ESTRATÉGIAS DE ARMAZENAMENTO PARALELAS APLICADAS AO MÉTODOS DOS ELEMENTOS FINITOS

Leonardo Muniz de Lima

DISSERTAÇÃO SUBMETIDA AO PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA DO CENTRO TECNOLÓGICO DA UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO, COMO REQUISITO PARCIAL PARA A OBTENÇÃO DO GRAU DE MESTRE EM INFORMÁTICA.

Aprovada por:

Profa. Lucia Catabriga, D.Sc.

Profa. Andrea Mª P. Valli, D.Sc.

Prof. Alvaro L. G. de A. Coutinho, D.Sc.

Prof. Neyval Costa Reis Jr., Ph.D.

VITÓRIA , ES – BRASIL
DEZEMBRO DE 2004

Dados Internacionais de Catalogação-na-publicação (CIP)

(Biblioteca Central da Universidade Fedreal do Espírito Santo, ES, Brasil)

Lima, Leonardo Muniz de, 1978 -

L628t Estratégias de armazenamento paralelas aplicadas ao método dos elementos finitos / Leonardo Muniz de Lima. - 2004.

98 f. :il.

Orientadora: Lucia Catabriga.

Co-orientadora: Andrea Maria Pedrosa Valli.

Dissertação (mestrado) - Universidade Federal do Espírito Santo, Centro Tecnológico

1. Método dos elementos finitos. 2. Programação paralela (Computação) I. Catabriga, Lucia. II. Valli, Andrea Maria Pedrosa. III. Universidade Federal Espírito Santo. Centro Tecnológico. IV. Título.

CDU: 004

Aos meus pais Antônio e Ivanete

Agradecimentos

Agradeço primeiramente a Deus pela vida, saúde e pelo entendimento concedido para completar esse trabalho.

Agradeço a minha orientadora Lucia Catabriga pela oportunidade e pela confiança em mim depositada.

Agradeço a minha co-orientadora Andrea M. P. Valli pela paciência e empenho na revisão desse trabalho.

Agradeço muito a minha noiva Suellen pelo apoio e pela compreensão nos momentos de stress.

Agradeço aos meus pais e minhas irmãs, Érica e Ellen, que sempre me apoiaram nessa árdua tarefa.

Agradeço a CAPES pela bolsa de estudos concedida, sem a qual não seria possível a realização desse trabalho.

Agredeço aos professores Neyval, Alberto e Raul pelas dúvidas esclarecidas.

Agradeço ao professor ao professor Elias pelas dicas e sugestões.

Agradeço ao meu amigo Sérgio Nery Simões que me ajudou muito no início do mestrado.

Agradeço ao meu amigo Bruno Zanetti Melotti pela parceria na implementação de algumas parte do código.

Agradeço aos meus amigos do LCAD, em especial, ao Filipe Thomaz Pedroni, ao Stiven Schwanz Dias e ao Dijalma Fardin Jr.

Enfim, agradeço a todos que forma direta ou indireta contribuíram para que esse trabalho fosse realizado.

Resumo da Dissertação apresentada ao PPGI/UFES como parte dos requisitos necessários para a obtenção do grau de Mestre

ESTRATÉGIAS DE ARMAZENAMENTO PARALELAS APLICADAS AO MÉTODO DOS ELEMENTOS FINITOS

Leonardo Muniz de Lima

Dezembro/2004

Orientadora: Lucia Catabriga
Co-orientadora: Andrea M. P. Valli
Programa: Informática

Técnicas especiais de armazenamento são importantes ferramentas para melhorar a eficiência computacional de um código de elementos finitos que utiliza a computação de alto desempenho. No presente trabalho, foi realizado um estudo comparativo entre as estratégias de armazenamento elemento-por-elemento (EDE), aresta-por-aresta (EDE) e *compressed sparse row* (CSR), considerando a técnica de decomposição de domínios com estruturas de blocos orientados e o método GMRES para a resolução dos sistemas linerares. Foram resolvidos quatro problemas exemplo: advecção em um campo de escoamento rotacional, cone em rotação, advecção e difusão com solução conhecida e uma equação diferencial parcial elíptica de 2ª ordem.

Experimentos numéricos confirmam que, com a técnica de paralelização empregada, foram encontradas as mesmas soluções aproximadas independentemente do número de processadores e das estratégias de armazenamento empregados. Foram medidos o *speedup* e a eficiência para cada caso. A estratégia CSR foi a mais rápida em todos os testes realizados. No entanto, a estratégia EDE pode ser classificada como aquela que melhor adequou-se ao processo de paralelização, por apresentar maiores valores de *speedup* e eficiência na maior parte dos casos.

Abstract of dissertation presented to PPGI/UFES as a partial fulfillment of the requirements for the degree of Master (MS)

PARALLEL STORAGE STRATEGIES FOR FINITE ELEMENT METHOD

Leonardo Muniz de Lima

December/2004

Advisor: Lucia Catabriga

Co-advisor: Andrea M. P. Valli

Department: Computing Science

Special storage techniques are important to improve computational efficiency of finite element codes that use high performance computing. In this work, we perform comparative studies among three storage strategies: element-by-element (EDE), edge-by-edge (EDE) and compressed sparse row (CSR). We consider a domain decomposition technique with a block-arrowhead structure of the finite element matrix and the GMRES method for the solution of the linear systems. We solve four example problems to validate and compare the implemented structures: advection in a field of rotational flow, rotational cone , advection and diffusion with known solution and an elliptical partial differential equation of second-order.

We confirm through the numerical experiments that, with the parallel techniques, same approximated solutions can be obtained independently of the number of processors and the storage strategies employed. Speedup and efficiency have been measured in each case. The CSR strategy is the fastest in all examples. However, in most of cases, EDE strategy presents highest values of speedup and efficiency.

"Há homens que lutam um dia e são bons.
Há outros que lutam um ano e são melhores.
Há os que lutam muitos anos e são muito bons.
Porém, há os que lutam toda a vida.
Esses são os imprescindíveis."
Bertolt Brecht.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Com a evolução da metodologia de elementos finitos, é possível a construção de soluções aproximadas para problemas cada vez mais complexos nas diversas áreas da engenharia. Como a solução desses problemas demandam grande tempo de processamento e alto consumo de memoria, faz-se necessário o uso da computação paralela. Uma eficiente implementação de um código de elementos finitos em paralelo conduz principalmente a redução no tempo de processamento. No entanto, esta redução muitas vezes está intimamente relacionada com o consumo de memória, uma vez que os dados a serem processados em uma única máquina com uma única memória, encontram-se agora distribuídos pelas memórias dos diversos processadores envolvidos na paralelização. Sendo assim, o principal objetivo desse trabalho é comparar a eficiência de três estratégias especiais de armazenamento utilizadas em elementos finitos e aplicadas a um problema modelo, considerando a estratégia de decomposição de domínios com estruturas de blocos orientados da matriz de discretização.

O método dos elementos finitos é uma técnica geral para construção de soluções aproximadas para problemas de valor de contorno e valor inicial que envolve divisão do domínio de solução em um número finito de subdomínios, denominados elementos finitos [18]. A discretização pelo método dos elementos finitos resulta na 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 são mais indicados nesse caso, pois apresentam facilidades no armazenamento das matrizes esparsas [35]. 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 armazenamento especiais das matrizes envolvidas.

Existem várias estratégias de armazenamento aplicadas ao método dos elementos finitos e utilizadas juntamente com métodos iterativos de solução de sistemas lineares. Nesse trabalho foram implementadas as estratégias elemento-por-elemento (EBE) [18], aresta-por-aresta (EDE) [6][28] e a estratégia Compressed Sparse Row (CSR) [35].A mais popular é o armazenamento elemento-por-elemento (EBE). Nessa estratégia a matriz de discretização por elementos finitos não é montada, mas, suas contribuições são armazenadas em estruturas de dados que representam cada elemento. A estratégia EBE apresenta uma conectividade elemento-nó que é utilizada toda vez que um produto matriz-vetor é requerido. A estratégia EDE é semelhante a estratégia EBE e, nesse caso, a matriz de discretização por elementos finitos também não é montada, mas suas contribuições são guardadas em estruturas de arestas dos elementos. Essa estratégia apresenta uma conectividade aresta-nó que é utilizada toda vez que um produto matriz-vetor deve ser realizado. A terceira estratégia de armazenamento utilizada, CSR, tem como principal característica o armazenamento de apenas as posições não nulas da matriz de discretização. E nesse caso, a matriz de discretização esparsa é subtituida por três vetores auxiliares.

Mesmo utilizando estratégias de armazenamento especiais, certas classes de problemas demandam grande tempo de processamento e alto consumo de memória. Atualmente, a computação 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 às versões paralelas de muitos algoritmos. Muitas são as opções de arquiteturas paralelas, as quais variam de supercomputadores, construídos com a finalidade específica de obter alto desempenho, a máquinas como os *clusters* [2], que utilizam computadores comuns interconectados por uma rede. Os supercomputadores são extremamentes eficientes, contudo o custo dessas máquinas é muito alto. Os *clusters* por sua vez possuem uma eficiência menor, entretanto essas máquinas apresentam a vantagem de um bom poder de processamento por um custo bem menor.

Como já foi mencionado um *cluster* é formado por diversas máquinas, as quais possuem seu próprio processador e sua própria memória. Assim, para que um processador conheça dados que não pertençam à sua respectiva memória, são realizadas operações de envios e recebimentos de mensagens. Contudo, para que estas ope-

rações sejam realizadas de maneira mais simples são utilizadas alguns aplicativos que constroem uma interface de software entre as várias máquinas. Existem muitos exemplos desse tipo de aplicativo, os quais se enquadram em classes bem distintas de paradigmas de programação paralela: o *Message Passing Interface* MPI [29] e o *Parallel Virtual Machine* PVM [13] com o paradigma simplesmente de troca de mensagens, o *TreadMarks* [24], com paradigma de *distributed shared memory* e o *High Performance Fortran* HPF [17] com o paradigma de *parallel data*. O padrão MPI foi o escolhido para ser utilizado no presente trabalho, uma vez que apresenta algumas vantagens sobre os demais. Maior desempenho em relação ao *TreadMarks* e HPF e maior versatilidade em relação ao PVM. Em MPI, as operações de envio e recebimento de mensagens são representadas pelas primitivas *send* e *receive* [8], respectivamente. Neste trabalho é utilizado o padrão MPI com a biblioteca lam-mpi, versão 6.5.9-tcp.1.[1]

O problema modelo é dado pela equação de advecção e difusão transiente e bidimensional discretizada por elementos finitos utilizando uma estabilização *Streamline Upwind Petrov-Galerkin* (SUPG) [5]. O sistema linear de equações proveniente da formulação do método dos elementos finitos é resolvido através de método iterativo não-estacionário do resíduo mínimo generalizado (GMRES)[36] para os casos mais gerais e o método dos gradientes conjugados [35] para casos envolvendo apenas difusão. Em ambos os métodos não foram utilizados pré-condicionadores. É realizada uma etapa anterior ao processamento paralelo denominada pré-processamento paralelo. Nesse etapa é gerada a malha de elementos finitos e a mesma é particionada com o auxílio do particionador METIS [23]. É utilizada uma técnica de decomposição de domínio a partir de uma estrutura de dados paralela [20] composta por blocos orientados. São aplicadas sobre a estrutura de dados paralela três estratégias de armazenamento especiais implementadas. Para verificar o desempenho da estrutura paralela proposta, são utilizados quatro problemas exemplos que são modelados pela equação de advecção e difusão.

Dentre os diversos trabalhos encontrados na literatura que apresentam soluções em paralelo para o método dos elementos finitos, pode ser destacado o estudo desenvolvido por Jimack e Touheed [20], em 2000. Neste trabalho é apresentada uma forma eficiente para escrever programas paralelos para o método dos elementos finitos utilizando o padrão MPI. Essa mesma forma abordada no presente trabalho.

[1] www.lam-mpi.org

Outro trabalho correlato é a dissertação de mestrado de Slobodcicov [38], de 2003, que apresenta uma proposta para paralelizar as equações de águas rasas discretizadas pelo método dos elementos finitos. Alguns objetos do trabalho de Slobodcicov foram importantes para o presente trabalho, como por exemplo, a escolha do tipo de particionamento sugerido pelo software METIS. Um ponto em comum entre os dois trabalhos é o uso do algoritmo de solução de sistemas lineares, o GMRES, embora o problema estudado por Slobodcicov envolvesse um conjunto de equações mais complexas que a equação de advecção e difusão, utilizada aqui. Outro trabalho correlato foi apresentado por Kaceniauskas e Rutshmann [22]. Eles descrevem um software para resolver em paralelo as equações de Poisson, de transporte convectivo e alguns problemas modelados pelas equações de Navier Stokes, todas discretizadas por elementos finitos. Utilizam a estratégia de decomposição de domínio e, em comum com o presente trabalho, utilizam para resolver o sistema linear o algoritmo GMRES e para o particionamento do domínio o METIS.

Em relação as estruturas de armazenamento, as discussões realizadas por Ribeiro e Coutinho [34], em 2003, têm uma relação estrita com o presente trabalho. Eles apresentam uma comparação entre as três estratégias de armazenamento aplicadas ao método dos elementos finitos que serão apresentadas no presente trabalho. Outro ponto em comum com o presente trabalho é a escolha dos métodos de solução de sistemas lineares: o método dos gradientes conjugados e o GMRES. As conclusões do trabalho mostram que a estratégia CSR é uma boa escolha em relação ao processamento tempo, mas se a prioridade for o uso de memória sem dúvida a melhor escolha é a estratégia EDE. A tese de mestrado apresentada por Gowda [15] também faz um estudo das estratégias de armazenamento. Ele faz uma comparação entre as estratégias de armazenamento EBE e CSR aplicadas à paralelização da equação de Poisson utilizando a decomposição de domínio referida em [20] e, nesse caso, utiliza o método dos gradientes conjugados para resolver o sistema linear resultante. Gowda conclui que a estratégia CSR é mais rápida que a EBE.

No que se refere aos tipos de máquinas e plataformas utilizadas para realizar os testes de desempenho, pode ser destacado o trabalho de Kaceniauskas e Rutshmann [22], onde a análise do desempenho foi medida através da resolução de alguns problemas modelo em três tipos de arquiteturas: em um convecional cluster de PC's, em um cluster de estações de trabalho IBM RISC [39] e no supercomputador SP2 [1] da IBM. O trabalho de Elias et al. [10] apresentou uma implementação paralela

de esquemas do tipo Newton inexato associados a formulações de estabilização do método dos elementos finitos e os testes de desempenho foram realizados em dois tipos de plataformas, uma plataforma Windows com um mini-cluster composto por 4 *laptops* e, uma plataforma Linux utilizando um cluster com 16 nós dual Intel Pentium III. Os testes relativos ao trabalho de Gowda [15], por sua vez, foram realizados utilizando duas arquiteturas: uma Sun UltraSPARCII[26] e uma SGI MIPS [27].

O presente trabalho consta de mais 5 capítulos além dessa introdução. No próximo capítulo é apresentado o processo de discretização da equação de advecção e difusão pelo método dos elementos finitos. No Capítulo 3 são discutidos os aspectos referentes ao pré-processamento paralelo, é apresentado um resumo sobre as arquiteturas paralelas mais comuns, são descritas as técnicas e estruturas paralelas abordadas em [20] e [31] e finalizando o capítulo são analisados os passos principais do algoritmo paralelo propriamente dito. No Capítulo 4 são realizados estudos sobre três estratégias de armazenamento de matrizes esparsas: EBE, EDE e CSR. O Capítulo 5 contém os resultados obtidos a partir 4 problemas exemplo implementados utilizando as estruturas paralelas e estratégias de armazenamento propostas. No último capítulo são apresentadas as principais conclusões obtidas.

Capítulo 2

Formulação Numérica para a Equação de Advecção e Difusão

Nesse capítulo é apresentada a discretização por elementos finitos da equação de advecção e difusão em duas dimensões usando a formulação SUPG (*Streamline Upwind Petrov Galerkin*)[5]. É realizado um estudo da formulação em nível local, onde cada matriz do elemento é explicitada classificando-se cada parte relativa a formulação SUPG. Na penúltima seção, é discutido o algoritmo utilizado para resolver a discretização do avanço no tempo. Por fim, são apresentados um breve histórico dos métodos de solução de sistemas lineares, o método utilizado para resolver o sistema linear resultante da discretização da equação de advecção e difusão e sugestões de estratégias especiais de armazenamento dos dados para as matrizes resultantes da discretização por elementos finitos.

2.1 Equação transiente de advecção e difusão bidimensional

Seja Ω uma região limitada de $\mathbb{R}^2$, com contorno Γ suave por partes, sendo $\boldsymbol{x} = \{x\ y\}^T = \{x_i\}$, um ponto de $\overline{\Omega} = \Omega \cup \Gamma$ e $\boldsymbol{n} = \{n_x\ n_y\}^T = \{n_i\}$ o vetor normal externo de Γ. Γ_g e Γ_h são subconjuntos de Γ satisfazendo as condições: $\Gamma_g \cup \Gamma_h = \Gamma$ e $\Gamma_g \cap \Gamma_h = \emptyset$. De acordo com Brooks e Hughes [5], a partir das leis de conservação resultantes do balanço da quantidade arbitrária u com seus fluxos σ_i^a e σ_i^d no interior

de Ω, tem-se:

$$u_{,t} + \sigma^a_{i,i} + \sigma^d_{i,i} = f \ , \tag{2.1}$$

onde $\sigma^a_i = \beta_i u$ é o fluxo advectivo; $\sigma^d_i = -k_{ij} u_{,j}$ é o fluxo difusivo; f é o termo de fonte; β_i é o campo de velocidade do fluido e k_{ij} é a difusividade volumétrica, dada por:

$$\boldsymbol{k} = \begin{bmatrix} k_x & 0 \\ 0 & k_y \end{bmatrix} \ . \tag{2.2}$$

Em coordenadas cartesianas a equação (2.1) apresenta a forma:

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x_i}(\beta_i u) - \frac{\partial}{\partial x_i}(k_{ij}\frac{\partial u}{\partial x_j}) = f \tag{2.3}$$

Considerando o campo de velocidade $\boldsymbol{\beta} = \{\beta_x \ \beta_y\}^T$ com divergência nula, ou seja, $div\boldsymbol{\beta} = 0$, tem-se:

$$\frac{\partial u}{\partial t} + \beta_x \frac{\partial u}{\partial x} + \beta_y \frac{\partial u}{\partial y} - \frac{\partial}{\partial x}(k_x \frac{\partial u}{\partial x}) - \frac{\partial}{\partial y}(k_y \frac{\partial u}{\partial y}) = f \tag{2.4}$$

ou

$$\frac{\partial u}{\partial t} + \boldsymbol{\beta}.\nabla u - \nabla.(\boldsymbol{k}\nabla u) = f \ . \tag{2.5}$$

Deseja-se resolver um problema de valor inicial que consiste em encontrar $u(\boldsymbol{x}, t)$ satisfazendo (2.5) em Ω, tal que:

$$u = g \ \text{ em } \ \Gamma_g \ , \tag{2.6}$$

$$-\sigma^d_i n_i = -k_{ij}\frac{\partial u}{\partial x_j} n_j = -\boldsymbol{k}\frac{\partial u}{\partial \boldsymbol{n}} = h \ \text{ em } \ \Gamma_h \ , \tag{2.7}$$

$$u(\boldsymbol{x}, 0) = u_0 \ , \tag{2.8}$$

sendo g e h funções de $\boldsymbol{x}$ e t, e u_0 função apenas de $\boldsymbol{x}$.

2.2 Formulação SUPG

A formulação padrão de Galerkin não gera bons resultados quando o escoamento é fortemente advectivo devido as imprecisões e instabilidades geradas pelo uso de funções testes usuais [18]. A formulação Petrov-Galerkin [5] utilizando funções testes

especiais tem apresentado resultados excelentes para o tratamento de problemas com advecção dominante. Sendo assim, neste trabalho é implementada a formulação SUPG sugerida por Brooks e Hughes [5].

Para definir a formulação fraca ou variacional para o problema (2.5), é necessário definir duas classes de funções [18]. A primeira deve ser composta pelas funções candidatas a solução ou funções de teste. Dentre essas funções, considere o conjunto daquelas que satisfazem a condição de contorno de valor prescrito e também são quadrado integráveis. Esse novo conjunto é denominado conjunto das funções de teste $\mathcal{S}$. O segundo conjunto, denominado $\mathcal{V}$, é formado pelas funções de peso, ou variações. Essas funções são similares as funções de teste, exceto que em Γ_g o valor das funções de peso deve ser igual a zero.

Considera-se o domínio Ω dividido em n_{el} elementos, Ω_e, $e = 1, 2, .., n_{el}$, tal que $\Omega = \bigcup_{e=1}^{n_{el}} \Omega_e$ e $\Omega_i \cap \Omega_j = \emptyset$. As funções $u \in \mathcal{S}$ e $w \in \mathcal{V}$ são aproximadas por funções de dimensão finita $u^h \in \mathcal{S}^h \subset \mathcal{S}$ e $w^h \in \mathcal{V}^h \subset \mathcal{V}$, respectivamente definidas por:

$$\mathcal{S}^h = \{u^h | u^h \in H^{1h}, u^h = g \; em \; \Gamma_g\} \tag{2.9}$$

e

$$\mathcal{V}^h = \{w^h | w^h \in H^{1h}, w^h = 0 \; em \; \Gamma_g\}. \tag{2.10}$$

onde H^{1h} representa o conjunto das funções quadrado integráveis aproximadas também em um espaço de dimensão finita.

No método tradicional de Galerkin, as funções testes consideradas são contínuas ao longo dos contornos entre os elementos. A formulação SUPG define uma função teste descontínua na forma:

$$\tilde{w} = w + p \; , \tag{2.11}$$

onde $w \in \mathcal{V}$ é a função teste contínua e p é a contribuição descontínua no contorno (*streamline upwind*). Estudos realizados por Tezduyar, Liou e Behr [42] demonstram que:

$$\tilde{w} = w + \frac{\tau}{\|\boldsymbol{\beta}\|} \boldsymbol{\beta} \nabla w \; , \tag{2.12}$$

onde $\boldsymbol{\beta}$ é o campo de velocidade do fluido definido em (2.1); $\tau = \frac{h}{2}$, sendo h igual a raiz quadrada da área do elemento e. Portanto, a formulação variacional aproximada

para a discretização SUPG de (2.5) é:

$$\sum_e \int_{\Omega_e} \tilde{w}^h (\frac{\partial u^h}{\partial t} + \boldsymbol{\beta}^h \nabla u^h - \nabla(\boldsymbol{k} \nabla u^h)) d\Omega = \sum_e \int_{\Omega_e} \tilde{w}^h f d\Omega \ . \tag{2.13}$$

A equação (2.13) pode ser representada por três contribuições distintas:

$$\boxed{\text{Galerkin (G)}} + \boxed{\text{Petrov-Galerkin (PG)}} = \boxed{\text{Fonte (F)}} \tag{2.14}$$

$$\begin{aligned} &\underbrace{\int_\Omega w^h (\frac{\partial u^h}{\partial t} + \boldsymbol{\beta}^h \nabla u^h - \nabla(\boldsymbol{k} \nabla u^h)) d\Omega}_{G} + \\ &\underbrace{\sum_e \int_{\Omega_e} p^h (\frac{\partial u^h}{\partial t} + \boldsymbol{\beta}^h \nabla u^h - \nabla(\boldsymbol{k} \nabla u^h)) d\Omega}_{PG} = \\ &\underbrace{\int_\Omega (w^h + p^h) f d\Omega}_{F} \ . \end{aligned} \tag{2.15}$$

Aplicando integração por partes e usando (2.12) para cada contribuição, obtêm-se:

$$\begin{aligned} &\underbrace{\int_\Omega w^h \frac{\partial u^h}{\partial t} d\Omega + \int_\Omega w^h \boldsymbol{\beta}^h \nabla u^h - \int_\Omega \nabla w^h . \boldsymbol{k} \nabla u^h d\Omega}_{G} + \\ &\underbrace{\sum_e \frac{\tau}{\|\boldsymbol{\beta}^h\|} (\int_{\Omega_e} \boldsymbol{\beta}^h . \nabla w^h \frac{\partial u^h}{\partial t} d\Omega + \int_{\Omega_e} \boldsymbol{\beta}^h . \nabla w^h \boldsymbol{\beta}^h . \nabla u^h d\Omega + \int_{\Omega_e} \boldsymbol{\beta}^h . \nabla w^h \nabla . (\boldsymbol{k} \nabla u^h) d\Omega)}_{PG} = \\ &\underbrace{\int_\Omega (w^h + \frac{\tau}{\|\boldsymbol{\beta}^h\|} \boldsymbol{\beta}^h \nabla w^h) f d\Omega}_{F} \end{aligned} \tag{2.16}$$

onde cada parcela recebe denominação relativa a sua representação física, ou seja,

$$\text{(“massa”):} \qquad \int_\Omega w^h \frac{\partial u^h}{\partial t} d\Omega \text{ e } \sum_e \frac{\tau}{\|\boldsymbol{\beta}^h\|} \int_{\Omega_e} \boldsymbol{\beta}^h . \nabla w^h \frac{\partial u^h}{\partial t} d\Omega \tag{2.17}$$

$$\text{(advecção):} \qquad \int_\Omega w \boldsymbol{\beta}^h \nabla u^h d\Omega \text{ e } \sum_e \frac{\tau}{\|\boldsymbol{\beta}^h\|} \int_{\Omega_e} \boldsymbol{\beta}^h . \nabla w^h \boldsymbol{\beta}^h . \nabla u^h d\Omega \tag{2.18}$$

$$\text{(difusão):} \qquad \int_\Omega \nabla w^h . \boldsymbol{k} \nabla u^h d\Omega \text{ e } \sum_e \frac{\tau}{\|\boldsymbol{\beta}^h\|} \int_{\Omega_e} \boldsymbol{\beta}^h . \nabla w^h \nabla . (\boldsymbol{k} \nabla u^h) d\Omega \tag{2.19}$$

$$\text{(fonte):} \qquad \int_\Omega (w^h + \frac{\tau}{\|\boldsymbol{\beta}^h\|} \boldsymbol{\beta}^h \nabla w^h) f d\Omega \tag{2.20}$$

A aproximação por elementos finitos é feita considerando

$$w^h(\boldsymbol{x},t) = \sum_{i=1}^{\eta-\eta_c} c_i(t)\phi_i(\boldsymbol{x}) \tag{2.21}$$

$$u^h(\boldsymbol{x},t) = \sum_{i=1}^{\eta-\eta_c} u_i(t)\phi_i(\boldsymbol{x}) \tag{2.22}$$

$$\frac{\partial u^h}{\partial t}(\boldsymbol{x},t) = \sum_{i=1}^{\eta-\eta_c} a_i(t)\phi_i(\boldsymbol{x}) \tag{2.23}$$

sendo η o total de nós da malha, η_c os nós de contorno físico, c_i um conjunto de constantes arbitrárias, a_i a derivada de u_i em relação a t e ϕ_i uma função de interpolação usual [18] relativa ao nó i. Decorrente dessa aproximação, pode-se representar a equação (2.16) como um sistema de equações diferenciais ordinárias de 1ª ordem:

$$\boldsymbol{M}\boldsymbol{a} + \boldsymbol{K}\boldsymbol{u} = \boldsymbol{F} \tag{2.24}$$

onde $\boldsymbol{u} = \{u_1\ u_2\ \ldots\ u_{\eta-\eta_c}\}^T$ é o vetor de valores nodais, $\boldsymbol{a} = \dfrac{\partial \boldsymbol{u}}{\partial t}$ é o vetor que contém os valores nodais da derivada de $\boldsymbol{u}$ e $\boldsymbol{M}$ é denominada matriz de massa consistente, é obtida através das substituições de (2.21) e (2.23) nos termos de (2.17). $\boldsymbol{K}$, denominada matriz de rigidez, é obtida pelas substituições de (2.21) e (2.22) nos termos de advecção e difusão, descritos em (2.18) e (2.19), respectivamente. O termo de fonte $\boldsymbol{F}$, por sua vez, é obtido pelas substituições de (2.21) e (2.22) em (2.20).

2.3 Estruturas de dados por elementos

As matrizes $\boldsymbol{M}$ e $\boldsymbol{K}$ e o vetor $\boldsymbol{F}$, considerados na seção anterior, podem ser obtidos a partir das contribuições individuais dos elementos. Assim, têm-se:

$$\begin{aligned}
\boldsymbol{M} &= \sum_e (\boldsymbol{m}_g^e + \boldsymbol{m}_{pg}^e)\ , \\
\boldsymbol{K} &= \sum_e (\boldsymbol{k}_g^e + \boldsymbol{k}_{pg}^e)\ , \\
\boldsymbol{F} &= \sum_e (\boldsymbol{f}_g^e + \boldsymbol{f}_{pg}^e)\ .
\end{aligned} \tag{2.25}$$

onde subescritos g e pg representam as contribuições obtidas a partir das formulações de Galerkin e Petrov-Galerkin descritas na seção anterior. As matrizes dos elementos

$\boldsymbol{m}_g^e$, $\boldsymbol{m}_{pg}^e$, $\boldsymbol{k}_g^e$, e $\boldsymbol{k}_{pg}^e$ serão definidas na próxima seção. Os vetores dos elementos $\boldsymbol{f}_g^e$ e $\boldsymbol{f}_{pg}^e$ representam as contribuições dos termos de fonte no elemento e referentes a formulação de Galerkin e Petrov-Galerkin, respectivamente. É importante lembrar que a matriz de rigidez $\boldsymbol{K}$ de (2.25) é obtida pela soma dos termos difusivos e advectivos. Desse ponto em diante, as contribuições armazenadas na matriz $\boldsymbol{K}$ serão desmembradas em duas outras matrizes: $\boldsymbol{D}$, que será considerada a matriz dos termos difusivos, e $\boldsymbol{C}$, a matriz dos termos advectivos. Assim, $\boldsymbol{K}$ pode ser reescrita como:

$$\boldsymbol{K} = \boldsymbol{D} + \boldsymbol{C} = \sum_e (\boldsymbol{d}_g^e + \boldsymbol{d}_{pg}^e + \boldsymbol{c}_g^e + \boldsymbol{c}_{pg}^e) \ , \tag{2.26}$$

onde $\boldsymbol{d}_g^e$ e $\boldsymbol{c}_g^e$ são as matrizes dos elementos relativas as contribuições difusiva e advectiva, respectivamente, referentes a formulação de Galerkin e $\boldsymbol{d}_{pg}^e$ e $\boldsymbol{c}_{pg}^e$ são as matrizes dos elementos relativas também as contribuições difusiva e advectiva, respectivamente, contudo referentes a formulação de Petrov-Galerkin. Essas matrizes serão definidas na próxima seção.

Nesse estudo serão considerados apenas elementos triangulares lineares. Assim, em um elemento genérico e, conforme descrito na Fig. 2.1, a aproximação (2.22) pode ser representada por:

$$u^e(\boldsymbol{x}, t) \cong \sum_{i=1}^{3} u_i(t) N_i(\boldsymbol{x}) \tag{2.27}$$

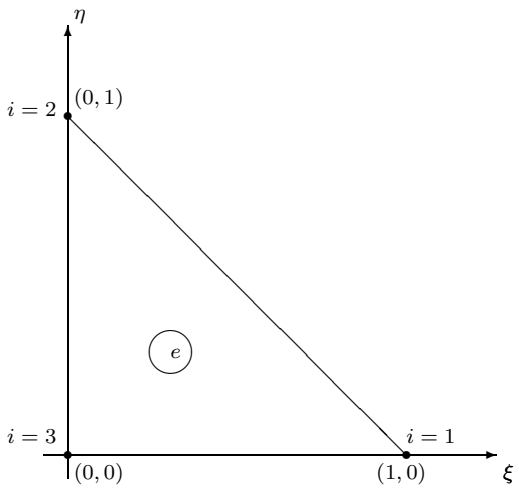


Figura 2.1: Elemento triangular linear

onde N_1, N_2 e N_3 são respectivamente as funções de interpolação no interior do elemento [18] relativas aos nós 1, 2 e 3 do elemento e. Mais precisamente, essas

funções para o caso de elementos triangulares lineares são dadas por:

$$\begin{aligned} N_1 &= \xi \\ N_2 &= \eta \\ N_3 &= 1 - \xi - \eta \end{aligned} \tag{2.28}$$

onde ξ e η representam as variáveis locais no elemento padrão, Fig. 2.1. A matriz Jacobiana que define a transformação das coordenadas globais para as coordenadas locais é dada por:

$$\boldsymbol{J} = \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial x}{\partial \eta} \\ \dfrac{\partial y}{\partial \xi} & \dfrac{\partial y}{\partial \eta} \end{bmatrix} = \begin{bmatrix} x_{13} & -x_{32} \\ -y_{31} & y_{23} \end{bmatrix} \tag{2.29}$$

sendo $x_{ij} = x_i - x_j$ e $y_{ij} = y_i - y_j$, com x_i e y_i coordenadas globais do elemento e e $i, j = 1, 2, 3$. A transformação inversa de $\boldsymbol{J}$ é dada por:

$$\boldsymbol{J}^{-1} = \begin{bmatrix} \dfrac{\partial \xi}{\partial x} & \dfrac{\partial \xi}{\partial y} \\ \dfrac{\partial \eta}{\partial y} & \dfrac{\partial \eta}{\partial y} \end{bmatrix} = \frac{1}{2A^e} \begin{bmatrix} y_{23} & x_{32} \\ y_{31} & x_{13} \end{bmatrix} \tag{2.30}$$

onde A^e é a área do elemento e. É comum definir o operador gradiente discreto das funções de interpolação por:

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{B}_x \\ \boldsymbol{B}_y \end{bmatrix} = \begin{bmatrix} \dfrac{\partial N}{\partial x} \\ \dfrac{\partial N}{\partial y} \end{bmatrix} = \frac{1}{2A^e} \begin{bmatrix} \dfrac{\partial N_1}{\partial x} & \dfrac{\partial N_2}{\partial x} & \dfrac{\partial N_3}{\partial x} \\ \dfrac{\partial N_1}{\partial y} & \dfrac{\partial N_2}{\partial y} & \dfrac{\partial N_3}{\partial y} \end{bmatrix} = \frac{1}{2A^e} \begin{bmatrix} y_{23} & y_{31} & y_{12} \\ x_{32} & x_{13} & x_{21} \end{bmatrix} \tag{2.31}$$

2.4 Matrizes dos elementos

Nessa seção são descritas as matrizes dos elementos a partir das parcelas de Galerkin e de Petrov-Galerkin definidas em (2.25) e (2.26). Em todos os casos são consideradas as funções de interpolação descritas em (2.28). Todas as integrais podem ser resolvidas sem uso de aproximações numéricas devido a simplicidade das funções de interpolação escolhidas.

2.4.1 Matriz de massa consistente de Galerkin

Aplicando as aproximações de elementos finitos descritas na Seção 2.2 a cada elemento na parcela da massa consistente de Galerkin, descrita em (2.16), verifica-se que:

$$\int_{\Omega_e} w^h \frac{\partial u^h}{\partial t} d\Omega \quad \Longrightarrow \quad \boldsymbol{m}_g^e = \int_{\Omega_e} \boldsymbol{N}^T \boldsymbol{N} d\Omega \ , \tag{2.32}$$

onde $\boldsymbol{N} = \{N_1, N_2, N_3\}^T$, conforme definido em (2.28). Devido as particularidades, já mencionadas de $\boldsymbol{N}$, tem-se:

$$\boldsymbol{m}_g^e = \frac{A^e}{12} \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} \ , \tag{2.33}$$

onde A^e é a área do elemento e.

2.4.2 Matriz de difusão de Galerkin

Como resultado das aproximações impostas pelo método de elementos finitos, a matriz referente a parcela de difusão de Galerkin de (2.16), para um dado elemento, pode ser obtida por:

$$\int_{\Omega_e} \nabla w^h . \boldsymbol{k} \nabla u^h d\Omega \quad \Longrightarrow \quad \boldsymbol{d}_g^e = \int_{\Omega_e} \boldsymbol{B}^T \boldsymbol{k} \boldsymbol{B} d\Omega \ , \tag{2.34}$$

Como o integrando $\boldsymbol{B}^T \boldsymbol{k} \boldsymbol{B}$ é constante e fazendo $b_1 = y_{23}$, $b_2 = y_{31}$, $b_3 = y_{12}$, $a_1 = x_{32}$, $a_2 = x_{13}$ $a_3 = x_{21}$ é fácil verificar que:

$$\boldsymbol{d}_g^e = \frac{1}{4A^e} \begin{bmatrix} k_x b_1 b_1 + k_y a_1 a_1 & k_x b_1 b_2 + k_y a_1 a_2 & k_x b_1 b_3 + k_y a_1 a_3 \\ k_x b_2 b_1 + k_y a_2 a_1 & k_x b_2 b_2 + k_y a_2 a_2 & k_x b_2 b_3 + k_y a_2 a_3 \\ k_x b_3 b_1 + k_y a_3 a_1 & k_x b_3 b_2 + k_y a_3 a_2 & k_x b_3 b_3 + k_y a_3 a_3 \end{bmatrix} \tag{2.35}$$

Na matriz descrita em (2.35), observa-se a propriedade de deficiência de posto, ou seja, observa-se que:

$$\begin{aligned} (k_x b_1 b_1 + k_y a_1 a_1) &= -(k_x b_1 b_2 + k_y a_1 a_2) - (k_x b_1 b_3 + k_y a_1 a_3) \\ (k_x b_2 b_2 + k_y a_2 a_2) &= -(k_x b_2 b_1 + k_y a_2 a_1) - (k_x b_2 b_3 + k_y a_2 a_3) \\ (k_x b_3 b_3 + k_y a_3 a_3) &= -(k_x b_3 b_1 + k_y a_3 a_1) - (k_x b_3 b_2 + k_y a_3 a_2) \end{aligned} \tag{2.36}$$

define-se então:

$$d_{ij} = \frac{1}{4A^e}(k_x b_i b_j + k_y a_i a_j), \quad i,j = 1,2,3 \ \text{ e } \ i \neq j \tag{2.37}$$

Observando que $d_{ij} = d_{ji}$, é possível escrever a matriz de difusão de Galerkin com apenas três coeficientes distintos:

$$\boldsymbol{d}_g^e = \frac{1}{4A^e}\begin{bmatrix} -(d_{12}+d_{13}) & d_{12} & d_{13} \\ d_{12} & -(d_{12}+d_{23}) & d_{23} \\ d_{13} & d_{23} & -(d_{13}+d_{23}) \end{bmatrix} \tag{2.38}$$

2.4.3 Matriz de advecção de Galerkin

As aproximações do método dos elementos finitos aplicadas a parcela de advecção em (2.16) definem a matriz de advecção de Galerkin em cada elemento:

$$\int_{\Omega_e} w^h \boldsymbol{\beta}^h \nabla u^h d\Omega \quad \Longrightarrow \quad \boldsymbol{c}_g^e = \int_{\Omega_e} \boldsymbol{N}^T \boldsymbol{\beta}^T \boldsymbol{B} d\Omega \tag{2.39}$$

considerando o campo de velocidade constante no elemento e novamente fazendo $b_1 = y_{23}$, $b_2 = y_{31}$, $b_3 = y_{12}$, $a_1 = x_{32}$, $a_2 = x_{13}$ e $a_3 = x_{21}$ obtém-se:

$$\boldsymbol{c}_g^e = \frac{1}{6}\begin{bmatrix} \beta_x b_1 + \beta_y a_1 & \beta_x b_2 + \beta_y a_2 & \beta_x b_3 + \beta_y a_3 \\ \beta_x b_1 + \beta_y a_1 & \beta_x b_2 + \beta_y a_2 & \beta_x b_3 + \beta_y a_3 \\ \beta_x b_1 + \beta_y a_1 & \beta_x b_2 + \beta_y a_2 & \beta_x b_3 + \beta_y a_3 \end{bmatrix}, \tag{2.40}$$

e observando a deficiência de posto:

$$\begin{aligned} (\beta_x b_1 + \beta_y a_1) &= -(\beta_x b_2 + \beta_y a_2) - (\beta_x b_3 + \beta_y a_3) \\ (\beta_x b_2 + \beta_y a_2) &= -(\beta_x b_1 + \beta_y a_1) - (\beta_x b_3 + \beta_y a_3) \\ (\beta_x b_3 + \beta_y a_3) &= -(\beta_x b_1 + \beta_y a_1) - (\beta_x b_2 + \beta_y a_2) \ , \end{aligned} \tag{2.41}$$

a matriz (2.40) pode ser reescrita como:

$$\boldsymbol{c}_g^e = \frac{1}{6}\begin{bmatrix} -(c_{12}+c_{13}) & c_{12} & c_{13} \\ c_{21} & -(c_{21}+c_{23}) & c_{23} \\ c_{31} & c_{32} & -(c_{31}+c_{32}) \end{bmatrix}, \tag{2.42}$$

onde

$$c_{ij} = \beta_x b_i + \beta_y a_i \ \ i,j = 1,2,3 \ \text{ e } \ i \neq j \ . \tag{2.43}$$

Observa-se a não simetria da matriz $\boldsymbol{c}_g^e$, porém, levando em consideração (2.40), nota-se que são necessários apenas três coeficientes distintos para representá-la:

$$\boldsymbol{c}_g^e = \frac{1}{6}\begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{11} & c_{12} & c_{13} \\ c_{11} & c_{12} & c_{13} \end{bmatrix} \tag{2.44}$$

2.4.4 Correção SUPG na matriz de massa

Após aplicar aproximações do método de elementos finitos a parcela de massa presente na expressão (2.16), obtém-se para cada elemento:

$$\frac{\tau}{\|\boldsymbol{\beta}^h\|}\int_{\Omega_e} \boldsymbol{\beta}^h.\nabla w^h \frac{\partial u^h}{\partial t} d\Omega \quad \Longrightarrow \quad \boldsymbol{m}_{pg}^e = \frac{\tau}{\|\boldsymbol{\beta}\|}\int_{\Omega_e} \boldsymbol{B}^T\boldsymbol{\beta}\boldsymbol{N} d\Omega \tag{2.45}$$

Assim, considerando constante o campo de velocidades no elemento tem-se:

$$\boldsymbol{m}_{pg}^e = \frac{\tau}{6\|\boldsymbol{\beta}\|}\begin{bmatrix} \beta_x b_1 + \beta_y a_1 & \beta_x b_1 + \beta_y a_1 & \beta_x b_1 + \beta_y a_1 \\ \beta_x b_2 + \beta_y a_2 & \beta_x b_2 + \beta_y a_2 & \beta_x b_2 + \beta_y a_2 \\ \beta_x b_3 + \beta_y a_3 & \beta_x b_3 + \beta_y a_3 & \beta_x b_3 + \beta_y a_3 \end{bmatrix}, \tag{2.46}$$

como em (2.46) ocorre a mesma deficiência de posto apresentada em (2.45), a matriz $\boldsymbol{m}_{pg}^e$ pode ser reescrita:

$$\boldsymbol{m}_{pg}^e = \frac{\tau}{6\|\boldsymbol{\beta}\|}\begin{bmatrix} -(m_{21}+m_{31}) & m_{21} & m_{13} \\ m_{21} & -(m_{21}+m_{32}) & m_{23} \\ m_{31} & m_{32} & -(m_{13}+m_{23}) \end{bmatrix}, \tag{2.47}$$

onde tem-se:

$$m_{ij} = \beta_x b_i + \beta_y a_i \quad i,j = 1,2,3 \ \text{ e } \ i \neq j \ . \tag{2.48}$$

Verifica-se que $\boldsymbol{m}_{pg}^e$ é igual a $(\boldsymbol{c}_g^e)^T$ a menos de uma constante. Portanto, $\boldsymbol{m}_{pg}^e$ também pode ser representada por apenas três coeficientes distintos:

$$\boldsymbol{m}_{pg}^e = \frac{\tau}{6\|\boldsymbol{\beta}\|}\begin{bmatrix} m_{11} & m_{11} & m_{11} \\ m_{22} & m_{22} & m_{22} \\ m_{33} & m_{33} & m_{33} \end{bmatrix} = \frac{\tau}{6\|\boldsymbol{\beta}\|}\begin{bmatrix} c_{11} & c_{11} & c_{11} \\ c_{12} & c_{12} & c_{12} \\ c_{13} & c_{13} & c_{13} \end{bmatrix}, \tag{2.49}$$

onde c_{11}, c_{12} e c_{13} são os termos da matriz (2.44).

2.4.5 Correção SUPG na matriz de difusão

A partir da parcela de difusão existente na expressão (2.16) obtém-se para cada elemento:

$$\frac{\tau}{\|\boldsymbol{\beta}^h\|}\int_{\Omega_e}\boldsymbol{\beta}^h.\nabla w^h\nabla.(\boldsymbol{k}\nabla u^h)d\Omega \quad \Longrightarrow \quad \boldsymbol{d}^e_{pg}=\frac{\tau}{\|\boldsymbol{\beta}\|}\int_{\Omega_e}\boldsymbol{B}^T\boldsymbol{\beta}\nabla(\boldsymbol{k}\nabla\boldsymbol{B})d\Omega \quad (2.50)$$

O integrando da expressão (2.50) depende das derivadas de segunda ordem das funções de interpolação. Neste caso considera-se funções lineares, portanto a contribuição SUPG do termo de difusão nulo, ou seja,

$$\boldsymbol{d}^e_{pg}=0 \quad (2.51)$$

2.4.6 Correção SUPG na matriz de advecção

A matriz de correção SUPG da parcela de advecção presente (2.16), para cada elemento, obtida após aproximações de elementos finitos, é obtida como:

$$\frac{\tau}{\|\boldsymbol{\beta}^h\|}\int_{\Omega_e}\boldsymbol{\beta}^h.\nabla w^h\boldsymbol{\beta}^h.\nabla u^h d\Omega \quad \Longrightarrow \quad \boldsymbol{c}^e_{pg}=\frac{\tau}{\|\boldsymbol{\beta}\|}\int_{\Omega_e}\boldsymbol{B}^T\boldsymbol{\beta}\boldsymbol{\beta}^T\boldsymbol{B}d\Omega \quad (2.52)$$

Considerando a velocidade constante no elemento e assumindo:

$$\begin{aligned}\beta_{xx}&=\beta_x\beta_x\\ \beta_{xy}&=\beta_x\beta_y\\ \beta_{yy}&=\beta_y\beta_y\ ,\end{aligned} \quad (2.53)$$

chega-se a:

$$(\boldsymbol{c}^e_{pg})_{ij}=\frac{\tau}{4A^e\|\boldsymbol{\beta}\|}(\beta_{xx}b_ib_j+\beta_{xy}b_ia_j+\beta_{xy}a_ib_j+\beta_{yy}a_ia_j)\ \ i,j=1,2,3\ . \quad (2.54)$$

Visto que $\boldsymbol{c}^e_{pg}$ é simétrica e que nesse caso também existe a mesma propriedade de deficiência de posto definida nas matrizes anteriores, ou seja, os elementos da diagonal de $\boldsymbol{c}^e_{pg}$ podem ser obtidos como a soma dos simétricos dos elementos de fora da diagonal em cada linha, obtém-se a matriz de correção de advecção de forma semelhante à matriz de difusão de Galerkin:

$$\boldsymbol{c}^e_{pg}=\frac{\tau}{4A^e\|\boldsymbol{\beta}\|}\begin{bmatrix}-(c^{pg}_{12}+c^{pg}_{13}) & c^{pg}_{12} & c^{pg}_{13}\\ c^{pg}_{12} & -(c^{pg}_{12}+c^{pg}_{23}) & c^{pg}_{23}\\ c^{pg}_{13} & c^{pg}_{23} & -(c^{pg}_{13}+c^{pg}_{23})\end{bmatrix}, \quad (2.55)$$

onde aparecem apenas 3 termos distintos são:

$$c_{12}^{pg} = \frac{\tau}{4A^e\|\boldsymbol{\beta}\|}(\beta_{xx}b_1b_2 + \beta_{xy}b_1a_2 + \beta_{xy}a_1b_2 + \beta_{yy}a_1a_2) \tag{2.56}$$

$$c_{13}^{pg} = \frac{\tau}{4A^e\|\boldsymbol{\beta}\|}(\beta_{xx}b_1b_3 + \beta_{xy}b_1a_3 + \beta_{xy}a_1b_3 + \beta_{yy}a_1a_3) \tag{2.57}$$

$$c_{23}^{pg} = \frac{\tau}{4A^e\|\boldsymbol{\beta}\|}(\beta_{xx}b_2b_3 + \beta_{xy}b_1a_3 + \beta_{xy}a_2b_3 + \beta_{yy}a_2a_3) \tag{2.58}$$

2.5 Algoritmo de avanço no tempo

Para resolver o sistema (2.24) foi utilizado um algoritmo preditor-multicorretor definido em [19]. Neste algoritmo a equação (2.24) é discretizada por diferenças finitas. Sendo n o contador de passos no tempo, as aproximações para $\boldsymbol{a}(t_n)$ e $\boldsymbol{u}(t_n)$, representadas por $\boldsymbol{a}_n$ e $\boldsymbol{u}_n$ respectivamente, Δt o passo de tempo e, dados $\boldsymbol{u}_n$ e $\boldsymbol{a}_n$, o algoritmo pode ser resumido nos seguintes passos:

Predição:

$$i = 0 \tag{2.59}$$

$$\boldsymbol{u}_{n+1}^{(0)} = \boldsymbol{u}_n + 0.5\Delta t\boldsymbol{a}_n \tag{2.60}$$

$$\boldsymbol{a}_{n+1}^{(0)} = 0 \tag{2.61}$$

Correção:

$$i = i + 1 \tag{2.62}$$

$$\text{Calcule : } \boldsymbol{R} = \boldsymbol{F} - (\boldsymbol{M}\boldsymbol{a}_{n+1}^{(i-1)} + \boldsymbol{K}\boldsymbol{u}_{n+1}^{(i-1)}) \tag{2.63}$$

$$\text{Resolva : } \boldsymbol{M}^*\Delta\boldsymbol{a} = \boldsymbol{R} \tag{2.64}$$

$$\text{Atualize : } \boldsymbol{a}_{n+1}^{(i)} = \boldsymbol{a}_{n+1}^{(i-1)} + \Delta\boldsymbol{a} \tag{2.65}$$

$$\text{Atualize : } \boldsymbol{u}_{n+1}^{(i)} = \boldsymbol{u}_{n+1}^{(i-1)} + 0.5\Delta t\Delta\boldsymbol{a} \tag{2.66}$$

onde $\boldsymbol{M}^* = \boldsymbol{M} + 0.5\Delta t\boldsymbol{K}$ é uma matriz esparsa da ordem do número de nós incógnitas, $\boldsymbol{R} = \boldsymbol{F} - [\boldsymbol{M}\boldsymbol{a} + \boldsymbol{K}\boldsymbol{u}]$ é o vetor de resíduos em função dos valores iniciais da multicorreção dos valores nodais de $\boldsymbol{a}$ e $\boldsymbol{u}$, e $\Delta\boldsymbol{a}$ é a correção dos valores de $\boldsymbol{a}$ para a próxima iteração. Como o Δt adotado é fixo, a matriz $\boldsymbol{M}^*$ não varia em cada passo no tempo, mas o mesmo não ocorre com o vetor $\boldsymbol{R}$ que depende dos valores atualizados de $\boldsymbol{a}$ e $\boldsymbol{u}$.

2.6 Resumo das matrizes dos elementos

Levando em consideração as matrizes dos elementos discutidas na presente seção, é possível explicitar as contribuições em cada elemento relativas as matrizes $\boldsymbol{M}$, $\boldsymbol{K}$ relacionados com o cálculo do vetor de resíduos $\boldsymbol{R}$ e as contribuições da matriz $\boldsymbol{M}^*$ referentes ao sistema (2.64):

$$\boldsymbol{M} = \mathop{\mathbf{A}}_{e=1}^{n_{el}} (\boldsymbol{m}^e) \tag{2.67}$$

$$\boldsymbol{K} = \mathop{\mathbf{A}}_{e=1}^{n_{el}} (\boldsymbol{k}^e) \tag{2.68}$$

$$\boldsymbol{M}^* = \mathop{\mathbf{A}}_{e=1}^{n_{el}} ((\boldsymbol{m}^*)^e) \tag{2.69}$$

Assim, a matriz $\boldsymbol{m}^e$ pode ser detalhada em nível de cada elemento da seguinte forma:

$$\boldsymbol{m}^e = \boldsymbol{m}^e_g + \boldsymbol{m}^e_{pg} = \frac{A^e}{12}\begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} + \frac{\tau}{6\|\boldsymbol{\beta}\|}\begin{bmatrix} c_{11} & c_{11} & c_{11} \\ c_{12} & c_{12} & c_{12} \\ c_{13} & c_{13} & c_{13} \end{bmatrix} \tag{2.70}$$

com $\boldsymbol{m}^e_{ij} = \dfrac{A^e}{12} + \dfrac{\tau}{6\|\boldsymbol{\beta}\|}c_{1i}$ para $i, j = 1, 2, 3 \quad i \neq j$ e $\boldsymbol{m}^e_{ii} = \dfrac{2A^e}{12} + \dfrac{\tau}{6\|\boldsymbol{\beta}\|}c_{1i}$ para $i = 1, 2, 3$. Nota-se que para armazenar os coeficientes de $\boldsymbol{m}^e$ seriam necessários 6 coeficientes, contudo são utilizados apenas 4, pois a soma descrita em (2.70) é efetuada somente quando $\boldsymbol{m}^e$ for solicitada, ou seja, para cada elemento são armazenados apenas o termo $\dfrac{A^e}{12}$ referente à $\boldsymbol{m}^e_g$ e 3 coeficientes distintos de $\boldsymbol{m}^e_{pg}$.

A construção da matriz $\boldsymbol{k}^e$ em cada elemento pode ser vista como:

$$\begin{aligned}\boldsymbol{k}^e = \boldsymbol{d}^e_g + \boldsymbol{d}^e_{pg} + \boldsymbol{c}^e_g + \boldsymbol{c}^e_{pg} =& \frac{1}{4A^e}\begin{bmatrix} -(d_{12}+d_{13}) & d_{12} & d_{13} \\ d_{12} & -(d_{12}+d_{23}) & d_{23} \\ d_{13} & d_{23} & -(d_{13}+d_{23}) \end{bmatrix} + \\ & \frac{1}{6}\begin{bmatrix} -(c_{12}+c_{13}) & c_{12} & c_{13} \\ c_{21} & -(c_{21}+c_{23}) & c_{23} \\ c_{31} & c_{32} & -(c_{31}+c_{32}) \end{bmatrix} + \\ & \frac{\tau}{4A^e\|\boldsymbol{\beta}\|}\begin{bmatrix} -(c^{pg}_{12}+c^{pg}_{13}) & c^{pg}_{12} & c^{pg}_{13} \\ c^{pg}_{12} & -(c^{pg}_{12}+c^{pg}_{23}) & c^{pg}_{23} \\ c^{pg}_{13} & c^{pg}_{23} & -(c^{pg}_{13}+c^{pg}_{23}) \end{bmatrix},\end{aligned} \tag{2.71}$$

onde $\boldsymbol{k}_{ij}^e = \frac{1}{4A^e} d_{ij} + \frac{1}{6} c_{ij} + \frac{\tau}{4A^e \|\boldsymbol{\beta}\|} c_{ij}^{pg}$ com $i, j = 1, 2, 3 \;\; i \neq j$, $d_{ij} = d_{ji}$ e $c_{ij}^{pg} = c_{ji}^{pg}$. Como $\boldsymbol{c}_g^e$ não é simétrica são necessários 6 coeficientes para representar a matriz $\boldsymbol{k}^e$.

Após definir as matrizes $\boldsymbol{m}^e$ e $\boldsymbol{k}^e$ a matriz $(\boldsymbol{m}^*)^e$ pode ser construída facilmente:

$$(\boldsymbol{m}^*)^e = \boldsymbol{m}^e + 0.5\Delta t \boldsymbol{k}^e \tag{2.72}$$

No presente trabalho, a matriz (2.72) não é montada para cada elemento, uma vez que suas partes integrantes $\boldsymbol{m}^e$ e $\boldsymbol{k}^e$ são utilizadas separadamente para o cálculo do vetor de resíduos $\boldsymbol{R}$ de (2.64). Também em (2.64) nota-se a necessidade de resolver um sistema linear. Na próxima seção será feito um breve histórico sobre alguns métodos de solução de sistemas lineares e principalmente o método utilizado para resolver o sistema linear em questão.

2.7 Resolução do sistema linear

O processo de discretização do método dos elementos finitos gera sistemas lineares de grande porte que precisam ser resolvidos através de um método numérico adequado para preservar a máxima precisão. Os métodos para solução de sistemas lineares são divididos em dois grupos: os métodos diretos e métodos iterativos. Os métodos diretos são aqueles que forneceriam a solução exata, não fossem os erros de arredondamento. Como exemplo tem-se o método de eliminação gaussiana, decomposição LU e método frontal. Os métodos iterativos são aqueles que permitem obter a solução de um sistema linear com uma dada precisão através de um processo convergente. Tais métodos são subdivididos em métodos iterativos estacionários, como por exemplo método de Jacobi, o método de Gauss-Seidel e método SOR. No outro grupo estão os métodos iterativos não-estacionários, como por exemplo o método dos gradientes conjugados, o método do resíduo mínimo generalizado (GMRES) e o método das direções conjugadas à esquerda (LCD). Esses métodos em sua maior parte são baseados em subespaços de Krylov.

Para resolver o sistema linear (2.64) efetivamente, foi utilizado o método GMRES [35]. Não foram utilizados pré-condicionadores ou quaisquer outros tipos de aceleradores de convergência. As matrizes $\boldsymbol{M}$, $\boldsymbol{K}$ e $\boldsymbol{M}^*$ geradas pelo processo de discretização são esparsas, mas devido a propriedades inerentes ao método dos elementos finitos, essa esparsidade não é trivial, ou seja, fora os coeficientes da diagonal,

os coeficientes não nulos estão dispostos ao longo das matrizes seguindo o padrão proposto pela conectividade dos elementos, o que gera coeficientes não nulos em posições bem alternadas. Visando reduzir o consumo de memória, as matrizes de discretização são armazenadas de forma alternativa. Algumas estratégias de armazenamento como elemento-por-elemento (EBE), aresta-por-aresta (EDE) e *compressed storage row* (CSR) são discutidas. O GMRES, assim como os demais métodos não-estacionários, possui duas operações principais: o produto matriz-vetor e o produto interno. Assim, as estratégias propostas devem ser definidas de forma a garantir uma boa eficiência na realização dessas duas operações.

Levando em conta estratégias de armazenamento especiais como as mencionadas anteriormente, um sistema linear da forma (2.64) com poucas milhares de incógnitas pode ser resolvido com certa facilidade e eficiência utilizando uma computador convencional. Entretanto, quando o número de equações do sistema linear cresce, os computadores convencionais passam a ter dois problemas: o consumo de memória e o tempo de processamento. Também em alguns casos, quando o computador está apto a resolver algum problema, o tempo de processamento é tão grande que inviabiliza tal procedimento. Nesses casos, a processamento paralelo tem mostrado grandes resultados, tanto no gerenciamento de uma quantidade de memória bem maior quanto na redução do tempo de execução dos algoritmos.

Capítulo 3

Processamento Paralelo

No presente capítulo são abordadas as principais etapas do processo de paralelização do métodos dos elementos finitos. Inicialmente, são apresentadas sucintamente as características das diversas arquiteturas paralelas. Em seguida, trata-se o problema da geração e particionamento da malha para o método dos elementos finitos. São discutidas algumas técnicas de pré-processamento para preparar o conjunto de dados para o processamento paralelo propriamente dito. Nesse capítulo também é apresentada a estrutura de dados utilizada para efetuar o processamento paralelo. Na seção seguinte, são comentados os passos principais do algoritmo paralelo e uma análise sobre a esparsidade gerada pelo sistema linear oriundo da discretização por elementos finitos aplicada à estrutura de dados paralela proposta.

3.1 Arquiteturas paralelas

Nos últimos 20 anos problemas como modelagem do clima, mapeamento do genoma humano, modelagem de semicondutores, supercondutores e dispersão de poluentes têm requerido um poder de processamento bem maior que aquele obtido através de computadores convencionais. Assim, surge um novo campo da computação denominado computação paralela.

Os estudos em computação paralela são motivados por fatores, como por exemplo, avanços nas tecnologias envolvidas na produção dos componentes eletrônicos que tornam-se cada vez mais acessíveis financeiramente, mais eficientes no consumo de potência e mais rápidos.

Arquiteturas de diversos tipos, elaboradas para aplicações específicas, podem ser utilizadas para acelerar a execução dos problemas que exigem alto desempenho. Sendo assim, faz-se necessário um rápido estudo sobre as classificações dessas arquiteturas. As arquiteturas podem ser classificadas de três maneiras: segundo os níveis de paralelismo, segundo a classificação de Flynn [12] e segundo o compartilhamento de memória.

Em relação aos níveis de paralelismo a classificação dá-se em nível de instrução, também denominada granularidade fina. Como exemplos têm-se as arquiteturas *pipelined* [33], superescalares [32] e VLIW [11]. Em nível de *thread*, conhecida também por granularidade média, têm-se como exemplos as arquiteturas *multithreaded* [21] e SMT[9]. E por último, paralelismo em nível de processo, ou granularidade grossa, cujos exemplos bem conhecidos são os multiprocessadores e multicomputadores [4].

De acordo com Flynn [12] as arquiteturas são classificadas baseado na forma como as instruções e os dados são processados. As combinações entre esses dois fatores geram quatro classificações: instrução única, dado único (SISD); instrução única, múltiplos dados (SIMD); múltiplas instruções, dado único (MISD); e múltiplas instruções, múltiplos dados (MIMD).

As arquiteturas são classificadas em relação ao uso de memória como máquinas com memória compartilhada, máquinas com memória distribuída e máquinas com memória centralizada. As máquinas com memória compartilhada possuem um único espaço de endereçamento e trabalham via operações de *load* e *store* [14]. Já as máquinas que não possuem memória compartilhada trabalham com múltiplos espaços de endereçamento privados para cada processador e, nesse caso, as operações utilizadas são *send* e *receive* [8]. A memória é classificada como distribuída de acordo com a localização física da mesma, ou seja, se a memória é implementada com diversos módulos e cada módulo é posto próximo de um processador. Por sua vez, memória centralizada é a memória que se encontra a uma mesma distância dos processadores, independente de ter sido implementada com um ou diversos módulos.

Alguns fatores que podem influenciar a escolha de uma arquitetura para a implementação de um dado algoritmo são as estruturas de comunicação requeridas, o tamanho da memória e a presença ou ausência de mecanismos de sincronismo [3][30]. No presente trabalho o tipo de arquitetura utilizada foi a de memória distribuída, mais particularmente clusters de estações de trabalho. Cluster pode ser definido

como um sistema onde dois ou mais computadores trabalham de maneira conjunta para realizar processamento pesado. Em outras palavras, os computadores dividem as tarefas de processamento e trabalham como se fossem um único computador. O principal motivo da referida escolha foi a opção de um grande poder de processamento por um baixo custo. Assim, todas as implementações utilizadas no presente trabalho seguem o paradigma aplicado a tal arquitetura.

3.2 Geração e particionamento da malha de elementos finitos

O método dos elementos finitos possibilita uma discretização de um domínio Ω numa grande generalidade de subdomínios, denominados elementos, tais como, triângulos e quadriláteros para o caso 2D e tetraedros e hexaedros para o caso 3D. Uma das características mais marcantes do método dos elementos finitos é a capacidade de tratar malhas não-estruturadas, o que possibilita modelagens de geometrias complicadas e irregulares de forma mais conveniente.

Gerar malhas não-estruturadas não é uma tarefa trivial. Foi utilizado um software de domínio público chamado *Generator Mesh – GMSH* [1]. Esse gerador de malhas 2D ou 3D possibilita a visualização gráfica da malha gerada e a criação de arquivos contendo a conectividade entre os nós da malha. Após gerar a malha, o próximo passo é criar as partições visando o processamento paralelo. Dessa forma, divide-se o domínio em subdomínios que fazem parte de processadores distintos. No presente trabalho, optou-se por utilizar o número de partições igual ao número de processadores.

Em geral, dividir a quantidade de nós ou elementos entre os processadores de forma que cada parte seja aproximadamente igual é algo trivial. Entretando, conseguir minimizar o número de comunicações mantendo o balenceamento não é uma tarefa simples. Sendo assim, decidiu-se pelo uso de um particionador disponível e de domínio público. O programa escolhido, METIS [23], forma um conjunto de softwares que podem ser utilizados como particionador de grafos, particionador de malhas e otimizador de matrizes esparsas. Dentre estas funcionalidades, foram utilizadas

[1] www.geuz.org/gmsh/

aquelas que são suficientes para particionar malhas estruturadas ou não-estruturadas geradas para o método dos elementos finitos. O programa METIS possibilita dois tipos de particionamentos: o particionamento dual e o particionamento nodal. Ambos convertem a malha em um grafo e em seguida particionam este grafo. No caso dual, o grafo é construído utilizando os elementos como vértices do grafo. Já no caso nodal, a malha é convertida em um grafo nodal, ou seja, cada nó da malha torna-se um vértice do grafo. A partir dos resultados obtidos em [38], decidiu-se aplicar o procedimento dual.

Visando facilitar o entendimento da qualidade do particionamento e de como um particiomento inadequado pode comprometer severamente o tempo de processamento, é considerada, sem perda de generalidade, uma malha exemplo na Fig. 3.1 contendo 49 nós e 72 elementos, cujos nós ao redor da malha são nós de valor prescrito. Considere também uma divisão dessa malha ao longo de quatro processadores. O particionamento precisa ser balanceado, ou seja, o número de nós ou elementos a serem processados devem ser aproximadamente o mesmo em cada um dos processadores. Uma outra preocupação deve ser a fronteira de comunicação, isto é, um mesmo processador deve compartilhar com os demais processadores o menor número de nós possível. Se o balanceamento for inadequado, há uma grande chance de ocorrer situações onde alguns processadores poderão trabalhar muito, enquanto outros ficarão ociosos, o que conduzirá a um mau aproveitamento do poder de processamento do equipamento. Se um processador compartilhar uma quantidade de nós maior que a mínima requerida, no caso de arquiteturas com memória distribuída, o número de mensagens de envio e recebimento poderá ser acrescido, o que conseqüentemente aumentará o tempo gasto com sincronização do processo e com o tráfego de informações na rede.

Foram efetuados dois tipos de particionamento visando o processamento paralelo. O primeiro deles, em destaque na Fig. 3.2(a), foi obtido da forma mais trivial possível, ou seja, o domínio foi dividido ao meio na direção horizontal e vertical. Repare que o particionamento está balanceado, mas cada processador tem uma fronteira de comunicação com outros três. O segundo particionamento, visto na Fig. 3.2(b), foi obtido utilizando o particionamento dual do METIS. Esse partionamento também manteve o balanceamento, uma vez que, a quantidade de elementos em cada um dos processadores é a mesma. Entretanto, dois dos processadores têm fronteira de particionamento apenas com outros dois, diminuindo a quantidade de mensagens

a serem trocadas entre os processadores.

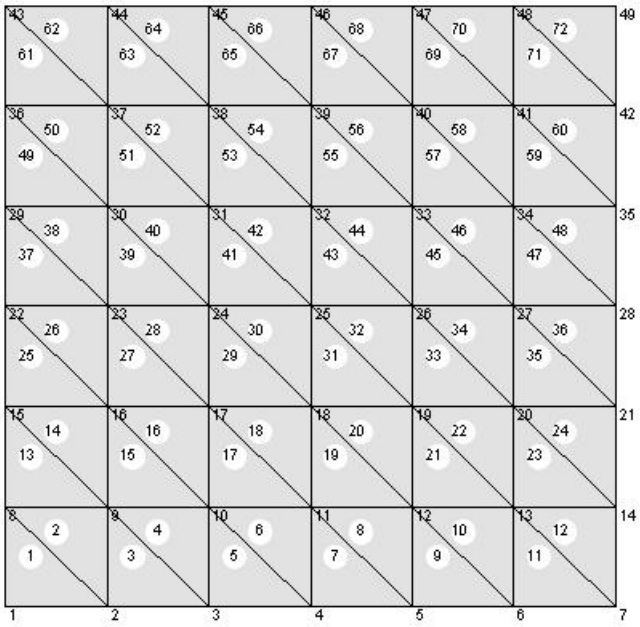
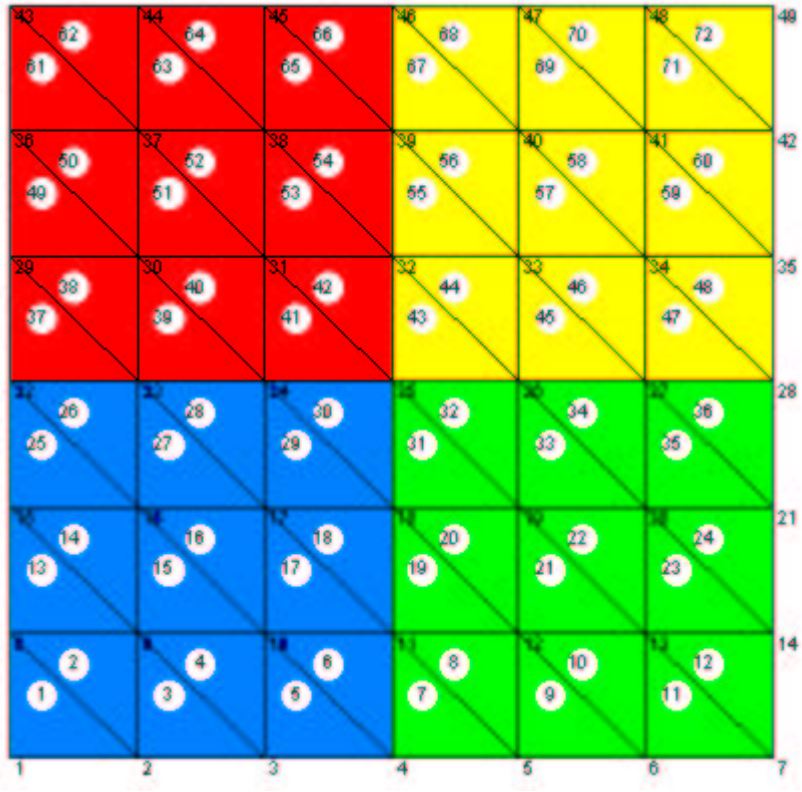
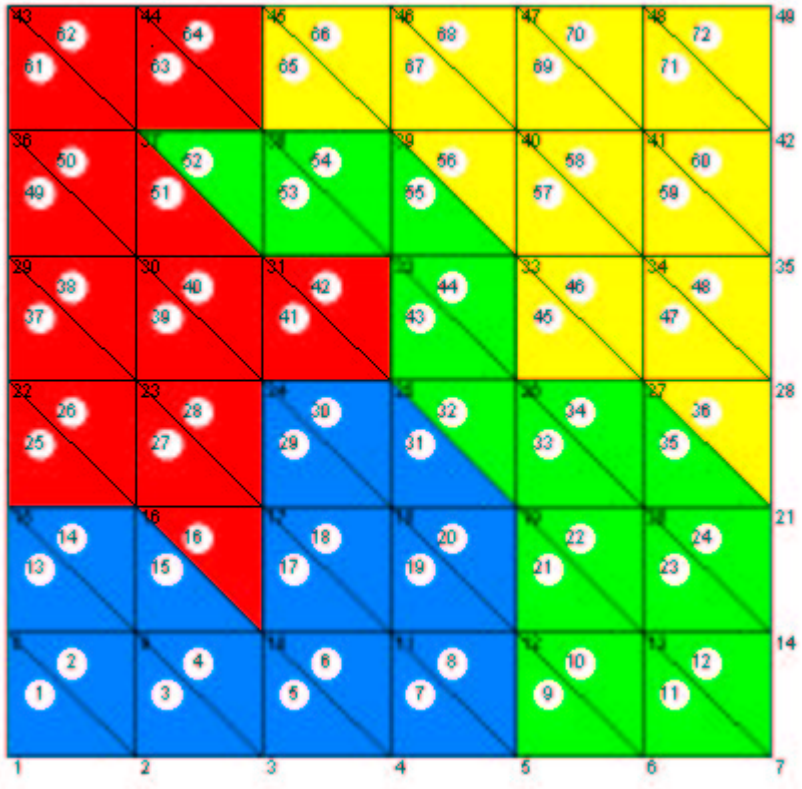


Figura 3.1: Malha exemplo

Fazendo uma análise mais criteriosa nos dois particionamentos apresentados, nota-se que na Fig. 3.2(a) a fronteira de comunicação tem um total de 9 nós (vale lembrar que os nós ao redor da malha foram considerados nós de valor prescrito, e nesse caso, não estão envolvidos no processo de comunicação), mas o nó 25 por exemplo está sendo compartilhado pelos 4 processadores. Já na Fig. 3.2(b) o número de nós na fronteira de comunicação é de 15 ao todo, contudo, não existe nenhum nó que seja compartilhado por 4 processadores. Em geral, é melhor optar por um particionamento com um número maior de nós na fronteira de comunicação do que escolher um particionamento que minimize o número de nós nas fronteiras mas necessite compartilhar um mesmo nó com uma quantidade um pouco maior de processadores, pois de acordo com os valores de latência e taxa de transferência encontrados em [37], o tempo de envio da uma mensagem maior pode ser compensado quando o número de máquinas envolvidas na comunicação é menor, uma vez que o número de latências é reduzido. Executado o particionamento, o próximo passo é gerar os arquivos de dados correspondentes a cada subdomínio. Na seção a seguir é indicado como realizar tal processo.



(a) Particionamento trival



(b) Particionamento utilizando o METIS

Figura 3.2: Exemplos de particionamento

3.3 Pré-processamento paralelo

Os algoritmos paralelos foram executados utilizando o paradigma de troca de mensagens para uma arquitetura de memória distribuída, mais precisamente em cluster, que trabalha com os serviços autenticação de usuários e gerenciamento de arquivos, denominados NIS e NFS [40], respectivamente. Esses serviços são realizados em uma máquina servidora que autentica cada usuário e efetua quando solicitado leitura e gravação de dados. Na execução de um programa em paralelo todas as máquinas envolvidas no processo acessam a máquina servidora para escrita ou simplesmente para leitura. Portanto, antes de iniciar a execução de um algoritmo em paralelo, é interessante fazer um pré-processamento que separe os dados a serem processados em número de arquivos igual ao número de processadores envolvidos na paralelização. Sendo que cada arquivo deve conter os dados específicos que cada processador deve conhecer, isso para evitar uma sobrecarga no NFS da máquina servidora ou mesmo excessivas trocas de mensagens entre as máquinas no início do processo.

O pré-processamento paralelo consiste, inicialmente, da leitura de um arquivo contendo as características geométricas da malha assim como uma seqüência de números inteiros com valores entre 1 e p indicando a qual processador pertence cada elemento, conforme lista fornecida pelo particionamento dual do METIS. Em seguida, o pré-processamento deve gerar como saída p arquivos que contenham os dados geométricos das p submalhas referentes as p partições. Considerando a malha exemplo na Fig. 3.1, é associado um arquivo `Mesh.dat`(Fig. 3.3) que contém todos os dados refentes a geometria da malha, tais como, o total de nós da malha, o total de elementos da malha, uma lista com o número correspondente a cada nó (caso esse valor seja negativo, o nó em questão trata-se de um nó de valor prescrito), as coordenadas x e y de cada nó, e uma lista com a conectividade desses elementos. Além disso, é indicado a qual processador pertence cada elemento.

Como na Fig. 3.2(b) a malha foi particionada em 4 submalhas, o arquivo `Mesh.dat` necessita ser também particionado em quatro outros arquivos para as respectivas submalhas. Na Fig. 3.4 é apresentado o arquivo de entrada de dados `Data03.in` referente aos dados da submalha que será processada no processador 3. Este arquivo contém informações sobre o número de nós e suas coordenadas x e y, uma especificação para o tipo de cada um dos nós, o número de elementos e suas devidas conectividades e, por fim, p listas de possíveis nós de fronteira de

```
nodes=49
elements=72
node coord x coord y
-1 -5.000000 -5.000000
-2 -3.333333 -5.000000
-3 -1.666667 -5.000000
...  ...  ...
10 -1.666667 -3.333333
11 -0.000000 -3.333333
12 1.666667 -3.333333
13 3.333333 -3.333333
...  ...  ...
-47 1.666667 5.000000
-48 3.333333 5.000000
-49 5.000000 5.000000
node1 node2 node3 rank
1 2 8          2
2 9 8          2
2 3 9          2
...  ...  ...      ...
22 23 29       3
23 30 29       1
23 24 30       1
...  ...  ...      ...
41 48 47       3
41 42 48       3
42 49 48       3
```

Figura 3.3: Descrição do arquivo `Mesh.dat`

comunicação de um dado processador. Note que cada elemento do arquivo teve sua conectividade mapeada a partir de uma numeração de nós locais e não na numeração global, uma vez que não faria sentido para o processador 3 conhecer os dados de outros processadores. Essa submalha é numerada independemente das demais. O primeiro elemento do processador 3, por exemplo, tem convectividade 1, 2 e 3, mas a numeração global desse elemento seria 21, 28 e 27. Da mesma forma, os nós da fronteiras de comunicação receberam uma numeração local. Repare que a fronteira com o processador 4 é composta pelos nós 4, 3, 5, 12 e 9, mas pode-se perceber na Fig.3.2(b) que a numeração global seria 26, 27, 33, 38 e 39, respectivamente.

Em termos gerais, um algoritmo para realizar o particionamento do arquivo de dados da malha pode ser encontrado em [20]. Esse tipo de algoritmo deve ler o arquivo `Mesh.dat` contendo as características geométricas da malha e escolhido um número p de partições, ele deve ter como saída p arquivos que contenham os dados geométricos das p submalhas referentes as p partições. Os passos principais para executar esse algoritmo são:

1. Abrir o arquivo contendo as características geométricas da malha;
2. Ler a quantidade de nós da malha;
3. Ler a quantidade de elementos da malha;
4. Ler, em uma estrutura adequada, as coordenadas x e y de cada nó;
5. Ler a conectividade da malha (quais nós representam cada elemento);
6. Verificar quais elementos da malha pertencem a quais processadores e renumerar esses elementos localmente em cada processador (basta utilizar a lista fornecida pelo particionamento dual do METIS);
7. Verificar quais nós da malha pertencem a quais processadores e renumerar esses nós localmente, classificando os nós como nós de valor prescrito, com tipo igual a -1, ou nós incógnitas com valor maior ou a igual zero (essa verificação deve ser feita dentro do conjunto dos elementos já divididos em seus respectivos processadores);
8. Escrever p arquivos contendo as informações referentes a cada uma das submalhas: número de nós, número de elementos, número do nó e suas coordenadas

```
nodes=17
elements=18
node coord x coord y type global-node
1 1.000000 0.333333 0 21
2 1.000000 0.500000 0 28
3 0.833333 0.500000 2 27
4 0.666667 0.500000 2 26
5 0.666667 0.666667 2 33
6 0.833333 0.666667 1 34
...  ...  ...  ...  ...
9 0.500000 0.833333 2 39
...  ...  ...  ...  ...
12 0.333333 0.833333 3 38
...  ...  ...  ...  ...
16 0.833333 1.000000 0 48
17 1.000000 1.000000 0 49
submesh-conectivity
1 2 3
4 3 5
3 6 5
3 2 6
...  ...  ...
8 10 15
10 16 15
10 11 16
11 17 16
rank1-boundary
1
12
rank2-boundary
0
rank3-boundary
0
rank4-boundary
5
4 3 5 12 9
```

Figura 3.4: Descrição do arquivo `Data03.in`

x e y, tipo do nó (um número de 0 a $p-1$) representa quantos processadores compartilham determinado nó, numeração global do nó, a conectividade da submalha e p listas contendo os nós vizinhos de cada processador.

Na seção seguinte será discutido como as informações descritas nos arquivos do pré-processamento serão utilizadas, ou seja, será apresentada uma estrutura de dados paralela para armazenar tais informações e possibilitar a realização da etapa efetiva do processamento paralelo.

3.4 Estrutura de dados paralela

Para resolver o sistema linear (2.64) foi utilizada uma técnica especial de paralelização, descrita em [20][31][35], que utiliza blocos orientados para representar o sistema linear a partir de uma divisão especial da estrutura de dados da malha. Neste processo o domínio Ω, particionado em p subdomínios, é composto por três tipos de nós: *IntNodes*, *IBNodes* e 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, e os nós de valor prescrito são aqueles onde a solução é conhecida. Como exemplo, é representa na Fig. 3.5 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* dos processadores 1, 3 e 4.

Utilizando a distribuição dos nós como descrita anteriormente, é possível verificar que um sistema linear resultante da discretização por elementos finitos escrito na forma

$$Ax = b \tag{3.1}$$

pode ser reorganizado da seguinte maneira:

$$Ax = \begin{bmatrix} A_1 & & & & B_1 \\ & A_2 & & & B_2 \\ & & \ddots & & \vdots \\ & & & A_p & B_p \\ C_1 & C_2 & \dots & C_p & A_s \end{bmatrix} \begin{bmatrix} \underline{x}_1 \\ \underline{x}_2 \\ \vdots \\ \underline{x}_p \\ \underline{x}_s \end{bmatrix} = \begin{bmatrix} \underline{b}_1 \\ \underline{b}_2 \\ \vdots \\ \underline{b}_p \\ \underline{b}_s \end{bmatrix} = b \tag{3.2}$$

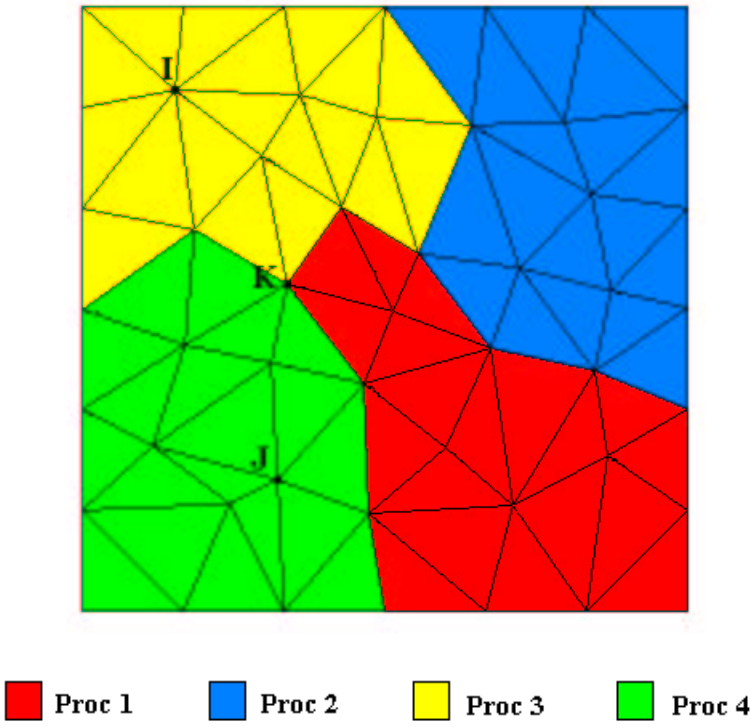


Figura 3.5: Exemplo de malha particionada em 4 subdomínios

onde os blocos de matrizes A_i, B_i, C_i e A_s, com $i = 1, ..., p$, armazenam as contribuições dos elementos referentes a cada tipo de nós como é detalhado a seguir. 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 armazemam 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 é 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 de vários outros 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.

Observe que, utilizando esta estratégia, é possível montar as contribuições da matriz A e do vetor b independentemente e concorrentemente em cada processador.

Além disso, no caso do sistema linear de interesse (2.64), basta relacionar A com a matriz M^*, o vetor b com o vetor R e a solução x com o vetor Δa. No entanto, com o objetivo de facilitar e simplificar o entendimento a notação utilizada em (3.1) e (3.2) será mantida na próxima subseção. Com a finalidade de ilustrar como são obtidos os blocos A_i, B_i, C_i, $A_{s(i)}$ e demais estruturas no processo de paralelização, a estrutura de dados paralela (3.2) é aplicada a um problema exemplo na próxima subseção.

3.4.1 A estrutura de dados aplicada a um problema exemplo

O problema exemplo considerado é uma particularização da equação de advecção e difusão descrita em (2.5). É um problema com solução em regime permanente, matriz de difusividade volumétrica $\boldsymbol{k}$ igual a matriz identidade, campo de velocidade $\boldsymbol{\beta} = \{0, 0\}^T$ e termo de fonte $f = 0$:

$$\begin{cases} -\nabla^2 u(x,y) = 0 & (x,y) \in \Omega \subset \mathbb{R}^2 \\ \\ u(x,y) = 10 & (x,y) \in \partial\Omega \end{cases} \tag{3.3}$$

Conforme discutido no Capítulo 2, o problema de valor de contorno (5.5) passa pelas etapas de solução pelo método dos elementos finitos e aplicadas as condições de contorno, chega-se a um sistema linear a ser resolvido:

$$\boldsymbol{K}\boldsymbol{u} = \boldsymbol{F} \tag{3.4}$$

A matriz $\boldsymbol{K}$ pode ser construída a partir das contribuições dos elementos, sendo conveniente identificar seus termos em nível de cada elemento:

$$\begin{aligned} \boldsymbol{K} &= \mathop{\mathsf{A}}_{e=1}^{n_{el}} (\boldsymbol{k}^e) \\ \boldsymbol{k}^e &= \int_{\Omega_e} \boldsymbol{B}^T \boldsymbol{B} d\Omega \end{aligned} \tag{3.5}$$

onde A é um operador de montagem e $\boldsymbol{B}$ é a mesma matriz definida em (2.35). Aqui também é adotada a notação descrita em (3.1), ou seja, desse ponto em diante, $\boldsymbol{K}$, $\boldsymbol{u}$ e $\boldsymbol{F}$ de (3.4) passam a ser representados como A, x e b de (3.2), respectivamente.

Como visto no capítulo anterior, o próximo passo é gerar uma malha para o método dos elementos finitos. É considerada a mesma malha da seção anterior

(Fig. 3.2(b)), ou seja, uma malha com 49 nós e 72 elementos, particionada em 4 subdomínios utilizando o particionador METIS[23]. Todos os nós ao redor da malha foram tomados como nós de valor prescrito.

Inicialmente, é feita uma verificação na classificação dos nós da malha exemplo e, em seguida, é empregada a estrutura (3.2) para a obtenção dos blocos A_i, B_i, C_i e $A_{s(i)}$, b_i e $b_{s(i)}$, com $i = \{1, 2, 3, 4\}$.

- **Nós de valor prescrito** – São os nós cujos valores da equação são conhecidos. Tais pontos formam as condições de contorno do problema. No exemplo, são os nós ao redor da malha.
- **IntNodes** – São nós incógnitas que pertencem apenas a um determinado processador. Os nós 9, 11 e 18, por exemplo, são nós do tipo *IntNodes* do processador 2.
- **IBNodes** – São nós incógnitas que pertencem a dois ou mais processadores. O nó 25 é um exemplo de *IBNodes* dos processadores 1, 2 e 4.

Dessa forma, têm-se os componentes do sistema (3.2) distribuídos por quatro processadores. Por conseguinte, as contribuições da matriz de discretização de (3.5) são dispostas nos processadores 1, 2, 3 e 4 conforme as Fig. 3.6, 3.7, 3.8 e 3.9, respectivamente.

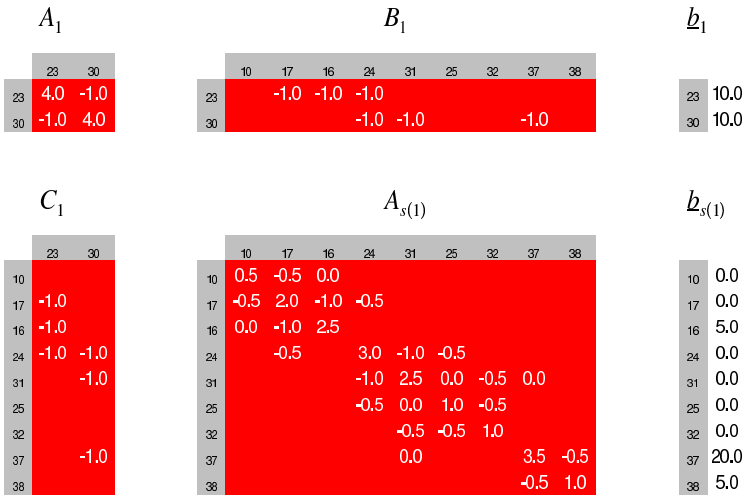


Figura 3.6: Contribuições no processador 1

Levando em consideração o sistema linear (3.2) e as contribuições dos quatro processadores descritas, conforme pode ser observado na Fig. 3.10, é possível mon-

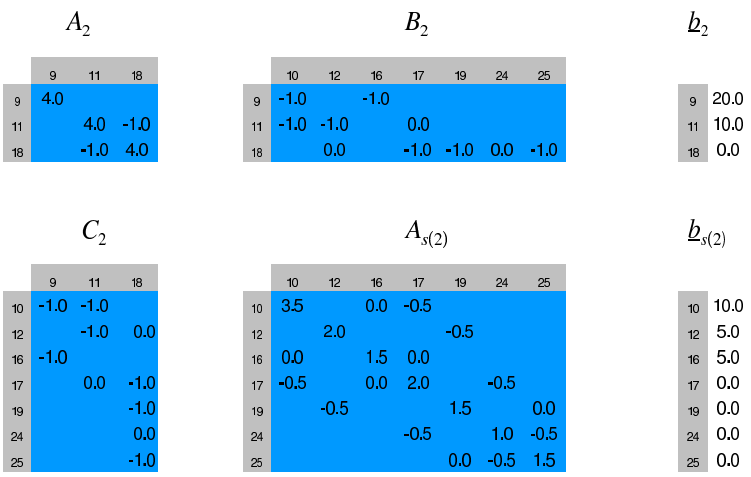


Figura 3.7: Contribuições no processador 2

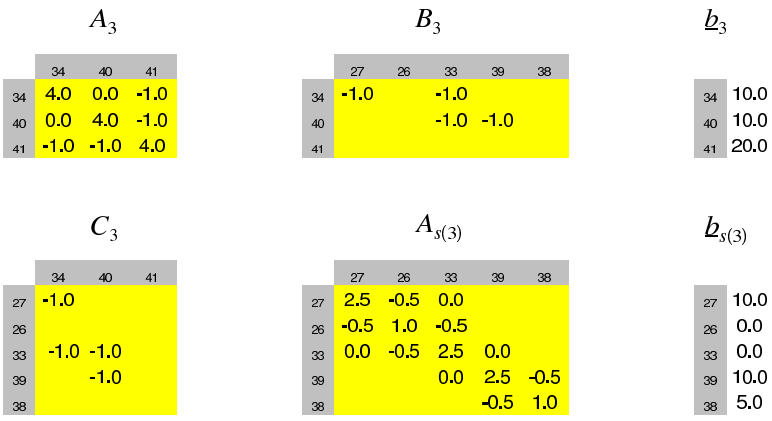


Figura 3.8: Contribuições no processador 3

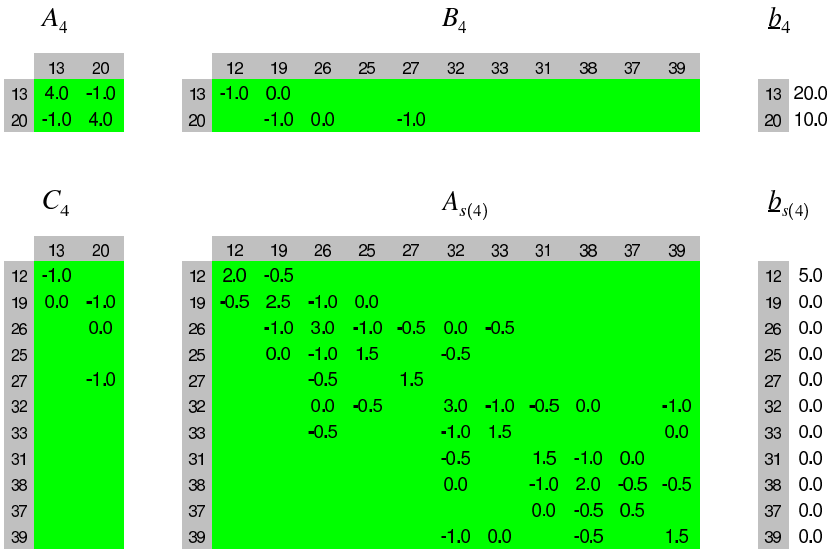


Figura 3.9: Contribuições no processador 4

tar as contribuições da matriz global A referente a malha exemplo (Fig. 3.2(b)). Tal montagem é aqui apresentada simplesmente para verificar as propriedades da estrutura paralela proposta em (3.2), sendo essa montagem desnecessária e mesmo inviável tratando-se do processamento paralelo propriamente dito. Pelo fato da equação (5.5), referente ao problema exemplo abordado, representar um problema bem simples e a malha exemplo (Fig. 3.2(b)) ser de dimensões bem menores daquelas que devem ser utilizadas na prática, algumas observações devem ser feitas.

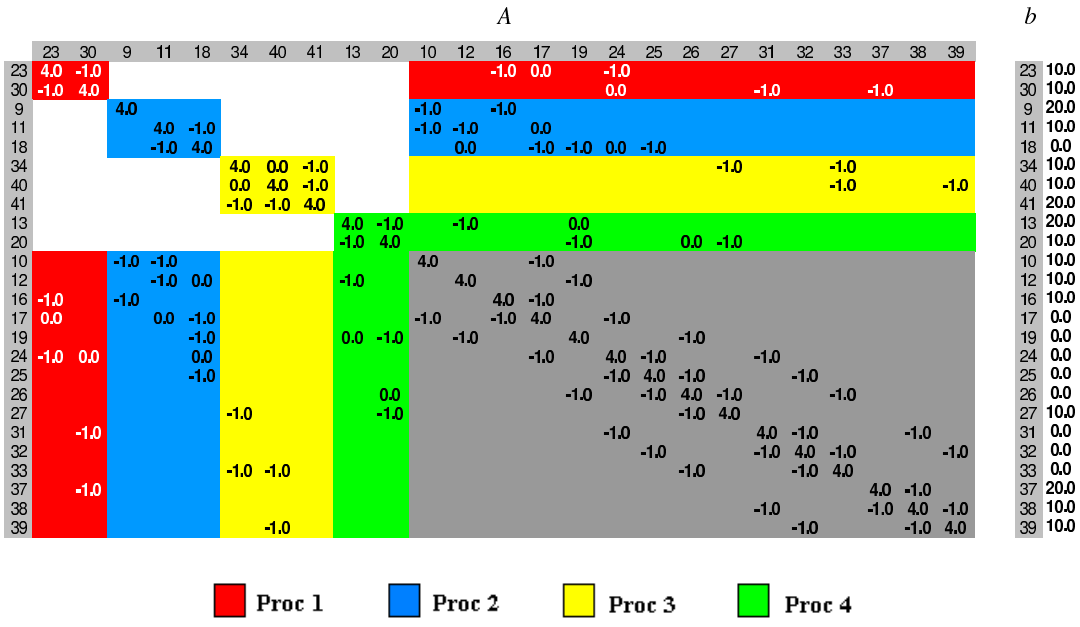


Figura 3.10: Montagem da matriz global

Primeiro, tanto as contribuições dos blocos A_i e $A_{s(i)}$, com $i = \{1, 2, 3, 4\}$, descritos nas Fig. 3.6, 3.7, 3.8 e 3.9, como as contribuições da matriz global A, descrita na Fig. 3.10, estão dispostas de forma simétrica, o que não acontece em problemas mais complexos, como por exemplo, a equação de advecção e difusão retratada em (2.5). Segundo, pode-se perceber que todas as estruturas visualizadas nas Fig. 3.6, 3.7, 3.8, 3.9 e 3.10 são esparsas, contudo, existem contribuições onde os zeros foram explicitados indicando que tais valores ocorreram devido às particularidades da malha e do problema de valor de contorno utilizado como exemplo, ou seja, para outros casos esse valores em destaque não serão necessariamente nulos. Por último, é importante observar a relação entre as dimensões dos blocos A_i, B_i, C_i e $A_{s(i)}$. No exemplo a estrutura $A_{s(i)}$ com $i = \{1, 2, 3, 4\}$ apresenta dimensões bem maiores que a dos outros blocos. Na prática, as malhas construídas são bem maiores que a

utilizada no exemplo, sendo o número de nós *IntNodes* muito maior que o número de nós *IBNodes*. Por conseguinte, a estrutura A_i dessas malhas é bem maior que as outras estruturas B_i, C_i e $A_{s(i)}$.

Como já mencionado no princípio deste capítulo, as particularidades da malha exemplo foram utilizadas apenas visando facilitar a compreensão. A partir desse exemplo é possível generalizar os procedimentos de forma a resolver problemas semelhantes para quaisquer malhas triangulares geradas para o método dos elementos finitos,adequando-as ao número de partições desejado.Todos os procedimentos discutidos podem ser estendidos para uma malha qualquer, seja ela estruturada ou não.

Após serem realizados todos os passos relativos ao pré-processamento paralelo e apresentada a estrutura de dados que será utilizada na paralelização, começa o processamento paralelo propriamente dito. Sendo assim, na próxima seção serão apresentados os aspectos que dizem respeito as modificações que ocorreram na passagem de um código serial para um código paralelo, sem o detalhamento dos métodos numéricos em si. É de interesse um estudo mais detalhado sobre as operações de troca de mensagem e os trechos do algoritmo que têm grande impacto sobre o processamento como um todo.

3.5 Algoritmo paralelo

As trocas de mensagem são realizadas utilizando o padrão MPI, mais precisamente a implementação lam-mpi 6.5.9-tcp.1. Informações sobre as rotinas MPI utilizadas, bem como, detalhes sobre o próprio padrão MPI podem ser vistos em [29]. Os algoritmos são apresentados através de fluxogramas com características especiais. A cor cinza indica o trecho do algoritmo envolvido no contexto de troca de mensagens e sincronização. As bordas em destaque representam subalgoritmos em que está concentrada a maior parte do processamento.

A Fig. 3.11 contém os passos principais do algoritmo paralelo. O passo 1 representa a utilização das subrotinas de inicialização MPI, tais como MPI_Init, MPI_Comm_Size e MPI_Comm_Rank. O passo 2 representa a etapa onde é feita a leitura de dados dos arquivos que foram pré-processados conforme descrito na seção

3.3. Conhecidas as informações da malha no passo 2, no passo 3 é feita a classificação dos nós em 3 categorias: nós incógnitas do tipo *IntNodes*, nós incógnita do tipo *IBNodes* e nós de valor prescrito. Em seguida, é criado uma estrutura onde são postas as listas de nós vizinhos de cada processador. No passo 4 são calculados os coeficientes dos blocos orientados A_i, B_i, C_i e $A_{s(i)}$ e do vetor de termos independentes. O passo 5, definido como algoritmo preditor-multicorretor, é o principal passo do algoritmo. Ele é encarregado por quase todo o tempo de execução do algoritmo, ou seja, algo em torno de 99,9% do tempo total de execução. No passo 6 são escritos os arquivos de saída de dados contendo informações sobre o valores obtidos para as incógnitas nodais. Para finalizar o algoritmo paralelo é chamada a subrotina MPI_Finalize, fechando assim o passo 7.

O algoritmo preditor-multicorretor paralelo é analisado mais detalhadamente na próxima subseção por ser o trecho principal do algoritmo descrito na Fig. 3.11. Semelhantemente, o objetivo é destacar as operações envolvendo troca de mensagens e tempo de processamento. Nessa subseção será utilizada a mesma notação da seção 2.5, onde são apresentados os passos do algoritmo preditor-multicorretor.

3.5.1 Algoritmo preditor-multicorretor paralelo

Na Fig. 3.12 há uma descrição do algoritmo preditor-multicorretor paralelo. Inicialmente é feita uma atualização no vetor de termos independentes $\boldsymbol{F}$ através de uma função denominada *Update*, ou seja, a parte $\boldsymbol{F}_{s(i)}$ de cada processador envia e recebe as contribuições espalhadas pelos processadores vizinhos. No passo 5.2 são aplicadas as condições iniciais do problema discretizado por elementos finitos. A partir do passo 5.3 o algoritmo preditor-multicorretor passa a ser efetivamente utilizado. Primeiro a fase de predição conforme (2.59), (2.60) e (2.61), em seguida, a fase de correção conforme (2.62), (2.63), (2.64), (2.65) e (2.66). O cálculo do vetor de resíduos $\boldsymbol{R}$ de (2.63) apresentado no passo 5.8 é precedido por quatro passos auxiliares: o produto matriz-vetor de $\boldsymbol{M}$ por $\boldsymbol{a}$ de (2.24) no passo 5.4, atualização da parte relativa os nós *IBNodes* do produto 5.4, outro produto matriz vetor, agora relativo a matriz $\boldsymbol{K}$ e ao vetor $\boldsymbol{u}$ também de (2.24) seguido de uma atualização refente os nós *IBNodes* do produto 5.6. No passo 5.9 é realizada a solução do sistema linear através do método GMRES. Os passos 5.10, 5.11, 5.12 e 5.13 foram descritos simplesmente como operações de produto interno, contudo a finalidade desses

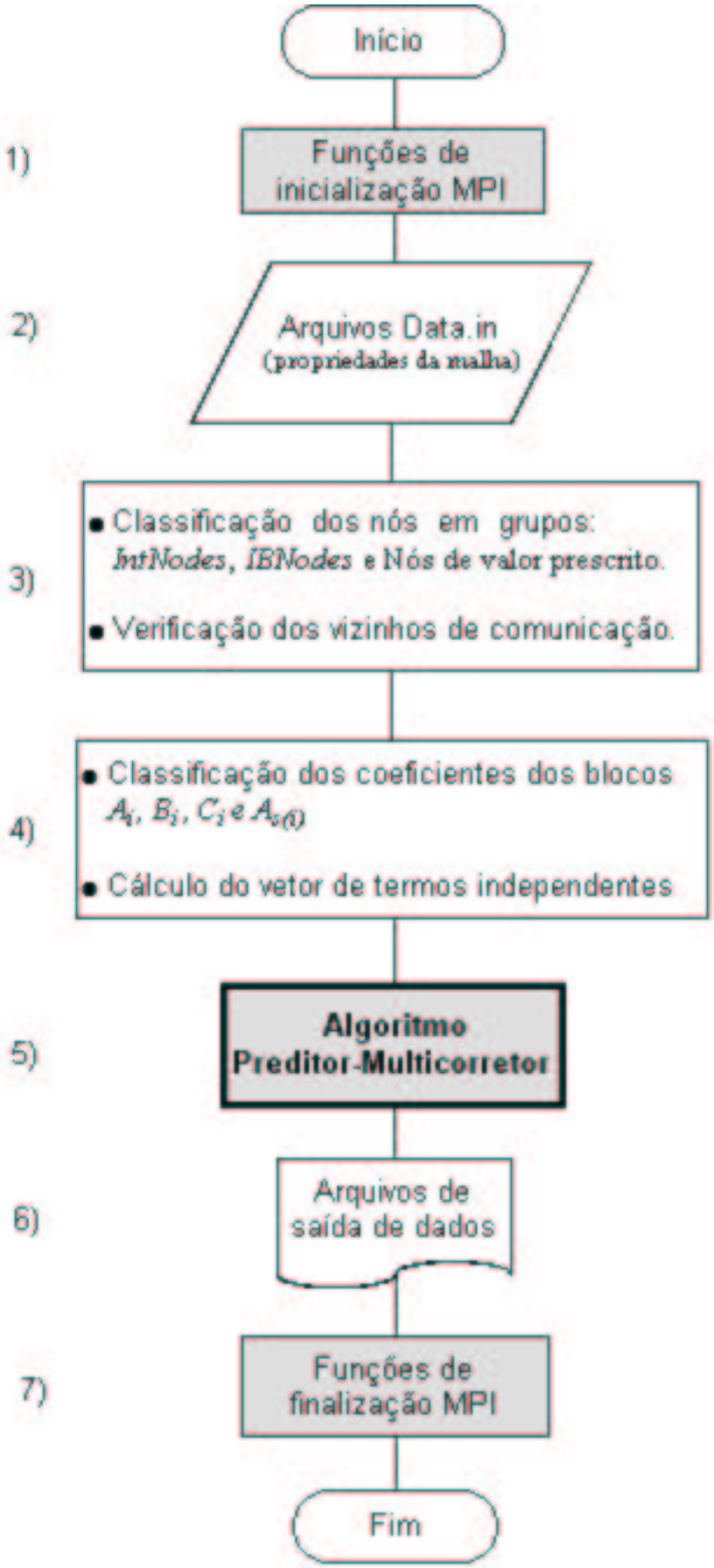


Figura 3.11: Algoritmo principal

é calcular as normas relativas ao seus respectivos vetores. Os passos 5.10 e 5.11 calculam as normas dos vetores $\boldsymbol{a}$ e $\boldsymbol{\Delta a}$ para verificação do teste de convergência entre os passos 5.11 e 5.12. Já a norma do vetor diferença entre o vetor de solução $\boldsymbol{u}$ e o vetor $\boldsymbol{u}$ da iteração anterior no tempo, denotada por $\boldsymbol{u} - \boldsymbol{u}_{old}$, e a norma do próprio vetor $\boldsymbol{u}$ são calculadas nos passos 5.12 e 5.13. O passo 5.14 é encarregado do incremento Δt de avanço no tempo. O algoritmo termina quanto o tempo t for igual ao tempo final t_f.

Na subseção seguinte o algoritmo GMRES é apresentado enfatizando os trechos envolvendo trocas de mensagens e operações essenciais como o produto matriz-vetor e o produto interno.

3.5.2 Algoritmo GMRES paralelo

O algoritmo GMRES paralelo é apresentado na Fig. 3.13. Assim como no algoritmo preditor-multicorretor, as operações de produto interno presentes nos passos 5.9.1, 5.9.4 e 5.9.7 são realizadas para calcular as normas do vetor de termos independentes e dos vetores da base de *Krylov*. Os passos 5.9.2 e 5.9.5 envolvem a operação mais custosa do algoritmo GMRES, o produto matriz-vetor. Nos passos 5.9.3 e 5.9.6 são realizadas as atualizações que sucedem cada produto matriz-vetor.

O produto matriz-vetor e o produto interno são as principais operações envolvidas na solução de sistemas lineares através de métodos iterativos não-estacionários, e portanto, estão relacionados diretamente com o desempenho do processo como um todo. Sendo assim, nas próximas subseções serão dercritos em detalhes estas operações quando é utilizada a estrutura paralela proposta em (3.2).

3.5.3 Produto interno

O produto interno utiliza todos os componentes de um dado vetor para computar um simples número de ponto flutuante que deverá ser conhecido por todos os processadores. Portanto o cálculo desse produto requer uma comunicação global. Por sua vez, o cálculo do produto matriz-vetor pode não causar necessariamente uma perda de desempenho significante, como resultado do custo da comunicação no processamento paralelo, como será visto na próxima subseção. Para elaborar essa

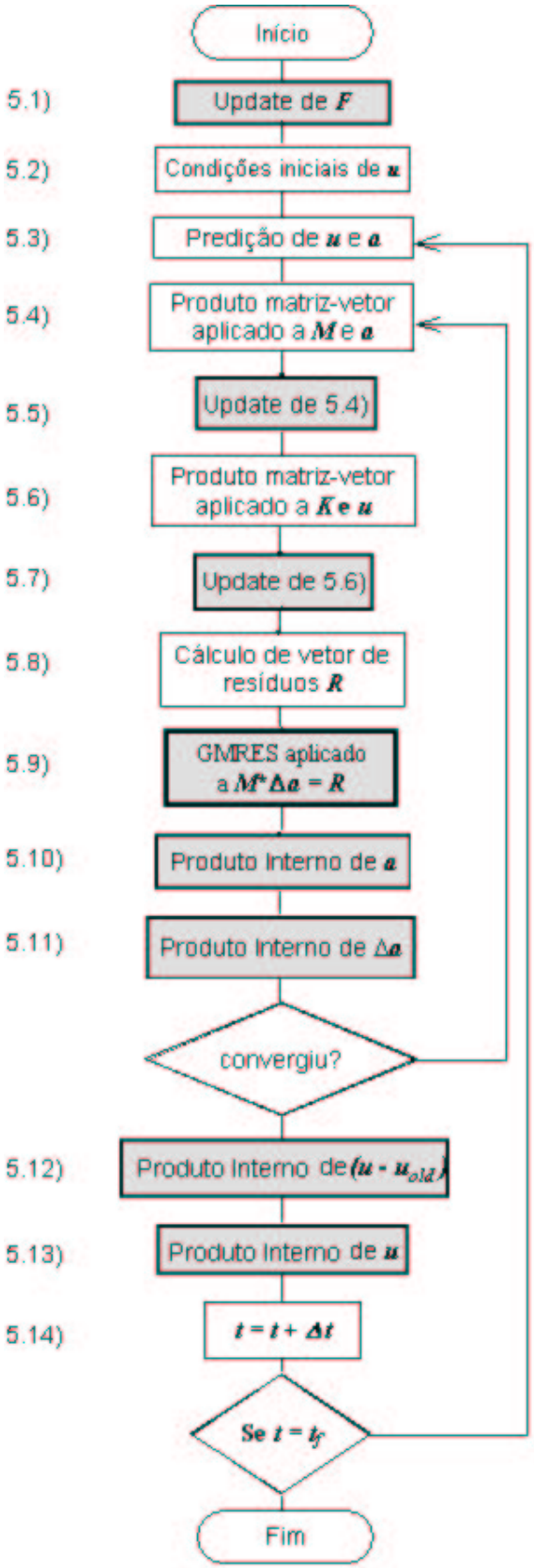


Figura 3.12: Algoritmo preditor-multicorretor

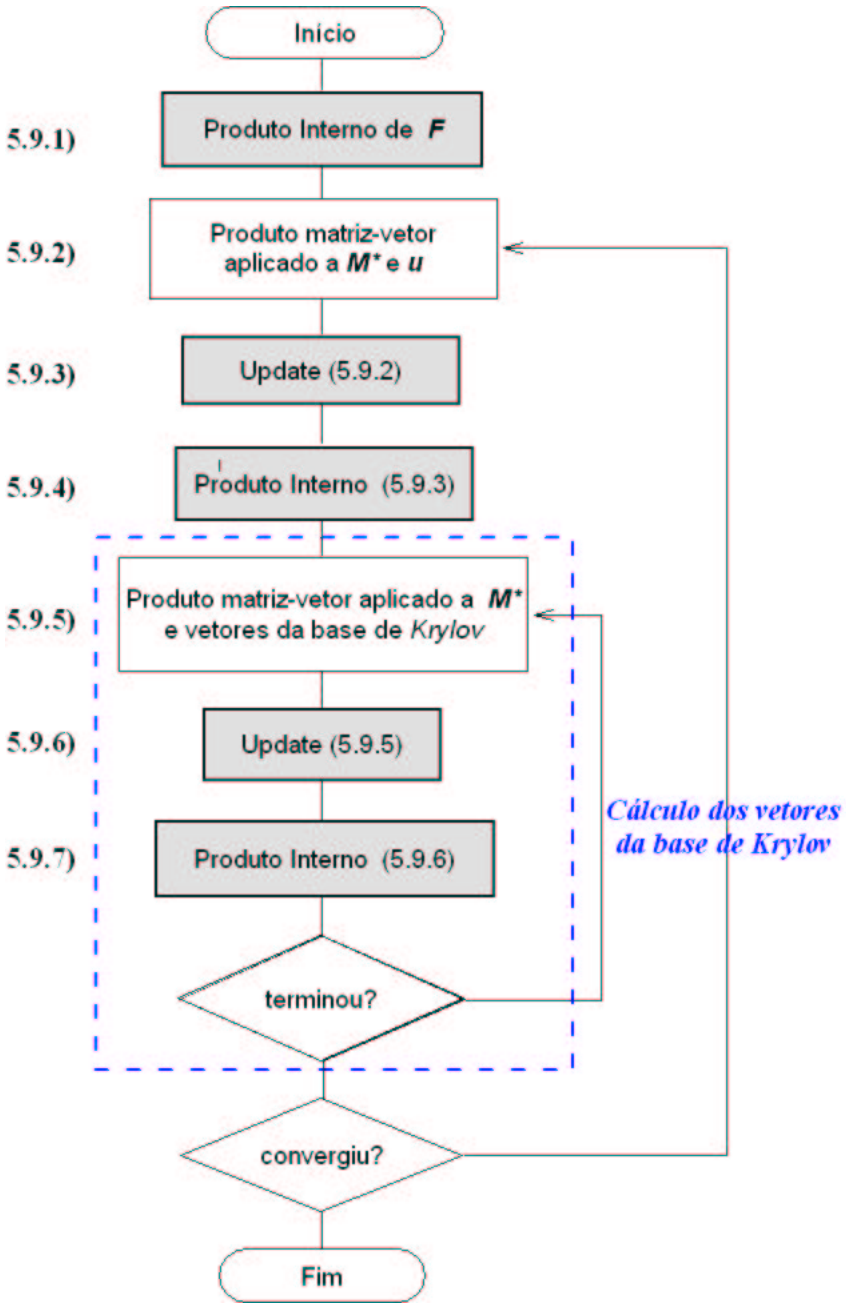


Figura 3.13: Algoritmo GMRES

comunicação global para o produto interno distribuído, considere dois vetores $\underline{u}$ e $\underline{v}$ dispostos de forma distribuída ao longo de p processadores

$$\underline{u} = \begin{bmatrix} \underline{u}_1 \\ \underline{u}_2 \\ \vdots \\ \underline{u}_p \\ \underline{u}_s \end{bmatrix} \tag{3.6}$$

e

$$\underline{v} = \begin{bmatrix} \underline{v}_1 \\ \underline{v}_2 \\ \vdots \\ \underline{v}_p \\ \underline{v}_s \end{bmatrix} \tag{3.7}$$

onde

$$\underline{u}_s = \mathop{\bigcup}_{i=1}^{p} \underline{u}_{s_{(i)}} \tag{3.8}$$

e

$$\underline{v}_s = \mathop{\bigcup}_{i=1}^{p} \underline{v}_{s_{(i)}} \tag{3.9}$$

Os vetores $\underline{u}_i$ e $\underline{v}_i$ pertencem ao processador i que também armazena $\underline{u}_{s_{(i)}}$ e $\underline{v}_{s_{(i)}}$, parte referentes aos nós do contorno de comunicação. Sendo assim, o produto interno pode ser expresso por

$$\underline{u} \cdot \underline{v} = \sum_{i=1}^{p} (\underline{u}_i \cdot \underline{v}_i + \underline{u}_{s_{(i)}} \cdot \underline{v}_{s_{(i)}}) \tag{3.10}$$

De acordo com [20], o produto interno paralelo pode ser calculado em cada processador conforme o algoritmo apresentado na Fig. 3.14. O algoritmo é composto por 5 passos essenciais. O primeiro passo inicializa a variável γ utilizada no cálculo do produto interno de cada processador. No passo 2 são calculadas as contribuições referentes aos nós *IntNodes*. No passo 3 são calculadas as contribuições referentes aos nós *IBNodes*. O termo $Shared[i]$ representa o número de processadores que estão compartilhando o nó *IBNodes* envolvido no processo (No exemplo ilustrado na Fig. 3.2(b), o $Shared[i]$ do nó 25 é igual a 3). No passo 4 é realizada uma operação de redução global através da primitiva $MPI\_AllReduce$. Todos os γ's dos processadores são somados e então é conhecido o produto interno global. O último passo simplesmente retorna esse valor global calculado.

$ProdInt(\underline{u}, \underline{u}_s; \underline{v}, \underline{v}_s)$

1. $\gamma = 0$
2. $Para\ i = 1, 2, ..., IntNodes$
   2.1. $\gamma = \gamma + \underline{u}[i] \cdot \underline{v}[i]$
3. $Para\ i = 1, 2, ..., IBNodes$
   3.1. $\gamma = \gamma + \dfrac{\underline{u}_s[i] \cdot \underline{v}_s[i]}{Shared[i]}$
4. $AllReduce(\gamma, sum, MPI\_SUM)$
5. $Retorne\ (sum)$

Figura 3.14: Função *ProdInt*: realiza o produto interno em cada processador

3.5.4 Produto matriz-vetor

Levando-se em consideração os blocos orientados definidos em (3.2) o produto matriz-vetor paralelo escrito na forma

$$Au = \begin{bmatrix} A_1 & & & & B_1 \\ & A_2 & & & B_2 \\ & & \ddots & & \vdots \\ & & & A_p & B_p \\ C_1 & C_2 & \dots & C_p & A_s \end{bmatrix} \begin{bmatrix} \underline{u}_1 \\ \underline{u}_2 \\ \vdots \\ \underline{u}_p \\ \underline{u}_s \end{bmatrix} = \begin{bmatrix} \underline{v}_1 \\ \underline{v}_2 \\ \vdots \\ \underline{v}_p \\ \underline{v}_s \end{bmatrix} = v \tag{3.11}$$

pode ser calculado como

$$\underline{v}_i = A_i \underline{u}_i + B_i \underline{u}_{s_{(i)}} \tag{3.12}$$

no processador i, para $i = 1, ..., p$ e

$$\underline{v}_{s_{(i)}} = C_i \underline{u}_i + A_{s_{(i)}} \underline{u}_{s_{(i)}} \tag{3.13}$$

Para montar as estruturas A_s, $\underline{u}_s$ e $\underline{v}_s$ seria necessária uma comunicação global entre os processadores, uma vez que as partes que formam cada uma dessas estruturas estão distribuídas ao longo dos p processadores. Entretanto, aqui essas estruturas foram desenvolvidas aqui de forma a evitar tais comunicações, ou seja, $\underline{v}_i$ e $\underline{v}_{s_{(i)}}$ são armazenadas no processador i e $\underline{v}_s$ não é totalmente montado.

Contudo, pode-se notar que os vetores $\underline{v}_{s_{(i)}}$, com $i = 1, ..., p$, têm contribuições que precisam ser sincronizadas, cada vez que o produto matriz-vetor for realizado.

$Update(a)$

1. $Para\ i = 1, 2, ..., p$

   1.1. $Para\ j = 1, 2, .., Common[i]$

      1.1.1. $Buf[j] = a[Neighbour[i, j]]$

   1.2. $Se\ (Common[i] > 0)$

      1.2.1 $MPI\_Send(Buf, Common[i], i)$

2. $Para\ i = 1, 2, ..., p$

   2.1. $Se\ (Common[i] > 0)$

      2.1.1. $MPI\_Receive(Buf, Common[i], i)$

   2.2. $Para\ j = 1, 2, ..., Common[i]$

      2.2.1. $a[Neighbour[i, j]] = a[Neighbour[i, j]] + Buf[j]$

Figura 3.15: *Update*: responsável pelo atualização de dados em cada processador

Ou seja, as contribuições distribuidas ao longo dos nós nas fronteiras do particionamento devem ser acopladas adequadamente. Para isso, foi utilizada uma função denominada *Update* que é acionada após cada produto matriz-vetor.

A Fig. 3.15 contém a descrição da função *Update* que realiza as atualizações necessárias de dados entre os processadores vizinhos, ou seja, essa função é utilizada em alguns trechos do algoritmo paralelo que envolvem envio e recebimento de mensagens entre os processadores que compartilham nós na fronteira de particionamento. No algoritmo preditor-multicorretor paralelo descrito em 3.12, a função *Update* é utilizada nos passos 5.1, 5.5, 5.7 e, no algortimo GMRES paralelo descrito em 3.13, ela é utilizada nos passos 5.9.3 e 5.9.6. Tal função é constituída de duas primitivas MPI básicas: o MPI_Send e o MPI_Receive. As contribuições presentes nos nós *IBNodes* de cada processador i são enviadas somente para os processadores que têm fronteira de comunicação com o processador i e as contribuições dos processadores vizinhos são recebidas no processador i. A Tab. 3.1 descreve quem são os termos *Buf*, *Common* e *Neighbour* utilizados na Função *Update*.

Como os sistemas lineares oriundos da discretização de equações diferenciais pelo método dos elementos finitos são compostos por matrizes extremamente esparsas, foram utilizados dois métodos de solução de sistemas lineares tradicionais, mais

$Common[i]$	= número de nós compartilhados entre processador o i e o processador atual (para o processador atual, o valor $Common[i]$ é tomado sempre igual a zero).
$Neighbour[i, j]$	= número local do nó no contorno da partição que é o j-éssimo nó compartilhado com o processador i.
Buf	= vetor auxiliar utilizado para transferir dados

Tabela 3.1: As estruturas de dados referentes ao *Update*

precisamente dois métodos iterativos não-estacionários, Gradientes Conjugados e Método do Resíduo Mínimo Generalizado (GMRES). Infelizmente, a estrutura paralela proposta em (3.2) não aproveita de forma eficaz a propriedade fundamental desses métodos que é a possibilidade de trabalhar apenas com os coeficiente não-nulos visando um melhor desempenho no processo de solução.

No próximo capítulo serão estudas formas alternativas para sanar o problema do consumo de memória. Serão apresentadas três estratégias especiais de armazenamento, a saber, versões paralelas das estratégias elemento-por-elemento (EBE) [18], aresta-por-aresta (EDE) [7][28] e estratégia das linhas esparsas comprimidas (CSR) [35].

Capítulo 4

Estratégias de Armazenamento

Nesse capítulo é comentada a necessidade de se explorar a esparsidade da matriz resultante da discretização pelo método dos elementos finitos em paralelo. São abordadas algumas técnicas de armazenamentos mais eficientes do ponto de vista de consumo de memória. São apresentadas três estratégias de armazenamento aplicadas à estrutura de dado paralela proposta no capítulo anterior. Primeiramente, é abordado o armazenamento elemento-por-elemento, em seguida é apresentada a estratégia aresta-por-aresta e a terceira e última estratégia apresentada é a estratégia *Compressed Sparse Row*. É feita uma análise comparativa entre as estratégias apresentadas. Finalizando o capítulo é apresentada a complexidade do produto matriz vetor e produto interno em relação as estratégias de armazenamento estudadas.

4.1 Considerações iniciais

Nessa seção é feita uma análise prévia para ajudar a compreender a tamanha esparsidade existente nas estruturas de dados paralelas propostas em (3.2). Para isso, considere uma matriz A, obtida a partir da discretização por elementos finitos da forma sequencial tradicional, tem ordem $N_I \times N_I$, onde N_I é o número de nós incógnitas. Se forem utilizados nesse processo elementos triangulares lineares, a matriz A tem em média 7 coeficientes não nulos por linha, uma vez que ao redor de um dado nó incógnita tem-se, também em média, seis elementos triangulares. Esses seis elementos contribuem em outros seis nós incógnitas além do próprio nó incógnita dado, o que totaliza as 7 contribuições por linha. A seguir é verificado que o caso em que há mais de seis elementos ao redor de um mesmo nó incógnita e o caso em

que há menos de seis elementos reduzem-se ao caso médio.

- **Mais de seis elementos ao redor de um mesmo nó:** imagine que para um dado nó incógnita exista ao redor dele um número de elementos maior que seis. Como os elementos são triangulares, a soma dos ângulos internos de cada elemento é 180° e a soma dos ângulos ao redor do nó incógnita é sempre 360°. Sendo o número de elementos ao redor do nó maior que seis, em média, os ângulos internos ao redor do nó serão menores que 60°. Como resultado disso, os outros ângulos dos elementos serão em média maiores que 60°, o que levaria alguns dos outros nós incógnitas terem menos que seis elementos ao redor deles, uma vez que a soma dos ângulos internos ao redor dos nós incógnitas deve ser igual a 360°. Na Fig. 4.1, o nó central está cercado por 15 elementos e os nós internos ou redor do nó central têm valores bem menores que 60°. Isso faz com que os vários outros nós ao redor tenham uma menor quantidade de elementos ao redor de si mesmos, uma vez que a soma dos seus ângulos internos já está comprometida com altos valores.

- **Menos de seis elementos ao redor de um mesmo nó:** de maneira análoga verifica-se que se um nó incógnita tiver um número de elementos ao redor menor que seis, conseqüentemente os outros nós dos elementos ao redor necessitariam de um número de elementos maior que seis para que soma dos seus ângulos internos ao redor de cada nó seja compensada (Fig 4.2).

Portanto, uma matriz A gerada pela discretização pelo método dos elementos finitos é extremamente esparsa, pois, quando o número de incógnitas aumenta a quantidade de coeficientes não nulos por linha é quase insignificante. Na malha exemplo considerada no capítulo 3 (Fig. 3.1), existem apenas 25 nós incónitas, entretanto a matriz A associada a essa malha tem ordem 25×25 com apenas 7 coeficientes não nulos por linha, ou seja, de um total de 625 coeficientes somente 175 são não nulos.

Quando a matriz A é uma estrutura de dados paralela, conforme descrito em (3.2), essas considerações podem ser aplicadas aos blocos A_i, com $i = 1, .., p$. Entretanto, em relação aos blocos B_i, C_i e $A_{s(i)}$, com $i = 1, .., p$, algumas adequações devem ser efetuadas. A análise de esparsidade desses blocos deve adequar-se aos seus tipos de nós incógnitas, ou seja, nós *IntNodes* e *IBNodes*. A Tab. 4.1 apresenta

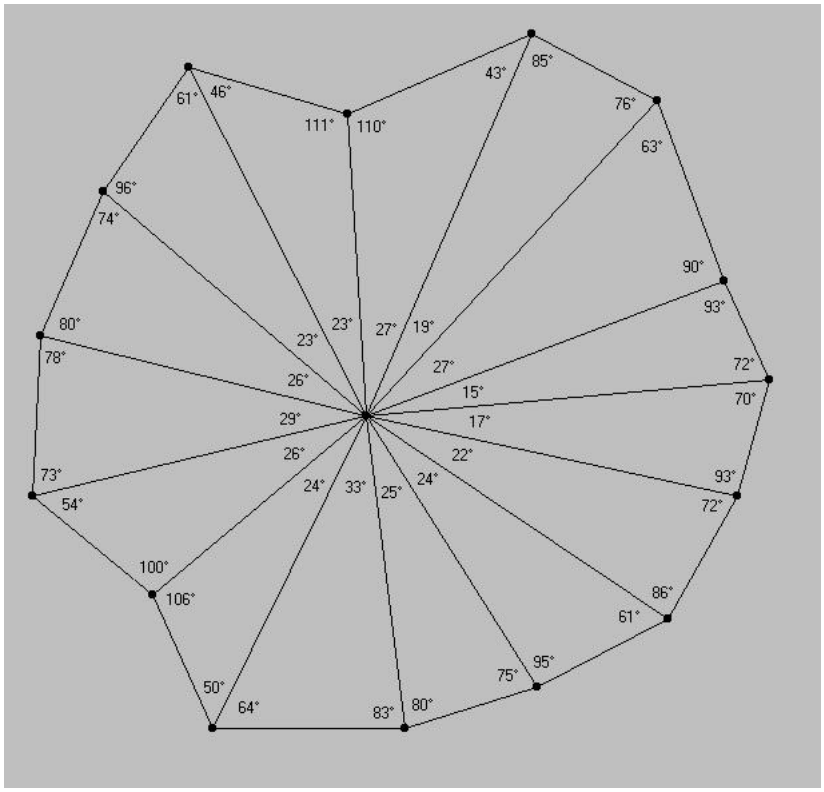


Figura 4.1: Mais que 6 elementos ao redor de um nó

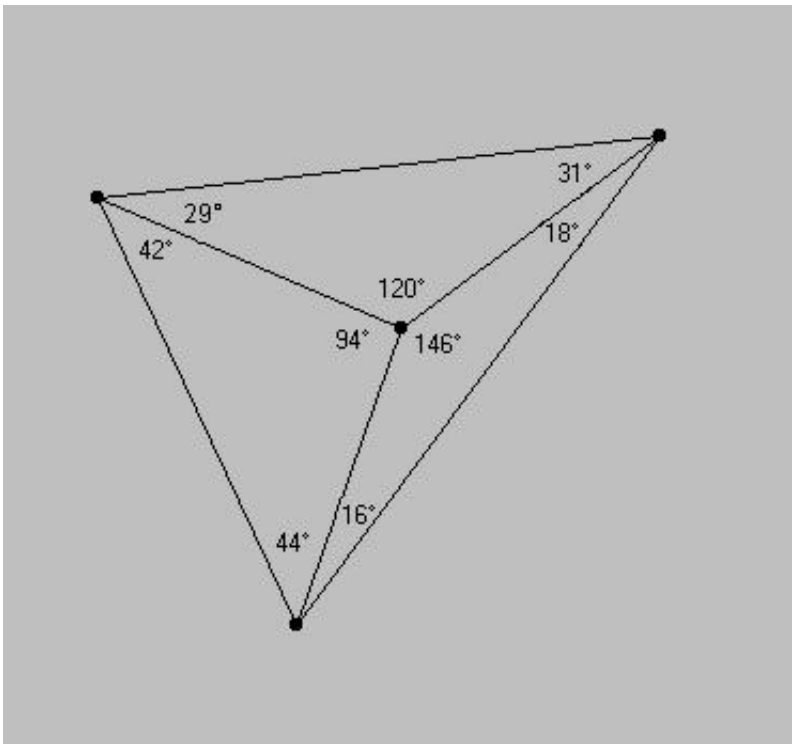


Figura 4.2: Menos que 6 elementos ao redor de um nó

as dimensões das estruturas A_i, B_i, C_i, $A_{s(i)}$, $\underline{x}_i$, $\underline{x}_{s(i)}$, $\underline{b}_i$ e $\underline{b}_{s(i)}$, com $i = \{1, 2, ..., p\}$, em função dos nós *IntNodes* e *IBNodes* de uma malha qualquer particionada para p processadores. O termo n_I representa o número de nós *IntNodes* em cada partição i e o termo n_B representa o número de nós *IBNodes* presentes na respectiva partição i. Note que cada bloco apresenta um ordem quadrática de termos, o que na prática representa um grande desperdício de memória, confirmando a esparsidade da estrutura paralela.

Estruturas	Dimensões
A_i	$n_I \times n_I$
B_i	$n_I \times n_B$
C_i	$n_B \times n_I$
$A_{s(i)}$	$n_B \times n_B$
$\underline{x}_i$ e $\underline{b}_i$	n_I e n_I
$\underline{x}_{s(i)}$ e $\underline{b}_{s(i)}$	n_B e n_B

Tabela 4.1: Dimensões das estruturas

Quando a ordem da matriz A é muito grande, os blocos A_i têm dimensões bem maiores que os outros blocos, e sua esparsidade é semelhante a da matriz A global, bastando apenas fazer analogia entre o termo N_I discutido no início do capítulo e o termo n_I que representa o número de nós incógnitas do tipo *IntNodes* do processador i. O bloco $A_{s(i)}$, por sua vez, tem sua esparsidade medida em função do termo n_B, onde n_B é o número de nós *IBNodes* da partição i. O bloco B_i contém um grande número de linhas inteiramente nulas, dado que para um malha de tamanho considerável, poucos são os nós *IntNodes* da partição i que estão associados com algum nó *IBNodes* de i. O bloco C_i quase não apresenta linhas inteiramente nulas, mas conforme pode ser verificado na Tab. 4.1 em relação ao número total de coeficientes em cada bloco, B_i e C_i apresentam rigorosamente a mesma quantidade, ou seja, $n_I \cdot n_B$ coeficientes para o bloco B_i e $n_B \cdot n_I$ para o bloco C_i. Tratando-se da esparsidade, ambos os blocos tem o mesmo número de coeficientes não nulos, pois para cada coeficiente de B_i obtido a partir das contribuições de um *IntNodes* em um *IBNodes* da partição i existe um coeficiente em C_i que foi obtido a partir das contribuições do mesmo *IBNodes* no mesmo *IntNodes*.

De acordo com a Tab. 4.1, observa-se que implementar tais estruturas sem utilizar alguma técnica especial de armazenamento de dados seria inviável, pois, não seria obtido o desempenho desejado, uma vez que a maior parte dos cálculos estaria sendo realizada desnecessariamente e muita memória estaria sendo consumida para armazenar apenas zeros. Sendo assim, nas próximas seções serão descritas as estratégias de armazenamento EBE, EDE e CSR.

4.2 Estratégia elemento-por-elemento - EBE

Na estratégia elemento-por-elemento tradicional, os coeficientes da matriz global A são armazenados em cada elemento [18]. 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 numero de nós incógnitas, é armazenada de forma mais compacta, ou seja, passaria a contar com *ebe* linhas e 9 colunas, onde *ebe* é o total 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 [18]. Assim, tem-se 6 colunas por linha ou invés de 9.

Na estratégia EBE, os blocos de matrizes A_i, B_i, C_i e $A_{s(i)}$ de (3.2), cujas dimensões são descritas na Tab. 4.1, são substituídos por estruturas correspondentes que têm suas dimensões relacionadas com quatro tipos de elementos: elementos ebe_{A_i}, elementos ebe_{B_i}, elementos ebe_{C_i} e elementos $ebe_{A_{s(i)}}$. Os elementos ebe_{A_i} são aqueles que armazenam suas contribuições em A_i de (3.2), da mesma forma que os elementos ebe_{B_i}, ebe_{C_i} e $ebe_{A_{s(i)}}$ são aqueles que armazenam suas contribuições em B_i, C_i e $A_{s(i)}$ de (3.2), respectivamente.

Dado um nó *IBNodes* pertencente a partição i, como ele está na fronteira entre duas partições, na maior parte dos casos, pode-se considerar que ao seu redor há em média 3 elementos da partição i. E portanto, esse nó está rodeado por elementos que têm 2 nós *IntNodes* e 2 *IBNodes* da partição i. Na Fig. 4.3 o nó *IBNodes* 20 da partição i está associado com os nós *IntNodes* 21 e 23 e nós *IBNodes* 8 e 19 também da partição i. A partir da mesma Fig. 4.3, algumas propriedades podem ser verificadas. A cada nó *IntNodes* da partição i associa-se em média 2 elementos ebe_{A_i} distintos de i. Isso significa que em média o número de elementos ebe_{A_i} é

2 vezes o número de *IntNodes*. A cada nó *IBNodes* de i associa-se em média, 2 elementos ebe_{B_i}, 2 elementos ebe_{C_i} e 2 elementos $ebe_{A_{s(i)}}$. Isso implica que em média o número de elementos dos tipos ebe_{B_i}, ebe_{C_i} e $ebe_{A_{s(i)}}$ é igual a 2 vezes o número de nós *IBNodes* de i. Vale lembrar que um mesmo elemento pode ser classificado em mais de uma categoria, dependendo dos tipos de nós pelos quais tal elemento é formado.

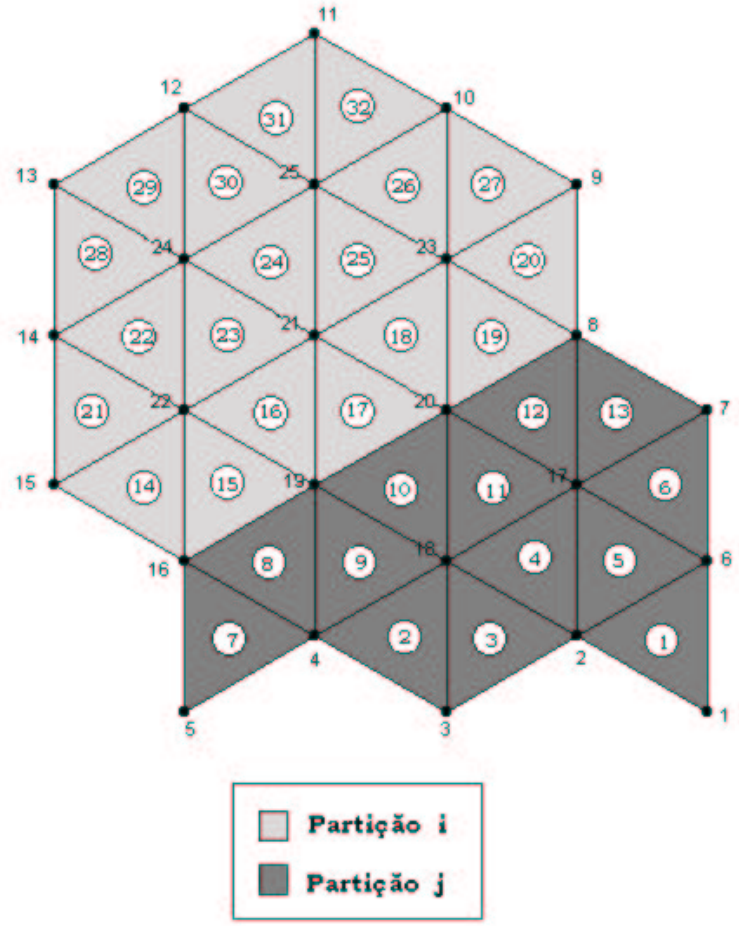


Figura 4.3: Fronteira entre as partições i e j

Foi discutido no Capítulo 2 que a matriz do elemento $(\boldsymbol{m}^*)^e$ de (2.72) não é montada, sendo necessário armazenar suas partes contidas nas matrizes $\boldsymbol{m}^e$ e $\boldsymbol{k}^e$ de (2.70) e (2.71), respectivamente. Observando o processo de discretização elemento-por-elemento, descrito também no Capítulo 2, nota-se que a matriz do elemento triangular linear $\boldsymbol{k}^e$ tem ordem 3×3, contudo, devido a propriedade de deficiência de posto dessa matriz, ao invés de armazenar os 9 coeficientes, armazena-se apenas 6, justamente aqueles que estão fora da diagonal da matriz do elemento. Analogamente,

para a matriz $\boldsymbol{m}^e$ são armazenados apenas 4 coeficientes. Nas Tab. 4.2 e 4.3 encontram-se, respectivamente, as dimensões das estrututas paralelas referentes as matrizes $\boldsymbol{M}$ de (2.67) e $\boldsymbol{K}$ de (2.68) adequadas à estratégia elemento-por-elemento. Para ambas as matrizes está sendo utilizada a notação introduzida em (3.1). Os termos n_I e n_B correspondem aos nós do tipo *IntNodes* e *IBNodes* do processador i, respectivamente.

Estruturas EBE para $\boldsymbol{M}$	**por elementos**	**por nós**
A_i	4 ebe_{A_i}	$8n_I$
B_i	4 ebe_{B_i}	$8n_B$
C_i	4 ebe_{C_i}	$8n_B$
$A_{s(i)}$	4 $ebe_{A_{s(i)}}$	$8n_B$

Tabela 4.2: Estratégia EBE para $\boldsymbol{M}$

Estruturas EBE para $\boldsymbol{K}$	**por elementos**	**por nós**
A_i	6 ebe_{A_i}	$12n_I$
B_i	6 ebe_{B_i}	$12n_B$
C_i	6 ebe_{C_i}	$12n_B$
$A_{s(i)}$	6 $ebe_{A_{s(i)}}$	$12n_B$

Tabela 4.3: Estratégia EBE para $\boldsymbol{K}$

Recordando o problema exemplo (5.5) do capítulo anterior, ao invés de utilizar os blocos A_i, B_i, C_i e $A_{s(i)}$ descritos em (3.2) para armazer as contribuições das matrizes dos elementos de $\boldsymbol{K}$ de (3.5) conforme visto nas Fig. 3.6, 3.7, 3.8 e 3.9, serão utilizadas as estruturas A_i, B_i, C_i e $A_{s(i)}$ da estratégia de armazenamento elemento-por-elemento. A Fig. 4.4 apresenta a parte da malha exemplo da Fig. 3.2(b) referente ao processador 1. Já a Fig. 4.5 mostra as estruturas paralelas da estratégia elemento-por-elemento referentes ao processador 1 da malha exemplo retratada na Fig. 4.4.

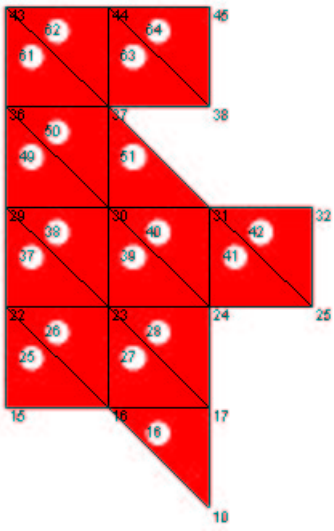


Figura 4.4: Parte da malha da Fig. 3.2(b) referente ao processador 1

A_1

	a_{12}	a_{13}	a_{21}	a_{23}	a_{31}	a_{32}
26	0	0	-0.5	-0.5	0	0
27	0	0	0	0	-0.5	0
28	0	0	0	0	0	-0.5
37	0	0	-0.5	0	0	0
38	-0.5	0	-0.5	-0.5	0	0
39	-0.5	-0.5	0	0	-0.5	0
40	0	0	0	0	0	-0.5
49	0	0	-0.5	0	0	0
50	-0.5	0	0	0	0	0
51	-0.5	-0.5	0	0	0	0

B_1

	a_{12}	a_{13}	a_{21}	a_{23}	a_{31}	a_{32}
26	0	0	-0.5	0	0	0
27	0	0	0	0	-0.5	0
28	0	0	0	0	0	-0.5
39	-0.5	0	0	0	0	0
40	0	0	0	0	0	-0.5
50	-0.5	0	0	0	0	0
51	-0.5	-0.5	0	0	0	0

C_1

	a_{12}	a_{13}	a_{21}	a_{23}	a_{31}	a_{32}
26	-0.5	0	0	0	0	0
27	0	-0.5	0	0	0	0
28	0	0	0	-0.5	0	0
39	0	0	-0.5	0	0	0
40	0	0	0	-0.5	0	0
50	0	0	-0.5	0	0	0
51	0	0	-0.5	0	-0.5	0

$A_{s(1)}$

	a_{12}	a_{13}	a_{21}	a_{23}	a_{31}	a_{32}
16	-0.5	0	-0.5	-0.5	0	-0.5
25	0	0	-0.5	0	0	0
26	-0.5	0	0	0	0	0
27	-0.5	-0.5	-0.5	0	0	0
28	-0.5	0	-0.5	-0.5	0	0
39	0	0	-0.5	0	0	0
40	-0.5	0	-0.5	-0.5	0	0
41	-0.5	-0.5	-0.5	0	-0.5	0
42	-0.5	0	-0.5	-0.5	0	-0.5
50	0	0	-0.5	-0.5	0	0
51	0	0	-0.5	0	-0.5	0
61	0	0	-0.5	0	0	0
62	-0.5	0	0	0	0	0
63	-0.5	-0.5	-0.5	0	0	0
64	-0.5	0	0	0	0	0

Figura 4.5: Contribuições elemento-por-elemento no processador 1

4.3 Estratégia aresta-por-aresta - EDE

Uma forma alternativa para controlar a esparsidade presente em (3.2) é considerar o armazenamento aresta-por-aresta [7][28]. A partir das matrizes dos elementos procedentes da formulação apresentada no Capítulo 2, define-se um desmembramento dos coeficientes das matrizes $\boldsymbol{m}^e$ de (2.70) e $\boldsymbol{k}^e$ de (2.71) gerando as contribuições em cada aresta, que podem ser representadas por:

$$\underbrace{\begin{bmatrix} \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \end{bmatrix}}_{elemento\ e} = \underbrace{\begin{bmatrix} \times & \times & 0 \\ \times & \times & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{aresta\ ij} + \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & \times & \times \\ 0 & \times & \times \end{bmatrix}}_{aresta\ jk} + \underbrace{\begin{bmatrix} \times & 0 & \times \\ 0 & 0 & 0 \\ \times & 0 & \times \end{bmatrix}}_{aresta\ ki} \tag{4.1}$$

Entretanto, dada uma aresta s incidem sobre ela as informações dos elementos adjacentes a ela, como pode ser observado na Fig. 4.6. Assim, todas as contribuições referentes à aresta s estão presentes nos elementos e e f:

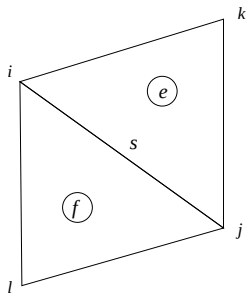


Figura 4.6: Elementos adjacentes a aresta s, formada pelos nós i e j

Dessa forma, as matrizes de aresta $\boldsymbol{m}^s$ e $\boldsymbol{k}^s$ são obtidas da seguinte forma:

$$\underbrace{\begin{bmatrix} \circ & \circ \\ \circ & \circ \end{bmatrix}}_{aresta\ s} = \underbrace{\begin{bmatrix} \times & \times \\ \times & \times \end{bmatrix}}_{elemento\ e} + \underbrace{\begin{bmatrix} \times & \times \\ \times & \times \end{bmatrix}}_{elemento\ f} \tag{4.2}$$

De forma semelhante a (2.67) e (2.68) as matrizes das arestas podem ser definidas globalmente como:

$$\boldsymbol{M} = \mathop{\mathbf{A}}_{s=1}^{nedges} (\boldsymbol{m}^s) \tag{4.3}$$

$$\boldsymbol{K} = \mathop{\mathbf{A}}_{s=1}^{nedges} (\boldsymbol{k}^s) \tag{4.4}$$

onde *nedges* é o número de arestas da malha. De acordo com Catabriga et al. [7], as matrizes $\boldsymbol{m}^s$ e $\boldsymbol{k}^s$ possuem a propriedade de deficiência de posto análoga às matrizes

(2.67) e (2.68), ou seja, $\boldsymbol{m}^s$ pode ser representada por 3 coeficientes e $\boldsymbol{k}^s$ pode ser representada por apenas 2 coeficientes distintos. É claro que a matriz $\boldsymbol{M}^*$ de (2.69) também pode ser calculada a partir das contribuições das arestas.

Como todas as contribuições foram obtidas em nível de cada elemento foi necessário criar uma função *hash* [25] para obter as matrizes (4.3) e (4.4) em função das arestas da malha. Dada a contribuição de um nó $Node_I$ em um nó $Node_J$ com $Node_I, Node_J = 1, 2, ..., N_I$, onde N_I é o número de nós incógnitas da malha, a *hash* é definida como:

$$\begin{aligned} & i = (Node_I + Node_J) \mod PrimeNode(N_I) \\ & Se\ Node_I < Node_J \\ & \quad j = Node_I \mod MaxEDNode \\ & Senão \\ & \quad j = Node_J \mod MaxEDNode \end{aligned} \tag{4.5}$$

onde $0 \leq i < PrimeNode(N_I)$, $0 \leq j < MaxEDNode$. $PrimeNode(N_I)$ é uma função que retorna o maior número primo menor ou igual a N_I e $MaxEDNode$ é o menor número primo maior ou igual ao número médio de arestas distintas chegando em cada nó. Segundo Knuth [25], para uma função *hash* da forma

$$h(K) = K \mod M \tag{4.6}$$

deve ser escolhido um número primo M, tal que M não divida $r^k \pm a$, onde k e a são números inteiros pequenos e $r = 10$ para o caso do sistema de base decimal utilizado para o números inteiros. Quando o tamanho da malha é considerável, $M = PrimeNode(N_I)$ obedece esse critério, e na maioria dos casos a função *hash* é satisfatória.

Em relação a estrutura paralela aresta-por-aresta, os conceitos são equivalentes àqueles aplicados a estratégia elemento-por-elemento, apenas substituindo os elementos pelas arestas. Os blocos de matrizes A_i, B_i, C_i e $A_{s(i)}$ de (3.2) agora são substituídos por estruturas correspondentes que têm suas dimensões relacionadas com quatro tipos de arestas: arestas ede_{A_i}, arestas ede_{B_i}, arestas ede_{C_i} e arestas $ede_{A_{s(i)}}$. As arestas ede_{A_i} são aquelas que armazenam suas contribuições em A_i de (3.2), da mesma forma que as arestas ede_{B_i}, ede_{C_i} e $ede_{A_{s(i)}}$ são aquelas que armazenam suas contribuições em B_i, C_i e $A_{s(i)}$ de (3.2), respectivamente.

Criando uma relação nó-aresta verifica-se que a cada nó de i associa-se apenas 3 arestas distintas de i. Dado um nó *IntNodes*, ele está associado a 3 arestas distintas do tipo ede_{A_i}. Tendo em vista um nó *IBNodes* pertencente a uma partição i, verifica-se que a tal nó *IBNodes* associa-se a uma aresta $ede_{A_{s(i)}}$, duas arestas ede_{B_i}, duas arestas ede_{C_i}. Analogamente à estratégia elemento-por-elemento, as matrizes das arestas (4.3) e (4.4) podem ser adaptadas a estratégia aresta-por-aresta. As Tab. 4.4 e 4.5 apresentam as dimensões das estruturas $\boldsymbol{M}$ e $\boldsymbol{K}$ de (4.3) e (4.4), respectivamente, utilizando a estratégia aresta-por-aresta em relação ao número de arestas ede_{A_i}, ede_{B_i}, ede_{C_i} e $ede_{A_{s(i)}}$ e em relação aos nós *IntNodes* e *IBNodes*. Note que as arestas ede_{B_i} e ede_{C_i} têm contribuições menores que as demais arestas. Isso porque uma aresta que liga um nó *IntNodes* a um nó *IBNodes* é uma aresta ede_{B_i} e guarda a contribuição apenas numa direção, ou seja, do nó *IntNodes* no nó *IBNodes*. Olhando a mesma aresta no sentido oposto, ou seja, do nó *IBNodes* para o nó *IntNodes*, tem-se uma aresta ede_{C_i}, e portanto, tal aresta guarda apenas a contribuição do nó *IBNodes* no nó *IntNodes*.

Estruturas EDE para $\boldsymbol{M}$	**por arestas**	**por nós**
A_i	3 ede_{A_i}	$9n_I$
B_i	2 ede_{B_i}	$4n_B$
C_i	2 ede_{C_i}	$4n_B$
$A_{s(i)}$	3 $ede_{A_{s(i)}}$	$3n_B$

Tabela 4.4: Estratégia EDE para $\boldsymbol{M}$

Estruturas EDE para $\boldsymbol{K}$	**por arestas**	**por nós**
A_i	2 ede_{A_i}	$6n_I$
B_i	ede_{B_i}	$2n_B$
C_i	ede_{C_i}	$2n_B$
$A_{s(i)}$	2 $ede_{A_{s(i)}}$	$2n_B$

Tabela 4.5: Estratégia EDE para $\boldsymbol{K}$

Retomando o problema exemplo (5.5), desta vez, os blocos A_i, B_i, C_i e $A_{s(i)}$ descritos em (3.2) são utilizados para armazenar as contribuições de $\boldsymbol{K}$ de (3.5) referentes as estruturas da estratégia aresta-por-aresta. A Fig. 4.7 apresenta as

estruturas paralelas da estratégia aresta-por-aresta referentes ao processador 1 do particionamento do problema exemplo retratado na Fig. 4.4. Repare que as arestas que contém um nó numerado por -1 representam um conjunto de arestas que tem como parte integrante um nó incógnita e um nó de valor prescrito. A aresta $(23, -1)$ de A_1 armazena as informações sobre as arestas $(23, 22)$ e $(23, 29)$. Já a aresta $(37, -1)$ de $A_{s(1)}$ armazena as informações das arestas $(37, 36)$, $(37, 43)$ e $(37, 44)$.

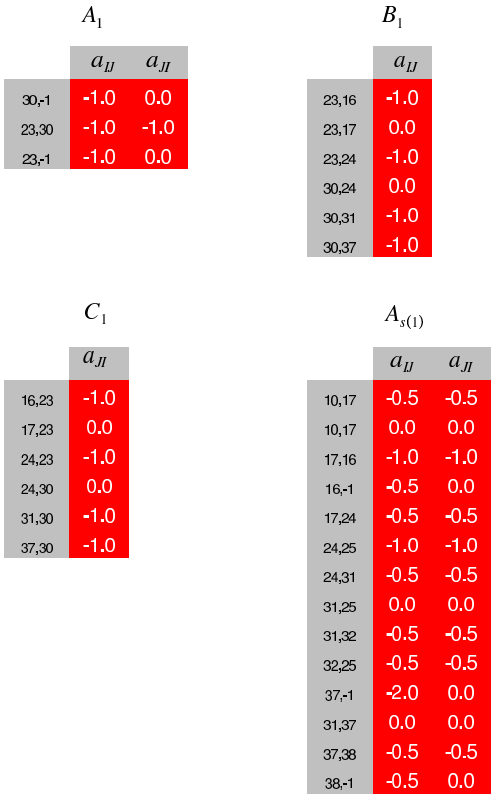


Figura 4.7: Contribuições aresta-por-aresta no processador 1

4.4 Estratégia compressed row storage - CSR

O armazenamento CSR [35] 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 armazena a coluna correspondente que cada coeficiente não nulo ocuparia em A e o vetor IA mapea em AA o primeiro elemento não nulo de cada linha de A. A Fig. 4.8 apresenta um exemplo de aplicação da estratégia de armazenamento CSR tradicional em uma matriz A de ordem 5×5, esparsa, composta por apenas 10 coeficientes não nulos.

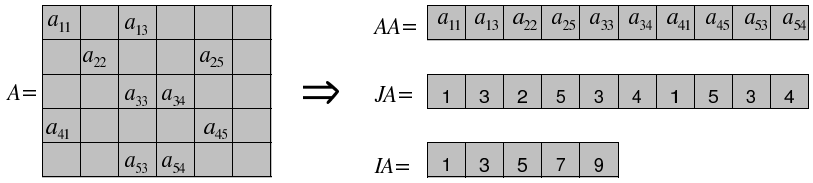


Figura 4.8: Armazenamento CSR Tradicional

A versão paralela da estratégia CSR pode ser derivada da tradicional com algumas adequações. A Tab. 4.6 apresenta as estruturas CSR que devem ser aplicadas à estrutura paralela de (3.2). Observe que cada um dos blocos A_i, B_i, C_i e $A_{s(i)}$ de (3.2) são substituídos por 3 vetores cujas funções são exatamente aquelas descritas para a estratégia CSR tradicional.

Blocos da estrutura padrão	Estruturas equivalentes no CSR
A_i	AA_i, JA_i e IA_i
B_i	BB_i, JB_i e IB_i
C_i	CC_i, JC_i e IC_i
$A_{s(i)}$	$AA_{s(i)}$, $JA_{s(i)}$ e $IA_{s(i)}$

Tabela 4.6: Estruturas CSR aplicadas a (3.2)

No início do capítulo foi discutido que dada uma matriz A discretizada pelo método dos elementos finitos, tem-se em média 7 coeficientes não nulos em cada linha. Quando a malha é de grande porte, os coeficientes dos blocos A_i de (3.2) têm comportamento equivalente aos coeficientes da matriz global A. Portanto, as influências guardadas em AA_i são da ordem de $7n_I$, onde n_I representa o número de nós do tipo *IntNodes*. Já em $A_{s(i)}$ de (3.2), são guardadas uma média de 3 contribuições não nulas por linha, uma vez cada nó *IBnodes* tem relação com outros dois nós *IBNodes*. Por conseguinte, as influências guardadas em $AA_{s(i)}$ resultam numa estrutura da ordem de $3n_B$, onde n_B é o número de nós *IntNodes* da partição i. Os blocos B_i e C_i de (3.2) contêm o mesmo número de coeficientes não nulos. Dado um nó *IBNodes* da partição i, tal nó está relacionado com uma média de 2 nós *IntNodes*. Assim, as estruturas BB_i e CC_i têm ordem $2n_B$. A Tab. 4.7 apresenta um resumo das dimensões das estruturas CSR em função dos nós *IntNodes* e *IBNodes*. Repare que as estruturas JA_i, JB_i, JC_i e $JA_{s(i)}$ possuem as mesmas

dimensões que as estruturas AA_i, BB_i, CC_i e $AA_{s(i)}$, respectivamente. Entretanto as quatro primeiras armazenam números inteiros e as quatro últimas números reais. As estruturas IA_i, IB_i, IC_i e $IA_{s(i)}$, por sua vez, têm sua dimensões equivalentes ao número de linhas dos blocos A_i, B_i, C_i e $A_{s(i)}$ de (3.2), respectivamente.

Estruturas CSR	Dimensões
AA_i, JA_i e IA_i	$7n_I$, $7n_I$ e n_I
BB_i, JB_i e IB_i	$2n_B$, $2n_B$ e n_I
CC_i, JC_i e IC_i	$2n_B$, $2n_B$ e n_B
$AA_{s(i)}$, $JA_{s(i)}$ e $IA_{s(i)}$	$3n_B$, $3n_B$ e n_B

Tabela 4.7: Dimensões das estruturas CSR

As estruturas B_i e C_i da formulação global (3.2) 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 de (3.2).

Novamente, o problema exemplo (5.5) pode ser aplicado a essa nova estratégia. A Fig. 4.9 apresenta as contribuições da matriz $\boldsymbol{K}$ de (3.5) referentes as estruturas da estratégia CSR do processador 1 conforme pode ser observado na Fig. 4.4. No exemplo, as estruturas $AA_{s(i)}$, $JA_{s(i)}$ e $IA_{s(i)}$ apresentam um maior número de coeficientes que as demais. Para problemas reais, a malha tem dimensões bem maiores, o que indica que o número de nós *IntNodes* é bem maior que o número de nós *IBNodes*, e conseqüentemente, o número de coeficientes das estruturas AA_i, JA_i e IA_i supera em muito o número de coeficientes das outras estruturas.

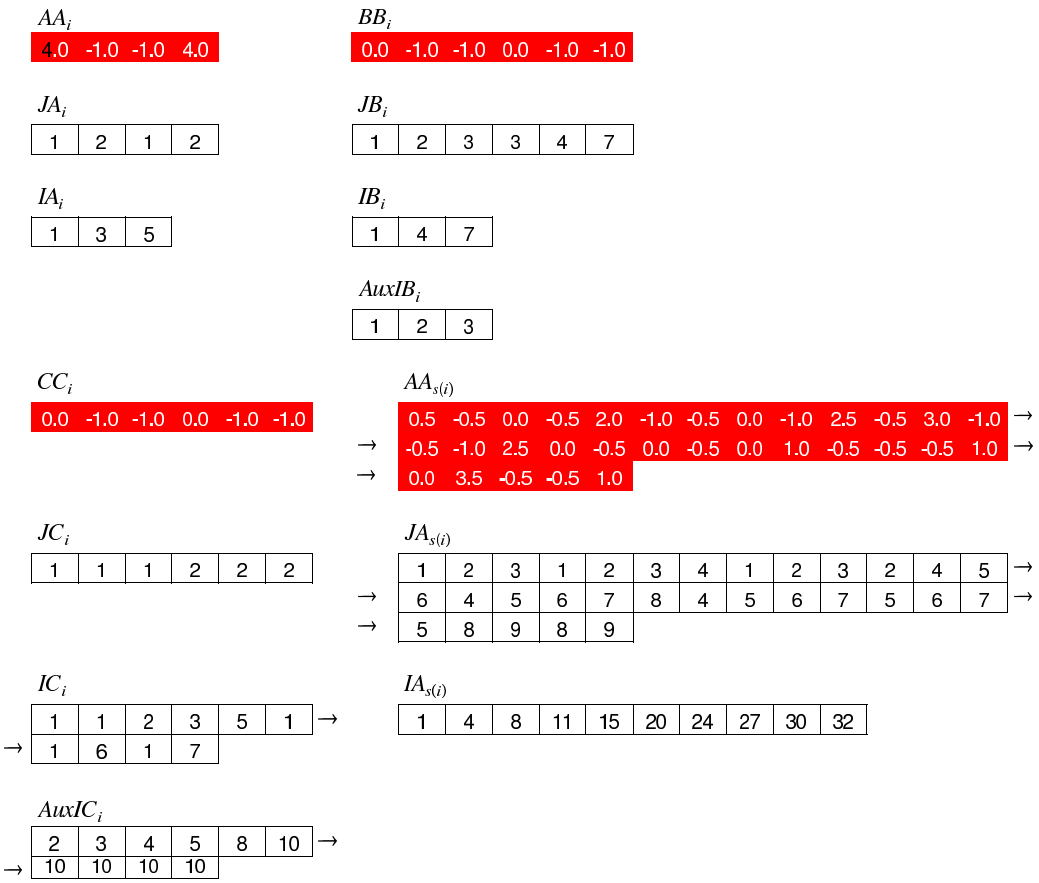


Figura 4.9: Contribuições CSR no processador 1

4.5 Análise comparativa entre as estratégias EBE, EDE e CSR

A Tab. 4.8 contém um resumo da análise teórica da relação entre os nós *IntNodes* e *IBNodes* com o tipos de elementos, os tipos de arestas e os nós *IntNodes* e *IBNodes* das estratégias elemento-por-elemento, aresta-por-aresta e CSR, respectivamente, conforme visto nas seções anteriores. O termo n_I representa o número de nós *IntNodes* e n_B representa o número de nós *IBNodes*.

Tipo de nó	**por elementos**	**por arestas**	**por nós**
IntNodes	2 ebe_{A_i}	3 ede_{A_i}	$7n_I$
IBNodes	2 ebe_{B_i}	2 ede_{B_i}	$2n_B$
	2 ebe_{C_i}	2 ede_{C_i}	$2n_B$
	2 $ebe_{A_{s(i)}}$	1 $ede_{A_{s(i)}}$	$3n_B$

Tabela 4.8: Relação entre os tipos de nós, tipos de elementos, arestas e nós

As relações vistas na Tab. 4.8 definem um número denominado coeficiente de proporcionalidade médio (CPM) para cada caso. Por exemplo, na estratétia EBE, o número de elementos ebe_{A_i} em cada partição i é aproximadamente 2 vezes o número de nós *IntNodes* dessa partição, ou seja, o CPM nesse caso é 2. A Tab. 4.9 apresenta os CPM dos diversos tipos de dados de cada estratégia. Na primeira coluna dessa tabela tem-se a estratégia de armazenamento, na segunda coluna há uma classificação dos dados de acordo com a estratégia abordada, e na terceira e última coluna tem-se o CPM teórico para cada tipo de dado de acordo com a estratégia escolhida.

Toda a análise teórica levou em consideração o caso médio de cada situação. Entretanto, para confirmar a argumentação teórica, foi considerada uma malha não-estruturada composta por 259299 nós e 516600 elementos, gerada pelo software GMSH e particionadonada pelo METIS em 4 partições, conforme discutido no Capítulo 3. A Tab. 4.10 apresenta os dados da malha não-estruturada citada, o número de nós *IntNodes* e *IBNodes* em cada processador e a soma desses números de nós referentes aos 4 processadores.

A Tab. 4.11 apresenta a quantidade de elementos ebe_{A_i}, ebe_{B_i}, ebe_{C_i} e $ebe_{A_{s(i)}}$ da

Estratégia	**Tipos de dados**	**CPM teórico**
EBE	elementos de A_i	2
	elementos de B_i	2
	elementos de C_i	2
	elementos de $A_{s(i)}$	2
EDE	arestas de A_i	3
	arestas de B_i	2
	arestas de C_i	2
	arestas de $A_{s(i)}$	1
CSR	nós de AA_i	7
	nós de BB_i	2
	nós de CC_i	2
	nós de $AA_{s(i)}$	3

Tabela 4.9: CPM teórico de acordo com a estratégia de armazenamento e tipos de dados

Proc	*Nós IntNodes*	Nós *IBNodes*
1	64121	547
2	64013	609
3	64062	548
4	64001	510
Totais	256197	2214

Tabela 4.10: Nós *IntNodes* e *IBNodes* dos processadores 1, 2, 3 e 4

malha não-estruturada citada anteriormente, apresenta as somas desses elementos ao longo dos 4 processadores e CPM para cada tipo de elemento. O CPM dos elementos ebe_{A_i} foi calculado através de uma divisão da soma dos elementos ebe_{A_i} dos quatro processadores pela soma dos nós *IntNodes* também dos quatro processadores. De maneira equivalente, os CPM dos elementos ebe_{B_i}, ebe_{C_i} e $ebe_{A_{s(i)}}$ foram calculadores através da divisão da soma respectiva de cada tipo desses elementos nos quatro processadores pela soma dos nós *IBNodes* dos quatro processadores. Repare que o CPM teórico apresentado na Tab. 4.9 é aproximandamente o mesmo obtido para a malha não-estrutura em questão.

Proc	**Elem** ebe_{A_i}	**Elem** ebe_{B_i}	**Elem** ebe_{C_i}	**Elem** $ebe_{A_{s(i)}}$
1	129264	1190	1190	1191
2	129092	1327	1327	1328
3	129168	1181	1181	1183
4	129067	1086	1086	1087
Totais	516591	4784	4784	4789
CPM	2.02	2.16	2.16	2.16

Tabela 4.11: EBE

Na Tab. 4.12 observa-se a quantidade de arestas ede_{A_i}, ede_{B_i}, ede_{C_i} e $ede_{A_{s(i)}}$ da malha não-estruturada citada no início da seção. Assim, como na estratégia EBE, os CPM são obtidos através de divisões das somas dos tipos de arestas ao longo dos 4 processadores pela soma dos nós *IntNodes* ou *IBNodes* dependo do tipo de aresta. Note que, também nesse caso, os CPM teóricos da estratégia EDE apresentados na Tab. 4.9 são aproximandamente os mesmos que os obtidos na malha não-estruturada.

A Tab. 4.13 contém informações sobre os CPM referente a estratégia de armazenamento CSR. A primeira coluna representa o número do processador, e as demais representam o número de coeficientes presentes em cada uma das estruturas associadas a estratégia CSR em cada processador. Assim como nas Tab. 4.11 e 4.12, existe uma linha contendo totais referentes as somas dos números de coeficientes de cada tipo em cada um dos quatro processadores. Os CPM também são obtidos através de divisões. O primeiro deles obtém-se dividindo a soma do número de coe-

Proc	**Aresta** ede_{A_i}	**Aresta** ede_{B_i}	**Aresta** ede_{C_i}	**Aresta** $ede_{A_{s(i)}}$
1	191719	1189	1189	548
2	191320	1326	1326	610
3	191551	1180	1180	550
4	191426	1085	1085	511
Totais	766016	4780	4780	2219
CPM	2.99	2.16	2.16	1.00

Tabela 4.12: EDE

ficientes de AA_i dos quatro processadores pelo total de nós *IntNodes* da Tab. 4.10. Os demais CPM são obtidos pela divisão das somas dos números de coeficientes de BB_i, CC_i e $AA_{s(i)}$ pelo total de nós *IBNodes* da Tab. 4.10. Mais uma vez, os dados experimentais confirmam a discussão teórica, ou seja, comparando os CPM teóricos da estratégia CSR descritos na Tab. 4.9 com os CPM desta tabela, gerados a partir da malha não-estruturada do início da seção, verifica-se que ambos valores são aproximandamente os mesmos.

Proc	**Coef de** AA_i	**Coef de** BB_i	**Coef de** CC_i	**Coef de** $AA_{s(i)}$
1	446567	1189	1189	1639
2	445695	1326	1326	1825
3	446132	1180	1180	1644
4	445685	1085	1085	1528
Totais	1784079	4780	4780	6636
CPM	6.96	2.16	2.16	3.00

Tabela 4.13: CSR

4.6 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 grande parte dos cálculos. Conseqüentemente, analisando a complexidade média do produto matriz-vetor, tem-se uma idéia do comportamento de toda implementação. A complexidade do produto matriz-vetor está relacionada diretamente com a estratégia de armazenamento utilizada. Em relação ao consumo de memória, ao comparar as Tab. 4.2, 4.3, 4.4, 4.5 e 4.7, pode-se verificar que a estratégia EDE tem o menor consumo, seguida da estratégia CSR.

O produto matriz-vetor é composto por operações de soma e multiplicação de pontos flutuantes além de 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 médio de multiplicações de pontos flutuantes.

Na Tab. 4.14 podem ser encontradas as expressões que representam o número médio de operações de multiplicação de ponto flutuante realizadas em cada estratégia de armazenamento, tanto para o caso paralelo, como para o sequencial. Os termos N_I e N_B representam, respectivamente, o soma de todos nós *IntNodes* e a soma de todos nós *IBNodes* ao longo de todos subdomínios (no caso sequencial não existem nós *IBNodes*). Observa-se que o produto matriz-vetor da versão CSR realiza menos operações que o EBE e o EDE, sugerindo que a estratégia CSR seja mais rápida que as demais.

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, conclui-se que a massa de dados a ser computada passa a ser muito maior que a envolvida em comunicação, o que pode tornar o tempo das comunicações propocionalmente menor em relação ao tempo de processamento. Assim, as versões paralelas tendem a ser p vezes mais rápidas que as versões sequenciais.

Operação	**Paralelo**	**Sequencial**
Matriz-vetor CSR	$\frac{1}{p}\ [7N_I + 7N_B]$	$7N_I$
Matriz-vetor EBE	$\frac{1}{p}\ [18N_I + 42N_B]$	$18N_I$
Matriz-vetor EDE	$\frac{1}{p}\ [12N_I + 12N_B]$	$12N_I$
Produto Interno	$\frac{1}{p}\ [N_I + 2N_B]$	N_I

Tabela 4.14: Complexidade média do produto matriz-vetor e produto interno

O produto interno é absolutamente o mesmo independente da estratégia de armazenamento escolhida. Dessa a forma, a definição mantém-se conforme apresentada em (3.14). Visando simplificar o modelo, o tempo para realizar a operação de cada divisão representada pelo termo $Shared[i]$ de (3.14) foi considerado o mesmo que o de uma multiplicação.

As operações de comunicação entre os processadores não ocorrem durante os cálculo do produto matriz-vetor. Somente após esse produto ocorre a comunicação e, independente da estratégia de armazenamento utilizada, é realizada essencialmente da mesma maneira, ou seja, cada processador calcula um vetor contendo seus respectivos nós *IBNodes* obtidos através do produto matriz-vetor; em seguida, a função *Update*, descrita na Fig. 3.15, é utilizada para efetuar a devida atualização. Dessa forma, o tempo gasto na comunicação que ocorre após cada produto matriz-vetor é o mesmo em cada estratégia. Assim, optou-se por omitir tais tempos na Tab. 4.14, uma vez que, o objetivo não era calcular a complexidade do algoritmo, mas criar alguns parâmetros teóricos para a comparação entre as estratégias de armazenamento.

No próximo capítulo serão apresentados alguns testes de desempenho visando verificar se a tendência proposta pela operação de produto matriz-vetor é mantida, ou seja, se os algoritmos que empregam a estratégia CSR são mais rápidos que os algoritmos que utiliza a estratégia EDE e se estes últimos por sua vez são mais rápidos que os que utilizam a estratégia EBE.

Capítulo 5

Testes de Desempenho

Nesse capítulo são apresentados os resultados de quatro problemas exemplos que visam comparar a eficiência de cada estratégia de armazenamento. Inicialmente é apresentado um breve histórico sobre a configuração do cluster onde foram realizados todos os testes. Em seguida, cada um dos problemas exemplos é tratado separadamente. Há uma descrição dos parâmetros físicos envolvidos em cada caso, informações como tamanho de domínio, tamanho da malha de discretização, método de solução de sistema linear utilizado e outras informações importantes, como o tempo de processamento em cada estratégia estudada, gráficos de speedup e eficiência. Para cada caso são utilizados um tamanho de malha médio e um grande.

5.1 Cluster Enterprise

Todos os testes foram realizados no cluster Enterprise do Laboratório de Computação de Alto Desempenho do Centro Tecnológico da Universidade Federal do Espírito Santo (LCAD - CT - UFES). O cluster é composto por 64 nós de processamento e um servidor. Todos os nós utilizam o sistema operacional Linux *Red Hat* 7.1 (kernel 2.4-20). Em relação as ferramentas de programação disponíveis no Enterprise, foram utilizadas o compilador gcc-2.96-112.7.1 da linguagem C para GNU/Linux, parte da distribuição *Red Hat* 7.1 e a biblioteca do padrão MPI lam-6.5.9-tcp.1. Os códigos sequenciais foram compilados utilizando "gcc -o file file.c" e os códigos paralelos "mpicc -o file file.c".

A rede de interconexão é composta basicamente por dois *switches* 3COM modelo

4300. Cada *switch* possui 48 portas e cada porta tem uma capacitadade teórica de 100Mb/s para comunicação com os nós de processamento e são interligados entre si através de um módulo *Gibabit Ethernet* (1000Mb/s). Para uma melhor utilização, a quantidade de nós de processamento é dividida igualmente entre os dois *switches* e um deles possui uma porta *Gibabit* a mais para conexão com o servidor.

Cada um dos 64 nós de processamento do Enterprise possui memória SDRAM de 256MB e disco rígido Ultra ATA de 20GB. O processador de cada nó é o ATHLON XP 1800+ fabricado pela AMD. Visto que cada processador possui duas unidades de ponto flutuante e frequência de operação de 1,53 GHz, tem-se um desempenho teórico de pico igual a $2 \times 1{,}53 = 3{,}06$ GFLOP/s por nó de processamento.

A máquina servidora possui praticamente a mesma configuração dos nós de processamento exceto que conta com 512MB de memória RAM, 80 GB de disco e duas placas de rede, uma 3COM *Gigabit Ethernet* para conexão com um dos *switches* do cluster e uma 3COM *Fast-Ethernet* para conexão externa. Esta máquina também é responsável pela distribuição dos processos nos nós de processamento (a gerência de filas de processos é feita pelo SGE - Sun Grid Engine [41]), pelo armazenamento das contas dos usuários, além das atividades de configuração, atualização e monitoramento dos nós de processamento. A Fig. 5.1 ilustra as conexões de rede entre as máquinas externas, o servidor e os nós de processamento.

Para medir o desempenho das implementações das estratégias de armazenamento executadas no cluster Enterprise, foram utilizadas algumas métricas. Na seção seguinte é detalhado como tais medidas foram definidas e como foram utilizadas.

5.2 Medidas de desempenho adotadas

O desempenho dos algoritmos paralelos desenvolvidos foi avaliado através de três medidas: tempo de processamento, *speedup* e eficiência. O tempo de processamento foi medido com o auxílio de algumas funções padrão da linguagem C que utilizam a biblioteca `time.h`. Quando o processo começa sua execução, a função `time` inicializa uma variável com um formato de tempo especial (formato time_t) e antes que a função de finalização do MPI seja acionada, uma segunda variável é marcada com o mesmo formato de tempo da variável inicial. Depois disso, é chamada a função

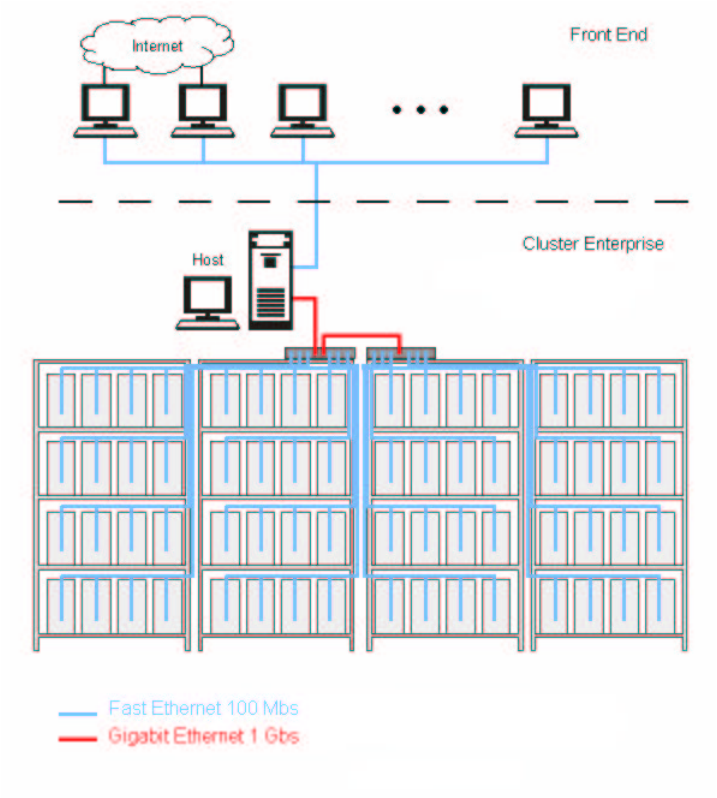


Figura 5.1: Conexões de rede do cluster Enterprise

`difftime` que calcula a diferença entre essas duas variáveis e retorna uma resposta em segundos. Assim, o tempo de execução total é calculado e escrito em um arquivo. Cada teste foi executado 5 vezes, o maior tempo e o menor foram desprezados, sendo o tempo de execução de cada teste calculado através de uma média entre os três valores intermediários.

O *speedup* (S_i) e a eficiência (E_i) foram definidos conforme as expressões

$$S_i = \frac{t_1}{t_i}, \qquad E_i = \frac{S_i}{i}, \tag{5.1}$$

onde t_1 representa o tempo processamento do algoritmo sequencial[1] implementado e t_i, com $i = 1, .., p$, o tempo de processamento do algoritmo paralelo executado utilizando i processadores, sendo p o número de processadores envolvidos no processamento paralelo.

Como já mencionado, os problemas exemplos foram executados em malhas de tamanho médio e tamanho grande. No caso das malhas de tamanho grande, o consumo de memória ultrapassou a quantidade de memória de cada nó do cluster,

[1] teoricamente esse é o tempo do melhor algoritmo sequencial possível

ou seja, a memória utilizada neste caso foi maior que 256MB. Como resultado disso, os testes para malhas maiores foram realizados apenas a partir de 4 processadores, e portanto, no caso dessas malhas o *speedup* e a eficiência foram redefinidos:

$$S_i = i \quad se\ i < 4 \quad e \quad S_i = \frac{4t_4}{t_i} \quad se\ i \geq 4 \tag{5.2}$$

$$E_i = 1 \quad se\ i < 4 \quad e \quad E_i = \frac{S_i}{i} \quad se\ i \geq 4 \tag{5.3}$$

onde t_4 representa o tempo processamento do algoritmo paralelo executado utilizando 4 processadores e, t_i, com $i = 4, .., p$, o tempo de processamento do algoritmo paralelo executado utilizando i processadores.

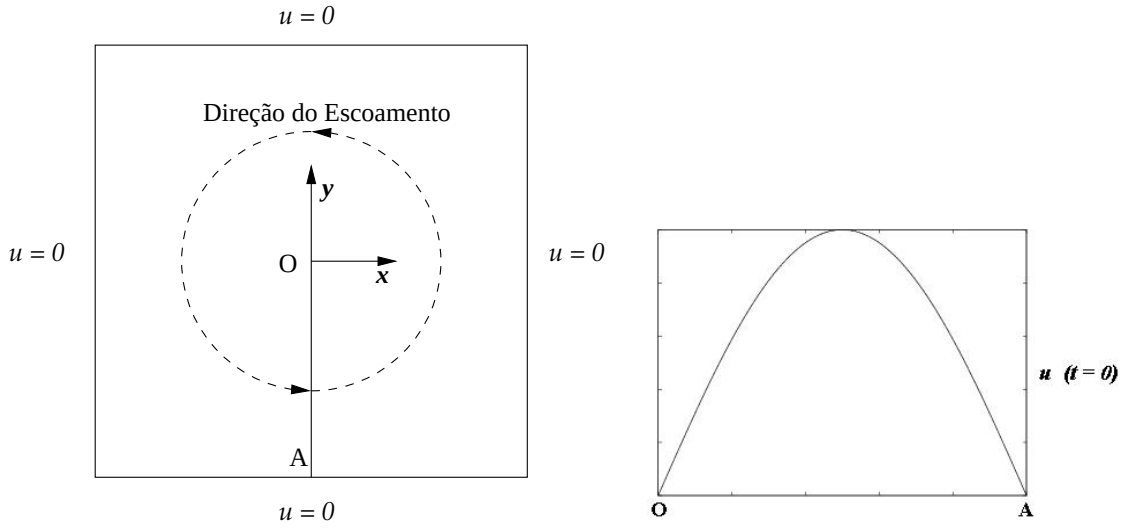
Para cada problema exemplo, relativo às malhas médias, foram executados testes que utilizaram 1, 2, 4, 8, 12, 16, 24, 32 e 48 processadores. Já em relação às malhas de tamanho grande, os testes foram realizados utilizando 4, 8, 12, 16, 24 32 e 48 processadores. Em cada caso, as estratégias de armazenamento EBE, EDE e CSR tiveram seus desempenhos avaliados. Assim, nas próximas seções serão apresentados os problemas exemplos e seus respectivos testes de desempenho. A quatidade de memória utilizado em cada problema exemplo pode ser estimada através das Tab. 4.2, 4.3, 4.4, 4.5 e 4.7 conforme a estratégia de armazenamento utilizada.

5.3 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. 5.2, com velocidade e difusividade dadas por:

$$\boldsymbol{\beta} = \begin{bmatrix} -y \\ x \end{bmatrix} \qquad \boldsymbol{\kappa} = \begin{bmatrix} 10^{-6} & 0 \\ 0 & 10^{-6} \end{bmatrix}. \tag{5.4}$$

Ao longo do contorno do domínio quadrado a solução é nula, e no contorno interno $\overline{OA}$ a solução possui variação cossenoidal, como mostrado na Fig. 5.2. A solução exata para esse problema é essencialmente uma advecção pura da condição no contorno $\overline{OA}$ ao longo das linhas de corrente circulares. A Fig. 5.3 representa a solução



(a) Descrição geométrica do problema

(b) Perfil de um instante $t = 0$ na direção $\overline{OA}$

Figura 5.2: Advecção em um campo de escoamento rotacional.

obtida utilizando a estratégia de armazenamento CSR, considerando uma malha estruturada com 40×40 células, contendo 2 elementos triangulares em cada célula e, executada em 4 processadores. A solução aproximada corresponde à solução exata esperada. A solução obtida considerando as estratégias EBE, EDE e CSR, 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 foram consideradas duas malhas estruturadas, uma com 512×512 células e outra 1024×1024 células, sendo cada célula dividida em dois elementos triangulares, totalizando 263169 nós e 524288 elementos para a primeira malha e 1050625 nós e 2097152 elementos para a segunda malha. Para o caso sequencial, o número de nós incógnitas, ou nós *IntNodes* da malha média é 260865 nós e para a malha grande 1046017 nós.

A Tab. 5.1 apresenta, para a malha média, os tempos necessários para a solução considerando 1, 2, 4, 8, 12, 16, 24, 32 e 48 processadores e as três estratégias de armazenamento. Além disso, apresenta para cada estratégia o correspondente número de objetos processados: o número de elementos para a estratégia EBE, o número de arestas para a estratégia EDE e o número de coeficientes não nulos para

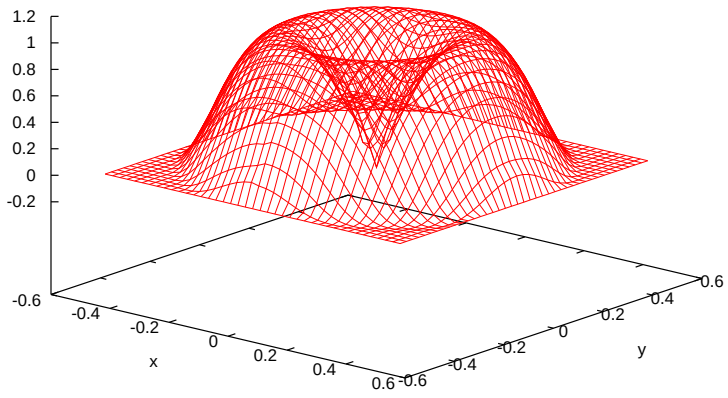


Figura 5.3: Advecção em um campo de escoamento rotacional - Solução com 4 processadores utilizando a estratégia de armazenamento CSR.

a estratégia CSR. Observa-se que a estratégia CSR é a mais rápida dentre as três, seguida pela estratégia EDE.

Estratégia	**Nº de objetos**	**Número de processadores**								
		1	2	4	8	12	16	24	32	48
EBE	524285	5140	2934	1578	1016	611	467	423	281	232
EDE	782592	4958	2522	1305	767	504	392	278	228	202
CSR	1820947	3732	1727	892	531	344	267	202	173	153

Tabela 5.1: Advecção em um campo de escoamento rotacional - Tempo de execução em segundos - Malha média.

A Tab. 5.2 apresenta, em relação à malha grande, os tempos necessários para a solução considerando 4, 8, 12, 16, 24, 32 e 48 processadores e as três estratégias de armazenamento, além do número de objetos processados em cada estratégia. Como pode ser observado, as proporcionalidades referentes ao número médio de operações do produto matriz-vetor descritas na Tab. 4.14 não foram mantidas. Isto porque, 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 produtos internos,

operações elementares do método GMRES e comunicações 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 *cache* [16] pela estratégia EBE ou mesmo um maior acesso à memória realizado pela estratégia EDE, reduzindo assim sua vantagem em relação ao menor número de operações de ponto flutuante realizadas. Outro ponto importante é a necessidade de acessos constantes às estruturas auxiliares presentes, por exemplo, na estratégia CSR.

Estratégia	**Nº de objetos**	**Número de processadores**						
		4	8	12	16	24	32	48
EBE	2097149	13557	7988	5353	4559	2976	2832	1920
EDE	3138048	13517	6864	4313	3650	2677	2109	1518
CSR	7311891	8105	4607	3118	2777	1744	1511	1163

Tabela 5.2: Advecção em um campo de escoamento rotacional - Tempo de execução em segundos - Malha grande.

O número de iterações do algoritmo GMRES para a malha média foi 12355 iterações em todas as estratégias e independente do número de processadores utililizados. Os tempos de montagem das estruturas referentes as estratégias EBE, EDE e CSR foram respectivamente 0.202s, 0.201s e 0.324s. Já em relação a malha grande, o número de iterações do algoritmo GMRES foi 29617 iterações com um tempo de montagem de 0.920s, 1.385s e 3.387s para as estratégias EBE, EDE e CSR, respectivamente. Em relação aos demais problemas exemplos, o número de iterações e tempo de montagem de estruturas utilizadas em cada estratégia de armazenamento manteve proporções semelhantes.

As Fig. 5.4 e 5.5 apresentam os gráficos de *speedup* e eficiência do problema de advecção em um campo de escoamento rotacional para a malha de tamanho médio. Observa-se que a estratégia CSR obteve maiores valores de *speedup* e eficiência nos casos com menos máquinas mas há uma tendência que para um número maior de máquinas a estratégia EDE obtenha melhores valores de *speedup* e eficiência.

As Fig. 5.6 e 5.7, por sua vez, apresentam os gráficos de *speedup* e eficiência do problema de advecção em um campo de escoamento rotacional para a malha

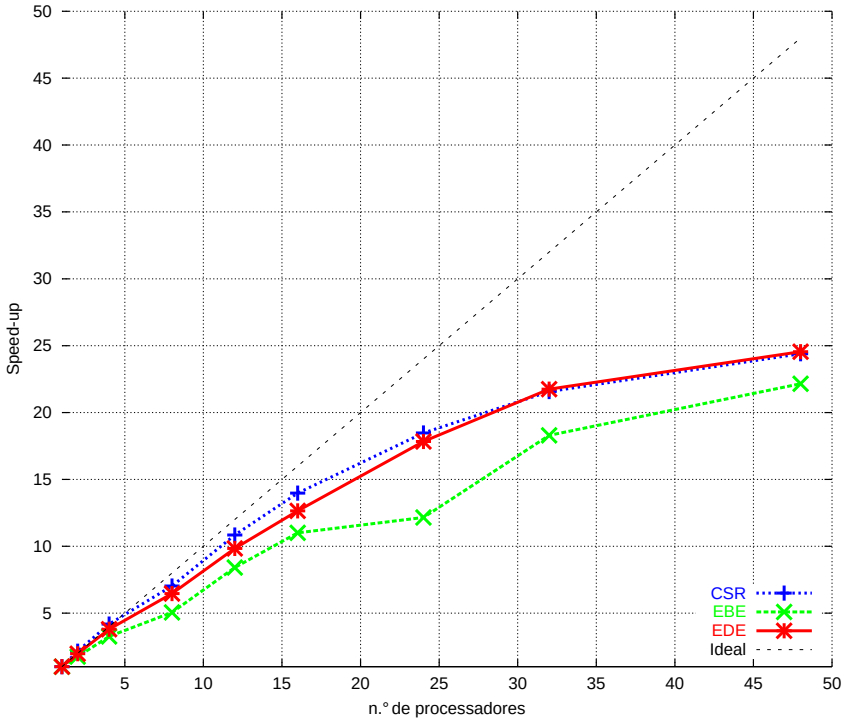


Figura 5.4: *Speedup* - Advecção em um campo de escoamento rotacional - Malha média

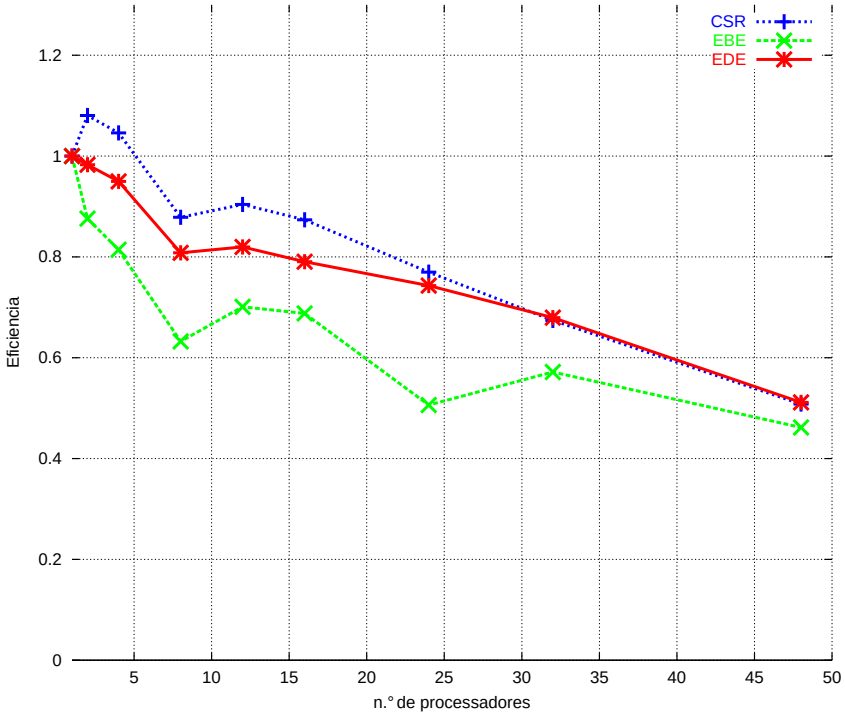


Figura 5.5: Eficiência - Advecção em um campo de escoamento rotacional - Malha média

de tamanho grande. Observa-se que a estratégia EDE obteve um melhor *speedup* e melhor eficiência em todos os pontos do gráfico. O *speedup* e a eficiência das estratégias CSR e EBE mantiveram-se quase iguais em todos os pontos.

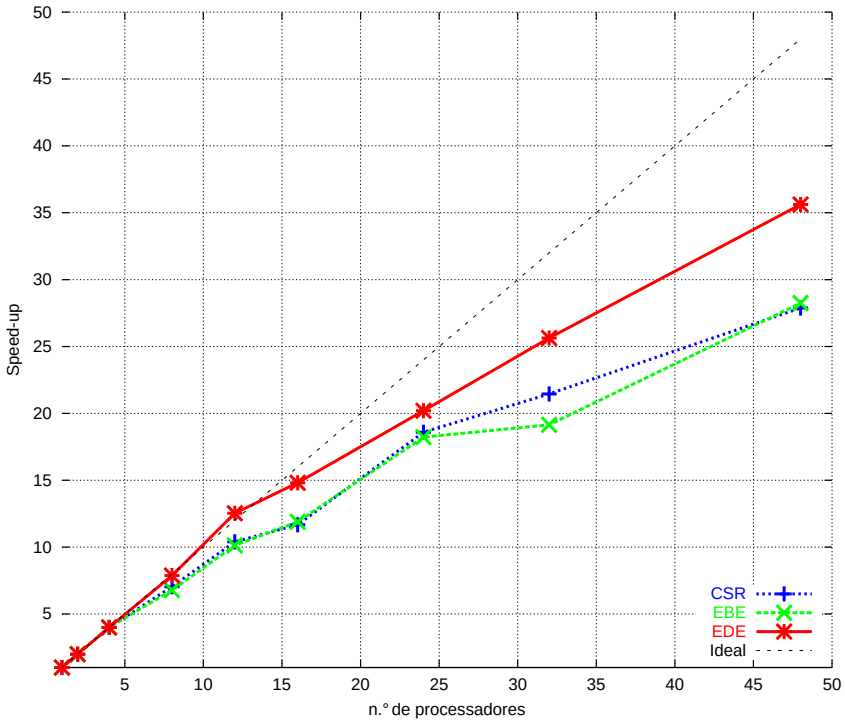


Figura 5.6: *Speedup* - Advecção em um campo de escoamento rotacional - Malha grande

5.4 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 rotação de um fluido ao longo de um cone no interior de um domínio quadrado com dimensões $[-5, 5] \times [-5, 5]$, conforme Fig. 5.8.

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 inicial é dada pela variação cossenoidal, mostrada na Fig. 5.8. A Fig. 5.9 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 e, executada em 4

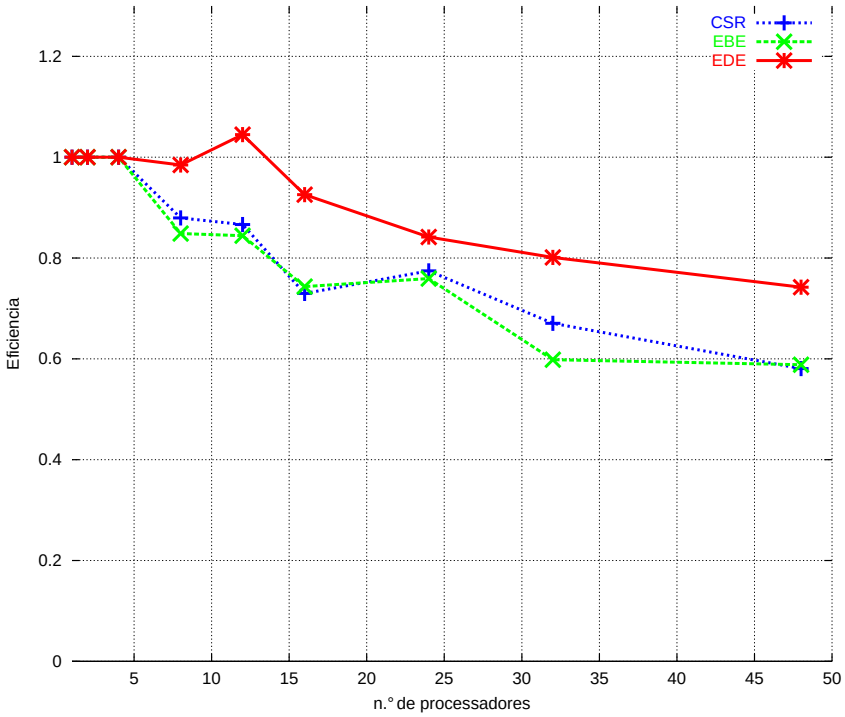


Figura 5.7: Eficiência - Advecção em um campo de escoamento rotacional - Malha grande

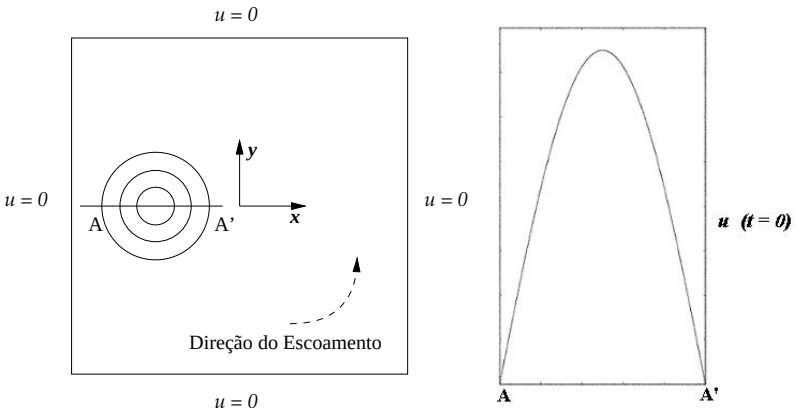
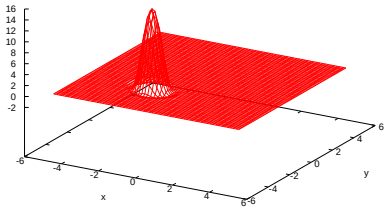
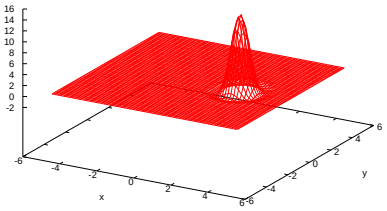


Figura 5.8: Cone em rotação - Descrição do problema.

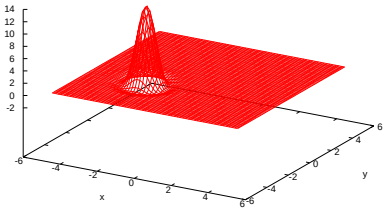
processadores. Como no exemplo anterior, a solução obtida considerando também as estratégias EBE e EDE, independente do número de processadores considerado, é essencialmente a mesma.



(a) $t = 1$ segundos.



(b) $t = 3$ segundos.



(c) $t = 7$ segundos.

Figura 5.9: Cone em rotação - Solução com 4 processadores utilizando a estratégia de armazenamento CSR.

Foram consideradas duas malhas não-estruturadas, uma malha média composta por 259620 nós e 517242 elementos triangulares e uma malha grande composta por 1033630 nós e 2065262 elementos triangulares. Para o caso sequencial o número de nós incógnitas da malha média é 259620 nós e para a malha grande é 1030634 nós.

A Tab. 5.3 apresenta, para a malha média, os tempos necessários para a solução do problema do cone em rotação considerando 1, 2, 4, 8, 12, 16, 24, 32 e 48 processadores e as três estratégias de armazenamento. Também nesse caso, apresenta para cada estratégia o correspondente número de objetos processados. Como no exemplo anterior, a estratégia CSR foi a mais rápida dentre as três.

A Tab. 5.4 apresenta, em relação à malha grande, os tempos necessários para a solução do problema do cone em rotação considerando 4, 8, 12, 16, 24, 32 e 48

Estratégia	Nº de objetos	Número de processadores								
		1	2	4	8	12	16	24	32	48
EBE	517238	17291	9113	4549	2220	1687	1084	659	555	442
EDE	772869	15694	7514	3596	2139	1165	823	535	446	373
CSR	1799202	10603	5195	2464	1214	756	550	382	341	297

Tabela 5.3: Cone em rotação - Tempo de execução em segundos - Malha média

processadores e as três estratégias de armazenamento, além do número de objetos processados em cada estratégia.

Estratégia	Nº de objetos	Número de processadores						
		4	8	12	16	24	32	48
EBE	2065262	22794	12459	7908	5981	4025	3159	2461
EDE	3091897	18853	9361	6137	4913	3153	2676	1702
CSR	7206092	12921	6553	4292	3471	2517	1725	1095

Tabela 5.4: Cone em rotação - Tempo de execução em segundos - Malha grande

As Fig. 5.10 e 5.11 apresentam o *speedup* e a eficiência refentes à malha média do problema do cone em rotação. Note que esse exemplo apresenta um comportamento superlinear para quase todos números de processados para as três estratégias, tendo como consequência uma eficiência superior a 1.0 nesses casos.

As Fig. 5.12 e 5.13 mostram o *speedup* e a eficiência refentes à malha grande do problema do cone em rotação. Os valores de *speedup* obtidos mantém um comportamento muito próximo do valor ideal. Na maior parte dos pontos o *speedup* referente a estratégia EDE supera os demais, contudo para o caso de 48 processadores nota-se que o melhor *speedup* é o referente à estratégia CSR. A eficiência apresentou um comportamento bem estável, somente para estratégia EBE com 12 processadores houve uma variação mais significativa.

Considerando o caso sequencial, foi observado que no caso de malhas não-estru-tu-radas, a lista dos elementos gerada pelo GMSH, em geral, não enumera elementos adjacentes mantendo-os numa sequência favorável ao uso da hierarquia de memória. Ou seja, no caso das malhas não-estruturadas alguns dados são dispostos de maneira

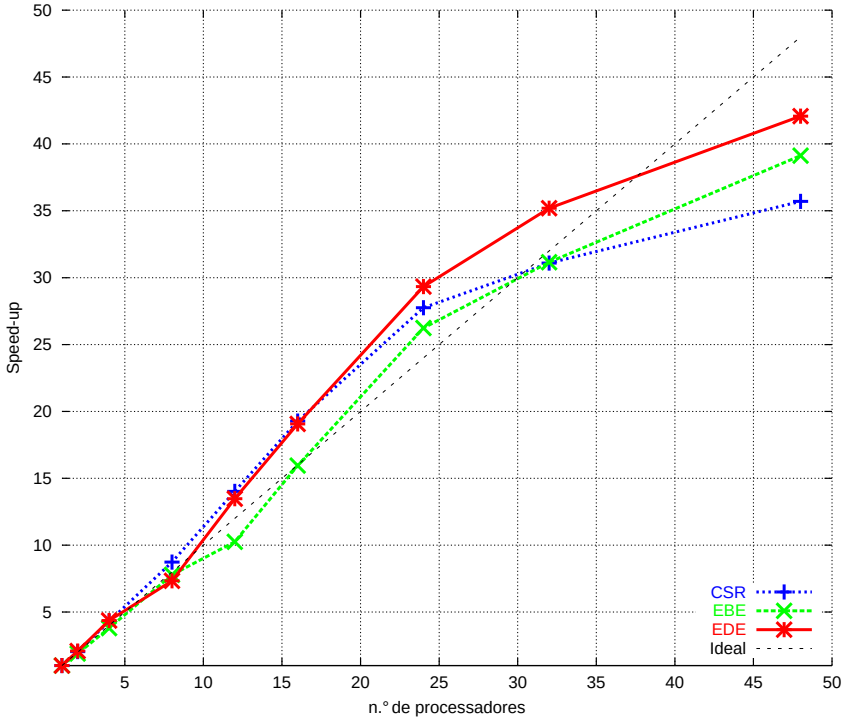


Figura 5.10: *Speedup* - Cone em rotação - Malha média

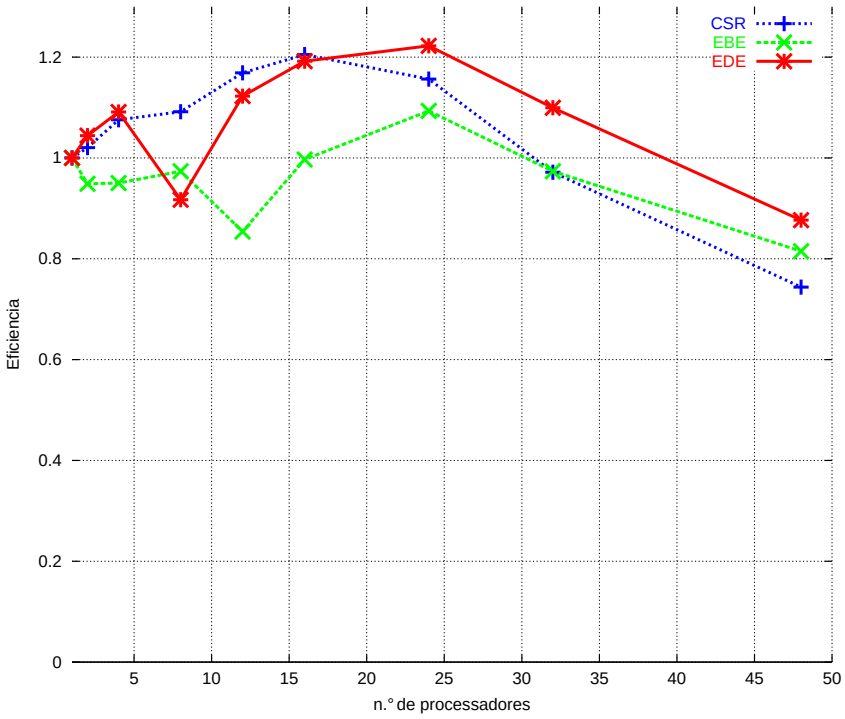


Figura 5.11: Eficiência - Cone em rotação - Malha média

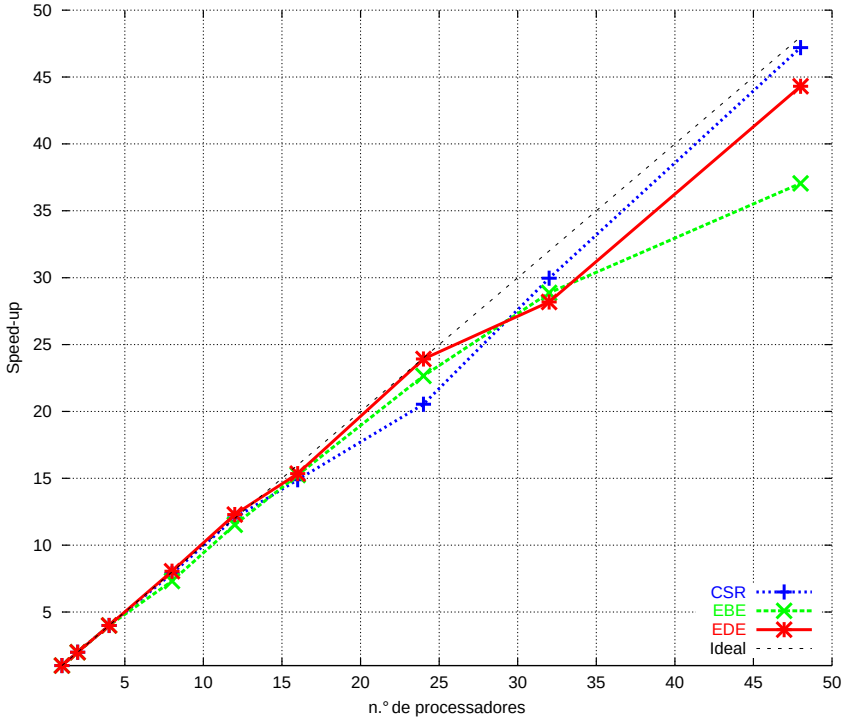


Figura 5.12: *Speedup* - Cone em rotação - Malha grande

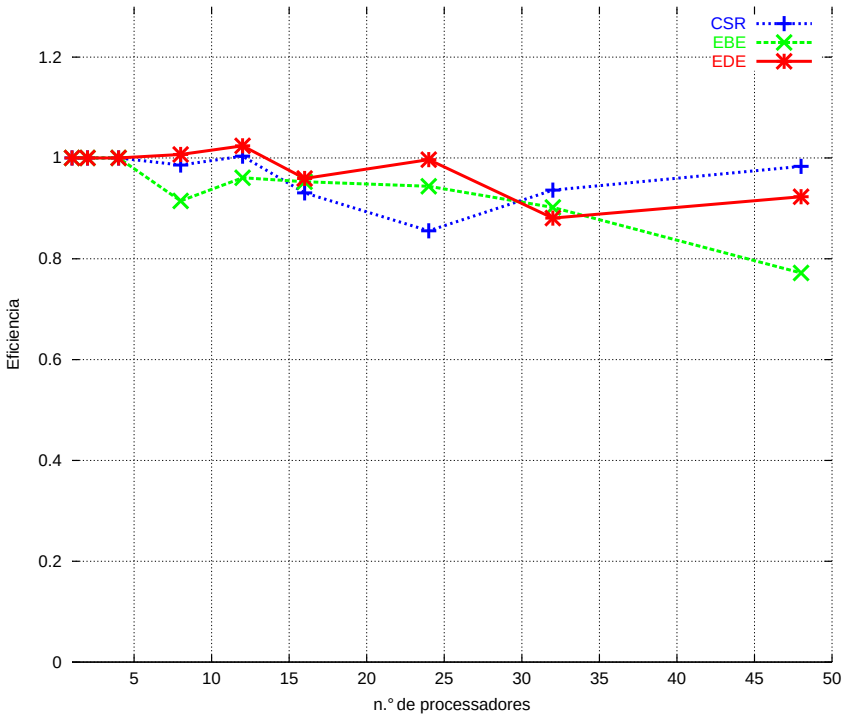


Figura 5.13: Eficiência - Cone em rotação - Malha grande

que não utilizem eficazmente os níveis L1 e L2 das memórias *cache*. Quando o problema é executado em paralelo, a malha é reduzida, e esse problema tem um efeito menor. Portanto, essa é uma possível explicação para valores de *speedups* superlineares verificados no problema exemplo da presente seção.

5.5 Um problema de advecção e difusão com solução conhecida

Também foi 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 tomamos $u(x,y) = 0.0$ e utilizamos f tal que a solução exata seja dada por $u(x,y) = 100xy(x-1)(y-1)$ para $(x,y) \in (0,1) \times (0,1)$. Foram utilizadas duas malha não-estruturadas: a primeira, uma malha média composta por 259324 nós e 516650 elementos triangulares e, a segunda, uma malha grande composta por 1034703 nós e 2065408 elementos triangulares. O número de nós incógnitas da malha média é 257328 e o número de nós incógnitas para a malha grande é 1030707. Foi utilizado o método GMRES com tolerância de 10^{-4} e 10 vetores na base de *Krylov*. A Fig. 5.14 representa a solução obtida utilizando a estratégia CSR, considerando uma malha estruturada com 64 células, contendo 2 elementos triangulares cada, executada em 4 processadores.

A Tab. 5.5 apresenta, em relação à malha média, os tempos necessários para a solução do problema de advecção e difusão com solução conhecida considerando 1, 2, 4, 8, 12, 16, 24, 32 e 48 processadores e as três estratégias de armazenamento, além do número de objetos processados em cada estratégia. Obedecendo a tendência sugerida pela Tab. 4.14, a estratégia CSR é a que apresenta um melhor desempenho no que se refere ao tempo de processamento, seguida da estratégia EDE e, por último, a estratégia EBE.

A Tab. 5.6 apresenta, em relação à malha grande, os tempos necessários para a solução do problema de advecção e difusão com solução conhecida considerando 4, 8, 12, 16, 24, 32 e 48 processadores e as três estratégias de armazenamento, além do número de objetos processados em cada estratégia. Note que nesse exemplo

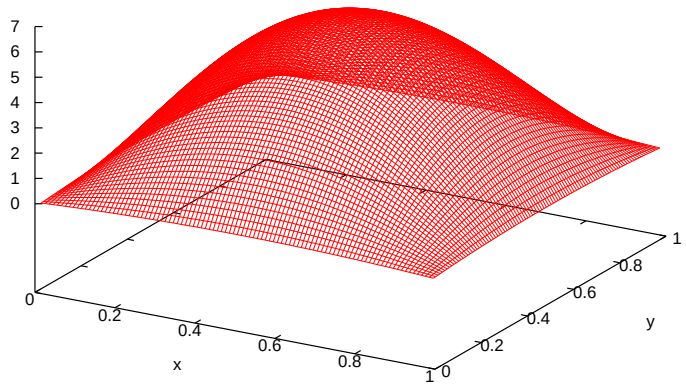


Figura 5.14: Advecção e difusão com solução conhecida - Solução com 4 processadores utilizando a estratégia de armazenamento CSR

Estratégia	**Nº de objetos**	**Número de processadores**								
		1	2	4	8	12	16	24	32	48
EBE	516646	27606	14214	6709	3195	1978	1367	961	705	582
EDE	771981	21361	10320	4811	2417	1418	1043	683	559	485
CSR	1797130	16114	7943	3760	1848	1187	855	607	507	458

Tabela 5.5: Advecção e difusão com solução conhecida - Tempo de execução em segundos - Malha média

os valores obtidos para as estratégias EBE e EDE estão nas proporcionalidades propostas pelo produto matriz-vetor na Tab. 4.14, ou seja, nesse caso, os tempos das estruturas EBE e EDE estão mantendo aproximadamente uma relação 3 : 2. Isso pode sugerir que nesse exemplo, ambas estruturas podem não estar tirando vantagem significativa das memórias *cache* e que o produto matriz-vetor está consumindo mais que 60% do tempo total de processamento.

Estratégia	**Nº de objetos**	**Número de processadores**						
		4	8	12	16	24	32	48
EBE	2065408	121228	60578	39667	30132	20244	16071	10474
EDE	3092116	88455	44351	31325	21863	14771	11586	6956
CSR	7206603	67880	33199	22466	17448	11218	8652	5828

Tabela 5.6: Adevcção e difusão com solução conhecida - Tempo de execução em segundos - Malha grande

As Fig. 5.15 e 5.16 apresentam os gráficos de *speedup* e eficiência do problema de advecção e difusão com solução conhecida em relação a malha de tamanho médio. Observa-se que os valores do *speedup* e eficiência das estratégias EBE e EDE superam os valores obtidos pela estratégia CSR, sugerindo que a estratégia CSR sequencial adequou-se melhor ao exemplo que as demais. Observe que, nos exemplos anteriores, os tempos sequenciais da estratégia CSR foram proporcialmente mais próximos dos tempos das estratégias EBE e EDE.

As Fig. 5.17 e 5.18 apresentam os gráficos de *speedup* e eficiência do problema de advecção e difusão com solução conhecida em relação a malha de tamanho grande. Observe que o *speedup* mantém-se próximo ao *speedup* ideal e a eficiência por sua vez próxima a 1. Note que para a estratégia EDE, o *speedup* para o caso de 48 processadores manteve-se acima do ideal, sugerindo que para um número maior de processadores a paralelização ainda é viável.

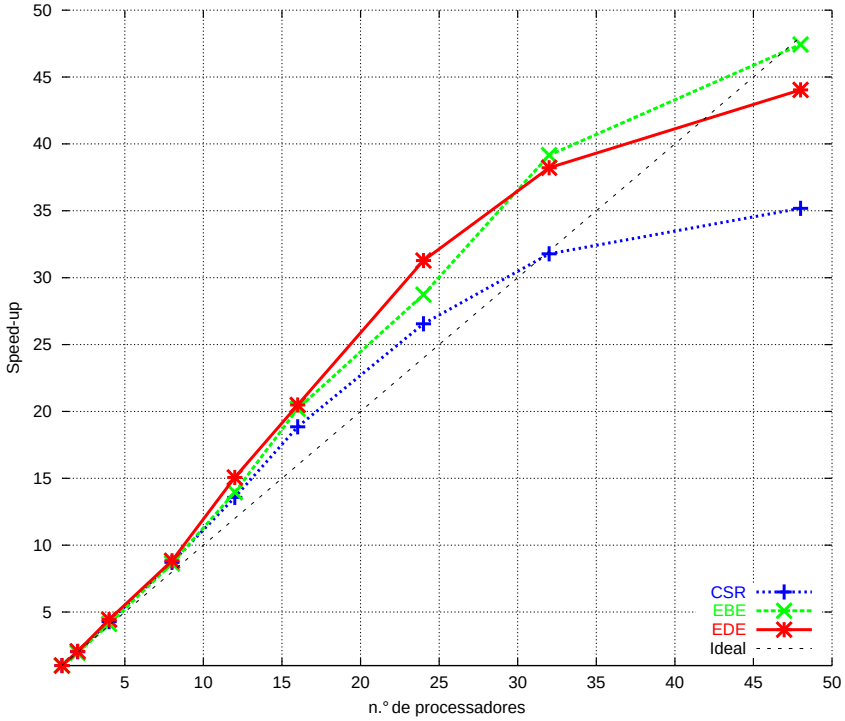


Figura 5.15: *Speedup* - Advecção e difusão com solução conhecida - Malha média

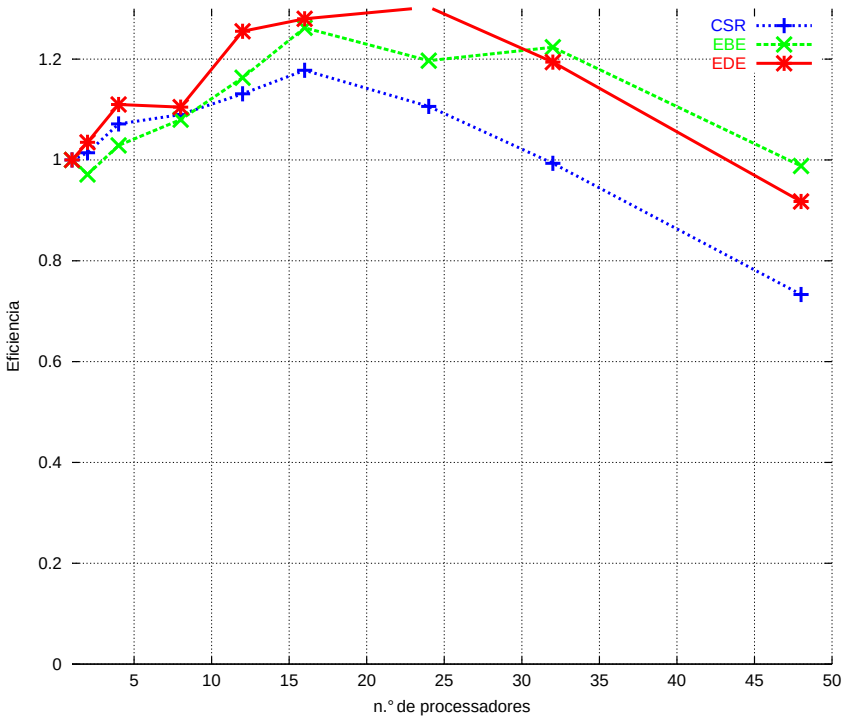


Figura 5.16: Eficiência - Advecção e difusão com solução conhecida - Malha média

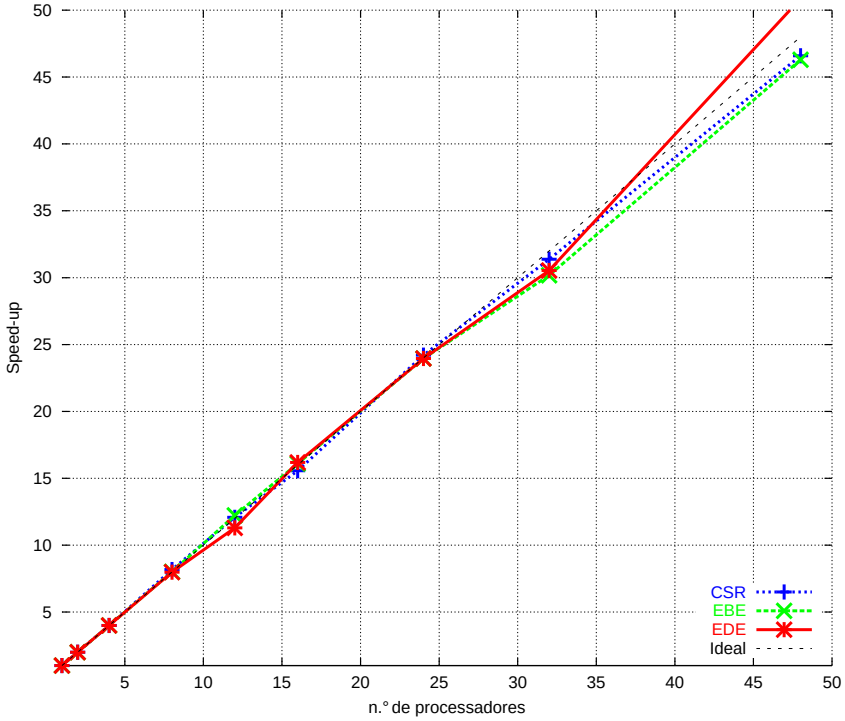


Figura 5.17: *Speedup* - Advecção e difusão com solução conhecida - Malha grande

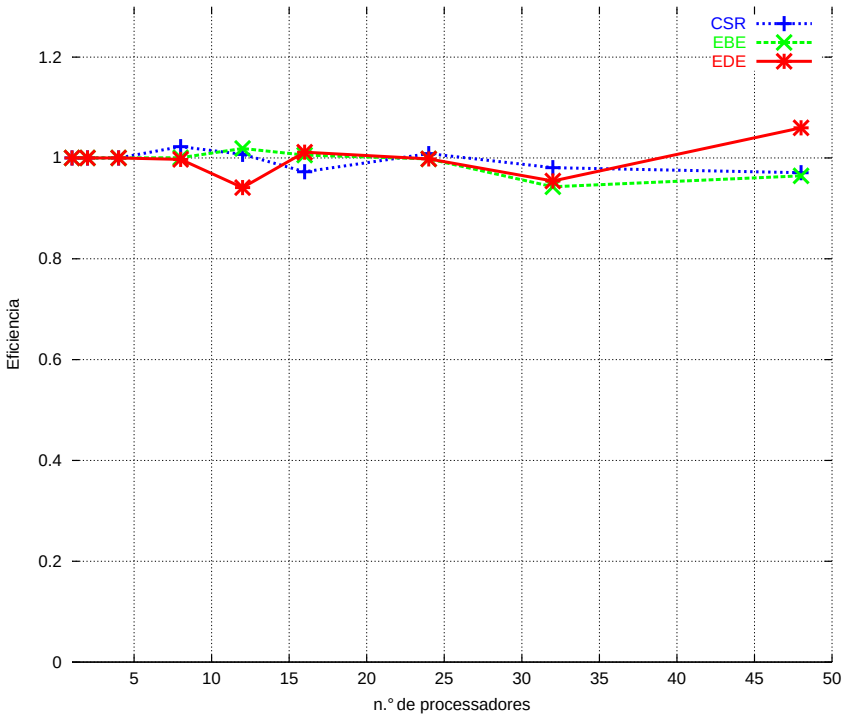


Figura 5.18: Eficiência - Advecção e difusão com solução conhecida - Malha grande

5.6 Uma equação diferencial parcial elíptica de 2ª ordem

Considere uma equação diferencial parcial elíptica de 2ª ordem que modela o seguinte problema de valor de contorno:

$$\begin{cases} -\nabla^2 u(x,y) = 0 \quad (x,y) \in \Omega \subset \mathbb{R}^2 \\ \\ u(x,y) = x + y \qquad (x,y) \in \partial\Omega \end{cases} \tag{5.5}$$

onde $\Omega = [-5,5] \times [-5,5]$ e, como pode-se perceber, é um problema com solução em regime permanente, matriz de difusividade volumétrica $\boldsymbol{k}$ igual a matriz identidade, campo de velocidade $\boldsymbol{\beta} = \{0,0\}^T$ e termo de fonte $f = 0$. Esse exemplo é o mesmo utilizado no Capítulo 3, diferindo apenas pela condição de contorno. A Fig. 5.19 representa a solução obtida utilizando a estratégia de armazenamento CSR, considerando uma malha estruturada com 100×100 células, contendo 2 elementos triangulares cada e, executada em 4 processadores. Também nesse caso, a solução obtida para qualquer uma das três estratégias de armazenamento é, independe do número de processadores considerado, é a mesma. Em todos os casos, são obtidos os mesmos resíduos e é realizado o mesmo número de iterações. Por se tratar de um problema simétrico, a solução do sistema linear proviniente da discretização por elementos finitos foi obtida através do algoritmo Gradientes Conjugados, com uma tolerância de 10^{-12}.

Foram consideradas duas malhas estruturadas, uma média com 512×512 células e outra grande com 1024×1024 células, sendo cada célula dividida em dois elementos triangulares, totalizando 263169 nós e 524288 elementos para a primeira malha e 1050625 nós e 2097152 elementos para segunda malha. O número de nós incógnitas para a malha média é 261121 nós e para a grande é 1046529 nós.

A Tab. 5.1 apresenta, para a malha média, os tempos necessários para a solução do problema de valor de contorno regido pela equação elíptica de 2ª ordem considerando 1, 2, 4, 8, 12, 16, 24, 32 e 48 processadores e as três estratégias de armazenamento. Também apresenta para cada estratégia o correspondente número de objetos processados. Observa-se que os tempos de processamento tiveram uma ordem de grandeza bem pequena, visto que nenhum deles ultrapassou 300 segundos. No entanto, a classificação das estratégias segundo o tempo de processamento manteve-se como nos caso anteriores, ou seja, a mais rápida é a CSR seguida da

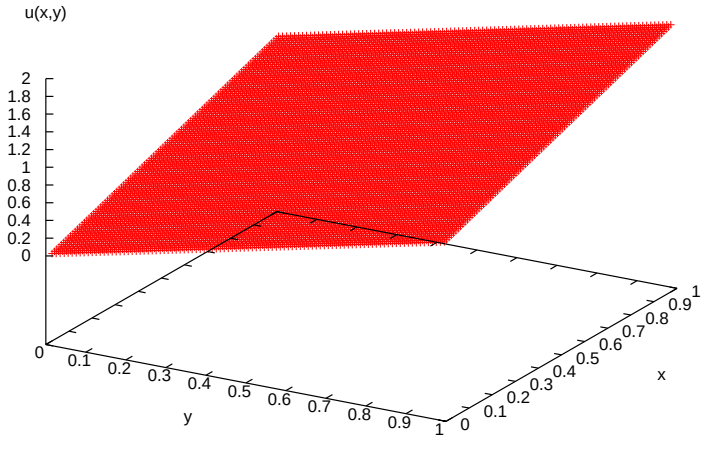


Figura 5.19: Equação elíptica de 2ª ordem - Solução com 4 processadores utilizando a estratégia de armazenamento CSR

estratégia EDE e depois a estratégia EBE.

Estratégia	**Nº de objetos**	**Número de processadores**								
		1	2	4	8	12	16	24	32	48
EBE	524286	290	181	94	54	36	28	20	16	13
EDE	783360	280	144	73	39	26	20	14	12	10
CSR	1823761	213	114	58	31	22	17	12	10	8

Tabela 5.7: Equação elíptica de 2ª ordem - Tempo de execução em segundos - Malha média

Tab. 5.6 apresenta, em relação à malha grande, os tempos necessários para a solução do problema de valor de contorno regido pela equação elíptica de 2ª ordem considerando 4, 8, 12, 16, 24, 32 e 48 processadores e as três estratégias de armazenamento, além número de objetos processados em cada estratégia. Semelhantemente aos exemplos anteriores observa-se que a estratégia CSR foi a mais rápida, seguida pela estratégia EDE e depois estratégia EBE.

As Fig. 5.20 e 5.21 apresentam os gráficos de *speedup* e eficiência do problema de valor de contorno regido pela equação elíptica de 2ª ordem em relação a malha

Estratégia	Nº de objetos	Número de processadores						
		4	8	12	16	24	32	48
EBE	2097150	649	336	254	180	128	100	71
EDE	3139584	566	291	200	153	108	85	63
CSR	7317521	457	242	162	122	86	69	51

Tabela 5.8: Equação elíptica de 2ª ordem - Tempo de execução em segundos - Malha grande

de tamanho médio. Como pode ser observado a estratégia EDE obteve um maior *speedup* e maior eficiência para todos os números processadores, a estratégia CSR manteve um comportamento semelhante com valores logo abaixo da estratégia EDE.

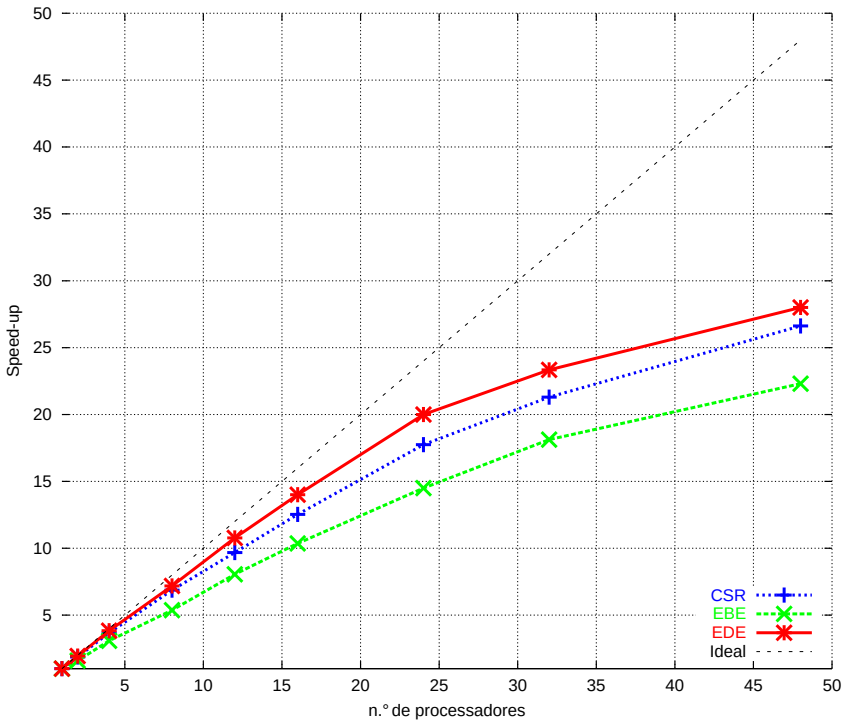


Figura 5.20: *Speedup* - Equação elíptica de 2ª ordem - Malha média

As Fig. 5.22 e 5.23 apresentam os gráficos de *speedup* e eficiência do problema de valor de contorno regido pela equação elíptica de 2ª ordem em relação a malha de tamanho grande. Em relação aos valores de *speedup* e eficiência as três estratégias obtiveram valores quase idênticos, com uma leve queda da eficiência para estratégia EBE para alguns números de processadores.

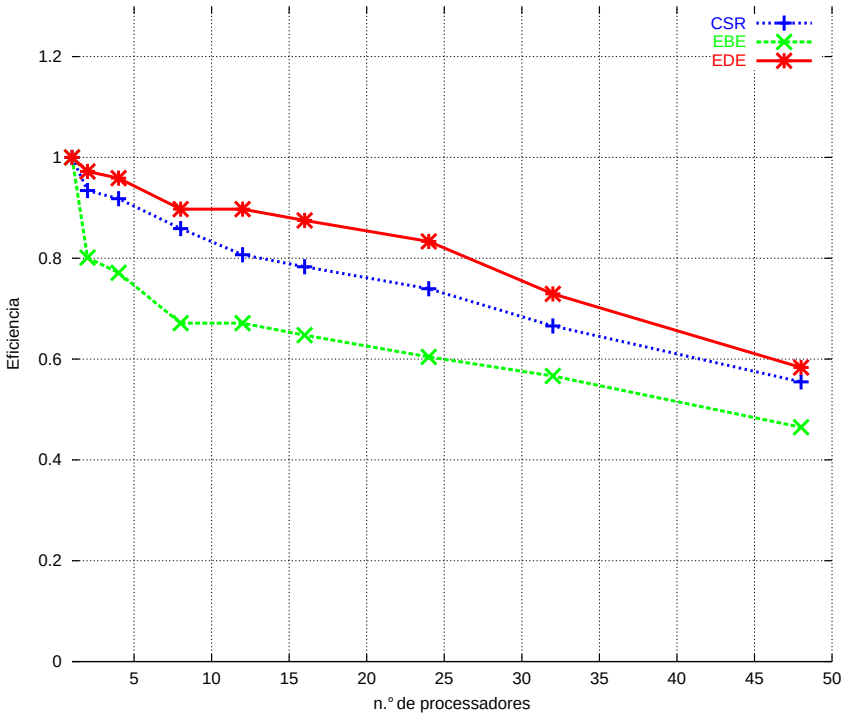


Figura 5.21: Eficiência - Equação elíptica de 2ª ordem - Malha média

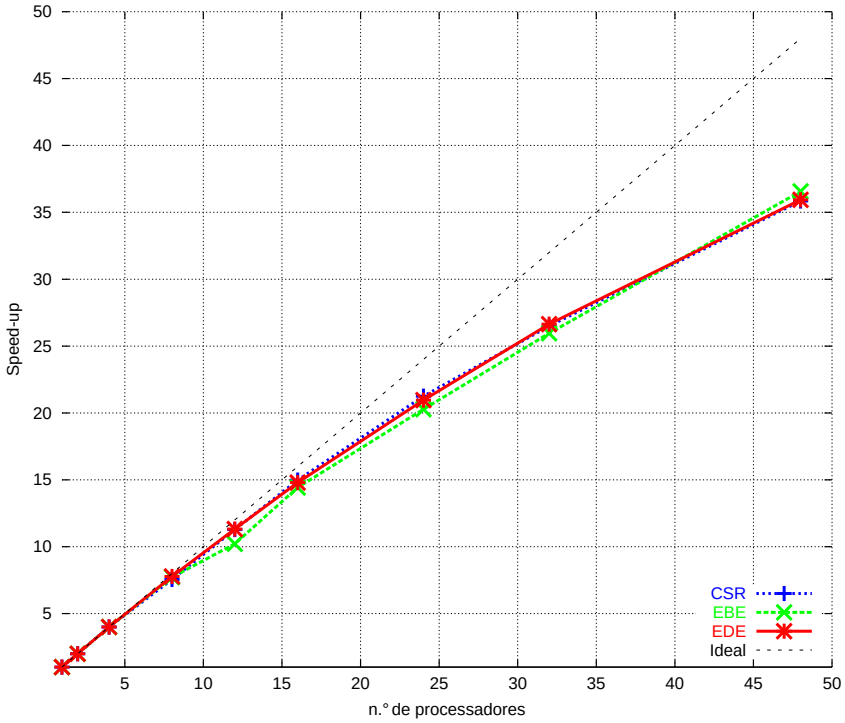


Figura 5.22: *Speedup* - Equação elíptica de 2ª ordem - Malha grande

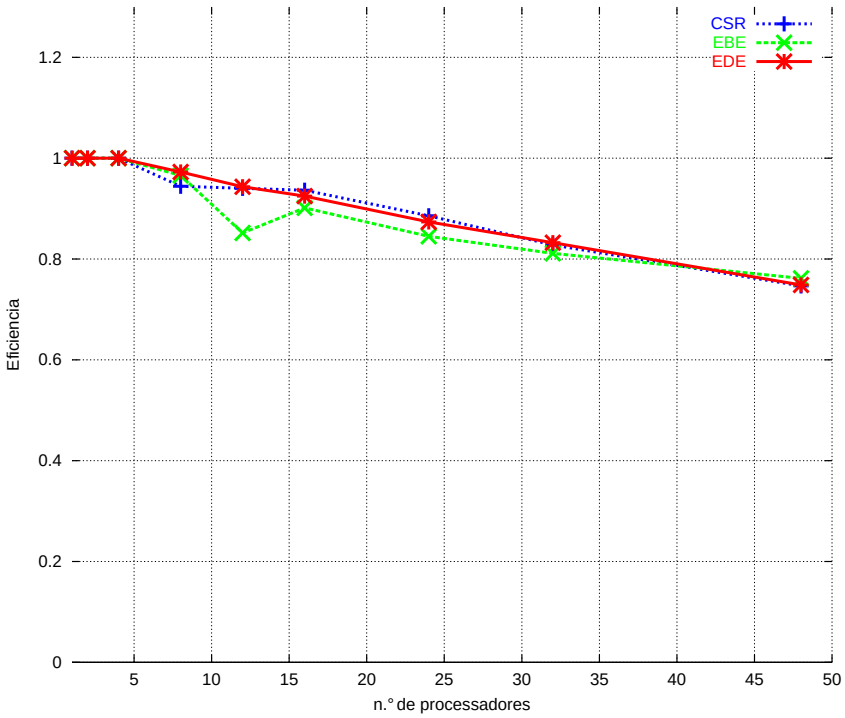


Figura 5.23: Eficiência - Equação elíptica de 2ª ordem - Malha grande

Capítulo 6

Conclusões

Neste trabalho foi realizado um estudo comparativo entre as estratégias de armazenamento elemento-por-elemento (EBE) [18], aresta-por-aresta (EDE) [7][28] e *compressed sparse row* (CSR) [35] aplicadas ao processamento paralelo em *clusters* de estações de trabalho. As referidas estratégias armazenaram as contribuições relativas às matrizes de discretização oriundas da equação de advecção e difusão discretizada pelo método dos elementos finitos com uma estabilização SUPG. Foi estudada uma técnica de decomposição de domínio onde as matrizes de discretização foram armazenadas em estruturas de dados que possibilitam resolver os sistemas lineares provenientes do processo de discretização através do algoritmo GMRES e gradientes conjugados. Antes do processamento paralelo ser efetuado foi realizada uma etapa de pré-processamento, onde a malhas de discretização foram geradas pelo GMSH e, com a ajuda do software METIS, o domínio foi particionado.

Observou-se, durante o decorrer dos estudos, que para as malhas com uma quantidade de nós em torno de 10000, a estrutura paralela já ultrapassava uma quantidade de 256MB na memória principal, o que tornava o processo de solução do problema de difusão simples praticamente inviável. Sendo assim, constatou-se a necessidade da implementação das estratégias de armazenamento especiais devido a grande esparsidade encontrada. Foi realizado um estudo teórico sobre as estratégias de armazenamento, onde foram investigados o consumo de memória e o número de operações realizadas em cada estratégia. Os resultados teóricos demonstraram que a estratégia CSR é a mais rápida dentre as três, seguidas pela estratégia EDE.

Para medir o desempenho das estratégias de armazenamento implementadas, escolheu-se quatro problemas exemplo. O primeiro exemplo foi um problema de

advecção em um campo de escoamento rotacional. Nesse exemplo foram utilizadas malhas estruturadas e os gráficos de *speedup* e eficiência apresentaram maiores valores para estratégia EDE na maior parte dos casos. Em relação ao tempo de processamento, os melhores tempos foram obtidos pela estratégia CSR.

O segundo problema apresentado foi o problema do cone em rotação e o terceiro foi um problema de advecção e difusão com solução conhecida. Ambos foram testados utilizando dois tamanhos de malhas não-estruturadas. Os gráficos de *speedup* e eficiência para os dois exemplos apresentaram um comportamento atípico, uma vez que, boa parte do valores obtidos envolviam *speedup* superlinear e é claro eficiência acima de 1. Também nesses casos o tempo de processamento da estratégia CSR foi o menor.

O quarto problema foi uma equação diferencial parcial elíptica de 2ª ordem. Diferente dos outros três exemplos, esse problema foi resolvido utilizando o método dos gradientes conjugados, sendo executado em dois tamanhos de malha estruturada. Assim como no primeiro problema, os gráficos de *speedup* e eficiência apresentaram maiores valores para a estratégia de EDE, seguida da estratégia CSR.

Em relação aos valores de *speedup* e eficiência obtidos, concluíu-se que os exemplos executados em malhas não-estruturadas obtiverem valores bem maiores que aqueles em exemplos de malhas estruturadas. Isso sugere que, se os elementos de uma malha não-estruturada não forem enumerados mantendo uma certa sequência, as vantagens proporcionadas pelos níveis de memória *cache* podem não ser bem aproveitadas.

Como não é possível comparar as estratégias de armazenamento entre si utilizando como referência o *speedup* e a eficiência, resta apenas comparar os valores obtidos no tempo de processamento de cada estratégia. A estratégia CSR foi a mais rápida em todos os testes realizados. No entanto, a estratégia EDE pode ser classificada como aquela que melhor adequou-se ao processo de paralelização, por apresentar maiores valores de *speedup* e eficiência na maior parte dos casos, comparada com seu próprio caso sequencial.

Um aspecto importante a ser considerado diz respeito a precisão da técnica de paralelização empregada. Foi verificado que o número de iterações de cada problema exemplo e o resíduo a solução foram exatamente os mesmos dentro da precisão estabelecida, independentemente do número de processadores, das estratégias de

armazenamento e do método de solução do sistema linear. 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 paralelo é obter rigorosamente os mesmos resultados dos algoritmos sequenciais equivalentes.

Como trabalhos futuros, são sugeridos implementações de pré-condicionadores para o algoritmo GMRES, e gradientes conjugados. Em relação ao pré-processamento, utilizar a versão paralela do METIS, denominada ParMETIS. Estender os conceitos apresentados no presente trabalho aos casos tri-dimensionais, a equações mais complexas e a problemas envolvendo mais de 1 grau de liberdade por nó.

Referências Bibliográficas

[1] T. Agerwala, J. L. Martin, J. H. Mirza, D. C. Sadler, and D. M. Dias. Sp2 system architecture. *IBM Systems Journal*, 1995.

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

[3] B. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, Englewood Cliffs, 1989.

[4] J. Blazewicz, M. Drabowski, and J. Weglarz. Scheduling multiprocessor tasks to minimize schedule length. 35(5):389–393, 1986.

[5] A. N. Brooks and T. J. R. Hughes. Streamline upwind/petrov-galerkin formulations for convection dominated flows with particular emphasis on the incompressible navier-stokes equations. *Comput. Methods Appl. Mech. and Engrg.*, 32:199–259, 1982.

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

[7] 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.

[8] R. Cypher and E. Leu. The semantics of blocking and nonblocking send and receive primitives. *Parallel Processing Symposium*, pages 729–735, 1994.

[9] S. J. Eggers, J. Emer, H. M. Levy, J. L. Lo, R. Stamm, and D. M. Tullsen. Simultaneous multithreading: A platform for next-generation processors. *IEEE Micro*, 17:12–19, 1997.

[10] R. N. Elias, A. L. G. A. Coutinho, M. A. D. Martins, and R. M. Sydenstricker. Parallel inexact newton-type methods for the supg/pspg solution of steady incompressible 3d navier-stokes equations in pc clusters. In *XXV CILAMCE - Iberian American Congress on Computational Methods in Engineering*, 2004.

[11] J. A. Fisher and J. J. O'Donnell. Vliw machines: Multiprocessors we can actually program. *CompCon 84*, pages 299–305, 1984.

[12] M. J. Flyn. Very high-speed computing systems. *IEEE*, 54:1901–1909, 1966.

[13] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM Parallel Virtual Machine, A User's Guide and Tutorial for Networked Parallel Computing*. MIT Press, Cambridge, Mass., 1994.

[14] K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta, and J. Hennessy. Memory consistency and event ordering in scalable shared-memory multiprocessors. *17th Annual International Symposium on Computer Architecture*, pages 15–26, 1990.

[15] S. B. Gowda. A comparison of sparse and element-by-element storage schemes on the efficiency of parallel conjugate gradient iterative methods for finite element analisys. Master's thesis, Clemson University, August 2002.

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

[17] High Performance Fortran Forum. High Performance Fortran language specification, version 1.0. Technical Report CRPC-TR92225, Houston, Tex., 1993.

[18] T. J. R. Hughes. *The Finite Element Method - Linear Static and Dynamic Finite Element Analysis*. Prentice-Hall International, Inc., 1987.

[19] T. J. R. Hughes and T. E. Tezduyar. Finit element methods for first-order hyperbolic systems with particular emphasis on the compressible euler equations. In *Computational Methods Applied Mechanics and Engineering.*, volume 45, pages 217–284, 1984.

[20] 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.

[21] R. H. Halstead Jr. and T. Fujita. Masa: a multithreaded processor architecture for parallel symbolic computing. *15th Annual International Symposium on Computer architecture*, (02):443–451, 1988.

[22] A. Kaceniauskas and P. Rutschmann. Parallel fem software for cfd problemas. Technical Report 3, Institution of Mathematics and Informatics, 2004.

[23] G. Karypis and V. Kumar. A software package for partitioning unstructure graphs, partitioning meshes, and computing fill-orderings of sparce matrices. Technical report, University of Minnesota, Department of Computer Science, 1998.

[24] P. Keleher, S. Dwarkadas, A. L. Cox, and W. Zwaenepoel. Treadmarks: Distributed shared memory on standard workstations and operating systems. In *Proc. of the Winter 1994 USENIX Conference*, pages 115–131, 1994.

[25] D. E. Knuth. *The art of computer programming: sorting and searching*, volume 3. Addison Wesley, second edition, 1998.

[26] M. E. Levitt. Designing ultrasparc for testability. In *IEEE Design & Test of Computers*, pages 740–747, 1997.

[27] Y. Luo. Mpi performance study on the sgi origin 2000. Technical report, Los Alamos National Laboratory, 1997.

[28] M. A. D. Martins. Solução iterativa em paralelo de sistemas de equações do método dos elementos finitos empregando estruturas de dados por arestas. Master's thesis, COPPE/UFRJ, Rio de Janeiro, RJ, Agosto 1996.

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

[30] D. I. Moldavan. *Parallel Processing: From Applications to Systems.* Morgan Kaufmann Publishers, San Mateo, California, 1993.

[31] 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.

[32] S. Palacharla, N. P. Jouppi, and J. E. Smith. Complexity-effective superscalar processors. *24th annual international symposium on Computer architecture*, 25(2):206–218, 1997.

[33] C. V. Ramamoorthy and H. F. Li. Pipeline architecture. *ACM Computing Surveys*, 9(1):61–102, 1977.

[34] F. L. B. Ribeiro and A. L. G. A. Coutinho. Comparison between element, edege and csr storage schemes for iterative solutions in finite element analyses. In *IJNME*, 2004.

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

[36] Y. Saad and H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM. J. Sci. Stat. Comput.*, 7(3):856–869, 1986.

[37] S. N. Simões. Uma comparação entre um algoritmo síncrono e um parcialmente assícrono para solução de problemas de mecânica dos fluidos. Master's thesis, Universidade Federal do Espírito Santo - UFES, Vitória, ES, Setembro 2004.

[38] I. Slobodcicov. Implementação em paralelo do método dos elementos finitos para as equações de águas rasas. Master's thesis, COPPE/UFRJ, Rio de Janeiro, RJ, Março 2003.

[39] W. Stallings. *Reduced Instruction Set Computer (RISC)*. IEEE Computer Society, Washington, D. C., 2nd edition, 1990.

[40] H. Stern, M. Eisler, and R. Labiaga. *Managing NFS and NIS*. O'Reilly & Associates, Inc., second edition, 2001.

[41] Sun. *Sun Grid Engine: Enterprise Edition 5.3 Administration User's Guide*. Sun Microsystems, 2002.

[42] T. E. Tezduyar, J. Lion, and M. Behr. 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*, 94(3):339–351, 1992.