Proposta de Metodologia de Avaliação de Desempenho de Hardware e Software por Meio do Modelo Roofline

Vitor de Sá, Bruno Schulze, Mariza Ferro

[1]Laboratório Nacional de Computação Científica – LNCC
Av. Getúlio Vargas, 333 – 25651-075 – Quitandinha, Petrópolis – RJ –Brasil

{vitorsa, schulze, mariza}@lncc.br

Abstract. *In this work, due to the growing demand for computational resources and energy limitations, a performance evaluation methodology is proposed based on theoretical and practical parameters of the Roofline model and using its graph that is part of tools that implement this model. This methodology allows identifying performance patterns in applications, their main computational requirements, factors that limit performance and suggesting the best architecture to run an application. Experiments were developed, focusing on the evaluation of Machine Learning algorithms, where the proposed methodology is evaluated and shown to be effective.*

Resumo. *Neste trabalho, devido a crescente demanda por recursos computacionais e limitações energéticas, é proposta uma metodologia de avaliação de desempenho com base em parâmetros teóricos e práticos do modelo Roofline e usando o gráfico bidimensional que faz parte de ferramentas que implementam este modelo. Essa metodologia permite identificar padrões de desempenho nas aplicações, seus principais requisitos computacionais, fatores que limitam o desempenho e sugerir a melhor arquitetura para executar uma aplicação. Foram desenvolvidos experimentos, com foco na avaliação de algoritmos de Aprendizado de Máquina, onde a metodologia proposta é avaliada se mostrando efetiva.*

1. Introdução

A Computação de Alto Desempenho (HPC) vem auxiliando no avanço da ciência possibilitando que simulações mais complexas e com grandes volumes de dados sejam processadas e analisadas de forma cada vez mais precisas e em um tempo viável de execução. Um dos domínios que vem alcançando resultados surpreendentes devido a essa convergência de desempenho computacional e grande volume de dados disponíveis é o Aprendizado de Máquina (AM), que é cada vez mais utilizada em ambientes de HPC. Entretanto, muitos desses algoritmos não estão otimizados para fazer uso eficiente de recursos HPC, aumentando o tempo de uso desses recursos e o consumo de energia elétrica. Portanto, este trabalho de pesquisa se insere na busca por soluções para conter este problema. A proposta é que mediante a compreensão e caracterização do desempenho das aplicações, seja possível realizar o uso eficiente dos recursos computacionais, e como consequência, reduzir o consumo de energia. Assim, o objetivo geral deste trabalho é identificar os aspectos que limitam o desempenho das aplicações e sugerir otimizações de *software* e *hardwares* ideais para sua execução. Para isso é proposta uma metologia de avaliação de desempenho com o uso do modelo *Roofline* [Williams et al. 2009]. A escolha deste modelo foi motivada por ele oferecer um modelo visual que relaciona o desempenho do processador ao tráfego de memória, apontando os fatores que limitam o desempenho da aplicação e possíveis pontos de otimização. O seu diferencial

é unir todas essas informações em um único ambiente, gerando informações pertinentes sobre o funcionamento do *hardware* e do *software*.

Assim, com base em parâmetros teóricos e práticos, os quais são a base do modelo *Roofline*, e usando o gráfico bidimensional que faz parte de ferramentas que implementam este modelo (p. ex Intel Advisor), foi desenvolvida uma metodologia de avaliação de desempenho que permite: i) Identificar padrões de desempenho nas aplicações; ii) Identificar os principais requisitos computacionais e fatores que limitam o desempenho de uma aplicação e; iii) com base nesses requisitos sugerir a melhor arquitetura para executar uma aplicação. Foram desenvolvidos diversos experimentos, com foco em algoritmos de AM. A metodologia proposta é usada para identificar padrões de desempenho e consumo de energia entre os algoritmos, medir a diferença no desempenho da aplicação, modificando características, tais como a linguagem de implementação e uso de multiprocessamento. Por fim, ainda por meio do modelo *Roofline*, é proposta uma modificação ou aquisição no *hardware* para que atenda de forma mais eficiente os requisitos da aplicação com base em seus gargalos computacionais.

2. Trabalhos Relacionados

Alguns trabalhos utilizam o modelo *Roofline* para a avaliação do desempenho das aplicações e suas limitações. Em [Kim et al. 2011], o *Roofline* é utilizado para compreender os limites do *hardware* e auxiliar na definição das otimizações necessárias ao algoritmo *Finite-Difference Time-Domain*. Além disso, o modelo foi utilizado para comparar o desempenho do algoritmo entre duas arquiteturas. Em [Ibrahim et al. 2020] é utilizado o modelo *Roofline* para analisar as características de desempenho de aplicações do *NAS* implementadas em *OpenACC*, e por meio da compreensão dos resultados obtidos pelo *Roofline*, foi possível otimizar em 2x o *Speedup* das aplicações. Em [de Sá et al. 2020] o modelo *Roofline* foi utilizado para a compreensão dos requisitos computacionais das aplicações do *NAS* e feitos estudos iniciais com algoritmos de AM, para efeitos comparativos do requisitos computacionais de domínios distintos, mas ambos usados em HPC. Os resultados mostraram que os requisitos computacionais das aplicações de CFD e de AM eram claramente distintas.

Entretanto, o diferencial deste trabalho está na utilização das métricas do modelo *Roofline* para identificar padrões de desempenho e consumo de energia entre as aplicações científicas. Além disso, o modelo *Roofline* foi utilizado para medir a diferença no desempenho da aplicação modificando características de implementação como a linguagem e multiprocessamento. Por fim, a metodologia adotada por este trabalho também propõe sugerir por meio do modelo *Roofline* uma modificação ou aquisição no *hardware* que atenda de forma mais eficiente os requisitos da aplicação com base em seus gargalos computacionais.

3. Proposta de Metodologia para Avaliação de Desempenho

A proposta de avaliação de desempenho é baseada nos três pilares que sustentam o modelo *Roofline*: a intensidade aritmética - IntA - (operações por byte de tráfego), a largura de banda - LB - (capacidade de transmissão) e o desempenho de ponto flutuante - DPF - (operações por segundo) dentro de um mesmo gráfico dimensional. Na Figura 1, é proposta uma forma de observar e relacionar esses três conceitos: a hierarquia de memória convencional, representada pelo triângulo, com memórias RAM, níveis de cache (LLC, L2 e L1) e registrador (RG); os requisitos computacionais das aplicações científicas (representados pelas setas *CPU Intensive* e *Memory Intensive*), que mostram os fatores que influenciam os parâmetros da CPU e da memória relacionados com a hierarquia de memória. Quanto mais próximo do registrador mais rápido é o processamento, significando uma maior intensidade no uso da CPU. O

mesmo pode ser observado para a memória, pois quanto mais requisições em memória RAM, mais intensivo é o uso de memória. E os parâmetros relacionados ao *Roofline* (representados pelas setas *Bandwidth* e *GFLOPs*) mostram que a LB tende a ser maior quando mais eventos de cache e memória ocorrem, enquanto o *GFLOPs* tende a ser maior quando mais eventos de CPU ocorrem. Já a IntA é uma relação entre o DPF e a LB. Quanto mais dados forem processados por byte trafegado, maior a IntA e mais intensivo em processamento. Quanto menor a IntA, maior é a intensidade em memória. É com base nessa relação que o *Roofline* é capaz de caracterizar os requisitos computacionais das aplicações e vincular isso às arquiteturas.

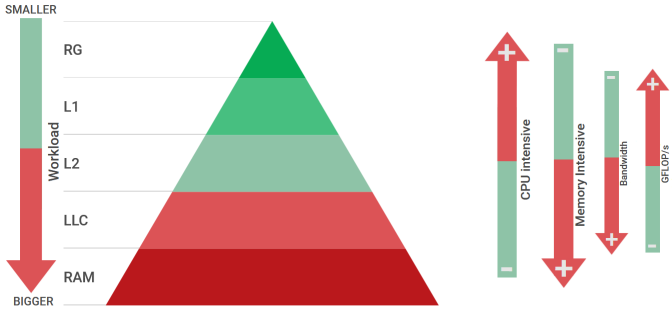

Figura 1. Relação entre a a hierarquia de memória, os requisitos das aplicações científicas e os parâmetros do *Roofline*. Fonte: autoria própria.

A implementação de um modelo *Roofline* depende da coleta de parâmetros práticos (parâmetros da aplicação) e teóricos (parâmetros do máximo atingível pela arquitetura), que podem variar a metodologia de coleta desses parâmetros de acordo com a arquitetura sendo utilizada. Existem algumas ferramentas que implementam o modelo *Roofline*, sendo uma das principais o *Intel Advisor*. Essa ferramenta é capaz de monitorar o desempenho de aplicações no geral, implementadas em linguagens como: C, C++, Fortran e Python3 (utilizando a métrica –profile-python). Além de analisar o desempenho da aplicação em relação ao *hardware*, dá ênfase para análise de eficiência de vetorização e *multithreading*. No gráfico do *Advisor* (Figuras 2-6), cada *loop*/função da aplicação representa um círculo, sendo 3 tipos diferentes de círculos: vermelhos (alto custo computacional), amarelos (custo moderado) e verdes (baixo custo). É possível, ainda, avaliar o quão otimizada se encontra a aplicação, relacionando a distância do ponto ao teto de desempenho (quanto mais longe, mais espaço para ganho de desempenho). O teto acima do ponto representa o limitador de desempenho daquele ponto em específico, onde cada teto representa seu próprio limite de *hardware*, caso não seja feita a otimização representada pelo próximo teto. A parte do gráfico onde a tonalidade é mais clara, é considerada intensiva em memória, tonalidade moderada é considerada um equilíbrio entre memória e processamento, já a tonalidade mais escura representa intensivo em processamento. Os pontos são deslocados entre essas áreas de acordo com sua IntA.

Assim, com base nos parâmetros teóricos e práticos e usando o gráfico bidimensional do *Intel Advisor*, a metodologia de avaliação de desempenho proposta permite: i)identificar padrões de desempenho, os principais requisitos computacionais e fatores que limitam o desempenho de uma aplicação e com base nesses requisitos sugerir a melhor arquitetura para executar uma aplicação. Na próxima seção são apresentados experimentos, em que é feita a avaliação de desempenho de algoritmos de AM com base nesta metodologia proposta.

4. Experimentos e Resultados

Os experimentos foram executados em uma arquitetura com processador i7-8700, *clock* de 3.20 GHz, pico teórico de 307.2 GFlops, 64GB de memória RAM e o pico teórico de largura de banda em 42.56GB. Para a Seção 4.3, além da arquitetura descrita acima, (Arch1), também foi utilizada uma arquitetura semelhante, com exceção do tamanho da memória RAM, que ao invés de 64GB tem 16GB (denominada Arch2).

Os experimentos foram divididos em três fases e para todas elas é utilizado o modelo *Roofline* com o método de avaliação proposto. Na Seção 4.1 (fase 1) o objetivo é identificar requisitos computacionais para aplicações de AM e os principais fatores que limitam seu desempenho. Foram executados dois algoritmos de Árvores de Decisão - AD - (C4.5 [Quinlan 1993] para classificação e CART [Breiman et al. 1984] regressão). Além disso, foram analisados outros quatro algoritmos de AM que já fizeram parte de um estudo preliminar[de Sá et al. 2020]: K-Means; CNN (*Convolutional Neural Network*); ANN (*Artificial Neural Network* modelo *Multilayer Perceptron*) e SVR (*Support Vector Regression*). Na Seção 4.2 (fase 2) os experimentos têm como objetivo avaliar diferentes formas de implementação de um mesmo algoritmo e o uso de técnicas de otimização analisando as mudanças no desempenho e nos requisitos computacionais. Nesta fase foram avaliados os algoritmos de AD C4.5 implementado em Python e em C, e o CART da biblioteca scikit-learn. A terceira fase (Seção 4.3) consiste em sugerir a arquitetura mais eficiente para a execução de uma aplicação com base em seu principal requisito computacional. Em todas as fases de experimentos os algoritmos foram treinados em uma base de dados sintética de 25 mil exemplos e 10 atributos. Porém, é importante ressaltar que este tamanho não afeta os resultados do *Roofline*.

4.1. Análise dos Requisitos Computacionais das Aplicações de AM

O entendimento sobre os gargalos das aplicações auxilia em otimizações no *software*. Compreender os requisitos computacionais de um conjunto de aplicações auxilia na aquisição, no desenvolvimento e na escolha de um *hardware* que melhor atende às necessidades de desempenho da aplicação. Além disso, segundo [García Martín 2020], a compreensão sobre as características e os requisitos computacionais das aplicações está diretamente associado ao consumo de energia elétrica, pois "os valores de energia variam dependendo do processador e da arquitetura. Por exemplo, uma instrução DRAM consome três ordens de magnitude a mais de energia do que uma operação ALU".

Nesta fase foram utilizados os algoritmos K-Means, CNN, MLP e SVR e os algoritmos de AD C4.5 (classificação) e CART (regressão). Os resultados mostraram que, apesar de cada algoritmo apresentar suas particularidades de implementação e algumas características distintas, foi possível observar um padrão de comportamento entre os algoritmos de AM. Os algoritmos de AM analisados por meio do modelo *Roofline* foram considerados intensivos em memória e possuem baixo requisito de processamento, e possuem gargalos voltados à memória DRAM. Assim, uma máquina com uma alta disponibilidade de memória e processamento baixo seriam adequados para a execução do algoritmo em seu estado atual.

Na Figura 2 é possível identificar dois resultados do algoritmo de AD com códigos sequenciais e implementados na linguagem *Python*. O quadrado representa a árvore de classificação (C4.5), e o círculo representa a de Regressão (CART). Baseado na tonalidade de cores de fundo de seus principais pontos (vermelhos e amarelos) é possível identificar o algoritmo CART como equilibrado na utilização de memória e processamento e o C4.5 como intensivo em memória. Também é possível observar que ambas possuem o mesmo gargalo,

a memória DRAM. Além da diferença nos requisitos computacionais, também houve uma diferença no desempenho dos algoritmos. Com base no gráfico podemos observar que o C4.5 possui menos GFLOPs que o CART.

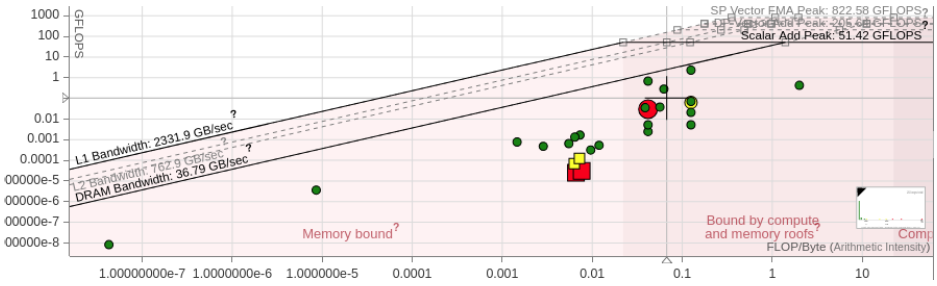

Figura 2. Comparação dos algoritmos de Árvore de Decisão: Classificação C4.5 (quadrado) x Regressão CART (círculo).

A partir dos resultados desta seção, é possível concluir que identificar os requisitos computacionais é crucial para se analisar as aplicações científicas. Esta identificação é útil não apenas para avaliar o desempenho das aplicações, assim sugerindo arquiteturas de *hardware* que melhor atenda as necessidades das aplicações, mas que também está associado a um padrão de consumo de energia. Em [García Martín 2020] é identificado que instruções de memória tendem a consumir mais energia que instruções voltadas ao processamento; em [Silva et al. 2021] foram medidas métricas como consumo de energia e tempo de execução, onde o CART tem tempo de execução e consumo de energia menor que o C4.5; na Figura 2 foi possível identificar que o C4.5 possui mais intensidade de memória que o CART. Estes resultados demonstraram que aplicações intensivas em memória tendem a consumir mais energia que aplicações intensivas em processamento. É possível realizar essa identificação dos requisitos computacionais por meio do modelo *Roofline* com base na métrica de IntA, onde otimizações nessa área podem aumentar o desempenho e reduzir o consumo de energia da aplicação.

4.2. Identificando Características de Desempenho

Com a utilização do modelo *Roofline*, por meio da análise de desempenho de ponto flutuante, é possível avaliar mudanças de desempenho baseadas em uma determinada característica (mudança na linguagem de implementação, *multithreading*, multiprocessamento, etc), possibilitando uma comparação entre o estado anterior e posterior a uma otimização, e se houve ganho de desempenho. Para esta fase foi utilizado o algoritmo C4.5 em linguagem *Python* e C, sem nenhuma técnica de otimização; algoritmo CART implementado *Python* por meio da biblioteca *scikit-learn*, que utiliza multiprocessamento [1].

Na Figura 3 é apresentado o resultado para o algoritmo C4.5 implementado em *Python*, sem nenhuma técnica de otimização. Por meio da tonalidade de cores de fundo de seus principais pontos (vermelhos), esta aplicação se caracteriza como intensiva em memória, e possui baixíssimo requisito computacional voltada a processamento. Seu gargalo é a memória DRAM, entretanto, com base na distância entre os pontos e o teto de memória DRAM, pode-se notar que o algoritmo faz uso ineficaz de memória e processamento. Isso mostra que há possibilidades para otimizações dentro dessas áreas e assim, aumentar o seu desempenho.

Na Figura 4 é apresentado o resultado do algoritmo C4.5 implementado em C, sem técnica de otimização. Com base nos requisitos computacionais de seus pontos, pode-se carac-

[1] Os dois algoritmos se diferenciando apenas pela função que calcula o ganho de cada atributo para particionar os nós da árvore, como demonstrado em [Silva et al. 2021]. O C4.5 utiliza a razão do ganho e CART o Gini.

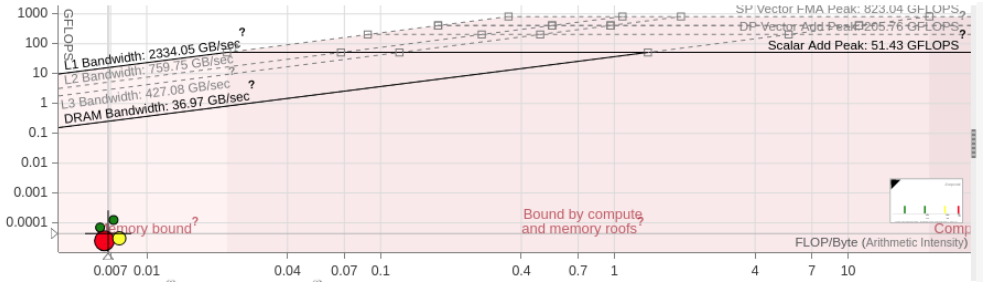

Figura 3. Resultado do algoritmo de AD C4.5 implementado em *Python*.

terizar esta implementação como equilibrada entre memória e processamento. Entretanto, esta implementação tende muito mais a memória do que processamento, chegando a ficar próximo da linha que separa essas duas caracterizações. Seu principal gargalo é a memória DRAM, e assim como a implementação em *Python*, também faz um uso ineficaz de memória e processamento. Porém, este algoritmo possui um desempenho e uma IntA maior que a implementação em *Python*. Como ambas possuem as mesmas características de implementação, a hipótese é que este ganho de desempenho é devido à diferença de linguagem de implementação, pois o C prevalece sobre o *Python* em questões de desempenho [Pereira et al. 2017].

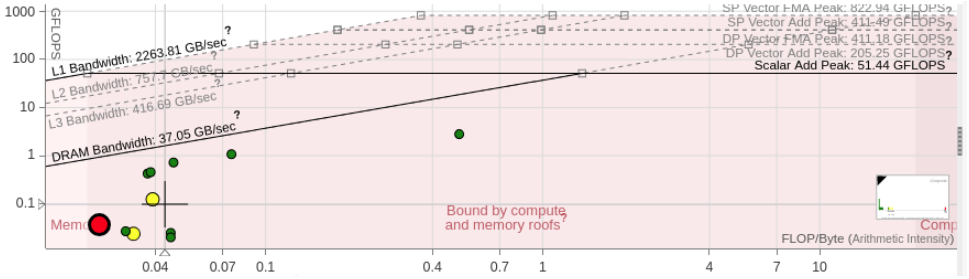

Figura 4. Resultado do algoritmo de AD C4.5 implementado em C.

Na Figura 5 é apresentado o resultado do algoritmo CART implementado em *Python* utilizando a biblioteca *scikit-learn*, e que possui multiprocessamento. Esta implementação é definida como equilibrada entre memória e processamento de acordo com o *Roofline*. O fator que limita seu desempenho é a memória DRAM, onde possui uma pequena lacuna para otimizações baseadas em DRAM. Neste caso, por meio de um melhor uso de memória cache compartilhada é possível melhorar significativamente o seu desempenho. Além disso, uma arquitetura com uma alta disponibilidade de memória seria o ideal para a execução do algoritmo em seu estado atual. O fato de possuir multiprocessamento provou ser uma otimização importante quando se busca otimizar uma aplicação, pois fez com que essa implementação obtivesse um desempenho e uma IntA muito maior que as demais implementações.

Com base na avaliação dos resultados desta seção, foi possível identificar a diferença do desempenho de diversas formas de implementação de um algoritmo e o quanto essas característica de implementação podem afetar os requisitos computacionais de determinada aplicação usando os resultados do modelo *Roofline*. Por exemplo, na Figura 5, que após aplicar uma otimização de multiprocessamento, a aplicação passou a ser mais intensiva em processamento do que a aplicação sem otimização (Figura 3). Apesar deste tipo de conhecimento de que aplicar técnicas otimização ser óbvio, o objetivo foi analisar se, e como, o *Roofline* pode auxiliar nessa tarefa de indicar as necessidades de otimização de uma aplicação, o que se mostrou bastante efetivo. Além disso, com o uso do modelo também foi possível identificar que a

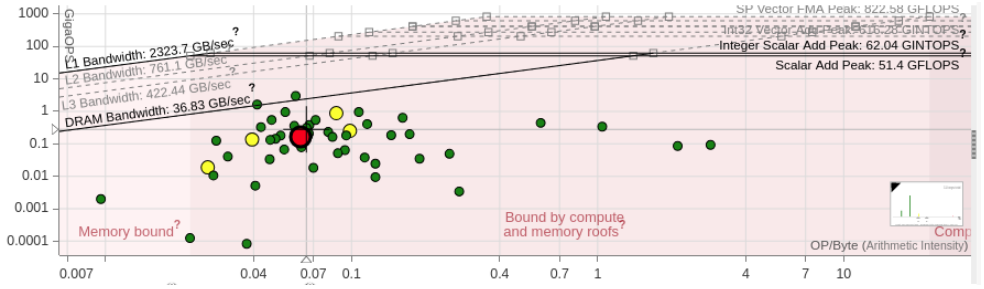

Figura 5. Resultado do algoritmo CART implementado com o *scikit-learn* (Python).

linguagem de implementação adotada contribui para a mudança no desempenho da aplicação.

4.3. Sugerindo a Arquitetura mais Eficiente

Com base nos resultados das fases anteriores, é possível identificar um certo padrão entre os algoritmos de AM, caracterizado pelo gargalo em memória DRAM. Portanto, para esta avaliação foram utilizadas duas arquiteturas que possuem diferenças significantes de memória DRAM. As arquiteturas Arch1 e Arch2, como a arquitetura com maior e menor memória RAM, respectivamente. Os experimentos foram realizadas com o algoritmo CART implementado em *Python*, sem adotar nenhuma técnica de otimização.

Na Figura 6 são apresentados dois resultados dentro de um mesmo ambiente, sendo o quadrado o resultado da execução do algoritmo na Arch1 e o círculo o resultado da execução na Arch2. É possível identificar com base nos principais pontos da aplicação (vermelhos e amarelos) uma diferença nítida de desempenho entre as duas arquiteturas, fazendo com que o resultado da Arch1 fique bem mais próximo ao teto de memória DRAM que o resultado da Arch2. Este resultado significa que o algoritmo possui menos lacunas de otimização para esta área, provando que o algoritmo aumentou sua eficiência de utilização de memória DRAM, possibilitando um desempenho maior em todos os seus *loops* e funções.

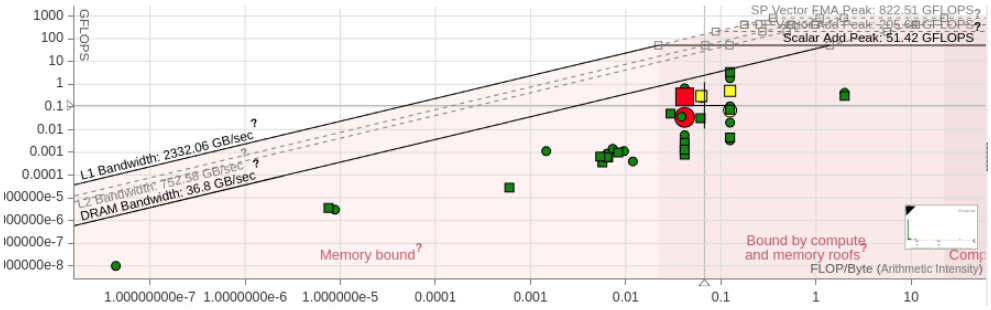

Figura 6. Resultado do CART em *Python* na Arch1 (quadrado) e Arch2 (círculo).

Com os resultados obtidos nesta seção, é possível observar que por meio do análise do modelo *Roofline*, pode-se identificar o gargalo de uma aplicação, e com isso, sugerir uma arquitetura que melhor atenda a sua necessidade, melhorando seu desempenho.

5. Conclusões e Trabalhos Futuros

Neste trabalho foi apresentado o modelo *Roofline* como proposta de metodologia de avaliação de desempenho de *Hardware* e *Software* e destacado os principais resultados alcançados por meio de três fases de experimentos. As contribuições deste trabalho são:

na Seção 4.1 é destacada a importância dos requisitos computacionais e como estão relacionados ao consumo de energia; na Seção 4.2 foi identificado a diferença no desempenho e nos requisitos computacionais das aplicações baseadas em uma determinada característica de implementação adotada, auxiliando na tarefa de indicar as necessidades de otimização das aplicações; na Seção 4.3 foi possível desenvolver uma metodologia para sugerir arquiteturas para a execução de aplicações de forma eficiente com base em seus principais gargalos computacionais, proporcionando um aumento no desempenho. Assim, a metodologia proposta neste trabalho fornece *insights* sobre como realizar um melhor uso dos recursos computacionais, aumentando o desempenho para aplicações científicas e reduzindo seu consumo de energia, contribuindo para uma computação mais verde e sustentável. Ainda, este projeto contribui para a sociedade por estar inserido diretamente no objetivo 13 (ações contra a mudança global do clima) dos Objetivos de Desenvolvimento Sustentável estabelecidos pela ONU para agenda até 2030 e pactuados pelo Brasil.

Em trabalhos futuros, será feita uma análise mais profunda sobre os padrões de consumo de energia do algoritmo utilizando a metodologia *Roofline*. Além disso, utilizando os parâmetros coletados pelo modelo e outras ferramentas (*Likwid* e *Perf*) desenvolver um modelo teórico que permita estimar o desempenho e o consumo de energia de uma aplicação.

Agradecimentos

Os autores agradecem o apoio financeiro do LNCC/MCTI, CNPq e FAPERJ.

Referências

[Breiman et al. 1984] Breiman, L., Friedman, J., Stone, C. J., and Olshen, R. A. (1984). *Classification and regression trees*. CRC press.

[de Sá et al. 2020] de Sá, V., Klôh, V., Schulze, B., and Ferro, M. (2020). Análise de desempenho e de requisitos computacionais utilizando o modelo roofline: Um estudo para aplicações de inteligência artificial e do nas-hpc. In *WSCAD 2020 - WIC ()*.

[García Martín 2020] García Martín, E. (2020). *Energy Efficiency in Machine Learning: Approaches to Sustainable Data Stream Mining*. PhD thesis, D. Computer Science.

[Ibrahim et al. 2020] Ibrahim, K., Williams, S., and Oliker, L. (2020). *Performance Analysis of GPU Programming Models Using the Roofline Scaling Trajectories*, pages 3–19.

[Kim et al. 2011] Kim, K.-H., Kim, K.-H., and Park, Q.-H. (2011). Performance analysis and optimization of three-dimensional fdtd on gpu using roofline model. *Computer Physics Communications*, 182:1201–1207.

[Pereira et al. 2017] Pereira, R., Couto, M., Ribeiro, F., Rua, R., Cunha, J., Fernandes, J. a. P., and Saraiva, J. a. (2017). Energy efficiency across programming languages: How do energy, time, and memory relate? In *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering*, SLE 2017, page 256–267, New York, NY, USA.

[Quinlan 1993] Quinlan, J. R. (1993). *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Silva et al. 2021] Silva, G., Schulze, B., and Ferro, M. (2021). Performance and energy efficiency analysis of machine learning algorithms towards green ai: a case study of decision tree algorithms. Master's thesis, National Lab. for Scientific Computing.

[Williams et al. 2009] Williams, S., Waterman, A., and Patterson, D. (2009). Roofline:an insightful visual performance model for multicore architectures. *Commun.ACM*, 52(4):65–76.