Avaliação de Desempenho para Aplicações Científicas utilizando o Modelo Roofline

Vitor de Sá, Vinícius P. Klôh, Bruno Schulze, Mariza Ferro

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

{vitorsa, viniciusk, schulze, mariza}@lncc.br

Abstract. *Understanding the factors that limit the performance of applications and their computational requirements can help in software optimization and in the acquisition, development and choice of hardwares that best meet the application's computational performance. This work proposes a methodology for evaluating and characterizing the performance of scientific applications using the Roofline model. A series of experiments with a set of applications were developed and analyzed following the proposed methodology. The results allow us to identify the aspects that limit the performance of applications, suggest software optimizations and the hardware that increases the performance of the applications.*

Resumo. *Entender os fatores que limitam o desempenho das aplicações e seus requisitos computacionais pode auxiliar nas otimizações de software e na aquisição, no desenvolvimento e na escolha de um hardware que melhor atende às necessidades de desempenho da aplicação. Neste trabalho é proposta uma metodologia de avaliação e caracterização de desempenho de aplicações científicas usando o modelo Roofline. Foram desenvolvidas uma série de experimentos com diferentes aplicações, as quais são analisadas seguindo a metodologia proposta. Os resultados permitem identificar os aspectos que limitam o desempenho, sugerir otimizações de software e de hardwares que aumentem a eficiência de execução das aplicações avaliadas.*

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*. Isso o diferencia das avaliações de desempenho usando *benchmarks*, os quais analisam separadamente o desempenho da aplicação e a capacidade computacional do *hardware*.

Assim, com base em parâmetros teóricos e práticos, os quais são a base do modelo *Roofline*, e usando o gráfico bidimensional que faz parte deste modelo, foi desenvolvida uma metodologia a ser seguida e diretrizes de interpretação para avaliação de desempenho. Foram analisadas seis aplicações de AM e seis do NAS-HPC [Frumkin et al. 2009] indicando como é possível, a partir dos resultados obtidos: i) identificar e caracterizar padrões de desempenho nas aplicações; ii) identificar os principais requisitos computacionais e fatores que limitam o seu desempenho e; iii) com base nesses requisitos sugerir otimizações e arquiteturas.

2. Trabalhos Relacionados

O modelo Roofline foi publicado em 2009 por [Williams et al. 2009] e diversos trabalhos demonstram que este modelo é capaz de fornecer informações práticas dos efeitos que degradam o desempenho de códigos, em diferentes modelos de implementação, domínios e hardwares variados e, adicionalmente, dando dicas para melhorá-los [Sato et al. 2009, Lorenzo et al. 2011, Kim et al. 2011, Ibrahim et al. 2020] .

[Então et al. 2013] propõem o *Cache-aware Roofline model* (CARM), uma abordagem que adiciona o reconhecimento de cache ao Roofline original para torná-lo mais eficaz e capaz de representar os limites de processamento paralelo em processadores *multicore* com hierarquias de memória complexas [Ilic et al. 2014, Denoyelle et al. 2019]. O modelo proposto foi avaliado experimentalmente para diferentes arquiteturas e os resultados demonstram que o CARM é capaz de sugerir otimizações que trouxeram melhorias de desempenho para as aplicações bem acima do modelo original [Marques et al. 2017b, Marques et al. 2020]. Em [Marques et al. 2017a] é proposta a integração do *CARM* ao *Intel Advisor*. Essa integração possibilitou a compreensão do desempenho da aplicação em diferentes níveis de memória. Este estudo também analisa os gargalos mais críticos e possíveis passos de otimização de dez aplicações com a ferramenta *Intel Advisor*. Posteriormente, [Ilic et al. 2017] propõem considerar como os principais aspectos da microarquitetura, como o acesso a diferentes níveis de memória afetam a potência alcançável e o consumo de energia usando o CARM.

Outra abordagem que modifica o Roofline original é o *Mansard Roofline Model* (MaRM) [Marques et al. 2021]. É proposto um conjunto mínimo de contadores de hardware que, segundo os autores, devem ser considerados para que o modelo possa representar de forma realista os limites de processamento paralelo em processadores modernos. Seguindo uma proposta de metodologia e diretrizes de interpretação do MaRM, nos resultados são obtidos *speedups* de até 5x ao otimizar uma aplicação real de bioinformática.

Em [Lopes et al. 2017], foi proposto a implementação de um modelo *Roofline* para GPUs, pois inicialmente o modelo foi desenvolvido apenas para CPUs. Para validação foram analisados 23 *benchmarks* executados em oito GPUs distintas e também para otimizar o processamento de imagens médicas [Serrano et al. 2018]. [Yang et al. 2018] propôs o desenvolvimento da ferramenta *Empirical Roofline Tool* que oferece um conjunto mais realista de limites de desempenho para GPU e CPU incorporando diferentes padrões de acesso à memória e contadores de FLOPs mais próximos ao consumo real.

É possível observar nestes diversos trabalhos que o modelo *Roofline* pode ser muito útil para avaliar o desempenho das aplicações e otimizá-las. Porém, ainda assim, interpretar o

modelo corretamente e fazer uso prático do seu gráfico não é uma tarefa trivial devido a falta de documentação clara sobre o modelo e também por inconsistências existentes na literatura de como interpretá-lo. Assim, neste trabalho é proposta uma metodologia a ser seguida e diretrizes para interpretação do modelo, as quais permitam identificar padrões de desempenho nas aplicações e possíveis otimizações de hardware e software para melhorarias no desempenho.

3. Proposta de Metodologia para Avaliação de Desempenho com o Roofline

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 interpretar 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 (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 seu uso. 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.

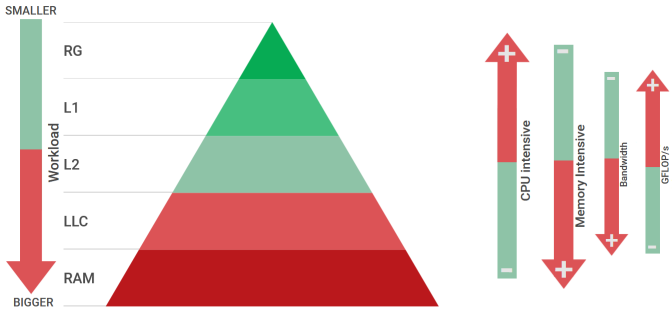


Figura 1. Relação entre a a hierarquia de memória, os requisitos das aplicações científicas e os parâmetros do *Roofline*. Fonte: autoria própria.

A implementação de um modelo *Roofline* depende da coleta de parâmetros práticos (parâmetros da aplicação) e teóricos (parâmetros do máximo atingível pela arquitetura). A implementação utilizada neste trabalho é a que está integrada ao *Intel Advisor*. Essa ferramenta é capaz de monitorar o desempenho de aplicações implementadas em linguagens 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*. Para interpretar o gráfico do *Advisor*, serão utilizados os resultados dos experimentos (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 média é considerada um equilíbrio entre memória e processamento, e 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 apresentados no gráfico do modelo *Roofline* e seguindo as diretrizes de interpretação apresentadas nesta seção, a metodologia consiste em: 1) Analisar e identificar os principais requisitos computacionais e fatores que limitam o desempenho da aplicação. Com base nesses requisitos 2) identificar mudanças no desempenho e nos requisitos computacionais da aplicação baseada na implementação das técnicas de otimização e 3) identificar a arquitetura mais eficiente para a execução de uma aplicação com base em seu principal gargalo computacional. Na próxima seção são apresentados experimentos em que é feita a avaliação de desempenho das aplicações com base nesta proposta.

4. Experimentos e Resultados

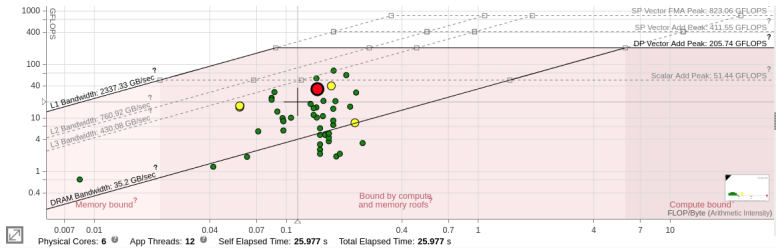
Os experimentos foram executados em uma arquitetura com processador i7-8700, *clock* de 3.20 GHz, pico teórico de 307.2 GFlops, 64GB de memória RAM e o pico teórico de largura de banda em 42.56GB. Para a Seção 5.2, além desta arquitetura (denominada de Arch1), também foi utilizada uma arquitetura semelhante, 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* implementado pelo *Intel Advisor*. Na Seção 5 o objetivo é identificar requisitos computacionais para aplicações de AM e HPC e os principais fatores que limitam seu desempenho. Foram analisadas seis aplicações do *NAS-HPC* [Frumkin et al. 2009]: BT, LU, CG, MG, SP e FT, implementadas em *OpenMP*, executadas 10 vezes para cada tamanho de problema entre A e C. Também foram executados e analisados os algoritmos de AM: Árvores de Decisão - AD - (C4.5 [Quinlan 1993] e CART [Breiman et al. 1984]), K-Means; Redes neurais CNN (*Convolutional Neural Network*) e *Multilayer Perceptron* (MLP) e SVR (*Support Vector Regression*). Na Seção 5.1 (fase 2) os experimentos têm como objetivo avaliar o uso de técnicas de otimização e analisar as mudanças no desempenho e nos requisitos computacionais. Nesta fase foram implementados e avaliados os algoritmos C4.5 em Python e C, e o CART com a biblioteca Scikit-learn [Pedregosa et al. 2011]. A terceira fase (Seção 5.2) consiste em sugerir a arquitetura mais eficiente para a execução da aplicação com base em seu principal requisito computacional. Em todas as fases de experimentos os algoritmos de AM 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*. Para todas as aplicações o número de *threads* foi predefinido para ocupar todos os núcleos de processamento disponíveis.

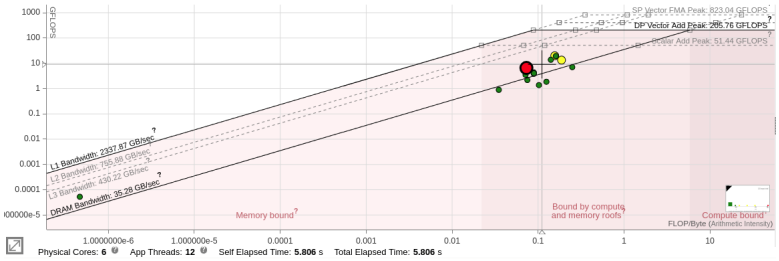
5. Fase 1 - Análise dos Requisitos Computacionais das Aplicações Científicas

Na Figura 2 (a) é apresentado o resultado para a aplicação BT do *NAS*. Baseado na tonalidade de cores de fundo de seus principais pontos (vermelhos e amarelos) é possível caracterizar a aplicação como sendo equilibrada na utilização de memória e processamento. Além disso, pode-se observar que existem dois principais gargalos de desempenho para essa aplicação: a memória L3 e a adição escalar. Por mais que haja uma quantidade significativa de pontos (verdes) limitados pela memória DRAM, são pontos que possuem baixo custo computacional e representariam um ganho de desempenho insignificante caso fossem otimizados.

Analisando esses resultados, a recomendação é fazer um melhor uso de memória cache L2 e vetorização das funções, ações que poderiam aumentar o desempenho dessa aplicação. Na



(a) Aplicação Block Tri-diagonal (BT)



(b) Aplicação Multi-Grid (MG)

Figura 2. Resultados das aplicações do NAS-HPC [Sá et al. 2020]

Figura 2 (b) é apresentado o resultado da aplicação MG do *NAS*. Entre as aplicações do *NAS*, a MG é a que apresentou uma maior utilização de memória e menor intensidade aritmética. Entretanto, fazendo uma interpretação sobre o modelo *Roofline*, ela ainda é caracterizada como uma aplicação equilibrada entre memória e processamento, tendendo mais para memória. Há espaço para otimizações de memória para essa aplicação, desde que se faça um uso eficiente da memória cache compartilhada. Porém, será necessária uma maior intensidade aritmética para que otimizações de processamento sejam eficazes. Isso significa reestruturar o código caso seja possível, visando reduzir o tráfego de memória. Poderia ser utilizada, por exemplo, uma técnica de compressão e descompressão que aumentam a intA (deslocando a aplicação para a direita no gráfico), permitindo espaço maior de desempenho atingível até a aplicação chegar ao seu teto de desempenho. Os resultados apresentados nas Figuras 2 (a) e (b) são para aplicações do *NAS* apenas do tamanho C, porém foram muito similares para os tamanhos A e B. Isso se explica pelo fato de o *Roofline* utilizar o tempo de execução da aplicação como um divisor para as operações de ponto flutuante, construindo no final de sua execução uma média do desempenho de ponto flutuante por segundo. Portanto, o que foi possível observar é que os resultados geralmente são independentes do tamanho do problema. Assim, pela limitação de espaço e pela similaridade entre os resultados, os tamanhos A e B não serão apresentados graficamente, bem como para as demais aplicações (LU, CG, SP e FT) [1]. Porém, os resultados indicam que todas as aplicações se caracterizaram como equilibradas entre memória e processamento, com algumas particularidades em relação aos seus gargalos. Por exemplo, as aplicações CG e FT são limitadas principalmente pelo teto de adição escalar, tendo como prioridade para otimização a vetorização de seus principais pontos. A aplicação LU é limitada principalmente pela memória compartilhada, além de possuir também um gargalo de adição escalar. Já a aplicação SP apresenta um gargalo que difere das demais aplicações do *NAS* com um gargalo de memória DRAM.

[1] Todos os gráficos e detalhes das demais aplicações estão disponíveis em [Sá et al. 2021]

Na Figura 3 é possível identificar dois resultados dos algoritmos de AM para AD, com códigos sequenciais e implementados na linguagem *Python*. O quadrado representa o C4.5, e o círculo representa o CART [2]. Baseado na tonalidade de cores de fundo de seus principais pontos (vermelhos e amarelos) é possível identificar o 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.

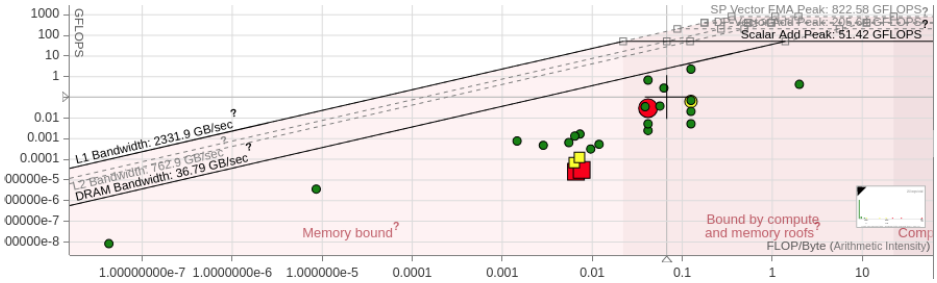


Figura 3. Resultados dos algoritmos C4.5 (quadrado) e CART (círculo).

Ainda foram analisados os algoritmos K-Means, CNN, MLP e SVR, cujos resultados são discutidos em [Sá et al. 2020]. Foi possível identificar um padrão de comportamento entre os dois conjuntos de aplicações. Enquanto as aplicações do *NAS* requerem mais processamento e mais possibilidades de otimizações voltadas a processamento, os algoritmos de AM analisados possuem mais requisitos de memória e requerem muito pouco processamento. Além disso, observa-se uma diferença entre os gargalos, enquanto a maioria dos algoritmos do *NAS* possuem gargalos voltados à memória cache compartilhada e vetorização, os algoritmos de AM apresentam um gargalo de memória DRAM.

Com base na análise dos resultados obtidos nesta seção, foi possível identificar os gargalos e os requisitos computacionais para os conjuntos de aplicações científicas. Esta compreensão é de grande importância para otimizações e tomadas de decisões voltadas para o desempenho. Ter o entendimento sobre os gargalos das aplicações auxiliam em otimizações no *software*, melhorando seu desempenho. Além disso, conhecer os requisitos da aplicação auxilia também na escolha de um *hardware* que melhor atenda às necessidades da aplicação.

5.1. Fase 2 - Identificando Características e Alterações de Desempenho

Com a utilização do *Roofline*, por meio da análise de desempenho de ponto flutuante, é possível avaliar mudanças de desempenho baseadas na alteração de 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 análise os algoritmos C4.5 e CART em linguagem *Python*, sem nenhuma técnica de otimização (Seção 5 - Figura 3) são comparados com o C4.5 implementado em C, sem nenhuma técnica de otimização e com o algoritmo CART implementado em *Python* por meio da biblioteca *scikit-learn*, utilizando multiprocessamento.

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 caracterizar esta implementação como equilibrada entre memória e processamento. Entretanto, esta

[2]Os dois algoritmos se diferenciam 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.

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 o mesmo algoritmo base, 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].

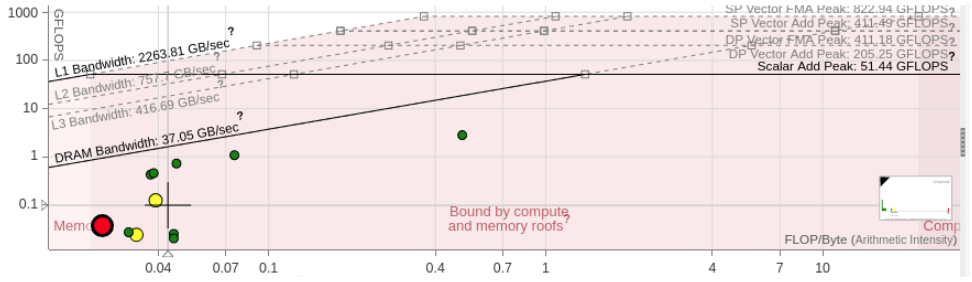


Figura 4. Resultado do algoritmo de AD C4.5 implementado em C.

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.

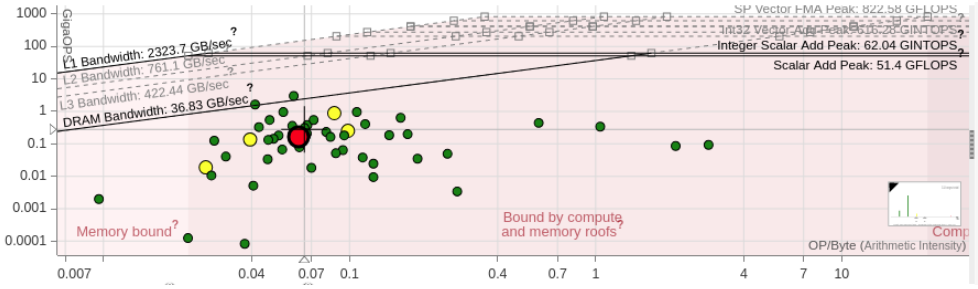


Figura 5. Resultado do algoritmo CART implementado com o *scikit-learn* (Python).

Com base na avaliação dos resultados desta seção, foi demosntrado como é possível identificar a diferença do desempenho de diferentes 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.

5.2. Fase 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.

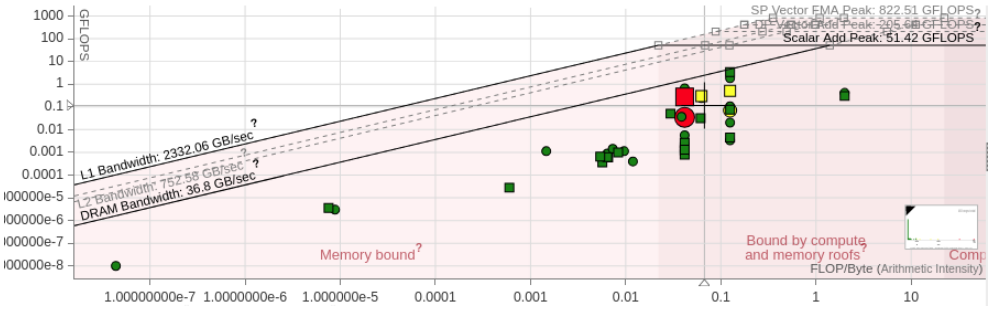


Figura 6. Resultado do CART em *Python* na Arch1 (quadrado) e Arch2 (círculo).

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.

6. Conclusões e Trabalhos Futuros

Neste trabalho foi apresentado o modelo *Roofline* como proposta de metodologia de avaliação de desempenho de *Hardware* e *Software*. A metodologia, com base nos parâmetros teóricos e práticos apresentados no gráfico do modelo *Roofline* e seguindo as diretrizes de interpretação apresentadas, consiste em: 1)Analisar e identificar os principais requisitos computacionais e fatores que limitam o desempenho da aplicação. Com base nesses requisitos 2) identificar mudanças no desempenho e nos requisitos computacionais da aplicação baseada na implementação das técnicas de otimização e 3) identificar a arquitetura mais eficiente para a execução de uma aplicação científica com base em seu principal gargalo computacional. Experimentos com dois conjuntos de aplicações,foram analisados seguindo a metodologia proposta e usando o *Roofline* implementado no *Intel Advisor*. Seguindo as diretrizes de interpretação foi possível identificar os aspectos que limitam o desempenho das aplicações, sugerir otimizações de *software* e sugerir *hardwares* que melhorem o desempenho.

Em trabalhos futuros, será feita uma análise mais profunda sobre os padrões de consumo de energia do algoritmo utilizando a metodologia *Roofline*.

Agradecimentos

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

Referências

[Antão et al. 2013] Antão, D., Taniça, L., Ilic, A., Pratas, F., Tomás, P., and Sousa, L. (2013). Monitoring performance and power for application characterization with the cache-aware roofline model. In Wyrzykowski, R., Dongarra, J. J., Karczewski, K., and Wasniewski, J., editors, *Parallel Processing and Applied Mathematics - 10th International Conference, PPAM 2013, Warsaw, Poland, September 8-11, 2013, Revised Selected Papers, Part I*, volume 8384 of *Lecture Notes in Computer Science*, pages 747–760. Springer.

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

[Denoyelle et al. 2019] Denoyelle, N., Goglin, B., Ilic, A., Jeannot, E., and Sousa, L. (2019). Modeling non-uniform memory access on large compute nodes with the cache-aware roofline model. *IEEE Trans. Parallel Distributed Syst.*, 30(6):1374–1389.

[Frumkin et al. 2009] Frumkin, M., Jin, H., and Yan, J. (2009). Implementation of nas parallel benchmarks in high performance fortran.

[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.

[Ilic et al. 2014] Ilic, A., Pratas, F., and Sousa, L. (2014). Cache-aware roofline model: Upgrading the loft. *IEEE Computer Architecture Letters*, 13(1):21–24.

[Ilic et al. 2017] Ilic, A., Pratas, F., and Sousa, L. (2017). Beyond the roofline: Cache-aware power and energy-efficiency modeling for multi-cores. *IEEE Trans. Computers*, 66(1):52–58.

[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.

[Lopes et al. 2017] Lopes, A., Pratas, F., Sousa, L., and Ilic, A. (2017). Exploring GPU performance, power and energy-efficiency bounds with cache-aware roofline modeling. In *2017 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2017, Santa Rosa, CA, USA, April 24-25, 2017*, pages 259–268. IEEE Computer Society.

[Lorenzo et al. 2011] Lorenzo, J. A., Pichel, J. C., Pena, T. F., Suarez, M., and Rivera, F. F. (2011). Study of performance issues on a SMP-NUMA system using the roofline model. In *2011 International Conference on Parallel and Distributed Processing Techniques and Applications*.

[Marques et al. 2017a] Marques, D., Duarte, H., Ilic, A., Sousa, L., Belenov, R., Thierry, P., and Matveev, Z. A. (2017a). Performance analysis with cache-aware roofline model in intel advisor. In *2017 International Conference on High Performance Computing & Simulation, HPCS 2017, Genoa, Italy, July 17-21, 2017*, pages 898–907. IEEE.

[Marques et al. 2017b] Marques, D., Duarte, H., Sousa, L., and Ilic, A. (2017b). Analyzing performance of multi-cores and applications with cache-aware roofline model. In *2017 International Conference on High Performance Computing & Simulation, HPCS 2017, Genoa, Italy, July 17-21, 2017*, pages 933–934. IEEE.

[Marques et al. 2020] Marques, D., Ilic, A., Matveev, Z. A., and Sousa, L. (2020). Application-driven cache-aware roofline model. *Future Gener. Comput. Syst.*, 107:257–273.

[Marques et al. 2021] Marques, D., Ilic, A., and Sousa, L. (2021). Mansard roofline model: Reinforcing the accuracy of the roofs. *ACM Trans. Model. Perform. Evaluation Comput. Syst.*, 6(2):7:1–7:23.

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

[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.

[Sato et al. 2009] Sato, Y., Nagaoka, R., Musa, A., Egawa, R., Takizawa, H., Okabe, K., and Kobayashi, H. (2009). Performance tuning and analysis of future vector processors based on the roofline model. In *Proceedings of the 10th Workshop on MEmory Performance: DEaling with Applications, Systems and Architecture*, MEDEA '09, page 7–14, New York, NY, USA. Association for Computing Machinery.

[Serrano et al. 2018] Serrano, E., Ilic, A., Sousa, L., García-Blas, J., and Carretero, J. (2018). Cache-aware roofline model and medical image processing optimizations in gpus. In Yokota, R., Weiland, M., Shalf, J., and Alam, S. R., editors, *High Performance Computing - ISC High Performance 2018 International Workshops, Frankfurt/Main, Germany, June 28, 2018, Revised Selected Papers*, volume 11203 of *Lecture Notes in Computer Science*, pages 509–526. Springer.

[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.

[Sá et al. 2020] 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 *Anais Estendidos do XXI Simpósio em Sistemas Computacionais de Alto Desempenho*, pages 22–29, Porto Alegre, RS, Brasil. SBC.

[Sá et al. 2021] Sá, V., Klôh, V., Schulze, B., and Ferro, M. (2021). Análise e avaliação de desempenho para aplicações científicas utilizando o modelo roofline. `http://pergamum.lncc.br/pergamumweb/vinculos/000000/00000013.pdf`.

[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.

[Yang et al. 2018] Yang, C., Gayatri, R., Kurth, T., Basu, P., Ronaghi, Z., Adetokunbo, A., Friesen, B., Cook, B., Doerfler, D., Oliker, L., Deslippe, J., and Williams, S. (2018). An empirical roofline methodology for quantitatively assessing performance portability. In *2018 IEEE/ACM Int. Workshop on Performance, Portability and Productivity in HPC (P3HPC)*, pages 14–23.