Comparação de Desempenho e Consumo de Energia de Aplicações Paralelizadas com OpenMP *taskloop* e *parallel for*

Luan Pereira[1], Sandro Matheus V. N. Marques[1], Fábio D. Rossi[2], Marcelo C. Luizelli[1] e Arthur F. Lorenzon[1]

[1]Laboratório de Otimização de Sistemas – Universidade Federal do Pampa
Campus Alegrete (UNIPAMPA) – Alegrete – RS – Brasil

[2]Instituto Federal Farroupilha - Campus Alegrete – Alegrete – RS – Brasil

luanvargas.aluno@unipampa.edu.br

Resumo. *A biblioteca OpenMP detém diretivas que possuem o mesmo propósito, porém com funcionalidade diferente. Um exemplo disso é a diretiva #pragma omp for e a #pragma omp taskloop. Neste trabalho apresentamos uma análise de tempo de execução e consumo energético de ambas diretivas. Nós mostramos que a utilização da diretiva parallel for é até 42% mais rápida e consome 27,21% menos energia que a implementação com taskloop.*[1]

1. Introdução

O uso de diferentes técnicas de programação paralela tem sido cada vez mais recorrente pela comunidade de alto desempenho [Lorenzon and Beck Filho 2019]. Estas técnicas são normalmente aferidas à bibliotecas de programação paralela, como por exemplo, o *Open MultiProcessing* (OpenMP). A biblioteca OpenMP possui conjuntos de diretivas que facilitam a exploração do paralelismo. Com isso é possível encontrar diretivas que possuem objetivos semelhantes com funcionalidades diferentes, como por exemplo, *#pragma omp taskloop* e *#pragma omp for*, utilizadas para a exploração de paralelismo em laços.

Na exploração de paralelismo com o uso de *taskloop*, *tasks* são criadas para um intervalo (chamado de *chunk*) de iterações do laço e são armazenadas em uma estrutura (i.e., *bag of tasks*) [Bronis R. de Supinski 2019]. Por outro lado, quando se usa o *parallel for*, as iterações do laço são divididas em *chunks* através de um *scheduler* definido a priori, que serão atribuídos para cada uma das *threads*. Neste sentido, muito embora ambas as estratégias têm o mesmo objetivo, é importante avaliar o comportamento dessas diretivas em relação ao desempenho e consumo de energia.

Diferentes trabalhos têm avaliado o uso das diretivas para exploração de paralelismo em laços do OpenMP. X. Teruel et al. [Teruel et al. 2013] propõe a diretiva *taskloop* e comparam o seu desempenho com outras implementações de laços paralelos. A. Podobas et al. [Podobas and Karlsson 2016] avaliam a escalabilidade e desempenho de ambas as diretivas, mostrando que o *taskloop* pode ser até 3.2% mais rápido que o *parallel for*. A. Rico et al. [Rico et al. 2019] analisam o desempenho e escalonamento de threads ao reimplementar um benchmark utilizando a diretiva *taskloop*. O autor conclui que a execução é até 30% mais rápida com a diretiva *taskloop* quando comparada com o *parallel-for*.

[1]Este trabalho foi parcialmente financiado pela FAPERGS nos projetos 19/2551-0001224-1 e 19/2551-0001689-1, CNPq PIBIC e FAPERGS PROBIC.

Diferentemente dos trabalhos já realizados, apresentamos uma avaliação das diretivas *taskloop* e *parallel for* em aplicações paralelas considerando o desempenho e o consumo de energia. Através da execução de quatro aplicações, mostramos que a implementação com *parallel for* é até 42% mais rápida e consome 27,21% menos energia que a implementação com *taskloop*.

2. Programação Paralela com OpenMP

O OpenMP é uma biblioteca para programação em memória compartilhada em C/C++ e FORTRAN, que consiste de um conjunto de diretivas de compiladores, bibliotecas de funções e variáveis de ambientes [Bronis R. de Supinski 2019]. O paralelismo é explorado através da inserção de diretivas no código sequencial que informam ao compilador como e quais partes do código devem ser executadas em paralelo. Duas destas diretivas compreendem a *#pragma omp taskloop* e *#pragma omp for*, que realizam a distribuição da carga de trabalho em uma região paralela. As principais características de cada uma delas são descritas abaixo.

Taskloop: É um construtor gerador de *tasks*. Quando é invocado, as iterações de um laço são particionadas em *tasks* para a execução paralela. A divisão dos dados de cada *task* é criada de acordo com as cláusulas de compartilhamento de dados (i.e., *private* e *shared*). As *tasks* são criadas e armazenadas em uma estrutura interna do OpenMP chamada *bag of tasks* [Bronis R. de Supinski 2019] até sua execução. Por padrão, a execução das *tasks* é feita de acordo com a disponibilidade das *threads* no Sistema Operacional.

Laços paralelos: Quando o *parallel for* é invocado, as iterações são dividas em uma série de *chunks*. Para cada *chunk*, é determinada qual *thread* será atribuída a sua execução. Essa atribuição ocorre de acordo com o tipo de *scheduler* definido na variável de ambiente *$OMP_SCHEDULE*. Contudo, o tipo *scheduler* pode ser alterado em código por meio do uso da cláusula (*schedule*) em conjunto com o *parallel for*. Os *schedulers* são divididos em quatro tipos: *dynamic*; *guided*; *auto*; e *runtime*. Por padrão, o compilador GCC 9.2 define o *scheduler* em *dynamic*. Neste caso, cada *thread* executa um *chunk* de iterações, e então, demanda um novo *chunk*, até que todas as iterações estejam finalizadas [Bronis R. de Supinski 2019].

3. Metodologia

Os experimentos foram realizados no processador AMD Ryzen 9 3900X de 12 núcleos com o Sistema Operacional Ubuntu, kernel v.4.15.0. Este processador possui tecnologia *Simultaneous Multi-Threading* (SMT), podendo executar até 24 *threads*. Cada núcleo físico possuí uma memória *cache* L1 de dados e instruções de 32KB cada e uma *cache* L2 de 512KB. Cada grupo de três núcleos compartilham uma memória *cache* L3 de 16MB. O sistema como um todo possui 32 GB de memória principal.

Foram escolhidas quatro aplicações escritas em C e paralelizadas com OpenMP, com diferentes domínios: *fast fourier transform* (FFT), *Jacobi* (JA), *Poisson* (PO) e *STREAM* (ST). As aplicações foram executadas no compilador GCC 9.2.0 com a *flag* de otimização -O3 e com as entradas padrões dos *benchmark* escolhidos. Consideramos a versão do OpenMP 5.0. Para cada uma das aplicações foram feitas execuções com diferentes números de *threads*: 1, 2, 4, 8, 12 e 24. O DVFS *governor* estava configurado em *ondemand*. Este *governor*, ajusta a frequência de cada núcleo de acordo com a carga

de trabalho da aplicação. Foram feitas cinco execuções para cada aplicação e número de threads. O tempo de execução de cada aplicação foi obtido através da função do OpenMP *omp_get_wtime* enquanto que o consumo de energia do processador e sistema de memória diretamente do *hardware* através da biblioteca RAPL (*running average power limit*).

4. Resultados

A Figura 1 apresenta o tempo de execução das aplicações executadas em função do número de *threads*, onde considerando a média de todas as aplicações utilizando *parallel for*, foi 42% menor em relação as aplicações utilizando a diretiva *taskloop*. No entanto, há casos em que essa diferença é menor, como por exemplo, nas aplicações FFT (Figura 1.a) e STREAM (Figura 1.D), onde a diferença é de apenas 2% e 0.1%, respectivamente. Este comportamento acontece pois ambas as aplicações fazem muitos acessos à memória durante a execução, o que gera um sobrecusto maior para a aplicação do que a diferença entre *threads* e *tasks* [Lorenzon et al. 2018]. Por fim, nas aplicações JA (Figura 1.b) e PO (Figura 1.c) a diferença no tempo de execução foi de 55.8% e 52%, respectivamente, assim apresentando um impacto maior na média geral de 42%.

Na Figura 2 pode-se observar o consumo de energia das aplicações em função do número de *threads* que foram executadas. Considerando a média de todas as aplicações, o uso da diretiva *task loop* gerou um aumento de 27% no consumo de energia. Por outro lado, em algumas aplicações o consumo de energia foi muito semelhante, como por exemplo, na aplicação FFT (Figura 2.a) e STREAM (Figura 2.d). Já a aplicação JA (Figura 2.b) implementada com *for* obteve uma média de consumo de energia 46% menor em comparação com a versão implementada com *taskloop*. Por fim, a PO (Figura 2.c) implementada com *for* mediu-se um consumo médio 42% menor do que na com *taskloop*.

Considerando os resultados obtidos nas execuções, a diretiva *parallel for* mostrou um melhor desempenho e eficiência energética em relação ao *taskloop*. Contudo, os benefícios do uso do *parallel for* se mostram mais presentes em aplicações cuja a dependência dados é estática. Sendo assim, em aplicações que envolvem algoritmos com dados dinâmicamente dependentes o *taskloop* se mostra mais adequado. Isto ocorre, por

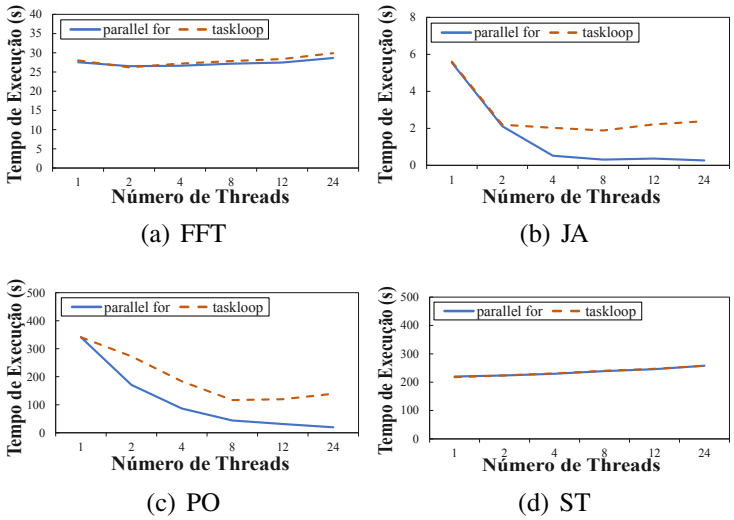

Figura 1. Tempo de execução de cada aplicação para cada configuração

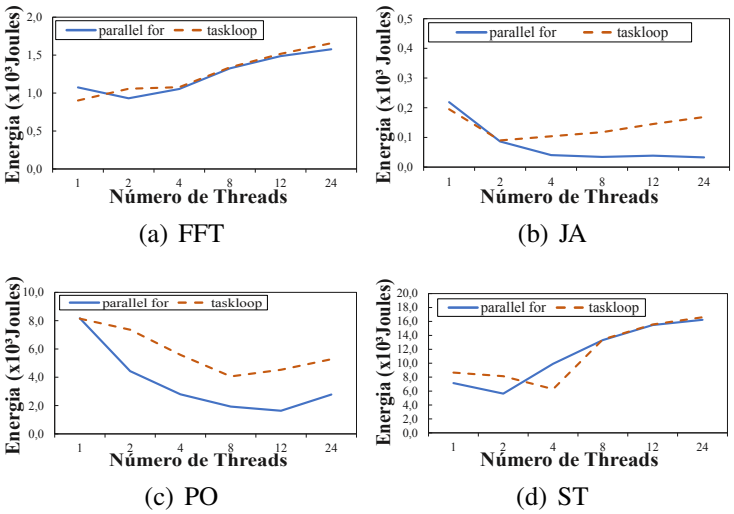

Figura 2. Consumo de energia de cada aplicação para cada configuração

que o uso do *taskloop* permite melhor balanceamento de carga entre as *threads* [Teruel et al. 2013] [Podobas and Karlsson 2016] [Rico et al. 2019].

5. Conclusão

Este trabalho realizou uma comparação do uso de duas diferentes diretivas de laços paralelos do OpenMP, o *taskloop* e *parallel for*. Mostrou-se que o uso da diretiva *parallel for* apresenta melhor desempenho, consumo de energia e escalabilidade em comparação ao *taskloop*. Como trabalhos futuros, pretende-se expandir o ambiente de execução para compreender outras aplicações e métricas.

Referências

Bronis R. de Supinski, M. K. (2019). Openmp technical report 8: Version 5.1 preview. Technical report, OpenMP Architecture Review Board.

Lorenzon, A. F. and Beck Filho, A. C. S. (2019). *Parallel Computing Hits the Power Wall Principles, Challenges, and a Survey of Solutions*. Springer.

Lorenzon, A. F., De Oliveira, C. C., Souza, J. D., and Beck, A. C. S. (2018). Aurora: Seamless optimization of openmp applications. *IEEE Transactions on Parallel and Distributed Systems*, 30(5):1007–1021.

Podobas, A. and Karlsson, S. (2016). Towards unifying openmp under the task-parallel paradigm. In *International Workshop on OpenMP*, pages 116–129. Springer.

Rico, A., Barrera, I. S., Joao, J. A., Randall, J., Casas, M., and Moretó, M. (2019). On the benefits of tasking with openmp. In *International Workshop on OpenMP*, pages 217–230. Springer.

Teruel, X., Klemm, M., Li, K., Martorell, X., Olivier, S. L., and Terboven, C. (2013). A proposal for task-generating loops in openmp. In *International Workshop on OpenMP*, pages 1–14. Springer.