Predição de Desempenho de Aplicações CUDA utilizando Aprendizado de Máquina e Características de Pré-Execução

Luan Siqueira[1], Erick Damasceno[1], Thiago J. S. Rodrigues[1], Fellipe Queiroz[1],

Marcos Amaris[1]

[1]Universidade Federal do Pará
Faculdade de Engenharia de Computação, Tucuruí - Pará

{erick.silva, fellipe.queiroz, luan.siqueira}@tucurui.ufpa.br,
amaris@ufpa.br, tj.vinci97@gmail.com

Resumo. *Com a evolução das unidades de processamento gráfico (GPU), as aplicações de computação paralela estão se tornando cada vez mais complexas. Predizer o desempenho dessas aplicações ajuda desenvolvedores a otimizar seus algoritmos escalonadores na distribuição de seus trabalhos. Neste trabalho, foram desenvolvidos e avaliados modelos de aprendizado de máquina para predizer o desempenho de aplicações CUDA utilizando características de pré-execução. Foram comparados os modelos Ridge Regression, Random Forest e Decision Tree em nove aplicações CUDA, utilizando a métrica MAPE. Os resultados mostram que o Decision Tree obteve os menores valores de MAPE, enquanto o Random Forest apresentou um desempenho consistente. Já o Ridge Regression teve desempenho variável devido à sua limitação em lidar com multicolinearidade. O estudo enfatiza a importância considerar as características específicas da aplicação e da GPU ao fazer predições de desempenho.*
Palavras-chave: *Desempenho, Aprendizado de máquina, GPU, CUDA, Computação Paralela.*

Abstract. *With the evolution of Graphics Processing Units (GPUs), parallel computing applications are becoming increasingly complex. Predicting the performance of these applications helps developers optimize their scheduling algorithms for workload distribution. In this work, machine learning models were developed and evaluated to predict the performance of CUDA applications using pre-execution features. The Ridge Regression, Random Forest, and Decision Tree models were compared across nine CUDA applications using the MAPE metric. The results show that Decision Tree achieved the lowest MAPE values, while Random Forest demonstrated consistent performance. Ridge Regression had variable performance due to its limitation in handling multicollinearity. The study emphasizes the importance of considering the specific characteristics of the application and GPU when making performance predictions.*
Keywords: *Performance, Machine Learning, GPU, CUDA.*

1. INTRODUÇÃO

A crescente demanda por aplicações de alta performance em áreas como inteligência artificial, simulações científicas e análise de *big data* tem impulsionado o desenvolvimento

e a otimização de arquiteturas de unidades de processamento gráfico (GPU, do inglês *Graphics processing unit*). As GPUs, têm se destacado por sua capacidade de paralelismo massivo, oferecendo uma alternativa eficiente para a execução de tarefas complexas [NVIDIA 2024].

Para usufruir desse poder computacional das GPUs, a Nvidia introduziu em 2006 a CUDA, uma plataforma que permite aos desenvolvedores tirar proveito da capacidade massiva de paralelismo das GPUs. A capacidade de predizer o desempenho dessas aplicações continua sendo um desafio devido à variabilidade dos fatores que influenciam a execução, como a arquitetura da GPU e as características específicas das aplicações [NVIDIA and CUDA 2015].

Com isso, este trabalho investiga o uso de técnicas de aprendizado de máquina para predizer o desempenho de aplicações CUDA da NVIDIA. Foram comparados os modelos *ridge regression*, *decision tree* e *random forest*. As previsões baseiam-se em características extraídas antes da execução das aplicações (características de pré-execução), juntamente com informações sobre a arquitetura das GPUs utilizadas.

2. FUNDAMENTOS

2.1. Arquitetura de GPU NVIDIA e CUDA

A arquitetura de GPU da NVIDIA é composta por núcleos de processamento organizados em multiprocessadores de streaming, permitindo a execução paralela de milhares de threads e a realização eficiente de tarefas complexas. Componentes como registradores, unidades de leitura e armazenamento, cache e velocidade de clock influenciam diretamente o desempenho das aplicações. CUDA é uma API de computação paralela da NVIDIA que permite aos desenvolvedores escrever programas que utilizam a capacidade de processamento paralelo das GPUs, organizando threads em blocos e grades para alto nível de paralelismo e eficiência [NVIDIA and CUDA 2015].

2.2. Características de Pré-execução CUDA

Para predizer o desempenho das aplicações CUDA, extrair características do código-fonte antes da execução completa é essencial. O processo depende de ferramentas como o *Nvidia Cuda Compiler* (NVCC) e o CUDA *Occupancy Calculator*. O NVCC é um compilador que fornece informações detalhadas sobre o uso de registradores, memória e configuração de *threads*, convertendo o código CUDA em instruções executáveis pela GPU. O cálculo de nível de ocupação da aplicação é realizado pelo CUDA *Occupancy Calculator* que permite ajustar os parâmetros do *kernel* para maximizar a ocupação da GPU e identificar gargalos de desempenho [NVIDIA and CUDA 2015].

2.3. Aprendizado de máquina

O aprendizado de máquina oferece uma abordagem promissora para prever o desempenho de aplicações complexas. Técnicas como *Ridge Regression*, *Decision Tree* e *Random Forest* são utilizadas para criar modelos preditivos que estimam a saída com base em entradas específicas. Esses modelos são treinados com dados de pré-execução, capturando as relações entre as características do código e o desempenho esperado [Susto et al. 2014].

Ridge Regression é uma extensão da regressão linear que incorpora regularização para lidar com multicolinearidade e melhorar a robustez dos coeficientes estimados [Hastie et al. 2009]. *Decision Tree* é uma técnica não paramétrica que cria um modelo preditivo dividindo a população em ramos baseados em regras de decisão simples [yan Song 2015]. *Random Forest* combina várias árvores de decisão para reduzir a variância do modelo e melhorar a precisão das previsões [Breiman 2001].

2.4. Métrica de desempenho (MAPE)

O MAPE (*Mean Absolute Percentage Error*) é uma métrica amplamente utilizada para avaliar a precisão de previsões devido à sua facilidade de interpretação e independência de escala. O MAPE calcula a média das diferenças absolutas entre os valores reais e previstos, expressas como uma porcentagem dos valores reais. Isso permite uma interpretação intuitiva do erro percentual absoluto [Kim and Kim 2020].

Como mostrado na equação (1), o MAPE é dado por:

$$MAPE = \frac{1}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right| \times 100 \tag{1}$$

Onde n é o número de observações, y_i é o valor real da i-ésima observação e $\hat{y}_i$ é o valor previsto da i-ésima observação.

3. TRABALHOS RELACIONADOS

Estudos anteriores demonstraram a eficácia do aprendizado de máquina na predição de desempenho de aplicações. [Amaris et al. 2016] propuseram um modelo baseado em *Bulk Synchronous Parallel* (BSP) para estimar os tempos de execução de aplicativos CUDA, destacando sua simplicidade e capacidade de lidar com diferentes configurações de GPU. [Mittal and Vetter 2016] discutiram a importância de combinar características específicas das aplicações com técnicas de aprendizado de máquina para melhorar a precisão das predições.

Recentemente, [Zhang et al. 2020] mostraram que redes neurais profundas podem capturar relações complexas entre variáveis, melhorando significativamente a precisão das predições de desempenho. [Wang et al. 2019] desenvolveram um *framework* híbrido que combina aprendizado supervisionado e não supervisionado, enfatizando a importância da seleção adequada de características para construir modelos precisos.

Neste artigo, implementamos um modelo de aprendizado de máquina afim de predizer o tempo de execução do *kernel* em GPUs NVIDIA. Embora alguns trabalhos utilizam métodos de aprendizado de máquina e análise estática, muitos extraem os dados de informação através de *profiling* no momento da execução da aplicação e, neste trabalho, a extração de dados é realizada no momento da compilação.

4. METODOLOGIA

4.1. Algoritmos de teste

Foram utilizados nove kernels CUDA, incluindo quatro para multiplicação de matriz, dois para soma de matriz, produto escalar, adição de vetor e problema de submatriz máxima.

As multiplicações de matriz abrangem estratégias com acessos não coalescidos e coalescidos à memória global, e acessos não coalescidos e coalescidos à memória compartilhada. As somas de matriz incluem algoritmos com acessos não coalescidos e coalescidos à memória global. A adição de vetor realiza cálculos em paralelo em GPUs, enquanto o produto escalar calcula a multiplicação de elementos de dois vetores. O problema de submatriz máxima otimiza para encontrar a submatriz contígua com o maior número de elementos combinados.

4.2. *Hardware* de GPUs

Utilizou-se as GPUs NVIDIA de diferentes arquiteturas, como Kepler (Compute Capability 3.X) e Maxwell (Compute Capability 5.X). As especificações detalhadas de cada GPU listadas na Tabela 1, incluindo versão de computação, quantidade de memória, largura de banda de memória, tamanho da cache L2 e número de núcleos por SM (*Streaming Multiprocessor*). Essas especificações são essenciais para entender o impacto das características de *hardware* no desempenho das aplicações.

Modelo	CC/Memória	Largura de banda	L2	Núcleos/SM
GTX-680	3,0/2 GB	192,2 G/s	0,5 MB	1536/8
Tesla-K40	3,5/12 GB	276,5 G/s	1,5 MB	2880/15
Tesla K-20	3,5/4 GB	200 G/s	1 MB	2496/13
Titan	3,5/6 GB	288,4 G/s	1,5 MB	2688/14
Q k5200	3,5/8 GB	192,2 G/s	1 MB	2304/12
Titan X	5,2/12 GB	336,5 G/s	3 MB	3072/24
GTX-970	5,2/4 GB	224,3 G/s	1,75 MB	1664/13
GTX-980	5,2/4 GB	224,3 G/s	2 MB	2048/16

Tabela 1. Especificações hardware GPUs

4.3. Conjunto de dados

A coleta de dados foi realizada compilando aplicações CUDA em diversas arquiteturas de GPU (3.0, 3.5 e 5.2) usando um *script* em *Python*. Diferentes números de blocos de encadeamento CUDA (8^2, 16^2, 32^2) foram considerados para avaliar a paralelização. O CUDA *Occcupancy Calculator* foi usado para analisar a ocupação da GPU, capturando dados como registradores e utilização da memória, armazenados em um arquivo CSV. A análise foi enriquecida com dados adicionais do estudo de [Amaris et al. 2016], resultando em 8459 amostras e doze *features*, garantindo a consistência na junção dos dados.

Características	Descrição
compute_version	Capacidade de computação da arquitetura
registers	Número de registradores
smem	Quantidade de memória compartilhada
cmem	Quantidade de memória constante
num_of_cores	Número de núcleos por GPU
l2	Cache
bandwidth	Largura de banda
theoretical_flops	Ponto flutuante
occupancy	Ocupação do multiprocessador kernel
input_size	Tamanho da entrada
duration	Duração
block_x	Número de *threads* por bloco

Tabela 2. *Features* utilizadas como entrada para a aprendizagem do modelo.

4.4. Pré-processamento dos dados

Inicialmente, foi realizada uma análise de correlação entre as variáveis para identificar relações significativas entre elas. A matriz de correlação, apresentada na Figura 1, mostra a presença de multicolinearidade, indicada por altas correlações entre várias variáveis, como L2 e theoretical_flops (0.91), num_of_cores e bandwidth (0.79), e bandwidth e theoretical_flops (0.85), dificultando a determinação dos efeitos individuais em um modelo de regressão.

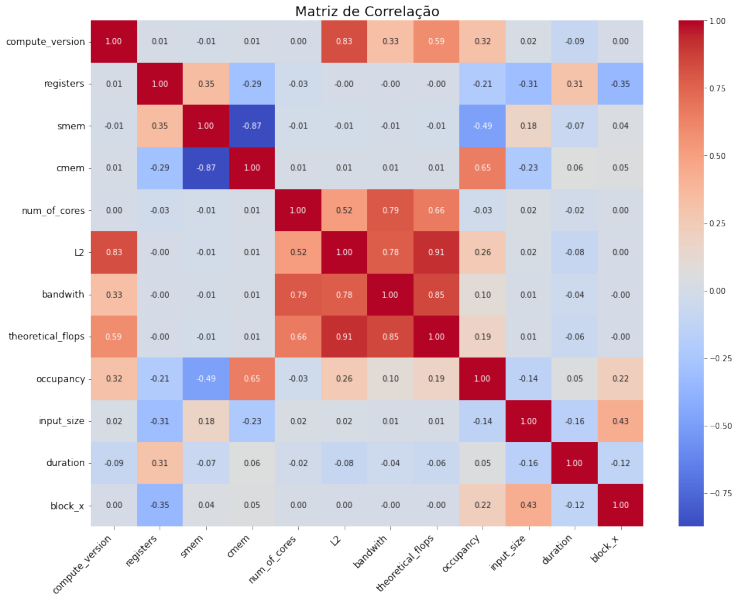

Figura 1. Matriz de correlação

Em seguida, foi analisado a distribuição em frequência da variável *"duration"* para entender melhor como o tempo de execução das aplicações estavam distribuídos. A Figura 2.Esq. mostra que a maioria das durações estava concentrada em valores baixos, com uma cauda longa à direita, indicando uma assimetria à esquerda ou assimetria negativa.

Para facilitar o treinamento dos modelos de aprendizado de máquina e melhorar a performance dos mesmos, foi aplicada uma transformação logarítmica à variável

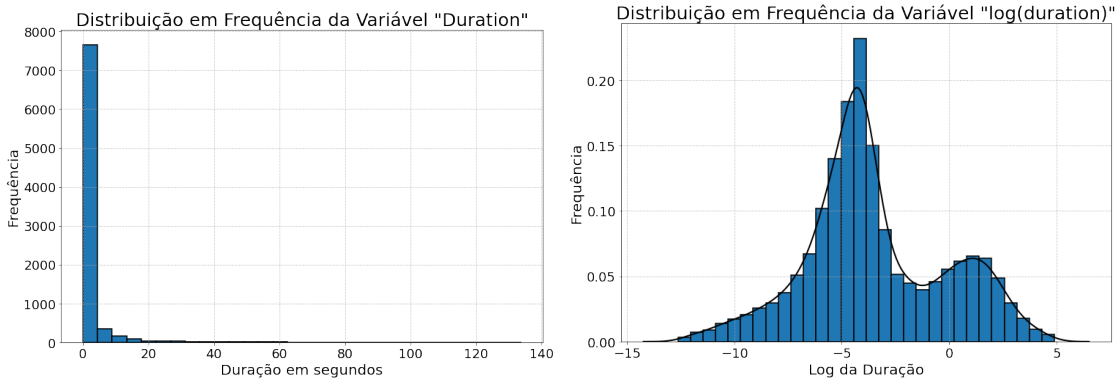

Figura 2. Esq: Distribuição em frequência da variável 'Duration'. Dir: Distribuição em frequência com aplicação logarítmica da variável 'Duration'

"duration". A transformação logarítmica ajuda a reduzir a assimetria dos dados, tornando a distribuição mais simétrica e aproximando-a de uma distribuição normal, o que é desejável para muitas técnicas de aprendizado de máquina. A Figura 2.Dir. mostra a distribuição da variável *"duration"* após a transformação logarítmica.

4.5. Treinamento dos dados

Para o desenvolvimento de cada modelo, foram escolhidos oito aplicações que serviram como conjunto de treinamento para os modelos enquanto uma aplicação foi reservada para o conjunto de teste. Este processo foi repetido para cada umas das nove aplicações.

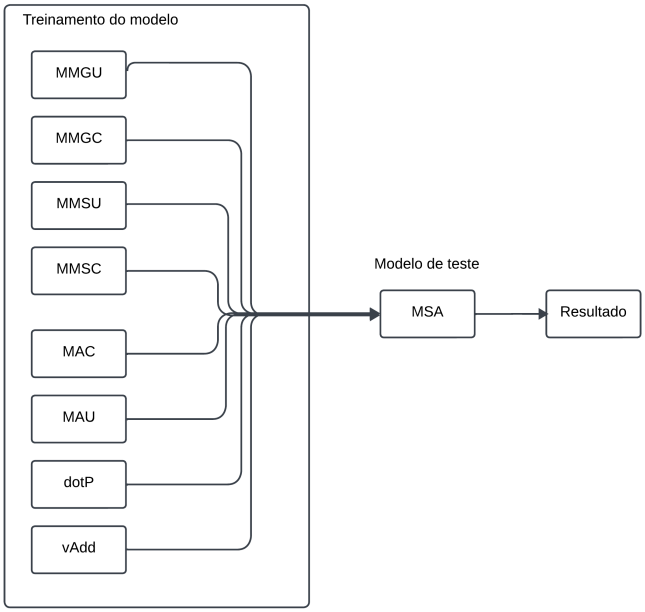

Figura 3. Fluxograma do processo de desenvolvimento e teste dos modelos.

A Figura 3 mostra a seleção das aplicações para treinamento e teste, esse fluxo é repetido para todas as nove aplicações selecionadas para este estudo.

Todos os *scripts* desenvolvidos durante a pesquisa estão disponíveis para uso

público no repositório GitHub para garantir que os resultados deste estudo sejam claros e replicáveis. Ele pode ser acessado através do seguinte link: GitHub - Repositório do Projeto.

5. RESULTADOS E DISCUSSÕES

Neste presente trabalho, utilizamos algoritmos de aprendizado de máquina. Modelos como *Ridge regression*, *Decision tree* e *Random forest* foram treinados e validados utilizando um conjunto diversificado de características de GPUs e dados de pré-execução. A divisão dos dados seguiu a proporção 89/11 para treino e teste, respectivamente. Cada modelo foi avaliado usando métricas de Erro Percentual Absoluto Médio (MAPE).

Aplicação	Random Forest	Ridge Regression	Decision Tree
MMGU	5.96%	0.2%	3.66%
MMGC	0.45%	1.98%	0.94%
MMSU	0.88%	0.86%	0.51%
MMSC	4.66%	1.53%	5.0%
MAC	0.4%	20.58%	0.03%
MAU	0.03%	13.07%	0.04%
dotP	0.17%	5.48%	0.04%
vAdd	0.07%	25.78%	0.03%
MSA	1.03%	13.6%	0.09%

Tabela 3. Comparação de MAPE para Modelos em Diferentes Aplicações

Os resultados da Tabela 3 apresentam como as relações entre variáveis impactam o desempenho dos modelos. O *Random Forests* teve um desempenho consistente, com baixos valores de MAPE em MAU (0.03%) e vAdd (0.07%), devido à sua capacidade de capturar relações não lineares e interações complexas, como a alta correlação positiva entre capacidade de computação e cache L2 (0.83). No entanto, apresentou um desempenho inferior em MMSC (4.66%) e MMGU (5.96%), devido à complexidade introduzida por correlações negativas fortes, como entre memória compartilhada e memória constante (-0.87). O *Ridge Regression* teve um desempenho satisfatório, com o melhor resultado em MMGU (0.2%), menos preciso em vAdd (25.78%) e MAC (20.58%), porque é um modelo linear regularizado que contém dificuldades com correlações fortes e não lineares. A alta correlação positiva entre largura de banda e número de núcleos (0.79) não foi bem modelada, resultando em maiores valores de MAPE. Já o *Decision Tree* mostrou excelente desempenho em MAC (0.03%) e vAdd (0.03%), capturando eficientemente relações não lineares. No entanto, teve desempenho inferior em MMSC (5.0%) e MMGU (3.66%), possivelmente devido ao ajuste inadequado dos parâmetros da árvore.

6. CONCLUSÕES E TRABALHOS FUTUROS

Os resultados mostraram que, embora cada modelo tivesse seus pontos fortes em diferentes cenários, as correlações complexas beneficiaram os modelos *Random Forests* e *Decision Tree*, devido à sua capacidade de capturar relações não lineares e interações entre variáveis. O Random Forest, por exemplo, destacou-se com baixos valores de MAPE em diversas aplicações, enquanto o Decision Tree teve excelente desempenho em outras,

como MAC e vAdd. Por outro lado, o modelo *Ridge Regression* apresentou um desempenho inconsistente, sendo eficaz em algumas aplicações, mas demonstrando altos erros em outras devido à sua limitação em lidar com multicolinearidade das variáveis no conjunto de dados.

Para futuros trabalhos, recomenda-se expandir o conjunto de dados para incluir mais aplicações CUDA e diferentes arquiteturas de GPU, além de prever o consumo de energia das aplicações para aumentar a robustez dos modelos. A extração de mais características do código-fonte utilizando o compilador LLVM pode ser valiosa. Além disso, aplicar técnicas de Processamento de Linguagem Natural (PNL) para analisar códigos de baixo nível como PTX pode melhorar a precisão das predições e fornecer ferramentas eficazes para otimizações no uso de GPUs.

Referências

[Amaris et al. 2016] Amaris, M., de Camargo, R. Y., Dyab, M., Goldman, A., and Trystram, D. (2016). A comparison of gpu execution time prediction using machine learning and analytical modeling. In *2016 IEEE 15th International Symposium on Network Computing and Applications (NCA)*, pages 326–333. IEEE.

[Breiman 2001] Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

[Hastie et al. 2009] Hastie, T., Tibshirani, R., and Friedman, J. H. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics. Springer, 2nd edition.

[Kim and Kim 2020] Kim, S. and Kim, H. (2020). A new metric of absolute percentage error for intermittent demand forecasts. *International Journal of Forecasting*, 36(3):1115–1127.

[Mittal and Vetter 2016] Mittal, S. and Vetter, J. S. (2016). A survey of methods for analyzing and improving gpu energy efficiency. *ACM Computing Surveys (CSUR)*, 49(3):41.

[NVIDIA 2024] NVIDIA (2024). Gpu accelerated solutions for data science. *NVIDIA Newsroom*.

[NVIDIA and CUDA 2015] NVIDIA, C. C. and CUDA, C. (2015). Programming guide, version 7.

[Susto et al. 2014] Susto, G. A., Schirru, A., Pampuri, S., McLoone, S., and Beghi, A. (2014). Machine learning for predictive maintenance: A multiple classifier approach. *IEEE Transactions on Industrial Informatics*, 11(3):812–820.

[Wang et al. 2019] Wang, X., Huang, K., Knoll, A., and Qian, X. (2019). A hybrid framework for fast and accurate gpu performance estimation through source-level analysis and trace-based simulation. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 201–214. IEEE.

[yan Song 2015] yan Song, Y. (2015). Decision tree methods: applications for classification and prediction. *International Journal of Data Analysis Techniques and Strategies*, 7(2):228–243.

[Zhang et al. 2020] Zhang, Y., Wang, S., and Chen, G. (2020). Deep learning-based performance prediction for gpu-accelerated applications. *Journal of Parallel and Distributed Computing*, 144(3):12–23.