Influência das Estratégias de Particionamento de Dados no Desempenho de Aplicações Numéricas Paralelas*

Claudio Schepke, Tiarajú Asmuz Diverio, Nicolas Maillard
Grupo de Processamento Paralelo e Distribuído
Instituto de Informática – Universidade Federal do Rio Grande do Sul
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil
{cschepke,diverio,nicolas}@inf.ufrgs.br

Resumo

The use of parallel implementations in numerical applications is recurrent. To parallel such applications makes possible the simulation of cases where monoprocessed machines have memory restrictions or cases where the results must be calculated in a limited time. However, the form as the data distribution is made can influence the performance of the application because numerical applications generally have data dependences. Thus, the present article evaluate the performance of different strategies of data partitioning in parallel numerical applications. Some data distributions had been tested in parallel implementations of the Lattice Boltzmann Method as case study. The results show that the difference of performance between improved strategies over simple strategies are significant.

1. Introdução

O uso de arquiteturas paralelas tem sido freqüentemente adotado para a resolução de aplicações numéricas oriundas da discretização de modelos físicos-matemáticos. A simulação destes modelos demanda uma grande capacidade de processamento e armazenamento dos dados, tanto pelo tamanho dos sistemas, quanto pela dinamicidade das operações, que são realizadas através de sussessivas iterações para alcançar a estabilidade ou precisão desejada. Paralelizar tais aplicações implica, desta forma, tornar viável a obtenção de soluções, além de aumentar a eficiência dos programas em relação às implementações monoprocessadas.

Uma das estratégias para o desenvolvimento de aplicações paralelas é fazer uso do Paralelismo de Dados [5]. O Paralelismo de Dados ou Decomposição de Domínios consiste em dividir e atribuir os dados aos processadores, de maneira que cada processador atue somente sobre o seu conjunto de dados. Para tanto, é necessário apenas um único fluxo de instrução.

Um aspecto importante no Paralelismo de Dados é a forma de particionar os dados. Uma boa distribuição para arquiteturas homogêneas ocorre quando todos os processadores têm a mesma carga de trabalho. No caso de todas as operações aplicadas sobre os elementos do conjunto de dados serem as mesmas e considerando que a capacidade de processamento dos processadores é semelhante entre si, a melhor forma de distribuir os dados é particionar o problema pelo número total de processadores disponível. No entanto, a maneira como isso ocorre pode influenciar no desempenho das aplicações devido a uma série de fatores tais como: distribuição e acesso aos dados em memória, taxa de acerto de *cache*, dependência de dados e equilíbrio de carga.

Uma maneira de aumentar o desempenho das aplicações paralelas é utilizar estratégias de particionamento de dados mais eficientes tais como o particionamento em blocos. Isso pode ser feito dividindo-se os dados dispostos em estruturas multidimensionais coordenadas em duas ou mais dimensões. O uso de algoritmos paralelos com particionamento em bloco tem como objetivo reduzir a movimentação de dados entre os diversos níveis de hierarquia da memória. Assim, é possível acessar um maior número de regiões de memória contínuas, bem como evitar falhas de acesso a *cache*. Em sistemas de memória distribuída, onde existe dependência de dados, é possível diminuir ainda os custos de comunicação interprocessos.

Neste contexto, o objetivo deste trabalho consiste em analisar o desempenho paralelo de diferentes formas de particionamento de dados, especialmente em blocos. Para tanto foram feitas implementações paralelas do Método de Lattice Boltzmann (MLB), tanto de um modelo bidimensional, quanto de um modelo tridimensional [9, 1, 8]. Os testes foram feitos em uma arquitetura de *cluster*. Foram testadas diferentes configurações de mapeamento de dados aos

* Apoio: CNPq

processadores usando, com isso, diferentes estratégias de distribuição de dados.

2. Particionamento em Blocos

Uma maneira eficiente de distribuição de dados entre processos paralelos é utilizar o particionamento em blocos [4, 6, 7]. O particionamento em blocos é definido como a divisão dos dados em mais de uma dimensão em estruturas multidimensionais de dados. Assim, um conjunto de dados de estrutura tridimensional, por exemplo, pode ser dividido em blocos tanto em duas como em três dimensões.

Estratégias de particionamento de dados em blocos são freqüentemente adotadas em operações numéricas paralelas como forma de agregar desempenho as operações. Por causa disso, muitas propostas podem ser encontradas, especialmente para métodos de resolução de sistemas lineares [3]. As modificações feitas sobre técnicas de resolução tradicionais tornaram possível o surgimento de novos algoritmos, os quais são mais indicados para as arquiteturas de computadores atualmente existentes. Cabe destacar também que o algoritmo de resolução de sistemas lineares Linpack, utiliza o particionamento em blocos para explorar ao máximo as arquiteturas paralelas, servindo, por isso, como *benchmark* de avaliação de desempenho de máquinas paralelas [2].

3. Implementação Paralela

3.1. Implementação do Algoritmo do MLB

Neste trabalho foram feitas implementações paralelas do Método de Lattice Boltzmann (MLB). O MLB é um método numérico iterativo discreto utilizado para a modelagem e simulação mesoscópica de fluxos de fluidos [9]. A estrutura principal do algoritmo do MLB é constituido de um laço de repetição no qual são feitas operações de movimentação dos dados, simulando um fluxo de fluido. Tais operações consistem na propagação e relação das partículas, além de cálculos de valores macroscópicos e tratamento das condições de contorno (*Bounce Back*, conforme apresentado na Figura 1. O critério de parada do laço principal pode ser determinado pelo número de iterações ou por outro fator, como a estabilização do fluxo.

Para as implementações foram adotados dois dos modelos de reticulados mais utilizados do método: um bidimensional, com 9 direções de propagação das partículas, e outro tridimensional, com 19 direções de propagação [1]. Tais modelos são conhecidos, respectivamente, por D2Q9 e D3Q19, sendo estes ilustrados na Figura 2.

A implementação feita consistiu em obter valores macroscópicos, tais como velocidade e pressão, para um fluxo de fluido através de um canal com obstáculos. Como

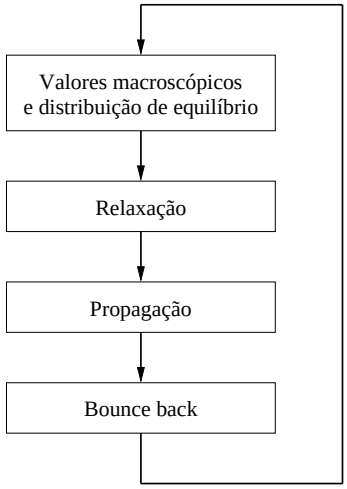

Figura 1. Algoritmo do MLB

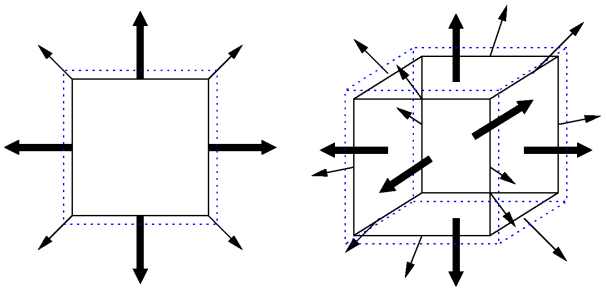

Figura 2. Modelos de reticulado bidimencional e tridimencional

parâmetro de entrada são repassadas algumas informações ao programa através de dois arquivos: um que contém informações genéricas tais como as propriedades macroscópicas e parâmetros de configuração, e o segundo que contém a estrutura de pontos do reticulado (limites e barreiras). Essas informações são armazenadas em duas estruturas de dados independentes.

3.2. Paralelização com MPI

A implementação paralela do método foi feita em linguagem C através do modelo *Single Program Multiple Data* (SPMD). Os dados operados foram divididos entre cada um dos processadores utilizados. Cada processador, portanto, é responsável pela execução de um único processo. Neste caso cada processo atua em uma região de domínio do problema. Como existe um fluxo de informações, é necessário que ao final de cada iteração do método exista comunicação entre os processos. Assim, os dados das fronteiras são trocados entre os processos vizinhos.

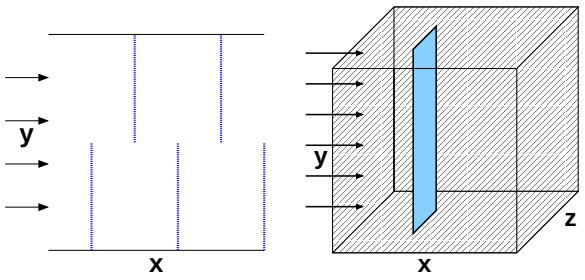

Figura 3. Disposição das barreiras no reticulado bidimensional e tridimensional

A comunicação interprocessos foi feita através da biblioteca de comunicação Message Passing Interface (MPI), a qual prove muitas formas de comunicação e recursos para a manipulação de estruturas de dados. Neste trabalho foram utilizadas operações de MPI para mapear algumas regiões de memória a fim de transmitir esses dados de forma contínua. Para tanto, foram utilizados as funções *MPI_Address* para o mapeamento das bordas dos subdomínios, *MPI_Type_extent* para estender um tipo de dados que armazena os dados mapeados e *MPI_Type_contiguous* para definir que o tipo de dados seja contínuo.

3.3. Estudo de Casos

Os casos de estudo escolhidos para avaliar as implementações paralelas consistem em simulações de fluxos de fluido cruzando canais com obstáculos.

O primeiro caso é um fluxo bidimensional com tamanho de reticulado igual a 512 X 512 elementos. A distribuição dos obstáculos é composta de 5 barreiras dispostas ciclicamente ao longo do eixo x, conforme indicado na ilustração da esquerda da Figura 3. O tamanho de cada barreira é igual a metade do número de pontos que compõem os elementos da dimensão y. Já a distância entre cada uma das barreiras também é fixa, tendo-se utilizado para isso o valor de $1/5$ do tamanho total de pontos da dimensão x.

O segundo caso é um fluxo que cruza um canal tridimensional, cujo tamanho do reticulado é de 128 X 128 X 128 elementos. Para esse problema existe uma única região retangular com obstáculos, a qual está disposta na parte anterior do eixo x, na posição definida por $1/3$ do tamanho total da dimensão x. Essa região engloba todos os pontos do eixo y que se encontram na parte mais central em relação ao eixo z. A largura dessa parte central em torno do eixo z possui uma extensão de $1/3$ do tamanho total de elementos que compõem essa dimensão. Graficamente o problema pode ser limitado conforme é exibido na ilustração da direita na Figura 3. Nessa figura, além dos obstáculos, são destacadas (listras) as faces que definem o canal, ficando duas faces abertas para a circulação do fluxo.

Os resultados computacionais foram obtidos através da média de 10 execuções, sendo que as duas piores execuções foram excluídas. O desvio padrão para os valores que compõem as médias ficaram abaixo dos 1% em relação à média total. Os tempos foram medidos usando o cluster *labtec* do Instituto de Informática da Universidade Federal do Rio Grande do Sul. A captura dos tempos ocorreu através da função *MPI_Wtime*.

4. Resultados

4.1. Modelo 2D

A divisão em duas dimensões foi feita particionando-se o reticulado entre 36 processos nas mais diferentes configurações possíveis. Os tempos de execução obtidos para esses casos são apresentados na Figura 4. Em relação ao ganho de desempenho, o tempo de execução do particionamento bidimensional foi até 8,23% menor do que o melhor caso da divisão unidimensional. Estes resultados foram semelhantes para outras configurações de distribuição de dados usando diferentes números de processos. Outro aspecto importante observado é de que sub-reticulados com uma estrutura mais quadrada alcançaram melhores resultados em virtude dos mesmos terem um custo de comunicação menor.

4.2. Modelo 3D

A divisão em múltiplas dimensões do reticulado de 128 X 128 X 128 pontos foi feita com diferentes combinações de particionamento usando 27 processadores. A Figura 5 compara os tempos de execução obtidos. Considerando o melhor tempo obtido através de uma das três formas de particionamento (particionamento nas dimensões x, y ou z) das execuções seqüenciais possíveis, isto é, $387,13\ s$, pode-se observar que houve uma redução de até $14,95\%$ para a melhor forma de particionamento em bloco. Da mesma forma que a implementação bidimensional, a divisão do reticulado de maneira a deixar cada sub-reticulado com uma estrutura mais quadrática ou cúbica apresentou os melhores resultados paralelos.

5. Conclusão

Este artigo apresentou uma análise do desempenho de diferentes estratégias de particionamento de dados em uma aplicação numérica paralela. Com base nos estudos de caso, percebe-se que o uso de estratégias de distribuição de dados que não se limitam ao particionamento unidimensional podem incrementar o desempenho paralelo das aplicações.

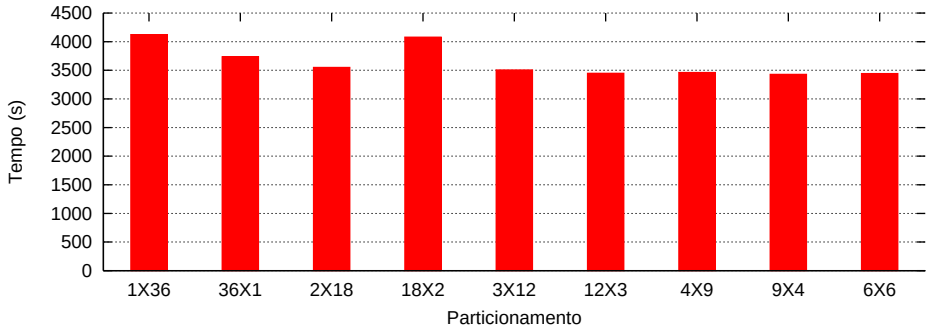

Figura 4. Tempo de execução de diferentes particionamentos do modelo 2D

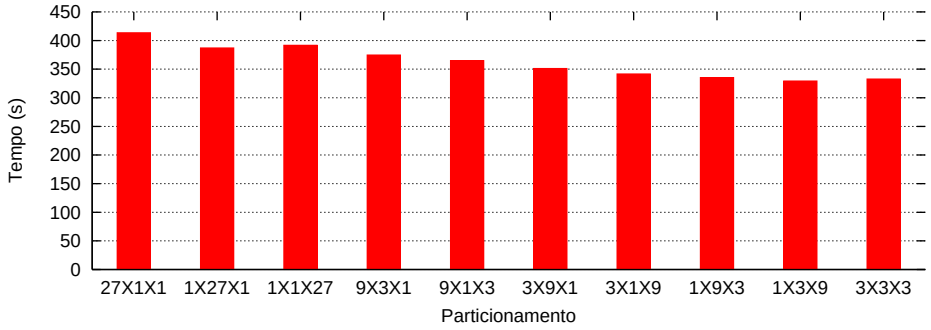

Figura 5. Tempo de execução de diferentes particionamentos do modelo 3D

Implementações usando particionamento em blocos proporcionaram um maior desempenho em relação à paralelização linear usando Decomposição de Domínios.

Outro fator observado em relação aos resultados obtidos foi que a divisão dos dados em blocos garante um menor custo de comunicação e, conseqüentemente, um menor tempo de execução total do método. A organização dos subreticulados em regiões mais ou menor quadradas ou cúbicas mostrou-se mais adequada considerando-se esse aspecto, visto que o volume de dados transmitido acabava sendo menor, bem como tornava a sincronização dos dados entre os processos mais rápida.

Referências

[1] S. Chen and G. D. Doolen. Lattice Boltzmann Method for Fluid Flows. *Annual Review of Fluid Mechanics*, 30:329–364, 1998.

[2] J. Dongarra. The linpack benchmark: An explanation. In *Proceedings of the 1st International Conference on Supercomputing*, pages 456–474, London, UK, 1988. Athens, Greece, [S.l.]: Springer-Verlag.

[3] J. Dongarra, I. Foster, G. Fox, W. Gropp, K. Kennedy, L. Torczon, and A. White, editors. *The Sourcebook of Parallel Computing*. Elsevier, November 2002.

[4] E. Elmroth, F. Gustavson, I. Jonsson, and B. Kågström. Recursive Blocked Algorithms and Hybrid Data Structures for Dense Matrix Library Software. *SIAM Rev.*, 46(1):3–45, Mar. 2004.

[5] I. Foster. *Designing and Building Parallel Programs: Concepts and tools for Parallel Software Engineering*. Addison Wesley, Reading, MA, 1995.

[6] I. Jonsson and B. Kågström. Recursive blocked algorithms for solving triangular systems - Part I: one-sided and coupled Sylvester-type matrix equations. *ACM Trans. Math. Softw.*, 28(4):392–415, 2002.

[7] M. D. Lam, E. E. Rothberg, and M. E. Wolf. The cache performance and optimizations of blocked algorithms. In *Proceedings of the 4th International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS*, pages 63–74, Santa Clara, California, United States, 1991. New York: ACM Press.

[8] C. Schepke. Distribuição de Dados para Implementações Paralelas do Método de Lattice Boltzmann. Dissertação (Mestrado em Ciência da Computação), Instituto de Informática, UFRGS, Porto Alegre, Março 2007.

[9] S. Succi. *The Lattice Boltzmann Equation for Fluid Dynamics and Beyond*. Oxford University Press, New York, USA, 2001.