Uso de Paralelismo em uma Aplicação de Meio Poroso

Ronaldo Canofre M. dos Santos, Mauricio Martinuzzi Fiorenza, Daniel Chaves Temp, Pablo Brauner Viegas, Claudio Schepke

[1]Programa de Pós-Graduação em Engenharia de Software (PPGES)
Universidade Federal do Pampa (UNIPAMPA)
Alegrete – RS – Brasil

{canofresantos,mauriciofiorenza, claudioschepke}@unipampa.edu.br

danieltemp,pabloviegas}.aluno@unipampa.edu.br

Resumo. *O aumento de desempenho dos processadores atuais, deve-se ao número cada vez maior de cores. Este trabalho apresenta melhorias aplicadas ao código de um projeto denominado Poros, utilizando o paralelismo através da biblioteca OpenMP. Dessa forma, com a análise da execução sequencial em comparação com as execuções paralelas, foi possível verificar uma melhoria de cerca de 23% no tempo de execução em uma arquitetura multicore.*

1. Introdução

O particionamento de um problema visando a sua execução de forma mais otimizada, não se restringe somente a problemas computacionais, sendo facilmente aplicável a problemas práticos das mais diversas áreas. Nesta linha, é possível buscar a utilização de paralelismo aplicado a problemas já conhecidos, como por exemplo, a adaptação de soluções aplicadas a dinâmica de fluidos, para uma separação de fluxo de grãos [de Oliveira 2020].

Atualmente, o trabalho em desenvolvimento, denominado projeto *Poros*, realiza esta resolução de forma sequencial, utilizando como base as equações de Navier-Stokes [Constantin and Foias 1988]. No entanto, o desempenho obtido fica aquém das necessidades de tempo e poder de processamento existentes, sendo desejada a otimização da execução na busca por ganhos de eficiência na resolução do problema. Dessa forma, este trabalho apresenta melhorias aplicadas ao código, com a utilização de paralelismo através da biblioteca OpenMP [Chandra et al. 2001].

No decorrer do trabalho, discutimos as alterações/paralelizações realizadas na Seção 2, os resultados obtidos na Seção 3 e as considerações finais do trabalho na Seção 4.

2. Implementação

A implementação realizada iniciou-se com a adaptação do código fonte para utilização do OpenMP, incluindo também como opções de compilação, as *flags* "-O3" para otimização e "-mp" para interpretar as diretivas e *pragmas* de programação paralela de memória compartilhada, tendo sido utilizado o compilador "pgf90" do pacote Nvidia HPC SDK[1].

A otimização do código foi realizada sobre o arquivo que reúne as sub-rotinas chamadas durante a execução, sendo concentrada a aplicação de paralelização nos principais laços existentes, observando-se o controle de compartilhamento das variáveis, bem como a dependência temporal das mesmas. Já as execuções foram realizadas como sendo sempre uma nova instância, buscando manter constante o número de iterações.

[1]https://www.pgroup.com/index.htm

3. Resultados

Para validação e coleta de resultados, realizou-se a execução do código sequencial (com uma *thread*) em comparação com execuções paralelas com 2, 3, 4, 5 e 6 *threads*, obtendo como resultado a média simples de três execuções para cada instância analisada. Tais execuções foram realizadas em um *host* físico executando Ubuntu Desktop 20.04, equipado com processador Intel(R) Xeon(R) CPU E5-2609 com 1.90GHz, 16 GB de memória e HD 140 GB, em um ambiente de ensino não controlado. Para coleta do tempo total de execução foi utilizado o comando *time*[2]. A Tabela 1 mostra os valores temporais coletados em cada execução, bem como a médias obtidas para cada operação.

Tabela 1. Comparativo de execuções sequencial e paralelas

	1 Thread	**2 Threads**	**3 Threads**	**4 Threads**	**5 Threads**	**6 Threads**
Execução 1	78,510s	60,486s	59,445s	55,640s	52,710s	50,090s
Execução 2	78,382s	60,713s	58,621s	55,315s	52,808s	50,140s
Execução 3	78,248s	60,953s	58,247s	55,694s	52,498s	50,296s
Média	**78,380s**	**60,717s**	**58,771s**	**55,550s**	**52,672s**	**50,175s**

Os resultados demonstram um ganho de 23% no tempo de execução, comparando a execução sequencial com a paralela com apenas 2 *threads*. A melhora no desempenho se confirma conforme são aumentadas os números de processadores, chegando ao máximo de 36% de melhoria do desempenho com 6 *threads* paralelas.

4. Conclusão

Analisando as execuções realizadas após a aplicação das otimizações com paralelismo via OpenMP, fica visível uma considerável melhoria no tempo de execução, demonstrando que o objetivo foi alcançado, apresentando como resultando uma aplicação com efetiva otimização.

Como trabalhos futuros, pretende-se buscar alternativas de codificação que permitam alcançar resultados melhores dos que os já obtidos, através de técnicas de execução em GPU como OpenACC ou CUDA [Hoshino et al. 2013].

Referências

Chandra, R., Dagum, L., Kohr, D., Menon, R., Maydan, D., and McDonald, J. (2001). *Parallel programming in OpenMP*. Morgan kaufmann.

Constantin, P. and Foias, C. (1988). *Navier-Stokes Equations*. University of Chicago Press.

de Oliveira, D. P. (2020). Fluid Flow Through Porous Media With The One Domain Approach: A Simple Model For Grains Drying. Master's thesis, Universidade Federal do Pampa, Alegrete.

Hoshino, T., Maruyama, N., Matsuoka, S., and Takaki, R. (2013). CUDA vs OpenACC: Performance case studies with kernel benchmarks and a memory-bound CFD application. In *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, pages 136–143. IEEE.

[2] `https://man7.org/linux/man-pages/man1/time.1.html`