para arquiteturas com múltiplos *cores* de CPU. Como este trabalho tem foco em GPUs, foi necessário realizar uma vasta extensão no escopo do projeto, conforme apresentado nos elementos marcados pela cor vermelha na Figura 2.3.

A fim realizar a simulação de desempenho de sistemas contendo GPUs, foi feita a introdução da ferramenta GPGPU-Sim [19, 20] e as aplicações dos benchmaks NVIDIA CUDA SDK [33] e Rodinia [34].

Em seguida, os modelos apresentados na Tabela 5.1 foram modelados para obtenção de estimativas físicas do projto, como área, potência, densidade de potência e a porcentagem de *dark-silicon*, usando o McPAT [35]. Para tal, a ferramenta McPAT foi estendida para fornecer a área de *dark-silicon* de um projeto contendo GPUs, como será descrito na seção seguinte.

Na etapa de exploração de espaço de projetos, foi adicionado ao fluxo o

preditor de desempenho de GPUs desenvolvido neste trabalho, por meio de algoritmos de aprendizagem de máquina. Ademais, foram acrescentados os modelos de GPUs, no banco de núcleos utilizado pelos algoritmos de DSE.

6.2 Estimativa de Dark-Silicon com GPU

A primeira extensão do MultiExplorer para suporte às arquiteturas de GPUs foi realizada no módulo de estimativa de *dark-silicon*, o qual é responsável por estimar a área em *dark-silicon* de uma plataforma inicial. A metodologia original apresentada por Santos et al. [8] oferecia a estimativa para sistemas *multicore* e foi estendido para sistemas GPU neste trabalho de mestrado, de acordo com a metodologia descrita a seguir.

Um projeto inicial de GPU é definido a partir de um processo tecnológico base de $90nm$, o qual é considerado livre de *dark-silicon*, de acordo com a escala Dennard [7]. Então, o projeto é evoluído usando processos tecnológicos de $65nm$, $45nm$, $32nm$, e $22nm$, aumentando a quantidade de UCs e mantendo a mesma área do chip.

A metodologia para estimar *dark-silicon* usa a diferença de densidade de potência, entre o projeto novo e o projeto original (Equação 6.1). A densidade de potência foi calculada dividindo a potência do chip pela área, antes e depois da escala do processo tecnológico.

Considerando que DP_{new} e DP_{base} são a densidade de potência do projeto novo e do projeto original, respectivamente, se a diferença em densidade de potência entre o novo projeto e o projeto original for maior do que zero, pode-se dizer que parte do chip está em *dark-silicon*.

$$\Delta DP = DP_{new} - DP_{base} \tag{6.1}$$

Nesse caso, o próximo passo é estimar a área afetada. Então, a potência excedida é calculada usando ΔDP e a área do chip A como mostrado na equação 6.2.

$$P_{exc} = \Delta DP \times A_{chip} \tag{6.2}$$

Cada chip de processamento tem componentes com diferentes densidades de potência. Tais componentes são candidatos para uma estratégia de exploração de espaço de projeto desde que mudanças nas configurações de tais componentes irá mudar a potência total do chip e, como consequência, a área em *dark-silicon* no chip.

A densidade de potência, o tamanho do circuito de referência, e a potência excedida foram usados para estimar a porcentagem de *dark-silicon* no chip.

Nos chips compostos de GPUs, as UCs são componentes com a menor densidade de potência, e por isso são usados como circuitos de referência.

Considerando que A_{circ} e a P_{circ} são a área e a potência do circuito de referência, respectivamente, a equação 6.3 apresenta a área em *dark-silicon* no chip (A_{DS}), que pode ser expressa em porcentagem de área do chip como mostramos na Equação 6.4:

$$A_{DS} = \frac{P_{exc}}{P_{circ}} \times A_{circ} \tag{6.3}$$

$$DS = \frac{A_{DS}}{A_{chip}} \tag{6.4}$$

A fim de estimar a área de *dark-silicon* no projeto de GPUs, foram realizados experimentos no seis modelos de GPUs da Tabela 5.1, modelados em tecnologia de fabricação de $90nm$. O projeto original foi evoluído para $65nm, 45nm, 32nm$, e $22nm$ (limite máximo suportado pela ferramenta GP-GPUSim), escalando o número de UCs, buscando manter a área o mais próximo possível da área do chip original, a mesma frequência de operação e mesma voltagem. Para cada modelo, alguns parâmetros físicos foram estimados e são mostrados na Tabela 6.1.

Verifica-se que há um significante aumento na densidade de potência nos distintos processos tecnológicos, as quais se justificam pelo aumento na potência total (dinâmica e estática). Verificou-se, também, que a potência estática e a potência gerada pela corrente de fuga são as principais responsáveis pelo aumento da potência total. Esse fator corrobora com a estimativa de Dennard, a respeito aumento na potência de projetos abaixo de $90nmm$ e, consequentemente, com a existência de *Dark silicon* .

É ilustrado na Tabela 6.1 as estimativas de *dark-silicon* de cada plataforma de GPU, usando uma unidade de computação como circuito de referência. A estimativa de *dark siliCon* segue o exposto nas Equações 6.1 a 6.4. Como exemplo, a plataforma GTX480 ($65nm$) tem estimativa de *dark-silicon* de 9.05%. A porcentagem de *dark-silicon* é calculada por:

$$\Delta DP = DP_{atual} - DP_{base} = 0.24 - 0.22 = 0.02$$

$$P_{exc} = \Delta DP \times A_{chip} = 0.02 \times 1822 \approx 37$$

$$A_{DS} = \frac{P_{exc}}{P_{circ}} \times A_{circ} = \frac{37}{15.03} \times 67 \approx 165$$

$$DS = \frac{A_{DS}}{A_{chip}} = \frac{165}{1822} \approx 0.0905 = 9.05\%$$

Se a área correspondente a esta porcentagem for inutilizada, chip de GPU

teria a mesma potência do projeto de chip original (projetada em $90nm$). Considerando que a maioria das plataformas de chip com $22nm$ tem mais do que 44% da área comprometida, a exploração arquitetural do sistema, em tempo de projeto, parece ser uma alternativa promissora para mitigar os efeitos de *dark-silicon* nos projetos.

Tabela 6.1: Estimativas físicas de GPUs

GTX480 (Fermi)					
Número de UCs	15	29	51	95	175
Tecnologia (nm)	90	65	45	32	22
Frequência (MHz)	700	700	700	700	700
Área do chip(mm^2)	1854	1822	1845	1828	1850
Densidade de Potência $(\frac{W}{mm^2})$	0.22	0.24	0.31	0.34	0.39
Área do circuito de referência (mm^2)	123	67	36	19	11
Potência do circuito de referência (W)	24.49	15.03	10.70	6.55	4.03
Porcentagem de dark-silicon (%)	-	9.05%	29.99%	36.28%	44.71%
QV100 (Volta)					
Número de UCs	80	145	270	503	920
Tecnologia (nm)	90	65	45	32	22
Frequência (MHz)	1132	1132	1132	1132	1132
Área do chip(mm^2)	8883	8824	8865	8802	8858
Densidade de Potência $(\frac{W}{mm^2})$	0.26	0.30	0.38	0.45	0.52
Área do circuito de referência (mm^2)	111	61	33	17	10
Potência do circuito de referência (W)	28.30	18.10	12.15	7.77	4.97
Porcentagem de dark-silicon (%)	-	13.66%	30.05%	41.45%	49.50%
TITAN V (Volta)					
Número de UCs	40	73	150	281	595
Tecnologia (nm)	90	65	45	32	22
Frequência (MHz)	1200	1200	1200	1200	1200
Área do chip(mm^2)	10495	10450	10471	10459	10430
Densidade de Potência$(\frac{W}{mm^2})$	0.34	0.39	0.55	0.62	0.73
Área do circuito de referência (mm^2)	261	143	70	37	19
Potência do circuito de referência (W)	87.83	54.60	38.30	23.05	13.87
Porcentagem de dark-silicon (%)	-	11.74%	38.27%	45.18%	53.03%
TITAN X (Pascal)					
Número de UCs	28	53	110	198	415

Tecnologia (nm)	90	65	45	32	22
Frequência (MHz)	1417	1417	1417	1417	1417
Área do chip(mm^2)	4477	4429	4442	4499	4456
Densidade de Potência $(\frac{W}{mm^2})$	0.45	0.51	0.67	0.77	0.89
Área do circuito de referência (mm^2**)**	159	86	45	24	13
Potência do circuito de referência (W**)**	69.49	43.30	29.88	18.36	11.25
Porcentagem de dark-silicon (%)	-	12.26%	33.85%	42.40%	49.86%
RTX 2060 (Turing)					
Número de UCs	30	55	110	205	395
Tecnologia (nm)	90	65	45	32	22
Frequência (MHz)	1365	1365	1365	1365	1365
Área do chip(mm^2)	5498	5440	5422	5417	5456
Densidade de Potência $(\frac{W}{mm^2})$	0.28	0.33	0.44	0.51	0.60
Área do circuito de referência (mm^2**)**	182	98	49	26	14
Potência do circuito de referência (W**)**	49.42	31.58	21.40	13.23	8.23
Porcentagem de dark-silicon (%)	-	14.75%	36.67%	44.84%	53.50%
TITAN (Kepler)					
Número de UCs	14	26	53	99	195
Tecnologia (nm)	90	65	45	32	22
Frequência (MHz)	837	837	837	837	837
Área do chip(mm^2)	3614	3651	3604	3622	3623
Densidade de Potência $(\frac{W}{mm^2})$	0.24	0.27	0.37	0.42	0.51
Área do circuito de referência (mm^2**)**	256	140	68	37	19
Potência do circuito de referência (W**)**	57.80	36.62	24.65	15.16	9.41
Porcentagem de dark-silicon (%)	-	12.95%	36.47%	44.10%	54.04%

6.3 Preditor de performance de GPU validado na exploração de espaço de projeto ciente de Dark Silicon

Originalmente, o módulo de DS-DSE da ferramenta MultiExplorer explorava novas alternativas de projeto utilizando um banco de dados com cores de CPUs. Neste trabalho, o banco de dados foi estendido com UCs de GPUs, o qual é ilustrado na Tabela 6.2. Além disso, código do preditor de desempenho foi inserido ao MultiExplorer, para ser utilizado durante a execução dos

algoritmos de exploração do espaço de projetos, para predizer o desempenho de todas as alternativas de projetos sendo avaliadas.

Tabela 6.2: Banco de Dados de CPUs e GPUs

	Cores	Tecnologia (nn)	Frequência (MHz)	Potência do core (W)	Área do core (mm^2)
CPU	1 - Smithfield	90	2800	8.9	111.12
	2 - Quark x1000	32	400	1.06	11.32
	3 - ARM A53	22	1600	5.5	6.82
	4 - ARM A57	22	2000	12.13	6.79
	5 - Atom Silvermont	22	500	2.51	6.15
GPU	1 - GTX480 (Fermi)	45	700	10.70	36.03
	2 - QV100 (Volta)	22	1132	7.77	17.48
	3 - TITAN V (Volta)	22	1200	13.86	19.12
	4 - TITAN X (Pascal)	22	1417	18.36	23.89
	5 - RTX 2060 (Turing)	22	1365	8.23	13.80
	6 - TITAN (Kepler)	32	837	15.16	36.51

A fim de explorar alternativas arquiteturais heterogêneas, foram realizados dois experimentos: o primeiro considerando como unidade computacional original a GTX480 6.3 com 15 cores, e o segundo considerando como unidade computacional original a Titanx com 28 cores 6.4. Cada um desses experimentos tem como limitantes a área e a densidade de potência, ordenados pela performance que eles são capazes de alcançar.

Para esse experimento, foi utilizado o algoritmo de força-bruta, ao invés do algoritmo genético NSGAII, uma vez que o espaço de projeto é pequeno o suficiente para o uso de um método exato. As Tabelas 6.3 e 6.4 apresentam os resultados encontrados pelo algoritmo de força-bruta. É importante ressaltar que as restrições do experimento sobre a GTX480 são: área máxima de $545.85mm^2$ e densidade de potência máxima de $0.31W/mm^2$. As restrições para o experimento sobre a TITANX são: área máxima de $672mm^2$ e densidade de potência de $0.77W/mm^2$, que são os valores respectivos a simulação da plataforma original GTX480 e TITANX com as configurações definidas no *datasheet* pela fabricante NVIDIA.

As Tabelas 6.3 e 6.4 mostram as quatro soluções alternativas mais promissoras para os projetos originais, as quais obedecem as restrições impostas e estão ordenadas pelos seus desempenhos em MIPS. As abreviaturas utilizadas para os parâmetros que descrevem as soluções alternativas são as seguintes:

- Id: Identificador de cada solução. As soluções são ordenadas pela performance atingida.
- N_{Or}: O número de unidades de computação original.O algoritmo foi configurado para ter pelo menos uma unidade de computação original na plataforma resultante.

- N_{IP}: O número de unidades de computação alternativas;
- T_{IP}: Tipo de unidade de computação alternativa (Tabela 6.2);
- AT: Área da plataforma;
- DP: Densidade de potência da plataforma;
- Y_{Pred}:Performance predita pelo modelo MLP(com respeito as métricas detalhadas na Seção 5.1) ;

Tabela 6.3: Exploração de espaço de projeto começando com GTX480 15 cores usando a aplicação AsyncAPI. Restrições: $AT \leq 545.85mm^2$ and $DP \leq 0.31W/mm^2$.

Id	N_O	N_{IP}	T_{IP}	$AT(mm^2)$	$DP(\frac{W}{mm^2})$	$Y_{Pred}(MIPS)$
S1	14	2	QV100	539.39	0.31	269974.58
S2	13	2	QV100	503.36	0.31	261758.65
S3	12	2	QV100	467.32	0.31	252898.28
S4	7	8	GTX480	540.47	0.29	248142.27

Os resultados da Tabela 6.3 podem ser interpretados da seguinte forma: a plataforma original GTX480 possuía 15 UCs ou cores originais, e área de dark-silicon estimada em 32.56%. A solução mais promissora para esse projeto seria utilizar 14 UCs originais da GTX480 e 2 UCs da QV100, resultando em um chip com área de 539.386mm^2 e densidade de potência 0.306 W/mm^2 e desempenho de 269974.58 (em MIPS).

Tabela 6.4: Exploração de espaço de projeto começando com Titanx 28 cores usando a aplicação AsyncAPI. $AT \leq 672mm^2$ e $DP \leq 0.77W/mm^2$.

Id	N_O	N_{IP}	T_{IP}	$AT(mm^2)$	$DP(\frac{W}{mm^2})$	$Y_{Pred}(MIPS)$
S1	12	22	QV100	671.07	0.58	1090227
S2	11	23	QV100	664.66	0.57	1087059
S3	10	24	TITANV	658.25	0.56	1081964
S4	15	16	QV100	664.18	0.75	1080762

Com a Tabela 6.4 pode se dizer que dada a plataforma original TITANX que contém 28 cores e porcentagem de dark-silicon de aproximadamente 43.19%. É viável propor como alternativa 12 cores do tipo TITANX e 22 cores do tipo QV100, com área aproximada em 671.06 mm^2 e DP de 0.583W/mm^2 e performance de 1090227.

O método de predição de performance de plataformas heterogêneas usando unidades computacionais de GPUs diferentes é a seguinte: nos aplicamos o preditor para cada grupo de GPUs separadamente, e a performance total é dada pela soma dos desempenhos isolados.

6.4 Considerações Finais

O algoritmos de DSE buscam soluções avaliando um conjunto de possibilidades, que crescem exponencialmente com o número de componentes disponíveis. Uma vez que uma nova plataforma alternativa é proposta, é necessário ser avaliada de acordo com pelo menos três aspectos: a área deve obedecer a restrição prevista, que vem de caracterizações da plataforma original; densidade de potência é também um limitante, para que a plataforma resultante seja livre de *dark-silicon*. Finalmente, as soluções que se encaixam nas restrições anteriores podem ser ranqueadas de acordo com os desempenhos que podem atingir. Nesse contexto, a realização de simulações para avaliar a performance de soluções alternativas seria inviável.

CAPÍTULO

7

Considerações Finais

O uso crescente de GPUs para computação de propósito geral motiva fabricantes de processadores a incluir essas unidades de processamento massivamente paralelas em projetos de processadores heterogêneos. Com isso, existe uma intensa necessidade de explorar, em tempo de projeto, as melhores soluções arquiteturais para aplicações de propósito geral para que possam explorar esse tipo de paralelismo adequadamente, gerando benefícios tanto em termos de desempenho quanto na eficiência energética do projeto.

Entretanto, avaliar desempenho de sistemas heterogêneos baseados em GPU, com base em simulação, é uma tarefa computacionalmente custosa e requer um árduo processo de modelagem. Além disso, os parâmetros que definem uma arquitetura de GPU e as métricas adequadas para avaliação de tais sistemas são muito distintas daqueles utilizados com frequência nas CPUs. Isso posto, esse trabalho propõe o desenvolvimento de preditores de desempenho de sistemas heterogêneos contendo GPUs distintas, com base em modelos de aprendizado de máquina, os quais serão acoplados a um *framework* de exploração do espaço de projetos ciente de *dark silicon*.

Foram apresentados os resultados obtidos a partir dos preditores de desempenho de sistemas baseados em GPUs homogêneas. Para tal, foram realizadas diversas simulações a fim de extrair atributos para montagem do conjunto de treinamento. Além disso, foi realizada uma avaliação dos preditores desenvolvidos, usando métricas erro absoluto percentual médio (MAPE) e validação cruzada, obtendo resultados promissores pelo uso do modelo de *MLP* com erros gerais de $6,3\%$ e erro de $11,6\%$, quando agrupado pelo atributo UCs. O melhor modelo baseado em SVR alcançou erro geral médio de 10% com kernel RBF.

Após a validação dos preditores, a etapa seguinte foi selecionar qual seria utilizado na ferramenta MultiExplorer. Sabe-se que dois parâmetros de qualidade são essenciais para um preditor de desempenho durante o processo de exploração do espaço de projetos: a acurácia e o tempo utilizado para fazer a estimativa. Portanto, os modelos foram também avaliados sob o ponto de vista da vazão, ou seja, da quantidade de predições por unidade de tempo (segundos). Constatou-se que o regressor linear é o mais rápido, porém muito impreciso. Verificou-se também que o regressor baseado em árvores de decisão é o segundo de maior vazão e o mais preciso, mas não tem boa generalização quando escalamos as UCs. Logo, o modelo MLP é considerado o mais promissor para integração à ferramenta MultiExplorer.

Como trabalhos futuros, vale ressaltar a extensão para níveis de heterogeneidade que explorem com mais profundidade os sistemas com CPUs e GPUs no mesmo sistema. Os principais desafios dessa abordagem seria encontrar simuladores e *benchmarks* robustos com esse nível de heterogeneidade. Além disso, há a necessidade de atualização do McPat [35], já que ele não tem suporte, em seu código original, a arquiteturas de GPU, e também não suporta processos tecnológicos mais recent es utilizados na indústria (abaixo de $22nn$).

Este trabalho de mestrado resultou nas seguintes publicações:

- Apresentação e Best Paper no Workshop de Computação Heterogênea do XIX Simpósio de Sistemas Computacionais de Alto Desempenho (WSCAD-2018) - resultados preliminares;
- Anais do XX Simpósio em Sistemas Computacionais de Alto Desempenho (WSCAD-2019)[28] - resultados preliminares;
- Anais do XXI Simpósio em Sistemas Computacionais de Alto Desempenho (WSCAD-2020)[75] - resultados finais resumidos para o escopo do Simpósio;
- Periódico *"Concurrency and Computation - Practice and Experience"* [29] - resultados finais do trabalho de mestrado.

Os códigos produzidos durante este trabalho, bem como toda a infraestrutura do MultiExplorer, estão disponíveis em https://github.com/lscad-facom-ufms/multiexplorer/tree/GPGPUsim

CAPÍTULO

8

Referências Bibliográficas

[1] Cristobal Navarro, Nancy Hitschfeld, and Luis Mateu. A survey on parallel computing and its applications in data-parallel problems using gpu architectures. *Communications in Computational Physics*, 15:285–329, 09 2013. Citado nas páginas xv e 7.

[2] Nvidia. Fermi compute architecture whitepaper. Citado nas páginas xv, 8, e 9.

[3] M. T. d. Santos, R. Sonohata, C. Krebs, D. Segovia, R. Santos, and L. Duenha. Performance models for heterogeneous systems applied to the dark silicon-aware design space exploration. In *2019 31st International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 9–16, 2019. Citado nas páginas xv, 11, e 12.

[4] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2018. Citado nas páginas xv, 13, 14, 15, e 16.

[5] J. Gama, K. Faceli, A.C. Lorena, and A.C.P.L.F. De Carvalho. *Inteligencia artificial: uma abordagem de aprendizado de maquina*. Grupo Gen - LTC, 2011. Citado nas páginas xvii, 15, 19, e 21.

[6] Gordon E Moore. Cramming more components onto integrated circuits. *Proceedings of the IEEE*, 86(1):82–85, 1998. Citado nas páginas 1 e 10.

[7] Robert H Dennard, Jin Cai, and Arvind Kumar. A perspective on today's scaling challenges and possible future directions. *Solid-State Electronics*, 51(4):518–525, 2007. Citado nas páginas 1 e 46.

[8] Tony Santos, Ana Silva, Liana Duenha, Ricardo Santos, Edward Moreno, and Rodolfo Azevedo. On the dark silicon automatic evaluation on multicore processors. In *2016 28th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 166–173. IEEE, 2016. Citado nas páginas 1, 3, 10, e 46.

[9] Stephen W Keckler, William J Dally, Brucek Khailany, Michael Garland, and David Glasco. Gpus and the future of parallel computing. *IEEE micro*, 31(5):7–17, 2011. Citado na página 1.

[10] John D Owens, Mike Houston, David Luebke, Simon Green, John E Stone, and James C Phillips. Gpu computing. *Proceedings of the IEEE*, 96(5):879–899, 2008. Citado na página 1.

[11] Design Guide. Cuda c programming guide. *NVIDIA, July*, 2013. Citado na página 1.

[12] Aaftab Munshi, Benedict Gaster, Timothy G Mattson, and Dan Ginsburg. *OpenCL programming guide*. Pearson Education, 2011. Citado na página 1.

[13] Ricardo Santos, Liana Duenha, Ana Caroline Silva, Matheus Sousa, Luiz Augusto Tedesco, João Carlos Melgarejo, Tony Santos, Rodolfo Azevedo, and Edward Moreno. Dark-silicon aware design space exploration. *Journal of Parallel and Distributed Computing*, 2017. Citado nas páginas 2, 3, e 11.

[14] M. J. Flynn. Very high-speed computing systems. *Proceedings of the IEEE*, 54(12):1901–1909, 1966. Citado nas páginas 5 e 6.

[15] Ashu Rege. An introduction to modern gpu architecture. *En ligne*, 2008. Citado na página 6.

[16] John Nickolls and William J Dally. The gpu computing era. *IEEE micro*, 30(2):56–69, 2010. Citado na página 8.

[17] Sylvain Collange, Marc Daumas, David Defour, and David Parello. Barra: A parallel functional simulator for gpgpu. In *2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pages 351–360. IEEE, 2010. Citado nas páginas 9 e 27.

[18] Rafael Ubal, Byunghyun Jang, Perhaad Mistry, Dana Schaa, and David Kaeli. Multi2Sim: a simulation framework for CPU-GPU computing. In *Proceedings of the 21st international conference on Parallel architectures*

and compilation techniques, pages 335–344. ACM, 2012. Citado na página 9.

[19] Ali Bakhoda, George L Yuan, Wilson WL Fung, Henry Wong, and Tor M Aamodt. Analyzing cuda workloads using a detailed gpu simulator. In *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, pages 163–174. IEEE, 2009. Citado nas páginas 9, 11, 25, 30, e 45.

[20] Wilson WL Fung, Ivan Sham, George Yuan, and Tor M Aamodt. Dynamic warp formation and scheduling for efficient gpu control flow. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 407–420. IEEE Computer Society, 2007. Citado nas páginas 9, 11, e 45.

[21] Tayler H. Hetherington Tor M. Aamodt, Wilson W.L. Fung. GPGPU-Sim 3.x Manual. http://gpgpu-sim.org/manual/index.php/Main_Page. [Online; accessed 6-Fev-2021]. Citado na página 9.

[22] Jingwen Leng, Tayler Hetherington, Ahmed ElTantawy, Syed Gilani, Nam Sung Kim, Tor M Aamodt, and Vijay Janapa Reddi. Gpuwattch: enabling energy optimizations in gpgpus. In *ACM SIGARCH Computer Architecture News*, volume 41, pages 487–498. ACM, 2013. Citado na página 10.

[23] Hadi Esmaeilzadeh, Emily Blem, Renee St Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*, pages 365–376. IEEE, 2011. Citado na página 10.

[24] Robert H Dennard, Fritz H Gaensslen, Hwa-Nien Yu, V Leo Rideout, Ernest Bassous, and Andre R LeBlanc. Design of ion-implanted mosfet's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, 1974. Citado na página 10.

[25] Eunsuk Kang, Ethan Jackson, and Wolfram Schulte. An approach for effective design space exploration. In *Monterey Workshop*, pages 33–54. Springer, 2010. Citado na página 11.

[26] Ricardo Santos, Liana Duenha, Ana Caroline Silva, Matheus Sousa, Luiz Augusto Tedesco, João Carlos Melgarejo, Tony Santos, Rodolfo Azevedo, and Edward Moreno. Dark-silicon aware design space exploration. *Journal of Parallel and Distributed Computing*, 120:295–306, 2018. Citado nas páginas 11 e 33.

[27] M.T. Santos, R. Sonohata, C. Krebs, D. Segovia, R.R. Santos, and L. Duenha. Performance models for heterogeneous systems applied to the dark silicon-aware design space exploration. *Proccedings of the 31st International Symposium on Computer Architecture and High Performance Computing*, 2019. Citado na página 11.

[28] Ricardo Santos, Rhayssa Sonohata, Casio Krebs, Daniela Catelan, Liana Duenha, Diego Segovia, and Mateus Tostes Santos. Exploraçao do projeto de sistemas baseados em gpu ciente de dark silicon. In *Anais do XX Simpósio em Sistemas Computacionais de Alto Desempenho*, pages 358–369. SBC, 2019. Citado nas páginas 11 e 54.

[29] Rhayssa Sonohata, Danillo Christi A Arigoni, Eraldo Rezende Fernandes, Ricardo Ribeiro dos Santos, and Liana Dessandre Duenha. Performance predictors for graphics processing units applied to dark-silicon-aware design space exploration. *Concurrency and Computation: Practice and Experience*, page e6877. Citado nas páginas 11 e 54.

[30] Trevor E Carlson, Wim Heirman, and Lieven Eeckhout. Sniper: exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 52. ACM, 2011. Citado na página 11.

[31] Christian Bienia, Sanjeev Kumar, Jaswinder Singh, and Kai Li. The parsec benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 72–81. ACM, 2008. Citado na página 11.

[32] Steven Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Singh, and Anoop Gupta. The splash-2 programs: Characterization and methodological considerations. *ACM SIGARCH Computer Architecture News*, 23:24–36, 1995. Citado na página 11.

[33] NVIDIA Corporation. CUDA toolkit documentation. https://docs.nvidia.com/cuda/cuda-samples/index.html. [Online; accessed 6-Fev-2021]. Citado nas páginas 11, 34, 35, e 45.

[34] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheaffer, Sang-Ha Lee, and Kevin Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *2009 IEEE International Symposium on Workload Characterization (IISWC)*, pages 44–54, 2009. Citado nas páginas 11, 34, e 45.

[35] Sheng Li, Jung Ahn, Richard Strong, Jay Brockman, Dean Tullsen, and Norman Jouppi. McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 469–480. IEEE, 2009. Citado nas páginas 11, 45, e 54.

[36] Moritz Hardt and Benjamin Recht. Patterns, predictions, and actions: A story about machine learning, 2021. Citado nas páginas 13, 15, e 16.

[37] Stuart Jonathan Russell and Peter Norvig. *Inteligência artificial*. Elsevier, 3 edition, 2013. Citado na página 14.

[38] A.C.P.L.F. Carvalho, J. Gama, K. Faceli, and A.C. Lorena. *Inteligência artificial: uma abordagem de aprendizado de máquina*. Grupo GEN-LTC, Rio de Janeiro, 2011. Citado nas páginas 14 e 21.

[39] George F Luger. *Inteligência artificial*. Pearson Education do Brasil, 6 edition, 2013. Citado na página 14.

[40] Xiaojin Zhu. Semi-supervised learning tutorial. In *International Conference on Machine Learning (ICML)*, pages 1–135, 2007. Citado na página 14.

[41] M Ikonomakis, Sotiris Kotsiantis, and V Tampakas. Text classification using machine learning techniques. *WSEAS transactions on computers*, 4(8):966–974, 2005. Citado na página 15.

[42] Laurens Van Der Maaten, Eric Postma, and Jaap Van den Herik. Dimensionality reduction: a comparative. *J Mach Learn Res*, 10(66-71):13, 2009. Citado na página 15.

[43] Marko Sarstedt, Erik Mooi, et al. A concise guide to market research. *The Process, Data, and*, 12, 2014. Citado na página 16.

[44] Sanford Weisberg. *Applied linear regression*, volume 528. John Wiley & Sons, 2005. Citado na página 17.

[45] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001. Citado na página 18.

[46] Adele Cutler, D Richard Cutler, and John R Stevens. Random forests. In *Ensemble machine learning*, pages 157–175. Springer, 2012. Citado na página 18.

[47] Alex Smola and SVN Vishwanathan. Introduction to machine learning. *Cambridge University, UK*, 32(34):2008, 2008. Citado na página 19.

[48] Stuart Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Prentice Hall, 2009. Citado na página 19.

[49] Mikhail Kanevski, Aleksey Pozdnukhov, Stéphane Canu, Michel Maignan, PM Wong, and SAR Shibli. Support vector machines for classification and mapping of reservoir data. In *Soft Computing for Reservoir Characterization and Modeling*, pages 531–558. Springer, 2002. Citado na página 20.

[50] Arti Patle and Deepak Singh Chouhan. Svm kernel functions for classification. In *2013 International Conference on Advances in Technology and Engineering (ICATE)*, pages 1–9. IEEE, 2013. Citado na página 20.

[51] Leonardo Noriega. Multilayer perceptron tutorial. *School of Computing. Staffordshire University*, 2005. Citado na página 22.

[52] Daniel Mesafint Belete and Manjaiah D Huchaiah. Grid search in hyperparameter optimization of machine learning models for prediction of hiv/aids test results. *International Journal of Computers and Applications*, pages 1–12, 2021. Citado na página 22.

[53] Petro Liashchynskyi and Pavlo Liashchynskyi. Grid search, random search, genetic algorithm: A big comparison for nas. *arXiv preprint arXiv:1912.06059*, 2019. Citado nas páginas 22 e 38.

[54] Daniel Berrar. Cross-validation., 2019. Citado nas páginas 22 e 23.

[55] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics. Springer-Verlag New York, 2 edition, 2009. Citado na página 23.

[56] Pang-Ning Tan, Michael Steinbach, Anuj Karpatne, and Vipin Kumar. *Introduction to Data Mining (2nd Edition)*. Pearson, 2nd edition, 2018. Citado na página 23.

[57] Chuanjun Zhang and Frank Vahid. Cache configuration exploration on prototyping platforms. In *Proceedings of the 14th IEEE International Workshop on Rapid Systems Prototyping*, pages 164–170. IEEE, 2003. Citado na página 26.

[58] Y. Zhang and J. D. Owens. A quantitative performance analysis model for gpu architectures. In *2011 IEEE 17th International Symposium on High Performance Computer Architecture*, pages 382–393, 2011. Citado nas páginas 26 e 30.

[59] Sara S Baghsorkhi, Matthieu Delahaye, Sanjay J Patel, William D Gropp, and Wen-mei W Hwu. An adaptive performance modeling tool for gpu architectures. In *Proceedings of the 15th ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 105–114, 2010. Citado nas páginas 27 e 30.

[60] Sunpyo Hong and Hyesoon Kim. An analytical model for a gpu architecture with memory-level and thread-level parallelism awareness. In *Proceedings of the 36th annual international symposium on Computer architecture*, pages 152–163, 2009. Citado nas páginas 27 e 30.

[61] Newsha Ardalani, Clint Lestourgeon, Karthikeyan Sankaralingam, and Xiaojin Zhu. Cross-architecture performance prediction (xapp) using cpu code to predict gpu performance. In *Proceedings of the 48th International Symposium on Microarchitecture*, MICRO-48, page 725–737, New York, NY, USA, 2015. Association for Computing Machinery. Citado nas páginas 28 e 30.

[62] Gene Wu, Joseph L Greathouse, Alexander Lyashevsky, Nuwan Jayasena, and Derek Chiou. Gpgpu performance and power estimation using machine learning. In *2015 IEEE 21st international symposium on high performance computer architecture (HPCA)*, pages 564–576. IEEE, 2015. Citado nas páginas 28 e 30.

[63] Ali Karami, Sayyed Ali Mirsoleimani, and Farshad Khunjush. A statistical performance prediction model for opencl kernels on nvidia gpus. In *The 17th CSI International Symposium on Computer Architecture & Digital Systems (CADS 2013)*, pages 15–22. IEEE, 2013. Citado nas páginas 28 e 30.

[64] Wenhao Jia, Kelly A Shaw, and Margaret Martonosi. Stargazer: Automated regression-based gpu design space exploration. In *2012 IEEE International Symposium on Performance Analysis of Systems & Software*, pages 2–13. IEEE, 2012. Citado nas páginas 29 e 30.

[65] Wenhao Jia, Kelly A Shaw, and Margaret Martonosi. Stargazer: Automated regression-based gpu design space exploration. In *Performance Analysis of Systems and Software (ISPASS), 2012 IEEE International Symposium on*, pages 2–13. IEEE, 2012. Citado na página 30.

[66] Peter N Glaskowsky. Nvidia's fermi: the first complete gpu computing architecture. *White paper*, 18, 2009. Citado na página 33.

[67] Tesla NVIDIA. V100 gpu architecture http://images. nvidia. com/content/volta-architecture/pdf/volta-architecture-whitepaper-v1. 0. pdf, 2017. Citado na página 33.

[68] N NVIDIA. Gp100 pascal whitepaper. 2016. Citado na página 33.

[69] C NVIDIA. Nvidia: Nvidias next generation cuda compute architecture: Kepler gk110 (whitepaper). 2012. Citado na página 33.

[70] IS NVIDIA. Nvidia turing gpu architecture whitepaper. 2018. Citado na página 33.

[71] Sheng Li, Jung Ahn, Richard Strong, Jay Brockman, Dean Tullsen, and Norman Jouppi. The McPAT framework for multicore and manycore architectures: Simultaneously modeling power, area, and timing. *ACM Transactions on Architecture and Code Optimization (TACO)*, 10(1):5, 2013. Citado na página 33.

[72] Wei Huang, Shougata Ghosh, Sivakumar Velusamy, Karthik Sankaranarayanan, Kevin Skadron, and Mircea R Stan. Hotspot: A compact thermal modeling methodology for early-stage vlsi design. *IEEE Transactions on very large scale integration (VLSI) systems*, 14(5):501–513, 2006. Citado na página 34.

[73] R.S. Witte and J.S. Witte. *Statistics*. Wiley, 2013. Citado na página 37.

[74] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011. Citado na página 38.

[75] Liana Duenha, Rhayssa Sonohata, Danillo Arigoni, and Ricardo Santos. Preditor de desempenho de gpus aplicado à exploração do espaço de projetos ciente de dark silicon. In *Anais do XXI Simpósio em Sistemas Computacionais de Alto Desempenho*, pages 299–310. SBC, 2020. Citado na página 54.