ALGORITMO PARALELO PARA SOLUÇÃO DAS EQUAÇÕES DE NAVIER-STOKES

Gustavo Prado Oliveira
Universidade Federal de Uberlândia – Faculdade de Engenharia Mecânica
gpoliveira@mecanica.ufu.br

Aristeu da Silveira Neto
aristeus@mecanica.ufu.br

Ellie Luiz Padilla
ellie@mecanica.ufu.br

Resumo: *A disciplina Dinâmica dos Fluidos Computacional (CFD) tem contribuído de forma significativa para a engenharia, sobretudo em problemas que envolve fluidos em movimento. As aplicações são numerosas e vão desde o projeto externo completo de um avião até o projeto de motores mais eficientes. Contudo problemas de dinâmica dos fluidos apresentam dificuldade em sua resolução, mesmo em casos simples. Para solucionar o problema com o alto custo computacional exigido, uma alternativa é paralelizar a implementação dos códigos computacionais, e como aqueles que servem à solução numérica das equação de Navier-Stokes.*

Palavras-chave: *Navier-Stokes, Paralelização, Dinâmica dos Fluidos Computacional.*

1. INTRODUÇÃO

Os métodos de solução das equações de Navier-Stokes evoluíram consideravelmente, transformando os softwares desta área em instrumentos de grande valor para a análise de problemas de mecânica dos fluidos e transferência de calor. Dessa forma, a fluidodinâmica computacional está se tornando uma área de grande interesse para a solução de muitos problemas práticos. Para se estudar a dinâmica dos fluidos utilizando métodos computacionais, há a necessidade de modelos matemáticos diferenciais os quais são estabelecidos com base nos princípios de conservação da massa, conservação da energia e a segunda lei de Newton. Essas equações, quando submetidas a condições de contorno e iniciais apropriadas, e quando discretizadas dão origem a sistemas lineares a sere resolvidos. Esses sistemas se tornam maiores à medida que mais pontos são empregados na discretização do domínio contínuo. A solução numérica destas equações (em especial as equações de Navier-Stokes para escoamentos incompressíveis) tornam-se uma tarefa computacional pesada, e, devido a quantidade de parâmetros de interesse destas aplicações, os requisitos de solução no tempo e espaço aumentaram muito. Conseqüentemente, um número enorme de graus de liberdade devem ser calculados, aumentando assim a necessidade por memória e processamento além da capacidade dos supercomputadores atuais.

2. MODELO MATEMÁTICO.

O modelo matemático baseado nas equações da continuidade e de quantidade de movimento linear para escoamentos incompressíveis (equações de Navier-Stokes) assume a seguinte forma:

$$\nabla \vec{u} = \frac{\partial u_j}{\partial x_j} = 0\,, \qquad (1)$$

$$\frac{\partial u_i}{\partial t}+\frac{\partial}{\partial x_j}(u_i u_j)=-\frac{1}{\rho_0}\frac{\partial P}{\partial x_i}+\frac{\partial}{\partial x_j}\left[\upsilon\left(\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_i}\right)\right], \tag{2}$$

onde ρ_0 é a massa específica do fluido, P a pressão, υ a viscosidade fluido e $\vec{u}$ é o vetor velocidade do fluido nas direções *x* e *y* .

Após definir o modelo matemático, deve-se definir as condições iniciais e condições de contorno para o problema, determinando a fase inicial do escoamento. Deve-se impor inicialmente um campo de velocidade tal que $\vec{\nabla}.\vec{V}=0$, e para o campo de pressão, como não existe valor de referência a ser utilizado num primeiro instante, adota-se P(x,y)=0 .

As condições de contorno, por sua vez, definem o domínio de cálculo. No caso da cavidade com tampa deslizante, considera-se que as paredes laterais e a parede inferior são sólidas, portanto, não há escorregamento das partículas, ou seja, a velocidade é nula (u = 0 e v = 0). Espera-se, com isso, que apareça uma grande recirculação no interior da cavidade, a qual deve começar próximo da tampa e com o passar do tempo deve migrar em direção do centro da cavidade, até atingir o regime permanente. Na parede superior, por sua vez, considera-se uma velocidade igual u_0. Já para o cálculo do campo de pressão, considera-se que o gradiente de pressão na parede é nulo. As condições de contorno são apresentadas na Figura 1, onde u é a velocidade na direção *x* e *v* na direção *y* .

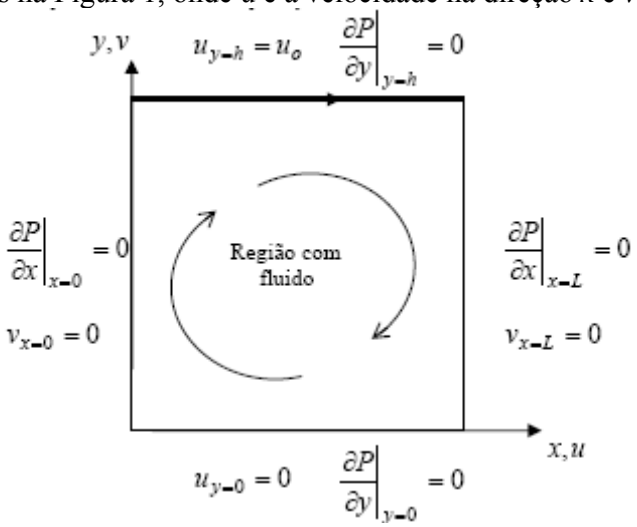


Figura 1: Condições de contorno para o problema da cavidade com tampa deslizante

Utilizando a metodologia LES (Simulação das Grandes Escalas), proposta por Smagorinsky (1963), a decomposição das escalas é feita através de um processo de filtragem das equações de Navier-Stokes e de separação das escalas. Para tanto as variáveis presentes no escoamento são separadas em grandes escalas e pequenas escalas também denominadas sub-malha. Aplicando-se o processo de filtragem nas Equações 1 e 2, obtendo as seguintes equações:

$$\frac{\partial \overline{u}_j}{\partial x_j}=0 \tag{3}$$

$$\frac{\partial \overline{u}_i}{\partial t}+\frac{\partial}{\partial x_j}(\overline{u_i u_j})=-\frac{1}{\rho}\frac{\partial \overline{P}}{\partial x_i}+\frac{\partial}{\partial x_j}\left[\upsilon_{ef}\left(\frac{\partial \overline{u}_i}{\partial x_j}+\frac{\partial \overline{u}_j}{\partial x_i}\right)\right] \tag{4}$$

Na Equação (4), nota-se que o termo não linear se apresenta na forma de um produto filtrado. Assim, para resolver este problema, faz-se a decomposição das escalas em uma parte filtrada e em uma flutuante, o que resulta na seguinte equação:

$$\frac{\partial \overline{u}_i}{\partial t}+\frac{\partial}{\partial x_j}(\overline{u}_i\overline{u}_j-\tau_{ij})=-\frac{1}{\rho}\frac{\partial \overline{P}}{\partial x_i}+\frac{\partial}{\partial x_j}\left[\upsilon\left(\frac{\partial \overline{u}_i}{\partial x_j}+\frac{\partial \overline{u}_j}{\partial x_i}\right)\right], \tag{5}$$

onde τij é o tensor de Reynolds sub-malha.

A Equação (5), no entanto, se apresenta na forma de um sistema não fechado devido ao termo não linear da equação da quantidade de movimento linear. Boussinesq (1877) propôs uma maneira de se resolver este problema expressando o tensor de Reynolds sub-malha em função da taxa de deformação gerada pelo campo de velocidade filtrado e da energia cinética turbulenta, assim como apresentado na Equação (6),

$$\tau_{ij}=-\upsilon_t\left(\frac{\partial \overline{u}_i}{\partial x_j}+\frac{\partial \overline{u}_j}{\partial x_i}\right)+\frac{2}{3}k\delta_{ij}, \tag{6}$$

onde υ_t é viscosidade turbulenta e k a energia cinética turbulenta .

A proposta de Boussinesq (1877) consiste em incorporar a energia cinética turbulenta à pressão estática e adotar uma metodologia para calcular a viscosidade turbulenta assim como apresentado na Equação (7).

$$\frac{\partial \overline{u}_i}{\partial t}+\frac{\partial}{\partial x_j}(\overline{u}_i\overline{u}_j)=-\frac{1}{\rho}\frac{\partial \overline{P}}{\partial x_i}+\frac{\partial}{\partial x_j}\left[\upsilon_{ef}\left(\frac{\partial \overline{u}_i}{\partial x_j}+\frac{\partial \overline{u}_j}{\partial x_i}\right)\right] \tag{7}$$

onde $\upsilon_{ef}=\upsilon+\upsilon_t$ é a viscosidade efetiva do fluido.

Nesse sentido, Smagorinsky (1963), baseando-se na hipótese do equilíbrio local para as pequenas escalas, propôs uma metodologia para captura e simular parte das estruturas turbilhonares na qual,

$$\upsilon_t=(C_s.l)^2.\left\|\overline{\overline{S}}\right\| \tag{8}$$

onde C_s é a constante de Smagorinsky, cujo valor é 0,18. $l=\sqrt{\Delta x\Delta y}$ é o comprimento característico sub-malha e $\left\|\overline{\overline{S}}\right\|=\sqrt{2\overline{S_{ij}}\,\overline{S_{ij}}}$ é o tensor de deformação do campo filtrado, no qual $\overline{S_{ij}}=\frac{1}{2}\left(\frac{\partial \overline{u}_i}{\partial x_j}+\frac{\partial \overline{u}_j}{\partial x_i}\right)$.

A metodologia de Smagorinsky apresentada anteriormente propõe, basicamente, que a produção de turbulência é função da taxa de cisalhamento do campo filtrado e a dissipação é função da escala de velocidade e do comprimento característicos sub-malha.

Enfim, a partir da solução numérica das equações anteriores, pode-se analisar e compreender o escoamento na cavidade bidimensional considerando o fluido incompressível.

2.1 Algoritmo Computacional Serial

O algoritmo computacional do programa serial pode ser resumido a partir dos seguintes itens:

Item 1: No instante $t=t_0$, utiliza-se um campo de velocidade e de pressão tal que $(i=0,...,n_x \,\&\&\, j=0,...,n_y+1): u_{i,j}=v_{i,j}=P_{i,j}=0$

Item 2: A partir dos valores de u, v e P, define-se, no tempo t, as condições de contorno nas paredes da cavidade, calcula-se os valores de $u^*_{i,j}, v^*_{i,j}$ definidos no tempo t + Δt , usando as condições de contorno e as equações de Navier-Stokes discretizadas**.**

Item 3: Para cada célula, a partir das condições de contorno nas paredes da cavidade, calcule a correção para a pressão $P_{i,j}$ .Usa-se o método iterativo S.O.R. para resolver o sistema linear.

Item 3.1**:** Etapas do método S.O.R.:

3.1.1. Para $(i = 1,...,n_x \,\&\&\, j = 1,...,n_y): P^*_{i,j} = P^*_{i,j} - w\dfrac{F(P^*_{i,j})}{F'(P^*_{i,j})}$

3.1.2. Após calcular $P^*_{i,j}$ normaliza-se o campo de pressão:

Para $(i = 1,...,n_x \,\&\&\, j = 1,...,n_y): P^*_{i,j} = P^*_{i,j} - norma$

Este procedimento é adotado, uma vez que não há um valor de referência para a pressão. Assim, para se evitar que os valores da pressão cresçam ou diminuam de forma arbitrária, a pressão deve ser normalizada após cada iteração do método S.O.R.

3.1.3. Verificar a Convergência do SOR:

Para

$$(i = 1,...,n_x \,\&\&\, j = 1,...,n_y): \left| P^*_{i,j} - \left(\frac{A_{E_{i,j}} P^*_{i+1,j} + A_{W_{i,j}} P^*_{i-1,j} + A_{N_{i,j}} P^*_{i,j+1} + A_{S_{i,j}} P^*_{i,j-1} - B_{i,j}}{A_{P_{i,j}}} \right) \right| < \varepsilon_2$$

Caso todos os nós $P^*_{i,j}$ passem pelo processo de convergência, avance para o item 4, caso contrário volte ao item 3.1.1.

Item 4: Com os valores de u^*, v^*, P^* atualize u, v e P a partir das equações apresentadas no passo 3. Os novos valores de u, v e P se referem ao instante de tempo t + Δt .

Item 5: Verifique se a massa se conserva, assim como apresentado no passo 4. Caso haja conservação da massa avance para o próximo passo de tempo, caso contrário interrompa a simulação → divergência numérica.

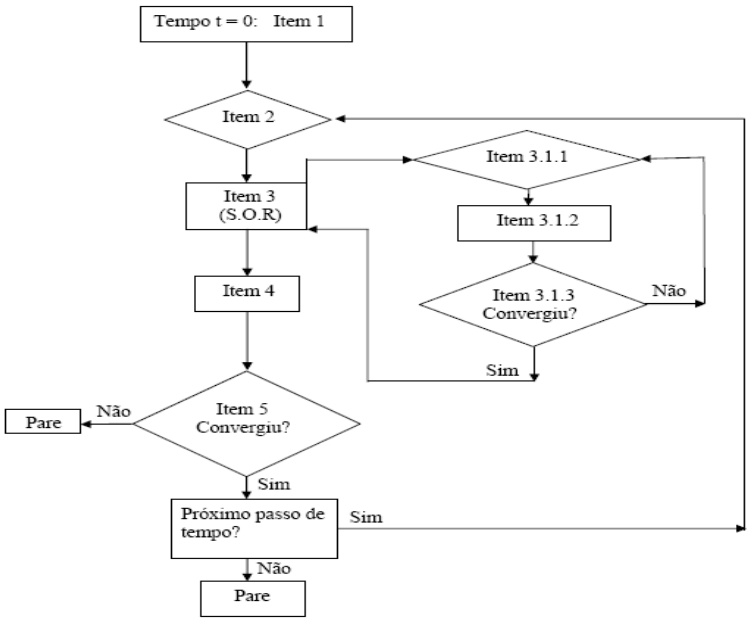


Figura 2: Forma resumida de acordo com os itens apresentados anteriormente

2.2 Computação Paralela

O processamento paralelo permite a solução de problemas que exigem muito processamento e memória, através da configuração de um cluster de computadores, uma vez que neste tipo de arquitetura utiliza-se a divisão de processamento do problema entre várias máquinas. Assim, contribui para o desenvolvimento da ciência, seja por reduzir limitações de memória, seja por aumentar a velocidade de processamento, ou na maioria dos casos ambas alternativas.

No presente trabalho, a arquitetura distribuída foi escolhida, pois para expandir a capacidade de processamento a um custo baixo e conseguir simular altos números de Reynolds, é necessário um alto poder computacional. As arquiteturas de memória compartilhada apresentam a grande vantagem de serem mais rápidas, pois não há necessidade da troca de dados e o acesso ao barramento é mais veloz.

A decomposição do problema, independente da forma como é feita, deve priorizar um balanço adequado de carga e a minimização de comunicação entre processadores no caso de memória distribuída, o que não será priorizado na elaboração do algoritmo, uma vez que já existem ferramentas próprias para executar tal operação. O balanço de carga adequado consiste em dividir igualmente o trabalho entre os processadores disponíveis. Já a minimização da comunicação entre os nós, como o nome já indica, visa diminuir ao máximo a comunicação ou troca de mensagens, pois representam tempo a menos de processamento que seria gasto na solução do problema.

2.3 Particionamento de Malhas Computacionais

A malha computacional representa um espaço discretizado descrito por células tridimensionais formadas por pontos em coordenadas cartesianas. A paralelização do código BrU3D utiliza o conceito de decomposição de domínios, ou seja, cada nó do cluster resolve as equações de Navier-Stokes para somente algumas partições de células. Por isso, é necessário previamente dividir/particionar o domínio/malha computacional para que todos os processadores atuem em seus respectivos domínios. Para isso, utilizou-se uma biblioteca robusta de particionamento de malhas e grafos chamada METIS. O resultado do particionamento de uma malha não-estruturada pode ser visualizado na Figura 3 abaixo.

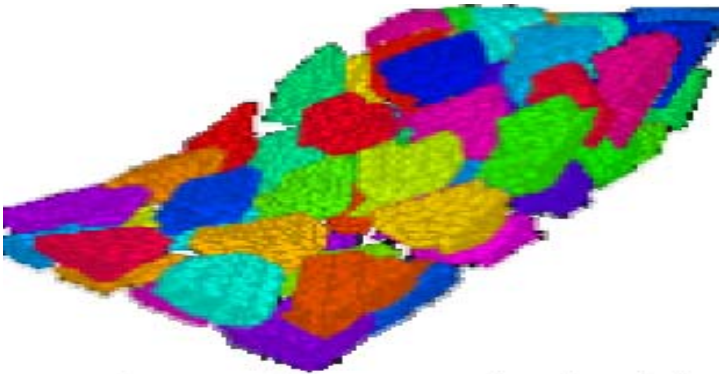


Figura 3: Visualização de uma malha computacional particionada com METIS

3. ESTRATÉGIAS DE PARALELIZAÇÃO

A analise das diferentes operações descritas na seção 2 lidam com as seguintes observações: (i) existe uma dependência no tempo no procedimento de passo de tempo; (ii) existe uma dependência encontrada nas derivadas espaciais. Com base nestas possibilidades as estratégias de paralelização são divididas.

3.1. Esquema A

No caso da linguagem C, a ordem de um array é armazenado na memória na ordem mostrada na figura 4, abaixo.

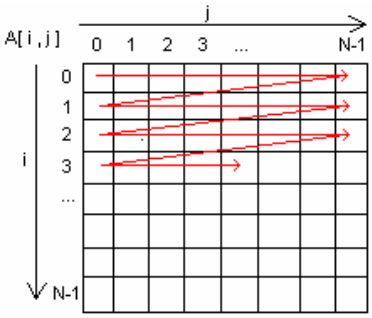


Figura 4: Como um array bi-dimensional é armazenado na memória

É mais eficiente acessar arrays sucessivamente na ordem em que são armazenados na memória, do que acessá-los aleatoriamente. As considerações feitas para paralelizar o código são de lidar com o esquema da malha deslocada (Figura 5) e o método de diferenças finitas centradas.

Nota-se que os cálculos com respeito ao eixo y demonstram dependência a qual é limitada por pontos adjacentes do tipo (*i,j-1*), (*i, j*) e (*i, j+1*). Sendo assim , a primeira opção é elaborar um esquema de paralelização de forma a manter partições do domínio computacional ao longo do eixo y durante todo o processo de cálculo computacional, ou ao longo do eixo x, transmitindo entre cada processador, somente fronteiras planas. Cada processador *p* pode atualizar os cálculos em todos os pontos da malha plana, exceto nos pontos no primeiro e no último plano y. Somente o ultimo plano-y do processador (*p-1*) e o primeiro plano-y do processador (*p-1*) tem que ser transmitidos para *p*. A figura 6 mostra o processo computacional paralelo descrito neste esquema.

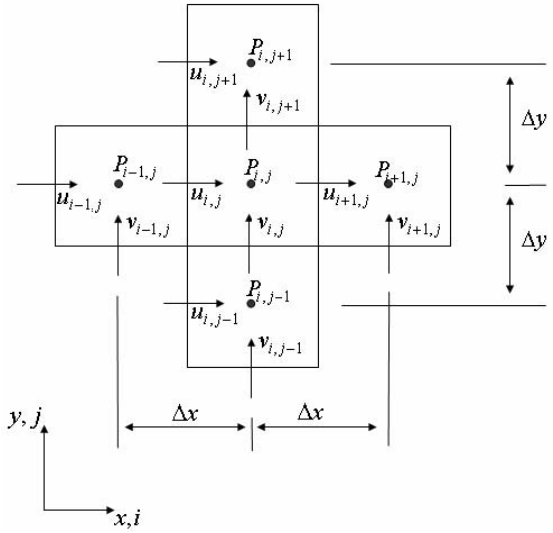
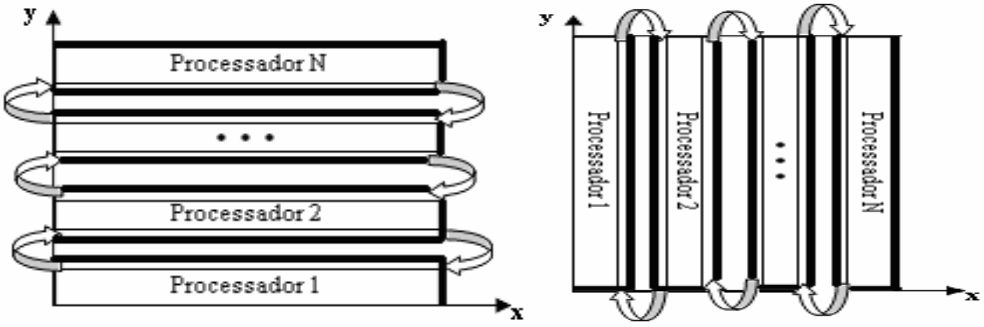


Figura 5: Método da malha deslocada



(a) (b)

Figura 6 : Processo computacional. Linhas indicam comunicação entre as fronteiras.

Apesar de os dois casos de paralelização parecerem semelhantes, o grande número de perda de memória cache devido ao grande passo de tempo, faz do loop (b) mais devagar do que o loop (a).

Além das considerações anteriores, em alguns casos, a distancia entre as iterações também importa na decisão de sua estratégia. Suponha que y > x, então o número de elementos que deve ser transmitido, será proporcional ao tamanho da fronteira entre os processadores.

3.2. Esquema B

Todos os loops da sessão anterior podem ser paralelizados tanto fixando-se a linha quanto a coluna. Contudo, pode-se paralelizar ambos elaborando um esquema de paralelização de forma a manter partições do domínio computacional ao longo de ambos os eixos x e y ao mesmo tempo, durante todo o processo de cálculo computacional, como mostrado na figura 7.

A figura mostra um exemplo de distribuição de uma matriz para quatro processadores, divido em duas dimensões, transmitindo entre cada processador, duas fronteiras. Neste caso, sempre que se dividir a malha em áreas cada vez menos retangulares no sentido do eixo x, menor será a quantidade de dados a serem transmitidos, de acordo com a comunicação de dados em C++.

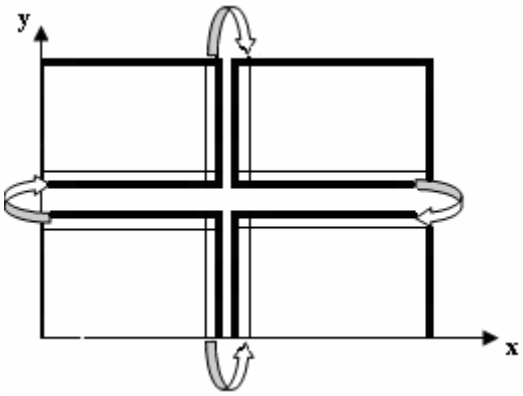


Figura 7: Processo computacional. Comunicação entre as fronteiras em linha e coluna.

A comunicação entre os domínios, juntamente com a quantidade de cálculos processados em cada um deles, requer um cuidado especial, pois afeta não somente o desempenho da simulação mas, também, a integridade dos resultados. Infelizmente, ainda não existe um software que seja capaz de paralelizar automaticamente, de forma eficiente e com segurança, todas as situações possíveis encontradas em processamento paralelo. Programas de maior desempenho ainda necessitam da intervenção do programador para se chegar ao resultado desejado.

4. ALGORITMO PARALELO PARA CAVIDADE COM TAMPA DESLIZANTE

Nesta seção, será mostrado como usar o modelo mestre escravo, executar a decomposição do domínio e implementar a troca de mensagens. Cada uma das tarefas citadas, é construída sobre a tarefa previa.

4.1. O Modelo Mestre-Escravo

Neste estágio deve-se criar, ou possuir, uma arquitetura mestre escravo, assim como a existente no LTCM (Laboratório de Transferência de Calor e Massa - FEMEC), onde o nó mestre irá escrever para os arquivos de saída. Após terminada a geração dos arquivos de saída, a malha cartesiana deve ser gerada de acordo com a topologia do modelo.

1º Passo: Encontrar o número de processadores existentes, configurando a quantidade na direção x e a quantidade na direção y. Criar a malha cartesiana bi-dimensional, de forma a mapear a localização dos dados para cada processador e, identificar os processadores visinhos mais próximos.

2º Passo: Inicializar os arrays com os tamanhos de x e y, lembrando que para o algoritmo funcionar corretamente, deve-se ter em cada direção do plano cartesiano arrays com tamanhos divisíveis pelo numero de processadores naquela direção. Para programação em C++ deve-se alocar um array global somente no processador 0. Os processos executáveis precisam alocar somente a memória que eles necessitam, para isto funcionar corretamente deverá ser mapeado o array 1D (somente em uma direção) para o 2D – isto garantira que a memória alocada seja contígua.

3º Passo: Encontrar as coordenadas cartesianas que delimitam cada processador, criando um tipo de dados para transferir dados locais para o processador 0 (nó mestre).

4º Passo: Se for o processador 0, fazer um loop do processador 1 até o número de processadores menos 1, para receber os dados do array; e escreva no arquivo de dados. Caso contrário, os processadores enviam linha de dados para o processador 0.

4.2. Decomposição do Domínio.

Na maioria dos problemas de decomposição de domínio é frequentemente necessário trocar dados da fronteira entre os domínios do processador. Esta operação é feita para minimizar a comunicação imediata entre processadores, onde os dados importados de um processador são frequentemente referenciados a uma região especifica.

Teoricamente, deve-se verificar se os dados locais possuem linhas e colunas extras para cada fronteira dos processadores, para que somente a região interna de cada subdomínio seja atualizadas e a região especificada (linhas e colunas extras) sirva para receber dados de outro processador e também pode ser utilizada para escrever dados recebidos para o arquivo.

Após esta verificação concluída, o próximo estagio da decomposição de domínio é trocar as fronteiras entre os processadores, operação esta que pode ser especificada a partir do pseudo código abaixo:

- Criar linha derivada dos dados locais
- Criar coluna derivada dos dados locais
- Encontrar o processador visinho mais próximo na direção x
- Encontrar o processador visinho mais próximo na direção y
- Trocar fronteiras com os processadores acima e abaixo
- Trocar fronteiras com os processadores à direita e à esquerda

A figura 8 ilustra a transferência de linhas através do domínio do processador.

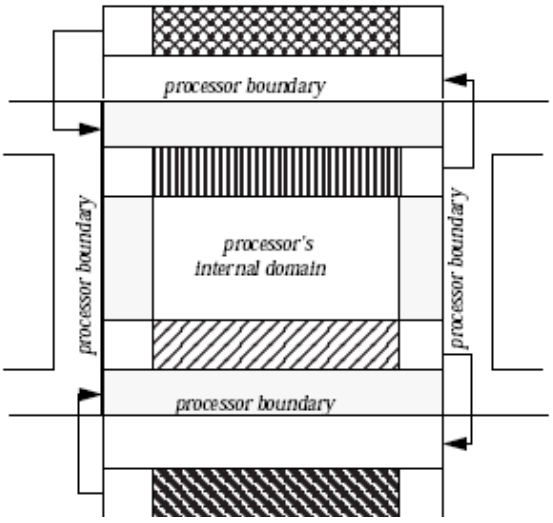


Figura 8: Troca de linhas e colunas entre domínios

4.3. Detalhes para implementação.

A primeira preocupação a ter no desenvolvimento da aplicação é garantir a região de transferência de dados do domínio do processador, para isso deve-se alocar para cada domínio local (dx +2) X (dy+2).

Para mapear os dados locais entre os dados globais, necessita-se de encontrar o canto mais superior a direita e o canto mais inferior a direita do domínio local como mostrado no pseudo código abaixo da figura 9, devido a forma de armazenamento dos dados na memória, como mostrado na figura 4.

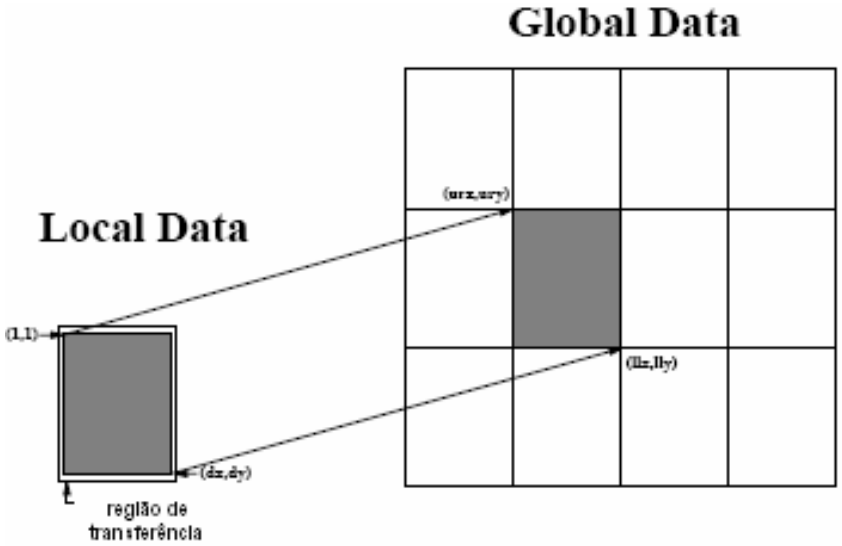


Figura 9: Mapeamento das regiões locais para a distribuição de dados do domínio global

Pseudo Código:

```
urx =  1 + coord[0]  * (dx-1)
ury =  1 + coord[1] * (dy-1)

llx =  urx + dx
lly =  ury + dy
```

Este mapeamento pode garantir que as condições iniciais sejam configuradas localmente em cada processador, e após determinada estas condições, pode-se utilizar o algoritmo da figura 2, adotando a limitação do domínio local, e acrescenta-lo aos passos do item 4.1 para cada célula.

4. CONCLUSÕES E TRABALHOS FUTUROS

Neste artigo, foi utilizado um algoritmo para a solução da equação de Navier-Stokes 2-D com a proposta de um esquema de paralelização também em 2-D escolhido mediante a eficiência do armazenamento em memória para a linguagem C++. Mostrou-se que o algoritmo é escalável para cálculos computacionais entre processadores visinhos.

Para o tipo de paralelismo adotado, a principal consideração a ser tomada é referente ao particionamento do domínio global, adotando uma estratégia de "dividir para conquistar", aumentando assim a eficiência do mesmo. Já para implementações futuras, além da linguagem C++, sugere-se o uso de MPI como sistema de troca de mensagens entre os processadores, para obter portabilidade e oferecer uma biblioteca de alto nível para o problema proposto.

5. REFERÊNCIAS BIBLIOGRÁFICAS

Averbuch, A., Ioffe, L., Israeli, M., Vozoroi, L., "Higly Scalable Two and Three-Dimensional Navier-Stokes Parallel Solvers on MIMD Multiprocessor", Journal of Supercomputing, Vol.11, No. 1, pp. 7-39, 1997.

Azevedo, J.L.F., "Paralelização do Código de Mecânica dos Fluidos BRU3D".

Fang, N. 1996, "Engineering Parallel Algoritms", IEEE Proceedingds of HPDC-5.

Fletcher, C.A.J., 1988, "Computational Techniques for Fluid Dynamics", Vol I, Spring-Verlag, New York.

Kortas, S., Angot, P., "A pratical and portable model of programming for iterative solvers on distributed memory machines", Parallel Computing 22, 487-512, Elsevier, 1996.

Menezes, G.D., "Simulação Numérica do Escoamento em uma Cavidade Bidimensional com Tampa Deslizante", XIV CREEM, 2007.

Karniadkis, G.E., Kirby, R.M., "Parallel Scientific Computing in C++ and MPI", Cambridge University Press, 2003.

PARALLEL ALGORITHM FOR SOLVING THE NAVIER-STOKES EQUATIONS IN A FLOW LID DRIVEN CAVITY

Gustavo Prado Oliveira
Universidade Federal de Uberlândia – Faculdade de Engenharia Mecânica
gpoliveira@mecanica.ufu.br

Aristeu da Silveira Neto
aristeus@mecanica.ufu.br

Abstract: *The subject of computational fluid dynamics(CFD) it already caused a great impact in the science and engineering, with wide applications that deal with complete external project of an airplane until project of more efficient motors. However, fluids mechanical problems have presented difficulty in resolution, even in simple cases. To solve the problem with the high cost computational demanded by CFD, a solution is to implement computational parallel codes, and using the Navier-Stokes equations like the government solution of these problems. Tthe objective of this work is to show how to implement an algorithm that can be solved in a parallel way, turning more agile the solution of this problem.*

Keywords: *Navier-Stokes, Algorithm, Parallelization, computational fluid dynamics(CFD)*