Sobre o Desenvolvimento e Avaliação dos Métodos Seqüencial e Paralelo das Diferenças Finitas para EDP's Elípticas em Plataforma Windows NT/2000/XP®

Daniel G. E. Trovó, J. M. Balthazar (orientador)
Departamento de Estatística, Matemática Aplicada e Computação, IGCE, UNESP,
13500-000, Rio Claro, SP
E-mail: d_trovo@yahoo.com.br

Resumo: *Este projeto de pesquisa visa à implementação e análise de desempenho dos algoritmos Seqüencial e Paralelo do Método de Diferenças Finitas para Equações Diferenciais Parciais Elípticas – particularmente a Equação de Poisson, sobre arquiteturas paralelas de memória distribuída. O modelo de processamento paralelo escolhido é o SPMD (Single Process/Program, Multiple Data), onde um único código é executado em diferentes elementos de processamento atuando sobre diferentes massas de dados.*

Palavras-chave: Equações Diferenciais Parciais Elípticas, Computação Paralela, Gauss-Seidel, Computadores Paralelos com Memória Distribuída.

Contribuições do Trabalho: A partir do desenvolvimento deste projeto de pesquisa pôde-se iniciar a construção de um pacote de métodos paralelos para a solução de problemas de valor de contorno utilizando-se ambiente Microsoft Windows XP com o ambiente de troca de mensagens MPICH do Argonne National Laboratory - Universidade de Chicago, de domínio público, para facilidade do usuário que já domina este ambiente, diferenciando-se das aplicações normalmente implementadas em plataformas UNIX. Concentrou-se em EDP's Elípticas, pois o método de Diferenças Finitas para sua resolução apresenta maiores dificuldades em se paralelizar do que as EDP's Parabólicas e Hiperbólicas. Desenvolveram-se as rotinas paralelas implementadas em linguagem C sobre camadas escaláveis que podem ser aplicadas em quaisquer problemas de contorno desta natureza. As dificuldades que surgirão dependerão das particularidades dos problemas abordados. Ressalta-se que a metodologia geral desenvolvida é adequada à solução destes problemas.

Tópicos do Trabalho: Desenvolveu-se este trabalho através dos seguintes tópicos: No item (1) efetuou-se uma introdução abordando a história das arquiteturas seqüenciais e paralelas e seus conceitos fundamentais; no item (2) apresentou-se o método de Diferenças Finitas Seqüencial e Paralelo, destacando-se o algoritmo de *particionamento* com máxima distribuição de carga, e a implementação em camadas; em (3) apresentou-se as considerações finais com a análise de desempenho; em (4) fez-se alguns agradecimentos; em (5) apresentou-se as referências bibliográficas de suporte ressaltando-se o portal do Grupo de Pesquisa de Aplicações em Computação Paralela citado em [7] onde está disponível uma extensa pesquisa bibliográfica e, finalmente, apresentou-se no Apêndice (6) uma pequena

introdução ao padrão MPI e à implementação do MPICH como complementação ao trabalho.

1. Introdução

Com a criação do primeiro computador eletrônico em 1946 e o estabelecimento dos conceitos básicos de organização dos computadores digitais surgiu, na década de 50, o modelo computacional que se tornaria a base de todo o desenvolvimento subseqüente - o *modelo de von Neumann*.

Desde então a computação passou por um processo evolutivo intenso, tanto em hardware como em software, com a finalidade de proporcionar maior desempenho e ampliar suas fronteiras.

Com o surgimento da filosofia VLSI de projeto de microprocessadores a partir de 1980 e o desenvolvimento de redes de alto desempenho na década passada foi possível a interligação dos computadores pessoais e workstations formando os Sistemas Distribuídos.

Paralelamente ao ciclo de desenvolvimento do modelo de *von Neumann* foram propostas, desde a década de 70, arquiteturas alternativas que provessem maior eficiência e facilidades no processo computacional. A criação destas arquiteturas alternativas contendo vários processadores atuando conjuntamente originou o estudo da Computação Paralela.

Embora tenham surgido por motivações diferentes, a Computação Paralela e a Distribuída demonstraram uma estreita relação por possuírem características e problemas semelhantes, como balanceamento de carga, *modularidade* e tolerância a falhas [5]. Por isso, há alguns anos, os sistemas em rede que suportavam os Sistemas Distribuídos começaram a ser utilizados para execução de programas paralelos, em virtude de seu menor custo e maior flexibilidade.

Como não há compartilhamento de memória global entre todos os processadores deste novo paradigma, a execução de programas paralelos necessita de comunicação e sincronização entre os processos através de trocas de mensagem.

Foram então desenvolvidos ambientes para troca de mensagem para facilitar o trabalho do programador. Tais ambientes constituem-se basicamente por uma biblioteca de comunicação atuando como uma extensão de linguagens seqüenciais (C e FORTRAN), e seus maiores representantes são o PVM (Parallel Virtual Machine) e o MPI (Message Passing Interface).

1.1 Computação Paralela

Computação Paralela é execução concorrente de programas em diferentes elementos de processamento visando à solução de um único problema. É utilizada para solucionar problemas muito ineficientes em sua forma seqüencial ou quando esta solução somente é válida se obtida em um limite máximo de tempo.

Um programa paralelo geralmente é projetado para ser executado em um tipo específico de arquitetura de computadores, podendo esta ser encontrada em uma das seguintes categorias:

- **Arquiteturas de Memória Compartilhada entre Processadores:** apresentam a importante vantagem, do ponto de vista do programador, de disponibilizar o mesmo conjunto de dados a todos os elementos de processamento. Em contrapartida, o desenvolvimento do hardware e sistema operacional é complexo e torna o custo desta implementação muito alto.

- **Arquiteturas de Memória Distribuída entre Processadores:** ganharam destaque atualmente por serem mais flexíveis, escaláveis e principalmente mais baratas do que as de memória compartilhada. Tornam o trabalho do programador mais difícil, já que é necessário troca de mensagem entre os elementos de processamento, mas em compensação não necessitam hardware específico para computação paralela. São altamente escaláveis e flexíveis, pois permitem a eventual troca da topologia de rede escolhida.

Para Arquiteturas de Memória Distribuída é imprescindível que haja certa independência entre os dados do problema, já que o tempo de comunicação é muito alto se comparado ao tempo de processamento. Problemas em que há muita computação e pouca comunicação apresentam bom ganho de desempenho quando paralelizados.

É necessário explicar que a classificação anterior é a mais simples e comum em se tratando de arquiteturas paralelas. Há, porém, subcategorias e categorias intermediárias como *Computadores paralelos de memória distribuída com endereçamento global,* explicitados em [3], além de outras classificações.

Para que se possa concluir que a aplicação paralela obteve ganho de desempenho são definidas algumas medidas de desempenho como as apresentadas a seguir.

Ganho ou Speedup

Ganho ou speedup é definido como a relação entre o tempo de execução de um algoritmo seqüencial e sua execução paralela [6].

$$Sp = \frac{Tseq}{Tp}$$

Onde

Tseq é o tempo para execução do algoritmo seqüencial e,

Tp o tempo para execução do algoritmo paralelo.

Para que o resultado possa ser utilizado como demonstração de que o problema pode ser resolvido mais eficientemente em uma máquina paralela, considera-se uma implementação otimizada das duas formas, ou seja, implementa-se um algoritmo seqüencial otimizado e outro otimizado para o paralelismo.

Eficiência

A eficiência se dá com a relação entre o ganho (speedup) e o número de processos do algoritmo paralelo [6].

$$E = \frac{Sp}{P}$$

Onde

Sp é o ganho e

P o número de processadores.

A eficiência dá uma idéia do tempo total gasto na aplicação por cada um dos processos.

2. O MÉTODO DAS DIFERENÇAS FINITAS

2.1 Introdução

Problemas de base matemática, envolvendo a solução de Equações Diferenciais Parciais (EDPs), são o núcleo da maioria dos problemas em exatas, tecnologia e indústria, ou seja, da classe de problemas associados a meios contínuos, à evolução no tempo de corpos imersos em meios contínuos, ou associados à evolução temporal do próprio meio contínuo.

Estes fenômenos resultam, quando modelados matematicamente, em equações ou sistemas de equações diferenciais parciais, exigindo a solução destas em meios reais com infinitas variáveis, de coordenadas infinitas. Tal modelo requisita discretização do contínuo em um sistema discreto composto por uma malha geométrica que reduz o número de variáveis do problema a uma quantidade finita, tornando tratável um sistema contínuo a partir de um modelo discreto equivalente. As EDPs representam assim, o fenômeno associado ao meio contínuo discretizado e podem ser classificadas em três categorias básicas [4]:

- Elípticas – que modelam fenômenos estacionários, ou seja, cuja propriedade de interesse não se altera com o passar do tempo;
- Parabólicas – que modelam fenômenos transientes e dissipativos, ou seja, que apresentam mecanismos de dissipação de energia;
- Hiperbólicas – que modelam fenômenos transientes e não-dissipativos, ou seja, quando os fenômenos dissipativos são mínimos ou podem ser desprezados.

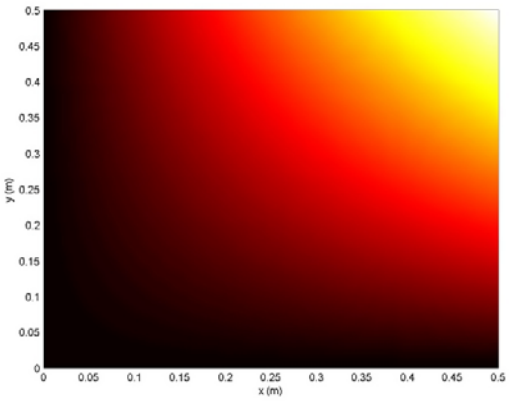

Figura 1 – Solução de uma EDP Elíptica representando a temperatura em uma chapa de metal

Uma característica importante destas categorias de equações é a formação de sistemas lineares com matrizes de banda quando o Método de Diferenças Finitas é aplicado para resolvê-las. Matrizes de banda implicam em fraca dependência de dados, excelente propriedade para a computação paralela em ambientes de memória distribuída.

2.2 Descrição do Problema

Definiu-se como o problema a ser estudado as equações diferenciais parciais elípticas, especificamente a equação de Poisson,

$$\nabla^2 u(x,y) \equiv \frac{\partial^2 u}{\partial x^2}(x,y) + \frac{\partial^2 u}{\partial y^2}(x,y) = f(x,y) \quad (1)$$

em $R = \{(x,y) \mid a < x < b, c < y < d\}$, com

$$u(x,y) = g(x,y) \text{ para } (x,y) \in S,$$

onde S indica o limite de R. Se f e g são contínuas em seus domínios então existe uma única solução para esta equação.

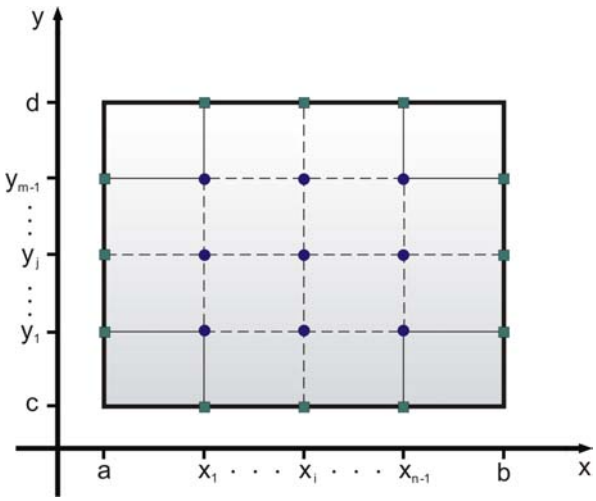

Figura 2 – Discretização dos domínios *R* e *S*. As circunferências representam pontos da grade pertencentes a *R*. Os quadrados representam os pontos de contorno pertencentes a *S*.

O primeiro passo para solucionar o problema é selecionar os números inteiros n e m e definir os

tamanhos de passo $h=(b-a)/n$ e $k=(c-d)/m$. Em seguida, deve-se realizar a partição do intervalo $[a,b]$ em n partes iguais de largura h e do intervalo $[c,d]$ em m partes iguais de largura k (Figura 2). O próximo passo é colocar uma quadrícula no retângulo R ao traçar linhas verticais e horizontais pelos pontos com coordenadas (x_i, y_j), onde

$$x_i = a + ih \text{, para todo } i = 0,1,...,n,$$

e

$$y_j = c + jk \text{, para todo } j = 0,1,...,m.$$

As linhas $x = x_i$ e $y = y_j$ são linhas da quadrícula, e suas intersecções são os pontos de rede da quadrícula. Para cada ponto de rede no interior da quadrícula (x_i, y_j) com $i = 1,2,...,n-1$ e com $j = 1,2,...,m-1$, utiliza-se a série de Taylor na variável x ao redor de x_i para gerar a fórmula das diferenças centrais [2]

$$\frac{\partial^2 u}{\partial x^2}(x_i, y_j) = \frac{u(x_{i+1}, y_j) - 2u(x_i, y_j) + u(x_{i-1}, y_j)}{h^2} - \frac{h^2}{12}\frac{\partial^4 u}{\partial x^4}(\xi_i, y_j), \quad (2)$$

onde $\xi_i \in (x_{i-1}, x_{i+1})$. Também utiliza-se a série de Taylor na variável y ao redor de y_j para gerar a fórmula das diferenças centrais

$$\frac{\partial^2 u}{\partial y^2}(x_i, y_j) = \frac{u(x_i, y_{j+1}) - 2u(x_i, y_j) + u(x_i, y_{j-1})}{k^2} - \frac{k^2}{12}\frac{\partial^4 u}{\partial y^4}(x_i, \eta_j), \quad (3)$$

onde $\eta_j \in (y_{j-1}, y_{j+1})$.

A utilização dessas fórmulas na Equação (1) permite expressar a equação de Poisson nos pontos (x_i, y_j) como [2]

$$\frac{u(x_{i+1}, y_j) - 2u(x_i, y_j) + u(x_{i-1}, y_j)}{h^2} + \frac{u(x_i, y_{j+1}) - 2u(x_i, y_j) + u(x_i, y_{j-1})}{k^2} = f(x_i, y_j) + \frac{h^2}{12}\frac{\partial^4 u}{\partial x^4}(\xi_i, y_j) + \frac{k^2}{12}\frac{\partial^4 u}{\partial y^4}(x_i, \eta_j),$$

para todo $i = 1,2,...,n-1$ e $j = 1,2,...,m-1$, e as condições de limite como

$u(x_0, y_j) = g(x_0, y_j)$ e $u(x_n, y_j) = g(x_n, y_j)$, para todo $j = 1,2,...,m$;

$u(x_i, y_0) = g(x_i, y_0)$ e $u(x_i, y_m) = g(x_i, y_m)$, para todo $i = 1,2,...,n-1$.

Na forma de equação de diferenças, isso resulta no método das Diferenças Finitas com um erro local de truncamento da ordem de $O(h^2 + k^2)$ [2]:

$$2\left[\left(\frac{h}{k}\right)^2 + 1\right]w_{ij} - \left(w_{i+1,j} + w_{i-1,j}\right) - \left(\frac{h}{k}\right)^2\left(w_{i,j+1} + w_{i,j-1}\right) = -h^2 f(x_i, y_j), \quad (4)$$

para todo $i = 1,2,...,n-1$ e $j = 1,2,...,m-1$, e

$$\begin{aligned} &w_{0j} = g(x_0, y_j) \text{ e } w_{nj} = g(x_n, y_j), \text{ para todo } j = 1,2,...,m; \\ &w_{i0} = g(x_i, y_0) \text{ e } w_{im} = g(x_i, y_m), \text{ para todo } i = 1,2,...,n-1; \end{aligned} \quad (5)$$

onde w_{ij} aproxima $u(x_i, y_j)$.

A equação em (4) envolve aproximações a $u(x, y)$ nos pontos

$$u(x_{i-1}, y_j), \quad u(x_i, y_j), \quad u(x_{i+1}, y_j), \quad u(x_i, y_{j-1}) \text{ e } u(x_i, y_{j+1}).$$

Reproduzindo a parte da quadrícula na qual esses pontos estão situados (Figura 3), observa-se que cada

equação contém aproximações em uma região em forma de cruz ao redor de (x_i, y_j). Agora, organizando-se a Equação (4), teremos

$$w_{ij} = \left(-h^2 f(x_i, y_j) + w_{i+1,j} + w_{i-1,j} + \lambda w_{i,j+1} + \lambda w_{i,j-1}\right)\mu, \quad (6)$$

onde $\lambda = \left(\frac{h}{k}\right)^2$ e $\mu = \frac{1}{2(1+\lambda)}$.

Utilizando a informação das condições de limite (5) sempre que for conveniente no sistema dado por (6), pode-se dizer que, em todos os pontos (x_i, y_j) adjacentes ao ponto de rede do limite, teremos um sistema linear $(n-1)(m-1)\times(n-1)(m-1)$, cujas incógnitas são as aproximações w_{ij} a $u(x_i, y_j)$ no interior dos pontos da rede.

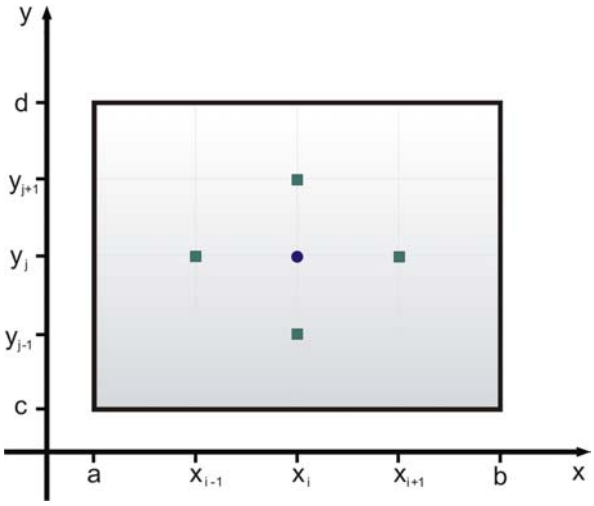

Figura 3 – Dependência de um determinado ponto em relação a seus vizinhos. O cálculo do ponto representado pelo círculo depende dos pontos representados pelos quadrados.

2.3 O Algoritmo Seqüencial

O sistema linear que contém as incógnitas w_{ij} em (6) será expresso mais claramente se introduzida uma remarcação dos pontos interiores da rede. Um sistema de remarcação desses pontos consiste em utilizar

$$P_l = (x_i, y_j) \text{ e } w_l = w_{i,j},$$

onde $l = i + (m-1-j)(n-1)$, para todo $i = 1,2,...,n-1$ e $j = 1,2,...,m-1$. Isso marca consecutivamente os pontos de rede da esquerda para a direita e de cima para baixo. Ao marcar os pontos desse modo, se garante que o sistema necessário para determinar w_{ij} seja uma matriz de banda com uma largura de banda de, no máximo, 2n-1.

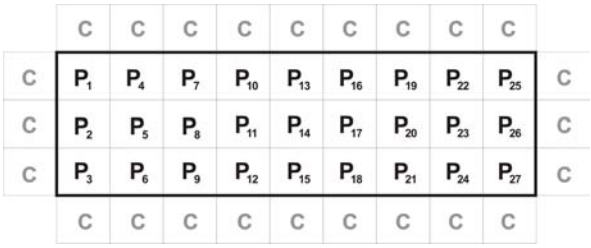

Figura 4 – Representação alternativa da malha de discretização com remarcação de pontos para n=10 e m=4.

É interessante utilizar também uma representação alternativa da malha de discretização como mostrado na Figura 4. Cada quadrado representa o valor de um determinado ponto remarcado, sendo que os quadrados marcados com P_l representam pontos do domínio R e os quadrados com um "C" representam as condições de contorno pertencentes ao domínio S. Os valores extremos da malha que fazem parte das condições de contorno foram omitidos na representação porque não são dependências de nenhum dos pontos pertencentes a R.

A representação desta forma permite uma análise computacional mais rica em relação à outra (Figura 2) porque é possível considerar cada quadrado como uma posição de um *array* bidimensional, facilitando a visualização do problema na escrita do programa.

O algoritmo a seguir utiliza a equação (6) para construir um sistema de equações lineares e resolvê-lo a partir das iterações de Gauss-Seidel sobre a matriz de discretização.

Método Seqüencial das Diferenças Finitas para a Equação de Poisson

Para aproximar a solução para a equação de Poisson

$$\nabla^2 u(x,y) \equiv \frac{\partial^2 u}{\partial x^2}(x,y) + \frac{\partial^2 u}{\partial y^2}(x,y) = f(x,y),$$

$$a \le x \le b, \quad c \le y \le d,$$

sujeita às condições de contorno (limites)

$$u(x,y) = g(x,y) \quad \text{se} \quad x = a \quad \text{ou} \quad x = b \quad \text{e}$$

$$c \le y \le d$$

e

$$u(x,y) = g(x,y) \quad \text{se} \quad y = c \quad \text{ou} \quad y = d \quad \text{e}$$

$$a \le x \le b;$$

ENTRADA: pontos extremos a, b, c, d; números inteiros $m \ge 3, n \ge 3$; tolerância *TOL*; número máximo de iterações *N*.

SAÍDA: aproximações $w_{i,j}$ a $u(x_i, y_j)$ para todo $i = 1,2,\ldots,n-1$, e para todo $j = 1,2,\ldots,m-1$, ou uma mensagem de que o número máximo de iterações foi excedido.

Passo 1 Faça $h = (a-b)/n$;

$k = (d-c)/n$.

Passo 2 Para $i = 1,\ldots,n-1$ faça $x_i = a + ih$;

$w_{i,0} = g(x_i, c)$;

$w_{i,m} = g(x_i, d)$. *(Condições de contorno)*

Passo 3 Para $j = 1,\ldots,m-1$ faça $y_j = c + jk$.

$w_{0,j} = g(a, y_j)$;

$w_{0,j} = g(b, y_j)$. *(Condições de contorno)*

Passo 4 Para $i = 1,\ldots,n-1$

para $j = 1,\ldots,m-1$, faça $w_{i,j} = 0$.

Passo 5 Faça $\lambda = h^2/k^2$;

$\mu = \dfrac{1}{2(1+\lambda)}$;

$l = 1$; $NORM = 0$.

Passo 6 Enquanto $l \le N$ siga os Passos 7 a 11 *(Os passos 7-11 realizam Gauss-Seidel)*

Passo 7 Para $i = 1,\ldots,n-1$

para $j = 1,\ldots,m-1$, faça

$z = (-h^2 f(x_i, y_j) + w_{i+1,j} + w_{i-1,j} + \lambda w_{i,j+1} + \lambda w_{i,j-1})\mu$;

se $|w_{i,j} - z| > NORM$, então

faça $NORM = |w_{i,j} - z|$;

faça $w_{i,j} = z$.

Passo 8 Se $NORM \le TOL$, então siga os Passos 9 e 10.

Passo 9 Para $i = 1,\ldots,n-1$

para $j = 1,\ldots,m-1$,

SAÍDA ($x_i, y_j, w_{i,j}$).

Passo 10 PARE.

(O procedimento foi bem-sucedido)

Passo 11 Faça $l = l + 1$.

Passo 12 SAÍDA("Número máximo de iterações excedido")

(O procedimento não foi bem-sucedido)

PARE.

2.4 O Algoritmo Paralelo

O algoritmo paralelo utiliza a mesma idéia de cálculo que foi proposta pelo seqüencial, ou seja, a discretização dos pontos da equação diferencial parcial elíptica de Poisson resultando em um sistema de equações lineares e a aplicação das iterações de

Gauss-Seidel para resolução deste sistema, mas agora com trocas de mensagem entre os processos.

Adicionalmente é preciso decompor o domínio dos dados entre os diversos processos que realizarão os cálculos. Um bom esquema de decomposição é deixar para os processos um bloco de linhas ou de colunas adjacentes, permitindo que a troca de mensagem entre eles se realize somente nos limites de cada bloco, como em [1]. Desta forma, os dados que necessitam ser recebidos são providos por, no máximo, dois processos diferentes, facilitando o controle e minimizando o *overhead* de comunicação.

A Figura 5 ilustra a mesma situação da Figura 4, mas agora com a decomposição do domínio de dados por colunas. Com este tipo de decomposição, um processo poderá realizar os cálculos em seu bloco independentemente e considerar como condições de contorno aquelas representadas pela Equação (5) e também colunas enviadas por um processo cujo bloco é adjacente. As flechas incluídas na Figura 5 permitem observar quais colunas de quais processos devem ser enviadas.

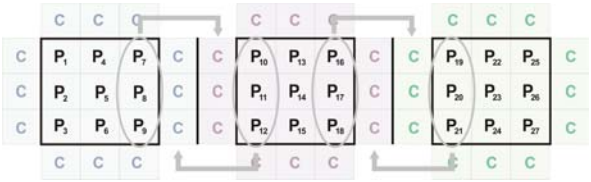

Figura 5 – Representação da malha de discretização com remarcação de pontos (n=10 e m=4) para 3 processos.

A escolha pelo método de decomposição por colunas ao invés de linhas se deu pelo fato de que a implementação utiliza Linguagem C. Como C implementa suas matrizes como um vetor de linhas e a matriz de discretização está transposta em relação à matriz da implementação (coordenada x que representa as colunas da grade é indexada pelas linhas no algoritmo), uma decomposição por colunas (linhas na matriz da implementação) permite que uma única chamada à rotina de envio ou recebimento faça a troca de mensagens entre dois processos vizinhos.

É importante observar que para um esquema de partição por blocos de colunas o número de linhas é fixo a todos os processos, enquanto que o número de colunas pode variar, pois nem sempre o número total de colunas da matriz de discretização será divisível pelo número de processos.

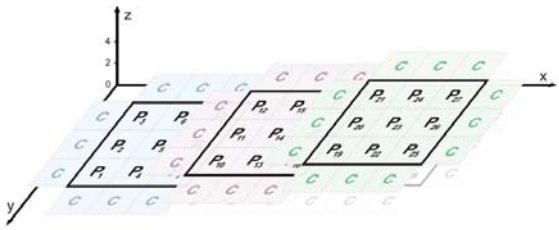

Figura 6 – Representação da malha de discretização com remarcação de pontos (n=10 e m=4) para 3 processos onde o eixo z representa o número do processo. Os pontos redundantes (passados aos vizinhos) estão sobrepostos. Uma projeção ortogonal da figura em z=1, indicada pela sombra, gera a Figura (4).

O algoritmo a seguir é uma modificação daquele citado em [1] e permite visualizar melhor a solução para o problema.

Iterações Paralelizadas de Gauss-Seidel em Diferenças Finitas

ENTRADA: Número de processos p, quantidade de linhas $m+1$ e de colunas $np_{id}+1$, onde id representa a identificação do processo.

SAÍDA: Cálculo das aproximações $w_{i,j}$.

(O algoritmo está simplificado a uma iteração)

Passo 1 Se $id \neq p-1$ então faça o envio da coluna $np_{id}-1$ de w para o processo $id+1$;
(Será condição de contorno)

Passo 2 Se $id \neq 0$ então faça o recebimento da

coluna 0 de w do processo $id-1$;

(Será condição de contorno)

Passo 3 Para $i = 1,...,np_{id} - 2$

para $j = 1,...,m-1$,

faça cálculo $w_{i,j}$ por Gauss-Seidel

Passo 4 Se $id \neq 0$ então faça o envio da coluna 1 de w para o processo $id-1$;

(Será condição de contorno)

Passo 5 Se $id \neq p-1$ então faça o recebimento da coluna np_{id} de w do processo $id+1$;

(Será condição de contorno)

Passo 6 Para $j = 1,...,m-1$,

faça o cálculo $w_{np_{id}-1,j}$ por Gauss-Seidel

Porém, antes do cálculo realizado pelo algoritmo anterior se faz necessário informar aos processos a quantidade de colunas que cada um se encarregará ($np_{id}+1$).

Para obter-se um balanceamento de carga razoável no particionamento é interessante que a diferença entre a quantidade de colunas dos blocos de dois processos quaisquer não ultrapasse o valor 1. Este fato permite que se possa afirmar a seguinte proposição:

Proposição: Seja nc o número de componentes participantes do particionamento e np o número de processos e $nc \geq np$. O tamanho máximo de uma partição qualquer, efetuado um particionamento com máxima distribuição de carga será

$$\|P\|_{\max} = \lceil nc/np \rceil, \text{ com } nc \geq np \text{ e } np > 0$$

Idéia da Demonstração: Dado nc qualquer, o tamanho de uma partição k com distribuição de carga razoável seria $\lfloor nc/np \rfloor + \varepsilon_k$, onde ε_k é parte do resto $nc \bmod(np)$, isto é,

$$\|P_k\| = \lfloor nc/np \rfloor + \varepsilon_k, \text{ onde } \sum_{i=1}^{np} \varepsilon_i = nc \bmod(np)$$

Para distribuição mais homogênea, divide-se $nc \bmod(np)$ entre $nc \bmod(np)$ partições, ou seja, $\varepsilon = 1$ para $nc \bmod(np)$ partições e $\varepsilon = 0$ para $np - (nc \bmod(np))$ partições. Portanto o tamanho máximo de uma partição qualquer será:

$$\|P\|_{\max} = \lfloor nc/np \rfloor + 1 = \lceil nc/np \rceil$$

O algoritmo a seguir realiza este particionamento balanceado.

Cálculo dos Limites das Partições do Domínio de Dados

ENTRADA: Número de processos p, quantidade total de colunas $n+1$, sendo $p < n$.

SAÍDA: Quantidade de colunas por processo $np_{id}+1$ e coluna inicial $inic_{id}$, onde id representa a identificação do processo.

Passo 1 Faça $quoc$ receber o quociente da divisão inteira de $n-1$ por p, e $rest$ receber o resto da mesma divisão;

Passo 2 Faça $np_{id} = quoc + 1$;

$inic_{id} = id \times quoc$;

Passo 3 Se $id < rest$ então faça

$inic_{id} = inic_{id} + id$ e $np_{id} = np_{id} + 1$;

senão faça

$inic_{id} = inic_{id} + rest$

Unindo os dois algoritmos anteriores com a versão seqüencial e, naturalmente, modificando o necessário, é possível construir um algoritmo paralelo que realize o Método das Diferenças Finitas, com razoável distribuição de carga, como se pode observar a seguir.

Método Paralelo das Diferenças Finitas para a Equação de Poisson

Para aproximar a solução para a equação de Poisson

$$\nabla^2 u(x,y) \equiv \frac{\partial^2 u}{\partial x^2}(x,y) + \frac{\partial^2 u}{\partial y^2}(x,y) = f(x,y),$$

$$a \le x \le b, \quad c \le y \le d,$$

sujeita às condições de contorno (limites)

$$u(x,y) = g(x,y) \quad \text{se} \quad x = a \quad \text{ou} \quad x = b \quad \text{e}$$
$$c \le y \le d$$

e

$$u(x,y) = g(x,y) \quad \text{se} \quad y = c \quad \text{ou} \quad y = d \quad \text{e}$$
$$a \le x \le b;$$

ENTRADA: pontos extremos a, b, c, d; números inteiros $m \ge 3, n \ge 3$; tolerância *TOL*; número máximo de iterações *N* e número total de processos p.

SAÍDA: aproximações $w_{i,j}$ a $u(x_i, y_j)$ para todo $i = 1,2,...,n-1$, e para todo $j = 1,2,...,m-1$, ou uma mensagem de que o número máximo de iterações foi excedido.

Passo 1 Faça o envio de p a todos os processos,
a atribuição de um número de identificação *id* aos processos;

Passo 2 Faça *quoc* receber o quociente da divisão inteira de $n-1$ por p,
rest receber o resto da mesma divisão;

Passo 3 Faça $np_{id} = quoc + 1$;
$inic_{id} = id \times quoc$;

Passo 4 Se $id < rest$ então faça $inic_{id} = inic_{id} + id$ e
$np_{id} = np_{id} + 1$;
senão faça $inic_{id} = inic_{id} + rest$

Passo 5 Faça $h = (a-b)/np_{id}$;
$k = (d-c)/np_{id}$.

Passo 6 Para $i = 1,..., np_{id} - 1$ faça

;
$w_{i,0} = g(x_i, c)$; $w_{i,m} = g(x_i, d)$.

(Condições de contorno)

Passo 7 Para $j = 1,..., m-1$ siga os passos 8 a 10

Passo 8 faça $y_j = c + jk$;

Passo 9 Se $id = 0$ então faça
$w_{0,j} = g(a, y_j)$;

Passo 10 Se $id = p-1$ então faça
$w_{0,j} = g(b, y_j)$.

(Condições de contorno)

Passo 11 Para $i = 1,..., np_{id} - 1$
para $j = 1,..., m-1$, faça $w_{i,j} = 0$.

Passo 12 Faça $\lambda = h^2 / k^2$;
$\mu = \frac{1}{2(1+\lambda)}$;
$l = 1$; $NORM = 0$.

Passo 13 Enquanto $l \le N$ siga os Passos 7 a 11

(Os passos 7-11 realizam Gauss-Seidel)

Passo 14 Se $id \neq p-1$ então faça o envio da coluna $np_{id}-1$ de w para o processo $id+1$;

(Será condição de contorno)

Passo 15 Se $id \neq 0$ então faça o recebimento da coluna 0 de w do processo $id-1$;

(Será condição de contorno)

Passo16 Para $i = 1,..., np_{id} - 2$

para $j = 1,..., m-1$, faça

$z = (-h^2 f(x_i, y_j) + w_{i+1,j} + w_{i-1,j} + \lambda w_{i,j+1} + \lambda w_{i,j-1})\mu$;

se $|w_{i,j} - z| > NORM$, então

faça $NORM = |w_{i,j} - z|$;

faça $w_{i,j} = z$.

Passo17 Se $id \neq 0$ então faça o envio da coluna 1 de w para o processo $id-1$;

(Será condição de contorno)

Passo18 Se $id \neq p-1$ então faça o recebimento da coluna np_{id} de w do processo $id+1$;

(Será condição de contorno)

Passo 19 Para $j = 1,..., m-1$,

$z = (-h^2 f(x_{np_{id}-1}, y_j) + w_{np_{id},j} + w_{np_{id}-2,j} + \lambda w_{np_{id}-1,j+1} + \lambda w_{np_{id}-1,j-1})\mu$

se $|w_{np_{id}-1,j} - z| > NORM$, então

faça $NORM = |w_{np_{id}-1,j} - z|$;

faça $w_{np_{id}-1,j} = z$

Passo 7 Faça o envio a todos os processos do maior valor de *NORM* para *GNORM*

Passo 8 Se $GNORM \leq TOL$, então siga os Passos 9 e 10.

Passo 9 Para $i = 1,..., np_{id} - 1$

para $j = 1,..., m-1$, SAÍDA

$(x_i, y_j, w_{i,j})$.

Passo 10 PARE. *(O procedimento foi bem-sucedido)*

Passo 11 Faça $l = l+1$.

Passo 12 SAÍDA("Número máximo de iterações excedido")

(O procedimento não foi bem-sucedido)

PARE.

2.5 A Implementação

Os métodos foram implementados em Linguagem C utilizando a implementação MPICH do padrão MPI para as trocas de mensagem.

*Uma abordagem por camadas foi **proposta** e utilizada para facilitar a depuração, a escalabilidade e o incremento da flexibilidade dos métodos como demonstrado na Figura 7.*

Figura 7 – Implementação em camadas do pacote de Métodos de Diferenças Finitas para problemas de valor de limite.

A camada STDLIB é formada pela biblioteca padrão da Linguagem C e, logicamente, não foi implementada.

Acima de STDLIB encontra-se BUFFER DE ARQUIVO, camada formada por módulos que

controlam a entrada e saída em arquivos binários necessários para problemas com muitas equações.

Os métodos numéricos utilizam as rotinas do buffer de arquivo e resolvem os sistemas lineares enviados pela INTERFACE PADRÃO DE DIFERENÇAS FINITAS. Há, inclusive, uma pequena modificação realizada para o aumento de desempenho no método paralelo de Gauss-Seidel para Diferenças Finitas: a passagem de mensagem aos pares.

Como MPI define suas rotinas padrão de passagem de mensagem ponto-a-ponto como bloqueantes, o algoritmo apresentado causa um gargalo nas trocas de condições de contorno, já que todos os processos ficam bloqueados até que o último receba as condições de seu vizinho. Portanto, se em uma primeira etapa os processos pares enviarem seus dados aos ímpares e na segunda etapa vice-versa, há um ganho maior de desempenho quando o número de processos é alto.

Atualmente somente EDP's Elípticas são suportadas, mas EDO's estão em fase de implementação e outras categorias de EDP's deverão ser projetadas com a finalidade de construir um pacote de métodos paralelos de resolução de problemas de valor de limite.

Como C é uma linguagem portável para várias plataformas e MPICH é disponível a sistemas baseados em UNIX e Microsoft Windows NT/2000/XP, o projeto pôde ser testado em Linux e Microsoft Windows XP com êxito.

3. Considerações Finais

A análise de desempenho, realizada com até 8 processos, demonstra que há ganho na utilização do método paralelo quando a ordem da matriz de discretização é 40, isto é, quando o número de equações resolvidas pelo método de Gauss-Seidel é maior que 1600.

Ressalta-se que a utilização de métodos que aumentam a convergência e o tempo de cálculo de cada iteração pode aumentar a eficiência da implementação paralela, pois cada iteração corresponde à troca de mensagens (norma calculada e condições de contorno atualizadas).

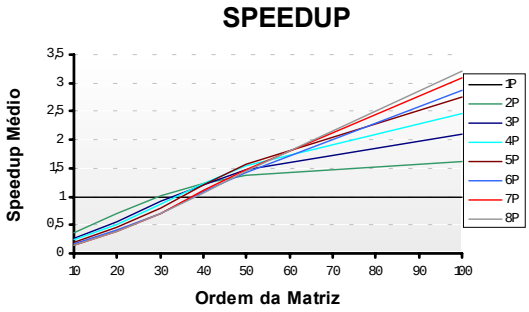

Figura 8 – Speedup do método para EDP Elíptica

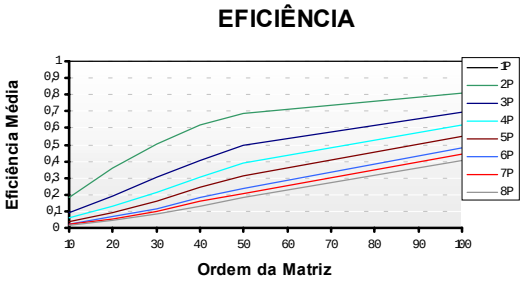

Figura 9 – Eficiência do método para EDP Elíptica

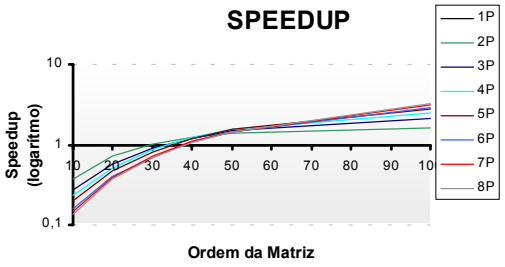

Figura 10 – Speedup do método para EDP Elíptica com speedup em escala logarítmica. Valores negativos indicam perda na utilização do algoritmo paralelo em relação ao seqüencial.

Com a implementação em camadas foi possível desenvolver de forma clara o método de Diferenças Finitas com grau tolerável de complexidade, já que a

plataforma Windows apresenta-se mais amigável em relação ao UNIX.

4. Agradecimentos

Este trabalho não poderia ser desenvolvido sem o apoio da Fundação de Amparo à Pesquisa do Estado de São Paulo – FAPESP - Processo 03/03631-0, do Departamento de Estatística, Matemática Aplicada e Computação da UNESP de Rio Claro e do Grupo de Pesquisa de Aplicações em Computação Paralela – GPACP também da UNESP de Rio Claro cujo portal é disponível em [7].

5. Referências de Suporte

[1] P. Amodio, F. Mazzia, A Parallel Gauss-Seidel Method for Block Tridiagonal Linear Systems, *SIAM J. Sci. Comput.,* (1995) Vol. 16, nº 6, 1451-1461.

[2] R. L. Burden, J. D. Faires, Análise Numérica, *Editora Thomson,* (2003) 605-614.

[3] R. D. da Cunha, Computação Paralela: Uma Breve Introdução, *XX CNMAC,* (1997) 5-20.

[4] A. O. Fortuna, Técnicas Computacionais para Dinâmica dos Fluidos, *Editora da Universidade de São Paulo,* (2000) 50-52, 174-186.

[5] R. H. C. Santana *et al*, Computação Paralela, USP - São Carlos, (1997) 1-2.

[6] E. M. Toledo, R. S. Silva, Introdução à Computação Paralela, *2ª Escola de Verão em Computação Científica - LNCC*, (1997) 20-21.

[7] *http://black.rc.unesp.br/gpacp*, Grupo de Pesquisa de Aplicações em Computação Paralela, (2004)

6. Apêndice – MPI e MPICH

O MPI foi desenvolvido para ser um padrão em ambientes de memória distribuída e troca de mensagem para computação paralela e, portanto, oferece as seguintes funcionalidades:

- **Comunicação Ponto-a-Ponto e Coletiva:** O MPI implementa todos os tipos de comunicação (bufferizada, não-bufferizada, bloqueante, não-bloqueante, etc).
- **Suporte a Grupos de Processos:** O MPI relaciona os processos em grupos e os identifica pela sua classificação dentro destes. Por exemplo, suponha um grupo contendo *n* processos; estes processos serão identificados utilizando-se números inteiros entre *0* e *n-1*. Essa classificação dentro do grupo é denominada ***rank***. Portanto, um processo no MPI é identificado por um grupo e por um *rank* dentro deste grupo. O MPI apresenta primitivas de criação e destruição de grupos de processos.
- **Suporte para Contextos de Comunicação:** Contextos são escopos que relacionam um determinado grupo de processos. São implementados para garantir que não existam mensagens que sejam recebidas ambiguamente por grupos de processos não relacionados. Assim, um grupo de processos ligados por um contexto não consegue se comunicar com um grupo que esteja definido em outro contexto. Este tipo de estrutura não é visível nem controlável pelo usuário, e seu gerenciamento fica a cargo do sistema. Para criação de contextos, o MPI utiliza-se do conceito de communicator, que é um objeto manuseado pelo programador e relaciona um grupo (ou grupos) de processos com um determinado contexto. Se existirem, por exemplo, aplicações paralelas distintas executando em um mesmo ambiente, para cada uma delas será criado um communicator. Isso criará contextos distintos que relacionarão os grupos de processos de cada aplicação e evitará que estes interfiram entre si.

- **Suporte para Topologias:** O MPI fornece primitivas que permitem o programador definir a estrutura topológica com a qual os processos de um determinado grupo se relacionarão. Como exemplo de uma topologia, pode-se citar uma malha, onde cada ponto de interseção na malha corresponde a um processo.

Atualmente existem diversas implementações do padrão, contendo todas as funcionalidades descritas anteriormente. Utilizamos a implementação MPICH do Argonne National Laboratory - Universidade de Chicago (Domínio Público);

MPICH é uma implementação de domínio público do padrão MPI portável para vários ambientes. As principais funcionalidades da versão 1.2.5 para Microsoft Windows NT4/2000/XP são [6]:

- Suporta completamente o MPI 1.2;
- Permite a execução de programas MPMD (Multiple Process Stream / Multiple Data Stream);
- Inclui ferramentas de visualização de performance - os jumpshots;
- Suporta os compiladores Microsoft Visual C++ 6.x, Microsoft Visual C++ .NET, Compaq Visual Fortran 6.X, Intel Fortran, gcc e g77;