UNIVERSIDADE FEDERAL DO PARANÁ

RELATÓRIO DE ATIVIDADES DE PESQUISA

Análise Preliminar da Eficiência de Diferentes Métodos de Paralelização na Solução Numérica da Equação de Laplace

Prof. Ricardo Carvalho de Almeida

Julho/2009

1- Objetivo

O objetivo do presente relatório é descrever as atividades realizadas para a avaliação preliminar da eficiência computacional obtida por meio do emprego de diferentes técnicas de paralelização de códigos computacionais para a solução numérica da equação de Laplace.

2- Definição do Problema

Na dinâmica dos fluidos computacional é bastante comum a necessidade de solucionar problemas de equilíbrio, que são representados por equações elípticas. Na análise e desenvolvimento de técnicas numéricas para a solução de problemas de equilíbrio emprega-se frequentemente a equação de Laplace bidimensional, considerada como um protótipo desse tipo de problema, que é expressa por:

$$\frac{\partial^2 \varphi}{\partial x^2} + \frac{\partial^2 \varphi}{\partial y^2} = 0$$

onde φ é a variável dependente, e x, y são coordenadas espaciais.

Entre os diversos métodos empregados para a solução da equação de Laplace, pode-se citar o método Gauss-Seidel, também conhecido como método de relaxação sucessiva, que se baseia na aplicação de correções sucessivas à solução numérica em cada ponto de grade (nó da malha), sendo os valores atualizados nos pontos de grade imediatamente utilizados para a atualização do próximo nó adjacente.

Em discretização por diferenças finitas centradas, empregando-se ordenamento lexicográfico, a aproximação obtida para o valor da variável dependente em um ponto de grade (i,j), na iteração ($k+1$), é obtida por:

$$\varphi_{i,j}^{k+1} = \varphi_{i,j}^{k} - \frac{R}{\beta},$$

$$R = \varphi_{i,j}^{k}\beta - \left(\frac{\varphi_{i+1,j}^{k} - \varphi_{i-1,j}^{k}}{\Delta x^2} + \frac{\varphi_{i,j+1}^{k} - \varphi_{i,j-1}^{k}}{\Delta y^2} \right),$$

$$\beta = \frac{2(\Delta x^2 + \Delta y^2)}{(\Delta x \Delta y)^2}$$

onde Δx, Δy representam os espaçamentos de grade nas direções x e y, respectivamente.

As atualizações de todo o domínio são processadas em ciclos, ou iterações, que são interrompidas quando a solução numérica converge, atendendo a um certo critério de tolerância baseado, em geral, na diferença dos valores das variáveis entre iterações sucessivas.

Em alguns problemas, particularmente aqueles em que são utilizadas malhas densas, com muitos nós, a convergência pode ser extremamente lenta, levando a um elevado custo computacional para obtenção da solução numérica. Neste estudo serão analisadas algumas técnicas de paralelização que podem ser empregadas para o aumento da eficiência computacional na solução desse problema.

3- O Problema da Paralelização do Método Gauss-Seidel

Na implementação de algoritmos paralelos para a solução de problemas numéricos há vários aspectos que devem ser observados, particularmente a chamada "dependência dos dados".

Inicialmente, definimos que uma posição de memória em um programa é qualquer entidade à qual o programa pode atribuir um valor, tal como um inteiro, valor de ponto flutuante ou caracter. Todas as vezes que um comando em um programa lê ou escreve uma posição de memória e um outro comando lê ou escreve na mesma posição de memória, e pelo menos um desses comandos escreve naquela posição, diz-se que há uma dependência dos dados entre os dois comandos naquela posição de memória.

Quando se deseja paralelizar um programa, dependências de dados são importantes, pois toda vez que elas existem entre dois comandos, eles não podem ser executados em paralelo, pois causariam uma "competição (ou corrida) dos dados" (data race).

Um programa em paralelo possui um competição de dados sempre que dois ou mais comandos possam ler ou escrever em uma certa posição de memória ao mesmo tempo, e pelo menos um deles escreve naquela posição. Em geral, competição de dados causa erros nos resultados de programas paralelos, pois eles poderão produzir resultados

diferentes de um programa equivalente sequencial (ou serial), executado em um único processador.

Para exemplificar essa situação, consideremos o seguinte loop, que se deseja paralelizar:

```
do i = 2,n
   a(i) = a(i) + a(i-1)
end do
```

Suponhamos que dois ou mais processadores estejam executando o loop, cada um realizando uma iteração (ou seja, calculando para um valor de i), que $n = 3$, e que os três primeiros elementos do array *a* foram inicializados com os valores 1, 2 e 3.

Após a execução serial (correta) do loop, os três primeiros valores do array seriam 1, 3 e 6. No entanto, em uma execução paralela, seria possível a atribuição do valor 3 ao elemento *a*(2) na primeira iteração executada por um processador, ocorrer tanto antes quanto depois da leitura de *a*(2) da segunda iteração, executada por um outro processador (os dois comandos estão competindo entre si). Caso a atribuição (escrita) da posição de memória do elemento *a*(2) ocorra após a leitura, o elemento *a*(3) receberá um valor incorreto, igual a 5. Nesse caso específico, como cada iteração do loop é executada em paralelo, mas dentro de cada iteração os comandos no loop são executados em sequência, existe uma dependência entre os comandos executados em diferentes iterações do loop. Este tipo de dependência é chamada de "carregada pelo loop" (loop carried).

O método Gauss-Seidel possui uma dependência dos dados, em virtude da atualização em cada ponto de grade depender do valor atualizado em um ou mais pontos de grade adjacentes. Caso processadores em paralelo estejam em iterações diferentes em pontos de grade adjacentes, a atualização do valor em um ponto de grade poderá estar utilizando o valor de um ponto adjacente ainda não atualizado. Devido às suas características, o método Gauss-Seidel paralelizado, mesmo com a dependência dos dados, eventualmente, convergirá para a solução desejada. No entanto, o número de iterações necessárias poderá ser diferente do que seria obtido em um código serial.

4- Técnicas de Paralelização sem Dependência dos Dados

Há possibilidade de paralelização do método Gauss-Seidel evitando-se a dependência dos dados, alterando-se o método de ordenamento, de lexicográfico, para um dos seguintes: red-black ou zebra.

4.1 – O método red-black

No método red-black cada elemento do array (nó ou ponto de grade) é conceitualmente colorido com as cores vermelho (red) ou preto (black), da seguinte forma:

- se (i+j) for par, o ponto (i,j) é red;
- se (i+j) for ímpar, o ponto (i,j) é black.

O método red-black atualiza os elementos vermelhos e pretos alternadamente. Note que elementos de uma cor são cercados por quatro elementos (norte, sul, leste e oeste) de outra cor. Dessa forma, os elementos de uma cor podem ser atualizados independentemente dos elementos de outra cor em cada ciclo de varredura da grade, possibilitando a paralelização sem dependência dos dados.

4.2 – O método zebra

No método zebra as colunas dos arrays são coloridas alternadamente, utilizando duas cores, preto e branco, por exemplo, Inicialmente, todos os elementos brancos são atualizados. Como cada coluna será varrida sequencialmente, da base para o topo (ou vice-versa), e os valores a leste e a oeste de cada ponto de grade não se alteram durante a varredura das colunas de uma cor, esse método também permite a paralelização sem dependência dos dados.

5 – Materiais e Métodos

Neste estudo preliminar, foram testadas e comparadas diferentes técnicas de paralelização do método Gauss-Seidel, para a solução da equação de Laplace, nas seguintes condições:

- Condições de contorno: $\varphi(x,y)$=xy, para x=0 e y=0;
- Domínio computacional: x = [0,1], y =[0,1];
- Aproximação inicial no interior do domínio: $\varphi^0(i,j) = 0$

Em todos os experimentos foi adotado como critério de convergência a condição da diferença relativa local entre iterações sucessivas ser igual ou inferior a 10^{-10}.

Foi empregado o compilador Fortran Visual Studio v. 6.1, 64 bits, com otimização para maximização da velocidade e precisão dupla. Para a paralelização foi empregado o modelo de programação paralela OpenMP.

O hardware utilizado foi um computador com 2 processadores Quad-Core Intel Xeon X5355, 2.66 GHz.

Os métodos de implementação dos algoritmos para a solução foram os seguintes:

a) serial red-black e serial zebra (ambos como resultados de referência);
b) paralelização em granulação fina (fine-grain), utilizando a instrução *parallel do*, em que o compilador determina a ordem (ordenamento) de execução do loop de varredura da grade; e
c) regiões paralelas red-black e regiões paralelas zebra, onde o domínio é explicitamente dividido em tantos blocos quantos sejam os processadores disponíveis, ficando cada um deles responsável por uma região.

Nos experimentos em que houve processamento paralelo, os processos foram divididos apenas nas instruções referentes às varreduras das grades e correções/atualizações dos valores das variáveis, para cada ciclo de atualização (iteração), utilizando sincronização implícita, controlada pelo compilador.

Os experimentos foram realizados para o domínio citado, com duas resoluções diferentes: 401×401 pontos de grade e 201×201 pontos de grade.

5.1 – Experimentos com a grade de 401×401 pontos

A tabela I apresenta os resultados obtidos nos experimentos com a grade de 401×401 pontos. O tempo de processamento refere-se ao tempo real (wall clock time).

Dos resultados apresentados na tabela I, observa-se que o método red-black serial, utilizando um processador, foi o que demandou maior tempo de computação. Dessa forma, a eficiência computacional obtida pelos demais métodos foi avaliada como a razão entre o tempo de processamento do método red-black serial e o tempo despendido pelo método em avaliação. Essa razão é definida como o fator de aceleração (speedup).

Tabela I – Resultados dos experimentos com a grade de 401×401 pontos

Método	Nº de processadores	Nº de iterações	Tempo de processamento (s)	Fator de aceleração
Red-black serial	1	238609	1768	-
Zebra serial	1	238768	1507	1.17
Granulação fina	2	238925	1544	1.15
Granulação fina	4	238924	1235	1.43
Granulação fina	6	238922	1099	1.61
Granulação fina	8	238921	1060	1.67
Red-black blocos	2	238609	1698	1.04
Red-black blocos	4	238609	1389	1.27
Red-black blocos	6	238609	1303	1.36
Red-black blocos	8	238609	1236	1.43
Zebra blocos	2	238768	1482	1.19
Zebra blocos	4	238768	1243	1.42
Zebra blocos	6	238766	1165	1.52
Zebra blocos	8	238768	1121	1.58

Os resultados desse experimentos são consolidados na figura 1, abaixo

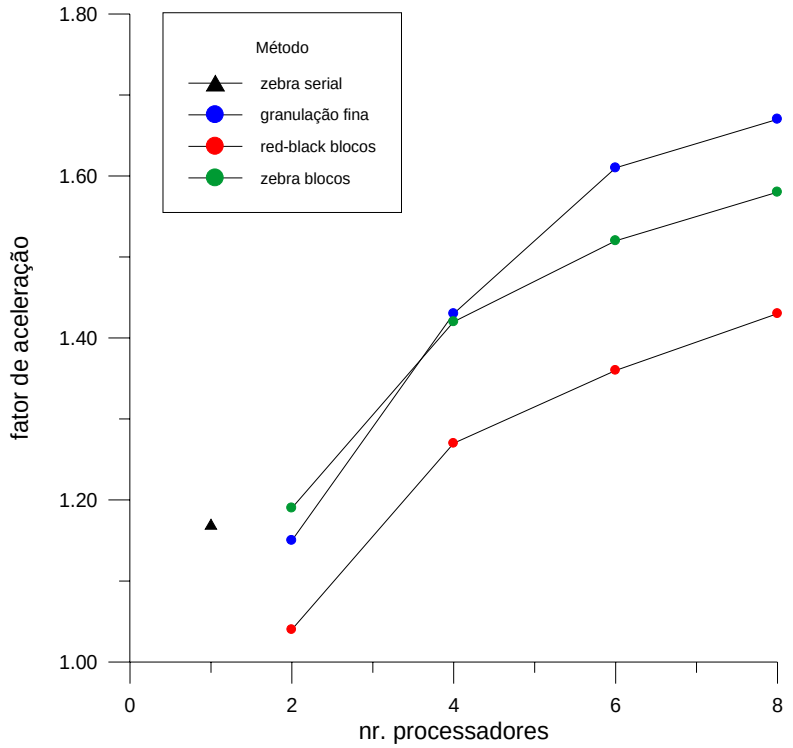

Figura 1- Fatores de aceleração dos experimentos com a grade de 401×401 pontos de grade.

5.1 – Experimentos com a grade de 201×201 pontos

A tabela II apresenta os resultados obtidos nos experimentos com a grade de 201×201 pontos de grade. O experimento com o método red-black serial foi o que apresentou maior tempo de processamento, juntamente com o método red-black blocos,

com dois processadores. Assim, o fator de aceleração também será calculado em relação ao resultado do experimento red-black serial.

Tabela II – Resultados dos experimentos com a grade de 201×201 pontos

Método	Nº de processadores	Nº de iterações	Tempo de processamento (s)	Fator de aceleração
Red-black serial	1	65269	112	-
Zebra serial	1	65348	102	1.1
Granulação fina	2	65426	102	1.1
Granulação fina	4	65425	82	1.37
Granulação fina	6	65423	74	1.51
Granulação fina	8	65422	71	1.58
Red-black blocos	2	65269	112	1
Red-black blocos	4	65269	97	1.15
Red-black blocos	6	65269	89	1.26
Red-black blocos	8	65270	85	1.32
Zebra blocos	2	65348	104	1.08
Zebra blocos	4	65348	87	1.29
Zebra blocos	6	65348	81	1.38
Zebra blocos	8	65346	81	1.38

Os resultados dos experimentos com a grade de 201×201 pontos são consolidados na figura 2, a seguir:

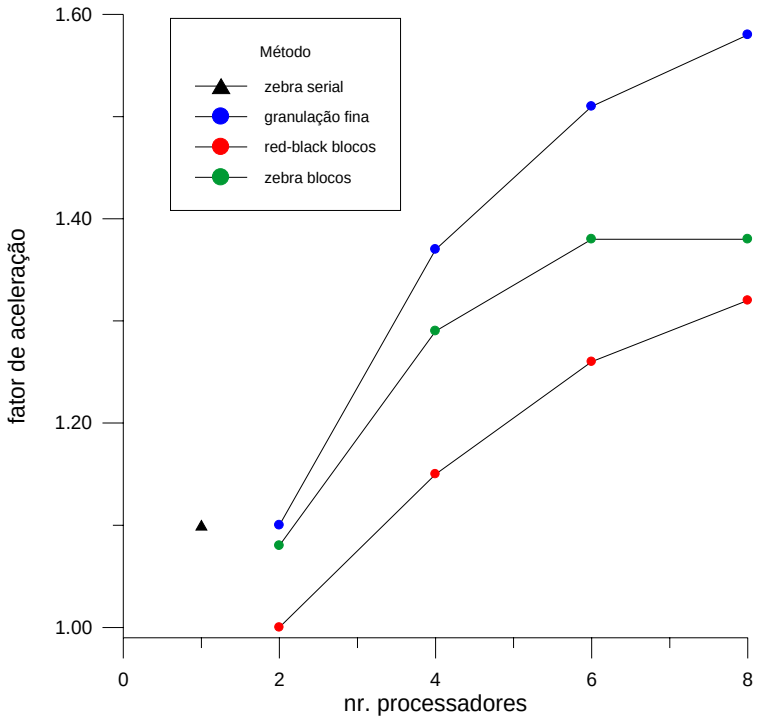

Figura 2 - Fatores de aceleração dos experimentos com a grade de 201×201 pontos de grade.

6 – Discussão e Conclusões

Idealmente, o fator de aceleração obtido em um programa paralelo em relação à sua versão serial deve ser igual ao número de processadores paralelos empregados na execução. No entanto, existe um custo computacional significativo associado ao gerenciamento dos diferentes comandos de criação e eliminação de "cópias" paralelas das seções do programa paralelizadas, além do tempo para leitura e escrita da memória, o que não permite a obtenção da aceleração ideal.

Os experimentos realizados mostram que para o problema analisado o ganho em eficiência computacional obtido pelos programas paralelos está muito abaixo do ideal. Alguns pontos podem ser destacados:

a) o método de granulação fina foi o que obteve maior eficiência computacional, seguido pelo método zebra, e por último, o red-black;
b) O melhor fator de aceleração obtido foi de apenas 1.67 para oito processadores, com o método de granulação fina, na grade 401×401 pontos de grade, que está muito abaixo do valor ideal;
c) Não existiu grande variação no número de iterações entre os diferentes métodos para atingir a tolerância estabelecida para a convergência, o que indica que o particionamento do problema não tem um impacto significativo na taxa de convergência; e
d) O uso de mais processadores teve um maior impacto positivo na malha com maior número de nós.

Os resultados indicam de forma geral que, embora o uso de processamento paralelo reduza o tempo computacional, o ganho em eficiência obtido com os algoritmos implementados neste estudo foi muito menor do que o esperado. Uma possível causa para a baixa eficiência obtida foi que o particionamento do domínio foi feito dentro do loop de iterações das aproximações, ou seja, antes do início de cada nova varredura do domínio eram criadas as cópias do programa (escravas) para cada processador adicional e após as varreduras dos pontos de grade por cada processador, essas cópias eram desfeitas, e o programa retornava ao comando do processador principal (master), para verificação do critério de convergência. O uso dessa abordagem tem a vantagem de tornar desnecessária a sincronização explícita dos processos realizados a cada iteração, porém elas aumentam significativamente o custo computacional.

Futuros estudos que serão realizados ao longo do projeto procurarão ampliar o conhecimento sobre o assunto, abordando, entre outros aspectos: o uso de comandos de sincronização explícita; a eficiência obtida em problemas diferentes, alterando as condições de contorno; técnicas visando a maior utilização da memória cache; e o emprego de outras técnicas numéricas para a solução de problemas de equilíbrio, particularmente o método multigrid.

Ricardo Carvalho de Almeida, D.Sc.
Professor Adjunto