Estratégias de Armazenamento para Implementações Paralelas do Método dos Elementos Finitos

Leonardo Muniz de Lima Bruno Zanetti Melotti Lucia Catabriga
Andréa Maria Pedrosa Valli
Laboratório de Computação de Alto Desempenho - LCAD
Universidade Federal do Espírito Santo
Av Fernando Ferrari, s/n Vitória, ES, Brazil
{lmuniz, bzmelotti, luciac, avalli}@inf.ufes.br

Resumo

O presente trabalho estuda o desempenho da paralelização do método dos elementos finitos utilizando estratégia de decomposição de domínio com estruturas de blocos orientados da matriz de discretização resultante e três formatos de armazenamento de matrizes esparsas. O sistema linear de equações proveniente da formulação do método dos elementos finitos é resolvido através do método iterativo não-estacionário GMRES. 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 da biblioteca 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 [5]. A discretização pelo método dos elementos finitos conduz à resolução 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 [10]. Além disso, a principal operação desses métodos, o produto matriz-vetor, pode ser realizado acessando apenas as posições não nulas da matriz dos coeficientes. Entretanto, para que essa vantagem dos métodos não-estacionários possa ser melhor aproveitada, há a necessidade de se definir 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) [5], mas existem outras conhecidas como a aresta por aresta (EDE) [2] [7] e a tradicional estratégia de linhas esparsas comprimidas, do inglês, *Compressed Sparse Row* (CSR) [10].

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* [1], por exemplo, trabalham com uma memória privada para cada processador. Para que um processador conheça dados que não pertençam a sua respectiva memória, são realizadas duas operações básicas chamadas *send* e *receive* [4]. Contudo, para que estas operações sejam realizadas de maneira mais simples, são utilizadas algumas bibliotecas que constroem uma interface de software entre as várias máquinas, diminuindo a complexidade da programação. Existem muitos exemplos desse tipo de biblioteca, dentre os quais podemos citar a *Message Passage Interface* (MPI) [8].

O objetivo deste trabalho é apresentar 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) [11]. Para realizar as operações necessárias à resolução do sistema linear são empregadas versões paralelas das estratégias EBE, EDE e CSR e através de um exemplo bidimensional simples, 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. Na seção 3, é feito um breve comentário sobre as técnicas e estruturas paralelas discutidas em [6] [9] e apresentado três estratégias de armazenamento de matrizes esparsas. A seção seguinte, discute a forma de realizar o produto matriz-vetor e o produto interno paralelos e apresenta um estudo resumido da complexidade dessas operações. Na seção 5, é apresentado os testes de desempenho realizados em um *cluster* linux comparando as três estratégias de armazenamentos abordadas para um problema simples de convecção e difusão bidimensional. 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 :

$$\boldsymbol{\beta}.\nabla c - \nabla.(\boldsymbol{\kappa}\nabla c) = f. \tag{1}$$

onde c 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 dado 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:

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

onde g e h são funções prescritas, $\boldsymbol{n}$ é o vetor unitário normal ao contorno, Γ_g e Γ_h são subconjuntos de Γ. 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 [5]. A formulação variacional aproximada de elementos finitos da equação (1) pode ser escrita como: encontrar $c^h \in \mathcal{S}^h$ tal que $\forall\ w^h \in \mathcal{V}^h$:

$$\int_\Omega \left(w^h \boldsymbol{\beta}^h.\nabla c^h - \nabla w^h.\boldsymbol{\kappa}\nabla c^h \right) d\Omega = \int_\Omega w^h f d\Omega. \tag{4}$$

Seja a aproximação de elementos finitos padrão [5] dada da seguinte forma:

$$c^h(\boldsymbol{x}) \cong \sum_{i=1}^{nnodes} u_i N_i(\boldsymbol{x}), \tag{5}$$

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 c. Então, substituindo (5) em (4) chega-se a um sistema linear de equações,

$$Ku = f, \tag{6}$$

onde $u = \{u_1, u_2, \ldots, u_{nnodes}\}^T$ é o vetor de valores nodais, K é chamada a matriz de rigidez e f é chamado o vetor de cargas. Neste trabalho é considerado apenas triângulos lineares, portanto, a interpolação dentro de um elemento é simplesmente,

$$u^e(\boldsymbol{x}) \cong \sum_{i=1}^{3} u_i N_i(\boldsymbol{x}), \tag{7}$$

onde N_1, N_2 e N_3 são as funções de forma convencionais [5]. A matriz K pode ser construída a partir das contribuições dos elementos, sendo conveniente identificar seus termos em nível de cada elemento por:

$$\begin{aligned} K &= \mathop{\mathbf{A}}_{e=1}^{nel}(k^e) \\ k^e &= \int_{\Omega e} (N^T(\boldsymbol{\beta}^h)^T B + B^T \boldsymbol{\kappa} B) d\Omega^e, \end{aligned} \tag{8}$$

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 (6) 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 [6] [9] [10]. A partir dessa estratégia é possível montar as contribuições da matriz K e do vetor f de (6) 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 a fronteira de particionamento. Como conseqüência do particionamento da malha, as incógnitas do

vetor de solução u 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 cargas f.

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 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:

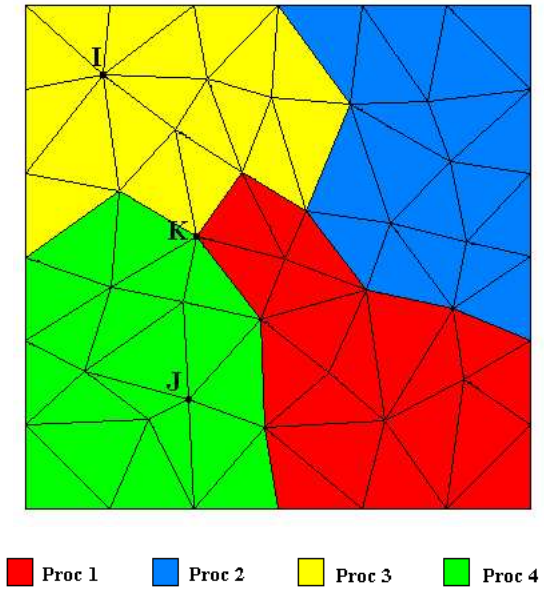

Figura 1. Exemplo de uma malha particionada em 4 subdomínios

$$Ax = \begin{bmatrix} A_1 & & & & B_1 \\ & A_2 & & & B_2 \\ & & \ddots & & \vdots \\ & & & A_p & B_p \\ C_1 & C_2 & \dots & C_p & A_s \end{bmatrix} \begin{bmatrix} \underline{x}_1 \\ \underline{x}_2 \\ \vdots \\ \underline{x}_p \\ \underline{x}_s \end{bmatrix} = \begin{bmatrix} \underline{b}_1 \\ \underline{b}_2 \\ \vdots \\ \underline{b}_p \\ \underline{b}_s \end{bmatrix} = b. \quad (9)$$

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, $A_s = \overset{p}{\underset{i=1}{\mathbf{U}}} A_{s(i)}$, 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, $\underline{x}_s = \overset{p}{\underset{i=1}{\mathbf{U}}} \underline{x}_{s(i)}$ e $\underline{b}_s = \overset{p}{\underset{i=1}{\mathbf{U}}} \underline{b}_{s(i)}$, 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 (9) também apresentam grande esparsidade. A Tab. 1 apresenta a dimensão de cada um desses blocos e o consumo médio de memória utilizada por cada uma das estruturas. O termo n_I representa o número de nós *IntNodes* e n_B representa a quantidade de nós *IBNodes*. Portanto para utilizar a estrutura (9) obtendo o desempenho desejado, deve-se aplicar alguma estratégia para armazenar seus blocos de forma compacta, evitando guardar os coeficientes nulos, as quais não têm influência alguma sobre os resultados das operações aritméticas envolvidas no processo. As estratégias elemento por elemento

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	$\approx (n_I, n_I)$
$\underline{x}_{s(i)}$ e $\underline{b}_{s(i)}$	n_B e n_B	$\approx (n_B, n_B)$

Tabela 1. Dimensões e consumo de memória das estruturas do sistema (9)

(EBE), aresta por aresta (EDE) e *compressed sparse row* (CSR) que serão apresentadas em seguida, demonstraram ser eficazes nesse tipo de armazenamento.

3.1. Estratégia elemento por elemento - EBE

Na estratégia elemento por elemento tradicional, os coeficientes da matriz de discretização 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 [5]. 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 a ser armazenado. 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 (9), reescritas elemento por elemento, bem como o consumo médio de memória utilizada. Analisando as propriedades geométricas de malhas formadas por elementos triangulares e confirmando esses resultados experimentalmente, conclui-se que, em média, cada nó da malha têm 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.

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

3.2. Estratégia aresta por aresta - EDE

A partir das matrizes dos elementos definidas em (8), é possível definir um desmembramento dos coeficientes gerando as contribuições de cada aresta dos elementos. Esta estratégia de armazenamento foi introduzida em [7] para problemas simétricos e estendida para problemas não simétricos em [2] [3]. Os coeficientes da matriz de discretização 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 teria 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 [2].

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 armazenam respectivamente nos blocos A_i, B_i, C_i e $A_{s(i)}$. É importante destacar que as aresta 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 (9) definida aresta por aresta. Utilizando o mesmo raciocínio descrito na estratégia elemento por elemento, 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.

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

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 (9), bem como suas respectivas dimensões. Como já discutido no armazenamento elemento

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)	$\approx (2n_B, 2n_B, n_B)$
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

por elemento, ao redor de um nó 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 (9) 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 as 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 minimizar 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* [6], que utiliza as primitivas *send* e *receive* da biblioteca MPI. Nas próximas subseções são discutidos o produto matriz-vetor e o produto interno adaptados às estruturas paralelas de (9) 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 (9) o produto matriz-vetor paralelo escrito na forma:

$$Au = \begin{bmatrix} A_1 & & & & B_1 \\ & A_2 & & & B_2 \\ & & \ddots & & \vdots \\ & & & A_p & B_p \\ C_1 & C_2 & \dots & C_p & A_s \end{bmatrix} \begin{bmatrix} \underline{u}_1 \\ \underline{u}_2 \\ \vdots \\ \underline{u}_p \\ \underline{u}_s \end{bmatrix} = \begin{bmatrix} \underline{v}_1 \\ \underline{v}_2 \\ \vdots \\ \underline{v}_p \\ \underline{v}_s \end{bmatrix} = v, \tag{10}$$

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 estratégias de armazenamento propostas. Utilizando os armazenamentos EBE e EDE a única diferença são as dimensões dos blocos de (9), que podem variar de acordo com a estratégia adotada (veja as Tab. 1, 2 e 3). Com relação ao armazenamento CSR, os blocos de (9) são substituídos pelas estruturas correspondentes da Tab. 4 e as operações são realizadas de acordo com a forma CSR tradicional descrita em [10].

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 consume maior tempo na troca de mensagens entre os processadores, uma vez que exige uma comunicação ao longo de todos os processadores, para isso, é utilizada uma função da biblioteca MPI, denominada *MPI_Allreduce* [6]. 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:

$$\underline{u} = \begin{bmatrix} \underline{u}_1 \\ \underline{u}_2 \\ \vdots \\ \underline{u}_p \\ \underline{u}_s \end{bmatrix} \quad \text{e} \quad \underline{v} = \begin{bmatrix} \underline{v}_1 \\ \underline{v}_2 \\ \vdots \\ \underline{v}_p \\ \underline{v}_s \end{bmatrix}, \tag{11}$$

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. Complexidade do produto matriz-vetor e produto interno

A complexidade de um algoritmo está relacionada com o consumo de memória e o tempo de processamento. 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. Conseqüentemente, analisando a complexidade média do produto matriz-vetor, tem-se uma idéia do comportamento de toda implentação.

A complexidade do produto matriz-vetor está relacionada diretamente com a estratégia de armazenamento utilizada. 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. A Tab. 5 mostra a complexidade média do produto matriz-vetor e do produto interno para as três estratégias. N_I representa o número total de nós internos da malha e N_B representa o número total de nós de interface. O produto matriz-vetor é composto por operações de soma, subtração, multiplicação e divisão de pontos flutuantes, acessos a memória e principalmente trocas de mensagens entre os processadores. Entretanto, é considerado apenas as complexidades médias das multiplicações/divisões 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, não considerando o tempo das comunicações, as versões paralelas tendem a ser p vezes mais rápidas que as versões seqüenciais. Considerações análogas podem ser feitas para o produto interno. As operações de comunicação entre os processadores ocorrerem apenas nas versões paralelas, e dependem apenas das parcelas que contém termos em função dos *IBNodes*.

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. Complexidade média do produto matriz-vetor e produto interno

5. Testes de desempenho

Todos os testes de desempenho foram realizados no *cluster* do Laboratório do Computação de Alto Desempenho (LCAD) do Departamento de Informática da UFES. As informações sobre a configuração das 64 máquinas, bem como informações adicionais sobre o *cluster* podem ser encontradas na página www.inf.ufes.br/∼lcad.

Para testar o desempenho das estruturas implementadas, é considerado um problema de convecção e difusão com solução conhecida em um domínio quadrado com dimensões $(0,1) \times (0,1)$. As componentes do vetor $\boldsymbol{\beta}$ foram dadas por $\beta_x = 1.0$ e $\beta_y = 1.0$, os coeficientes de condutividade em cada direção foram $\kappa_x = 1.0$ e $\kappa_y = 1.0$, no contorno Γ, $c(x,y) = 0.0$ e f é tal que $c(x,y) = 100xy(x-1)(y-1)$ para $(x,y) \in (0,1)\times(0,1)$. Uma malha composta por 160,801 nós e 320,000 elementos triangulares é considerada, sendo utilizado o método GMRES com tolerância de 10^{-4} e 10 vetores na base de *Krylov*.

A Fig. 2 mostra a solução aproximada obtida a partir de 4 processadores utilizando a estratégia EBE e representa a solução exata esperada. As Tab. 6, 7 e 8 apresentam o tempo de processamento, o número de iterações GMRES e o resíduo obtido na solução do sistema linear para 1, 2, 4, 8, 16, 32 e 64 processadores para as 3 estratégias de armazenamento. Conforme as expectativas, a estratégia CSR foi mais rápida que as demais (veja as Tab. 6, 7 e 8). Entretanto

as proporcionalidades do produto matriz-vetor descritas na Tab. 5 não foram mantidas. A necessidade de acessos a estruturas auxiliares, como no caso da estratégia CSR, é um dos fatores que contribuem para que essa proporcionalidade não seja mantida.

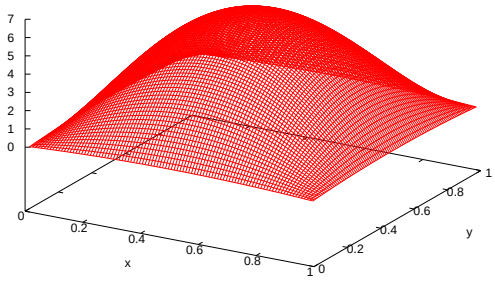

Figura 2. Gráfico da solução utilizando a estratégia EBE com 4 processadores

N. Proc.	Tempo (s)	Iterações	Resíduo
01	4846	2837	1.785505×10^{-5}
02	2846	2837	1.785505×10^{-5}
04	1503	2837	1.785505×10^{-5}
08	834	2837	1.785505×10^{-5}
16	483	2837	1.785505×10^{-5}
32	351	2837	1.785505×10^{-5}
64	319	2837	1.785505×10^{-5}

Tabela 6. Tempo de CPU, número de iterações e resíduo do método GMRES utilizando estratégia EBE

Outro ponto importante a ser considerado diz respeito a precisão do método. As Tab. 6, 7 e 8 mostram que independentes do número de processadores, as estratégias de armazenamento aplicadas ao método GMRES necessitaram do mesmo número de iterações para convergir e dentro da precisão estabelecida, obtiveram o mesmo resíduo. Os valores obtidos para a solução do problema foram rigorosamente os mesmos em cada caso. Vale lembrar que o estudo de prova e correção de algoritmos paralelos é ainda recente e uma das maneiras mais simples para validar um algoritmo

N. Proc.	Tempo (s)	Iterações	Resíduo
01	4634	2837	1.785505×10^{-5}
02	2430	2837	1.785505×10^{-5}
04	1241	2837	1.785505×10^{-5}
08	718	2837	1.785505×10^{-5}
16	398	2837	1.785505×10^{-5}
32	284	2837	1.785505×10^{-5}
64	281	2837	1.785505×10^{-5}

Tabela 7. Tempo de CPU, número de iterações e resíduo do método GMRES utilizando estratégia EDE

N. Proc.	Tempo (s)	Iterações	Resíduo
01	3848	2837	1.785505×10^{-5}
02	2033	2837	1.785505×10^{-5}
04	1073	2837	1.785505×10^{-5}
08	587	2837	1.785505×10^{-5}
16	354	2837	1.785505×10^{-5}
32	282	2837	1.785505×10^{-5}
64	263	2837	1.785505×10^{-5}

Tabela 8. Tempo de CPU, número de iterações e resíduo do método GMRES utilizando estratégia CSR

paralelo é obter rigorosamente os mesmos resultados dos algoritmos sequenciais equivalentes.

As Fig. 3 e 4 apresentam os gráficos de *speedup* e eficiência. Observa-se que apesar da estratégia CSR ser a mais rápida, foi menos eficiente que a EDE para a maioria dos casos. Infelizmente os testes com 64 processadores comprometeram os resultados, uma vez que o tempo de processamento foi quase o mesmo que o tempo para 32 processadores, ou seja, a eficiência, nesse caso, manifestou uma queda considerável.

6. Conclusão

Foi realizado um estudo comparativo entre as estratégias de armazenamento EBE, EDE e CSR aplicadas a equação de convecção e difusão discretizada pelo método dos elementos finitos, sendo considerada uma técnica de decomposição de domínio com estrutura de blocos orientados para a matriz dos coeficientes. A estratégia CSR foi um pouco mais rápida que as demais. Contudo, a necessidade de diversas estruturas auxiliares para realização do produto matriz-vetor dificulta

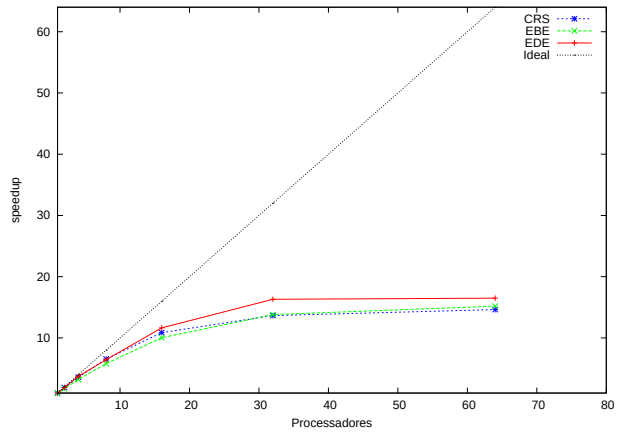

Figura 3. Gráfico de Speedup

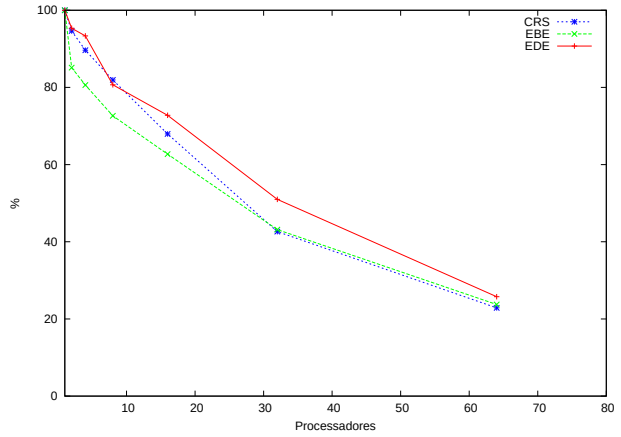

Figura 4. Gráfico de Eficiência

a programação. A estratégia EDE em relação ao tempo de processamento manteve-se intermediária, mas no consumo de memória foi a mais econômica. Já a estratégia EBE, mesmo perdendo em tempo de processamento e consumo de memória, tem a vantagem de ser a estratégia de mais fácil implementação.

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

[1] Thomas E. Anderson, David E. Culler, and David A. Patterson. A case for networks of workstations: NOW. IEEE Micro, feb 1995.

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

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

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

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

[6] P. K. Jimack and N. Touheed. Developing parallel finite element software using mpi. In B.H.V. Topping and L. Lammer, editors, *High Performance Computing for Computational Mechanics*, pages 15–38. Saxe-Coburg Publications, 2000.

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

[8] Message Passing Interface Forum. MPI: A Message Passing Interface. In *Proceedings of Supercomputing '93*, pages 878–883. IEEE Computer Society Press, 1993.

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

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

[11] F. Shakib, T. J. R. Hughes, and Z. Johan. A multi-element group preconditioned gmres algorithm for non-symmetric systems arising in finite element analysis. *Computer Methods in Applied Mechanics and Engineering*, 65:415–456, 1989.