

Comparação entre as Diretivas `parallel do` e `teams distribute` de OpenMP em uma Aplicação de Meios Porosos*

Gabriel Tremarin[†], Claudio Schepke

¹Laboratório de Estudos Avançados em Computação (LEA)
Universidade Federal do Pampa (UNIPAMPA) – Alegrete – RS – Brazil

{gabriel.tremarin.aluno, claudio.schepke}@unipampa.edu.br

Resumo. *O presente artigo tem por objetivo avaliar as diretivas `do` e `teams` da interface de programação paralela OpenMP. Para tanto, foram feitas adições de diretivas em uma aplicação de simulação de secagem de grãos. O estudo de caso variou o número de threads para avaliar o desempenho paralelo. A partir dos resultados obtidos, concluiu-se que o uso de `teams` pode ser uma alternativa tão eficiente quanto o clássico paralelismo de laços.*

1. Introdução

Acelerar a execução de uma aplicação passa pelo processo de paralelizar trechos de código. Uma das maneiras que isto pode ser feito é através do uso de diretivas de compilação. Diretivas de compilação são comandos processados em tempo de pré-compilação. Isso possibilita inserir instruções, como a invocação de novos fluxos de execução, tornando possível a instanciação de threads concorrentes.

Neste sentido, a interface de programação paralela OpenMP permite a criação de threads de maneira implícita através de diferentes diretivas [OpenMP 2023]. Diante disso, este artigo tem por objetivo investigar a performance de algumas diretivas (`do` e `teams`) de distribuição de tarefas paralelas de OpenMP. Para tanto, o trabalho utilizará uma aplicação de simulação de meios porosos desenvolvido por [de Oliveira 2020].

2. Estudo de Caso: Simulação de Meios Porosos

A aplicação consiste de um programa de simulação de secagem de grãos. Ela caracteriza-se como um problema de Dinâmica dos Fluidos, ou mais especificamente de meios porosos, dado a natureza das operações. O domínio físico foi modelado por volumes finitos. Tempo e espaço (em duas dimensões) foram discretizados, a fim de simular a transferência de temperatura e, conseqüentemente, a remoção da umidade dos grãos [da Silva et al. 2022b].

A aplicação modela os seguintes passos [da Silva et al. 2022c]: Leitura de valores de entrada; Alocação e inicialização dos dados; Iteração do passo de tempo discreto; Escrita dos resultados em arquivos; Desalocação de memória. O Algoritmo 1 apresenta os passos da etapa iterativa da aplicação. O laço externo itera o tempo discreto da simulação. Para cada passo de tempo, as propriedades físicas são computadas enquanto não atingirem um critério de parada (`convergence()`) ou um limite máximo de iterações.

*Trabalho parcialmente financiado pelo Edital PqG FAPERGS 07/2021: Projeto 21/2551-0002055-5.

[†]Bolsista PROBIC/FAPERGS 2022/2023.

Algoritmo 1: Etapa iterativa do algoritmo

```
max_iterations  $\leftarrow$  20.000 // número máximo de iterações
t0  $\leftarrow$  0 // tempo inicial
t  $\leftarrow$  0.04 // tempo final
dt  $\leftarrow$  0.01
while t0 < t do
    t0  $\leftarrow$  t0 + dt
    i  $\leftarrow$  1
    // calculando a convergência
    while i  $\neq$  max_iterations do
        solve_U()
        solve_V()
        solve_P()
        solve_Z()
        convergence()
    end while
end while
```

As rotinas *solve_U*, *solve_V*, *solve_P* e *solve_Z* operam as equações de quantidade de movimento na dimensão horizontal e vertical, de continuidade e de energia, respectivamente. Tais equações invocam sub-rotinas que iteram sobre todo o domínio bi-dimensional, para cada uma das propriedades físicas computadas. Além disso, operações especiais são feitas para elementos do entorno do domínio, bem como a invocação de sub-rotinas para a aplicação das condições de contorno.

3. Metodologia

Primeiramente identificamos 36 laços simples ou aninhados. Estes foram paralelizados (versão 1V) usando ambas as diretivas: `!$omp parallel do` e `!$omp target teams distribute`, com seus respectivos parâmetros, de acordo com cada laço (`private`, `num_teams`, `thread_limit`, ...).

Em trabalhos anteriores [da Silva et al. 2022a], usando a ferramenta PERF, constataram que as sub-rotinas *ResU* e *ResV*, invocadas pelas rotinas *solve_U* e *solve_V*, demandavam mais de 40% cada do tempo total de execução. Essas sub-rotinas invocam outras sub-rotinas, que por sua vez podem ou não invocar novas sub-rotinas, dependendo do tipo de operação, conforme apresentado na Figura 1. Diante disso, em um segundo momento, o código foi modificado de maneira a diminuir o número de chamadas de sub-rotinas (versão 2V).

A sub-rotina *ResU* e *ResV* é chamada 3 vezes em cada chamada de *solve_U* e *solve_V*. Além disso, a sub-rotina *Quick*, invocada em *ResU* e *ResV* manipula 18 argumentos, mas realiza apenas 9 operações envolvendo instruções de multiplicação (0, 2 ou 6) e soma (1, 4, 5 ou 8). Diante disso, optou-se por incluir o código da sub-rotina *Quick* nas rotinas *ResU* e *ResV*, ao invés de invocá-la, uma vez que o código encontra-se no meio de um trecho de um laço aninhado. Isso garante uma granularidade paralela maior em cada uma das funções.

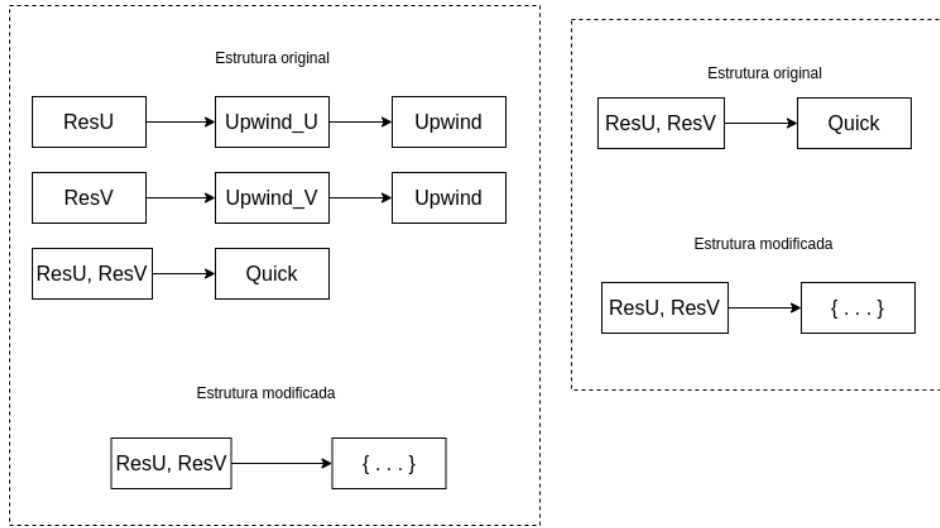


Figura 1. Modificações introduzidas no código-fonte

As sub-rotinas `Upwind_U` e `Upwind_V` são invocadas ao final de `ResU` e `ResV` e são diretamente chamados em um laço simples. Neste trabalho, optou-se por manter a chamada dessas sub-rotinas.

Para medir o tempo de cada rotina foi utilizada a rotina `cpu_time()`. Foram realizadas 25 medições para se obter uma média aritmética do tempo gasto. Com essas medições, também se calculou a variância e desvio padrão dos tempos obtidos.

Como estudo de caso, utilizou-se uma matriz de dimensão discreta de 100×124 pontos. O passo de tempo discreto foi de 0.01. O intervalo de tempo considerado nas simulações foi de 0.00 até 0.04. A geração de código binário foi feita utilizando o compilador `pgf90` (`nvfortran`) do `HPC_SDK` toolkit da NVidia, versão 23.1. A compilação utilizou as diretivas: `-O3 -mp=multicore -Minfo=all`. Os testes foram realizados em uma *workstation* com 2 processadores com 8 cores físicos cada.

4. Resultados

Todas as medições de tempo encontram-se na Figura 2. Nela são apresentadas as medições de tempo usando 1, 2, 4, 8, 16 e 32 threads, para ambas as implementações. Os resultados 1V representam os tempos utilizando diretivas, sem nenhuma modificação de código. Os resultados 2V foram coletados do código paralelizado cujas chamadas das sub-rotinas foram modificadas. Os resultados das rotinas paralelizadas apresentaram redução de tempo de execução em relação à execução sequencial para ambas versões paralelas. O tempo de execução diminui à medida que mais threads são utilizadas. O speedup usando 16 threads (número de cores físico) é em torno de 7, 4.

Os resultados da paralelização usando (`do` e `teams`) podem ser considerados semelhantes, mesmo que em alguns casos uma das implementações seja um pouco mais rápida que a outra, pois a diferença de tempo está dentro do desvio padrão obtido ($< 2\%$ em relação à média). Já as diferenças entre os resultados das versões 1V e 2V variam de 30% a 55%. Observa-se que mesmo na versão sequencial, o tempo de execução diminui consideravelmente para o código modificado. Tal modificação não havia sido incluída nas avaliações em [Lucca et al. 2023].

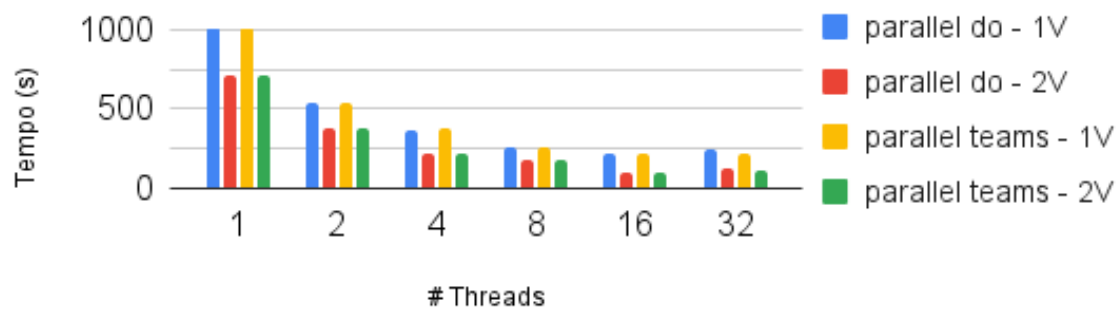


Figura 2. Tempo de Execução

5. Considerações Finais

Neste artigo foi feita uma avaliação entre as diretivas `do` e `teams` de OpenMP em uma aplicação de simulação de meios porosos. Aplicações deste tipo demandam de um tempo de processamento significativo. Os resultados obtidos mostram que `teams` pode ser uma boa alternativa em detrimento do clássico paralelismo de tarefas. Neste trabalho também foi explorado a modificação no código-fonte, de maneira a tornar a computação mais eficiente. Para isso, alterou-se a chamada da sub-rotina `Quick`, incluindo o código da mesma diretamente nas rotinas que a invocavam.

Como trabalhos futuros pretende-se avaliar as demais diretivas de OpenMP que possibilitam instanciar de threads ou distribuir a carga de trabalho de outra forma.

Referências

- da Silva, H. U., Lucca, N., Schepke, C., de Oliveira, D. P., and da Cruz Cristaldo, C. F. (2022a). Parallel OpenMP and OpenACC Porous Media Simulation. *The Journal of Supercomputing*.
- da Silva, H. U., Schepke, C., da Cruz Cristaldo, C. F., de Oliveira, D. P., and Lucca, N. (2022b). An Efficient Parallel Model for Coupled Open-Porous Medium Problem Applied to Grain Drying Processing. In Gitler, I., Barrios Hernández, C. J., and Meneses, E., editors, *High Performance Computing*, pages 250–264, Cham. Springer International Publishing.
- da Silva, H. U., Schepke, C., Lucca, N., da Cruz Cristaldo, C. F., and de Oliveira, D. P. (2022c). Parallel OpenMP and OpenACC Mixing Layer Simulation. In *2022 30th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, volume 1, pages 181–188.
- de Oliveira, D. P. (2020). Fluid Flow Through Porous Media with the One Domain Approach: A Simple Model for Grains Drying. Dissertação de mestrado, Universidade Federal do Pampa.
- Lucca, N., Schepke, C., and Tremarin, G. D. (2023). Parallel Directives Evaluation in Porous Media Application: A Case Study. In *2023 31th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, volume 1.
- OpenMP (2023). The OpenMP API Specification for Parallel Programming.