UMA IMPLEMENTAÇÃO PARALELA EFICIENTE DO MÉTODO DOS ELEMENTOS FINITOS PARA A EQUAÇÃO DE ADVECÇÃO E DIFUSÃO

Leonardo Muniz de Lima
Bruno Zanetti Melotti
Lucia Catabriga
Andréa Maria Pedrosa Valli
{*lmuniz, bzmelotti, luciac, avalli*}*@inf.ufes.br*
Laboratório de Computação de Alto Desempenho - LCAD
Departamento de Informática - Universidade Federal do Espírito Santo
Av Fernando Ferrari, 514, Goiabeiras, 29075/910, Vitória, ES, Brasil

Resumo. *O presente trabalho apresenta uma implementação do método dos elementos finitos para a equação de convecção-difusão bidimensional utilizando estratégia de decomposição de domínio com estruturas de blocos orientados da matriz de discretização resultante. O sistema linear de equações proveniente da formulação do método dos elementos finitos é resolvido através do método iterativo não-estacionário GMRES. São apresentados três formatos de armazenamento de matrizes esparsas oriundas da discretização sendo feito um estudo comparativo entre eles. Os esquemas de armazenamento empregam versões paralelas da estratégia elemento por elemento, aresta por aresta e do tradicional formato de linhas esparsas comprimidas. A implementação é desenvolvida para arquiteturas de memória distribuída, particularmente para clusters de estações de trabalho, e a troca de mensagens entre os processadores é efetuada através do padrão MPI.*

Palavras chaves. *Elementos finitos, Estratégias especiais de armazenamento, GMRES, Processamento paralelo, MPI.*

1. INTRODUÇÃO

O método dos elementos finitos é uma técnica geral para construção de soluções aproximadas para problemas de valor de contorno que envolve divisão do domínio de solução em um número finito de subdomínios, denominados elementos finitos (Hughes, 1987). A discretização pelo método dos elementos finitos conduz à solução de sistemas lineares de equações. As matrizes geradas por essa discretização têm uma enorme esparsidade e seus coeficientes não nulos não são dispostos uniformemente. Os métodos iterativos não-estacionários, também conhecidos como métodos livres de matrizes, são mais indicados nesse caso, pois apresentam facilidades no armazenamento das matrizes esparsas (Saad, 1995). Além disso, a principal operação desses métodos, o produto matriz-vetor, pode ser realizada acessando apenas as posições não nulas da matriz dos coeficientes. Entretanto, para que essa vantagem dos métodos não-estacionários possa ser melhor aproveitada, há necessidade de se definir estratégias de armazenamentos especiais das matrizes envolvidas. Existem várias estratégias de armazenamento aplicadas ao método dos elementos finitos. A mais popular é o armazenamento elemento por elemento (EBE) (Hughes, 1987), mas existem outras conhecidas, como aresta por aresta (EDE) (Martins, 1996; Catabriga, 2000) e a tradicional estratégia de linhas esparsas comprimidas, do inglês, *Compressed Sparse Row* (CSR) (Saad, 1995).

Mesmo utilizando estratégias de armazenamento especiais certas classes de problemas demandam grande tempo de processamento e alto consumo de memória. Atualmente, a programação paralela tem mostrado ser uma forma bastante eficaz na busca por um melhor desempenho para realizar essa tarefa. Diversas inovações têm sido apresentadas nessa área, desde arquiteturas específicas a versões paralelas de muitos algoritmos. Máquinas como os *clusters* (Anderson et al., 1995), por exemplo, trabalham com uma memória privada para cada processador. Para que um processador conheça dados que não pertençam à sua respectiva memória, são realizadas duas operações básicas, chamadas *send* e *receive* (Eicken et al., 1992). Neste trabalho é utilizado o padrão MPI com a biblioteca lam-mpi, versão 6.5.9-tcp.1.

O objetivo deste trabalho é apresentar uma implementação considerando estratégias eficientes de armazenamento para paralelização do método dos elementos finitos utilizando estratégia de decomposição de domínio com estruturas de blocos orientados da matriz de discretização. O sistema linear de equações proveniente da formulação do método dos elementos finitos é resolvido através do método iterativo não-estacionário do resíduo mínimo generalizado (GMRES) (Saad, 1995). Para realizar as operações necessárias à resolução do sistema linear são consideradas versões paralelas das estratégias EBE, EDE e CSR e através de problemas compressíveis bidimensionais é discutido o desempenho de cada uma delas.

O presente trabalho consta de mais 5 seções além dessa introdução. Na próxima seção é apresentado resumidamente o processo de discretização da equação de convecção e difusão pelo método dos elementos finitos. Um breve comentário sobre as técnicas e estruturas paralelas discutidas em Jimack e Touheed (2000) e Nadeem (2001) e três estratégias de armazenamento de matrizes esparsas são apresentados na seção 3. A seção seguinte apresenta a forma de realizar o produto matriz-vetor e o produto interno paralelos e apresenta um estudo resumido da complexidade dessas operações. A seção 5 apresenta os testes de desempenho realizados em um *cluster* linux comparando as três estratégias de armazenamento abordadas para dois problemas convectivos bidimensionais. Na última seção são apresentadas as principais conclusões obtidas.

2. FORMULAÇÃO DE ELEMENTOS FINITOS PARA A EQUAÇÃO DE CONVECÇÃO E DIFUSÃO

Considere a equação de convecção e difusão na forma conservativa definida em um domínio Ω com contorno Γ :

$$\frac{\partial u}{\partial t} + \boldsymbol{\beta}.\nabla u - \nabla.(\boldsymbol{\kappa}\nabla u) = f. \tag{1}$$

onde u representa a quantidade sendo transportada (temperatura, concentração, por exemplo), $\boldsymbol{\beta}$ é o vetor de velocidade, f é uma função conhecida e $\boldsymbol{\kappa}$ é a matriz de difusividade volumétrica dada por,

$$\boldsymbol{\kappa} = \begin{bmatrix} \kappa_x & 0 \\ 0 & \kappa_y \end{bmatrix}. \tag{2}$$

As condições de contorno essencial e natural adicionadas à equação (1) são:

$$\begin{aligned} u &= g \quad \text{em } \Gamma_g, \\ \boldsymbol{n}.\boldsymbol{\kappa}\nabla u &= h \quad \text{em } \Gamma_h, \end{aligned} \tag{3}$$

onde g e h são funções prescritas de (x, y), $\boldsymbol{n}$ é o vetor unitário normal ao contorno, Γ_g e Γ_h são subconjuntos de Γ. As condições iniciais são dadas por:

$$u(0) = u_o \quad \text{em } \Omega. \tag{4}$$

Considere uma discretização de elementos finitos sobre o domínio Ω em elementos Ω_e, $e = 1, \ldots, n_{el}$, onde n_{el} é o número de elementos. Seja $\mathcal{S}^h \subset \mathcal{S}$ e $\mathcal{V}^h \subset \mathcal{V}$ subespaços das funções de base e funções de peso, detalhes podem ser vistos em Hughes (1987). A formulação estabilizada de elementos finitos da equação (1) pode ser escrita como a seguir. Encontrar $u^h \in \mathcal{S}^h$ tal que $\forall\, w^h \in \mathcal{V}^h$:

$$\begin{aligned} &\int_\Omega \left(w^h \frac{\partial u^h}{\partial t} + w^h \boldsymbol{\beta}^h.\nabla u^h - \nabla w^h.\boldsymbol{\kappa}\nabla u^h \right) d\Omega \quad + \\ &\qquad \sum_{e=1}^{n_{el}} \int_{\Omega^e} \tau \boldsymbol{\beta}^h.\nabla w^h \left(\frac{\partial u^h}{\partial t} + \boldsymbol{\beta}^h.\nabla u^h \right) d\Omega \quad = \\ &\qquad\qquad \int_\Omega w^h f d\Omega + \sum_{e=1}^{n_{el}} \int_{\Omega^e} \tau \boldsymbol{\beta}^h.\nabla w^h f d\Omega, \end{aligned} \tag{5}$$

Na equação (5), τ é o parâmetro de estabilização da formulação SUPG dado por $\tau = \dfrac{\alpha}{2\|\boldsymbol{\beta}\|}$, onde α o parâmetro de malha (Tezduyar et al., 1992). Seja a aproximação de elementos finitos padrão (Hughes, 1987) dada da seguinte forma:

$$u^h(x, y) \cong \sum_{i=1}^{nnodes} N_i(x, y) u_i \,, \tag{6}$$

onde $nnodes$ é o número de nós, N_i é uma função de forma correspondente ao nó i e u_i são os valores nodais de u. Então, substituindo (6) em (5) chega-se a um sistema de equações diferenciais,

$$Ma + Kv = F, \tag{7}$$

onde $v = \{u_1, u_2, \ldots, u_{nnodes}\}^T$ é o vetor de valores nodais de u, a é um vetor contendo as derivadas de u, M é chamada de matriz de massa, K é chamada a matriz de rigidez e F é chamado o vetor de cargas nodais. Para resolver o sistema (7) é utilizado o algoritmo preditor-multicorretor descrito em Hughes e Tezduyar (1984). Neste algoritmo, em cada multicorreção, é necessário resolver o sistema linear não-simétrico:

$$M^* \Delta a = R \,, \tag{8}$$

onde $M^* = M + \alpha \Delta t K$ é uma matriz esparsa da ordem do número de nós incógnitas, $R = F - [Ma^* + Kv^*]$ é o vetor de resíduos em função dos valores iniciais da multicorreção de v e a, ou seja, v^*, a^*, e Δa é a correção dos valores nodais de a para a próxima iteração. É considerado $\alpha = 0.5$ que representa uma aproximação de segunda ordem no tempo e Δt é o tamanho do passo no tempo. A matriz M^* não varia em cada passo de tempo, mas o mesmo não acontece com o vetor R, pois depende dos valores atualizados v^* e a^*.

Neste trabalho é considerado apenas triângulos lineares. Portanto, a interpolação dentro de um elemento é simplesmente,

$$u^e(x, y) \cong \sum_{i=1}^{3} N_i(x, y) u_i, \tag{9}$$

onde N_1,N_2 e N_3 são as funções de forma convencionais (Hughes, 1987). Portanto, as matrizes M e K podem ser construídas a partir das contribuições dos elementos, sendo conveniente identificar seus termos em nível de cada elemento por:

$$\begin{aligned} M &= \mathop{\mathbf{A}}_{e=1}^{nel} (m^e) \\ m^e &= \int_{\Omega e} N^T N d\Omega^e + \int_{\Omega e} \tau_{SUPG} B^T \boldsymbol{\beta}^h N d\Omega^e \end{aligned} \tag{10}$$

$$\begin{aligned} K &= \mathop{\mathbf{A}}_{e=1}^{nel} (k^e) \\ k^e &= \int_{\Omega e} (N^T (\boldsymbol{\beta}^h)^T B + B^T \boldsymbol{k} B) d\Omega^e + \int_{\Omega e} \tau_{SUPG} B^T (\boldsymbol{\beta}^h)(\boldsymbol{\beta}^h)^T B d\Omega^e \end{aligned} \tag{11}$$

onde $\mathbf{A}$ é um operador de montagem, $N = \{N_1, N_2, N_3\}^T$ é um vetor contendo as funções de forma e B é uma matriz contendo as derivadas parciais de N com respeito às suas coordenadas. O vetor F é armazenado de forma similar.

3. ESTRATÉGIAS PARALELAS DO MÉTODO DOS ELEMENTOS FINITOS

Para resolver o sistema (8) é usado uma técnica especial de paralelização denominada decomposição de domínio utilizando blocos orientados para resolver sistemas lineares resultantes de formulações de elementos finitos, descrita em Saad (1995); Jimack e Touheed (2000); Nadeem (2001). A partir dessa estratégia é possível montar as contribuições da matriz M^* e do vetor R de (8) independentemente e concorrentemente em cada processador. As partições resultantes possuem três tipos de nós, denominados por *IntNodes*, *IBNodes* e nós de valor prescrito. Os nós *IntNodes* são nós incógnitas que não pertencem a fronteira de particionamento. Já os nós *IBNodes* são nós incógnitas que pertencem a mais de uma partição simultaneamente, ou seja, pertencem à fronteira de particionamento. Como conseqüência do particionamento da malha,

as incógnitas do vetor de solução v ficam distribuídas ao longo dos p subdomínios tais que parte do vetor de solução, por exemplo $\underline{u}_i$, pertence unicamente ao processador i e $\underline{u}_{s(i)}$ pertence ao processador i mas também a outros processadores. Essa mesma distribuição é aplicada ao vetor de resíduos R.

A Fig. 1 representa uma malha com 50 nós e 74 elementos que foi particionada em quatro subdomínios. Os nós I e J, por exemplo, são nós *IntNodes* dos processadores 3 e 4, respectivamente. Já o nó K é um nó *IBNodes* tanto para o processador 1 como para os processadores 3 e 4. Com o objetivo de facilitar a compreensão dos conceitos empregados, será utilizada a partir

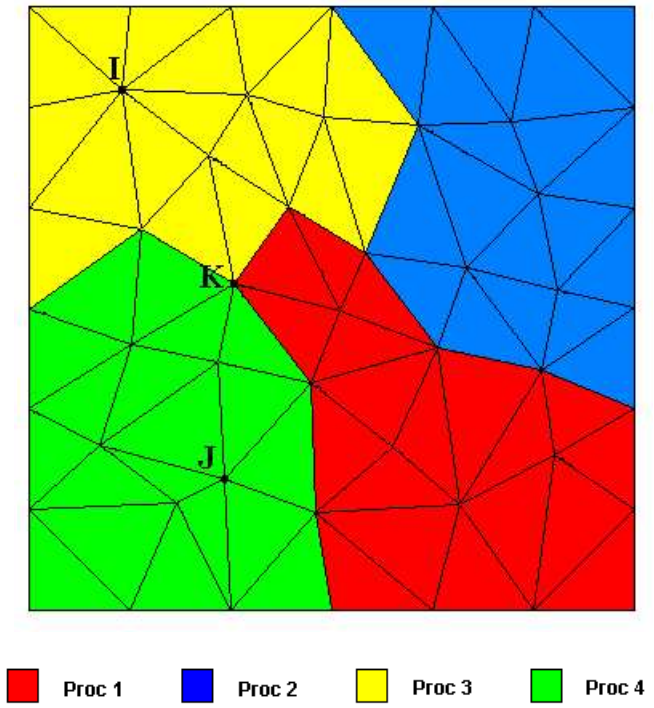


Figura 1: Exemplo de uma malha particionada em 4 subdomí nios

deste ponto a notação convencional para um sistema linear de equações. Portanto, um sistema linear $Ax = b$, resultante da discretização pelo método dos elementos finitos, pode ser descrito da seguinte forma:

$$Ax = \begin{bmatrix} A_1 & & & & B_1 \\ & A_2 & & & B_2 \\ & & \ddots & & \vdots \\ & & & A_p & B_p \\ C_1 & C_2 & \dots & C_p & A_s \end{bmatrix} \begin{bmatrix} \underline{x}_1 \\ \underline{x}_2 \\ \vdots \\ \underline{x}_p \\ \underline{x}_s \end{bmatrix} = \begin{bmatrix} \underline{b}_1 \\ \underline{b}_2 \\ \vdots \\ \underline{b}_p \\ \underline{b}_s \end{bmatrix} = b. \tag{12}$$

Os blocos de matrizes A_i, B_i, C_i e A_s, com $i = 1, ..., p$, onde p é o número de processadores envolvidos na paralelização, armazenam as contribuições dos elementos que formam a matriz A. Os blocos A_i representam as contribuições dos nós do tipo *IntNodes* do processador i nos nós do tipo *IntNodes* do mesmo processador i. Os blocos B_i armazenam as contribuições que os nós do tipo *IntNodes* do processador i têm sobre os nós do tipo *IBNodes*, também do processador i. Similarmente, os blocos C_i representam as contribuições dos nós do tipo *IBNodes* do processador i nos nós *IntNodes* do processador i. O bloco A_s representa uma montagem de vários blocos ao longo dos p processadores , de modo que, $A_s = \bigcup_{i=1}^{p} A_{s(i)}$, onde cada um dos $A_{s(i)}$

armazena as contribuições dos nós do tipo *IBNodes* nos nós do tipo *IBNodes* do processador i. Os blocos de vetores $\underline{x}_i$ e $\underline{b}_i$, com $i = 1, ..., p$, representam as incógnitas relativas aos nós do tipo *IntNodes* e seus respectivos termos independentes para o processador i. Já os blocos de vetores $\underline{x}_s$ e $\underline{b}_s$, assim como o bloco A_s, são formados por montagens dos vários blocos ao longo dos p processadores, ou seja, $\underline{x}_s = \bigcup_{i=1}^{p} \underline{x}_{s(i)}$ e $\underline{b}_s = \bigcup_{i=1}^{p} \underline{b}_{s(i)}$, onde $\underline{x}_{s(i)}$ e $\underline{b}_{s(i)}$ representam as incógnitas e os termos independentes dos nós do tipo *IBNodes* do processador i respectivamente.

Levando em consideração que uma matriz A, obtida a partir da discretização por elementos finitos de uma malha de elementos triangulares lineares, tem ordem $n \times n$, onde n é o número de nós incógnitas, e A possui uma quantidade muita pequena de coeficientes não nulos por linha, conclui-se que os blocos A_i, B_i, C_i e A_s de (12) também apresentam grande esparsidade. A Tab. 1 apresenta a dimensão de cada um desses blocos e o consumo médio de memória utilizada para cada uma das estruturas. O termo n_I representa o número de nós *IntNodes* e n_B representa a quantidade de nós *IBNodes*.

Estruturas Padrão	**Dimensões**	**Consumo Médio**
A_i	$n_I \times n_I$	$\approx n_I^2$
B_i	$n_I \times n_B$	$\approx n_I \cdot n_B$
C_i	$n_B \times n_I$	$\approx n_I \cdot n_B$
$A_{s(i)}$	$n_B \times n_B$	$\approx n_B^2$
$\underline{x}_i$ e $\underline{b}_i$	n_I e n_I	$\approx n_I$ e $\approx n_I$
$\underline{x}_{s(i)}$ e $\underline{b}_{s(i)}$	n_B e n_B	$\approx n_B$ e $\approx n_B$

Tabela 1: Dimensões e consumo de memória das estruturas do sistema (12)

Observações:

1. É importante ressaltar que para utilizar a estrutura (12) obtendo o desempenho desejado, deve-se aplicar alguma estratégia para armazenar seus blocos de forma compacta, evitando guardar os coeficientes nulos, os quais não têm influência alguma sobre os resultados das operações aritméticas envolvidas no processo. As estratégias elemento por elemento (EBE), aresta por aresta (EDE) e *compressed sparse row* (CSR) que serão apresentadas nos próximos itens, demonstraram ser eficazes nesse tipo de armazenamento.

2. O sistema (8) deve ser resolvido em cada multicorreção do algoritmo preditor-multicorretor. Como a Matriz M^* não varia ao longo do tempo, mas o vetor de termos independentes R varia, seria necessário armazenar as matrizes M^*, M e K, respectivamente, definidas em (8), (10) e (11). Porém, com o objetivo de otimizar a implementação, será considerado somente o armazenamento das matrizes M e K, sendo que as mesmas serão utilizadas para efetuar as operações sobre a matriz M^* e o vetor R.

3. Nos itens a seguir será mostrado como é construída uma matriz com as características da matriz K quando forem considerados armazenamentos dos tipos EBE, EDE e CSR. Considerações similares podem ser feitas para a matriz M.

3.1 Estratégia elemento por elemento - EBE

Na estratégia elemento por elemento tradicional, os coeficientes da matriz global A são armazenados em cada elemento. Considerando elementos triangulares lineares, tem-se uma matriz de ordem 3 para cada elemento, ou seja, 9 coeficientes. Assim a matriz A, com dimensões

$n \times n$, onde n é o número de nós incógnitas, é armazenada de forma mais compacta, ou seja, passaria a contar com ebe linhas e 9 colunas, onde ebe é o número de elementos da malha. Entretanto os coeficientes da diagonal principal da matriz do elemento podem ser obtidos através de combinações dos outros coeficientes, não sendo necessário armazená-los (Hughes, 1987). Assim, tem-se 6 colunas por linha ao invés de 9.

Na estratégia EBE paralela os blocos de matrizes A_i, B_i, C_i e $A_{s(i)}$, cujas dimensões são descritas na Tab. 1, são redimensionados visando reduzir o número de coeficientes nulos armazenados. Esta estratégia relaciona os novos blocos com quatro tipos de elementos distintos: elementos ebe_{A_i}, elementos ebe_{B_i}, elementos ebe_{C_i} e elementos ebe_{A_s}. Os elementos ebe_{A_i} são aqueles cujos nós armazenam suas contribuições em A_i, da mesma forma que os elementos ebe_{B_i}, ebe_{C_i} e ebe_{A_s} são aqueles cujos nós armazenam suas contribuições nos blocos B_i, C_i e $A_{s_{(i)}}$, respectivamente. As matrizes dos elementos ebe_{B_i} e ebe_{C_i} não possuem contribuições na diagonal principal, sendo necessário apenas armazenar os coeficientes de fora da diagonal. A Tab. 2 apresenta as dimensões das estruturas descritas em (12), reescritas elemento por elemento, bem como o consumo médio de memória utilizada. Conforme Lima (2004), cada nó

Estruturas EBE	**Dimensões**	**Consumo Médio**
A_i	$6\ ebe_{A_i}$	$\approx 12n_I$
B_i	$6\ ebe_{B_i}$	$\approx 12n_B$
C_i	$6\ ebe_{C_i}$	$\approx 12n_B$
A_s	$6\ ebe_{A_s}$	$\approx 12n_B$

Tabela 2: Dimensões e consumo de memória utilizada na estratégia EBE

da malha, indenpedente de ser uma malha estruturada ou não, tem em média 6 elementos ao seu redor. Assim, a região em torno de um nó incógnita pode ser dividida em três sub-regiões contendo 2 elementos, onde cada sub-região é associada a um nó incógnita diferente. Também é considerado que em média os elementos do tipo ebe_{A_i} têm dimensão igual a $2n_I$ e os elementos dos tipos ebe_{B_i}, ebe_{C_i} e ebe_{A_s} têm dimensões iguais a $2n_B$, como está confirmado nos dados da Tab. 2. É importante ressaltar que um mesmo elemento pode ser classificado em mais de uma categoria, dependendo dos tipos de nós pelos quais é constituído.

3.2 Estratégia aresta por aresta - EDE

A partir das matrizes dos elementos definidas em (10) e (11) é possível definir um desmembramento dos coeficientes gerando as contribuições de cada aresta dos elementos. Esta estratégia de armazenamento foi introduzida em Martins (1996) para problemas simétricos e estendida para problemas não simétricos em Catabriga et al. (1998); Catabriga (2000). Os coeficientes globais de A são agora armazenados em estruturas de arestas ao invés de elementos. Tem-se em cada aresta uma matriz de ordem 2, com 4 coeficientes. Portanto, a nova matriz em função das arestas tem ede linhas e 4 colunas, onde ede é o número de arestas da malha. Porém, assim como na estratégia elemento por elemento, os coeficientes da diagonal principal da matriz da aresta não precisam ser armazenados, reduzindo o número de colunas das novas estruturas para 2 ao invés de 4 (Catabriga, 2000).

Na estratégia EDE paralela, analogamente ao processo descrito para a estratégia EBE, os blocos de matrizes A_i, B_i, C_i e $A_{s(i)}$ são reestruturados em função das arestas, sendo esses relacionados a quatro tipos distintos de arestas: arestas ede_{A_i}, arestas ede_{B_i}, arestas ede_{C_i} e arestas ede_{A_s}. As arestas ede_{A_i} armazenam as contribuições que os nós *IntNodes* que a compõem armazenam no bloco A_i, e as arestas ede_{B_i}, ede_{C_i} e ede_{A_s} armazenam as contribuições que os nós *IntNodes* e *IBNodes* que as constituem devem armazenar respectivamente nos blocos A_i,

B_i, C_i e $A_{s(i)}$. É importante destacar que as arestas ede_{B_i} e ede_{C_i} estão associadas aos mesmos nós, e cada uma tem a necessidade de armazenar apenas metade das informações, ou seja, uma armazena informações em relação a um nó da aresta e a outra armazena informações em relação ao outro nó da aresta. A Tab. 3 contém as dimensões das estruturas de (12) definida aresta por aresta. Utilizando o mesmo raciocínio descrito na estratégia elemento por elemento,

Estruturas EDE	Dimensões	Consumo Médio
A_i	$2\ ede_{A_i}$	$\approx 6n_I$
B_i	$2\ ede_{B_i}$	$\approx 4n_B$
C_i	$2\ ede_{C_i}$	$\approx 4n_B$
$A_{s(i)}$	$2\ ede_{A_s}$	$\approx 4n_B$

Tabela 3: Dimensões e consumo de memória utilizada na estratégia EDE

conclui-se que ao redor de um nó tem-se em média 6 arestas, mas como uma mesma aresta está associada a dois nós distintos, pode-se considerar que ao redor de um nó tem-se em média 3 arestas distintas. Desse modo, cada nó *IntNodes* tem ao seu redor, em média, 3 arestas ede_{A_i} e cada nó *IBNodes* tem, em média, 2 arestas ede_{B_i}, 2 arestas ede_{C_i} e 1 aresta ede_{A_s}. Assim como na estratégia por elemento, uma mesma aresta pode ser classificada em grupos distintos.

3.3 Estratégia compressed sparse row - CSR

O armazenamento CSR tradicional substitui a matriz global de discretização A por três vetores auxiliares AA, JA e IA. O vetor AA armazena todas as contribuições não nulas da matriz global A. O vetor JA, por sua vez, armazena a coluna correspondente que cada coeficiente não nulo ocuparia em A. Já o vetor IA mapeia em AA o primeiro elemento não nulo de cada linha de A. A versão paralela pode ser derivada da tradicional, com algumas modificações. A Tab. 4 apresenta as estruturas CSR que devem substituir as estruturas padrão de (12), bem como suas respectivas dimensões. Como já discutido nos armazenamentos anteriores, ao redor de um nó

Padrão	Estruturas CSR	Consumo Médio
A_i	(AA_i,JA_i,IA_i)	$\approx (7n_I, 7n_I, n_I)$
B_i	(BB_i,JB_i,IB_i)	$\approx (2n_B, 2n_B, n_I)$
C_i	(CC_i,JC_i,IC_i)	$\approx (2n_B, 2n_B, n_B)$
$A_{s(i)}$	$(AA_{s(i)},JA_{s(i)},IA_{s(i)})$	$\approx (3n_B, 3n_B, n_B)$

Tabela 4: Dimensões e consumo de memória utilizada na estratégia CSR

tem-se em média 6 elementos triangulares, e portanto cada linha da matriz A tem em média 7 coeficientes não nulos, representando as contribuições do nó nele mesmo e as contribuições do nó nos outros 6 nós ao seu redor. Portanto, cada nó *IntNodes* está relacionado com outros 6 nós *IntNodes*, em média, o que implica que os coeficientes de AA_i são da ordem de $7\ n_I$. Cada nó *IBNodes* está relacionado com outros 2 nós *IBNodes* e 3 outros nós *IntNodes*, em média. Como conseqüência, as estruturas BB_i e CC_i têm dimensões da ordem de $2\ n_B$ e AA_s tem dimensões da ordem de $3\ n_B$. Também vale lembrar que as estruturas B_i e C_i da formulação global (12) podem conter linhas inteiramente nulas, o que inviabilizaria essa paralelização, uma vez que IB_i e IC_i não conteriam informações precisas sobre o armazenamento CSR de BB_i e CC_i. Sendo assim, mais duas estruturas auxiliares necessitam ser criadas: os vetores $AuxIB_i$ e $AuxIC_i$ que gerenciam as posições dos vetores IB_i e IC_i referentes às possíveis linhas inteiramente nulas de B_i e C_i.

4. SOLUÇÃO DO SISTEMA LINEAR

No método iterativo não-estacionário GMRES as principais operações envolvidas são o produto matriz-vetor e o produto interno entre dois vetores. As estratégias de armazenamento descritas na seção anterior serão usadas para reduzir os esforços computacionais de acessos aos coeficientes armazenados e da memória necessária para realizar essas operações. Na versão paralela dos métodos iterativos não-estacionários, após a operação produto matriz-vetor deve ser feita uma atualização na parte dos vetores que contém valores relacionados com os nós *IBNodes*. Para isso, é considerada uma função, denominada *Update* (Jimack e Touheed, 2000), que utiliza as primitivas *send* e *receive* do padrão MPI. Qualquer que seja a estratégia de armazenamento considerada, a função de atualização da comunicação entre os processadores é absolutamente a mesma, uma vez que ela será aplicada a parcela referente aos *IBNodes* do vetor resultante do produto matriz-vetor. Nas próximas subseções são discutidos o produto matriz-vetor e o produto interno adaptados às estruturas paralelas de (12) e às estratégias de armazenamento da seção anterior.

4.1 Produto matriz-vetor

O produto matriz-vetor é a principal operação da resolução de sistemas lineares através de métodos iterativos não-estacionários, e portanto está relacionado diretamente com o desempenho do processo como um todo. Levando-se em consideração os blocos definidos em (12) o produto matriz-vetor paralelo escrito na forma:

$$Au = \begin{bmatrix} A_1 & & & & B_1 \\ & A_2 & & & B_2 \\ & & \ddots & & \vdots \\ & & & A_p & B_p \\ C_1 & C_2 & \dots & C_p & A_s \end{bmatrix} \begin{bmatrix} \underline{u}_1 \\ \underline{u}_2 \\ \vdots \\ \underline{u}_p \\ \underline{u}_s \end{bmatrix} = \begin{bmatrix} \underline{v}_1 \\ \underline{v}_2 \\ \vdots \\ \underline{v}_p \\ \underline{v}_s \end{bmatrix} = v, \tag{13}$$

pode ser calculado como sendo $\underline{v}_i = A_i\underline{u}_i + B_i\underline{u}_{s_{(i)}}$ no processador i, para $i = 1, ..., p$ e $\underline{v}_{s_{(i)}} = C_i\underline{u}_i + A_{s_{(i)}}\underline{u}_{s_{(i)}}$, nas fronteiras de comunicação. O produto matriz-vetor assim definido pode ser efetuado levando-se em consideração as três estratégia de armazenamento propostas. Utilizando os armazenamentos EBE e EDE a única diferença são as dimensões dos blocos de (12), que podem variar de acordo com a estratégia adotada (veja as Tab. 1, 2 e 3). Com relação ao armazenamento CSR, os blocos de (12) são substituídos pelas estruturas correspondentes da Tab. 4 e as operações são realizadas de acordo com a forma CSR tradicional descrita em Saad (1995).

4.2 Produto interno

O produto interno utiliza todos os componentes de dois vetores para computar um simples ponto flutuante que deverá ser conhecido por todos os processadores. Essa é a operação que consome maior tempo na troca de mensagens entre os processadores, uma vez que ela exige uma comunicação ao longo de todos os processadores. Para isso, é utilizada uma função MPI, denominada *MPI_Allreduce* (Forum, 1994). Para elaborar essa comunicação global para o produto interno distribuído, são considerados dois vetores $\underline{u}$ e $\underline{v}$ dispostos de forma distribuída ao

longo de p processadores:

$$\underline{u} = \begin{bmatrix} \underline{u}_1 \\ \underline{u}_2 \\ \vdots \\ \underline{u}_p \\ \underline{u}_s \end{bmatrix} \quad \text{e} \quad \underline{v} = \begin{bmatrix} \underline{v}_1 \\ \underline{v}_2 \\ \vdots \\ \underline{v}_p \\ \underline{v}_s \end{bmatrix}, \tag{14}$$

onde $\underline{u}_s = \bigcup_{i=1}^{p} \underline{u}_{s_{(i)}}$ e $\underline{v}_s = \bigcup_{i=1}^{p} \underline{v}_{s_{(i)}}$. Os vetores $\underline{u}_i$ e $\underline{v}_i$ pertencem ao processador i, que também armazena $\underline{u}_{s_{(i)}}$ e $\underline{v}_{s_{(i)}}$. Finalmente, o produto interno pode ser expresso por $\underline{u} \cdot \underline{v} = \sum_{i=1}^{p} (\underline{u}_i \cdot \underline{v}_i + \underline{u}_{s_{(i)}} \cdot \underline{v}_{s_{(i)}})$. Independente da estratégia de armazenamento adotada, essa definição do produto interno é precisamente a mesma.

4.3 Número de operações e consumo de memoria do produto matriz-vetor e produto interno

O produto matriz-vetor é a principal operação de todo o processo, sendo que nele é utilizado a grande maioria dos recursos de memória e onde são realizados quase todos os cálculos. Consequentemente, analisando o número de operações e o consumo médio de memória do produto matriz-vetor, tem-se uma idéia da eficiência de toda implementação.

Em relação ao consumo de memória pode-se verificar nas Tab. 2, 3 e 4 que a estratégia EDE tem o menor consumo, seguida da estratégia CSR. Em relação ao número de médio de operações realizadas, pode-se encontrar na Tab. 5 uma comparação entre as 3 estratégias. O produto matriz-vetor é composto por operações de soma, e multiplicação de pontos flutuantes e acessos à memória. As operações de trocas de mensagens entre os processadores são realizadas apenas após o produto matriz-vetor e independente da estratégia de armazenamento adotada essa operação é a mesma. Entretanto, é considerado apenas o número de operações médias das multiplicações de pontos flutuantes. Na Tab. 5 observa-se que o produto matriz-vetor da versão CSR realiza menos operações que o EBE e EDE. Se a ordem da matriz A for muito grande, n_I será muito maior que n_B, ou seja, considerando que o número de nós *IBNodes* cresce linearmente, enquanto o número de nós *IntNodes* cresce quadraticamente, o tempo das comunicações torna-se propocionalmente menor em relação ao tempo de processamento. Assim as versões paralelas tendem a ser p vezes mais rápidas que as versões sequenciais. Considerações análogas podem ser feitas para o produto interno. As operações de comunicação entre os processadores

Operação	Paralelo	Sequencial
Matriz-vetor CSR	$\frac{1}{p}\left[7n_I + 7n_B\right]$	$7n_I$
Matriz-vetor EBE	$\frac{1}{p}\left[18n_I + 42n_B\right]$	$18n_I$
Matriz-vetor EDE	$\frac{1}{p}\left[12n_I + 12n_B\right]$	$12n_I$
Produto Interno	$\frac{1}{p}\left[n_I + 2n_B\right]$	n_I

Tabela 5: Número médio de operações do produto matriz-vetor e produto interno

ocorrem apenas nas versões paralelas, e dependem apenas das parcelas que contêm termos em função dos *IBNodes* e o tempo de comunicação é o mesmo para cada uma das estratégias de armazenamento utilizadas, uma vez que a comunicação só é realizada após ter sido executado o produto matriz-vetor e o produto interno.

5. RESULTADOS NUMÉRICOS

Nesta seção são mostrados os resultados obtidos para dois problemas regidos pela equação linear de advecção-difusão bidimensional: advecção em um campo de escoamento rotacional e o problema do cone em rotação. Nos dois exemplos é utilizado o método GMRES com tolerância de 10^{-6} e 10 vetores na base de *Krylov*. No algoritmo de avanço no tempo é considerado uma tolerância de 10^{-3} nas multicorreções para cada passo no tempo e um tempo final igual a 7 segundos. Os códigos foram implementados em linguagem C, utilizando a biblioteca lampi versão 6.5.9-tcp.1. Todos os testes de desempenho foram realizados no *cluster Enterprise* do Laboratório do Computação de Alto Desempenho (LCAD) do Departamento de Informática da UFES. O cluster é composto por 64 nós de processamento que utilizam o sistema operacional Linux Red Hat 7.1 (kernel 2.4-20) e interligados através de dois *switches Fast Ethernet* de 48 portas de 100Mb/s cada. Cada um dos nós de processamento possui memória SDRAM de 256MB e processador ATHLON XP 1800+. Informações adicionais sobre sobre o *Enterprise*, podem ser encontradas na página www.inf.ufes.br/∼lcad.

5.1 Advecção em um campo de escoamento rotacional

Considere um escoamento fortemente advectivo regido pela rotação de um fluido no centro de um domínio quadrado de dimensões $[-0.5, 0.5] \times [-0.5, 0.5]$, conforme Fig. 2, com velocidade e difusividade dadas por:

$$\boldsymbol{\beta} = \begin{bmatrix} -y \\ x \end{bmatrix} \qquad \boldsymbol{\kappa} = \begin{bmatrix} 10^{-6} & 0 \\ 0 & 10^{-6} \end{bmatrix}. \tag{15}$$

Ao longo do contorno do domínio quadrado a solução é nula, e no contorno interno OA a

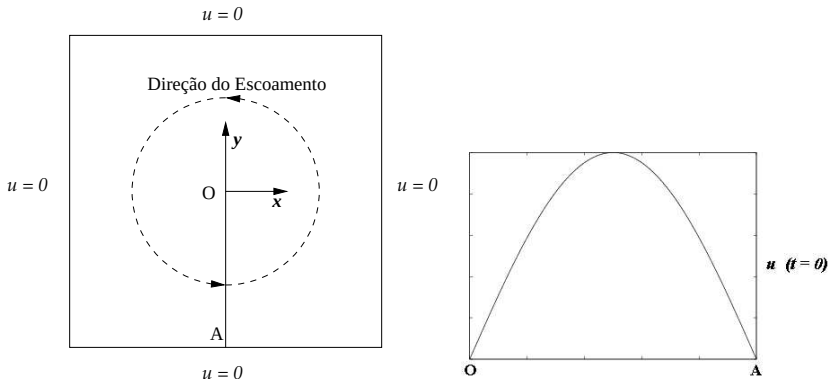


Figura 2: Advecção em um campo de escoamento rotacional - Descrição do problema.

solução possui variação cossenoidal, como mostrado na Fig. 2. A solução exata para esse problema é essencialmente uma advecção pura da condição no contorno OA ao longo das linhas de corrente circulares. A Fig. 3 representa a solução obtida utilizando a estratégia de armazenamento CSR, considerando uma malha estruturada com 40×40 células, contendo 2 elementos triangulares em cada célula, executada em 4 processadores. A solução aproximada corresponde à solução exata esperada. A solução obtida considerando as estratégias EBE e EDE, independente do número de processadores considerado, é essencialmente a mesma, ou seja, nas três estratégias são realizados os mesmos números de iterações lineares e o mesmo número de passos no tempo. Além disso, dentro da tolerância considerada, o resíduo do sistema linear em cada passo de tempo é absolutamente o mesmo. Para análise do desempenho paralelo foi con-

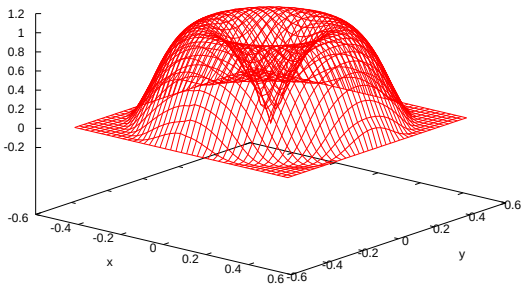


Figura 3: Advecção em um campo de escoamento rotacional - Solução com 4 processadores utilizando a estratégia de armazenamento CSR.

siderada uma malha estruturada com 512×512 células, sendo cada célula dividida em dois elementos triangulares, totalizando 263169 nós e 524288 elementos. A Tab. 6 apresenta os tempos necessários para a convergência considerando as três estruturas de armazenamento e 1, 2, 4, 8, 16, 32, 64 processadores, além da quantidade de objetos processados em cada estratégia, sendo o número de elementos para a estratégia EBE, o número de arestas para a estratégia EDE e o número de coeficientes não nulos para a estratégia CSR. Observa-se que a estratégia CSR é a mais rápida dentre as três. Entretanto, as proporcionalidades referentes ao número médio de operações do produto matriz-vetor descritas na Tab. 5 não foram mantidas. Isto porque, primeiramente, nos exemplos utilizados, as funções relacionadas com o produto matriz-vetor ocuparam cerca de 40% a 60% do tempo total do processamento, o restante do tempo foi gasto nas funções que são comuns as três estratégias, como a produto interno, operações elementares do método GMRES e comunição entre os processadores. Comparando os desempenhos das estratégias EBE e EDE, verifica-se um tempo de processamento muito próximo, o que pode implicar em um melhor aproveitamento dos níves de memória *cash* (Hennessy e Patterson, 2003) pela estratégia EBE. Outro ponto importante é a necessidade de acessos constantes às estruturas auxiliares presentes, por exemplo, na estratégia CSR.

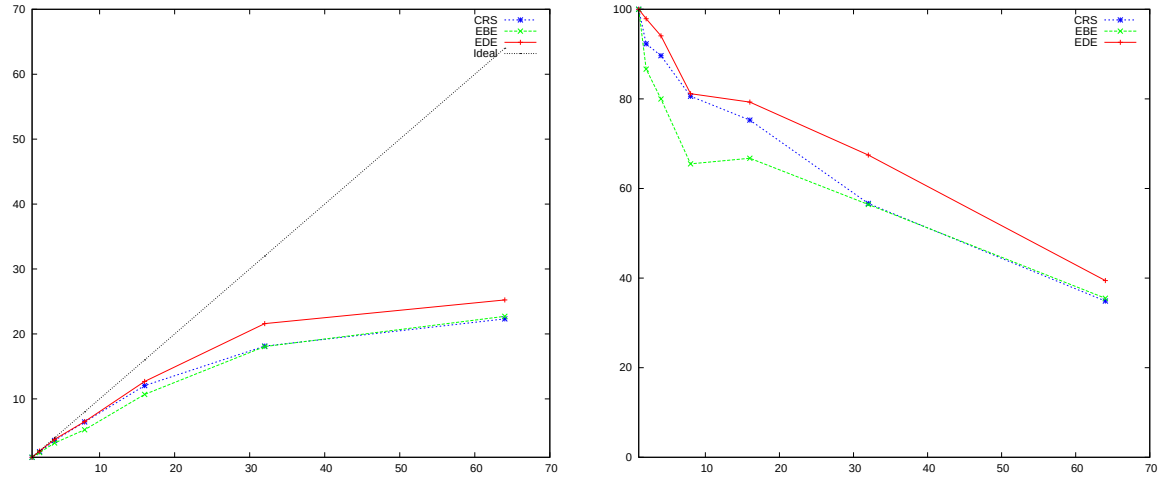
Estratégia	**Número de objetos processados**	**Número de processadores**						
		1	2	4	8	16	32	64
EBE	524285	5020	2897	1569	958	470	278	221
EDE	782592	4922	2514	1308	758	388	228	195
CSR	1820947	3191	1729	890	495	265	176	143

Tabela 6: Advecção em um campo de escoamento rotacional - Tempo de execução em segundos.

As Fig. 4(a) e 4(b) apresentam os gráficos de *speedup* e eficiência do problema de advecção em um campo de escoamento rotacional. Observa-se que a estratégia EDE obteve um melhor *speedup* e melhor eficiência, seguida da estratégia CSR e EBE. Nota-se também que para 64 processadores o *speedup* e a eficiência foram maiores na estratégia EBE que na CSR.

5.2 O problema do cone em rotação

O problema do cone em rotação vem sendo utilizado como um dos problemas padrão para escoamentos transientes fortemente compressíveis. O escoamento é caracterizado pela



(a) Speedup. (b) Efi cência (%).

Figura 4: Advecção em um campo de escoamento rotacional - Desempenho paralelo.

rotação de um fluido ao longo de um cone no interior de um domínio quadrado com dimensões $[-5, 5] \times [-5, 5]$, conforme Fig. 5. A velocidade e difusividade são as mesmas consideradas no exemplo anterior. Ao longo do contorno do domínio quadrado a solução é nula e a condição ini-

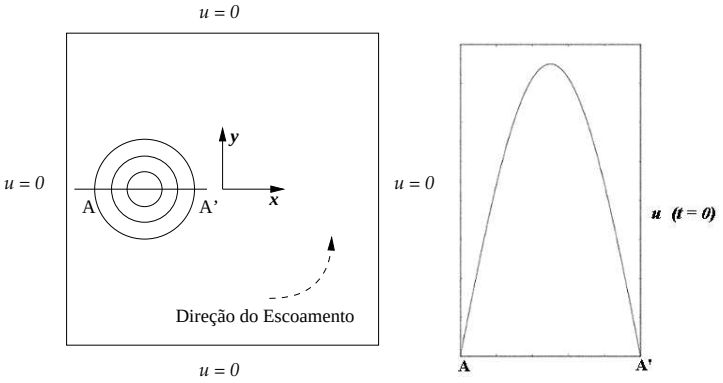
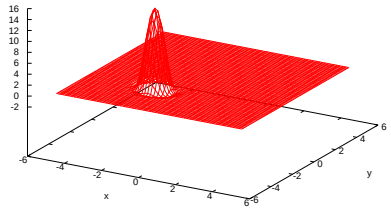


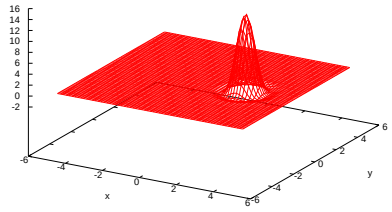
Figura 5: Cone em rotação - Descrição do problema.

cial é dada pela variação cossenoidal, mostrada na Fig. 5. A Fig. 6 representa a solução obtida para os tempos $t = 1$, $t = 3$ e $t = 7$ segundos, utilizando a estratégia de armazenamento CSR e considerando uma malha estruturada com 64×64 células, contendo 2 elementos triangulares em cada célula, executada em 4 processadores. Como no exemplo anterior, a solução obtida considerando as estratégias EBE e EDE, independente do número de processadores considerado, é essencialmente a mesma.

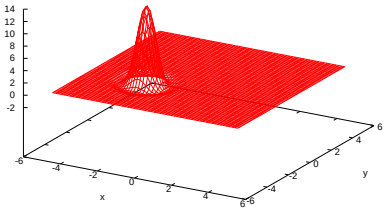
Considerando uma malha não estruturada composta por 259160 nós e 516322 elementos triangulares, a Tab. 7 apresenta os tempos necessários para a convergência considerando as três estruturas de armazanamento e 1, 2, 4, 8, 16, 32, 64 processadores e também a quantidade de objetos processados em cada estratégia: número de elementos na estratégia EBE, número de arestas na estratégia EDE e coeficientes não nulos na CSR. Como no exemplo anterior, a estratégia CSR é a mais rápida dentre as três e a estrutura EDE apresentou o maior *speedup* e maior eficiência como mostrado na Fig. 7. Observa-se na Fig. 7(a) um comportamento



(a) $t = 1$ segundos.



(b) $t = 3$ segundos.



(c) $t = 7$ segundos.

Figura 6: Cone em rotação - Solução com 4 processadores utilizando a estratégia de armazenamento CSR.

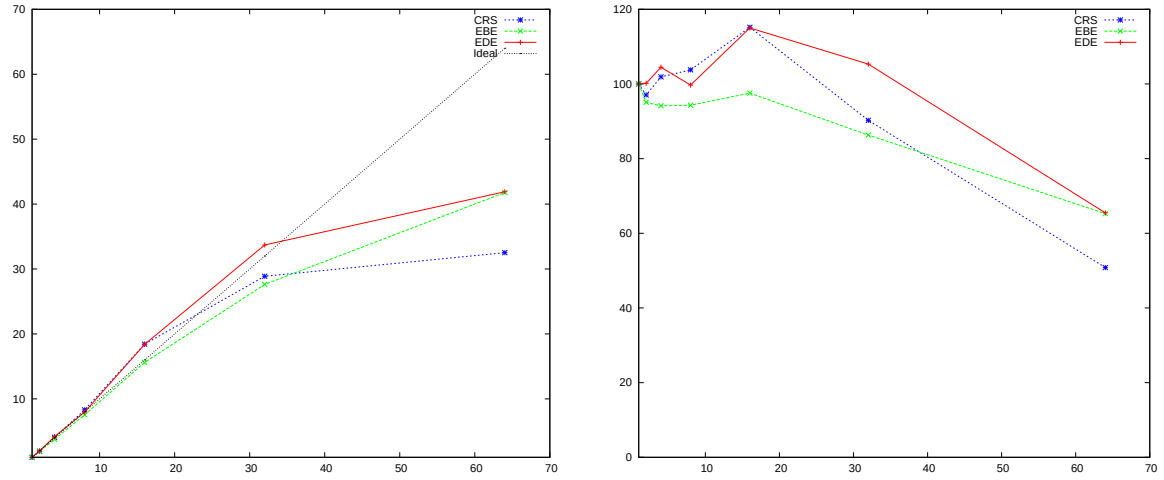
superlinear para este exemplo em alguns casos para as estratégias EDE e CSR, tendo como consequência uma eficiência superior a 100% nesses casos. Esse comportamento acontece devido à hierarquia de memória, possibilitando que as operações sejam realizadas nos níveis L1 e L2 das memórias *cache*, tendo como conseqüência maior rapidez no processamento.

Estratégia	**Número de objetos processados**	**Número de processadores**						
		1	2	4	8	16	32	64
EBE	516322	17044	8964	4525	2259	1092	617	408
EDE	775481	14997	7485	3588	1880	815	445	358
CSR	1810122	10080	5193	2473	1214	547	349	310

Tabela 7: Cone em rotação - Tempo de execução em segundos.

6. CONCLUSÕES

Uma implementação paralela para a equação linear de advecção-difusão bidimensional utilizando uma técnica de decomposição de domínio com estrutura de blocos orientados para a matriz dos coeficientes foi apresentada. Foi realizado um estudo comparativo entre as estratégias de armazenamento EBE, EDE e CSR para as matrizes resultantes da discretização por elementos finitos de dois problemas teste: advecção de um campo de escoamento rotacional e um cone em rotação. Além disso, foi apresentado um estudo sobre o consumo médio de memória utilizada nas nas três estratégias de armazenamento abordadas, assim como o número médio de operações aritméticas do produto matriz-vetor e do produto interno, que aparecem no método



(a) Speedup. (b) Efi cência (%).

Figura 7: Cone em rotação - Desempenho paralelo.

GMRES. Em relação ao tempo de processamento, a estratégia CSR obteve os melhores resultados nos dois exemplos testados. A estratégia EDE apresentou o menor consumo de memória, mas o tempo de processamento manteve-se bem próximo ao da estratégia EBE.

O presente trabalho teve como objetivo principal apresentar três formas alternativas para realizar o armazenamento e conseqüentemente a solução de sistemas lineares oriundos da discretização pelo método dos elementos finitos. Embora os problemas exemplos utilizados sejam de pequeno porte, não justificando uma implementação paralela, eles tiveram um papel importante no desenvolvimento do tema abordado, de forma que os conceitos aplicados podem ser estendidos a problemas mais complexos.

Agradecimentos

O autor Leonardo M. Lima agradeçe à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pela bolsa de mestrado e o autor Bruno Z. Melotti agradeçe ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pela bolsa de Iniciação Científica. Este trabalho também recebeu o apoio parcial da CAPES, dentro do projeto de cooperação internacional CAPES - Universidade do Texas, CAPES/UT Nº 11/04.

Referências

Anderson, T. E., Culler, D. E., & Patterson, D. A., 1995. A case for networks of workstations: NOW. feb, IEEE Micro.

Catabriga, L., 2000. *Soluções implícitas das equações de Euler empregando estruturas de dados por aresta*. Tese de Doutorado, COPPE/UFRJ.

Catabriga, L., Martins, M. D. A., Coutinho, A. L. G. A., & Alves, J. L. D., 1998. Clustered edge-by-edge preconditioners for non-symmetric finite element equations. In *4th World Congress on Computational Mechanics*.

Eicken, T., Culler, D. E., Goldstein, S. C., & Schauser, K. E., 1992. Active messages: A mechanism for integrated communication and computation. In *19th International Symposium on Computer Architecture*, pp. 256–266, Gold Coast, Australia.

Forum, M. P. I., 1994. MPI: A message-passing interface standard. Technical Report UT-CS-94-230.

Hennessy, J. L. & Patterson, D. A., 2003. *Computer Architecture: a Quantitative Approach*. Morgan Kaufmann Publishers, Inc., 3nd edition.

Hughes, T. J. R., 1987. *The finite element method*. Prentice-Hall International.

Hughes, T. J. R. & Tezduyar, T. E., 1984. Finite element methods for first-order hyperbolic systems with particular emphasis on the compressible Euler equations. *Comput. Methods Appl. Mech. and Engrg.*, vol. 45, pp. 217–284.

Jimack, P. K. & Touheed, N., 2000. Developing parallel finite element software using mpi. In Topping, B. & Lammer, L., eds, *High Performance Computing for Computational Mechanics*, pp. 15–38. Saxe-Coburg Publications.

Lima, L. M., 2004. Estratégias de armazenamento para implementações paralelas para o método dos elementos finitos. Dissertação de Mestrado, Universidade Federal do Espírito Santo (UFES).

Martins, M. A. D., 1996. *Solução iterativa em paralelo de sistemas de equações do método dos elementos finitos empregando estruturas de dados por arestas*. Dissertação de Mestrado, COPPE/UFRJ,.

Nadeem, S. A., 2001. *Parallel domain decomposition preconditioning for the adaptive finite element solution of elliptic problems in three dimensions*. PhD thesis, The Univerty of Leeds, Leeds, UK.

Saad, Y., 1995. *Iterative methods for sparse linear systems*. PWS Publishing Company.

Tezduyar, T. E., Lion, J., & Behr, M., 1992. A new strategy for finite element computations involving moving boundaries and interfaces - the dsd/st procedure: I. the concept and the preliminary numerical tests. *Computer Methods in Applied Mecanics and Engineering*, vol. 94(3), pp. 339–351.