6
Processamento em Paralelo

6.1
Modelos

O algoritmo seqüencial do Gradiente Conjugado baseia-se em operações elementares de álgebra linear. A paralelização do algoritmo consistirá essencialmente em realizar essas operações básicas de forma concorrente, bem como em aproveitar a divisão de tarefas intrínsica ao método de Schwarz para distribuí-las entre os processadores.

No algoritmo 3 podemos observar as rotinas a serem paralelizadas:

1- Operações básicas com vetores, como soma, subtração, multiplicação por escalar e produto interno.
2- Multiplicação da matriz C, por um vetor p.

2.1- Sabemos que $C = BA^{-1}B^T$, logo teremos que nos preocupar com a multiplicação por vetores de cada uma das matrizes B, A e B^T, bem como com a paralelização do método usado para invertermos a matriz A, que é o próprio GC.

Cada um dos operadores B, B^T e A podem ser distribuídos localmente por cada processador e as operações com vetores podem ser realizadas de forma autônoma desde que sejam atualizadas as bordas, que são os pontos onde haveria troca de informações entre processadores. Vejamos um exemplo para o operador A analisando a figura (6.1).

Vamos mostrar como os cálculos da multiplicação do operador A por um vetor podem ser feitos em paralelo a partir de operadores locais construídos individualmente por cada processador desa-tualizando apenas as bordas, isto é, as variáveis que pertencem aos dois processadores, o que vai exigir comunicação entre os processadores para atualização destas variáveis. As operações relativas ao eixo x estão descritas adiante.

Vejamos como ocorre o processamento serial para os elementos K_1 e K_2:

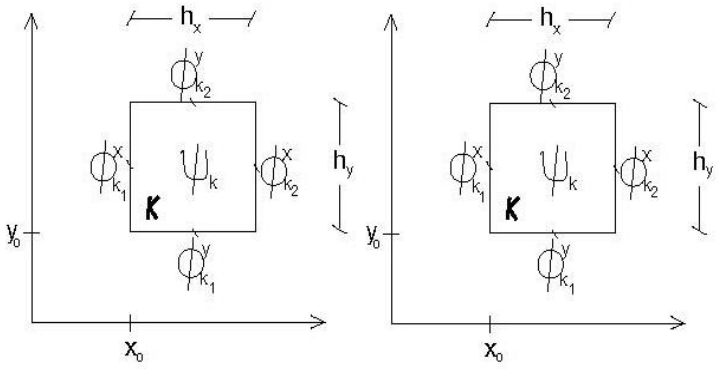

Figura 6.1: Fronteira entre processadores.

$$A[b,a] = \int_{k_1}(\lambda^{-1}\phi_a^x\phi_b^x)dxdy,$$
$$A[b,b] = \int_{k_1}(\lambda^{-1}\phi_b^x\phi_b^x)dxdy + \int_{k_2}(\lambda^{-1}\phi_b^x\phi_b^x)dxdy,$$
$$A[b,g] = \int_{k_2}(\lambda^{-1}\phi_b^x\phi_g^x)dxdy.$$

Assumamos agora que, para o programa em paralelo, os elementos K_1 e K_2 pertencem a processadores diferentes.

Para o processador que possui o elemento K_1 :

$$A_1[b,a] = \int_{k_1}(\lambda^{-1}\phi_a^x\phi_b^x)dxdy,$$
$$A_1[b,b] = \int_{k_1}(\lambda^{-1}\phi_b^x\phi_b^x)dxdy.$$

Para o processador que possui o elemento K_2 :

$$A_2[b,b] = \int_{k_2}(\lambda^{-1}\phi_b^x\phi_b^x)dxdy,$$
$$A_2[b,g] = \int_{k_2}(\lambda^{-1}\phi_b^x\phi_g^x)dxdy.$$

Como cada processador possui uma parte do vetor $u = [a,b,g]$, tal que $u_1 = [a,b]$ e $u_2 = [b,g]$ e b é duplicado para os dois processadores. Temos que $v = A * u$ pode ser calculado localmente, desde que depois os elementos das posições de interseção sejam somados e distribuídos, como abaixo:

$$(A * u)[b] = A[b,a] * u[a] + A[b,b] * u[b],$$
$$(A_1 * u_1)[b] = A_1[b,a] * u_1[a] + A_1[b,b] * u_1[b],$$
$$(A_2 * u_2)[b] = A_2[b,b] * u_2[b] + A_1[b,g] * u_2[b].$$

Reparemos que ambos os processadores possuem a posição b de seus vetores locais relacionados com a velocidade desatualizadas no momento imediatamente depois de ocorrer a multiplicação por A, pois $A[b,b] = A_1[b,b] + A_2[b,b]$, logo é necessário somar as contribuições locais e distribuí-las.

Para o operador B, como o resultado de sua multiplicação por uma vetor relativo à pressão é um vetor relativo à velocidade e, portanto, também ocorre desatualização, faremos um procedimento idêntico.

Para o operador B^T cujo resultado de sua multiplicação por um vetor relativo à velocidade é um vetor relativo à pressão, e sendo este último de natureza totalmente local, então não há necessidade de comunicação entre os processadores.

6.2 Rotinas da Biblioteca de Passagem de Mensagens entre os Processadores

Para desenvolver as rotinas em paralelo foi usado o modelo de Troca de Mensagens, através da biblioteca chamada MPI (Message Passage Interface).

As principais rotinas usadas para realizar estas tarefas estão explicadas abaixo:

MPI_Carte_create: Rotina para criar topologia.
Os processadores são numerados de zero em diante na topologia padronizada. Para facilitar o desenvolvimento das rotinas foi criada uma nova topologia em que cada processador passa a ser conhecido por sua linha e coluna. Esta rotina também escolhe os processadores que melhor convém para uma possível comunicação entre processadores "vizinhos".

MPI_Cart_coords: Determina a coordenada do processador em determinada topologia.
Esta rotina é usada para que cada processador conheça a sua topologia criada por
MPI_Carte_create.

MPI_Isend: Rotina de envio de mensagem ponto-a-ponto.
Esta rotina e usada para que possa haver a comunicação entre dois processadores somente. Foi usada no programa para a troca de informações que permitiram a atualização das variáveis relativas à velocidade após a multiplicação por A ou B.

MPI_Recv: Rotina de recebimento de mansagem ponto-a-ponto.
Esta rotina e usada para que possa haver a comunicação entre dois processadores somente. Foi usada no programa para a troca de informações

que permitiram a atualização das veriáveis relativas à velocidade após a multiplicação por A ou B.

MPI_Allreduce: Rotina que combina valores de vários processadores e envia o resultado de volta para os mesmos.
Esta rotina foi usada especialmente para o produto interno de variáveis, com os valores de cada processador combinados em soma e distribuídos a todos.