Ordenação de Mensagens e Pré-Condicionamento na Solução Paralela do Gradiente Conjugado em Clusters de PCs Multiprocessados

Delcino Picinin Jr1, André L. Martinotto1, Rogério L. Rizzi1,2, Ricardo V. Dorneles1,3, Tiaraju A. Diverio1, Philippe O. A. Navaux1.

1PPGC, Instituto de Informática, UFRGS
CP 15064, 91501-970, Porto Alegre, RS, Brasil.
2Centro de Ciências Exatas e Tecnológicas, UNIOESTE, Campus de Cascavel
Rua Universitária, 2069, 85801-110, Cascavel, PR, Brasil.
3Departamento de Informática, Centro de Ciências Exatas e Tecnologia, UCS
Rua Francisco Getúlio Vargas, 1130, 95001-970, Caxias do Sul, RS, Brasil
{picinin, almartin, rizzi, cadinho}@inf.ufrgs.br

Resumo

Este trabalho apresenta uma solução paralela para o algoritmo do gradiente conjugado pré-condicionado de modo a resolver sistemas de equações lineares simétricos definidos-positivos em cluster multiprocessado. Essa solução paralela é obtida via decomposição de dados, onde o domínio computacional é particionado usando o algoritmo RCB (Recursive Coordinate Bisection), de modo a minimizar as comunicações e balancear a carga computacional. O modelo de programação utilizado é o SPMD e o paralelismo entre os nodos no cluster de PCs é explorado via troca de mensagens usando a biblioteca MPICH. A troca de mensagens entre os processos usa dois algoritmos de ordenação para evitar deadlock e melhorar o desempenho computacional. O paralelismo intra-nodal é explorado empregando a biblioteca Pthreads.

I - Introdução

O uso de *clusters*, na execução de algoritmos paralelos, vem tendo um aumento significativo nos últimos anos, devido ao seu baixo custo, a escalabilidade da arquitetura e o surgimento de rápidas redes de interconexão. *Clusters* são arquiteturas baseadas no agrupamento de máquinas independentes, interconectadas por uma rede de comunicação rápida. Podem ser formados por máquinas monoprocessadas e/ou multiprocessadas. No caso de multiprocessadas, além do paralelismo entre nodos, pode-se explorar o paralelismo intra nodos.

Neste trabalho explora-se o paralelismo em *clusters* multiprocessados dando ênfase nos aspectos de ordenação das mensagens entre os processos e no emprego de métodos para acelerar a convergência do algoritmo do gradiente conjugado, objeto de paralelização neste trabalho.

II. Métodos do Subespaço de Krylov Pré-condicionados

À classe do subespaço de Krylov pertencem aqueles métodos cuja iteração atual depende da iteração anterior. Os algoritmos dessa classe resolvem o sistema de equações $A\varphi=b$, encontrando soluções aproximadas φ^m para um subespaço m-dimensional, onde $\varphi^0 \perp K^m$, em que φ^0 é a aproximação inicial, impondo a condição $b-A\varphi^m \perp L^m$, onde L^m é um subespaço de dimensão m, e $K=K(A,r^0)$ é o subespaço de Krylov definido por $K(A,r^0)=\text{span}(r^0, Ar^0, A^2r^0, ..., A^{m-1}r^0)$, onde $r^0=b-A\varphi^0$ é o resíduo [7].

A escolha do método iterativo mais apropriado em cada situação depende das características da matriz dos coeficientes do sistema de equações. Neste trabalho, busca-se soluções paralelas onde as matrizes são esparsas, de grande porte e simétricas definidas-positivas (SDP). Nesse caso, o método de solução mais efetivo é aquele do gradiente conjugado (GC), dada sua eficiência, robustez e grau de paralelismo. A Fig. 1 mostra o algoritmo do GC empregado nesse trabalho.

$i \Leftarrow 0$
$r \Leftarrow b - Ax$
$d \Leftarrow M^{-1}r$
$\delta_{novo} \Leftarrow r^T d$
$\delta_0 \Leftarrow \delta_{novo}$
enquanto $i < i_{max}$ e $\delta_{novo} > \varepsilon^2 \delta_0$ faça
$q \Leftarrow Ad$
$\alpha \Leftarrow \delta_{novo} / d^T q$
$x \Leftarrow x + \alpha d$
$r \Leftarrow r - \alpha q$
$s \Leftarrow M^{-1}r$
$\delta_{velho} \Leftarrow \delta_{novo}$
$\delta_{novo} \Leftarrow r^T s$
$\beta \Leftarrow \delta_{novo} / \delta_{velho}$
$d \Leftarrow r + \beta d$
$i \Leftarrow i + 1$

Fig. 1: Algoritmo do GC pré-condicionado [7]

A. Pré-condicionamento

Não obstante a sua eficiência computacional, pode-se acelerar a convergência do algoritmo do GC através do emprego de pré-condicionadores. A taxa de convergência de algoritmos iterativos depende da distribuição dos autovalores da matriz, que pode ser melhorada transformando o sistema de equações original $A\varphi=b$ em um outro equivalente na forma $MA\varphi=Mb$, onde M é a matriz pré-condicionadora.

O pré-condicionamento pode ser implícito ou explícito. Na abordagem implícita obtém-se M de modo a aproximar a matriz A sem a necessidade de calcular M^{-1} diretamente. Na abordagem explícita deve-se calcular a matriz M^{-1} para aproximar A^{-1} diretamente. Em qualquer caso, o pré-condicionador deve ser obtido eficientemente e de modo que $M^{-1}A \approx I$, assegurando uma boa taxa de convergência com mínimo de custo computacional. Como a eficiência do pré-condicionador é, em geral, inversamente proporcional ao custo de sua obtenção [2] deve-se equilibrar o ganho da convergência com o custo do pré-condicionamento, de modo a diminuir o tempo total de execução do algoritmo.

Neste trabalho fez-se experiências numéricas com dois tipos de pré-condicionadores. Um deles é aquele baseado em aproximações polinomiais, e é empregado para obter o pré-condicionador para o solver global. Aproximações polinomiais baseadas na série de Neumann mantêm as características da matriz original quanto a positividade e simetria e, quando convenientemente truncada, também mantêm a esparsidade de A.

O segundo pré-condicionador emprega a abordagem de decomposição de domínio aditivo de Schwarz com sobreposição (MDDA) de modo a se obter o pré-condicionador já distribuído entre os processos. Nesse caso emprega-se a fatoração de Cholesky incompleta - IC(0) - e a fatoração de Cholesky diagonal incompleta - DIC(0) - para obter as aproximações para as soluções locais na etapa de pré-condicionamento.

A_1. Pré-condicionamento polinomial

Para se obter uma aproximação polinomial, que aproxima A^{-1} explicitamente, onde A é uma matriz SDP, decompõe-se inicialmente A como $A=D+L+L^T$, onde D, L e L^T são, respectivamente, a diagonal principal, a matriz triangular inferior e a matriz triangular superior, e escreve-se J como $J = -D^{-1}(L+L^T)$ e faz-se $A=D(I-J)$. Com isso, a inversa de A é escrita como uma série de potências como $A^{-1}=\sum(-1)^k[D^{-1}(L+L^T)]^kD^{-1}$, onde $0\le k\le m$. Para $k=0$ obtém-se o pré-condicionador de Jacobi (D^{-1}), e para $k=1$ o polinômio gerado não destrói a esparsidade de A. Nesse caso o pré-condicionador gerado é $M^{-1}=D^{-1}-D^{-1}(L+L^T)D^{-1}$.

A_2. Pré-condicionamento baseado no MDDA de Schwarz

O MDDA de Schwarz decompõe a solução do domínio $\Omega\subset\Re^{n\le3}$ em soluções de N subdomínios sobrepostos Ω_k, tal que $\Omega\subseteq\cup^n\Omega_k$, onde os contornos internos são denotados por $\Gamma_{ij}=\partial\Omega_i\cap\Omega_j$, onde $\partial\Omega$ denota a fronteira de Ω. A fronteira artificial Γ_i é à parte de Ω_i que é interior de Ω, e $\partial\Omega_i\backslash\Gamma_i$ são os pontos de $\partial\Omega_i$ que não estão em Γ_i. Nessa abordagem os subdomínios usam a solução da última iteração como condição de contorno, de modo que cada um deles é resolvido independentemente, e as comunicações ficam restritas às fronteiras.

O objetivo é usar essa abordagem gerando um pré-condicionador, implicitamente, já distribuído, onde o grau de paralelismo só depende do número de processadores disponíveis. Nesse caso, o passo $d\Leftarrow M^{-1}r$ (e o passo $s\Leftarrow M^{-1}r$) do GC da Fig. 1 pode ser obtido de modo distribuído para os p-subdomínios gerados pelo algoritmo de particionamento de malha, onde p é o número de processos.

Esse pré-condicionador é obtido através das soluções dos subproblemas $d_i=M_i^{-1}r_i$, com $0\le i\le p$. Especificamente, empregando as notações como em [9], se R_i denota a matriz, que aplicada ao vetor solução, retorna os valores associados com determinados nodos, e R_i^T mapeia os valores à matriz global A, então os valores de uma submatriz A_i, associada com uma sub-região Ω_i, são obtidos pela operação $A_i=R_iAR_i^T$. Usando essas notações pode-se mostrar que a matriz pré-condicionadora, gerada via MDDA de Schwarz, é expressa por $M^{-1}=\Sigma_iR_i^TA_i^{-1}R_i$. Essa abordagem gera uma solução chamada de Krylov-Schwarz (KS), que é um método de Schwarz acelerado por um método de Krylov [8]

Para aproximar o problema localmente, ou seja obter M_i^{-1} tal que $M_i^{-1}A_i\approx I_i$ foram empregadas duas fatorações incompletas, a IC(0) e a DIC(0). No primeiro caso escreve-se A como $A=LDL^T$, e o IC(0) é obtido em dois passos. No primeiro, determina-se um fator incompleto para L, e no segundo, resolve-se, a cada iteração do método do GC, dois sistemas triangulares. Um algoritmo para determinar o fator incompleto L_{ic} de $M_{ic}=L_{ic}DL_{ic}^T$ é mostrado na Fig. 2.

```
para k=1,2,...,n-1
    para i=k+1,...,n
        se (i,k)∈ S faça
            aux =a_ik
            a_ik=a_ik/a_kk
                para j=k+1,...,i
                    se (i,j)∈S e (j,k)∈S faça
                        a_ii=a_ii-aux.a_ik
```

Fig. 2: Algoritmo para o fator incompleto L_{ic} [4]

onde $S=\{(i,j);\ \forall a_{ij}\neq0\}$. Se os elementos a_{kk}, com $1\le k\le n$, são todos positivos, o processo termina com os fatores aproximados L_{ic} e D da matriz SDP, no qual se tem $d_{kk}=a_{kk}$ e $l_{ik}=a_{ik}$ se $i\in S$, ou 0 se $i\notin S$

Dispondo da aproximação L_{ic}, o pré-condicionador para o algoritmo do GC, montado no passo $d=M_{ic}^{-1}r$ (e $s=M_{ic}^{-1}r$), é

obtido como $d=M_{ic}^{-1}r \Leftrightarrow d=(L_{ic}DL_{ic}^{T})^{-1}r \Leftrightarrow (L_{ic}DL_{ic}^{T})d=r \Leftrightarrow L_{ic}Dz=r$ (para $z=L_{ic}^{T}d$) $\Leftrightarrow L_{ic}y=r$ (para $Dz=y$). Resolvendo o sistema triangular de equações $L_{ic}y=r$ obtém-se y. Sendo $Dz=y$ tem-se $z=D^{-1}y$ e resolvendo o sistema triangular de equações $z=L_{ic}^{T}d$ tem-se a solução d. Note-se que sendo a matriz SDP vale que $L_{ic}=L_{ic}^{T}$ e os dois sistemas são resolvidos com passos *backward* e *forward*.

No segundo caso, o DIC(0), é uma versão simplificada do ILC(0), onde escreve-se a matriz como $A=L+D+L^{T}$ e o pré-condicionador como $M=(D_{dic}+L)D_{dic}^{-1}(D_{dic}+L^{T})$, sendo D_{dic} o fator diagonal incompleto que deve ser calculado. Seu algoritmo está apresentado na Fig. 3.

```
para i=1,2,...
    d_ii=a_ii
    para i=1,2,....
        d_ii=1/d_ii
            para j=i+1,i+2,...
                se (i,j) ∈ S e (j,i)∈ S faça
                    d_jj=d_jj-a_ji d_ii a_ij
```

Fig. 3: Algoritmo para o fator incompleto D_{dic} [1]

Dispondo da aproximação D_{dic}, o pré-condicionador para o algoritmo do GC é montado no passo $d=M_{ic}^{-1}r$. Nesse caso não há necessidade de calcular M^{-1} diretamente, pois pode-se obtê-la como $d=M_{ic}^{-1}r \Leftrightarrow [(D_{dic}+L)D_{dic}^{-1}.(D_{dic}+L^{T})]d=r \Leftrightarrow [(D_{dic}+L)D_{dic}^{-1}]z=r$ (para $z=(D_{dic}+L^{T})d$) $\Leftrightarrow (D_{dic}+L)y=r$ (para $D^{-1}z=y$). Resolvendo o sistema triangular $(D_{dic}+L)y=r$ obtém-se y. Sendo $D^{-1}z=y$ tem-se $Dz=y$ e resolvendo-o sistema triangular $(D_{dic}+L^{T})d=z$ tem-se a solução d. Sendo a matriz SDP vale que $L_{ic}=L_{ic}^{T}$ e os dois sistemas triangulares são resolvidos com varreduras, *backward* e *forward*.

Como foi usado o estêncil de 5-pontos nas construções dos algoritmos numéricos, é necessário um halo de uma célula nas fronteiras dos sub-domínios. Portanto, após a obtenção das soluções locais, ou seja, do cálculo de $d_i=M_i^{-1}r_i$, com $0\leq i\leq p$, deve-se corrigir os valores das células adjacentes as fronteiras artificiais a fim de estabelecer condições de contorno do tipo Dirichlet homogêneas entre os sub-domínios. Neste trabalho a opção foi trocar, a cada iteração do GC, os valores do estêncil entre as células adjacentes e pegar a média de seus valores de modo a estabelecer as CC homogêneas nos passos $d\Leftarrow M^{-1}r$ e $s\Leftarrow M^{-1}r$ do algoritmo do GC.

III. Particionamento do Problema

Para se obter a solução paralela deve-se distribuir os dados entre os processadores gerando os subdomínios (ou processos). Esses processos devem ter as características que cada um tenha carga proporcional a sua capacidade de processamento e um mínimo de fronteiras de modo a minimizar o tempo de sincronização entre os processos, dado que aplicações paralelas são, geralmente, síncronas, e as comunicações são restritas às fronteiras.

O problema de particionamento de malha pode ser visto como um problema de particionamento de grafos, quando se considera o grafo $G=(V,A,p_a,p_v)$ composto pelo conjunto $V=\{0,...,n-1\}$ com n nodos; pelo conjunto $A\subseteq V\times V$ de arestas; pelos pesos p_a dos nodos; e pelos pesos p_v dos vértices. Então, pode-se associar, a cada nodo, processos ou dados, às arestas as comunicações, aos pesos p_a a carga computacional, e aos pesos p_v a carga de comunicação.

O problema de particionar o grafo em tantas partes quantos forem os processadores, visando a minimizar o número de arestas entre eles, é conhecido como k-particionamento, que consiste, no caso da malha gerada via discretização por diferenças finitas, em percorrer as células efetivas de cálculo do domínio, identificando-as e associando a cada uma delas a sua carga computacional (vértices) e a sua dependência de dados (arestas). A partir daí, pode-se gerar uma lista que contém a quantidade de vértices e de arestas, mais uma enumeração das relações entre cada vértice. Sendo esse um problema NP-difícil [3] as abordagens heurísticas são as únicas viáveis.

Neste trabalho empregou-se o algoritmo RCB (*Recursive Coordinate Bisection*) para gerar os sub-domínios não retangulares que são resolvidos via GC em paralelo. A Fig. 4 mostra o particionamento em 23 processos para um domínio usando o algoritmo RCB.

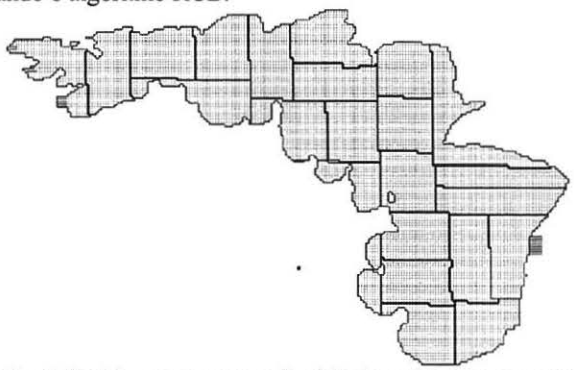


Fig. 4: Particionamento para o domínio do estudo de caso em 23 processos usando o algoritmo RCB

Esse particionamento foi gerado para o espaçamento horizontal $\Delta x=\Delta y=100$m. As espessuras das camadas verticais foram obtidas observando que a camada superior varia no tempo, pois depende da superfície livre. Com essas escolhas tem-se 185.124 vértices (células efetivas de cálculo) e 366.032 arestas quando com apenas 1 camada

IV. Ordenação de Mensagens e Armazenamento

No particionamento do domínio efetuado pelos algoritmos de particionamento de grafos, os subdomínios resultantes possuem n vizinhos, que devem se comunicar durante a solução do sistema de equações em paralelo para a troca de dados. Além disso, deve-se utilizar uma estrutura de dados

flexível de modo a conciliar a geração e o armazenamento da matriz e dos pré-condicionadores.

Na comunicação através de troca de mensagens, cada primitiva de envio em um processo deve estar associada a uma primitiva de recebimento no processo destino. Se isso não ocorrer as informações enviadas serão armazenadas em um *buffer*. De qualquer modo, mais importante que o controle das primitivas de envio, é o controle das primitivas de recebimento, pois se dois processos quiserem se comunicar simultaneamente recebendo dados um do outro eles ficarão em estado *deadlock*, mesmo com a existência de *buffer*.

Um exemplo de uma situação onde existe a possibilidade de ocorrer um *deadlock* pode ser apresentado baseando-se no domínio apresentado na Fig. 5. Neste domínio se o processo 14 esperar uma mensagem do processo 7, e ao mesmo tempo, o processo 7 esperar uma mensagem do processo 15, e o 15 uma mensagem do processo 14, os três processos ficarão em *deadlock*.

Além disso, embora o uso de *buffer* facilite o desenvolvimento de implementações, o uso de uma ordenação pode aumentar o desempenho da implementação. Assim, neste trabalho são desenvolvidas e implementadas duas estratégias de ordenação de mensagens que visam à diminuição do uso de *buffer* e a eliminação da possibilidade de ocorrer *deadlock*.

A. Ordenação de mensagens: método global

Essa estratégia é baseada na análise de um grafo gerado a partir do particionamento do domínio. Neste grafo os vértices representam os subdomínios (ou processos) e as arestas a comunicação. A estratégia consiste em eliminar as arestas em n fases de modo que cada eliminação seja armazenada em ordem de ocorrência em uma tabela de fases. Essa tabela deve conter a identificação dos dois vértices ligados pela aresta eliminada, formando uma *tupla*. A Fig. 5 ilustra a primeira fase, onde os processos e as comunicações são inicialmente identificadas.

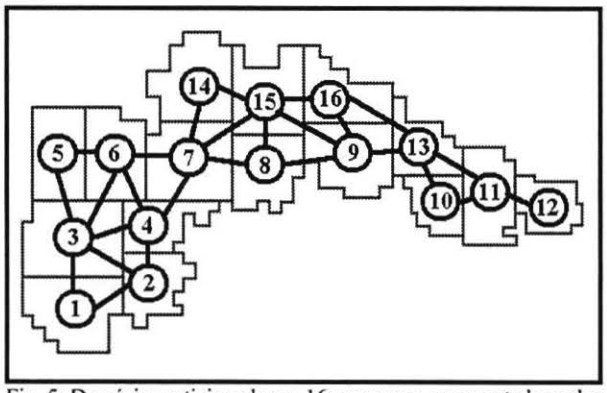


Fig. 5: Domínio particionado em 16 processos, representados pelos vértices, e onde a comunicação entre eles é representada pelas arestas

A decisão de qual a melhor ordem de eliminação das arestas adotada nessa estratégia é baseada em uma heurística que dá prioridade para a eliminação das arestas ligadas a vértices que possuem mais vizinhos. Devido ao fato dessa estratégia ser global, os dados obtidos na ordenação devem ser distribuídos entre os subdomínios de modo que eles recebam as *tuplas* em que sua identificação estiver contida. Após o recebimento das *tuplas* cada processo irá criar suas tabelas de ordenação de mensagens baseando-se na ordem em que as *tuplas* foram recebidas.

Essa tabela de ordenação deve conter duas vezes a identificação de cada vizinho, uma abaixo da outra, sendo uma associada à ação de envio e a outra a ação de recebimento. A decisão por enviar ou receber mensagens para cada vizinhos é baseada no valor de suas identificações dando prioridade de envio para os processos com identificador maior, assim eliminando a possibilidade de criação de um ciclo de dependência e conseqüentemente um *deadlock*. A Tab. 1 mostra a seqüência completa de fases para a ordenação das mensagens considerando-se o domínio da Fig. 5. Nessa tabela as *tuplas* sombreadas mostram a seqüência de dados que devem ser enviados na distribuição, onde o processo receptor deve ser o # 7.

Tab. 1: Fases globais para o domínio ilustrado na Fig. 5

Ciclo	Vértice	Vértice	Ciclo	Vértice	Vértice
1	16	9	2	4	3
1	15	8	3	16	15
1	14	7	3	13	9
1	13	10	3	11	10
1	12	11	3	7	4
1	6	4	3	3	1
1	5	3	4	15	14
1	2	1	4	9	8
2	16	13	4	7	6
2	15	9	4	4	2
2	13	11	5	15	7
2	8	7	5	3	2
2	6	5	6	6	3

As Tab. 2 e 3 apresentam, respectivamente, um exemplo de uma parte da tabela global de fases atribuída a um processo na distribuição (processo #7), e da tabela contendo a ordem de mensagens resultante.

Tab. 2: Fases locais para o processo # 7 (método global)

Ciclo	Vértice	Vértice	Ciclo	Vértice	Vértice
1	14	7	3	7	4
2	8	7	4	7	6
continua ao lado...			5	7	15

Tab. 3: Ordenação resultante para o processo # 7

Vértice	Ação	Vértice	Ação
14	Recebe	4	Recebe
14	Envia	6	Envia
8	Recebe	6	Recebe
8	Envia	15	Recebe
4	Envia	15	Envia

B. Ordenação de mensagens: método local

Essa abordagem requer apenas informações locais, que são os identificadores dos subdomínios (processos) vizinhos. Semelhante à estratégia descrita anteriormente, essa segunda estratégia também é desenvolvida em n fases. Porém, esse algoritmo cria apenas a tabela de fases local e a utiliza para criar uma tabela de ordenação. A decisão de qual a ordem em que o identificador dos vizinhos deve entrar nessa tabela de ordenação e qual a ação que deve ser associada (recebimento ou envio de mensagens) é baseada na criação de identificadores lógicos para cada processo vizinho e para o processo local. A Fig. 6 ilustra a primeira fase, onde os processos e as comunicações são identificadas, destacando o processo de número 7 que é utilizado no exemplo.

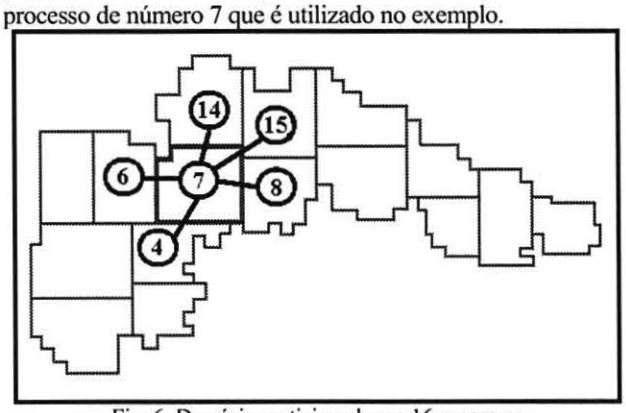


Fig. 6: Domínio particionado em 16 processos

Esses identificadores lógicos devem ser armazenados em uma linha da tabela de fases, sendo que em cada fase do algoritmo esses identificadores lógicos devem ser alterados até que todas as comunicações tenham sido ordenadas. O algoritmo se baseia no fato de que cada identificador lógico de um subdomínio (processo) deve estar na tabela de seus vizinhos com o mesmo valor e que suas alterações sigam a mesma regra.

Após possuir os identificadores lógicos é aplicado um algoritmo em *n* fases, sendo que em cada fase são efetuados 2 passos. O primeiro passo é executar uma estratégia par-ímpar onde os nodos cujo identificador lógico tem paridades diferentes da paridade do identificador lógico local são substituídos por um elemento nulo da tabela de identificadores lógicos e seus identificadores reais são colocados na tabela de ordenação de mensagens. O segundo passo é alterar os identificadores lógicos restantes e voltar para o passo anterior até que o único identificador lógico existente seja o do processo local. A alteração dos identificadores lógicos é baseada em sua paridade, os pares são divididos por dois e os ímpares são transformados em pares com a soma da unidade e, então, também são divididos por dois.

A colocação dos identificadores reais na tabela de ordem de mensagens deve ser de tal forma que os processos com identificador lógico par sejam colocados em duas posições na tabela, na primeira delas associados à ação de envio e na segunda posição associados à ação de recebimento. Como uma fase da comunicação só inicia após a anterior terminar e como com pares e ímpares não é possível criar um ciclo em apenas uma fase não existe a possibilidade de ocorrer *deadlock*. As Tab. 4 e 5 apresentam um exemplo de fases e da ordem de mensagens, resultante para esse processo (processo #7) ao final do procedimento de ordenação das mensagens.

Tab. 4: Fases para o processo # 7 (método local)

	Vizinhos					Local
	Identificadores Reais					
	6	4	8	15	14	7
Fase	Identificadores Lógicos					
1	6	4	8	15	14	7
2	–	–	–	8	–	4
3	–	–	–	4	–	2
4	–	–	–	2	–	1
5	–	–	–	–	–	1

Tab.5: Ordenação resultante para o processo # 7

Tabela de Ordenação do Processo 7			
Vértice	Ação	Vértice	Ação
6	Recebe	4	Envia
4	Recebe	8	Envia
8	Recebe	14	Envia
14	Recebe	2	Recebe
6	Envia	2	Envia

As implementações dos algoritmos que usam essas tabelas (Tab. 5 e Tab. 3) de ordenação permitem o recebimento de mensagens antecipadamente a sua colocação na tabela.

C. Geração e armazenamento do sistema de equações

Empregando o estêncil de 5-pontos gera-se sistemas de equações lineares SDP escritos, onde "D" , "E", "C", "F" e "S" são os coeficientes do sistema; "B" é o vetor dos termos independentes; e Φ é a variável a ser calculada em cada célula do domínio, como [6]:

$$\left.\begin{aligned} E_{(i,j-1)}\Phi^{n+1}_{(i,j-1)} + D_{(i,j+1)}\Phi^{n+1}_{(i,j+1)} + C_{(i,j)}\Phi^{n+1}_{(i,j)} + F_{(i-1,j)}\Phi^{n+1}_{(i-1,j)} \\ + S_{(i+1,j)}\Phi^{n+1}_{(i+1,j)} = B^{n}_{(i,j)} \end{aligned}\right\} (1)$$

Para montar o sistema de equações um procedimento varre todas as células testando se as fronteiras são internas ou contornos do domínio e, se são abertas ou fechadas após o particionamento do domínio e, consequentemente, a geração de fronteiras artificiais entre os subdomínios. Identifica-se quais são as faces *interna e aberta*, relativo ao coeficiente "C" entre os tipos "F", "S", "D" e "E". Considerando-se a CC *outflow* gradiente nulo entre as faces das fronteiras abertas adiciona-se ao coeficiente "C" da célula, na montagem da matriz, o respectivo coeficiente da CC *outflow*. As CC verdadeiras (de Dirichlet) são agregadas ao vetor dos termos independentes. Ao final desse processo a matriz dos coeficientes está gerada.

Note-se, porém, que dado a dependência de dados decorrente do estêncil computacional e visando construir pré-

condicionadores via MDDA, é necessário uma sobreposição (estêncil) de no mínimo uma célula para armazenar a posição e o conteúdo dos elementos adjacentes a essas fronteiras. A Fig. 7 ilustra um exemplo para as células internas e de fronteiras de subdomínio, caracterizando as comunicações necessárias na solução do sistema de equações e quando da formação do pré-condicionador para o método MDDA.

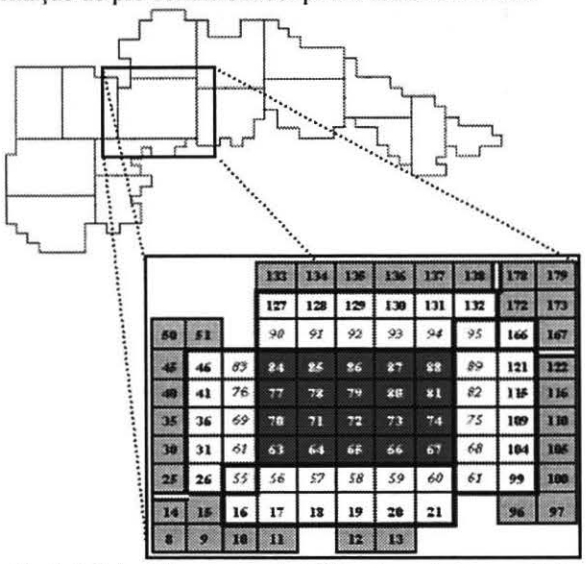


Fig. 7: Subdomínio gerado pelo RCB onde se destaca as células internas e as células de fronteira e as suas enumerações locais

Cabe observar que o procedimento de geração da matriz é feito após o particionamento do domínio e, portanto, as partes do sistema de equações (e as submatrizes locais) são geradas em paralelo e já estão distribuídas, cada processo possuindo apenas parte da matriz global. Também possuem apenas parte do vetor que durante a solução iterativa será multiplicado pela matriz. Para efetuar essa multiplicação a parte da matriz gerada a partir das informações locais do domínio possui os elementos do vetor localmente, e seu acesso pode ser feito pelo índice da coluna na matriz.

Note-se que sendo os domínios não retangulares e o estêncil utilizado do tipo 5-pontos, a matriz de coeficientes gerada é esparsa, de grande porte e contêm, além do elemento da diagonal principal, até quatro outros elementos por linha. A matriz dos coeficientes é armazenada no formato CSR (*Compressed Sparse Row*) que armazena somente os elementos não nulos em posições contíguas na memória. Se n é o número de elementos não nulos da matriz, o formato CSR armazena n+N+1 posições, em vez de N^2. Porém, o armazenamento dos elementos das fronteiras dos vizinhos adjacentes (estêncil) é feito individualmente em vetores pré-ordenados.

Na Fig. 8 são representados os elementos do vetor local, denotados por "x", e os elementos da matriz, denotados por "a" e por "c", onde "a" e "c" são os coeficientes "E" ou "D" ou "F" ou "S" ou "C", respectivamente de (1). Já os elementos da matriz formados pelo estêncil são representados por "e", e são aqueles que necessitam receber elementos do vetor que estão em outros processos. Esses elementos são representados por "v".

$$
\begin{array}{ll}
\textit{Vetor} & v_{n-3}\ \ 0\ \ v_{n-1}\ \ \{x_n\ \ x_{n+1}\ \ x_{n+2}\ \ x_{n+3}\ \ x_{n+4}\ \ x_{n+5}\}\ \ v_{n+6}\ \ 0\ \ v_{n+8} \\
\textit{Matriz} & \left\{\begin{array}{cccccccccccc}
e_{n-3,n} & 0 & 0 & c_n & a_{n+1,n} & 0 & a_{n+3,n} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & a_{n,n+1} & c_{n+1} & a_{n+2,n+1} & 0 & 0 & 0 & 0 & 0 & e_{n+8,n+1} \\
0 & 0 & e_{n-1,n+2} & 0 & a_{n+1,n+2} & c_{n+2} & 0 & a_{n+4,n+2} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & a_{n,n+3} & 0 & 0 & c_{n+3} & 0 & a_{n+5,n+3} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & a_{n+2,n+4} & 0 & c_{n+4} & a_{n+5,n+4} & e_{n+6,n+4} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & a_{n+3,n+5} & a_{n+4,n+5} & c_{n+5} & 0 & 0 & 0
\end{array}\right.
\end{array}
$$

Fig. 8: Representação para os elementos necessários em Ax

Baseando-se na Fig. 8 será descrita a estratégia empregada para armazenar os elementos necessários para obter o produto matriz por vetor Ax.

Uma primeira abordagem seria alocar o vetor local (x) com tamanho correspondente a todo o domínio e colocar os elementos nas posições correspondentes a sua localização no vetor global, mas, nesse caso, se utilizaria uma grande quantidade de memória sem necessidade real, como pode ser visto na Fig 9.

$$\textit{Vetor}\ \ \{v_{n-3}\ \ 0\ \ v_{n-1}\ \ x_n\ \ x_{n+1}\ \ x_{n+2}\ \ x_{n+3}\ \ x_{n+4}\ \ x_{n+5}\ \ v_{n+6}\ \ 0\ \ v_{n+8}\}$$

Fig. 9: Representação para o vetor x local de tamanho global

Alternativamente, outra abordagem seria armazenar as partes dos vetores recebidas separadamente para cada vizinho, como pode ser visto na Fig. 10. Desse modo, tem-se a identificação completa dos elementos de cada vizinho sem uso desnecessário de memória. Porém essa abordagem não se mostrou totalmente satisfatória devido à necessidade de se ter um controle para saber quais são as células (elementos) na matriz (A) que representam os estênceis , e a qual estêncil essas células pertencem.

$$\textit{Vetor}\ \ \{v_{n-3}\ \ v_{n-1}\}\ \ \ \{x_n\ \ x_{n+1}\ \ x_{n+2}\ \ x_{n+3}\ \ x_{n+4}\ \ x_{n+5}\}\ \ \ \{v_{n+6}\ \ v_{n+8}\}$$

Fig. 10: Representação para o vetor x de modo separado

A solução adotada foi armazenar a parte da matriz pertencente ao estêncil separadamente para cada vizinho, e em forma de vetores. Assim as partes da matriz equivalente aos estênceis estariam associadas aos vetores recebidos na comunicação e a um vetor contendo a enumeração da posição onde os resultados das operações devem ser colocados. Essas informações são obtidas baseando-se na enumeração das células do domínio do problema, conforme pode ser visto na Fig 7. Uma ilustração do formato de armazenamento do estêncil utilizado é como na Fig. 11.

Vetor x recebido	$\{v_{n-3}$	$v_{n-1}\}$	$\{v_{n+6}$	$v_{n+8}\}$
Vetor estencil	$\{e_{n-3,n}$	$e_{n-1,n+2}\}$	$\{e_{n+6,n+4}$	$e_{n+8,n+1}\}$
Vetor linhas	$\{n$	$n+2\}$	$\{n+4$	$n+1\}$

Fig. 11: Representação para o formato de armazenamento empregado neste trabalho para os estênceis

Desse modo, a multiplicação matriz esparsa por vetor é dividida em duas partes. Ou seja, $Ax=Ax_{cel\_int}+Ax_{cel\_est}$, onde Ax_{cel_int} denota a parte desse produto matriz×vetor em que apenas se considera as células internas ao domínio, e onde Ax_{cel_est} denota a parte desse produto matriz×vetor referente aos estênceis. Essa estrutura facilita a incorporação dos pré-condicionadores baseados em decomposição de domínio.

V. Solução Paralela: Paralelismo Inter e Intra-Nodal

O algoritmo do gradiente conjugado, é composto por operações entre vetores e matrizes, como soma e produto escalar de vetores e multiplicação matriz vetor. Na versão implementada do GC, as operações de álgebra linear que foram paralelizadas são:

1) Soma/subtração de vetores, onde cada nodo executa a operação sobre suas partes do vetor;
2) Multiplicação de escalar por vetor, onde cada nodo executa a multiplicação do escalar pela parte do vetor que possui;
3) Produto escalar, onde cada nodo calcula a sua parte do produto escalar com os dados que possui, e depois é feita uma operação de redução com outros nodos para que o produto escalar final possa ser calculado somando os resultados obtidos por cada nodo;
4) Multiplicação matriz esparsa por vetor, que é a operação de maior custo computacional, pois para executar esta, cada nodo necessita saber informações sobre parte dos vetores dos nodos vizinhos.

Na paralelização do algoritmo do gradiente conjugado, existem dois pontos de sincronismo: após as operações de produto escalar locais, onde ocorre uma operação de redução para obter o produto escalar global, e antes das operações de multiplicação matriz esparsa por vetor, onde são trocadas informações sobre partes dos vetores.

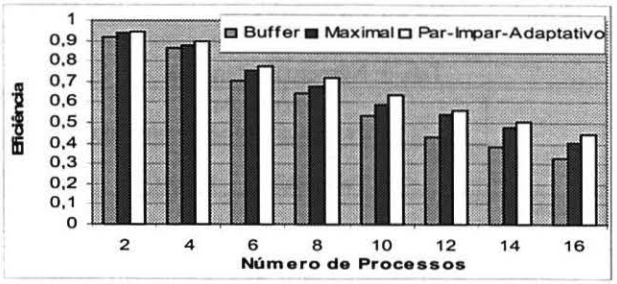
No sincronismo intra nodos, surgiram duas alternativas de implementação. A primeira seria separar as operações em pequenas *threads*, cria-las e destruí-las de acordo com a necessidade, sendo a thread principal responsável por criar as pequenas *threads* e após a execução de todas destruí-las, e então repetir o processo na próxima operação. Essa abordagem foi avaliada no trabalho [5]. Já a segunda alternativa seria manter a estrutura do programa que explorava o paralelismo entre nodos, e criar *threads* secundárias responsáveis por todas as operações, sendo o sincronismo entre as operações contidas nas *threads* feito através de barreiras.

A opção foi pela segunda alternativa pois ela mostrou-se mais eficiente devido ao fato de nem todas as operações necessitarem de sincronismos, somente as operações que antecedem a comunicações entre os processos.

VI Resultados Obtidos

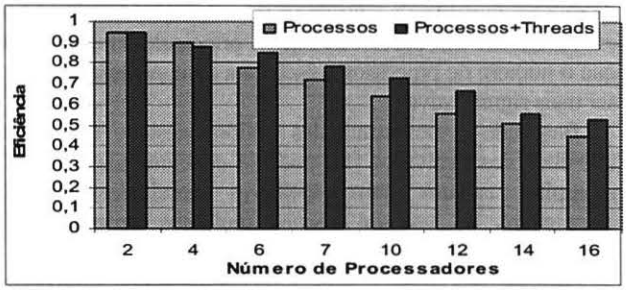
Todos os algoritmos foram implementados utilizando a linguagem de programação C e a biblioteca de trocas de mensagens MPICH 1.2.1, sobre o S.O. Linux. Os testes foram feitos em um *cluster* formado por nodos *Pentium* III 550MHz duais conectados pela rede *Fast Ethernet*. O erro utilizado como critério de convergência do GC foi 10^{-8}.

Conforme já descrito foram usadas duas estratégias de ordenação de mensagens, as quais foram comparadas com o uso apenas do *buffer*. Os resultados estão apresentados na no Graf. 1. Os demais testes foram feitos usando a estratégia local, pois foi a abordagem que apresentou melhores resultados.



Graf. 1: Resultados obtidos com estratégias de ordenação.

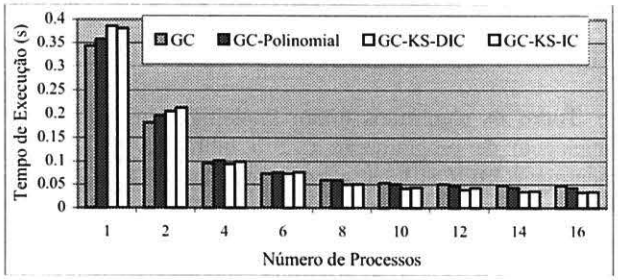
Os resultados obtidos com o uso de múltiplas *threads* são apresentados no Graf. 2, onde se pode notar que quando da diminuição da carga computacional, as múltiplas *threads* obtiveram uma maior eficiência. No Graf. 2 comparou-se execuções com o mesmo número de processadores e não de processos, pois um processo com duas *threads* faz uso do mesmo número de processadores que dois processos.



Graf. 2: Comparação entre o uso ou não de múltiplas *threads*.

Nos demais testes do GC, onde se analisou o desempenho computacional do algoritmo com e sem pré-condicionadores, utilizou-se apenas múltiplos processos. No decorrer do trabalho pretende-se desenvolver as versões pré-condicionadas do GC também fazendo o uso de múltiplas *threads*. Assim, no

graf 3. comparou-se os tempos de execução (em segundos) do algoritmo do GC sem pré-condicionamento com os obtidos com três diferentes tipos de pré-condicionadores.



Graf. 3: Resultados obtidos no uso de pré-condicionadores

A Tab. 6 mostra o número de iterações do algoritmo do GC usando os diferentes pré-condicionadores. Nesse caso procurar-se diminuir o número de iterações em cada ciclo, onde o ciclo é definido como o número total de iterações necessárias para obter a acurácia desejada

Tab. 6: Número de iterações

	Versões do GC (sem e com pré-condicionamento)			
	GC	GC Polinomial	GC KS-DIC	GC-KS-IC
# Iterações	8	4	3	3

VII Conclusões

As principais contribuições do artigo é o desenvolvimento e a implementação de uma política de comunicação, que pode ser indispensável quando da falta de *buffer* na biblioteca de troca de mensagens, e de um formato e padrão para o armazenamento e a montagem dos sistemas de equações pré-condicionados por métodos de decomposição de domínio.

Os resultados obtidos mostram um bom ganho de desempenho no uso de algoritmos de ordenação de mensagens em relação ao uso apenas do buffer, principalmente à medida que o número de processos cresceu e a comunicação passou a ser mais significativa.

Baseando-se nos resultados obtidos neste e em outros trabalhos desenvolvidos [5], pode-se concluir que o uso de múltiplas *threads* tem melhores resultados quando comparados ao uso de apenas múltiplos processos em subdomínios de dimensões menores.

Note-se que o GC com pré-condicionador polinomial reduziu o número de iterações de 8 para 4, mas apresentou tempos de execução similares ao obtidos com o GC sem pré-condicionador. Os pré-condicionadores tipo fatoração incompleta reduziram o número de iterações de 8 para 3 e apresentaram um menor tempo de execução em relação ao GC sem pré-condicionador, à medida que o número de subdomínios cresceu e conseqüentemente as matrizes de coeficientes dos mesmos ficaram menores.

Agradecimentos

Este trabalho foi financiado parcialmente pelo CNPq e FAPERGS. Agradecemos ao Prof. Dr. César A. F. De Rose por nos possibilitar o uso do *cluster* do CPAD PUCRS/HP.

Referências Bibliográficas

[1] BARRETT, R., BERRY, M., CHAN, T. F., DEMMEL, J., DONATO, J. M., DONGARRA, J., EIJKHOUT, V. POZO, ROLDAN, ROMINE, C., VORST, H. V. Templates for the Solution of Linear Systems: building Blocks for Iterative Methods. 1994. Disponível em http://www.netlib.org.

[2] CATABRIGA, L. Estudo de Pré-condicionadores para o Método GMRES usando Estruturas de Dados por Arestas. Seminário de Qualificação ao doutorado, COPPE, UFRJ, Maio 1998. Disponível em http://www.coc.ufrj. br/~luci/.

[3] GAREY, M. R., JOHNSON, D. S. Computer and Intractability: a guide to the theory of NP-completeness. Freeman, San Francisco, 1979.

[4] JÚDICE, J. J., PATRICIU, J.M. Sistemas de Equações Lineares. Departamento de Matemática da Universidade de Coimbra, Coimbra, Portugal, 1996.

[5] MARTINOTTO, A. L., FRIZZO, E. J., DORNELES R. V., DIVERIO, T. A. Paralelização do Método do GMRES com MPI e *Threads*, SCIENTIA, São Leopoldo, 2002. (em avaliação)

[6] RIZZI, R. L., ZEFERINO, C. A., DORNELES, R. V., NAVAUX, P. O. A., BAMPI, S., SUZIN, A. A., DIVERIO, T. A. Fluvial Flowing of Guaiba River Estuary: A Parallel Solution for the Shallow Water Equations Model in: Proceedings of the Fourth Vecpar. 2000. p. 895-896, Porto, Portugal, 2000.

[7] SHEWCHUCK, Jonathan. R. An Introduction to the Conjugate Gradient Method Without the Agonizing Pain. School of Computer Science. Carnegie Mellon University, 1994.

[8] SILVA, R. S. et al. Iterative Local Solvers for Distributed Krylov-Schwarz Method Applied to Convection-Diffusion Problems. Computer Methods for Applied Mechanics and Engineering, Vol. 149, 353-362, 1997.

[9] SMITH, B., BJORSTAD, P., GROPP, W., Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations. Cambridge University, 1996.