Avaliação de Desempenho da Implementação Baseada em Tarefas e Fluxo de Dados do Método de Lattice-Boltzmann

Gabriel Freytag[1], **João V. F. Lima**[1], **Claudio Schepke**[2]

[1]Universidade Federal de Santa Maria (UFSM) – Santa Maria – RS – Brasil

[2]Universidade Federal do Pampa (UNIPAMPA) – Campus Alegrete – RS – Brasil

{gfreytag,jvlima}@inf.ufsm.br, schepke@unipampa.edu.br

Resumo. *Várias classes de fluidodinâmica computacional utilizadas em diversas áreas normalmente demandam um grande poder computacional. Neste trabalho apresentamos uma implementação do método de Lattice-Boltzmann baseada em tarefas com dependências OpenMP com speedup de 30.20 em relação à implementação baseada no paralelismo de iterações de laços de repetição.*

1. Introdução

Arquiteturas *multicore* de memória compartilhada são comumente uma das alternativas utilizadas na aceleração do processamento de aplicações que demandam um grande poder computacional, um exemplo disso são aplicações de fluidodinâmica computacional. O método de Lattice-Boltzmann (LBM) é uma aplicação de fluidodinâmica computacional que se destaca dos demais métodos convencionais pela simplicidade e propensão ao paralelismo [Chen and Doolen 1998]. Na literatura a paralelização do método em arquiteturas de memória compartilhada comumente é realizada baseada no paralelismo de iterações de laços de repetição e, dessa forma, o objetivo deste trabalho é avaliar o desempenho da paralelização do LBM utilizando tarefas OpenMP com dependências, introduzidas na versão 4.0, em uma arquitetura *multicore* de memória compartilhada do tipo NUMA em comparação à uma implementação baseada no paralelismo de iterações. Ambas as implementações são extensões da implementação de Schepke e Diverio (2007).

2. Implementações

Na implementação apresentada por Schepke e Diverio (2007) existem seis funções que são executadas por cada nó sobre o seu subdomínio e que aplicam suas operações sobre os dados armazenados em duas matrizes tridimensionais, uma principal e outra temporária. Já na implementação baseada em tarefas cada função foi transformada em uma tarefa por meio da cláusula `task` da API OpenMP. Além disso, definiu-se as dependências de cada tarefa por meio da cláusula `depend` de acordo com as operações de leitura e escrita que a tarefa realiza em ambas as matrizes de um subdomínio. Apesar das dependências das seis tarefas às impedir de serem executadas paralelamente em um mesmo subdomínio, não às impede de serem executadas paralelamente em subdomínio distintos, com exceção de uma tarefa. Como a tarefa de cópia das bordas dos subdomínios depende da leitura da matriz temporária de subdomínios vizinhos e escrita na matriz temporária do subdomínio para o qual foi gerada, essa tarefa somente pode ser executada paralelamente em subdomínios não vizinhos. Para remover essa restrição e permitir que todas sejam executadas paralelamente nos diferentes subdomínios desenvolveu-se uma segunda implementação que utiliza matrizes auxiliares (*buffers*) para a cópia das bordas, onde adicionou-se uma sétima

tarefa que copia as bordas dos vizinhos para os *buffers* que posteriormente são copiadas pela tarefa de cópia das bordas dos *buffers* para a matriz temporária do subdomínio.

3. Resultados e Discussões

Os experimentos foram realizados em uma máquina com 8 nós NUMA, cada nó com um processador Intel Xeon SandyBridge E5-4617 de 6 núcleos de processamento e 64 GB de memória RAM, um total de 48 núcleos e 512 GB de memória. Além disso, os tamanhos dos reticulados utilizados foram 32, 64, 96, 128, 192, 256 (múltiplos de 32). Nas implementações baseadas em tarefas os reticulados foram particionados em 2, 4, 8 e 16 subdomínios por dimensão. Cada experimento foi repetido 10 vezes. Na Figura 1 é apresentado a média dos melhores tempos de execução obtidos em cada reticulado, independentemente do particionamento utilizado (nas implementações baseadas em tarefas), das implementações baseada no paralelismo de iterações de laços de repetição (OpenMP), baseada no paralelismo de tarefas com dependências (OpenMP Tasks) e baseada no paralelismo de tarefas com dependências e com *buffers* (OpenMP Tasks Buffer) com ambos os tamanhos de reticulado utilizando 12, 24, 36 e 48 núcleos (2, 4, 6 e 8 nós NUMA). Como é possível observar, especialmente a partir do tamanho de reticulado 128, o desempenho da implementação baseada em tarefas com *buffers* se sobressai expressivamente em relação às demais. Além disso, com 24, 36 e 48 núcleos a implementação baseada em tarefas obteve os maiores tempos de execução em todos os tamanhos de reticulado.

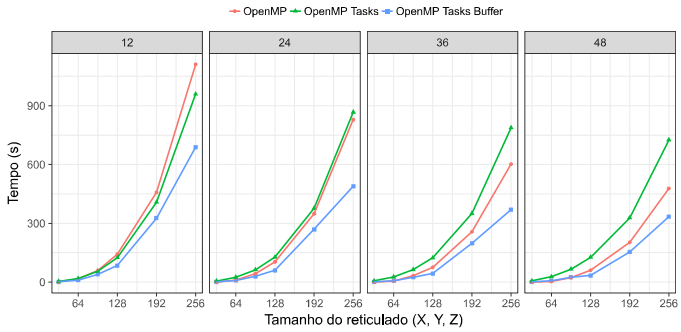

Figura 1. Tempos de execução por tamanho do reticulado em ambas as implementações.

4. Conclusão e Trabalhos Futuros

Como pôde ser observado, a implementação baseada em tarefas com *buffers* obteve os melhores tempos de execução. No reticulado de tamanho 256 e 48 núcleos obteve um *speedup* de 30.20 em relação à implementação baseada no paralelismo de iterações. Como um trabalho futuro sugere-se a análise do desempenho de uma implementação baseada em tarefas OpenMP com dependências para arquiteturas de memória distribuída.

Referências

Chen, S. and Doolen, G. D. (1998). Lattice boltzmann method for fluid flows. *Annual review of fluid mechanics*, 30(1):329–364.

Schepke, C. and Diverio, T. A. (2007). Distribuição de dados para implementações paralelas do método de lattice boltzmann. Master's thesis, Universidade Federal do Rio Grande do Sul.