Uma Metodologia para Reduzir o Custo de Aprendizado para Técnicas *Offline* de Otimização de Aplicações Paralelas

Gustavo P. Berned[1], **Arthur Lorenzon**[1]

[1]Laboratório de Otimização de Sistemas - Universidade Federal do Pampa
Campus Alegrete Caixa Postal 810 – 97546-550 – Alegrete – RS – Brasil

{gustavoberned, aflorenzon}@unipampa.edu.br

1. Introdução

A exploração do paralelismo em nível de *threads* (TLP - *Thread Level Parallelism*) tem sido amplamente utilizada para melhorar o desempenho de aplicações de diferentes domínios. Entretanto, muitas aplicações não escalam conforme o número de *threads* aumenta, ou seja, executar uma aplicação utilizando o máximo de *threads* não trará, necessariamente, o melhor resultado para tempo, energia ou o produto entre estas denominado EDP (*Energy Delay Product*), que visa medir a eficiência de consumo de energia de um circuito, devido a questões relacionadas à *hardware e software* [Raasch and Reinhardt 2003],[Lorenzon and Filho 2019].

Portanto, é preciso utilizar metodologias que consigam identificar um número ideal de *threads* para tais aplicações, sejam estas *online* (busca enquanto a aplicação é executada) ou *offline* (busca antes da execução da aplicação). Entretanto, metodologias *online* acabam adicionando uma sobrecarga na execução da aplicação, o que não acontece nas abordagens *offline* [Lorenzon et al. 2018]. Com base nisto, este trabalho apresenta uma metodologia genérica para reduzir significativamente o tempo de busca pelo número de *threads* ideal para aplicações paralelas que utilizam a metodologia *offline*, inferindo o ambiente de execução das aplicações paralelas utilizando apenas pequenos conjuntos de entrada de dados[1].

2. Reduzindo o Custo de Aprendizado de Estratégias *Offline*

Este trabalho propõe um *framework* onde o usuário provê apenas uma aplicação paralela (e.g., binário) com um pequeno conjunto de entrada. O *framework* aplica um algoritmo de busca (fase de aprendizado) sobre esta pequena entrada para encontrar um número adequado de *threads*. Uma vez que este número é encontrado, a aplicação é executada (fase estável) com este número de *threads* sobre o conjunto de dados original.

O algoritmo utilizado durante a fase de aprendizado recebe como entrada um conjunto de núcleos C, um conjunto pequeno de entrada da aplicação A e outros três parâmetros: (i) α número inicial de *threads*; (ii) β taxa de aumento do número de *threads*; (iii) número máximo de núcleos n. O processo de aprendizado inicia aumentando exponencialmente o número de *threads* α a uma taxa β até n, visando minimizar uma função de custo. Ao alcançar o máximo local, é realizado uma busca utilizando o método *hill-climbing* modificado, no intervalo onde o algoritmo encontrou o menor valor da função de custo. Para evitar mínimos locais e platôs durante este processo, o algoritmo

[1]Este estudo foi financiado pela FAPERGS - Códigos de Financiamento - 19/2551-0001224-1, 19/2551-0001689-1

realiza movimentos laterais, onde, este ocorre antes de selecionar o número de *threads* ideal, testando as configurações vizinhas em outro ponto ainda não testado. Sendo assim, será considerado o melhor número de *threads* o conjunto que possua o menor valor na função de custo.

O *framework* proposto foi implementado na linguagem de programação Python3, sendo que, durante todo o processo aprendizado não há modificação e nem recompilação de código. Além do mais, o *framework* pode ser utilizado por qualquer uma das interfaces de programação paralela mais utilizadas, como *OpenMP*, *MPI*, *Pthreads*.

3. Resultados Experimentais

Dezoito aplicações paralelas já implementadas em C e C++ foram executadas em dois ambientes multiprocessados: (*i*) processador AMD 1700, contendo 8 núcleos e 16 *threads* com 16 GB de memória; (*ii*) processador Intel Core I9-7920X com 12 núcleos e 24 *threads*, dispondo de 64GB de memória. A metodologia proposta foi comparada com os seguintes cenários: (**STD**): execução da aplicação com o maior número de *threads* disponível no ambiente, sendo este o cenário padrão de execução, sem a intervenção do usuário; **OMP_DYN**: recurso presente no OPEN_MP que ajusta dinamicamente o número de *threads* para cada região paralela visando utilizar da melhor maneira possível os recursos presentes no sistema.

Os resultados mostram que, se avaliarmos o conjunto inteiro de *benchmarks* (média geométrica), a metodologia proposta obteve melhorias de 31% para EDP em relação a **STD** e de 35% em relação ao **OMP_DYN**. Além do mais, a metodologia proposta convergiu para o melhor número de *threads* para as aplicações ou muito próxima desta, quando comparada a busca sobre o pequeno conjunto de entrada ao maior, tendo um custo de aprendizado 88.3% e 82.6% menor nos processadores AMD e Intel respectivamente.

4. Conclusão

Apresentamos uma metodologia genérica que reduz significativamente o custo de busca pelo número ideal de *threads* em estratégias *offline*, inferindo o ambiente das aplicações utilizando pequenas entradas. Através de um grande conjunto de aplicações, mostramos que a metodologia consegue convergir para resultados ideais ou muito próximos. Como trabalhos futuros, a metodologia será aprimorada para identificar o número de *threads* para cada região paralela de uma aplicação.

Referências

Lorenzon, A. F. and Filho, A. C. S. B. (2019). *Parallel Computing Hits the Power Wall - Principles, Challenges, and a Survey of Solutions*. Springer Briefs in Computer Science. Springer.

Lorenzon, A. F., Oliveira, C. C. D., Souza, J. D., and Filho, A. C. S. B. (2018). Aurora: Seamless optimization of openmp applications. *IEEE Transactions on Parallel and Distributed Systems*, pages 1–15.

Raasch, S. E. and Reinhardt, S. K. (2003). The impact of resource partitioning on smt processors. In *PACT*, pages 15–25.