

Avaliação de técnicas de autotuning

Virgilio S. Tagliavini¹, Aleardo M. Junior¹

¹Instituto de Biociências, Letras e Ciências Exatas - Universidade Estadual Paulista
15054-000 São José do Rio Preto - SP - Brasil

{virgilio.s.tagliavini, aleardo.manacero}@unesp.br

Abstract. *Autotuning systems were developed due to the complexity of modern applications and hardware, which makes impractical the manual tuning of said applications. Many heuristic algorithms are utilized since an exhaustive search of the configuration space would be too costly. However, it is difficult to compare the effectiveness of autotuning techniques presented by different approaches since there is no universal metric to measure it as well as their effectiveness can be affected by many factors. Therefore, it is interesting to study the different autotuning techniques and assess their effectiveness to tune different problems. In this study we revise modern autotuning techniques and implement them to configure programs that are representative of modern applications.*

Resumo. *Os sistemas de autotuning foram desenvolvidos devido à complexidade de aplicações e de hardwares modernos, o que torna a configuração manual dessas aplicações impraticável. Vários algoritmos heurísticos são utilizados porque a busca exaustiva do espaço de configurações é custosa demais. Contudo, é difícil comparar a efetividade de técnicas de autotuning apresentadas por diferentes abordagens porque não existe uma métrica universal, assim como sua efetividade pode ser afetada por diversos fatores. Assim, é interessante estudar as diferentes técnicas de autotuning e avaliar sua efetividade no tuning de diferentes problemas. Neste estudo revisamos técnicas modernas de autotuning e as implementamos para otimizar programas representativos de aplicações modernas.*

1. Introdução

Ao usar computadores é ideal utilizar os recursos de hardware de forma ótima, diminuindo o tempo e energia gastos pelas aplicações. Para isso, aplicações devem ser configuradas de forma ótima para o ambiente de execução. Porém, isto se tornou uma tarefa complexa demais para ser realizada por programadores na maioria dos casos porque as aplicações modernas possuem um grande número de configurações disponíveis. Além disso, as máquinas atuais também possuem uma grande quantidade e variedade de características que devem ser consideradas ao configurar uma aplicação.

Por isso, foram desenvolvidos os sistemas de configuração automática, ou *autotuning*. De acordo com [Roy et al. 2016], esses sistemas tem como objetivo automatizar o processo de configuração de aplicações em um dado sistema de hardware. Contudo, uma busca exaustiva do espaço de configurações é computacionalmente custoso para a maioria das aplicações, o que leva aos sistemas de *autotuning* modernos buscarem configurações eficientes por meio de algoritmos heurísticos ou empíricos.

Entretanto, é difícil comparar resultados obtidos por diferentes trabalhos porque, como explicado por [Willemsen et al. 2024], não existe uma metodologia padrão para avaliação de técnicas de *autotuning* e a efetividade de suas medidas podem ser alteradas de diversas formas como omitir ou enfatizar partes de seu treinamento, ou manipular o conjunto de treinamento. Além disso, a eficácia de uma técnica de *autotuning* baseada em aprendizado de máquina também é dependente do conjunto de treinamento utilizado, como explicado por [Tørring and Elster 2022]. Assim, não é trivial comparar os resultados de dois trabalhos diferentes. Por isso, é interessante estudar a efetividade de diferentes técnicas de *autotuning* em problemas representativos de aplicações atuais como processamento de dados [Masouros et al. 2024], *kernels* algébricos [Tørring and Elster 2022] e compilação de programas [Xiao et al. 2024].

2. Técnicas de autotuning e aproximação de funções

Para configurar automaticamente um programa é necessário um algoritmo de otimização que explore o espaço de configurações de forma a maximizar ou minimizar uma métrica desejada, como o tempo de execução ou a energia consumida por uma aplicação. Neste trabalho, foram implementados sete algoritmos de otimização iterativos, que são: 1) Otimização Bayesiana: iterativamente explora o espaço de configurações de uma função de aquisição auxiliada por uma função que aproxima o desempenho do programa.; 2) Algoritmos genéticos: Define uma população de indivíduos que representam configurações do programa, a população de cada geração é criada pela recombinação dos indivíduos da geração anterior; 3) Evolução diferencial: Similar aos Algoritmos genéticos mas seus indivíduos são criados pela diferença dos valores de outros três indivíduos; 4) Enxame de partículas: Similar aos Algoritmos Genéticos mas cada indivíduo é atualizado em relação ao seu melhor desempenho prévio e o melhor desempenho da população; 5) *Simulated Annealing*: Aleatoriamente altera a posição atual e probabilisticamente escolhe se a nova posição será aceita; 6) *Basin Hopping*: Similar ao *Simulated Annealing* mas utiliza um algoritmo de otimização local após alterar a posição atual; 7) Algoritmo de Nelder-Mead: Iterativamente move os vértices de um simplex que representa uma configuração do programa pelas operações de reflexão, expansão e contração.

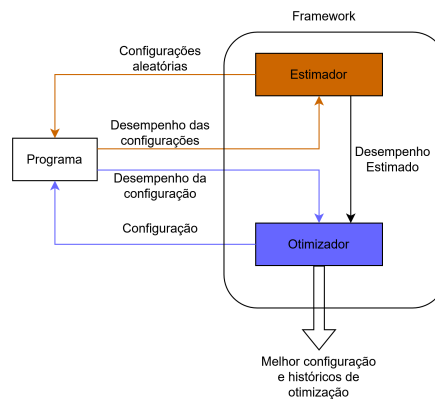
Além das técnicas de otimização, uma aplicação de *autotuning* pode utilizar técnicas de aproximação de funções, modeladas a partir de pontos de referência, para reduzir seu custo computacional. Este trabalho implementou 12 técnicas de aproximação de funções ao todo, que podem ser divididas em seis grupos: 1) MLP, CNN, RNN, LSTM e GRU: Arquiteturas de redes neurais artificiais; 2) Árvores de Decisão, Florestas Aleatórias e *Gradiente Boosting*: Algoritmos que dividem o espaço de configurações e decidem o valor de um ponto através de testes sucessivos ao percorrer a árvore da raiz até suas folhas; 3) Processos Gaussianos: Algoritmos que modelam uma função parametrizada com o auxílio de uma função de covariância; 4) Gradiente Descendente Estocástico: Atualiza os parâmetros de funções a fim de minimizar o custo da função a partir do gradiente da amostra; 5) K Vizinhos Mais Próximos: Algoritmo que aproxima o valor de um ponto como a média das k entradas mais próximas; 6) Regressão por Vetores de Suporte: Variante de Máquinas de Vetores de Suporte para problemas de regressão linear.

3. Framework

Para comparar a efetividade das técnicas de *autotuning* apresentadas acima foi desenvolvido um *framework* que utiliza essas técnicas para otimização caixa-preta de programas e registra o histórico de otimização do programa, salvando o desempenho de cada configuração testada e a configuração de melhor desempenho.

O *framework* possui dois módulos principais: Otimizador e Estimador. Estimador utiliza as técnicas de aproximação de funções descritas anteriormente para modelar o desempenho de configurações do programa a partir de um conjunto com n entradas e medidas de desempenho do programa. Se esse conjunto não existir, o módulo o criará com n execuções do programa com configurações aleatórias. Otimizador explora o espaço de configurações utilizando os algoritmo de otimização apresentados acima, ele otimiza o programa até alcançar a verba de execuções definida pelo programador e salva o histórico de otimização em um arquivo externo para avaliação futura. O Otimizador pode também utilizar um modelo do módulo Estimador treinado com uma determinada quantidade de pontos de referência, nesse caso a verba do Otimizador será a quantidade de aproximações utilizadas na exploração do espaço. Segue um diagrama da estrutura do *framework*.

Figura 1. *Framework* implementado



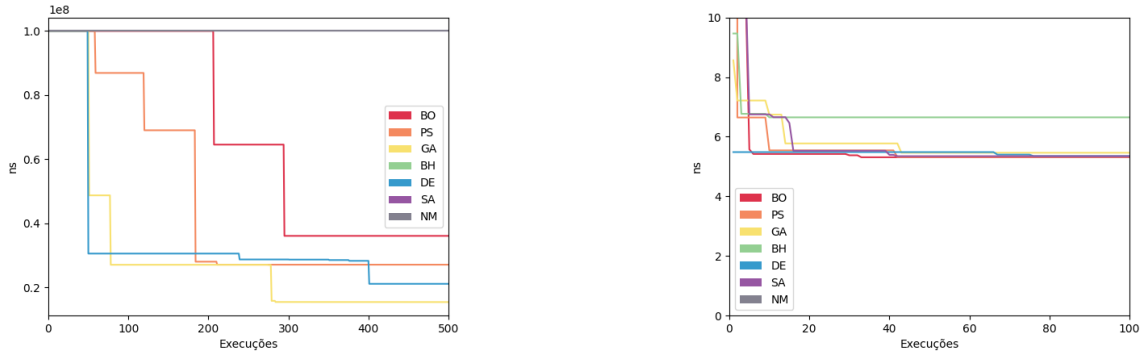
4. Resultados preliminares

Neste momento o trabalho está na etapa de desenvolvimento e execução: as técnicas de *autotuning* e aproximação de funções descritas na Seção 2 foram implementadas e utilizadas em dois problemas de otimização de programas: *kernels* algébricos de GPUs, especificamente um *kernel* de multiplicação geral de matrizes, e programas escritos na linguagem C utilizando o benchmark PolyBench/C, essas otimizações foram realizadas com o objetivo de minimizar o tempo de execução desses programas. O histórico de otimização das técnicas para *kernels* de GPU e compilação em C, respectivamente, é apresentado na Figura 2.

Analisando o histórico de otimização do *kernel* de GPU para multiplicação de matrizes é perceptível que os Algoritmos Genéticos alcançaram o melhor desempenho para a maioria do número de execuções do *kernel*, sendo superados apenas pelo Enxame de Partículas para menos de 100 execuções do *kernel*. Observando o histórico de otimização de compilação de programas escritos em C nota-se que todos os algoritmos, com exceção do *Basin Hopping* e o algoritmo de Nelder-Mead, rapidamente encontraram configurações

eficientes e convergiram em configurações com execuções de aproximadamente 5.3 ns. Assim, os resultados preliminares indicam que os Algoritmos Genéticos e Enxame de Partículas são os mais efetivos para aplicações de *autotuning* já que rapidamente convergiram em configurações eficientes em ambos os testes realizados.

Figura 2. Históricos de otimização



5. Conclusão

Os resultados apresentados na Figura 2 mostram que os Algoritmos Genéticos e Enxame de Partículas obtiveram o melhor desempenho em ambos os programas otimizados porque rapidamente convergiram nas configurações de melhor desempenho. Porém esses resultados não são representativos de todas as aplicações modernas de *autotuning* então no desenvolvimento futuro deste trabalho será necessário otimizar outros programas, como acesso paralelo de dados em sistemas distribuídos ou programas de processamento de dados. Além disso é interessante no modificar o *framework* apresentado para registrar outras métricas do desempenho das técnicas de *autotuning*, como tempo de execução e memória consumida pelas próprias técnicas.

Referências

- Masouros, D., Retsinas, G., Xydis, S., and Soudris, D. (2024). Sparkle: Deep learning driven autotuning for taming high-dimensionality of spark deployments. *IEEE Transactions on Cloud Computing*, 12(4):1058–1073.
- Roy, A., Balaprakash, P., Hovland, P. D., and Wild, S. M. (2016). Exploiting performance portability in search algorithms for autotuning. In *International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, Chicago, Illinois. IEEE, IEEE.
- Tørring, J. O. and Elster, A. C. (2022). Analyzing search techniques for autotuning image-based gpu kernels: The impact of sample sizes. In *International Parallel and Distributed Processing Symposium Workshops*, EUA. IEEE, IEEE.
- Willemsen, F. J., Schoonhoven, R., Filipovič, J., Tørring, J. O., Nieuwpoort, R. V., and Werkhoven, B. V. (2024). A methodology for comparing optimization algorithms for auto-tuning. *Future Generation Computer Systems*, 1(159):489–504.
- Xiao, G., Qin, S., Li, K., Chen, J., and Yin, J. (2024). EA tuner: Comparative study of evolutionary algorithms for compiler auto-tuning. In *Proceedings of the 2024 27th International Conference on Computer Supported Cooperative Work in Design*, Tianjin, China. IEEE, IEEE.