UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA CIVIL

COMPUTAÇÃO PARALELA NA ANÁLISE DE PROBLEMAS DE ENGENHARIA UTILIZANDO O MÉTODO DOS ELEMENTOS FINITOS

JOÃO RICARDO MASUERO

Porto Alegre
2009

JOÃO RICARDO MASUERO

COMPUTAÇÃO PARALELA NA ANÁLISE DE PROBLEMAS DE ENGENHARIA UTILIZANDO O MÉTODO DOS ELEMENTOS FINITOS

Tese apresentada ao Programa de Pós-Graduação em Engenharia Civil da Universidade Federal do Rio Grande do Sul, como parte dos requisitos para obtenção do título de Doutor em Engenharia.

Orientação: Prof. Dr. ARMANDO MIGUEL AWRUCH

Porto Alegre

2009

M424c Masuero, João Ricardo

Computação paralela na análise de problemas de engenharia utilizando o Método dos Elementos Finitos / João Ricardo Masuero. – 2009.

Tese (doutorado) – Universidade Federal do Rio Grande do Sul. Escola de Engenharia. Programa de Pós-Graduação em Engenharia Civil. Porto Alegre, BR-RS, 2009.

Orientação: Prof. Dr. Armando Miguel Awruch

1. Elementos finitos. 2. Dinâmica dos sólidos computacional. 3. Dinâmica dos fluidos computacional. 4. Estruturas (Engenharia). I. Awruch, Armando Miguel, orient. II. Título.

CDU-624.04(043)

JOÃO RICARDO MASUERO

COMPUTAÇÃO PARALELA NA ANÁLISE DE PROBLEMAS DE ENGENHARIA UTILIZANDO O MÉTODO DOS ELEMENTOS FINITOS

Esta tese de doutorado foi julgada adequada para a obtenção do título de DOUTOR EM ENGENHARIA, Área de Concentração Estruturas, e aprovada em sua forma final pelo orientador e pelo Programa de Pós-Graduação em Engenharia Civil da Universidade Federal do Rio Grande do Sul.

Orientação: Prof. Dr. Armando Miguel Awruch

Porto Alegre, 20 de março de 2009

Prof. Armando Miguel Awruch
Dr. Pela Universidade Federal do Rio de Janeiro
Orientador

Prof. Fernando Schnaid
Coordenador do PPGEC/UFRGS

BANCA EXAMINADORA

Prof. José Alberto Cuminato
Dr. pela Universidade de Oxford/ UK(USP)

Prof. Philippe Olivier Alexandre Navaux
Dr. Pelo Institut National Polytechnique de Grenoble/Fr (UFRGS)

Prof[a]. Adriane Prisco Petry
Dr.[a]. pela Universidade Federal do Rio Grande do Sul (UFRGS)

Prof. Inácio Benvegnu Morsh
Dr. pela Universidade Federal do Rio Grande do Sul (UFRGS)

À minha família.
Por todo amor e apoio.

AGRADECIMENTOS

Gostaria de agradecer inicialmente ao prof. Armando Miguel Awruch, pelo entusiasmo, companheirismo e apoio com que me orientou ao longo desses muitos anos. E também pela pressão, gentil e bem humorada, mas firme e constante, sem a qual eu certamente não chegaria ao fim deste trabalho.

Ao colegas Gustavo Bono, Alexandre Luis Braun e Martin Poulsen Kessler, pela valiosa ajuda para resolver diversas dúvidas sobre as formulações abordadas, por permitirem que eu trabalhasse sobre os códigos por eles desenvolvidos e por me fornecerem exemplos adequados aos objetivos do meu trabalho, sem os quais teria sido impossível realizar as comparações e testes necessários.

Ao meus colegas do Departamento de Engenharia Civil da UFRGS, pelo incentivo. Especialmente ao colega e amigo Inácio Benvegnú Morsch, que nunca deixou de demonstrar de forma concreta seu apoio e que, como chefe do DECIV, fez o possível para eu pudesse dedicar o máximo de tempo a este trabalho.

Ao pessoal da secretaria do PPGEC: Liliani, Carmen e Ana Luiza, pela eficiência e pelo carinho.

Ao Departamento de Engenharia Civil, pelas valiosas horas de trabalho disponibilizadas para o doutoramento.

Ao Programa de Pós-Graduação em Engenharia Civil, por todo o apoio recebido.

Às agências de fomento CAPES e CNPq, sem cujo apoio boa parte da pesquisa no país não seria possível, esta inclusive.

Aos meus pais, Ítalo e Maria do Carmo, por me darem os valores e o amor necessários para me tornar aquilo que hoje eu sou.

Aos meus filhos, Guilherme e Carolina, pelo amor irrestrito e incondicional, e pela compreensão pelas muitas e muitas horas que não pude estar com eles em função “da tese”.

E finalmente, à minha esposa Ângela, pelo amor, pelo companheirismo, pela compreensão, pelo carinho, pelo apoio e por me fazer uma pessoa melhor.

RESUMO

MASUERO, J.R. Computação paralela na análise de problemas de engenharia utilizando o Método dos Elementos Finitos. 2009. Tese (Doutorado em Engenharia) – Programa de Pós-Graduação em Engenharia Civil, UFRGS, Porto Alegre

O objetivo deste trabalho é estudar algoritmos paralelos para a solução de problemas de Mecânica dos Sólidos, Mecânica dos Fluídos e Interação Fluido-Estrutura empregando o Método dos Elementos Finitos para uso em configurações de memória distribuída e compartilhada. Dois processos para o particionamento da estrutura de dados entre os processadores e divisão de tarefas foram desenvolvidos baseados na aplicação do método de particionamento em faixas e do método da bissecção coordenada recursiva não sobre a geometria da malha mas sim diretamente sobre o sistema de equações, através de reordenações nodais para minimização da largura da banda. Para ordenar a comunicação entre os processadores, foi desenvolvido um algoritmo simples e genérico baseado em uma ordenação circular e alternada que permite a organização eficiente dos processos mesmo em cenários nos quais cada processador precisa trocar dados com todos os demais. Os algoritmos selecionados foram todos do tipo iterativo, por sua adequabilidade ao paralelismo de memória distribuída. Foram desenvolvidos códigos paralelos para o Método dos Gradientes Conjugados utilizado em problemas de Mecânica dos Sólidos, para o esquema explícito de Taylor-Galerkin com um passo e iterações utilizado na simulação de escoamentos compressíveis em regime transônico e supersônico, para o esquema explícito de Taylor-Galerkin com 2 passos para simulação de escoamentos incompressíveis em regime subsônico e para interação fluído-estrutura usando o esquema explícito de dois passos para o fluído e o método implícito de Newmark no contexto do método de estabilização α-Generalizado para a estrutura, com acoplamento particionado. Numerosas configurações foram testadas com problemas tridimensionais utilizando elementos tetraédricos e hexaédricos em *clusters* temporários e permanentes, homogêneos e heterogêneos, com diferentes tamanhos de problemas, diferentes números de computadores e diferentes velocidades de rede.

Palavras-chave: Computação Paralela, Dinâmica dos Sólidos Computacional, Dinâmica dos Fluídos Computacional, Método dos Elementos Finitos, Computação de Alto Desempenho

ABSTRACT

MASUERO, J.R. Computação paralela na análise de problemas de engenharia utilizando o Método dos Elementos Finitos. 2009. Tese (Doutorado em Engenharia) – Programa de Pós-Graduação em Engenharia Civil, UFRGS, Porto Alegre

Analysis and development of distributed memory parallel algorithms for the solution of Solid Mechanics, Fluid Mechanics and Fluid-Structure Interaction problems using the Finite Element Method is the main goal of this work. Two process for mesh partitioning and task division were developed, based in the Stripwise Partitioning and the Recursive Coordinate Bisection Methods, but applied not over the mesh geometry but over the resultant system of equations through a nodal ordering algorithm for system bandwidth minimization. To schedule the communication tasks in scenarios where each processor must exchange data with all others in the cluster, a simple and generic algorithm based in a circular an alternate ordering was developed. The algorithms selected to be parallelized were of iterative types due to their suitability for distributed memory parallelism. Parallel codes were developed for the Conjugate Gradient Method (for Solid Mechanics analysis), for the explicit one-step scheme of Taylor-Galerkin method (for transonic and supersonic compressible flow analysis), for the two-step explicit scheme of Taylor-Galerkin method (for subsonic incompressible flow analysis) and for a Fluid-Structure Interaction algorithm using a coupling model based on a partitioned scheme. Explicit two-step scheme of Taylor-Galerkin were employed for the fluid and the implicit Newmark algorithm for the structure. Several configurations were tested for three-dimensional problems using tetrahedral and hexahedral elements in uniform and non-uniform clusters and grids, with several sizes of meshes, numbers of computers and network speeds.

Key-words: Parallel Computing, Computational Solid Dynamics, Computational Fluid Dynamics, Finite Element Method, High Performance Computing

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS, SIGLAS E SÍMBOLOS

Letras Romanas Maiúsculas

Letras Romanas Minúsculas

Letras Gregas Maiúsculas

Letras Gregas Minúsculas

1 INTRODUÇÃO

A análise numérica de problemas de engenharia é uma das áreas de maior demanda por capacidade computacional. Os modelos numéricos empregados geralmente implicam na solução de sistemas que alcançam milhões de equações, necessitando para isso de grandes velocidades de processamento, e a manipulação e armazenamento das enormes estruturas de dados correspondentes exigem grande quantidade de memória local e secundária. Quando os problemas envolvidos são transientes, a necessidade de simulação dos fenômenos na escala de tempo real faz com que grandes recursos computacionais sejam empregados por longos períodos de tempo.

Apesar dos altos custos envolvidos, a simulação computacional de fenômenos físicos complexos em modelos com características e geometria detalhadas é mais barata e mais flexível que a análise experimental dos mesmos problemas, sendo, portanto, utilizada em larga escala no projeto e desenvolvimento de sistemas de energia, aeroespaciais, estruturais, bioquímicos e meteorológicos, complementando e substituindo parcialmente a análise experimental.

O emprego do Método dos Elementos Finitos na simulação de problemas científicos e de engenharia nas áreas de Dinâmica dos Sólidos Computacional, Dinâmica dos Fluídos Computacional e Interação Fluído-Estrutura, entre outros, estão entre as aplicações que exigem grande capacidade de processamento e de armazenamento, em especial em simulações tridimensionais, em função da complexidade das geometrias estudadas, do grau de discretização dos modelos numéricos necessários e das limitações de estabilidade para avanço temporal dos esquemas explícitos usualmente empregados. Oliveira Jr. (2006) relata tempos de CPU de até 480 horas para a simulação de modelos de escoamento turbulento em domínios de geometria simples utilizados para validação dos códigos computacionais implementados. Xavier (2008) indica tempos de CPU em torno de 350 horas para aplicações similares com malhas de 72 mil elementos hexaédricos, e de 100 a 840 horas para modelos com 76 mil elementos hexaédricos, tempos consideráveis para estruturas de dados relativamente

pequenas. Estes exemplos mostram como mesmo pequenos centros de pesquisa necessitam de computação de alto desempenho para o desenvolvimento de algoritmos numéricos e sua validação.

1.1 COMPUTAÇÃO DE ALTO DESEMPENHO

Como em várias outras áreas da computação, a busca por maior desempenho computacional no processamento científico de alto desempenho (*High Performance Computing* –HPC) têm levado desenvolvimento de várias novas tecnologias de *software* e *hardware,* e a disponibilidade dessas tecnologias propicia o uso de modelos e algoritmos mais complexos com ainda maior demanda computacional.

Duas estratégias principais tem sido utilizadas na busca por elevadas capacidades de processamento. Cronologicamente, a primeira a ser empregada foi a construção de unidades de processamento de grande desempenho, em geral vetoriais, os super computadores. Por seu elevado custo de desenvolvimento e pela demanda mundial de um reduzido número de unidades, impedindo a diluição desses custos no processo de fabricação, os supercomputadores têm um custo de implantação e manutenção bastante alto e, pela velocidade de desenvolvimento tecnológico do *hardware*, uma vida útil relativamente curta antes que seja superado em desempenho por soluções muito mais baratas e, principalmente, com um custo de operação muito menor. Desta forma, têm sido utilizados principalmente em áreas estratégicas, para fins militares, em laboratórios estatais, em centros de pesquisa e universidade e em grandes empresas de tecnologia.

A segunda estratégia é a da adição. Utilizar um determinado número de unidades de processamento de menor desempenho de forma coordenada para obter uma maior capacidade de processamento pela execução simultânea de instruções, dando origem aos computadores paralelos. A união de um número muito grande de processadores relativamente simples forma um computador massivamente paralelo.

Com a popularização dos computadores pessoais, a estratégia para HPC tem sido a utilização não mais de processadores especificamente desenvolvidos para o processamento científico, mas sim a utilização de processadores destinados a uso geral ou de derivativos de alto desempenho das arquiteturas empregadas nesses processadores, bem como do maior número

possível de componentes presentes nos computadores de uso geral como memória, circuitos de controle, unidades de armazenamento ou, novamente, derivativos de alto desempenho das tecnologias utilizadas nesses componentes. Isso proporcionou um custo bastante mais baixo na aquisição e principalmente na manutenção dos computadores de alto desempenho, uma vez que boa parte do custo de desenvolvimento da tecnologia básica de seus componentes é amortizada pelo enorme mercado mundial de computadores pessoais atualmente existente.

Na lista mantida pelo site Top500 (Top500, 2008) com os 500 computadores mais rápidos do mundo, em novembro de 2008, seis deles utilizavam o processador AMD Opteron (AMD, 2008), um derivativo para estações de trabalho e servidores dos processadores Athlon 64 ou Phenom, de uso geral , e um utilizava o processador Intel Xeon (Intel, 2008), também um derivativo para estações de trabalho e servidores do processador Core 2 de uso geral.

Para que um computador paralelo obtenha alto desempenho, é preciso que os processadores comuniquem-se entre si de maneira rápida, seja através do acesso comum à memória principal (paralelismo de memória compartilhada), seja através de uma rede de comunicação (paralelismo de memória distribuída). Em geral o paralelismo de memória compartilhada limita-se a um número relativamente pequeno de processadores, uma vez que os circuitos de lógica e controle do acesso de diversos processadores simultaneamente à memória tornam-se exponencialmente mais complexos e caros com o aumento do número de processadores.

No paralelismo de memória distribuída, cada processador têm acesso somente à memória a ele diretamente alocada, e o acesso aos dados na memória de outros processadores ocorre através de mensagens entre os processadores por uma rede de conexão. O desempenho computacional obtido é dependente da latência, taxa de transferência e topologia dessa rede, e contínuos esforços têm sido feitos para encontrar tecnologias de interconexão que proporcionem maior desempenho.

A junção dos dois modelos dá origem a configurações híbridas, em que conjuntos de processadores com acesso a uma memória comum, chamados de nós ou nodos, comunicam-se com outros conjuntos através de uma rede de interconexão.

Todas essas características de *hardware* voltadas para o paralelismo necessitam de *softwares* adequados para que se tornem operacionais, desde sistemas operacionais, protocolos de comunicação, compiladores até as aplicações. Escrever códigos que tirem proveito do paralelismo, seja ele de memória compartilhada, memória distribuída ou ambas implica em

entender o funcionamento de cada uma das configurações, os gargalos ao desempenho e em utilizar e desenvolver algoritmos que sejam adequados ao ambiente paralelo.

Qualquer que seja a configuração de paralelismo, os algoritmos utilizados devem permitir que tarefas sejam feitas simultaneamente. Os processos de solução iterativos, nos quais os valores das variáveis do problema na iteração i+1 são calculados de forma independente a partir dos valor das variáveis na iteração i são especialmente adequados. No paralelismo de memória compartilhada, a principal preocupação é evitar problemas de conflito de memória, que ocorrem quando mais de um processador tenta alterar simultaneamente o conteúdo de uma mesma variável, seja por uma organização de dados e/ou de operações que a evite, seja pelo uso de instruções de sincronização. No paralelismo de memória distribuída, os algoritmos empregados devem prever formas de dividir as tarefas entre os processadores, balancear a carga de trabalho entre eles e definir quais variáveis devem ser comunicadas entre processadores e em que etapa do processo. Uma visão geral desses tópicos pode ser encontrada em Martins et al. (2008) e um painel bastante completo em Dorneles (2003).

1.2 TERMINOLOGIA E CONCEITOS BÁSICOS SOBRE PARALELISMO

Até 2005 todos os processadores de computadores pessoais continham um único núcleo físico de processamento, ou seja, um conjunto de circuitos eletrônicos capazes de executar de forma autônoma um determinado código de processamento. Paralelismo de memória compartilhada era obtido pela adição de mais processadores a uma placa-mãe, com circuitos de controle especialmente projetados para isso. Esse tipo de sistema atualmente é chamado de multi-soquete (*multi-socket)*, pois os processadores são conectados às placas-mãe através de um soquete.

Em 2005 tanto a Intel como a AMD lançaram processadores hoje denominados de processadores de núcleos múltiplos (*multi core*). Na sua versão mais simples, cada núcleo ou *core* corresponde a circuitos eletrônicos de um processador completo, encapsulados juntos em um mesmo invólucro e compartilhando as vias de conexão com a placa mãe. Em sua versão mais sofisticada, os núcleos estão interconectados através de vias especiais de alta velocidade, e tem um *cache* e e uma controladora de memória comum.

Em 2001 a Intel desenvolveu uma tecnologia chamada Hyper-Threading, na qual um núcleo de processamento físico podia ser configurado no equivalente a dois processadores lógicos, um com a quase totatlidade dos recursos físicos do processador e o outro aproveitando unidades lógicas e aritméticas e de processamento de ponto flutuante que não estivessem em uso. Os processadores lógicos assim formados são bastante heterogêneos na capacidade de processamento.

Assim, quanto ao paralelismo de memória compartilhada, um computador pode ter uma placa mãe multi-soquete, em cada soquete pode estar conectado um processador de múltiplos núcleos, cada núcleo pode representar mais de um processador lógico.

Neste texto, em situações genéricas que podem ser aplicadas tanto um computador, a um processador físico de um computador, a um núcleo de processamento de um processador físico de um computador ou a um processador lógico de um núcleo de processamento de um processador físico de um computador, será utilizado o termo *processador lógico* para designar indistintamente qualquer uma das entidades citadas como uma unidade autônoma de processamento. Quando, em função da características específicas do assunto em discussão, se fizer necessário distingüi-los, isso será feito através dos termos correspondentes *computador*, *processador* e *núcleo*.

Os *clusters* são agrupamentos de computadores independentes, homogêneos ou não, interconectados por uma rede de comunicação. As principais vantagens nesse modelo são os custos baixos e alta escalabilidade (PEZZI, 2006). A capacidade computacional da arquitetura pode ser facilmente aumentada pela adição de mais computadores ao longo do tempo.

Originalmente referindo-se a um computador específico construído em 1994 por Thomas Sterling e Donald Becker na NASA, *Beowulf* é a denominação empregada para uma classe de *clusters* formados por computadores pessoais comuns de uso geral executando um sistema operacional livre e de fonte aberta como o BSB, Linux ou Solaris e conectados por uma pequena rede local TCP/IP. De todas as alternativas disponíveis para HPC, é a que apresenta o menor custo de implantação e manutenção, pois todos os seus componentes de *hardware* são adquiridos de fornecedores de computadores de uso geral, e não há custos com o software. *Clusters* com computadores pessoais podem eventualmente utilizar *software* proprietário, não sendo mais classificados de *Beowulf*.

Clusters são denominados permanentes quando especificamente configurados para o processamento paralelo e utilizados somente para essa atividade. *Clusters* temporários são conjuntos de computadores de emprego geral temporariamente interconectados através de uma rede local para a solução conjunta de tarefas de HPC. Os *Grids* distinguem-se dos *clusters* temporários por serem em geral mais heterogêneos e geograficamente mais dispersos, podendo pertencer a redes diferentes e executando tarefas que não necessitem de uma forte sincronização.

Quando um computador faz parte de um *cluster*, ele recebe a denominação de nó ou nodo. Neste texto, será utilizada a designação nodo para evitar confusão com os nós das malhas de elementos finitos utilizadas na discretização espacial dos problemas. Se os nodos de um *cluster* têm, cada um, um único processador lógico, esse *cluster* é classificado como de paralelismo de memória distribuída. Havendo mais de um processador lógico por nodo, dentro do nodo há paralelismo de memória compartilhada e entre nodos há paralelismo de memória distribuída.

Programas para processamento paralelo são geralmente implementados em linguagens correntes de alto nível para processamento científico como C, C++ ou Fortran, com a adição de bibliotecas de paralelização que permitem mobilizar os diversos processadores lógicos presentes . As duas APIs (*Aplication Programming Interface*) mais comumente utilizadas para implementar processamento paralelo são OpenMP (Open Multi Processing) e MPI (*Message Passing Interface*).

A biblioteca OpenMP destina-se a paralelismo de memória compartilhada, permitindo que os diversos processadores lógicos presentes sejam mobilizados para a execução de um mesmo código e tenham acesso direto às variáveis, que são comuns. O emprego mais usual do OpenMP é permitir o paralelismo de laços do programa, repartindo as repetições do laço em tantos grupos seqüenciais quanto forem os processadores lógicos e atribuindo a cada processador lógico uma das seqüências de repetições. As variáveis e vetores presentes no código são duplicados tantas vezes quantos forem os processadores lógicos, de modo que, ao iniciar o laço, cada processador tem uma cópia idêntica e independente das variáveis do programa. Ao final do laço as variáveis são unificadas com os diversos valores gerados em cada laço. Existem na biblioteca uma série de comandos que permitem que algumas variáveis sejam comuns a todos os processadores lógicos (não sejam duplicadas nem independentes), que sejam privadas (duplicadas e independentes para cada laço), que determinados trechos do

código sejam executados por um processador lógico de cada vez (não haja modificação simultânea de uma mesma variável ou posição de vetor pelos diversos processadores), e impor em quantas partes um laço deve ser dividido (OPENMP, 2008).

A biblioteca MPI é destinada tanto a máquinas de memória compartilhada quanto distribuída, e funciona basicamente implementando comandos que permitem a transferência de conjuntos de dados (vetores) entre diversos processadores lógicos através da rede de dados. Os códigos que estão sendo executados nos diversos processadores lógicos são independentes e autônomos, não sendo obrigatoriamente iguais (ARGONE, 2005).

1.3 MÉTRICA DE PROGRAMAS PARALELOS

Na execução de um programa seqüencial, a capacidade computacional disponível de um único processador é utilizada para o processamento do código em um tempo transcorrido total indicado por t_s. Quando o mesmo programa é executado de forma paralela por p processadores lógicos idênticos, considera-se que a capacidade computacional disponível é p vezes maior, e que portanto o tempo transcorrido para a execução do programa deve ser p vezes menor. Esta condição corresponde a aplicações nas quais nenhum esforço computacional é associado à divisão de tarefas entre os processadores, comunicação, sincronização ou agrupamento de dados, somente possível quando não existe dependência entre os dados ou tarefas de cada processador (*embarrassingly parallel problems*).

Se t_p é o tempo efetivamente transcorrido na execução do programa com p processadores, define-se como a aceleração ou ganho de velocidade de processamento (*speed-up)* a razão (Wilkinson e Allen,1999)

$$S_p = \frac{t_s}{t_p} \tag{1.1}$$

Denomina-se *speed-up* absoluto quando o tempo de execução do programa seqüencial t_s corresponde ao melhor algoritmo seqüencial possível, e *speed-up* relativo quanto é utilizado o mesmo algoritmo paralelo com apenas um processador. Quando não especificado, deve-se considerar o *speed-up* como relativo, vez que ele não requer a implementação de um código seqüencial diferente do utilizado com vários processadores.

O *speed-up* ideal ou teórico é aquele no qual $S_p = p$, indicando que toda a capacidade computacional disponível é utilizada para acelerar a execução do programa.

Denomina-se *speed-up* superlinear quando o valor obtido com *p* processadores é maior que *p*. *Speed-up* superlinear ocorre na maioria das vezes com estruturas de dados relativamente pequenas com o uso de um número reduzido de processadores, e em geral é decorrente de um melhor desempenho da estrutura de *cache* dos processadores com uma quantidade menor de dados por processador no código paralelo em relação ao seqüencial, resultando num melhor desempenho da memória e, conseqüentemente, maior velocidade de processamento.

Define-se como eficiência de paralelização para *p* processadores lógicos E_p a razão

$$E_p = \frac{S_p}{p} = \frac{t_s}{p\,t_p} \tag{1.2}$$

A eficiência de paralelização inidica quanto do ganho de velocidade teórico ou ideal foi alcançado.

As equações acima são válidas apenas quando os *p* processadores têm a mesma capacidade de processamento. Quando o conjunto de processadores utilizados é heterogêneo, uma metodologia diferente precisa ser utilizada para a avaliação dos parâmetros *speed-up* e eficiência de paralelização.

O tempo transcorrido na execução do código seqüencial por cada processador é indicado por t_s^i. Considerando-se como tempo de referência t_s^{ref} o tempo de execução de um processador qualquer (por exemplo o do primeiro processador t_s^1), a velocidade relativa v_r^i de cada processador pode ser calculada por

$$v_r^i = \frac{t_s^{ref}}{t_s^i} \tag{1.3}$$

A velocidade relativa define a capacidade computacional relativa disponível de cada processador em termos de velocidade de processamento daquele código. Se um determinado número de processadores for utilizado para a execução paralela do programa, a velocidade relativa ideal ou teórica seria a soma das velocidades relativas dos diversos processadores. A velocidade relativa de execução em paralelo com *p* processadores v_{rp} é dada por:

$$v_{rp} = \frac{t_s^{ref}}{t_p} \tag{1.4}$$

A eficiência de paralelização E_p para p processadores é dada por

$$E_p = \frac{v_{rp}}{\sum_{i=1}^{p} v_r^i} \tag{1.5}$$

O *speed-up* é então calculado por

$$S_p = p.E_p \tag{1.6}$$

Desta forma, o valor intuitivo do *speed-up* continua o mesmo, ou seja, o valor teórico ou ideal para p processadores é p.

Uma outra métrica para avaliar o desempenho de implementações paralelas é a fração serial α definida pela métrica de Karp-Flatt (1990) dada por`

$$\alpha = \frac{t_{ser} + t_{red}}{t_s} \tag{1.7}$$

onde t_s é o tempo de execução de um programa de modo seqüencial em um único processador, t_{ser} é a parte do tempo de execução do programa em modo paralelo associado a algoritmos seqüenciais ou seriais que não são acelerados pela presença de mais processadores e t_{red} é a parte do tempo de execução do programa em modo paralelo associado à redundância do esforço computacional, comunicação de dados e a procedimentos de controle (*parallel overhead*). A fração serial α é determinada experimentalmente em um programa sendo executado por p processadores em função do *speed-up* S_p obtido através da equação

$$\alpha = \frac{\frac{1}{S_p} - \frac{1}{p}}{1 - \frac{1}{p}} \tag{1.8}$$

e indica a parcela do tempo de execução do programa em modo seqüencial que não diminui com o aumento do número de processadores. Quanto menor o valor de α, melhor é a

paralelização. Se α cresce com *p*, é um indicador de que a redundância do esforço computacional cresce com *p*.

Define-se como escalabilidade de um programa paralelo a propriedade do mesmo de manter a eficiência de paralelização à medida que são adicionados mais processadores ao conjunto. Um *speed-up* linear do tipo

$$S_p = k.p \tag{1.9}$$

com *k* próximo da unidade resulta em uma escalabilidade considerada muito boa. Quando a fração serial α mantém-se constante (e com valor baixo) com o aumento do número de processadores, o programa tem boa escalabilidade.

A maioria dos códigos paralelos não apresenta escalabilidade linear para um problema de tamanho fixo, perdendo eficiência para cada processador adicionado ao conjunto. Considerando α a porção do tempo de processamento de um programa executado por um único processador e associado a um algoritmo seqüencial, que não sofre redução com a adição de mais processadores, e (1-α) a porção do tempo de processamento que está associado a um algoritmo que pode ser paralelizado. Se o mesmo programa for executado por *p* processadores idênticos, o tempo de processamento será

$$t_p = \alpha + \frac{(1-\alpha)}{p} \tag{1.10}$$

e o valor de *speed-up* correspondente será de

$$S_p = \frac{1}{\alpha + \frac{(1-\alpha)}{p}} \tag{1.11}$$

As equações (1.10) e (1.11) são conhecidas como lei de Amdahl, em função de um artigo pioneiro em que Amdahl (1967) descreve literalmente o comportamento expresso pelas equações acima. A equação (1.11) pode ser utilizada para prever o maior valor de *speed-up* que pode ser obtido através da paralelização de um problema de tamanho fixo, levando $p \to \infty$, o que resulta em

$$S_{\infty} = \frac{1}{\alpha} \tag{1.12}$$

Assim, se um programa tem 20% de seu tempo de execução associado a um algoritmo seqüencial que não pode ser acelerado por paralelização para um determinado tamanho de problema, o maior valor de *speed-up* que pode ser obtido é de 5 para o mesmo problema, independentemente de quantos processadores são adicionados ao processo. A curva de *speed-up* versus número de processadores utilizados obtida pela equação (1.11) é assintótica ao valor S_{∞}.

A lei de Amdahl parece indicar um limitante bastante severo para configurações massivamente paralelas. Se uma máquina massivamente paralela conta com 1024 processadores, um programa atinge uma eficiência de paralelização de 50% somente se tiver aproximadamente 0,2% do tempo de processamento associado a um algoritmo seqüencial ou eventos que não são acelerados pela presença de mais processadores, como operações de entrada e saída (leitura e gravação de dados). Na prática, um mesmo problema de tamanho fixo somente é analisado com um número crescente de processadores para fins de pesquisa. Quanto maior o tamanho do problema, maior o número de processadores que são alocados para a tarefa, e a parcela seqüencial do tempo de processamento diminui com o aumento dos dados. Por exemplo, a relação entre o tempo de leitura de dados para a solução de um sistema de equações lineares (parte seqüencial) e o tempo necessário para solução do sistema (parte paralela) não se mantém constante com o aumento do tamanho do sistema.

Quem primeiro revisou os conceitos por trás da Lei de Andahl foi Gustafson (1988), baseado em resultados obtidos com um computador massivamente paralelo no Sandia National Laboratories (Estados Unidos). A chamada Lei de Gustafson estabelece que, se no processamento de um programa em um computador paralelo, o tempo de execução com *p* processadores é dado por $\alpha + \beta = 1$, onde α é a porção seqüencial e $\beta = (1-\alpha)$ a porção paralela, então se o programa fosse executado seqüencialmente por um único processador, o tempo de execução seria $\alpha + \beta p$ e o *speed-up* seria dado por

$$S_p = \alpha + p(1-\alpha) \tag{1.13}$$

ou, de forma equivalente, por

$$S_p = p - \alpha(p-1) \tag{1.14}$$

ou seja, quanto maior o número de processadores incluídos no processo, maior será o *speed-up* obtido. O comportamento previsto pelas leis de Amdahl e de Gustafson para $\alpha = 0{,}1$ está mostrado na figura 1.1.

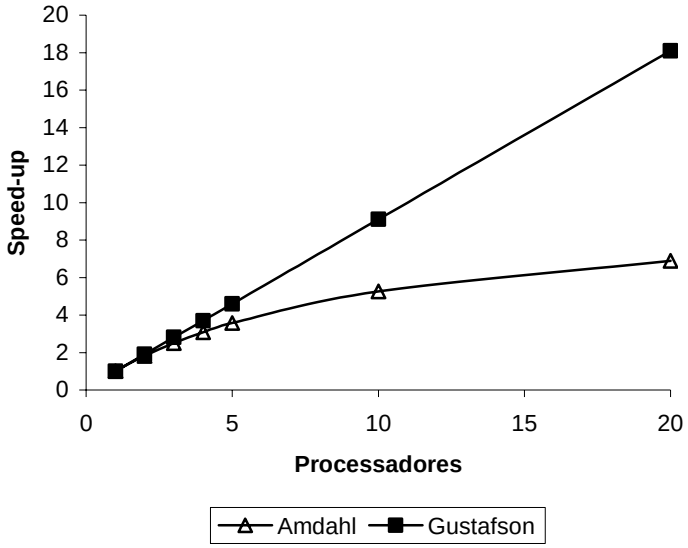


Figura 1.1: Previsão de speed-up pelas leis de Amdahl e Gustafson

É importante considerar que o valor de α na Lei de Amdahl representa a parcela de tempo de execução não paralelizável para o programa **seqüencial**, enquanto que na Lei de Gustafson representa a parcela de tempo não paralelizável no programa **paralelo**, sendo constante para qualquer número de processadores. A lei de Gustafson se aplica ao casos em que o coeficiente α torna-se gradativamente menor com o aumento do tamanho do problema, de modo que ao se adicionar processadores ao conjunto, sempre existirá um tamanho de problema grande o suficiente que fará com que α permaneça constante, mantendo a escalabilidade.

Em computação paralela, define-se como granularidade uma medida qualitativa da razão entre o processamento das tarefas e a comunicação de dados entre processadores (BARNEY, 2007). A granularidade pode ser classificada como grossa, quando grande quantidade de trabalho computacional é feita entre eventos de comunicação, e fina quando o trabalho computacional entre eventos de comunicação é pequeno.

1.4 COMUNICAÇÃO DE DADOS, DIVISÃO DE TAREFAS E BALANCEAMENTO

Em uma implementação usando MPI para paralelismo de memória distribuída, a rede lógica de conexão torna-se um gargalo importante ao desempenho, pois sua velocidade é muitas vezes menor e sua latência muitas vezes maior que a existente nos diversos outros subsistemas dos computadores. Em termos de velocidade de acesso e latência, o subsistema mais rápido é o *cache* dos processadores, seguido pela memória principal, os discos-rígidos locais e finalmente a rede lógica. Valores típicos de vazão de dados para esses sub-sistemas são de 6400 MB/s para uma memória DDR2-800 (*single-channel*, o dobro disso em *dual-channel*) , 300 MB/s para uma interface de disco-rígido SATA II e 100 MB/s para uma conexão de Gigabit Ethernet. *Clusters* de alto desempenho utilizam conexões de rede mais rápidas e de menor latência, como os padrões Myrinet (495 MB/s) e Infiniband (830 MB/s), ou conexões como a NUMAlink (3200 MB/s), que proporciona compartilhamento global da memória entre os nodos (SGI, 2008), mas mesmo essas conexões não atingem a velocidade de memória principal.

Desta forma, qualquer implementação usando MPI para ser bem sucedida deve minimizar ao máximo a transferência de dados entre os diversos processadores lógicos. A configuração ideal é aquela em que cada processador fica responsável por uma grande carga de trabalho computacional, e troca com um número mínimo de outros processadores uma quantidade mínima de dados, o mais espaçadamente possível (situações de granularidade grossa).

Em redes de baixo desempenho como a *Fast Ethernet* (10 MB/s), tão importante quanto a quantidade de dados é o número de processos de comunicação, em função da latência de comunicação em rede. Implementações de granularidade fina tendem a ser altamente ineficientes nessas condições, e os algoritmos utilizados devem ser modificados. (ver exemplo em MURARO JR. Et al., 2008).

Mesmo em aplicações com poucos processos de comunicação, quando o tamanho do problema é muito pequeno, a latência dos processos de comunicação em rede representam tempos consideráveis frente ao tempo de processamento da carga de trabalho alocada a cada processador, fazendo com que soluções paralelas tenham *speed-up* inferior à unidade (ver exemplo em Cuminato et al., 1999). Implementações com granularidade fina necessitam de

interconexões entre os processadores com baixa latência e alta velocidade para serem eficientes.

A quantidade de dados a ser trocada pelos processadores através da rede e o número de processos de comunicação são decorrentes tanto do algoritmo de solução escolhido quanto da forma como os dados são divididos entre os diversos processadores.

A divisão de tarefas ou particionamento de estruturas de dados, como as decorrentes da aplicação do Método dos Elementos Finitos, são geralmente tratados através de técnicas de paticionamento de grafos ou decomposição de domínios. A malha de Elementos Finitos é considerada como um conjunto de nós e arestas que definem a relação entre os nós. Um custo computacional é associado aos nós e às arestas, que representa o tempo de processamento associado aos nós e elementos da malha, e o objetivo da divisão é obter partições com o custo computacional proporcional à capacidade computacional do processador associado à cada partição (garantindo igual tempo de processamento para todos os processadores) e menor número de arestas cortadas (representando uma menor quantidade de dados trafegando na rede de comunicação entre processadores).

O particionamento da malha pode ser nodal (os nós são distribuídos entre as diversas partições, sendo partidas as arestas) ou dual (os elementos da malha são distribuídos entre as partições, e os nós são partidos)(ELIAS, 2008). Para a partição dual, o objetivo da partição é minimizar o número de nós partidos.

O problema de particionamento de grafos tem enorme importância para uma série de áreas de conhecimento, como mapeamento de genoma, programação linear, transporte e integração de larga escala de circuitos (VLSI) (KARYPIS e KUMAR, 2008). Diversos métodos de particionamento foram desenvolvidos. Uma visão bastante detalhada sobre as características de cada método pode ser encontrada em (DORNELES, 2003).

Duas classes de métodos relativamente simples serão detalhados neste texto pois servirão de base para os métodos empregados no capítulo 3. A primeira é caracterizada pelo particionamento em faixas (*stripwise partitioning* ou STRIP). Este método, consiste em seccionar a malha segundo planos paralelos entre si, de modo a limitar regiões com o mesmo custo computacional, conforme mostra a figura 1.2.

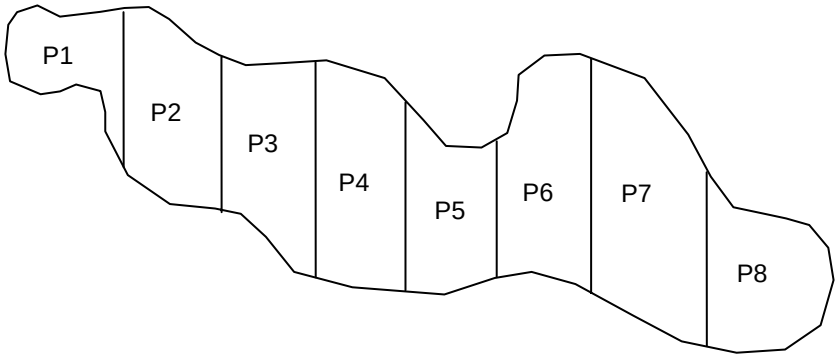


Figura 1.2: Particionamento em faixas.

Este método é comumente empregado em aplicações utilizando o Método das Diferenças finitas, onde a malha é obrigatoriamente estruturada segundo os eixos coordenados, tornando esse tipo de divisão natural. Exemplos podem ser encontrados em Cuminato et al. (1999) e Dorneles (2003).

Variantes incluem permitir que a fronteira entre os subdomínios não seja reta, mas que possa ter uma endentação ou degrau, conforme a figura 1.3. Isso aumenta ligeiramente a comunicação entre os subdomínios, mas permite um balanceamento das cargas bastante mais preciso.

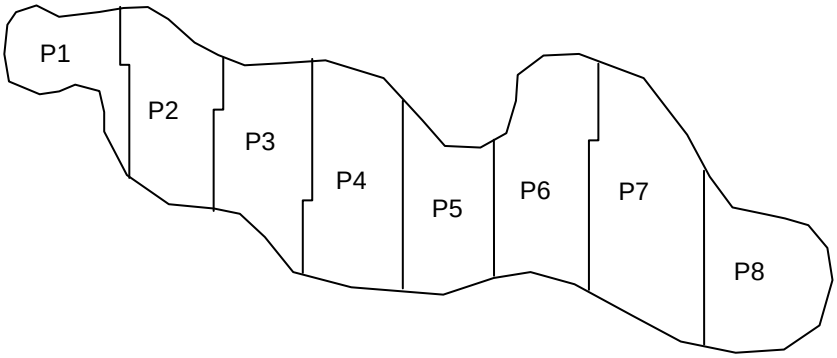


Figura 1.3: Particionamento em faixas com fronteira não retas.

O particionamento inercial em faixas consiste em determinar a maior direção da malha considerando-se os nós como unidades de massa, e calculando os eixos principais centrais de inércia. As partições são feitas por planos normais ao eixo de menor inércia. O resultado pode ser visto na figura 1.4, onde o eixo de menor inércia está indicado pela seta pontilhada. Pode-se utilizar um critério de ponderação atribuindo como massa a cada nó o custo computacional a ele associado. Para particionamentos nodais a falta de alinhamento entre a orientação das fronteiras dos sub-domínios e a orientação das faces dos elementos da malha não constitui

problema. Contudo, para paticionamentos duais, a fronteira entre domínios não permanece retilínea ou plana, devendo se adaptar à orientação das faces dos elementos, dando origem a fronteiras irregulares ou serrilhadas, especialmente em malhas estruturadas.

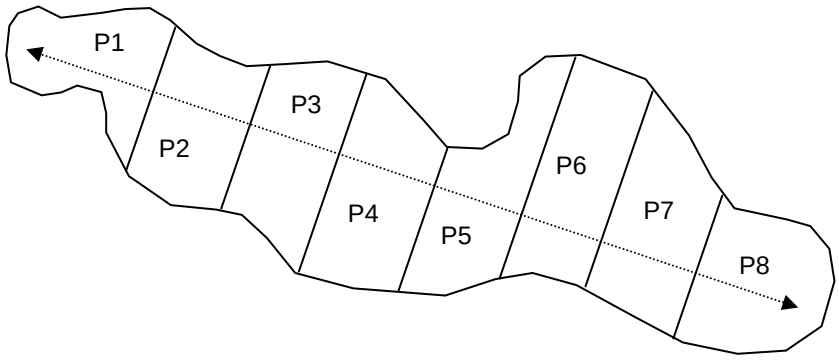


Figura 1.4: Particionamento inercial em faixas.

Quando o número de divisões do domínio é grande, os subdomínios correspondentes a cada processador tornam-se faixas estreitas, e as fronteiras entre os sub-domínios tornam-se grandes em relação ao volume dos sub-domínios. Como as fronteiras correspondem à comunicação de dados entre processadores, e os volumes à cargas de trabalho de cada processador, com o aumento do número de divisões a granularidade tende a tornar-se mais fina. Podendo levar a baixas eficiências em clusters com redes muito lentas. Contudo, o processo de particionamento em faixas é o que garante o número mínimo de fronteiras para um dado sub-domínio e, conseqüentemente, o menor número de processos de comunicação, o que pode é vantajoso em redes com grande latência.

A segunda classe é caracterizada pelo particionamento utilizando bissecção recursiva por coordenadas. (*Recursive Coordinate Bissection-RCB*). A partir das coordenadas dos nós, a bissecção é aplicada recursivamente para dividir um domínio em dois subdomínios utilizando planos normais a qualquer um dos eixos (*x, y, z*). O custo computacional associado a cada nó da malha e a capacidade de processamento dos processadores podem ser utilizados para ponderar a bissecção. Um exemplo desse tipo de divisão é mostrado na figura 1.5, onde os números junto a cada fronteira indicam a ordem de bissecção..

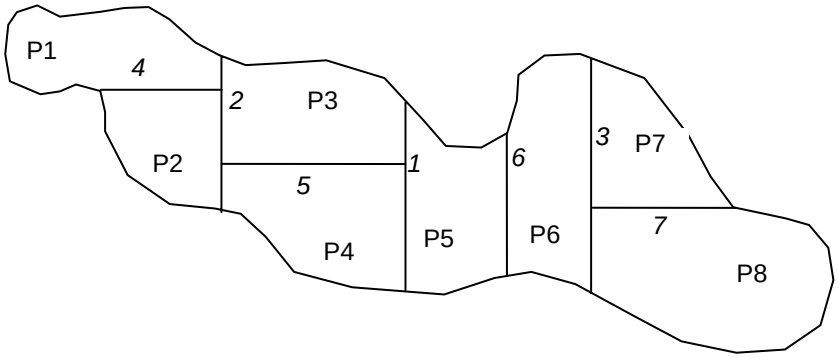


Figura 1.5: Particionamento através de bissecção recursiva por coordenadas.

O processo de bissecção recursiva por coordenadas tem a vantagem de ser simples e de baixo custo computacional (FÖSTER, 1994). Um problema desse método é que ele não otimiza a comunicação, podendo levar a subdomínios com grandes fronteiras.

O particionamento inercial por bissecção recursiva consiste em, antes de aplicar a bissecção, determinar os eixos principais centrais de inércia do domínio ou subdomínio, de modo a seccioná-lo segundo um plano normal ao eixo de menor inércia, prosseguindo o processo recursivamente. Um exemplo desse processo está mostrado na figura 1.6. O particionamento inercial por bissecção recursiva leva a fronteiras bem menores que as obtidas com a bisscecção sendo feita segundo planos normais aos eixos (x,y,z), embora tenha um custo computacional maior. Os particionamentos por bissecção resultam em fronteiras menores que o particionamto por faixas, mas o número de vizinhos de cada subdomínio é maior, representando mais operações de comunicação. Em redes com grande latência, isso pode vir a contrabalançar os ganhos decorrentes da menor quantidade de dados em cada transmissão. Além disso, se comunicações blocantes são utilizadas, o ordenamento dessas comunicações torna-se bastante mais complexo à medida que o número de processadores cresce.

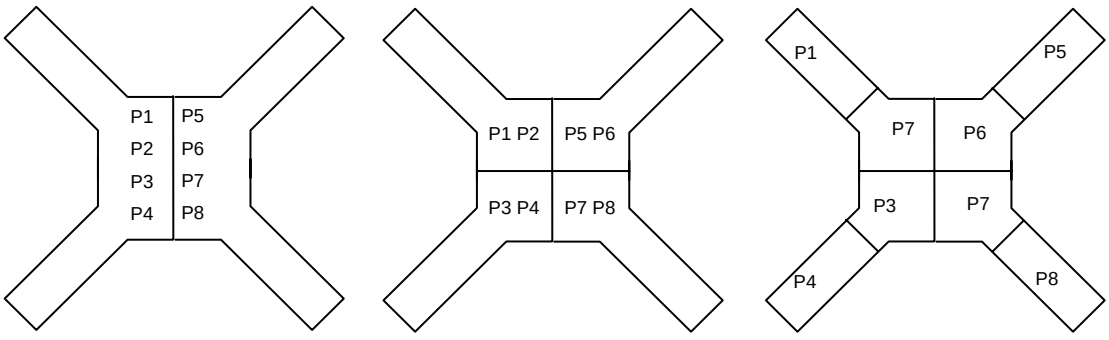


Figura 1.6: Particionamento inercial por bissecção recursiva por coordenadas.

Diversos *softwares* para o particionamento de malhas estão disponíveis, dentre os quais o CHACO (HENDRICKS e LELAND, 2008), o PARTY (PREISS e DIEKMANN, 2008), o JOSTLE (WALSHAW e CROSS, 2008), o SCOTCH (PELLEGRINI, 2008) e o METIS (KARYPIS e KUMAR, 2008).

Como no processamento paralelo utilizando MPI há freqüentemente a necessidade de sincronização dos diversos processadores lógicos em etapas determinadas do código sendo processado, a velocidade do conjunto será sempre limitada pelo processador mais lento, ou seja, aquele que leva mais tempo para processar seu conjunto de dados. Uma distribuição ideal de tarefas é aquela que faz com que todos os processadores lógicos presentes levem o mesmo tempo para processar seu conjunto de dados entre um ponto de sincronização e o seguinte, de modo que o tamanho do conjunto de dados de cada processador é proporcional ao seu desempenho relativo ou velocidade de processamento relativa.

Em clusters não homogêneos, a velocidade de processamento relativa de cada processador lógico presente não é facilmente quantificada *a priori*, e nem tem o mesmo valor para todos os tipos de problemas. A freqüência (*clock)* do processador é um indicativo inicial, mas se num *cluster* estão presentes processadores de arquiteturas diferentes, os mesmos podem executar um número diferente de operações por ciclo de *clock*, levando a desempenhos diferentes para uma mesma freqüência. Algumas arquiteturas tem melhor desempenho que outras no processamento de inteiros, mas não em operações de ponto flutuante, causando dependência com o tipo de código sendo executado. Além disso, processadores dentro de uma mesma arquitetura podem apresentar freqüências diferentes, tamanhos de *cache* diferentes e utilizar memórias de um mesmo padrão com velocidades e latências diferentes, dando origem a uma infinidade de combinações com características diferentes de desempenho para cada tipo de programa e tamanho de problema. Isso faz com que, em clusters heterogêneos, seja extremamente difícil quantificar a capacidade computacional dos diversos processadores lógicos de modo a distribuir as cargas de trabalho de forma proporcional e equilibrada.

Além disso, a própria quantificação das cargas ou custos computacionais associados a um nó ou elemento de uma malha de elementos finitos não é uniforme. Cargas e condições de contorno são aplicadas apenas sobre alguns elementos ou nós, não-linearidades como plasticidade e dano contínuo desenvolvem-se ao longo do processo de análise, alterando a distribuição do esforço computacional à medida que o processo de solução avança. Implementações com malhas adaptativas criam e removem elementos e nós localmente ao

longo do processo de solução, alterando a distribuição de cargas entre os processadores. Processos adaptativos para a integração no tempo como a técnica de sub-ciclos implicam que cada iteração tem uma distribuição espacial diferente do esforço computacional.

Em função das características acima, balanceamentos de carga estáticos, semi-estáticos ou dinâmicos podem ser adequados (DEMEL, 2007). O balanceamento estático ocorre quando toda a informação necessária está disponível antes do balanceamento. Para problemas como solução de sistemas de equações, esquemas explícitos de Taylor-Galerkin para simulação de escoamentos de fluídos sem o uso de malhas adaptativas ou de técnicas de sub-ciclos o esforço computacional é constante ao longo do processo de solução, e um esquema estático de balanceamento pode ser utilizado desde que se tenham estimadores precisos tanto do custo computacional associado a cada nó e elemento da malha quanto da capacidade computacional relativa dos processadores utilizados para o tipo de aplicação e tamanho do problema em questão.

Balanceamento semi-estático é utilizado em situações em que a distribuição do esforço computacional se altera lentamente ao longo do processo de solução (plasticidade, dano contínuo, malhas adaptativas), permitindo que um esquema estático seja utilizado no início do processo e um certo desbalanceamento seja tolerado ao longo de um certo número de iterações, onde um processo estático para balanceamento é novamente aplicado.

Balanceamento dinâmico é utilizado onde a carga de trabalho associada aos nós e elementos varia continuamente ao longo do processo de solução (HENDRICKSON e DEVINE, 2000, CATALYUREK et al., 2007), como quando são utilizados métodos adaptativos na integração temporal como a técnica de sub-ciclos. Nestas situações é preciso uma avaliação cuidadosa para verificar se o custo computacional associado ao contínuo rebalanceamento de cargas não é maior que os ganhos advindos do balanceamento resultante. Para muitos autores, não há diferenças entre o balanceamento de cargas semi-estático e o dinâmico, pois somente diferem na freqüência com que o rebaleanceamento é aplicado.

Existem 2 abordagens básicas para o reparticionamento do domínio, uma vez detectada a necessidade de rebalanceamento de cargas: realizar um particionamento de cargas inteiramente novo, partindo do zero, ou utilizar algoritmos de difusão. Embora a primeira abordagem possa levar a um particionamento melhor da malha (distribuição de cargas equilibrada, fronteiras entre domínios pequenas, pequena quantidade de subdomínios vizinhos

a um subdomínio) tem um custo computacional mais alto e implica em uma grande movimentação de dados através da rede, sendo mais adequada para balanceamentos semi-estáticos que dinâmicos. Nos algoritmos por difusão, os processadores comparam continuamente suas cargas de trabalho e, havendo uma diferença acima de determinado limite, nós e elementos são passados de uma partição para outra. (PELLEGRINI, 2007).

Uma das formas de se avaliar a carga de trabalho de cada processador lógico considerando tanto as variabilidades decorrentes do hardware, do algoritmo de solução empregado quanto da malha utilizada na discretização é através de métodos empíricos, como a medição direta do tempo de processamento de cada processador. Kwok et al. (1999) utilizam um mecanismo de balanceamento de cargas centralizado, no qual, em intervalos de tempos regulares, os diversos processadores lógicos enviam informações a um processador central sobre o tempo de CPU utilizado no intervalo para processar sua carga de trabalho. A comparação entre os tempos dos diversos processadores indica a necessidade de redistribuição de cargas, e o tempo de CPU de cada processador e sua carga de trabalho servem de parâmetros para estimar a nova carga de trabalho. A mesma técnica foi utilizada por Dorneles (2003) no programa HIDRA.

1.5 DINÂMICA DOS FLUÍDOS COMPUTACIONAL EMPREGANDO O MÉTODO DOS ELEMENTOS FINITOS.

Na Dinâmica dos Fluídos Computacional, a grande maioria dos modelos numéricos empregam como método de discretização o Método das Diferenças Finitas, o Método dos Volumes Finitos e o Método dos Elementos Finitos. Dentre eles, o Método dos Elementos Finitos é o que apresenta maiores vantagens no tratamento de problemas com geometrias complexas e na aplicação de condições de contorno não convencionais, embora demande maior quantidade de memória e maior tempo de processamento. Três importantes referências na aplicação do Método dos Elementos Finitos na análise de escoamentos são encontradas em Zienckiewicz et. al (2005), Grecho e Sani (1999) e Reddy e Gartling (1994).

O método de Taylor-Galerkin, proposto por Donea (1984) consiste em uma expansão em série de Taylor no tempo das equações de Navier-Stokes e de Euler e uma discretização através do método de Bubnov-Galerkin sobre o domínio espacial e tem sido um dos métodos mais empregados na solução de escoamentos de fluídos compressíveis e incompressíveis.

A integração temporal das equações de Navier-Stokes e de Euler pode ser feita via esquemas explícitos, implícitos ou uma combinação de ambos. Os esquemas implícitos tem um melhor desempenho em problemas estacionários por não apresentar restrições de estabilidade, possibilitando empregar um elevado passo de tempo para obtenção da solução.

Os esquemas explícitos tem como vantagem a simplicidade, pouca demanda por memória e boas características em problemas transientes, sendo largamente utilizados na simulação numérica desses problemas. O emprego de esquemas explícitos com o Método dos Elementos Finitos resulta em um processo iterativo no qual o sistema de equações envolvendo as variáveis do problema é desacoplado dentro de cada iteração, permitindo o cálculo das variáveis na iteração $i+1$ a patir de seus valores na iteração i, fazendo com que formulações baseadas nesse esquema sejam especialmente adequadas ao emprego de paralelismo de memória distribuída. Por outro lado, a condição de Courant-Friedrichs-Levy (1928) relacionando o máximo passo de tempo utilizado na discretização temporal com as dimensões dos elementos utilizados na discretização espacial e com a velocidade de propagação de onda elástica no meio deve ser atendida para garantir a estabilidade, de modo que os esquemas explícitos exigem passos de tempo muito menores que os utilizados nos esquemas implícitos. Em escoamentos transientes e turbulentos, a própria variabilidade dos fenômenos físicos envolvidos impõe uma limitação natural ao passo de tempo máximo que pode ser utilizado, anulando a principal vantagem dos métodos implícitos e fazendo com que os métodos explícitos sejam largamente utilizados na simulação desses escoamentos.

Do ponto de vista de capacidade computacional, a simulação de escoamentos transientes e turbulentos em problemas de geometria complexa faz com que sejam necessárias muitas vezes malhas com alto grau de refinamento, seja para reproduzir as características geométricas das superfícies em estudo, seja para representar corretamente os altos gradientes das variáveis do problema em regiões específicas do domínio de estudo. O refinamento espacial da malha, que por si só impõe uma grande demanda por recursos computacionais de capacidade de processamento e memória, implica também em um refinamento na discretização temporal, exigindo passos de tempos menores pela condição de estabilidade de Courant-Friedrichs-Levy decorrentes do menor tamanho dos elementos e exigindo ainda mais recursos computacionais, especialmente capacidade de processamento. Desse duplo impacto sobre o desempenho computacional vem a enorme necessidade de computação de alto desempenho das aplicações em Dinâmica dos Fluídos Computacional.

No âmbito dos programas de Pós-Graduação em Engenharia Civil e de Pós-Graduação em Engenharia Mecânica da UFRGS, os esquemas explícitos têm sido utilizados em diversos trabalhos para a solução de escoamentos compressíveis (BURBRIDGE, 1999, AZEVEDO, 1999, TEIXEIRA, 2001, KESSLER, 2002, BONO 2008) e de escoamentos incompressíveis (GONZÁLEZ, 1993, PETRY, 2002, POPIOLEK, 2005, OLIVEIRA Jr., 2006, SANTOS, 2007, BRAUN, 2007, XAVIER, 2008)

As características indicadas acima e a ampla experiência dos grupos de pesquisa locais na aplicação dos esquemas explícitos com o método dos elementos finitos justificam sua escolha para este trabalho.

Nos casos de escoamentos compressíveis, a pressão é determinada pelos valores locais e instantâneos da massa específica e energia interna através da equação de estado. Por este motivo, geralmente são empregadas as variáveis conservativas nas equações para esses escoamentos. Os algoritmos para escoamentos compressíveis tendem a reduzir sua precisão e convergência quando o número de MACH diminui, devido à grande disparidade que existe entre os termos acústicos e convectivos, e portanto sua aplicação a escoamentos incompressíveis torna-se restrita (Turkel et al., 1997).

Nos escoamentos incompressíveis, a pressão é somente função do campo de velocidades. As principais formulações empregadas na solução de escoamentos incompressíveis estão baseadas no método de correção de pressão ou no conceito de compressibilidade artificial ou pseudo-compressibilidade (Chorin, 1967, Kawahara e Hirano, 1983, Kwak et al., 2005, Zienckiewicz et al. 2005).

Neste trabalho, foram utilizados duas formulações diferentes para a solução de escoamentos compressíveis e incompressíveis. Para escoamentos incompressíveis foi utilizado um esquema explícito de dois passos empregado por Braun (2007) e para escoamentos incompressíveis um esquema explícito de um passo com iterações empregado por Bono (2008). Computacionalmente as formulações são semelhantes, permitindo um tratamento também semelhante na implementação paralela para memória distribuída.

1.6 DINÂMICA DOS SÓLIDOS COMPUTACIONAL EMPREGANDO O MÉTODO DOS ELEMENTOS FINITOS.

Na Mecânica dos Sólidos computacional, o Método dos Elementos Finitos tem sido preponderante como método de discretização espacial desde o começo da década de 70 do século passado. Quanto à discretização temporal da equação de equilíbrio dinâmico, os dois modelos numéricos mais empregados são o método explícito de Taylor-Galerkin (Azevedo, 1999) e o método implícito de Newmark (Bathe, 1996).

Tendo as mesmas vantagens que apresenta quando empregado para a Dinâmica dos Fluídos Computacional, a manutenção da estabilidade numérica no uso do método de Taylor-Galerkin implica no uso de passos de tempo muito menores que os utilizados na simulação dos escoamentos em função da muito maior velocidade de propagação da onda elástica em meios sólidos que em fluídos, impondo uma enorme demanda de capacidade computacional.

O método implícito de Newmark é incondicionalmente estável para problemas lineares, podendo utilizar passos de tempo bastante grandes, limitados somente pelas características dos fenômenos dinâmicos e vibratórios em estudo. Contudo, o método demanda maior consumo de memória, pela necessidade de montagem de matrizes globais e a solução de sistemas de equações, responsável pela maior parte do tempo de processamento do método quando aplicado a grandes estruturas de dados.

Em problemas dinâmicos de estruturas aeronáuticas, a presença de fenômenos vibratórios de alta freqüência e de efeitos dinâmicos de alta velocidade justificam o emprego do método explícito de Taylor-Galerkin. Já na Engenharia de Vento Computacional, o comportamento dinâmico é dominado por modos vibratórios de baixa freqüência, e o método implícito de Newmark mostra-se mais vantajoso.

Como o método de Newmark não é capaz de conservar energia, além de perder seu caráter dissipativo em problemas com não-linearidade, ele deixa de ser incondicionalmente estável em presença de não-linearidade. Diversos métodos de estabilização foram criados a partir do final da década de 70 do século passado, como o método da energia restringida de Hughes et al. (1978), o método do momento de energia de Simo e Tarnow (1992) e o método α-Hilber (HILBER et al. 1977), baseado em dissipação numérica. Khul e Ramm (1996) desenvolveram uma formulação para obtenção de parâmetros otimizados para os métodos α e os estenderam

a problemas não-lineares. A vantagem dessa formulação é que ela mantém a simetria da matriz de rigidez tangente e seu custo computacional é similar aos esquemas de integração no tempo baseados no método de Newmark tradicional.

A solução de sistemas de equações é freqüentemente responsável pela maior parte do tempo de processamento de simulações numéricas científicas e de engenharia (BENZI, 2002), e a solução de problemas com o Método dos Elementos Finitos através de esquemas implícitos tem essa característica.

Os métodos diretos de solução são empregados quando a confiabilidade da resposta é a principal preocupação. Esses métodos são muito robustos e tendem a requerer uma quantidade previsível de recursos em termos de tempo de processamento e memória. Infelizmente, o número de operações necessárias e a memória utilizada nos métodos diretos escalam de forma quadrática com o aumento do tamanho do problema, especialmente em sistemas resultantes da discretização de equações diferenciais parciais no espaço tridimensional, como o Método dos Elementos Finitos. Simulações tridimensionais detalhadas multi-físicas como as que estão sendo realizadas dentro do programa ASCI do Departamento de Energia do governo dos Estados Unidos resultam em sistemas de equações lineares da ordem de centenas de milhões a bilhões de equações e incógnitas. Em novembro de 2008, 7 dos 10 computadores mais rápidos do mundo pertenciam a instalações operadas pelo departamento de energia (TOP500, 2008). Para tais tamanhos de problemas, sistemas iterativos são a única opção possível. Mesmo aplicações mais modestas e rotineiras geram sistemas com alguns milhões de equações e incógnitas, tornando os métodos iterativos obrigatórios.

Enquanto os métodos iterativos requerem uma quantidade de memória muito menor que os métodos diretos e freqüentemente requerem um número menor de operações, especialmente quando uma solução aproximada com relativamente pouca precisão é suficiente para a aplicação, eles não têm a confiabilidade dos métodos diretos e, em muitas classes de problemas, falham, necessitando de précondicionamento.

Na indústria do petróleo e de energia nuclear, os métodos iterativos sempre foram populares e forneceram o estímulo para muitas das pesquisas iniciais em métodos iterativos. Em contraste, os métodos diretos têm sido tradicionalmente os preferidos em áreas como análise estrutural, modelamento de semi-condutores e em Dinâmica de Fluídos Computacional (métodos

implícitos). Contudo, mesmo nessas áreas os métodos iterativos têm aumentado grandemente seu emprego nos anos recentes.

O método dos Gradientes Conjugados é considerado um dos melhores métodos iterativos para a classe de matrizes simétricas e positivo definidas (SAAD, 1996, SAAD e VAN DER VOST, 2000, VAN DER VOST, 2000). A grande atratividade do método para sistemas muito grandes é que ele referencia a matriz de coeficientes (a matriz de rigidez global da estrutura utilizando o Método dos Elementos Finitos) somente através de sua multiplicação por um vetor, ou de sua transposta por um vetor. Ou seja, não é necessário montar a matriz global explicitamente, apenas computar o efeito de pré-multiplicar a matriz por um vetor. Esta vantagem é explorada por estratégias elemento-por-elemento (*element-by-elemento* ou EBE), que realizam a multiplicação da matriz por vetor em nível de elemento. A quantidade de memória requerida por estratégias iterativas EBE tendem a crescer linearmente com o número de elementos, ao passo que estratégias que envolvam a montagem de uma matriz global crescem quadraticamente (PHON et al., 2002).

A eficiência do método dos Gradientes Conjugados é grandemente aumentada com o uso de precondicionadores. O termo precondicionamento refere-se a transformar o sistema de equações em um outro com características mais favoráveis a uma solução iterativa. Um precondicionador é uma matriz que efetua tal transformação.

O mais antigo e simples dos precondicionadores é o precondicionador diagonal ou de Jacobi. Em 1845 Jacobi usou rotações planas para obter dominância diagonal de sistemas de equações normais resultantes de problemas de mínimos quadrados (Shewchuk, 1994). O precondicionador diagonal é extremamente simples, requer apenas o armazenamento dos coeficientes da diagonal principal da matriz global, que podem ser computados elemento-por-elemento, e tem um custo computacional muito baixo. Contudo, o desempenho é muitas vezes medíocre, em especial em sistemas mal condicionados.

O precondicionador de maior importância para a popularização do método dos Gradientes Conjugados é o precondicionador incompleto de Cholesky., introduzido por van der Vorst em 1977 (BENZI, 2002). Ele consiste no emprego da triangularização de Cholesky aplicada de forma incompleta. Nas formulações elemento-por-elemento, isso é feito aplicado-se a triangularização de Cholesky em nível de elemento, e não sobre a matriz global. Do ponto de vista computacional, o precondicionador de Cholesky implica no armazenamento das matrizes

locais do elemento fatoradas, mas a escalabilidade com o número de elementos é linear. O custo computacional é mais alto que o do próprio método dos Gradientes Conjugados, mas a taxa de convergência obtida é em geral bastante alta, resultando em um tempo total de solução menor que com o precondicionador diagonal ou sem precondicionamento.

O precondicionador incompleto de Cholesky é muito eficiente para sistemas simétricos e positivo definidos, como é o caso da maioria dos sistemas resultantes da aplicação do Método dos Elementos Finitos. Sistemas não simétricos ou simétricos mas indefinidos decorrentes de muitas aplicações reais necessitam de sistemas de solução mais robustos (WU et al, 2008).

1.7 INTERAÇÃO FLUÍDO-ESTRUTURA.

Os primeiros modelos numéricos para análise acoplada de problemas de interação fluído estrutura utilizavam uma descrição cinemática do movimento puramente Euleriana, na qual os pontos materiais do fluído movem-se em relação a uma malha fixa. Essas formulações permitiam apenas análises em que a estrutura apresentava deslocamentos pequenos o suficientes para serem desprezados na descrição do contorno sólido do fluído. O acoplamento era feito através da imposição de condições de compatibilidade de velocidades e equilíbrio de forças sobre a interface fluído-estrutura. Exemplo desse tipo de abordagem pode ser encontrado em Petry (1993).

Modelos de acoplamento mais abrangentes foram desenvolvidos com uma descrição cinemática Arbitrária Lagrangeana-Euleriana – ALE (DONEA et al., 2004). Nessa descrição cinemática, existe um domínio de referência (a malha) que se move de maneira arbitrária em relação aos pontos materiais(o fluído) e aos pontos espaciais (o sistema de coordenadas global), e as equações da Mecânica do Contínuo passam a ser descritas a partir de pontos fixos nos sistema de referência (os nós da malha). Os primeiros trabalhos com esse tipo de formulação consideravam problemas em que o corpo imerso era considerado rígido com restrições elásticas, ou seja, que as deformações eram pequenas e podiam ser desprezadas frente aos movimentos de corpo rígido (GONZÀLEZ, 1993, BRAUN, 2002). Graças a maiores recursos computacionais, trabalhos mais recentes consideram corpos deformáveis com inclusão de não-linearidades. É comum nas implemetações ALE a consideração de um subdomínio em torno da estrutura, abrangente o suficiente para permitir a movimentação da mesma e na qual a descrição cinemática é Arbitrária Lagrangeana-Euleriana, enquanto que no

restante do domínio a descrição é puramente Euleriana. Essa organização do domínio permite um melhor desempenho computacional, pois restringindo a movimentação da malha ao subdomínio ALE, restringe-se também a parcela do domínio que sofre remalhamentos.

Existem duas formas básicas de se considerar o acoplamento fluído-estrutura: o tratamento simultâneo ou monolítico e o tratamento particionado. No tratamento monolítico, malha e estrutura são considerados como uma única entidade, sendo todos os sistemas integrados no tempo simultaneamente (AZEVEDO, 1999).

No tratamento particionado, os subsistemas são tratados com entidades diferentes e integrados no tempo de forma seqüencial, permitindo o uso de esquemas de solução diferentes (explicito / implícito) e passos de tempo diferentes no processo de integração temporal para o fluído e para a estrutura, em um esquema de sub-ciclos, e mesmo de malhas não conformes para fluído e estrutura. (TEIXEIRA, 2001, BRAUN, 2007).

1.8 OBJETIVOS E METODOLOGIA DO PRESENTE TRABALHO

O objetivo deste trabalho é estudar algoritmos paralelos para solução de problemas de Mecânica dos Sólidos, Mecânica dos Fluídos e Interação Fluído-Estrutura empregando o Método dos Elementos Finitos que possam ser utilizados tanto em um conjunto heterogêneo de computadores disponíveis em um laboratório de pesquisa e desenvolvimento conectados através da rede lógica existente sob a forma de um *cluster* temporário como em um *cluster* especificamente configurado para processamento de alto desempenho, de forma a obter uma velocidade de processamento significativamente maior que a correspondente de um único computador, e permitir a solução de problemas com grandes estruturas de dados inviáveis em apenas um computador.

O critério fundamental para a seleção dos algoritmos adequados a cada problema foi a possibilidade desses algoritmos serem paralelizados em um ambiente de memória distribuída, a fim de que pudessem ser utilizados em *clusters* temporários de computadores pessoais ou em *clusters* permanentes de baixo custo construídos a partir de computadores pessoais. Nessas condições, a existência de computadores com mais de um processador ou com processadores com mais de um núcleo de processamento (*core)* não foi priorizada na escolha dos algoritmos em função da facilidade de se adaptar um algoritmo otimizado para memória distribuída de

forma a ele se tornar eficiente também em um ambiente de memória compartilhada. Os algoritmos selecionados para as diversas aplicações são todos eles do tipo iterativo e foram desenvolvidos em trabalhos anteriores de mestrado e doutorado no âmbito do CEMACOM - Centro de Mecânica Aplicada e Computacional do Curso de Pós-Graduação em Engenharia Civil da Universidade Federal do Rio Grande do Sul, em especial os trabalhos de Burbridge (1999), Duarte Filho (2002), Braun (2007) e Bono (2008).

Optou-se por uma programação paralela que pudesse ser utilizada com uma mínima dependência do sistema operacional ou da estrutura do hardware. Desta forma, toda a lógica de distribuição de tarefas, balanceamento das cargas de trabalho e organização da comunicação dos dados entre computadores está implementada diretamente no código, sem nenhuma outra camada de software de gerenciamento. Todos os códigos foram implementados em FORTRAN 90. A comunicação de dados entre computadores é feita utilizando-se a API MPI. Como em processos iterativos os pontos de comunicação de mensagens são também pontos de sincronização do algoritmo, todas as comunicações são empregadas no modo blocante. O sistema operacional utilizado é o Windows XP, unicamente por ser o sistema operacional padrão do laboratório de computação do CEMACOM / UFRGS, evitando desta forma alterações nos computadores e interferência na forma de trabalho dos usuários. Como a única exigência em relação ao sistema operacional é o suporte à comunicação em rede e à API MPI, e como tanto FORTRAN 90 quanto MPI têm padrões que garantam sua portabilidade, os códigos desenvolvidos podem ser utilizados em outras plataformas e outros sistemas operacionais. Para uso de múltiplos processadores ou múltiplos núcleos em um único computador em um ambiente de memória compartilhada, o suporte é fornecido pelos compiladores através das APIs MPI e OpenMP, exigindo do sistema operacional apenas o suporte a multi-processamento.

Considerando que a rede de comunicação existente em laboratórios de pesquisa é, muitas vezes, de um padrão lento (*Fast Ethernet* com taxa de transferência de 100 Mbps), a implementação paralela foi feita de forma a se ter a granularidade mais grossa possível, procurando minimizar o tempo de comunicação de dados entre processadores frente ao tempo de processamento e garantir a maior eficiência de paralelização. Em um processo iterativo e paralelo, isso seria obtido com apenas uma comunicação entre os processos ao final de cada iteração para que cada processador recebesse o valor atualizado das variáveis nodais

processadas pelos outros processadores envolvidos. Nos algoritmos estudados, isso não foi possível, sendo necessários de 2 a 3 processos de comunicação por iteração.

Pelas mesmas limitações, o modelo de programação paralela utilizado é o de programa sem mestre (*hostless program)*, de modo que todos os processadores executam o mesmo código e as trocas de dados são efetuadas diretamente pelos processadores uns com os outros sem a intervenção de nenhum controlador. A opção por este modelo é adequada a redes relativamente pequenas com uma topologia em estrela, como a de computadores conectados diretamente a um único *switch* de rede, pois descentraliza e agiliza o processo de comunicação entre os processadores por possibilitar comunicações simultâneas, e diminui o tráfego de dados na rede em comparação a um modelo mestre-escravo em que os dados precisam vir de um computador escravo para o computador mestre para então serem enviados novamente a outro computador escravo. A maior complexidade do gerenciamento das comunicações resultante foi resolvida com um esquema genérico de fácil implementação e bom desempenho desenvolvido a partir de um padrão de comunicação circular e alternado. Nas implementações feitas, a leitura de dados é feita individualmente por cada processador a partir de seu disco rígido local, e o valor das variáveis nodais é agrupado em um dos computadores ao final do processo de solução ou a cada determinado número de passos nos processos de integração no tempo.

A divisão de tarefas utilizada empregou o particionamento nodal da estrutura de dados, baseado no pressuposto de que o esforço computacional associado a cada nó da malha é constante, e em um indicador inicial da capacidade de processamento de cada processador, como por exemplo a freqüência do processador. Desta forma, a malha de elementos finitos correspondente ao problema foi tratada como um sistema de equações desacopladas dentro de cada iteração dos processos de solução, e o problema de dividir tarefas entre processadores foi trabalhado como dividir equações. A divisão da malha segundo sua menor dimensão geométrica para definir as cargas de trabalho de cada processador com a menor interface possível e, portanto, menor comunicação de dados, foi tratada como dividir um sistema de equações segundo sua menor dimensão. Para tanto, dois tratamentos de malhas, baseados em reordenações nodais para minimização da largura de banda da matriz do sistema de equações, foram utilizados para minimizar o volume de comunicação de dados no processo paralelo de solução. Esses tratamentos podem ser considerados como variantes do método global de particionamento em faixas (*stripwise partitioning – STRIP)*(DORNELES, 2003) e do método

de bissecção coordenada recursiva ou RCB (*Recursive Coordinate Bissection*) (FOX et al. 1994). Elementos fantasmas (*ghosts elements*) são empregados para minimizar a comunicação entre processadores, às custas de redundância no esforço computacional (Elias et al. 2008, Demkowicz et al., 2007).

O balanceamento da cargas de trabalho é feito no início do processo de solução de forma estática simulando-se um determinado número de iterações e redistribuindo os nós da malha entre processadores em função do tempo de processamento individual até que o equilíbrio seja alcançado. Esta metodologia é aplicável a processos de solução em que o esforço computacional é constante ao longo das iterações. (KWOK et al., 1999)

A estrutura de dados utilizada foi a Elemento-por-Elemento (*Element-by-Element* ou *EBE*) para todos os algoritmos, evitando a necessidade de montagem de matrizes globais que, em geral, inviabilizam a análise de problemas grandes pelas necessidades de memória a elas associadas.

Uma vez otimizados para paralelismo de memória distribuída, os códigos de algumas aplicações foram modificados para execução em configurações de memória compartilhada com o uso da API OpenMP. Todos os laços de nós e elementos não relacionados a operações de entrada e saída foram tornados paralelos, e estruturas de dados auxiliares foram criadas quando necessárias para remover a dependência de dados, evitando o alto custo computacional associado às operações de sincronização.

Os algoritmos selecionados para paralelização são o Método dos Gradientes Conjugados para a solução de sistemas de equações lineares com os precondicionadores Diagonal e de Cholesky, o esquema explícito de Taylor-Galerkin com um passo e iterações utilizado na solução de problemas de escoamentos compressíveis em regime transônico e supersônico, incluindo considerações sobre o emprego da técnica de sub-ciclos para integração no tempo com múltiplos passos de tempo, o esquema explícito de Taylor-Galerkin de dois passos utilizado na solução de problemas de escoamentos incompressíveis em regime sub-sônico e o algoritmo para solução de problemas de Interação Fluído-Estrutura usando acoplamento particionado em que o fluído é resolvido pelo esquema explícito de Taylor Galerkin de 2 passos e a estrutura pelo método implícito de Newmark no contexto α-generalizado. Os algoritmos foram implementados para problemas tridimensionais empregando elementos

finitos hexaédricos ou tetraédricos com integração reduzida realizada de forma analítica para maior velocidade computacional (Engelmam et al. 1982, Shultz, 1997, Duarte filho, 2002)

1.9 ORGANIZAÇÃO DO TRABALHO

O presente texto está organizado da seguinte forma:

No presente capítulo é feita uma apresentação do problema de pesquisa, uma contextualização de sua importância, e são abordados principais tópicos envolvidos na paralelização de códigos computacionais científicos para simulação numérica, em especial com o emprego do Método dos Elementos Finitos, e sobre os algoritmos para a solução de problemas de Dinâmica dos Sólidos e Dinâmica dos Fluídos com ênfase para grandes estruturas de dados. Finalmente, são indicados os objetivos e apresentada a metodologia empregada.

No capítulo 2 é discutido o problema da divisão de tarefas na paralelização de códigos de elementos finitos, em especial para configurações de máquinas paralelas com memória distribuída, abordando aspectos relacionados à divisão da estrutura de dados através de partição nodal, redundância do esforço computacional, comunicação de dados entre processadores e previsão da velocidade de processamento dos mesmos. Dois processos de divisão de tarefas baseados em reordenação nodal são propostos para minimizar a comunicação de dados e a redundância de esforço computacional, e o padrão de comunicação entre processos decorrente de cada um deles é avaliado para identificação de cenários de aplicação. Por fim, uma forma de balanceamento das cargas de trabalho para soluções iterativas é proposta.

No capítulo 3 é apresentado o problema do ordenamento dos processos de comunicação quando há a necessidade de troca de dados entre um número muito grande de processadores.

No capítulo 4 são mostrados os algoritmos empregados na solução de problemas de Dinâmica dos Sólidos e Dinâmica dos Fluídos que foram selecionados para aplicações paralelas em configurações de memória distribuída., detalhando o número de processos de comunicação necessários e etapa do algoritmo onde os mesmos ocorrem, e quais os dados envolvidos em tais processos.

No capítulo 5 são mostrados os resultados de desempenho paralelo obtidos com os algoritmos apresentados no capítulo 4 em diversas configurações de memória distribuída. Testes com *clusters* temporários e permanentes, homogêneos ou não quanto à capacidade de processamento dos computadores envolvidos, com diferentes números de computadores e diferentes velocidades de rede, com estruturas de dados de diversos tamanhos com malhas estruturadas e não estruturadas são apresentados e os resultados obtidos discutidos.

No capítulo 6, o algoritmo do esquema de Taylor-Galerkin de um passo com iterações para a solução de problemas de escoamento transônico e supersônico, implementado para elementos tetraédricos e hexaédricos, é utilizado como base para uma série de testes em configurações de memória compartilhada e configurações híbridas de memória compartilhada e distribuída. Os resultados obtidos são discutidos em função do tipo do elemento, tamanho do problema e forma de paralelismo empregada.

Finalmente, no capitulo 7 são apresentadas conclusões e recomendações para trabalhos futuros.

2 PARTICIONAMENTO DOS DADOS E BALANCEAMENTO

2.1 PRINCÍPIOS BÁSICOS

Em um código paralelo, é comum a existência de *barreiras de sincronização*, etapas do algoritmo implementado em que todos os processadores lógicos envolvidos no processamento precisam alcançar para que cada um, individualmente, possa prosseguir na execução do código. As *barreiras de sincronização* são geralmente implementadas quando o valor de uma dada variável calculada por um determinado processador lógico é necessária para o processamento dos dados de um ou de todos os outros processadores envolvidos.

Um dos fatores chave para a obtenção do máximo desempenho computacional em um código paralelo é garantir que em nenhum momento os processadores lógicos envolvidos estejam ociosos. Quando uma divisão de tarefas ou carga de trabalho computacional ótima não é obtida, alguns processadores irão processar as tarefas a eles alocadas mais rapidamente que outros, aguardando pelos demais em uma *barreira de sincronização.* Como a execução do código não prossegue sem que todos os processadores tenham alcançado a *barreira de sincronização*, o tempo total de processamento do conjunto é o tempo de processamento do processador mais lento, aqui entendido como aquele que leva mais tempo para processar as tarefas a ele alocadas.

Em um conjunto de processadores idênticos, a divisão de tarefas ótima seria a divisão uniforme, onde cada laço do código computacional teria suas iterações igualmente divididas entre os processadores presentes. Para que as tarefas alocadas fossem absolutamente idênticas, os laços não poderiam conter instruções condicionais.

Se a divisão uniforme dos laços pode ser eficientemente obtida em máquinas paralelas de memória compartilhada, o mesmo não pode ser dito para máquinas paralelas de memória

distribuída, pois não é possível utilizar com eficiência uma estratégia de paralelização baseada na divisão de cada laço do programa.

Considere-se um código de Elementos Finitos para análise de problemas de Mecânica dos Sólidos. Enquanto uma série de laços relacionados a elementos ou nós são homogêneos, realizando as mesmas operações para todos os elementos ou nós envolvidos, outros, relacionados por exemplo a condições de contorno ou a cargas, não o são. Fenômenos não lineares como plasticidade e dano, por serem de característica local, também afetam a carga de trabalho computacional associada a cada nó ou elemento de forma desigual. Garantir que todos os laços sejam divididos igualmente entre os processadores corresponde a atribuir a um dado processador um grupo de elementos em um laço que calcula as matrizes de rigidez, e um grupo diferente de elementos em um laço ao calcular cargas de volume ou de superfície. Grupos diferentes de elementos e nós atribuídos a cada processador a cada laço levam a uma intensa troca de dados entre os processadores envolvidos, pois as variáveis correspondentes a um determinado nó em um dado laço de um dado processador têm de ser passadas para um processador diferente no laço seguinte. Em uma máquina paralela com memória distribuída, uma troca intensa de informações entre processadores corresponde a tráfego intenso na rede de intercomunicação, que tende a ser lenta e de alta latência, levando, por conseqüência, a um código ineficiente. As mesmas considerações podem ser aplicadas a códigos de Elementos Finitos para análise de problemas de Mecânica dos Fluídos ou de Interação Fluído-Estrutura.

Quando o conjunto é composto por processadores com desempenhos computacionais desiguais, como é o caso de *clusters* formados por nós heterogêneos e de máquinas de memória compartilhada com processadores cujos núcleos físicos podem ser divididos em dois ou mais núcleos lógicos com diferente poder de processamento, a divisão de tarefas uniforme deixa de ser ótima e a diferença de velocidade de processamento dos participantes precisa ser considerada. Em máquinas de memória compartilhada, isso pode ser feito dividindo cada laço do código em um grande número de grupos com um número pequeno de iterações. Inicialmente um grupo de iterações é designado a cada processador e um novo grupo é alocado de forma independente a cada processador na medida em que o processamento do grupo anterior é encerrado. Processadores mais rápidos processam um número maior de grupos de iterações que processadores mais lentos. Contudo, em máquinas de memória distribuída, essa abordagem somente pode ser implementada às expensas de uma intensa

comunicação através da rede e, na maioria dos casos, ineficiência do desempenho computacional.

Para que uma implementação paralela seja igualmente eficiente em máquinas de memória compartilhada e distribuída, uma alternativa possível é que a divisão de tarefas seja baseada em grupos de variáveis alocadas aos diversos processadores que permaneçam constantes ao longo do processamento. Especificamente em códigos de Elementos Finitos, uma divisão de tarefas baseada em nós ou em elementos, na qual cada processador fica responsável por um mesmo grupo ao longo de todo o processamento, parece adequada.

Contudo, se uma divisão de tarefas variável ao longo das várias partes do código a ser processado traz consigo um alto tráfego de dados entre os processadores, a manutenção de uma divisão de tarefas única ao longo de todo o código traz consigo o problema de quantificar *a priori* a velocidade de processamento relativa de cada processador em configurações não homogêneas, em que os processadores lógicos tem diferentes capacidades de processamento. Do ponto de vista do *hardware* , a arquitetura de cada processador, sua freqüência, quantidade de memória *cache* e a quantidade de memória principal são fatores que devem ser levados em consideração para a determinação da velocidade relativa. Do ponto de vista do código a ser processado, a existência de laços baseados em elementos e laços baseados em nós entre *barreiras de sincronização* torna difícil a escolha de sobre quais desses elementos a divisão de tarefas será baseada. O fato de alguns elementos ou nós trazerem consigo uma carga de trabalho computacional maior pelo fato de a eles estarem associados, por exemplo, cargas, condições de contorno ou fenômenos físicos locais dificulta a divisão do conjunto total de dados entre os processadores envolvidos em grupos cuja velocidade de processamento relativa possa ser prevista.

2.2 DIVISÃO DE TAREFAS BASEADAS EM NÓS

No Método dos Elementos Finitos, toda carga de trabalho computacional está associada aos nós e aos elementos nos quais o domínio foi discretizado. Qualquer um deles poderia ser utilizado como parâmetro para a divisão de tarefas computacional em uma solução paralela. Uma divisão baseada em nós é chamada de particionamento nodal, ao passo que a baseada em elementos é chamada de particionamento dual (ELIAS et al. 2008). Como as tarefas designadas a cada processador lógico não são independentes, faz-se necessário que cada

processador comunique a alguns outros ou a todos os demais parte ou a totalidade dos resultados obtidos no seu processamento, e da mesma forma, receba o resultado de outros, tornando a comunicação de dados entre os processadores implícita à divisão de tarefas. Quando a comunicação de dados entre processadores tem um custo computacional muito grande, pode ser vantajosa uma implementação do código com redundância, onde cada processador trabalha também sobre parte da tarefa atribuída a outros processadores para diminuir a necessidade de comunicação de dados com os mesmos. Ambas alternativas, comunicação de dados e redundância, implicam em perda de eficiência computacional, uma vez que parte do tempo total de processamento é gasto ou em comunicação de dados ou no processamento de um mesmo conjunto de dados por mais de um processador de forma redundante. O balanço entre uma maior ou menor presença de comunicação de dados e redundância no código implementado depende do algoritmo a ser paralelizado e do tipo de máquina paralela para o qual o código é otimizado. Em máquinas de memória compartilhada, a eliminação da redundância parece vantajosa, visto que a comunicação de dados se dá através da memória, com um custo computacional baixo. Por outro lado, em máquinas paralelas de memória distribuída onde processadores poderosos são interconectados por uma rede lógica comparativamente lenta, a minimização da comunicação entre processadores, mesmo às custas de redundância, parece ser a alternativa mais eficiente.

As variáveis nodais são as variáveis em função das quais o sistema de equações resultante é montado. Nos métodos explícitos para Mecânica dos Fluídos, dentro de cada passo na qual o tempo é discretizado, as equações que representam a discretização espacial são desacopladas, o que permite a solução independente de sub-conjuntos das mesmas, sem nenhuma dependência com a ordem de solução das equações. Na solução de sistemas de equações usando processos iterativos, as equações são também desacopladas dentro de cada iteração, permitindo a separação em sub-conjuntos e independência na ordem de solução. Ambas as situações podem ser caracterizadas por:

$$x_i^{j+1} = f\left(x_1^j, x_2^j, \ldots, x_n^j\right) \tag{2.1}$$

onde x_i representa o valor da i-ésima equação nodal, j representa a iteração j (na solução iterativa de um sistema de equações) ou o tempo t (na discretização temporal) e $j+1$ representa a iteração seguinte ou o tempo $t+\Delta t$.

Essas características apontam para uma divisão de tarefas baseadas nos nós (mais especificamente nas variáveis nodais), vendo a solução do problema em elementos finitos como um conjunto de equações independentes que pode ser divido em sub-conjuntos menores resolvidos simultaneamente dentro de cada iteração ou passo de tempo.

Os elementos nos quais a malha é discretizada são os responsáveis pelo acoplamento das equações entre cada iteração. Para que se possa calcular o valor das variáveis nodais x_i^{j+1} em um sub-conjunto qualquer em uma iteração *j+1*, é preciso não somente conhecer o valor dessas mesmas variáveis ao final de iteração *j* mas também de todas as demais variáveis pertencentes a outros sub-conjuntos e que a elas estejam acopladas, ou seja, o valor das variáveis nodais de todos os nós pertencentes a elementos que estejam conectados ao conjunto de nós atribuídos a um dado processador. Desta forma, ao se dividir uma determinada malha de elementos finitos em grupos de nós alocados a processadores lógicos diferentes, o conjunto de dados a ser comunicado entre os processadores fica automaticamente definido como o valor das variáveis nodais correspondentes aos nós de outros grupos pertencentes aos elementos conectados os nós do grupo considerado.

Mesmo o sistema de equações resultante sendo postulado em nós (ou variáveis nodais), boa parte da implementação do Método de Elementos Finitos resulta em esforço computacional associado aos elementos (como o cálculo das matrizes de rigidez e a organização de soluções iterativas de sistemas de equações em esquemas elemento a elemento). Como muitas vezes é difícil implementar códigos nos quais o processamento possa ser feito nos elementos somente para os nós pertencentes a um dado grupo, e não para o elemento como um todo, a divisão de tarefas baseada em nós acarreta em redundância, representada pelos elementos conectados aos nós de um grupo mas que estão conectados também a nós de outros grupos, os *elementos comuns* ou elementos fantasmas (*ghost elements*). O esforço computacional associado aos *elementos comuns* é feito de forma redundante por todos processadores cujos grupos contenham nós conectados a esses elementos.

As características associadas a essa divisão de tarefas pode ser vista na figura 2.1, onde o esforço computacional referente a uma malha com 24 nós e 15 elementos é dividido igualmente entre dois processadores lógicos, cada um responsável por 12 nós (conjuntos azul, e amarelo). A divisão é feita segundo a ordenação nodal existente na malha (por linha), com o primeiro grupo sendo composto pelos 12 primeiros nós e o segundo pelos 12 seguintes. Com a mesma cor dos nós estão os elementos associados exclusivamente a cada processador, e em

cinza, os *elementos comuns*. Os retângulos azul e amarelo externos abrangem a totalidade de elementos próprios e comuns sob a responsabilidade de cada processador, bem como os nós cujas variáveis são necessárias às equações alocadas a cada processador, ou seja, os nós do grupo de cada processador e os nós de outros grupos conectados a elementos comuns conectados aos nós do grupo de cada processador.

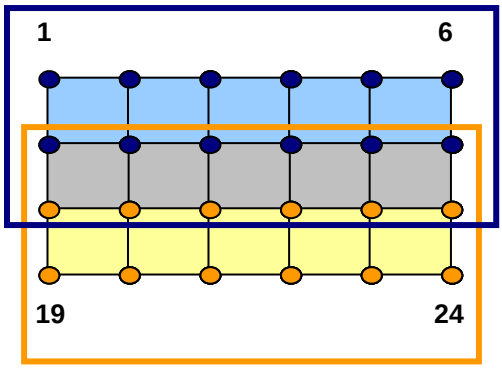


Figura 2.1: Partição nodal da malha baseada em um ordenação arbitrária dos nós.

Por esta divisão de tarefas, o processador lógico azul fica responsável pelos nós 1 a 12 e por 10 elementos, e o processador amarelo pelos nós 13 a 24 e por 10 elementos.. Assim, os processadores trabalham sobre 20 elementos, contra 15 se a malha inteira fosse processada por um único processador, levando a uma eficiência no processamento dos elementos de 15/20 = 75%. Em termos de comunicação de dados, o processador azul precisa das variáveis nodais associadas a 6 nós do processador amarelo, e este precisa de 6 nós do processador azul, totalizando um tráfego em rede das variáveis nodais associadas a 12 nós a serem comunicadas entre os processadores a cada iteração do processo de solução.

O problema da divisão de tarefas pode ser visto geometricamente como um problema de particionar uma malha, mas também pode ser visto de forma equivalente como particionar um sistema de equações. A malha representada na figura 2.1 é equivalente ao sistema de equações mostrado de forma simbólica na figura 2.2. A montagem é baseada na ordenação nodal, de modo que o processador 1 ficou responsável pelas equações de 1 a 12, que correspondem aos nós 1 a 12, e o processador 2 as equações de 13 a 24, correspondentes aos nós 13 a 24.

Em um processo de solução iterativo, o valor de uma variável nodal na iteração *i+1*, representada pelas posição correspondente na diagonal principal, seria calculado a partir do valores de todas as variáveis a elas relacionada, representadas pelas posições em cinza na linha correspondente. Contudo, nem todas as variáveis nodais necessárias estão alocadas ao

mesmo processador. Para exemplificar, 2 variáveis nodais pertencentes ao grupo do processador 1, marcadas em azul escuro, necessitam das variáveis indicadas em amarelo claro, as quais pertencem ao processador 2. As posições correspondentes dessas variáveis estão marcada em laranja na diagonal principal do sistema. Fica claro perceber que o número de variáveis nodais pertencentes ao processador 2 que são necessárias ao processador 1 estão relacionadas com a semi-largura de banda do sistema.

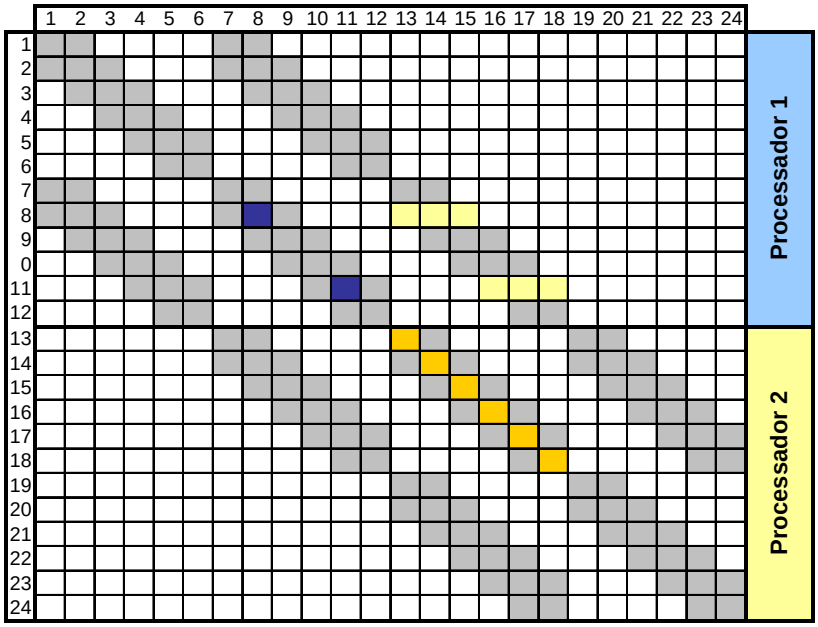


Figura 2.2: Partição de um sistema de equações baseada em um ordenação arbitrária dos nós.

Do ponto de vista geométrico, o particionamento que corresponde à menor interface entre as partes (menor número de *elementos comuns*, associado à menor redundância de esforço computacional, e menor número de nós de interface, associados ao menor tráfego de dados pela rede) é aquele representado pelo plano normal à maior dimensão da malha. Se o particionamento é feito através da ordem dos nós, a melhor divisão corresponderia à ordenação mostrada na figura 2.3 (por coluna).

Cada processador fica responsável por 9 elementos. A eficiência de processamento dos elementos correspondente a essa divisão é de 15 / 18 = 83,3% e o tráfego em rede é de 8 nós para cada iteração do processo de solução.

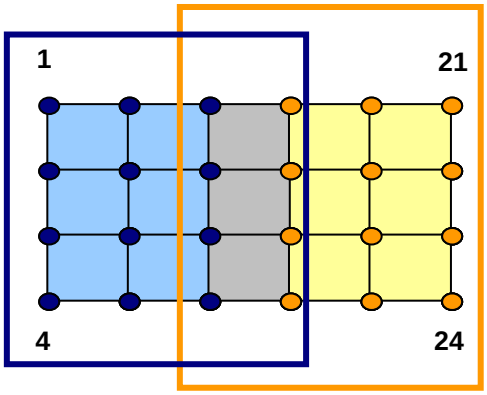


Figura 2.3: Partição nodal da malha baseada em um ordenação ótima dos nós.

Do ponto de vista do sistema de equações, mudar a orientação do plano geométrico para que a divisão seja feita com a menor interface possível é equivalente a mudar a ordem das variáveis para que a divisão encontra a menor largura de banda possível. O sistema de equações correspondente à figura 2.3 está mostrado esquematicamente na figura 2.4. A menor semi-largura de banda corresponde a um menor número de variáveis nodais necessárias às equações do processador 1 pertencentes ao processador 2, e vice-versa.

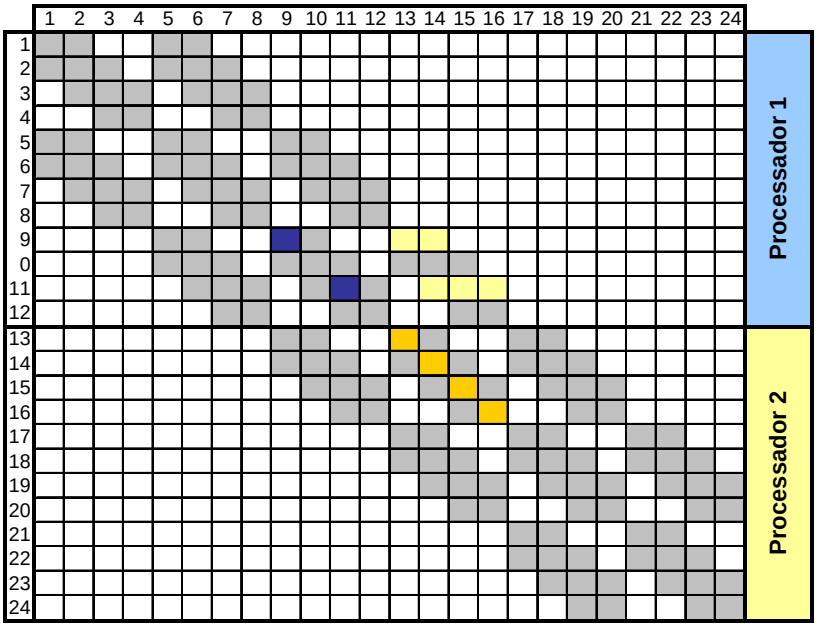


Figura 2.4: Partição de um sistema de equações baseada em um ordenação ótima dos nós.

Se o sistema de equações tem uma largura de banda constante, atribuir conjuntos não uniformes de equações aos processadores leva ao mesmo número de variáveis que devem ser comunicadas entre os processadores e ao mesmo número de *elementos comuns*. Havendo

variação da largura de banda ao longo do sistema, uma ordenação eficiente garante que ela é mínima em qualquer ponto, e portanto a divisão em 2 partes corresponderá sempre à melhor opção de divisão, qualquer que seja o número de equações atribuído a cada parte.

Considerando que o esforço computacional associado à solução de cada equação é uniforme, uma divisão de tarefas entre processadores heterogêneos pode ser feita atribuindo-se a cada uma delas um número de equações (ou de nós) proporcional à sua capacidade de processamento, seguindo diretamente a ordenação nodal ótima. Na figura 2.5 a ordenação nodal anterior é utilizada para dividir tarefas entre dois processadores lógicos com capacidades de processamento relativas de 1 e 1,4, de forma que o primeiro processador fica responsável por 10 nós e o segundo por 14. A eficiência de processamento de elementos é de 15/19 = 78,9%, e o tráfego na rede de 10 nós para cada iteração do processo de solução. A divisão não uniforme de tarefas pode ser decorrente tanto da presença de processadores lógicos com velocidades de processamento diferentes quanto de condições de contorno, de carregamento e de não linearidade física ou geométrica não uniformemente distribuídas pela malha, fazendo com que o processamento de um pequeno grupo de nós submetidos a condições diferenciadas exija tanto esforço computacional quanto um grupo maior sem essas condições.

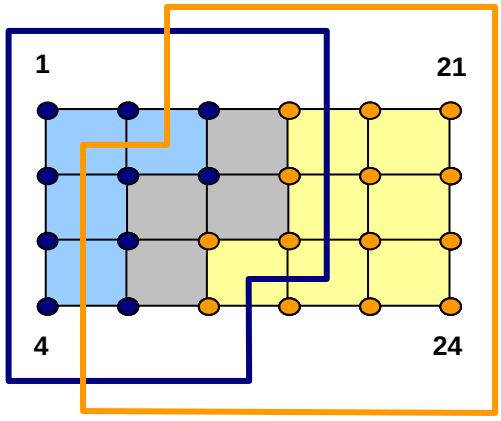


Figura 2.5: Partição nodal não uniforme da malha baseada em um ordenação ótima dos nós.

O problema de encontrar uma ordenação nodal eficiente para minimizar o número de *elementos comuns* (associados à redundância de esforço computacional) e de nós ligados a estes elementos (associados ao tráfego em rede) geometricamente pode ser entendido como subdividir a malha segundo sua menor dimensão e, do ponto de vista do sistema de equações associado, subdividir o sistema onde ele tenha a menor largura de banda. Ordenar os nós da malha para que as divisões ocorram segundo um menor número de nós e elementos é em tudo

similar ao problema de minimização de largura de banda da matriz de rigidez global enfrentado pelos *solvers* diretos. Os algoritmos de reordenação nodal desenvolvidos para esses *solvers* atuam sobre o sistema de equações ordenando os nós segundo a direção de menor largura (seja ela física da malha ou relacional dos nós), fazendo com que, se aplicadas para o problema de divisão de tarefas, tornem a malha como que fisicamente "estreitas" e propiciem divisões com o menor número de *elementos comuns* e de nós de interface.

2.3 OTIMIZAÇÃO DA DIVISÃO DE TAREFAS ATRAVÉS DE UMA ÚNICA REORDENAÇÃO NODAL PRÉVIA

O primeiro tratamento proposto para otimizar a divisão de tarefas baseada em nós consiste na execução de uma única reordenação nodal prévia da malha através de um algoritmo minimizador de banda, com os grupos de nós alocados a cada processador lógico definidos por uma divisão da malha original segundo a ordenação nodal obtida e proporcional ao desempenho relativo de cada processador. O algoritmo de minimização utilizado baseou-se no trabalho de Teixeira (1991), e é composto por uma reordenação de elementos, segundo os critérios de Silvester e Auda (1984) para reduzir o *Front* (IRONS, 1970, HINTON e OWEN, 1977), seguida de uma reordenação nodal segundo os critérios de Hoit e Willson (1983) em seu algoritmo PFM (*Profile-Front Minimization*). Como índice de desempenho relativo de cada núcleo utiliza-se inicialmente a freqüência do processador. Este tratamento será referenciado doravante como "1RN".

Pode-se considerar o tratamento 1RN como uma variante do método global de particionamento em faixas (*stripwise partitioning*) conhecido como STRIP (DORNELES, 2003). Enquanto o STRIP divide a malha geometricamente segundo faixas horizontais ou verticais proporcionais à capacidade de processamento de cada computador, o tratamento 1RN divide o sistema de equações com a largura de banda minimizada em faixas proporcionais à capacidade de processamento de cada processador. Como o sistema de equações é tratado como uma lista de equações, dividir em faixas corresponde a simplesmente dividir a lista ordenada segundo o minimizador de banda em grupos com número de equações proporcionais a capacidade de cada processador.

O tratamento 1RN é bastante eficiente na redução do número de *elementos comuns* e de nós no tráfego em rede para um número pequeno de processadores lógicos em malhas

geometricamente alongadas, que tenham uma direção predominante (grandes sistemas de equações com largura de banda pequena). Com este tratamento, desde que o número de equações alocadas a cada processador seja maior que a semi-largura de banda do sistema, cada grupo de nós tem interface com no máximo dois grupos vizinhos, correspondentes ao processador imediatamente anterior e imediatamente posterior. Esta característica é especialmente interessante para as seguintes configurações:

a) configurações nas quais a comunicação entre processadores depende mais da latência que da taxa de transferência (*throughput*), uma vez que um reduzido número de processos de comunicação é realizado (HENDDRICKSON e LELAND, 1995). Problemas pequenos, nos quais cada processador trabalha sobre uma quantidade reduzida de dados entre cada comunicação, e a quantidade de dados a ser comunicada entre os processadores é reduzida, em máquinas paralelas de memória distribuída, se enquadram nesta configuração;

b) configurações nas quais vários processadores lógicos compartilham uma única interface de rede, como é o caso de *cluster* formados por nodos com múltiplos processadores / múltiplos núcleos. A comunicação entre os processadores / núcleos de cada computador permanecem internas ao próprio computador, sendo feita através da memória e não sobrecarregando o tráfego em rede. As únicas comunicações que utilizariam a interface de rede seriam as do primeiro processador / núcleo do computador com o imediatamente anterior (externo) na ordem do cluster, e o do último processador / núcleo com o imediatamente posterior (externo) na ordem do cluster. Com uma única interface de rede para atender aos diversos processadores lógicos internos, minimizar a comunicação em rede com processadores lógicos externos ao computador é altamente desejável;

c) configurações de memória distribuída nas quais a rede de comunicação está dividida fisicamente em duas sub-redes, como *clusters* onde o número de portas do *switch* de rede disponível é menor que o número de nodos do cluster a serem conectados, necessitando de 2 ou mais *switches* independentes: qualquer nó do cluster poderia servir de ponte para comunicação entre duas redes, bastando para isso estar conectado a cada par de *switches* por uma interface de rede independente.

2.4 OTIMIZAÇÃO DA DIVISÃO DE TAREFAS ATRAVÉS DE REORDENAÇÕES NODAIS SUCESSIVAS

Quando o número de processadores lógicos pelos quais deve ser feita a divisão de tarefas se torna muito grande, o tratamento 1RN deixa de ser eficiente, pois a direção inicial na qual há um número mínimo de *elementos comuns* muda para cada grupo de nós à medida que os grupos vão sendo sub-divididos em grupos menores.

Na figura 2.6, percebe-se que a divisão da malha da figura 2.3 em quatro partes, correspondendo a quatro processadores lógicos de igual capacidade, resultou em 22 *elementos comuns* (4 para os grupos verde e lilás, 7 para os grupos azul e amarelo) e 28 nós a serem comunicados em rede, o que não corresponde à divisão de tarefas mais eficiente. Cada um dos grupos de nós da figura 2.3 foi subdividido em 2, mas não segundo a sua menor dimensão, e sim segundo a maior (vertical), seguindo a ordenação nodal (1 a 6, 7 a 12, 13 a 18, 19 a 24).

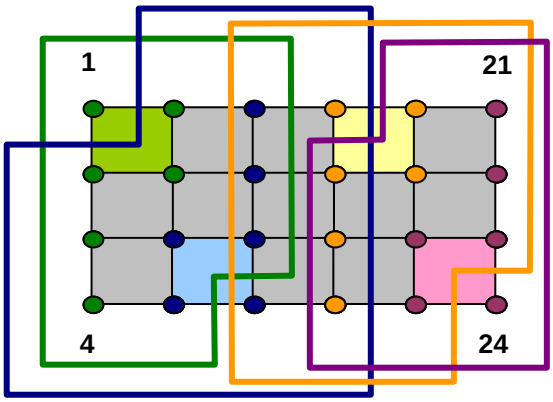


Figura 2.6: Divisão ineficiente de tarefas para quatro processadores lógicos segundo o tratamento 1RN.

A divisão geométrica da figura 2.6 corresponde à divisão do sistema de equações mostrada na figura 2.7. O processador 2 (em azul), necessita das equações nodais de 5 nós do processador 1 (em verde) e 4 do processador 3 (em amarelo).

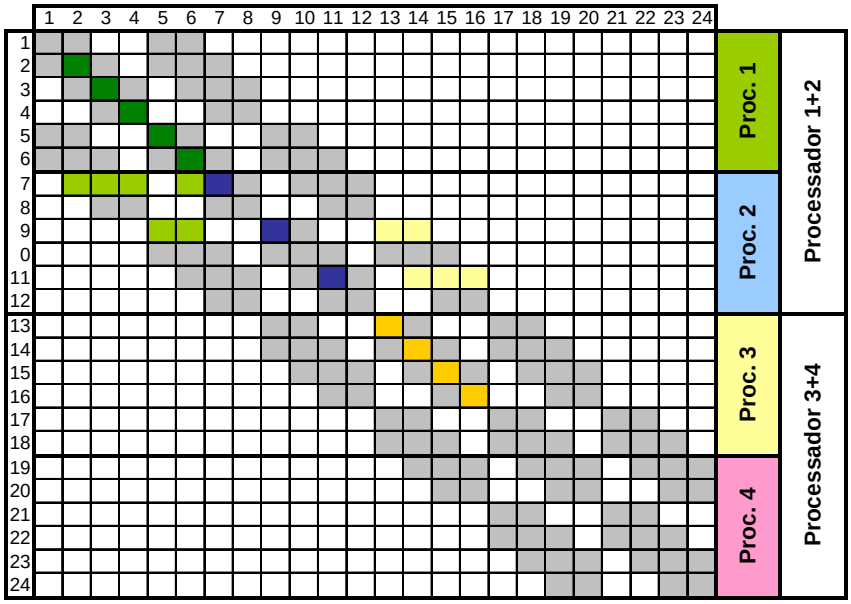


Figura 2.7: Partição ineficiente de um sistema de equações por quatro processadores lógicos.

Geometricamente, a melhor divisão seria obtida primeiro dividindo a malha como um todo por um plano normal à sua maior dimensão (vertical), repetindo o processo a seguir para cada uma da partes, consideradas como malhas independentes (plano horizontal). Essa divisão está mostrada na figura 2.8.

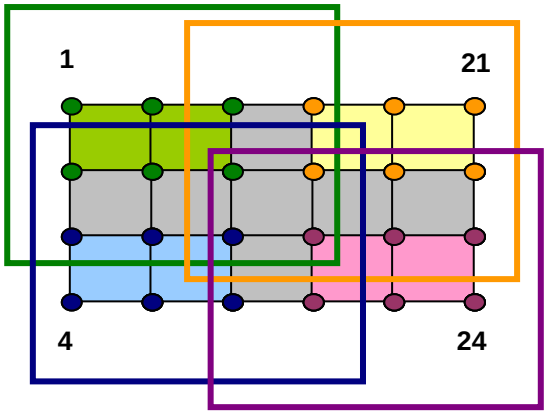


Figura 2.8: Divisão ótima de tarefas para quatro processadores lógicos

A divisão resultante, onde cada grupo da figura 2.4 foi subdividido segundo sua menor dimensão, é mais eficiente, com 16 *elementos comuns* (4 para cada grupo) e 24 nós a serem comunicados em rede.

Como o algoritmo de minimização de banda é equivalente a determinar a maior dimensão física da malha (sendo a separação das equações em grupos equivalente a seccionar a malha segundo o plano normal a essa dimensão), um particionamento equivalente ao geometricamente feito na figura 2.8 pode ser obtido reordenando-se a malha original uma primeira vez para a minimização da largura de banda e dividindo-se as equações em dois grupos, segundo essa ordenação, conforme figuras 2.9 e 2.10.

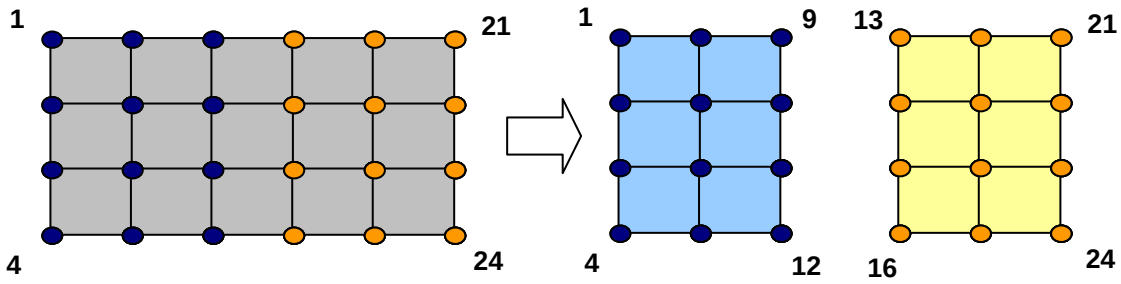


Figura 2.9: Divisão de tarefas por bissecção para quatro processadores lógicos – primeira etapa.

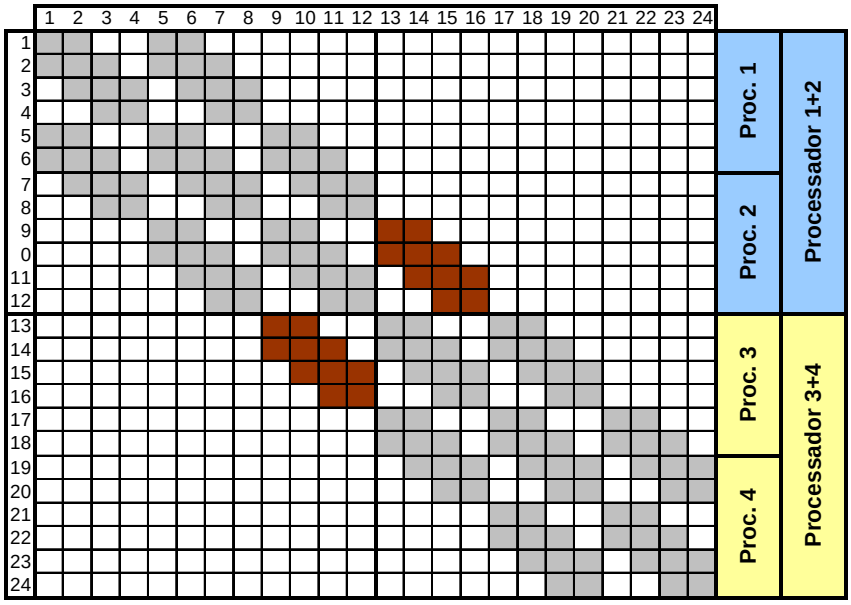


Figura 2.10: Partição do sistema de equações por bissecção para quatro processadores lógicos – primeira etapa.

Cada grupo passa a ser considerado geometricamente como uma malha independente e como um sistema de equações independentes. Os termos das equações que fazem a ligação entre as variáveis de cada grupo, marcados em marrom na figura 2.10, são considerados como não pertencentes a nenhum dos grupos.

Cada parte sofre então uma nova reordenação nodal para minimização da largura de banda, com os nós assumindo a nova numeração em azul nas figuras 2.11 e 2.12. Os termos em marrom, correspondentes às variáveis de um grupo que são necessárias às equações do outro grupo, acompanham a reordenação. Cada parte agora pode ser dividida em dois conjuntos de nós segundo a nova ordenação nodal de cada parte. Geometricamente isso dá origem à divisão de domínios mostrada na figura 2.8. A correspondente divisão no sistema de equações pode ser visto na figura 2.13.

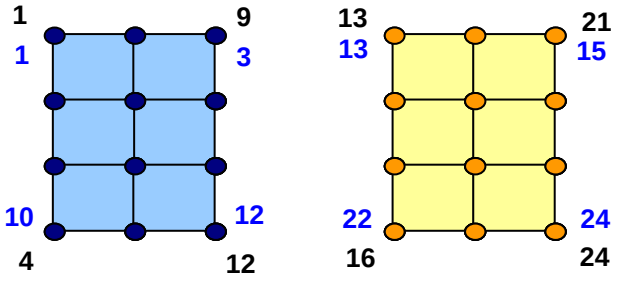


Figura 2.11: Reordenação nodal para cada parte da malha de forma independente.

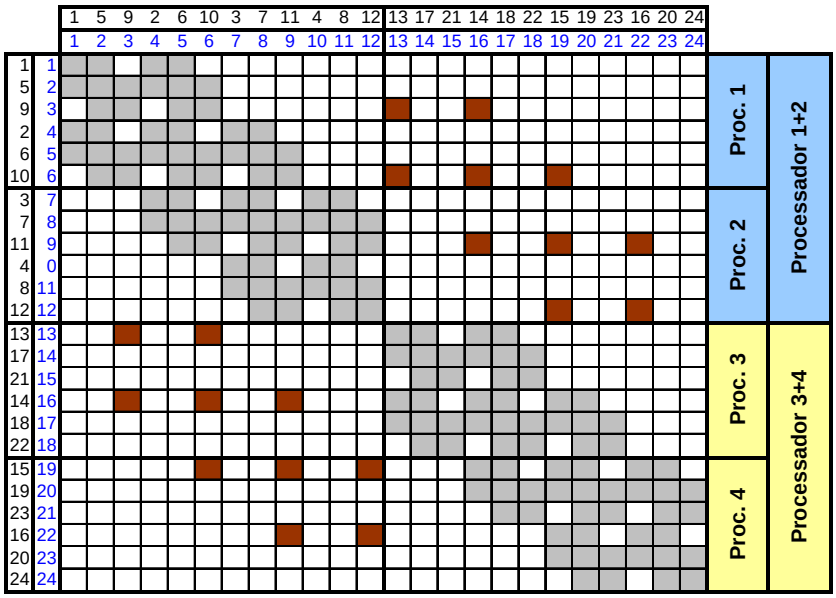


Figura 2.12: Reordenação nodal para cada parte do sistema de equações de forma independente.

Na figura 2.13 estão marcadas sobre a diagonal principal do sistema, em azul escuro, algumas das variáveis pertencentes ao segundo processador. Na linha correspondente a cada uma delas, estão marcadas as variáveis necessárias àquela equação e que pertencem a outros

processadores, na cor do respectivo processador. Marcados sobre a diagonal principal estão as variáveis correspondentes dos outros processadores.

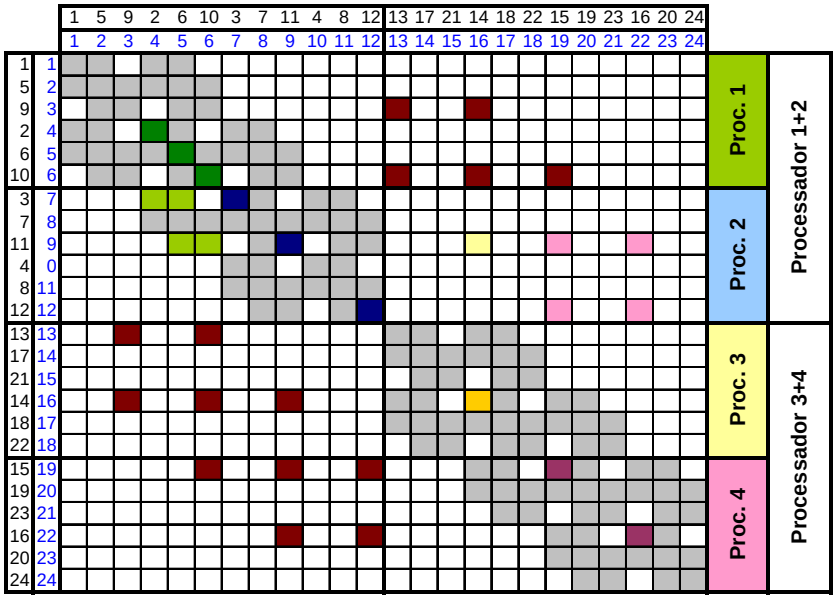


Figura 2.13: Sistema de equações dividido com um mínimo de comunicação de dados entre processadores.

Esse processo, denominado doravante de tratamento nRN (pois implica em *n* reordenações nodais) pode ser considerado como uma variante do método de bissecção coordenada recursiva ou RCB (*Recursive Coordinate Bissection*) (FOX et al. 1994), mas ao invés de ser aplicado geometricamente sobre a malha é aplicado diretamente sobre o sistema de equações, com a minimização da largura de banda fazendo o papel da identificação da direção de bissecção.

Para garantir que o processo possa ser aplicado para um número qualquer de processadores heterogêneos quanto à capacidade de processamento, e não apenas potências de 2, o seguinte algoritmo é utilizado:

a) Uma reordenação nodal para minimização da banda é feita para toda a malha, após o qual a mesma é subdividida em 2 grupos: um com o número de processadores correspondente à maior potência de 2 que seja inferior ao número total de processadores lógicos pelos quais deva ser feita a divisão, e um segundo correspondente ao restante dos processadores. O número de nós correspondente a

cada grupo é proporcional à soma dos desempenhos relativos dos processadores lógicos que a ele pertencem;

b) a mesma metodologia é aplicada novamente a cada um dos grupos de nós: inicialmente é feita uma reordenação nodal para minimização de banda somente dos nós pertencentes àquele grupo, após a qual o grupo é dividido em 2 subgrupos de nós, um correspondendo à maior potência de 2 que é inferior ao número de processadores lógicos do grupo, e outro subgrupo com o restante dos processadores. O número de nós da malha pertencente a cada subgrupo é proporcional à soma do desempenho relativo dos processadores lógicos pertencentes ao subgrupo;

c) o passo b) é repetido até que a malha original esteja dividida em tantos subgrupos quantos forem os processadores lógicos do conjunto.

A aplicação deste tratamento está esquematizada nas figuras 2.14 a 2.18, onde uma malha genérica original foi dividida em 15 partes, cada uma correspondendo a um processador lógico. Os números em cada parte correspondem ao número de processadores alocados em cada divisão e as linhas com dupla seta correspondem à reordenação nodal para minimização de banda, com a direção das setas indicando a direção de maior dimensão da malha para cada subdivisão ou parte da malha. A malha deve ser subdividida no sentido perpendicular ao da linha de dupla seta.

I. Malha original cujos nós devem ser divididos em 15 grupos, 1 para cada processador lógico (figura 2.14).

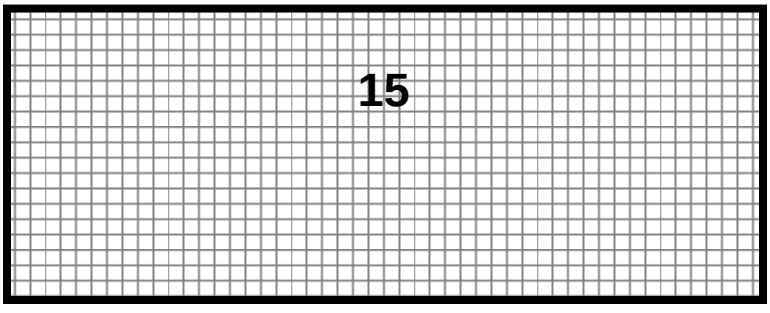


Figura 2.14: Malha inicial onde o tratamento nRN será aplicado .

II. Primeira reordenação nodal para minimização da largura de banda. O grupo original correspondente a 15 processadores lógicos é subdividido em 2, um correspondendo a 7 processadores e outro a 8, com número de nós em cada grupo proporcional à soma das

capacidades de processamento relativas dos correspondentes processadores do grupo (figura 2.15).

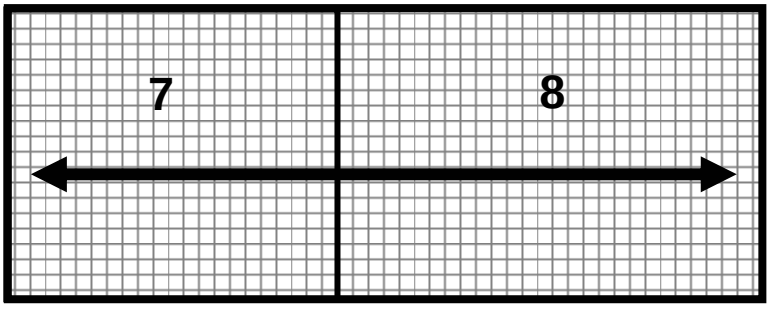


Figura 2.15: Primeira divisão do tratamento nRN.

III. O grupo de nós correspondente aos 7 processadores é reordenado como se fosse uma malha independente para minimização de banda e subdividido em um subgrupo de nós correspondente a 3 processadores outro a 4. O grupo correspondente a 8 processadores sofre igual processo (figura 2.16).

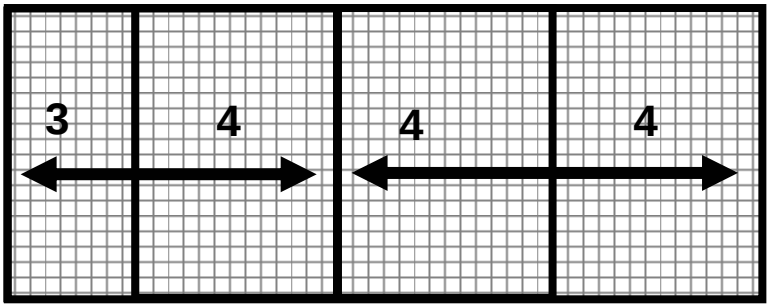


Figura 2.16: Segunda etapa de divisão do tratamento nRN.

IV. Cada subgrupo de nós é reordenado como se fosse uma malha independente e subdividido de forma proporcional à velocidade relativa dos processadores lógicos correspondentes (figura 2.17).

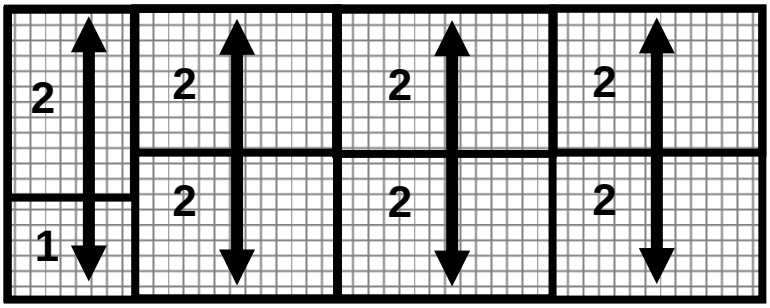


Figura 2.17: Terceira etapa de divisão do tratamento nRN.

V. As reordenações e subdivisões sucessivas são feitas até que se obtenha tantos subgrupos quantos processadores lógicos. A quantidade de nós da malha alocada a cada processador é proporcional a sua capacidade de processamento relativa.

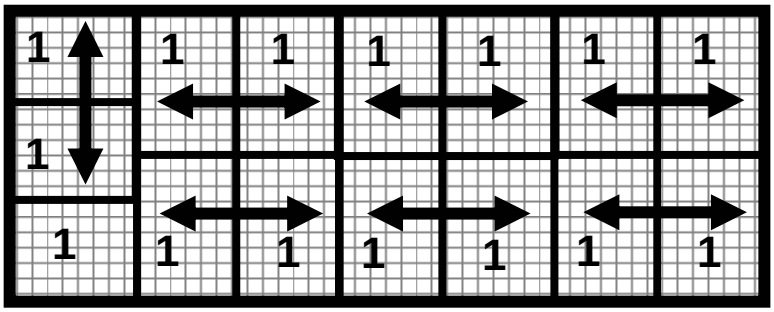


Figura 2.18: Ultima etapa do tratamento nRN

Ao contrário do tratamento 1RN, o tratamento nRN faz com que cada processador lógico presente no conjunto tenha *elementos comuns* e processos de troca de dados com vários outros processadores presentes no conjunto, e não apenas o imediatamente anterior e posterior. Contudo, a quantidade de *elementos comuns* e de dados comunicados entre os processadores é menor que no tratamento 1RN. Desta forma, tende a ser vantajoso nas seguintes configurações:

a) configurações nas quais a comunicação entre processadores depende mais da taxa de transferência (*throughput*) que da latência, uma vez que um grande número de processos de comunicação é realizado. Problemas grandes, nos quais cada processador trabalha sobre uma grande quantidade de dados entre cada comunicação, e a quantidade de dados a ser comunicada entre os processadores é grande, em máquinas paralelas de memória distribuída, se enquadram nesta configuração;

b) configurações nas quais poucos processadores lógicos compartilham a interface de rede. Como não há garantia de que a maior parte da comunicação entre os processadores / núcleos de cada computador permanece interna ao próprio computador, a interface de rede é tanto mais solicitada simultaneamente quanto maior for o número de processadores / núcleos que ela deve atender;

c) configurações com um grande número de processadores lógicos presentes.

Os dois tratamentos desenvolvidos, 1RN e nRN, tem custo computacional baixo comparado com os processos de solução iterativos ou explícitos propriamente ditos, e o tempo de

processamento necessário para a execução de qualquer um deles é, em geral, amplamente compensado pelos ganhos em tempo de processamento obtidos em configurações paralelas.

Um exemplo de como os nós e elementos de uma malha real ficam distribuídos entre os diversos grupos está mostrado na figura 2.19, onde a malha utilizada no exemplo 5.2.3. para simulação do escoamento transônico em torno de uma esfera foi dividida em 16 grupos. Nenhum tratamento foi aplicado, de modo que a distribuição é feita conforme a ordenação dos nós resultante do gerador de malha. Para cada grupo de nós foi atribuída uma cor, conforme as escalas que aparecem na figura. A faixa de cada grupo corresponde a (g – 0,25 a g + 0,25), onde g é o número do grupo, de modo que o valor de grupo dos nós está sempre no centro de cada faixa. Todas as faixas correspondentes a (g + 0,25 a g + 0,75) são cinza, para marcar os *elementos comuns*. Se todos os nós de um elemento pertencem a um mesmo grupo, o elemento como um todo estará colorido com a mesma cor do grupo do nó. Caso um elemento possua nós de grupos diferentes, aparecerão faixas cinza em sua parte central, tantas mais quanto mais distantes forem os números de grupo dos nós desse elemento. Dessa forma, o esquema de cores utilizado se aproxima bastante daquele utilizado nas figuras 2.1 a 2.8.

Pela figura 2.19, percebe-se que o gerador de malha utilizado começou a geração dos nós pela superfície externa, onde há um número reduzido de grupos, indicando, aparentemente, uma divisão eficiente pelo reduzido número de *elementos comuns* (em cinza), relacionados com a redundância de esforço computacional e com a quantidade de dados que devem ser trocados entre os processadores lógicos (as variáveis nodais dos nós ligados a esses elementos). Contudo, a vista da superfície do contorno sólido da malha (em detalhe) revela a existência de um número bastante grande de *elementos comuns* nessa região, e um corte interno da malha (os dois últimos detalhes) revela uma proximidade muito grande de diversos grupos diferentes numa mesma região, com um número bastante grande de *elementos comuns*. É importante perceber que no corte interno, há vários elementos que unem nós de grupos muito diferentes, aparecendo com uma série de faixas cinza intercaladas com as faixas das cores dos grupos intermediários aos grupos dos nós, causando a ilusão de que o elemento pertence a um grupo (é colorido) e não um *elemento comum* (em cinza). Uma observação atenta indica que boa parte dos elementos são comuns.

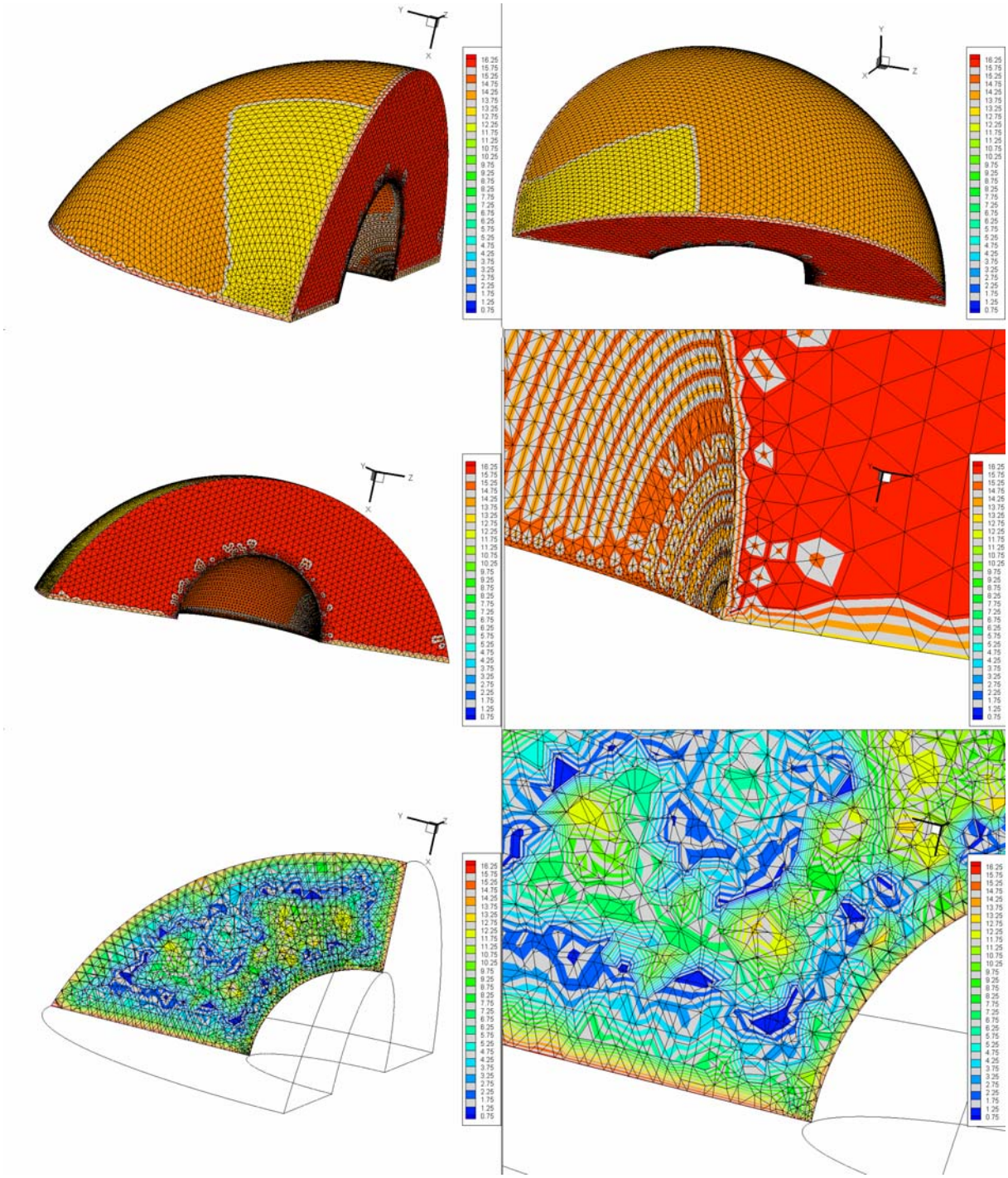


Figura 2.19: Distribuição de nós e elementos em grupos após divisão de tarefas por nós com a malha sem nenhum tratamento. Malha utilizada no exemplo 5.2.3 para simulação de escoamento transônico em torno de uma esfera.

A divisão de tarefas foi efetuada sobre a mesma malha com o tratamento 1RN, e o resultado pode ser visto na figura 2.20. A distribuição dos grupos de nós e elementos é uma versão tridimensional daquela indicada nas figuras 1.2 a 1.4, 2.3 e 2.5, com cada grupo fazendo fronteira somente com o grupo imediatamente anterior e imediatamente seguinte. A

característica pode ser vista tanto na superfície externa da malha quanto nos três cortes apresentados. Em relação à divisão de tarefas da malha sem tratamento da figura 2.19, percebe-se uma quantidade muito menor de *elementos comuns*, indicando menor redundância de esforço computacional e menor tráfego de dados entre processadores.

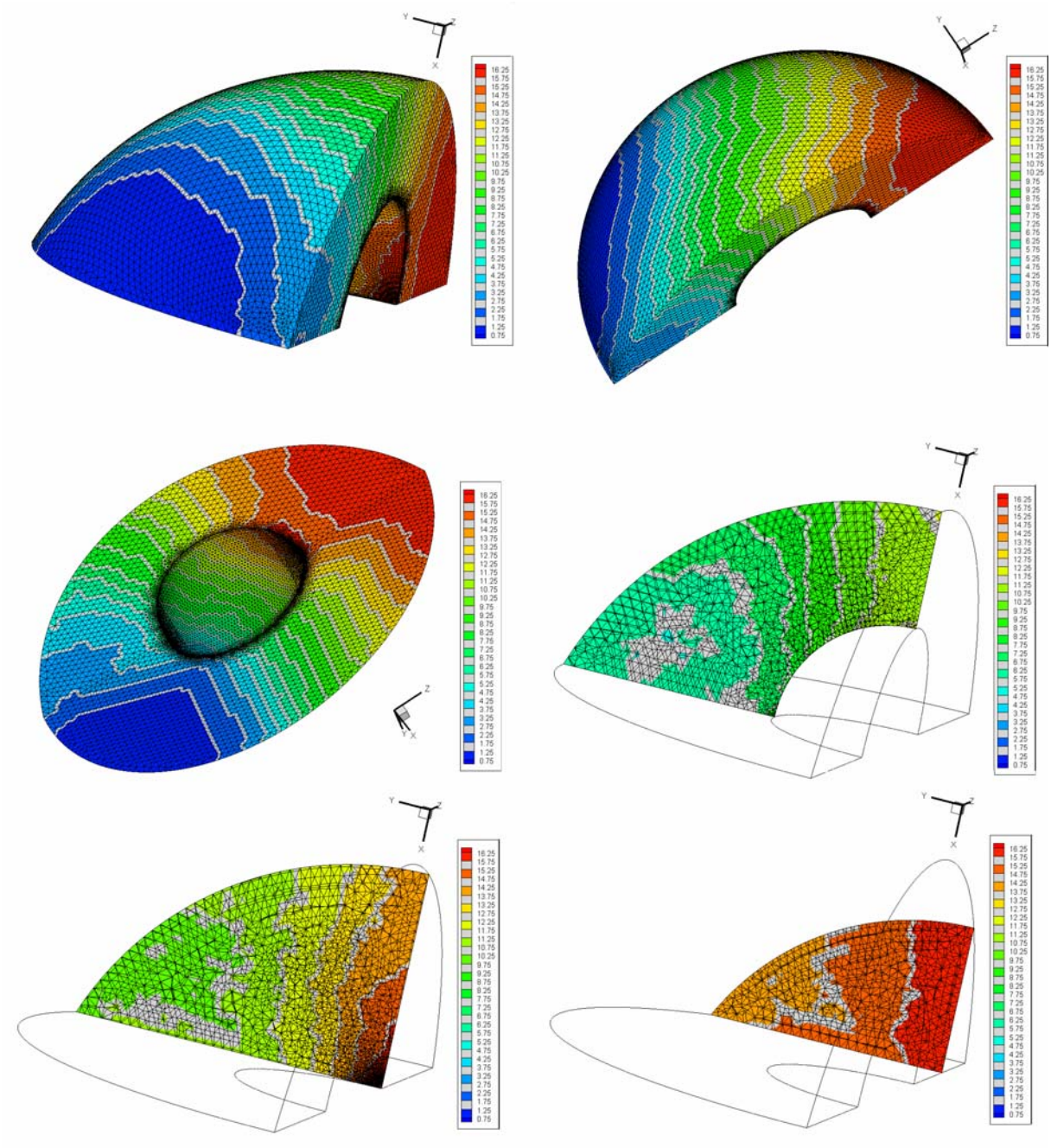


Figura 2.20: Distribuição de nós e elementos em grupos após divisão de tarefas por nós com a malha com tratamento 1RN. Malha utilizada no exemplo 5.2.3 para simulação de escoamento transônico em torno de uma esfera.

O resultado de uma divisão de tarefas sobre essa malha com o tratamento nRN pode ser visto na figura 2.21, que corresponde a uma versão tridimensional das figuras 1.5 e 2.8. As características são as mesmas: redução no número de *elementos comuns*, os quais conectam nós pertencentes a quaisquer grupos, e não somente grupos de números consecutivos. Essa características podem ser vistas tanto na superfície externa quanto nos cortes.

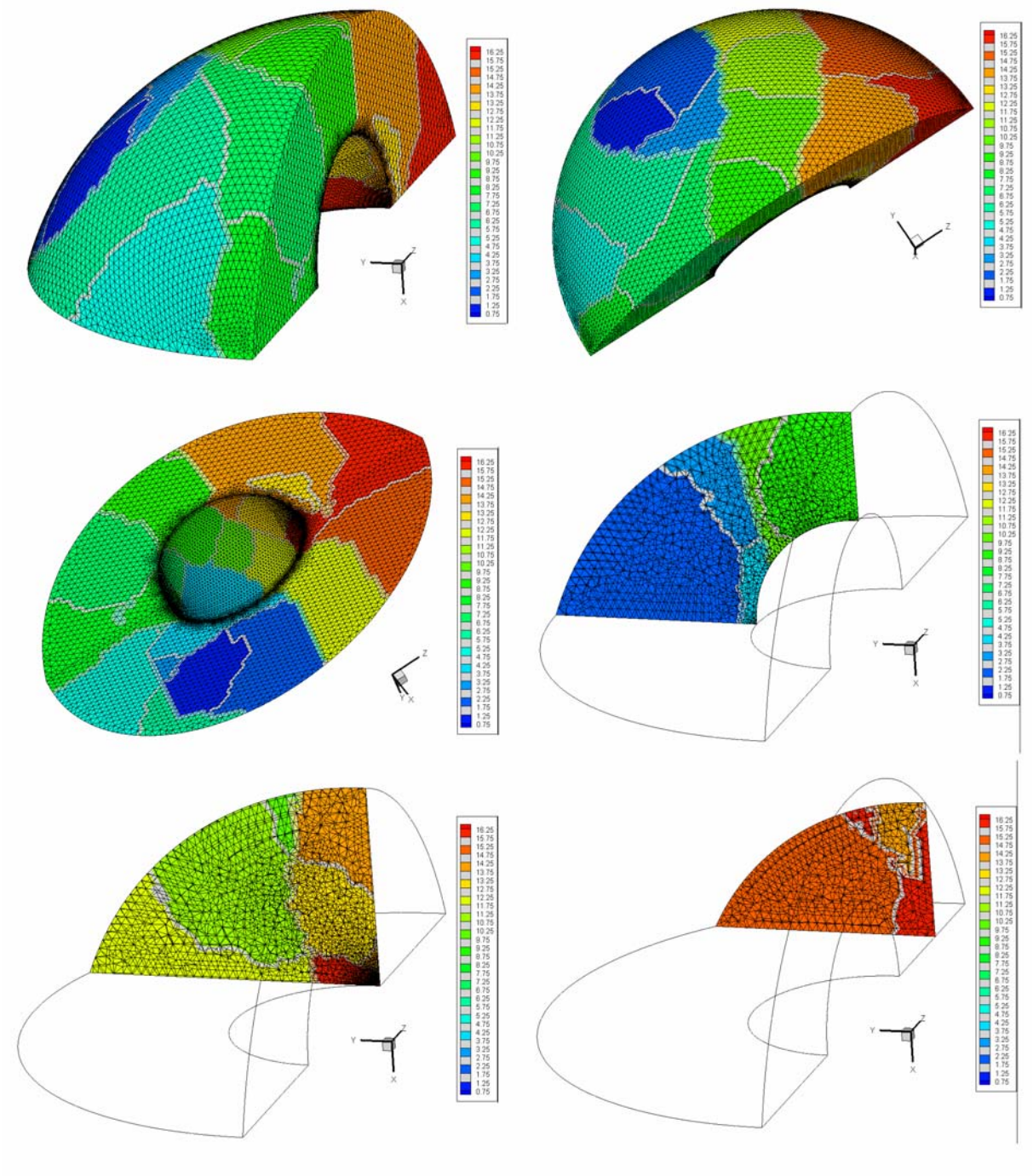


Figura 2.21: Distribuição de nós e elementos em grupos após divisão de tarefas por nós com a malha com tratamento nRN. Malha utilizada no exemplo 5.2.3 para simulação de escoamento transônico em torno de uma esfera.

Os grupos resultantes da divisão de tarefas aplicada sobre uma malha de elementos hexaédricos lineares utilizada no exemplo 5.2.2 para simular o escoamento supersônico em torno de um veículo espacial estão mostradas na figura 2.22. Os grupos para a malha original (sem nenhum tratamento) e com o tratamento 1RN são visualmente idênticos, indicando que a ordem utilizada na geração da malha estruturada resultou em uma malha com a banda minimizada. Próxima à parte frontal do veículo os grupos se distribuem como "conchas" esféricas sobrepostas, e, a partir do corpo cilíndrico, como discos circulares vazados "empilhados". Ambas distribuições são versões tridimensionais da distribuição característica do tratamento 1RN esquematizado nas figuras 1.2. a 1.4, 2.3 e 2.5.

Já para o tratamento nRN, os grupos se apresentam tendendo mais a volumes sem nenhuma dimensão preponderante. Na parte frontal da malha, os grupos não aparecem mais como "conchas" concêntricas, mas dividindo a região esférica em 4 setores segundo planos radiais, aproximadamente. A partir do corpo cilíndrico, as regiões são semelhantes às obtidas com os outros tratamentos, mas com o dobro da espessura e divididas ao longo de um plano longitudinal, aproximadamente. A distribuição de grupos obtida se assemelha a uma versão tridimensional das figuras 1.5 e 2.8.

Para quantificar o impacto teórico dos tratamentos 1RN e nRN sobre o tráfego de dados entre processadores lógicos e sobre a redundância de trabalho computacional, uma malha de aproximadamente 1,5 milhões de elementos tetraédricos lineares e 270 mil nós (1,36 milhões de equações) utilizada para análise do escoamento tridimensional supersônico em torno de um modelo simplificado de avião (geometria *canard* / corpo / asa / estabilizador vertical), exemplo 5.2.4, teve seus nós divididos em 16 grupos, correspondendo a um *cluster* com 16 processadores lógicos não homogêneo (processadores com diferentes capacidades de processamento).

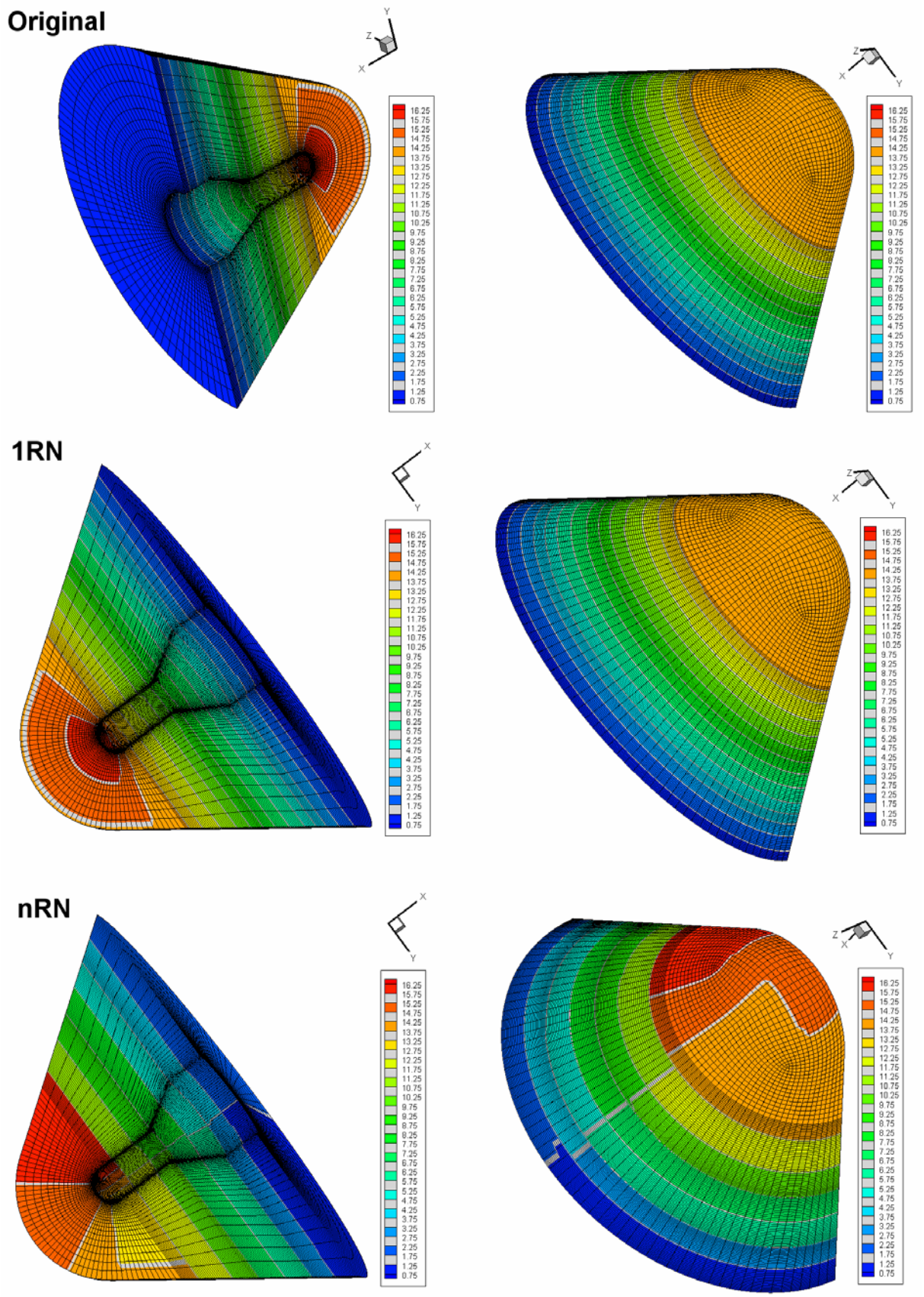


Figura 2.22: Distribuição de nós e elementos em grupos após divisão de tarefas por nós com a malha original (sem tratamento), com o tratamento 1RN e nRN. Malha utilizada no exemplo 5.2.2 para simulação de escoamento supersônico em torno de um veículo espacial.

Na tabela 2.1 estão apresentados os dados de tráfego na rede e redundância computacional para a malha sendo dividida sem nenhum tratamento, seguindo a ordenação original dos nós decorrente do processo de geração de malha. Nas colunas estão numerados os processadores lógicos, e nas linhas, para cada processador com o qual há troca de dados, a quantidade de dados recebida desse processador, expressa em milhares de nós (cada nó corresponde a 5 graus de liberdade ou 5 variáveis de dupla precisão, cada dado ocupando 8 bytes). Os blocos em amarelo indicam que não há comunicação de um processador com ele mesmo. Na linha indicada por "Nós" está a quantidade total de dados recebidos de outros processadores (em milhares de nós = x 5000 dados = x 40000 bytes). Na linha "Elementos" está o número total de elementos que é processado por cada processador lógico, incluindo tanto os exclusivos daquele processador quanto os *elementos comuns*, em milhares de elementos.

Tabela 2.1: Comunicação de dados entre processadores e redundância computacional para malha sem tratamento.

Processadores	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1		2,8	1,3	3,4	3,8	3,6	4,1	5,5	5,3	5,3	4,4	5,0	5,6	5,8	4,3	4,3
2	3,5		3,2	4,2	5,0	5,8	6,0	5,4	5,0	5,4	6,7	7,3	6,8	6,5	5,2	6,7
3	1,7	3,0		4,8	3,8	2,9	3,1	2,6	2,3	2,4	1,6	1,4	1,7	1,8	4,4	5,0
4	9,8	11,5	4,9		7,3	6,1	5,5	5,0	3,8	3,4	4,4	3,2	2,3	1,6	1,3	1,5
5	10,1	11,9	3,8	7,5		6,7	5,9	5,4	4,3	4,1	4,8	3,7	2,8	2,0	1,2	1,7
6	8,7	13,0	3,5	6,1	6,8		6,2	5,9	4,8	5,0	5,0	3,9	3,1	2,7	1,9	2,1
7	9,3	12,4	3,5	5,5	6,1	6,3		7,0	5,7	5,2	4,7	4,0	3,5	2,8	2,2	2,2
8	12,2	10,3	3,0	5,0	5,3	5,8	7,1		6,5	5,4	5,2	4,7	4,0	3,5	2,6	2,4
9	11,4	9,3	3,0	3,6	3,9	4,6	5,6	6,4		6,2	4,8	4,8	4,6	4,5	3,3	2,3
10	11,3	9,6	2,8	3,1	3,8	4,6	5,0	5,3	6,3		5,2	5,3	5,2	5,3	4,0	2,9
11	9,7	12,9	1,7	4,3	4,6	5,0	4,8	5,4	5,2	5,4		9,8	7,4	5,5	2,7	3,3
12	10,8	12,2	1,6	3,0	3,5	3,7	3,8	4,6	4,7	5,4	9,4		9,8	7,9	3,9	4,0
13	11,4	11,3	1,9	2,2	2,7	2,9	3,3	3,8	4,4	5,0	7,4	9,7		10,3	5,7	4,8
14	11,8	10,7	2,0	1,5	1,9	2,6	2,6	3,5	4,3	5,3	5,6	7,9	10,3		9,0	6,1
15	7,2	7,5	7,3	1,2	1,1	1,8	2,0	2,4	3,0	3,7	2,6	3,7	5,2	7,5		7,4
16	7,9	11,4	5,5	1,6	1,9	2,4	2,7	3,0	2,9	3,5	3,6	4,5	5,6	7,0	8,5	
Nós	**137**	**147**	**48**	**54**	**58**	**61**	**64**	**66**	**63**	**66**	**71**	**74**	**72**	**69**	**56**	**52**
Elementos	**308**	**399**	**208**	**209**	**216**	**225**	**229**	**235**	**223**	**228**	**253**	**260**	**258**	**254**	**215**	**211**

Total de nós comunicados entre processadores:	**1156**
Total de elementos processados:	**3932**
Redundância em elementos:	**162%**
Eficiência no processamento de elementos:	**38%**
Índice de troca de dados:	**425%**

No fundo da tabela está indicada a quantidade total de dados transmitidos entre os processadores (em milhares de nós) e o total de elementos processados por todos os processadores lógicos (em milhares de elementos). A redundância computacional (expressa percentualmente) é calculada por:

$$Redundância = \frac{Total\ de\ elementos\ \ processados}{Total\ de\ elementos\ da\ malha} - 1 \tag{2.2}$$

indicando, na divisão de tarefas, quanto do esforço computacional relacionado aos elemento é realizado a mais que o correspondente ao da malha sendo inteiramente processada por um único processador lógico. A eficiência no processamento de elementos é calculada por:

$$Eficiência = \frac{Total\ de\ elementos\ da\ malha}{Total\ de\ elementos\ \ processados} \tag{2.3}$$

e indica quanto do esforço computacional relacionado aos elementos não é redundante. Finalmente, um indicador do volume relativo de dados comunicado entre processadores a cada iteração é definido como Índice de Troca de Dados (ITD) e é calculado por:

$$Índice\ de\ troca\ de\ dados = \frac{Total\ de\ nós\ comunicados\ entre\ \ processadores}{Total\ de\ nós\ da\ malha} \tag{2.4}$$

Da análise da tabela 2.1 verifica-se que a ordenação original da malha, resultante do processo de geração de malha utilizado e de processos de refinamento de malha que porventura tenham sido empregados, leva a uma matriz esparsa, com comunicação de dados entre todos os possíveis pares de processadores. A cada iteração do processo de solução, 240 trocas de dados entre processadores lógicos são realizadas, e o total de dados comunicado entre eles é de 1,156 milhões de nós ou aproximadamente 44,1 MB (Megabytes). O número total de elementos processados é de 3,9 milhões, ou 162% a mais que o correspondente à malha inteira sendo processada por um único processador lógico (1,5 milhões), resultando que, na divisão de tarefas entre 16 processadores, somente 38% do esforço computacional relacionado a elementos não é redundante. O ITD obtido foi de 425%, indicando que mais que 4 vezes o total de variáveis nodais da malha é comunicado entre os processadores a cada iteração. Os resultados obtidos para a malha sem tratamento serão usados como valores de referência para os tratamentos 1RN e nRN. Os grupos obtidos pela divisão de tarefas sobre a malha sem tratamento podem ser vistos na figura 2.23, que é uma representação gráfica da tabela 2.1.

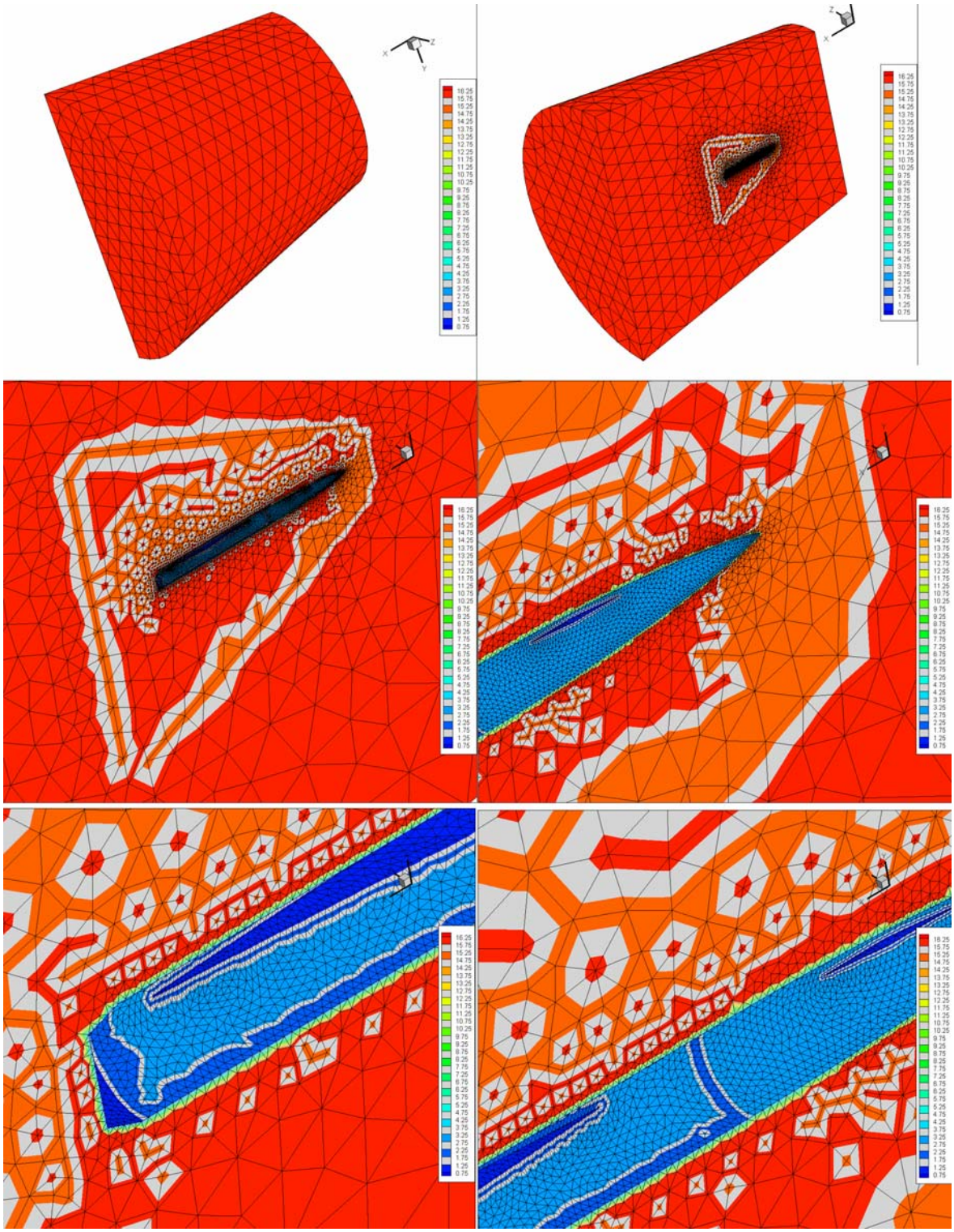


Figura 2.23: Distribuição de nós e elementos em grupos após divisão de tarefas para a malha sem nenhum tratamento. Malha utilizada no exemplo 5.2.4.

A ordem dos nós é resultado do gerador de malha utilizado, com dependência da forma como as diversas regiões do domínio foram divididas em superfícies matematicamente conhecidas (ver faixa azul escura no último detalhe). Apesar de toda a parte externa da malha pertencer a

um mesmo grupo, na parte interna os grupos de elementos estão bastante misturados, especialmente em torno do contorno sólido do avião, onde a malha é mais refinada, como pode ser visto nos cortes mostrados na figura 2.24.

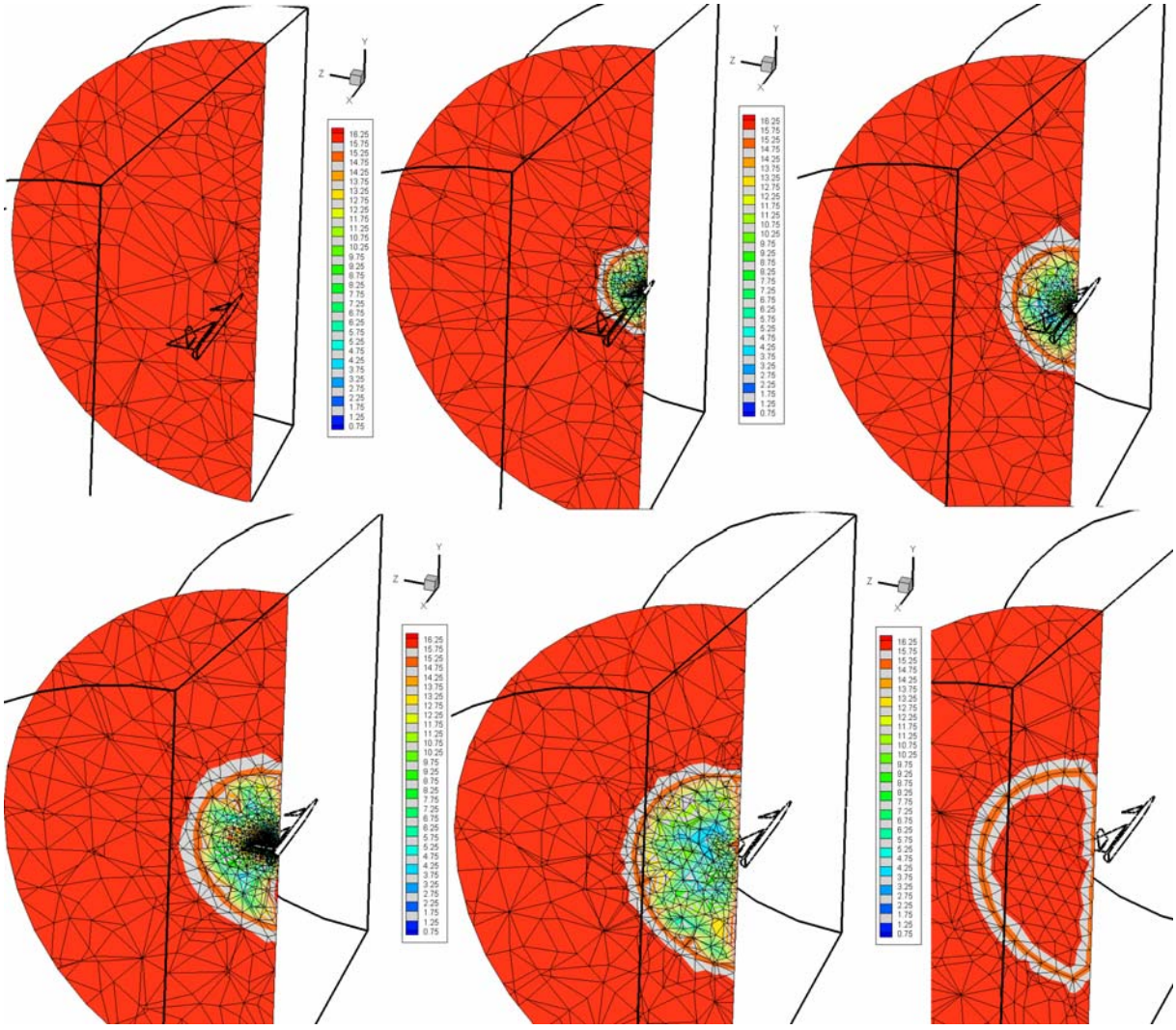


Figura 2.24: Distribuição de nós e elementos em grupos após divisão de tarefas com a malha sem nenhum tratamento - cortes. Malha utilizada no exemplo 5.2.4.

Na tabela 2.2 são apresentados os mesmos dados para a malha submetida ao tratamento 1RN, ou seja, tendo seus nós reordenados uma única vez visando a minimização de banda antes da divisão de tarefas ser efetuada.

O tratamento 1RN leva a uma matriz de comunicação de dados concentrada em torno da diagonal principal (tipo banda), com cada processador lógico trocando dados somente com o processador imediatamente anterior e o imediatamente seguinte. A cada iteração do processo de solução, o número de troca de dados entre processadores é de 30, ou 12,5% do original, e o

volume total de dados trocados entre processadores de aproximadamente 0,243 milhões de nós (ou 9,27 MB), 21% do original. O número total de elementos processados por todos os processadores lógicos é de 2,19 milhões (56% do original), reduzindo a redundância para 46%, aumentando a eficiência para 69% e reduzindo o ITD para 90%. Os dados indicam que, apesar da simplicidade do tratamento empregado, há um grande ganho de eficiência na paralelização.

Tabela 2.2: Comunicação de dados entre processadores e redundância computacional para malha com tratamento 1RN.

Processadores	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1		2,8														
2	3,5		4,3													
3		5,1		5,5												
4			6,6		6,5											
5				7,8		7,8										
6					9,6		9,1									
7						10,7		9,6								
8							10,9		9,0							
9								9,7		8,9						
10									9,9		9,2					
11										10,7		9,4				
12											10,6		9,8			
13												10,7		9,4		
14													10,3		8,1	
15														8,6		6,1
16															5,9	
Nós	**4**	**5**	**11**	**13**	**16**	**18**	**20**	**19**	**19**	**20**	**20**	**20**	**20**	**18**	**14**	**6**
Elementos	**110**	**121**	**129**	**137**	**145**	**152**	**156**	**154**	**145**	**147**	**147**	**147**	**144**	**135**	**123**	**94**

Total de nós comunicados entre processadores:	**243**
Total de elementos processados:	**2186**
Redundância em elementos:	**46%**
Eficiência no processamento de elementos:	**69%**
Índice de troca de dados:	**90%**

A representação gráfica da divisão de tarefas obtida com o tratamento 1RN está mostrada na figura 2.25. Apesar da complexidade da malha, a estrutura básica da distribuição dos nós e elementos em grupos que somente fazem fronteira com o grupo imediatamente anterior e posterior se mantém. Essa característica, representada graficamente por grupos se dispondo em faixas ou camadas, é bem visível na parte externa da malha, e também sobre o contorno sólido. Internamente, a distribuição dos grupos segue o mesmo padrão, mostrando-se bem diferente da divisão em grupos decorrente da malha sem tratamento, como pode ser visto na figura 2.26.

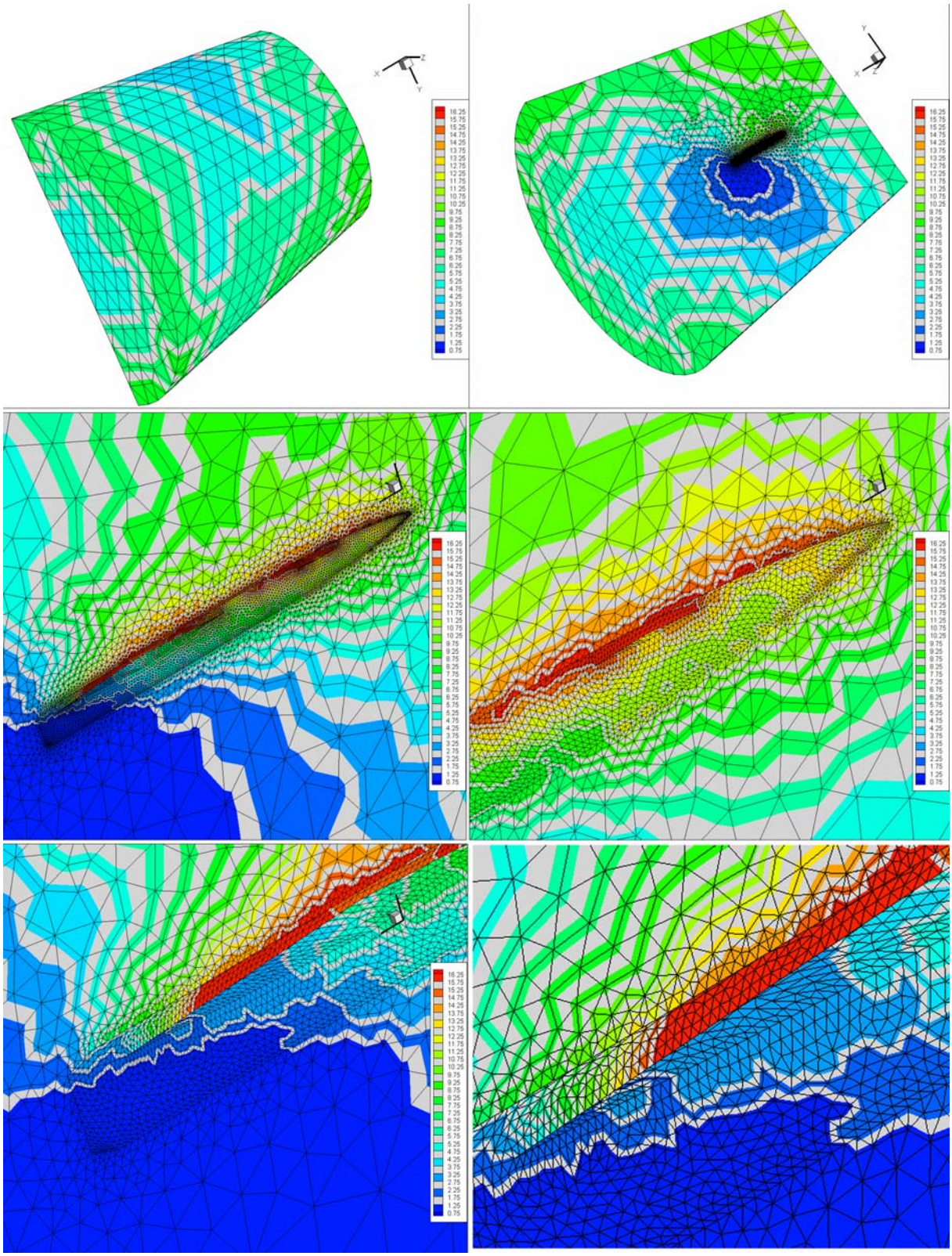


Figura 2.25: Distribuição de nós e elementos em grupos após divisão de tarefas para a malha com o tratamento 1RN. Malha utilizada no exemplo 5.2.4.

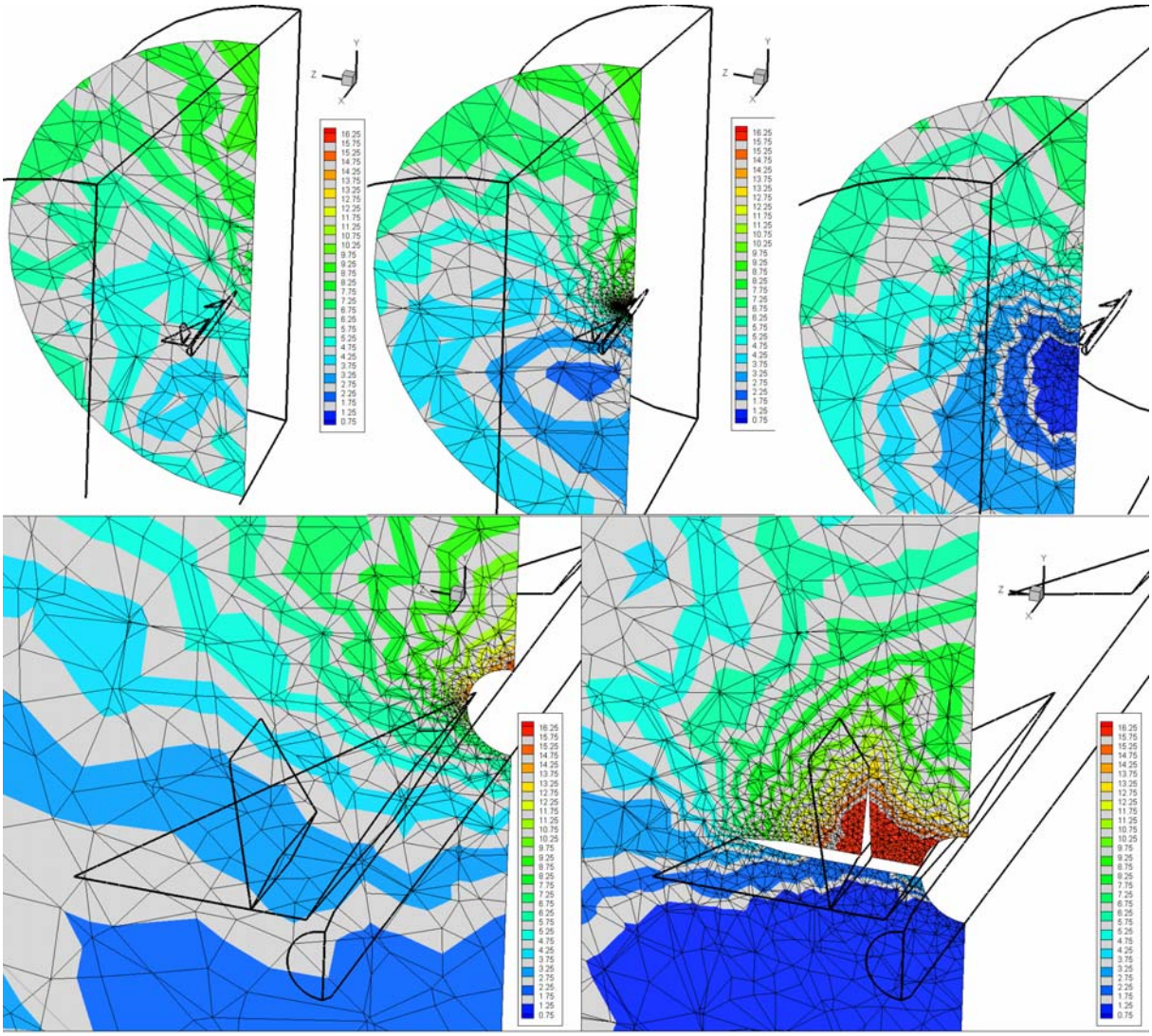


Figura 2.26: Distribuição de nós e elementos em grupos após divisão de tarefas para a malha com o tratamento 1RN - cortes. Malha utilizada no exemplo 5.2.4.

Na tabela 2.3 são apresentados os dados referentes à malha submetida ao tratamento nRN. Este tratamento diminui ainda mais a comunicação de dados entre os processadores, obtendo um valor total de 0,163 milhões de nós (6,22 MB) por iteração do processo de solução, 14,1% do valor original. A quantidade total de elementos processados também foi reduzida, atingindo 1,96 milhões, 50,3% do original.

Quando comparado ao tratamento 1RN, o tratamento nRN resulta em uma menor quantidade de dados sendo comunicados entre processadores, uma menor redundância no processamento de elementos e, conseqüentemente, uma maior eficiência. A única desvantagem é que a matriz de comunicação entre processadores se torna mais esparsa, com cada processador podendo ter que trocar dados com vários outros. O total de processos de comunicação entre processadores

é de 69 por iteração, contra 30 do tratamento 1RN, fazendo com que a latência de comunicação tenha maior impacto sofre a eficiência do processo e tornando mais complexa a ordenação dos diversos processos de comunicação entre os processadores para que não haja filas de espera ou saturação da rede. Dependendo da configuração da máquina paralela e do tipo e tamanho do problema a ser analisado, a vantagem obtida pela redução do volume de dados a serem comunicados e pela redução da redundância pode ser contrabalançada pela latência decorrente da necessidade de comunicação de cada núcleo com diversos outros, não resultando em uma maior velocidade de processamento do conjunto.

Tabela 2.3: Comunicação de dados entre processadores e redundância computacional para malha com tratamento nRN.

Processadores	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1		1,5	0,1	1,1												
2	1,8		1,6	0,7					1,9							
3	0,2	2,0		1,4	0,7	1,8	0,2		1,1							
4	1,2	0,7	1,6		0,4	0,7	1,4	0,1	0,1							
5			0,8	0,6		1,4	0,8	0,3	1,1							
6			2,0	0,9	1,8		1,4	1,2	1,3							
7			0,2	1,4	0,8	1,7		0,7	0,8							
8				0,1	0,2	1,5	0,8		2,8							
9		1,8	1,0	0,1	1,2	1,6	1,0	3,5		8,9						
10									9,9		4,9	4,4				
11										5,4		0,4	4,5			
12										5,3	0,5		5,3			
13											4,8	6,3		9,4		
14													10,3		8,1	
15														8,6		6,1
16															5,9	
Nós	**3**	**5**	**7**	**5**	**5**	**9**	**6**	**6**	**19**	**20**	**10**	**11**	**20**	**18**	**14**	**6**
Elementos	**107**	**117**	**123**	**114**	**118**	**125**	**112**	**114**	**145**	**147**	**119**	**122**	**144**	**135**	**123**	**94**

Total de nós comunicados entre processadores:	**163**
Total de elementos processados:	**1958**
Redundância em elementos:	**30%**
Eficiência no processamento de elementos:	**77%**
Índice de troca de dados:	**60%**

Os grupos de nós e elementos decorrentes da divisão de tarefas com o tratamento nRN podem ser vistos na figura 2.27. Apesar de próximo do contorno sólido do avião a distribuição dos grupos ser semelhante à obtida com o tratamento 1RN (em camadas ou faixas), um pouco mais afastado a distribuição segue o padrão esquematizado nas figuras 1.5 e 2.8 e obtida também para o escoamento em torno da esfera (figura 2.20) com as regiões do domínio correspondentes a cada grupo tendendo para um volume sem nenhuma dimensão preponderante. Na parte interna da malha, figura 2.28, a distribuição é semelhante, tendendo a faixas perto do avião e a volumes sem uma dimensão preponderante na região um pouco mais afastada.

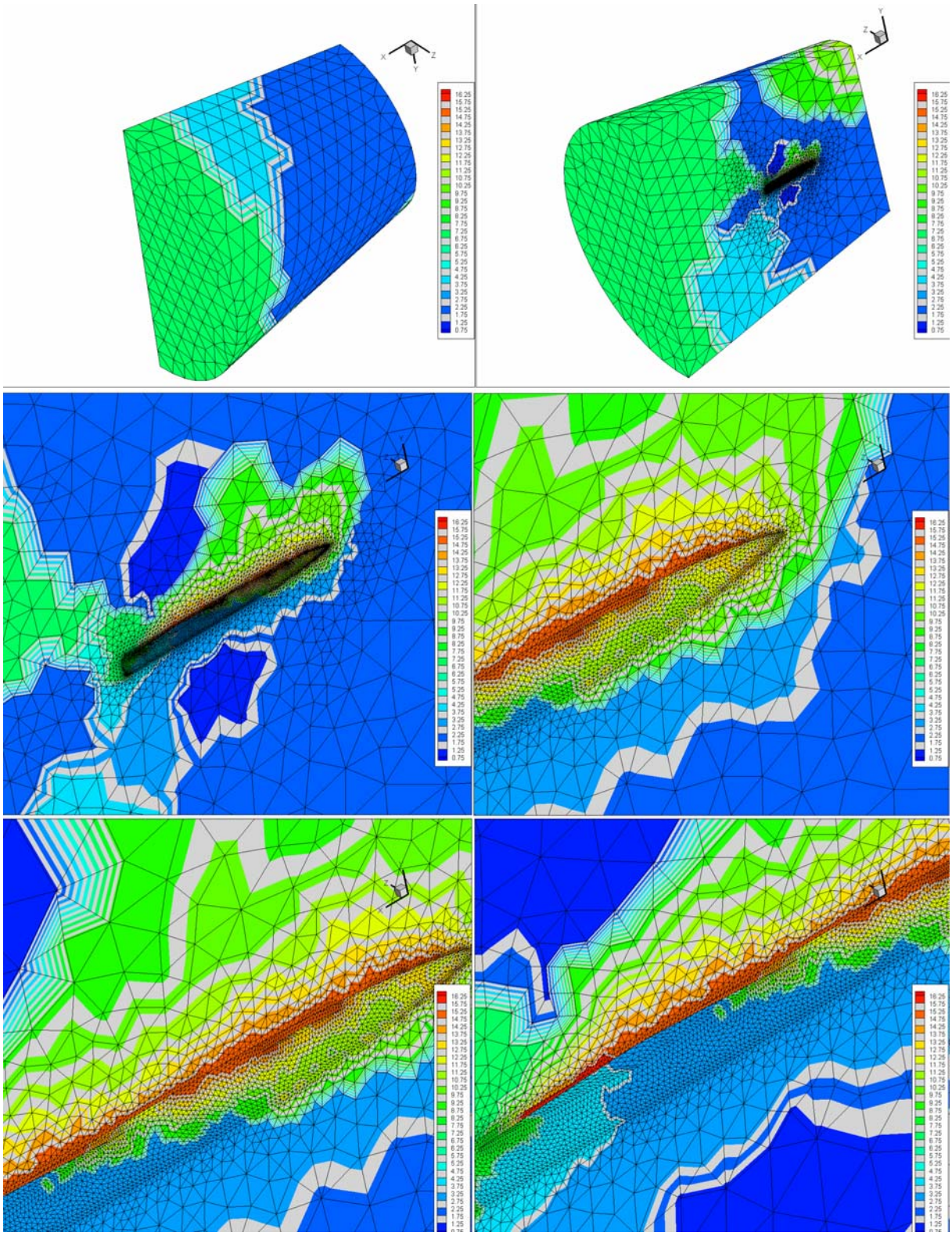


Figura 2.27: Distribuição de nós e elementos em grupos após divisão de tarefas para a malha com o tratamento nRN. Malha utilizada no exemplo 5.2.4.

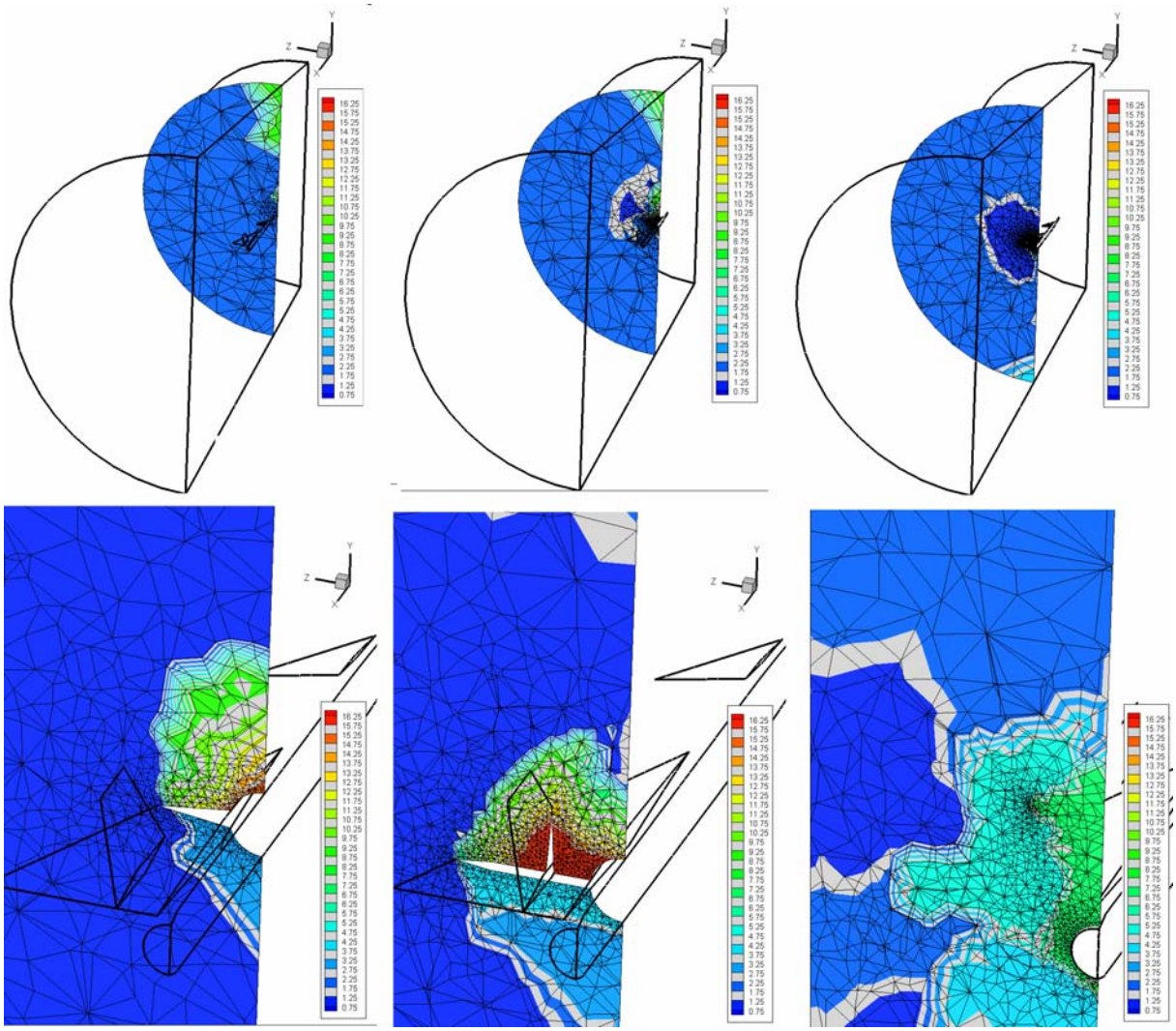


Figura 2.28: Distribuição de nós e elementos em grupos após divisão de tarefas para a malha com o tratamento nRN - cortes. Malha utilizada no exemplo 5.2.4.

2.5 BALANCEAMENTO DAS CARGAS DE TRABALHO

Os tratamentos apresentados nas seções 2.3 e 2.4 minimizam o tráfego de rede e a redundância de esforço computacional, mas por si só não são suficientes para garantir uma distribuição de tarefas equilibrada na qual todos os processadores lógicos trabalhando sobre seu conjunto de nós tenham o mesmo tempo de processamento.

Obter o balanceamento das cargas de trabalho dos diversos processadores unicamente através da atribuição a cada um deles de um conjunto de nós cujo tamanho é proporcional à velocidade relativa de cada processador traz implícita a consideração de que o esforço computacional associado a todos os nós é o mesmo, que cada nó está conectado a um mesmo

número de elementos e que o esforço computacional associado a todos os elementos é o mesmo. Condições de contorno, cargas e não linearidades físicas fazem com que os diversos nós e elementos demandem esforços computacionais distintos, e a necessidade de adaptar a malha à geometria do problema físico a ser simulado faz com que raramente o número de elementos conectados aos diversos nós seja constante. Essa característica é particularmente evidente em problemas tridimensionais de geometria complexa discretizados por elementos tetraédricos nos quais a malha é refinada localmente, onde o número de elementos conectados a um mesmo nó pode variar de um a mais de uma centena.

Desse modo, é bastante difícil realizar *a priori* uma divisão de tarefas equilibrada sem que a mesma precise ser balanceada de alguma forma pelo desenvolvimento do próprio processo de simulação computacional. Considerando que um bom número dos algoritmos de simulação empregados para problemas grandes são iterativos, empregou-se uma metodologia de balanceamento da carga de trabalho de cada processador lógico retro-alimentada pelo tempo gasto por cada processador na simulação computacional do próprio problema a ser processado, tempo este estimado ao longo de um determinado número de passos iterativos da solução (KWOK et al. 1999). Este método pode ser classificado como um método estático (DORNELES, 2003), pois o balanço de cargas é feito no início da solução e permanece inalterado ao longo do processamento. Esse processo somente é adequado quando o esforço computacional por nó é constante ao longo das iterações. Não-linearidades físicas (como plasticidade, dano contínuo, propagação de trinca), processos adaptativos no espaço (refinamento e desrefinamento da malha – ver Chang et al., 1993, Popiolek e Awruch, 2006) e no tempo (uso da técnica de sub-ciclos na integração temporal – ver Belytchko e Lu, 1993, van der Vem et al. 1997, Maurits et al. 1998, Teixeira e Awruch, 2001) ou uma combinação deles (Hooker et al., 1992) alteram a distribuição espacial do esforço computacional ao longo do processo de solução, exigindo modelos de balanceamento dinâmico das cargas de trabalho.

A metodologia empregada segue as seguintes etapas:

a) Um índice de desempenho relativo é obtido por:

$$IDR_i = \frac{ID_i}{\sum_{j=1}^{n} ID_j} \tag{2.5}$$

onde *IDi* é o índice de desempenho absoluto do processador lógico *i* e *IDRi* o índice de desempenho relativo desse mesmo processador. Um estimador aproximado é utilizado como índice de desempenho absoluto inicial, como por exemplo a freqüência (*CPU clock*) de cada processador.

b) A quantidade de nós da malha N_i alocada para cada processador *i* é dada por:

$$N_i = IDR_i \, N \tag{2.6}$$

onde *N* é o número total de nós da malha.

c) A divisão de tarefas é feita através de um dos tratamentos descritos anteriormente. Os nós atribuídos a cada grupo ou processador são tomados seqüencialmente conforme a ordenação dos nós. Assim, os primeiros N_i nós são atribuídos ao processador *i*, os N_{i+1} ao processador seguinte *i+1* e assim por diante.:

d) Um determinado número de passos dos processo iterativo é executado, onde o tempo de processamento puro (tempo de CPU) t_i ,calculado como o tempo total de execução menos o tempo gasto em comunicação de dados através da biblioteca de comunicação empregada, é avaliado para cada processador. Dentro do tempo gasto em comunicação está o tempo gasto em estado de espera por um processador até que o outro processador com o qual ele irá trocar dados esteja pronto para a comunicação, de modo que se a carga de trabalho entre processadores estiver desbalanceada, isso irá afetar o tempo de comunicação de um dos processadores, mas não seu tempo de processamento puro.

e) O maior tempo total (processamento puro + tempo de comunicação) dentre todos os processadores t_{max} é armazenado e a média do tempo de processamento puro de todos os processadores, t_m, é calculada.

f) Se a condição de balanceamento dada por:

$$(1-tol).t_m \leq t_i \leq (1+tol).t_m \tag{2.7}$$

for satisfeita, considera-se que a distribuição de tarefas está balanceada e o código é executado com essa distribuição. O parâmetro *tol* é a tolerância especificada para que a condição de balanceamento seja satisfeita.

g) Caso o balanceamento não seja alcançado, uma nova estimativa do índice de desempenho relativo IDR_i^* para cada processador é obtida pelo produto do índice de desempenho original IDR_i pela razão entre a média dos tempos de processamento t_m e o tempo de processamento puro respectivo t_i.

$$IDR_i^* = \frac{t_m}{t_i} IDR_i \tag{2.8}$$

h) Os novos índices de desempenho relativos são normalizados por

$$IDR_i = \frac{IDR_i^*}{\sum_{j=1}^{n} IDR_j^*} \tag{2.9}$$

e o processo é reiniciado a partir da etapa b).

Se em um determinado número de tentativas a condição de balanceamento não for alcançada, adota-se a distribuição de tarefas que levou ao menor valor de t_{max}.

O processo de balanceamento não necessariamente garante que o tempo associado à comunicação de dados ou à redundância no processamento associado aos elementos diminua. O que o processo procura garantir é que tempo total de processamento diminua por uma melhor distribuição da carga de trabalho, mesmo que isso acarrete uma maior carga de trabalho total entre todos os processadores. Isso pode ser percebido na tabela 3.4, onde o processo de balanceamento foi utilizado na malha do modelo simplificado de avião sem tratamento, com o tratamento 1RN e nRN. Nos 3 casos, se a condição de balanceamento não fosse atingida em 10 tentativas, a tentativa com menor t_{max} era utilizada para o processamento. Somente para o tratamento nRN o balanceamento não foi obtido dentro de 10 tentativas, como pode ser observado pela diferença entre os valores de tempo de processamento puro mínimo e máximo.

A metodologia proposta para balanceamento das cargas de trabalho atribuídas aos diversos processadores lógicos somente é válida se o esforço computacional relacionado aos nós e elementos permanecer constante ao longo do processo de solução. Não linearidades e alterações locais no passo de tempo utilizado na discretização temporal de problemas transientes acarretam alteração do esforço computacional associado aos nós e elementos

envolvidos, de modo que novos balanceamentos devem ser feitos ao longo do processo de solução para manter a eficiência da paralelização.

Tabela 2.4: Balanceamento da divisão de tarefas para do modelo simplificado de avião, sem tratamento e com os tratamentos 1RN e nRN.

		s/ tratamento		**1RN**		**nRN**	
Proces.	**ID**	**IDR inic**	**IDR final**	**IDR inic**	**IDR final**	**IDR inic**	**IDR final**
1	2500	6,52%	5,00%	6,52%	8,87%	6,52%	6,71%
2	2500	6,52%	3,48%	6,52%	7,92%	6,52%	7,23%
3	2500	6,52%	3,37%	6,52%	7,24%	6,52%	6,43%
4	2500	6,52%	6,05%	6,52%	6,66%	6,52%	7,61%
5	2500	6,52%	9,05%	6,52%	6,14%	6,52%	7,97%
6	2500	6,52%	8,16%	6,52%	5,95%	6,52%	7,36%
7	2500	6,52%	7,92%	6,52%	5,81%	6,52%	7,76%
8	2500	6,52%	7,64%	6,52%	5,85%	6,52%	7,56%
9	2300	5,98%	6,70%	5,98%	5,16%	5,98%	4,27%
10	2300	5,98%	6,53%	5,98%	5,05%	5,98%	4,18%
11	2300	5,98%	5,78%	5,98%	5,01%	5,98%	5,78%
12	2300	5,98%	5,50%	5,98%	5,03%	5,98%	5,52%
13	2300	5,98%	5,61%	5,98%	5,14%	5,98%	4,19%
14	2300	5,98%	5,57%	5,98%	5,45%	5,98%	4,51%
15	2300	5,98%	5,99%	5,98%	6,07%	5,98%	5,34%
16	2300	5,98%	7,67%	5,98%	8,65%	5,98%	7,58%
Tempo de CPU min. (s)		**34,37**	**50,12**	**16,94**	**24,36**	**14,58**	**11,04**
Tempo de CPU máx. (s)		**84,00**	**53,55**	**27,51**	**25,10**	**23,84**	**21,14**
Tempo proces. Total (s)		**287,77**	**207,34**	**37,10**	**36,51**	**36,84**	**33,38**

Os índices de desempenho relativos *IDR* iniciais (antes do balanceamento) foram calculados pela equação (2.5) e estão expressos na tabela 2.4 de forma percentual. Os índices de desempenho finais foram os obtidos após o balanceamento.

Em todos os casos, o balanceamento da carga de trabalho trouxe ganhos quanto ao tempo de processamento total em relação à divisão de tarefas original, baseada exclusivamente no índice de desempenho inicial (a freqüência dos processadores), mesmo quando o balanceamento deu origem a divisões de tarefas finais com maior quantidade de dados comunicados entre processadores e maior redundância no processamento de elementos, como foi o caso da malha sem tratamento (tabela 2.5) e com tratamento 1RN (tabela 2.6). Para o tratamento nRN, a divisão de tarefas final obtida tem uma menor índice de troca de dados e menor redundância de elementos (tabela 2.7)

Tabela 2.5: Comunicação de dados entre processadores e redundância computacional para malha sem tratamento após balanceamento da divisão de tarefas.

Processadores	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1		1,5	1,0	0,9	3,8	4,3	4,3	4,1	4,0	4,3	4,4	2,9	3,3	3,5	3,1	2,6
2	2,0		2,0	1,7	0,1	0,2	2,9	3,5	3,5	3,2	6,7	3,8	4,2	4,5	4,6	4,0
3	1,1	1,9		1,9	2,6	4,9	3,1	3,2	2,9	2,9	1,6	3,8	3,4	3,2	3,1	3,3
4	1,2	1,9	2,8		6,7	3,8	3,6	3,5	2,8	2,9	4,4	2,1	2,3	2,2	3,2	6,6
5	11,6	0,1	7,0	14,6		10,9	8,7	7,6	5,3	4,8	4,8	3,8	2,9	2,1	1,9	3,0
6	12,0	0,5	13,5	6,8	10,3		9,4	7,9	6,3	5,7	5,0	4,6	3,6	2,8	2,3	2,2
7	10,5	9,3	7,7	6,2	8,2	9,6		9,0	7,1	6,6	4,7	4,6	3,7	3,2	3,2	2,7
8	9,6	9,9	7,1	5,8	7,3	7,8	9,1		8,0	6,8	5,2	5,0	4,4	3,8	3,4	2,8
9	8,4	9,2	6,2	4,7	5,0	5,7	6,8	7,8		7,2	4,8	4,9	4,7	4,7	4,5	2,6
10	9,9	7,8	6,0	4,6	4,3	5,1	6,2	6,5	7,2		5,2	5,5	5,4	5,5	5,3	2,1
11	6,7	7,7	8,1	4,1	4,8	5,4	5,4	5,6	5,6	5,8		9,1	7,0	5,5	3,6	3,2
12	7,1	7,8	7,2	3,3	3,3	4,1	4,2	4,6	4,6	5,3	9,4		8,6	7,3	4,6	3,6
13	7,7	8,2	6,4	3,4	2,6	3,3	3,4	4,0	4,4	5,0	7,4	8,7		9,6	7,0	4,3
14	8,1	8,4	5,8	3,1	1,9	2,5	2,9	3,5	4,3	5,1	5,6	7,2	9,4		9,7	5,3
15	7,4	9,2	5,7	4,9	1,8	2,2	3,0	3,4	4,5	5,3	2,6	5,0	7,2	9,8		9,1
16	5,4	8,3	5,9	11,9	2,8	2,6	3,1	3,5	3,1	3,9	3,6	4,2	5,1	6,2	10,9	
Nós	**109**	**90**	**91**	**77**	**61**	**68**	**72**	**74**	**69**	**71**	**71**	**72**	**72**	**70**	**67**	**55**
Elementos	**219**	**218**	**215**	**250**	**274**	**268**	**270**	**268**	**245**	**246**	**253**	**241**	**245**	**242**	**246**	**244**

Total de nós comunicados entre processadores:	**1190**
Total de elementos processados:	**3946**
Redundância em elementos:	**163%**
Eficiência no processamento de elementos:	**38%**
Índice de troca de dados:	**438%**

Tabela 2.6: Comunicação de dados entre processadores e redundância computacional para malha com tratamento 1RN após balanceamento da divisão de tarefas.

Processadores	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1		3,3														
2	4,0		5,0													
3		6,1		6,2												
4			7,5		7,4											
5				9,1		8,9										
6					10,6		9,4									
7						10,5		9,4								
8							10,5		9,1							
9								10,1		8,5						
10									9,8		9,4					
11										10,9		9,4				
12											10,7		9,2			
13												10,5		9,3		
14													10,4		8,9	
15														9,5		7,4
16															7,5	
Nós	**4,0**	**6,1**	**12,5**	**15,3**	**18,0**	**19,4**	**19,8**	**19,5**	**18,9**	**19,4**	**20,1**	**19,9**	**19,6**	**18,8**	**16,4**	**7,4**
Elementos	**147**	**147**	**145**	**145**	**144**	**145**	**144**	**145**	**132**	**133**	**132**	**131**	**131**	**132**	**131**	**134**

Total de nós comunicados entre processadores:	**255**
Total de elementos processados:	**2220**
Redundância em elementos:	**48%**
Eficiência no processamento de elementos:	**68%**
Índice de troca de dados:	**94%**

Tabela 2.7: Comunicação de dados entre processadores e redundância computacional para malha com tratamento nRN após balanceamento da divisão de tarefas.

Processadores	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1		1,7	1,9		0,9	0,7	0,4	0,1							0,0	0,3
2	1,9		1,7		0,5	0,3	0,1								0,2	0,3
3	2,2	2,1		3,8	0,1	0,5	0,1	0,0								0,5
4			4,3		0,1	0,7	0,5	0,5								0,6
5	1,1	0,6	0,2	0,1		1,5	1,4		0,1	0,1	0,0	0,0	0,2	0,3	0,6	0,6
6	0,9	0,4	0,7	0,9	1,9		1,4							0,0	0,1	0,7
7	0,3	0,1	0,1	0,6	1,8	1,4		2,6	0,2	0,3	0,4	0,3	0,3	0,3	0,2	0,2
8	0,0		0,0	0,5			2,9		1,7	0,5	0,2	0,0	0,2	0,1	0,1	
9					0,1		0,3	2,0		0,6			0,5			
10					0,1		0,3	0,6	0,7		1,4		0,2			
11							0,5	0,3		1,8		2,3	0,8			
12							0,3	0,0	0,6		2,3		0,9			
13					0,1		0,4	0,2		0,3	1,0	1,0		2,6		
14					0,3	0,0	0,3	0,1					2,9		2,6	
15	0,0	0,2		0,0	0,5	0,1	0,2	0,1						2,9		2,5
16	0,3	0,4	0,6	0,6	0,6	0,8	0,2								3,0	
Nós	**6,8**	**3,7**	**7,5**	**6,5**	**6,1**	**5,4**	**9,0**	**6,3**	**3,4**	**3,6**	**5,4**	**3,6**	**5,9**	**6,2**	**6,7**	**5,5**
Elementos	**125**	**125**	**124**	**130**	**147**	**128**	**145**	**130**	**70**	**72**	**104**	**90**	**77**	**81**	**96**	**123**

Total de nós comunicados entre processadores:	**91**
Total de elementos processados:	**1766**
Redundância em elementos:	**18%**
Eficiência no processamento de elementos:	**85%**
Índice de troca de dados:	**34%**

3 COMUNICAÇÃO DE DADOS ATRAVÉS DA REDE

3.1 O PROBLEMA DA ORDEM DE COMUNICAÇÃO

Em máquinas paralelas de memória compartilhada, a comunicação de dados entre os processos sendo executados em cada processador lógico é realizada através da memória principal ou, no caso de processadores multi-núcleo, parcial ou totalmente através da memória *cache*. Estes são sub-sistemas com taxa de transferência (*throughput*) bastante mais alta e latência bastante mais baixa que os demais sub-sistemas de um computador, especificamente as unidades de armazenamento e a rede lógica.

Em máquinas paralelas de memória distribuída, no entanto, a comunicação entre processadores lógicos ou grupos de processadores lógicos se dá obrigatoriamente através da rede, sendo, muitas vezes, um importante fator de perda de eficiência na paralelização de códigos nesse tipo de máquina. Essa perda é tanto maior quanto mais importante (do ponto de vista do tempo de processamento) for a comunicação de dados entre processadores frente ao processamento propriamente dito dos dados alocados a cada processador. Cenários onde essa influência pode ser notada é o de problemas pequenos sendo processados em paralelo por um conjunto muito grande de processadores.

O tempo gasto na comunicação de dados depende de vários fatores: da latência e taxa de transferência do padrão de rede lógica empregada (relacionadas fundamentalmente com as características de *hardware* das interfaces de rede, *switches* e cabeamento utilizados), da quantidade de dados a ser trocada entre os processadores (relacionada com o problema sendo processado e com a divisão de tarefas utilizada) e do número de comunicações entre processadores (relacionado com a divisão de tarefas empregada e com o algoritmo utilizado) e do ordenamento ou organização dos processos de comunicação.

Quando o número de processadores lógicos utilizados torna-se grande, e a divisão de tarefas empregada faz com que exista troca de dados entre cada processador presente e vários outros, a ordem como os diversos processadores se comunicam entre si torna-se importante para que não sejam criadas filas de espera no processo de comunicação.

Nos *switches* normalmente utilizados para conexão de rede, a máxima largura de banda oferecida pode ser utilizada por cada porta desde que não hajam comunicações simultâneas endereçadas a uma mesma porta. Se um processador lógico conectado a uma porta A está se comunicando a um outro conectado a uma porta B, e um terceiro conectado a uma porta C está se comunicando simultaneamente a um quarto na porta D, essas comunicações acontecem na máxima taxa de transferência permitida pela rede (ou *switch*). Já se um processador lógico conectado a uma porta A se comunica simultaneamente a outros processadores conectados às portas B, C e D, as comunicações acontecem com um terço da máxima taxa de transferência. Assim, considerando o desempenho, é sempre mais interessante implementar comunicações exclusivas entre dois computadores (quando um computador está se comunicando com um segundo computador, todas as demais comunicações com esses computadores ficam em estado de espera até que a primeira se encerre, quando então outra também é iniciada em caráter exclusivo) do que comunicações simultâneas (ver *blocking send e blocking receive* em ARGONE, 2005). Contudo, essa implementação traz consigo a existência de *estados de espera* e de *filas de espera*.

Considere-se um processo paralelo no qual 8 processadores precisam, cada um, trocar a mesma quantidade de dados com todos os demais processadores presentes. A troca de dados poderia ser organizada fazendo com que cada processador seguisse a seguinte ordem: “1 até 8, com exceção de si mesmo”. A comunicação somente seria efetivada quando, em um dado instante, um processador A iniciasse o processo de comunicação com B e houvesse reciprocidade. Se não há reciprocidade, o processador A entra em *estado de espera* até que o processador B inicie a comunicação com A. Há, nesse processo, a formação de uma *fila de espera* para o processador B. A organização acima está mostrada na figura 3.1, onde os processadores são representados por hexágonos que assumem a cor cinza quando, ao final de uma dada etapa, todas as trocas de dados que deveriam ser feitas pelo processador correspondente estão completas. Um par de setas em linha cheia indicam uma comunicação em que há reciprocidade, sendo, portanto, efetuada naquela etapa de forma exclusiva e blocante. Linhas pontilhadas indicam uma comunicação iniciada por um processador que, não

havendo reciprocidade, fica em *estado de espera*, sendo o símbolo do processador alterado para um quadrado pontilhado.

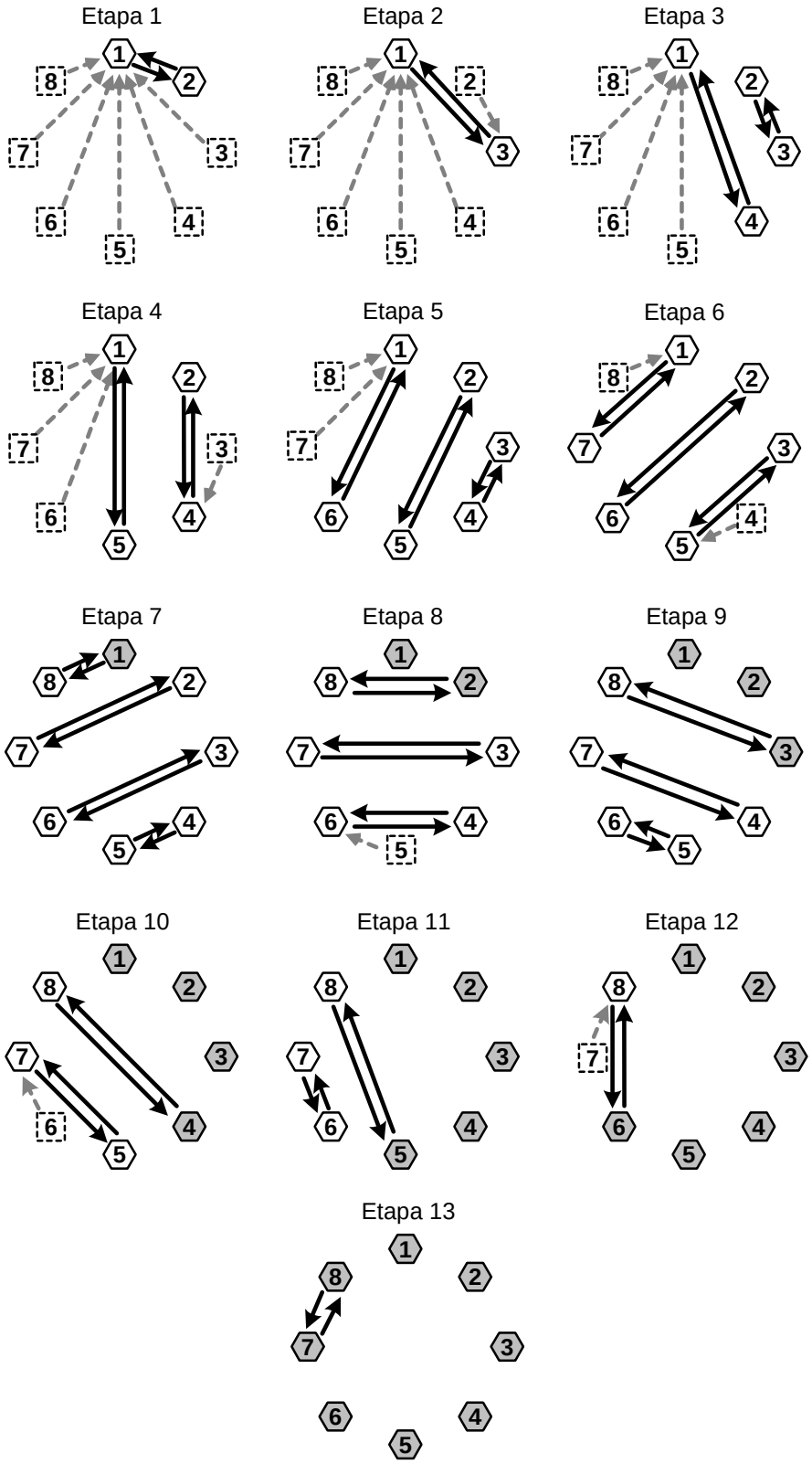


Figura 3.1: Comunicação entre 8 processadores organizada segundo uma ordem seqüencial “1 até 8 exceto a si mesmo”

Pela necessidade de reciprocidade e exclusividade na comunicação entre os processadores, e em função da regra de ordenamento adotada, são necessárias treze etapas para completar o processo total de comunicação de cada um dos oito processadores com todos os demais. O número de etapas necessárias para completar o processo de comunicação para a regra utilizada é definido por *2n-3*, onde *n* é o número de computadores no conjunto. Muito mais interessante seria a adoção de uma regra de ordenamento na qual nunca houvesse um *estado de espera*, mas nem sempre isso é possível para qualquer número de processadores, somente para algumas configurações.

Utilizando a API MPI, a execução de uma operação de comunicação blocante de um processador lógico com outro coloca automaticamente o processador que iniciou o processo em *estado de espera* quando não há reciprocidade na comunicação. O processador sai automaticamente do *estado de espera* é resolvido quando a reciprocidade se estabelece.

3.2 UMA ORDEM DE COMUNICAÇÃO GENÉRICA E EFICIENTE

Quando o processo de divisão de tarefas resulta em um padrão de comunicação entre processadores em que um processador precise somente se comunicar com os processadores imediatamente anterior e imediatamente posterior na ordem do conjunto, como é o caso do tratamento 1RN descrito na seção 3.2, uma regra simples e eficiente para a comunicação pode ser postulada da forma: “Se o número do processador é ímpar, primeiro comunique com o processador posterior, depois com o anterior; se é par, o contrário”. A aplicação dessa regra pode ser visualizada na figura 3.2 para um conjunto de 8 processadores. Com apenas 2 etapas todos os processos de comunicação entre processadores são concluídos.

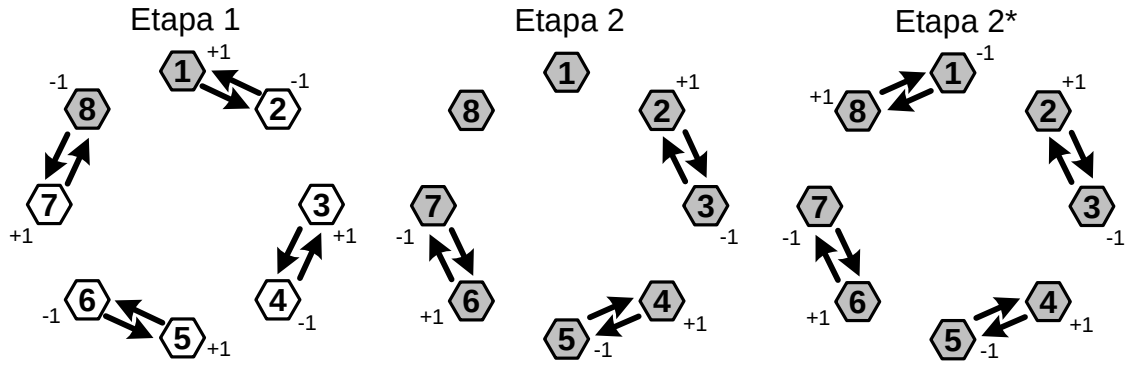


Figura 3.2: Comunicação entre 8 processadores em que cada processador se comunica apenas com o imediatamente anterior e imediatamente posterior, adequada ao tratamento 1RN para divisão de tarefas.

Os números colocados ao lado do símbolo de cada processador indicam o quanto deve ser somado ao número do próprio processador para designar o processador com o qual a comunicação será realizada.

Caso a malha do problema tivesse uma geometria circular (um toro ou anel, sendo dividida no processo de divisão de tarefas ao longo do seu eixo circular) haveria também a necessidade de comunicação entre os processadores 1 e 8. Uma ordem adequada pode ser obtida considerando-se também a lista de processadores " circular" ou " sem fim"; o processador posterior ao 8 seria o 1, e o anterior ao 1 seria o 8. Aplicando-se a esses processadores o que foi feito com os demais para identificar o número do processador com o qual a comunicação será realizada (se par, primeiro somar 1 ao número do processador, depois diminuir 1; se ímpar, o contrário), os processos de comunicação são totalmente concluídos na segunda etapa, como mostrado na figura 3.2 como Etapa 2*.

Por observação, uma extensão da configuração circular de comunicação pode ser estabelecida para um caso genérico de *n* processadores que precisam trocar dados com vários outros (ou mesmo todos os demais *n-1* processadores do conjunto).

O algoritmo desenvolvido segue os seguintes passos:

I. Os processadores presentes no conjunto são colocados em uma lista circular. O processador seguinte ao último é o primeiro, e o anterior ao primeiro é o último.

II. Cria-se uma matriz de incrementos de dimensões *n x n-1* com a lei de formação de sus termos INC_{ij} dada por

$$k = \text{int}\left(\frac{i+1}{2}\right) \quad l = \text{mod}\left(\frac{i+1}{2}\right) \quad INC_{ij} = -k(-1)^{\text{int}\left(\frac{j+k-1}{k}+l\right)} \tag{3.1}$$

onde a linha *i* representa a etapa de comunicação para um determinado processador designado pela coluna *j, int* representa a parte inteira da divisão e *mod* o resto da divisão inteira.

III. O processo é iniciado adicionando-se ao número de cada processador o incremento correspondente à respectiva coluna da primeira linha da matriz. O resultado indica o número do processador com o qual deverá haver comunicação.

IV. Se não há troca de dados prevista com o processador indicado pela matriz de incrementos, o processador avança individualmente para sua próxima etapa de comunicação (próxima linha na matriz). Em uma dada etapa global, cada processador poderá estar em uma etapa diferente de comunicação (linha diferente da matriz).

V. Se não há reciprocidade na comunicação, o processador fica em *estado de espera* até que ela ocorra.

VI. Se há reciprocidade, a comunicação é efetuada e cada processador envolvido avança individualmente para sua próxima etapa de comunicação.

VII. Os passos IV a VI são repetidos até que todos os processadores completem todas as *n-1* etapas de comunicação.

Um exemplo da matriz de incrementos para 16 processadores está mostrado na Tabela 3.1

Tabela 3.1: Exemplo de matriz de incrementos para um conjunto de 16 processadores lógicos.

		Processadores															
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Etapas	**1**	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1
	2	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1	-1	1
	3	2	2	-2	-2	2	2	-2	-2	2	2	-2	-2	2	2	-2	-2
	4	-2	-2	2	2	-2	-2	2	2	-2	-2	2	2	-2	-2	2	2
	5	3	3	3	-3	-3	-3	3	3	3	-3	-3	-3	3	3	3	-3
	6	-3	-3	-3	3	3	3	-3	-3	-3	3	3	3	-3	-3	-3	3
	7	4	4	4	4	-4	-4	-4	-4	4	4	4	4	-4	-4	-4	-4
	8	-4	-4	-4	-4	4	4	4	4	-4	-4	-4	-4	4	4	4	4
	9	5	5	5	5	5	-5	-5	-5	-5	-5	5	5	5	5	5	-5
	10	-5	-5	-5	-5	-5	5	5	5	5	5	-5	-5	-5	-5	-5	5
	11	6	6	6	6	6	6	-6	-6	-6	-6	-6	-6	6	6	6	6
	12	-6	-6	-6	-6	-6	-6	6	6	6	6	6	6	-6	-6	-6	-6
	13	7	7	7	7	7	7	7	-7	-7	-7	-7	-7	-7	-7	7	7
	14	-7	-7	-7	-7	-7	-7	-7	7	7	7	7	7	7	7	-7	-7
	15	8	8	8	8	8	8	8	8	-8	-8	-8	-8	-8	-8	-8	-8

A aplicação do algoritmo para conjuntos com 4, 5, 7, 8 e 11 processadores está mostrado nas figuras 3.3 a 3.7. Considerando que, numa configuração ideal com *n* processadores, o número mínimo de etapas de comunicação seria *n-1*, pode-se definir a penalização como sendo o número de etapas globais de comunicação que excede esse valor mínimo.

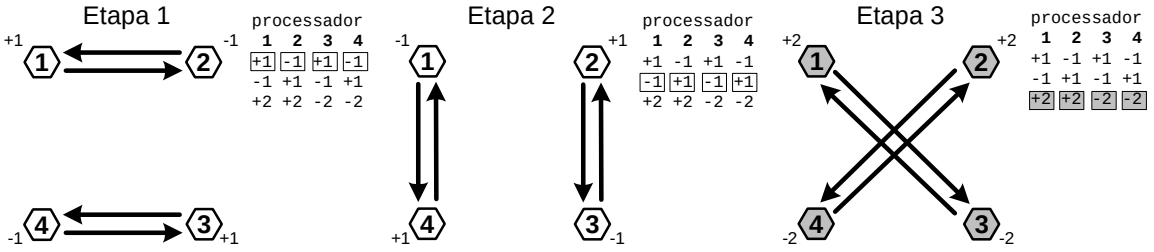


Figura 3.3: Comunicação entre 4 processadores em 3 etapas globais, configuração sem *tempo de espera*, 3 etapas, penalização nula .

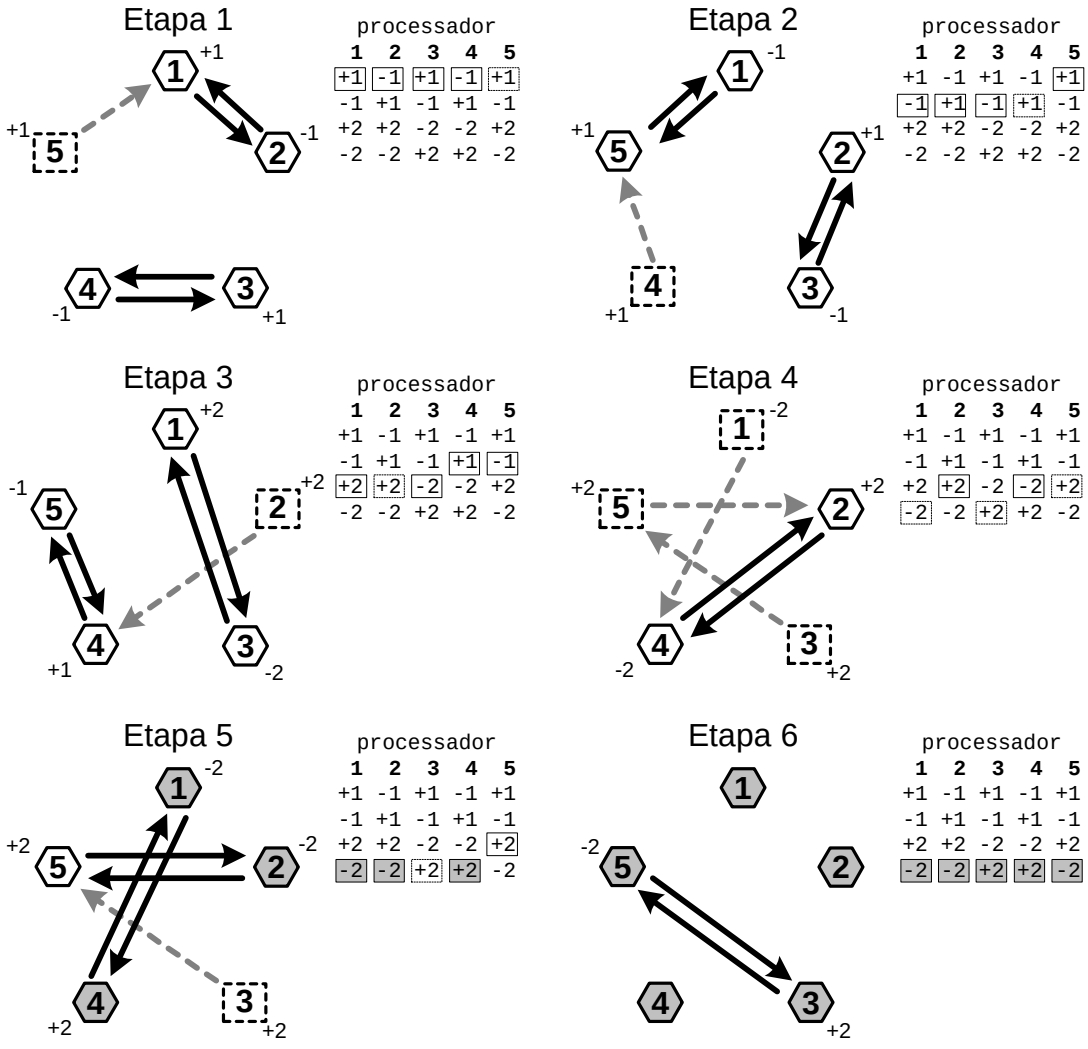


Figura 3.4: Comunicação entre 5 processadores em 6 etapas globais, configuração com *tempo de espera* e penalização de 2 etapas.

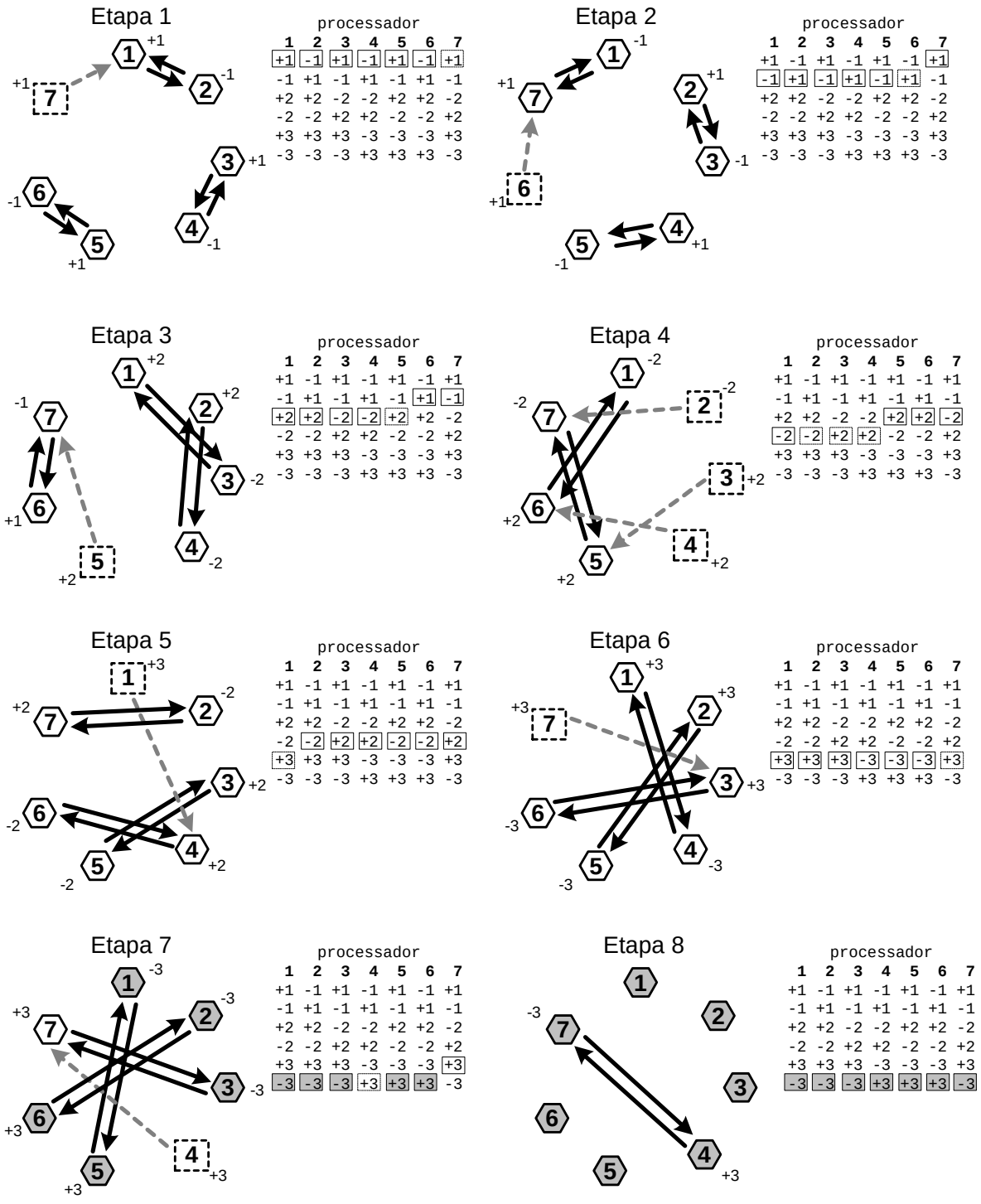


Figura 3.5: Comunicação entre 7 processadores em 8 etapas globais, configuração com *tempo de espera* e penalização de 2 etapas.

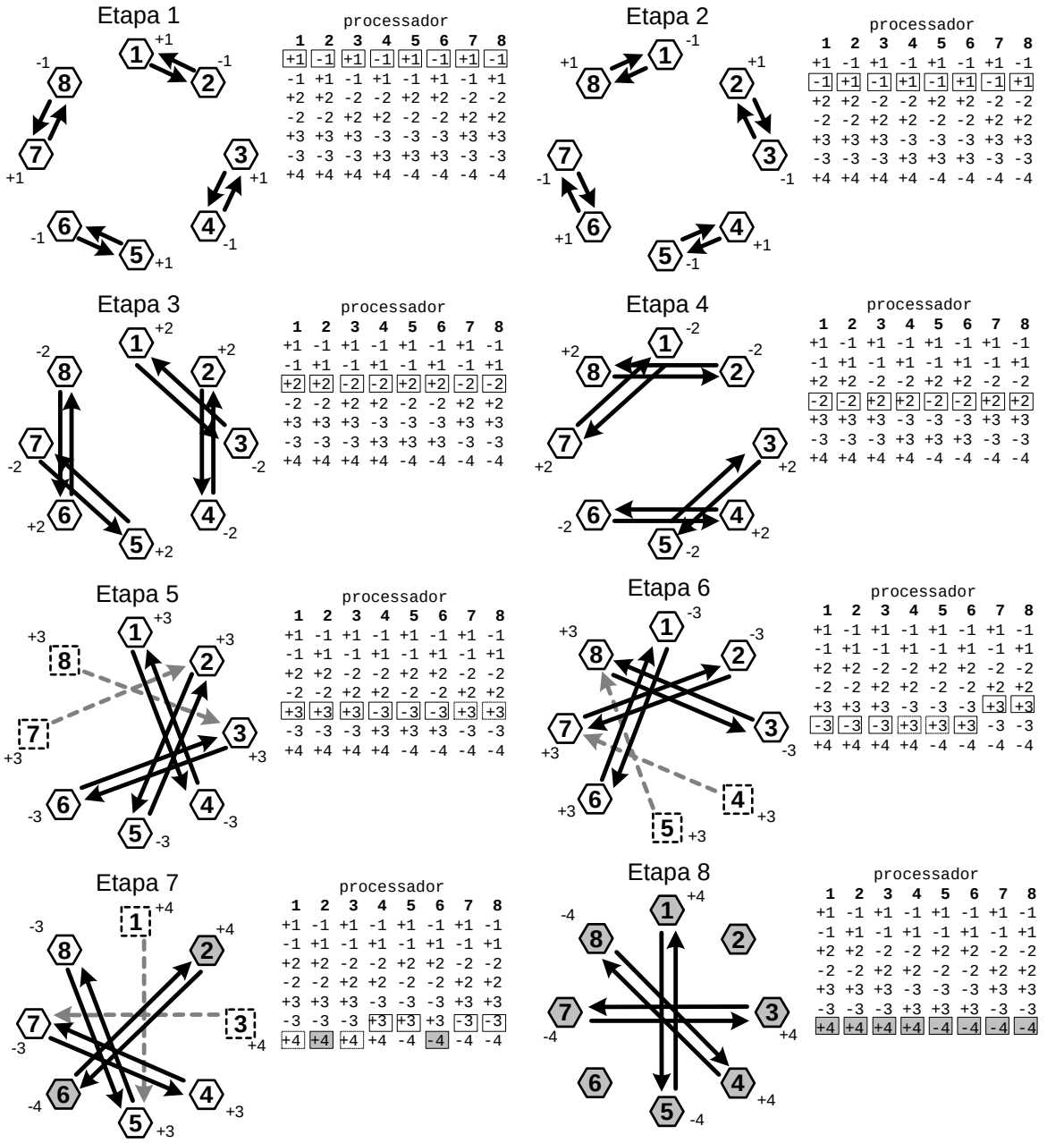
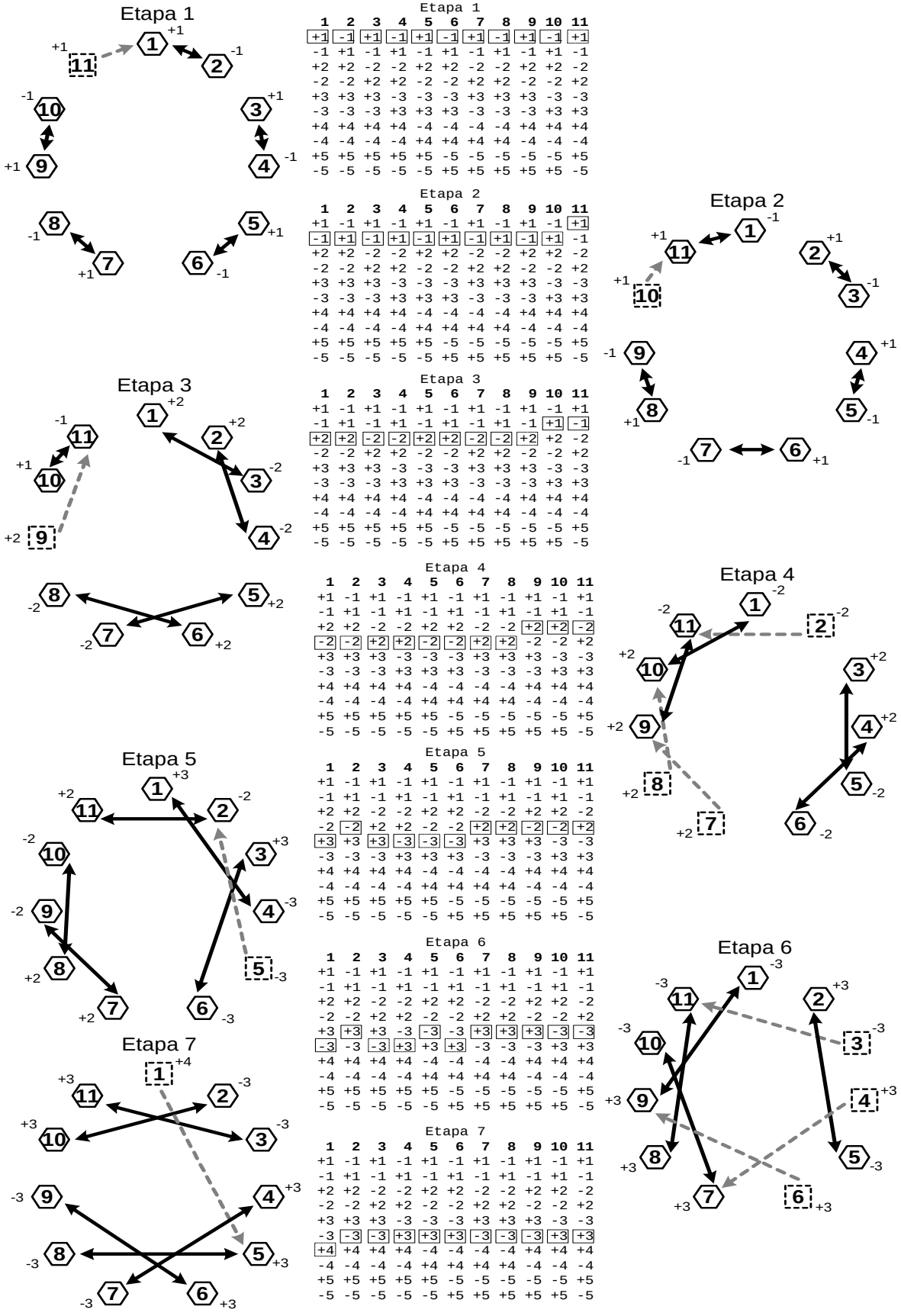


Figura 3.6: Comunicação entre 8 processadores em 8 etapas globais, configuração com *tempo de espera* e penalização de 1 etapa.

A penalização obtida nas diversas configurações tem valores relativamente baixos, embora a mesma tenda a aumentar com o número de processadores envolvidos.

O algoritmo proposto é adequado a ser usado de forma geral para todos os casos. Os casos representados nas figuras 3.3. a 3.7 correspondem à matrizes de comunicação totalmente esparsas, como as tipicamente obtidas pelo procedimento de divisão de tarefas baseado em nós para malhas sem tratamento.



Etapa 1

1	2	3	4	5	6	7	8	9	10	11
+1	-1	+1	-1	+1	-1	+1	-1	+1	-1	+1
-1	+1	-1	+1	-1	+1	-1	+1	-1	+1	-1
+2	+2	-2	-2	+2	+2	-2	-2	+2	+2	-2
-2	-2	+2	+2	-2	-2	+2	+2	-2	-2	+2
+3	+3	+3	-3	-3	-3	+3	+3	+3	-3	-3
-3	-3	-3	+3	+3	+3	-3	-3	-3	+3	+3
+4	+4	+4	+4	-4	-4	-4	-4	+4	+4	+4
-4	-4	-4	-4	+4	+4	+4	+4	-4	-4	-4
+5	+5	+5	+5	+5	-5	-5	-5	-5	-5	+5
-5	-5	-5	-5	-5	+5	+5	+5	+5	+5	-5

Etapa 2

1	2	3	4	5	6	7	8	9	10	11
+1	-1	+1	-1	+1	-1	+1	-1	+1	-1	+1
-1	+1	-1	+1	-1	+1	-1	+1	-1	+1	-1
+2	+2	-2	-2	+2	+2	-2	-2	+2	+2	-2
-2	-2	+2	+2	-2	-2	+2	+2	-2	-2	+2
+3	+3	+3	-3	-3	-3	+3	+3	+3	-3	-3
-3	-3	-3	+3	+3	+3	-3	-3	-3	+3	+3
+4	+4	+4	+4	-4	-4	-4	-4	+4	+4	+4
-4	-4	-4	-4	+4	+4	+4	+4	-4	-4	-4
+5	+5	+5	+5	+5	-5	-5	-5	-5	-5	+5
-5	-5	-5	-5	-5	+5	+5	+5	+5	+5	-5

Etapa 3

1	2	3	4	5	6	7	8	9	10	11
+1	-1	+1	-1	+1	-1	+1	-1	+1	-1	+1
-1	+1	-1	+1	-1	+1	-1	+1	-1	+1	-1
+2	+2	-2	-2	+2	+2	-2	-2	+2	+2	-2
-2	-2	+2	+2	-2	-2	+2	+2	-2	-2	+2
+3	+3	+3	-3	-3	-3	+3	+3	+3	-3	-3
-3	-3	-3	+3	+3	+3	-3	-3	-3	+3	+3
+4	+4	+4	+4	-4	-4	-4	-4	+4	+4	+4
-4	-4	-4	-4	+4	+4	+4	+4	-4	-4	-4
+5	+5	+5	+5	+5	-5	-5	-5	-5	-5	+5
-5	-5	-5	-5	-5	+5	+5	+5	+5	+5	-5

Etapa 4

1	2	3	4	5	6	7	8	9	10	11
+1	-1	+1	-1	+1	-1	+1	-1	+1	-1	+1
-1	+1	-1	+1	-1	+1	-1	+1	-1	+1	-1
+2	+2	-2	-2	+2	+2	-2	-2	+2	+2	-2
-2	-2	+2	+2	-2	-2	+2	+2	-2	-2	+2
+3	+3	+3	-3	-3	-3	+3	+3	+3	-3	-3
-3	-3	-3	+3	+3	+3	-3	-3	-3	+3	+3
+4	+4	+4	+4	-4	-4	-4	-4	+4	+4	+4
-4	-4	-4	-4	+4	+4	+4	+4	-4	-4	-4
+5	+5	+5	+5	+5	-5	-5	-5	-5	-5	+5
-5	-5	-5	-5	-5	+5	+5	+5	+5	+5	-5

Etapa 5

1	2	3	4	5	6	7	8	9	10	11
+1	-1	+1	-1	+1	-1	+1	-1	+1	-1	+1
-1	+1	-1	+1	-1	+1	-1	+1	-1	+1	-1
+2	+2	-2	-2	+2	+2	-2	-2	+2	+2	-2
-2	-2	+2	+2	-2	-2	+2	+2	-2	-2	+2
+3	+3	+3	-3	-3	-3	+3	+3	+3	-3	-3
-3	-3	-3	+3	+3	+3	-3	-3	-3	+3	+3
+4	+4	+4	+4	-4	-4	-4	-4	+4	+4	+4
-4	-4	-4	-4	+4	+4	+4	+4	-4	-4	-4
+5	+5	+5	+5	+5	-5	-5	-5	-5	-5	+5
-5	-5	-5	-5	-5	+5	+5	+5	+5	+5	-5

Etapa 6

1	2	3	4	5	6	7	8	9	10	11
+1	-1	+1	-1	+1	-1	+1	-1	+1	-1	+1
-1	+1	-1	+1	-1	+1	-1	+1	-1	+1	-1
+2	+2	-2	-2	+2	+2	-2	-2	+2	+2	-2
-2	-2	+2	+2	-2	-2	+2	+2	-2	-2	+2
+3	+3	+3	-3	-3	-3	+3	+3	+3	-3	-3
-3	-3	-3	+3	+3	+3	-3	-3	-3	+3	+3
+4	+4	+4	+4	-4	-4	-4	-4	+4	+4	+4
-4	-4	-4	-4	+4	+4	+4	+4	-4	-4	-4
+5	+5	+5	+5	+5	-5	-5	-5	-5	-5	+5
-5	-5	-5	-5	-5	+5	+5	+5	+5	+5	-5

Etapa 7

1	2	3	4	5	6	7	8	9	10	11
+1	-1	+1	-1	+1	-1	+1	-1	+1	-1	+1
-1	+1	-1	+1	-1	+1	-1	+1	-1	+1	-1
+2	+2	-2	-2	+2	+2	-2	-2	+2	+2	-2
-2	-2	+2	+2	-2	-2	+2	+2	-2	-2	+2
+3	+3	+3	-3	-3	-3	+3	+3	+3	-3	-3
-3	-3	-3	+3	+3	+3	-3	-3	-3	+3	+3
+4	+4	+4	+4	-4	-4	-4	-4	+4	+4	+4
-4	-4	-4	-4	+4	+4	+4	+4	-4	-4	-4
+5	+5	+5	+5	+5	-5	-5	-5	-5	-5	+5
-5	-5	-5	-5	-5	+5	+5	+5	+5	+5	-5

Figura 3.7: Comunicação entre 11 processadores em 13 etapas globais, configuração com *tempo de espera* e penalização de 3 etapas *(continua...)*.

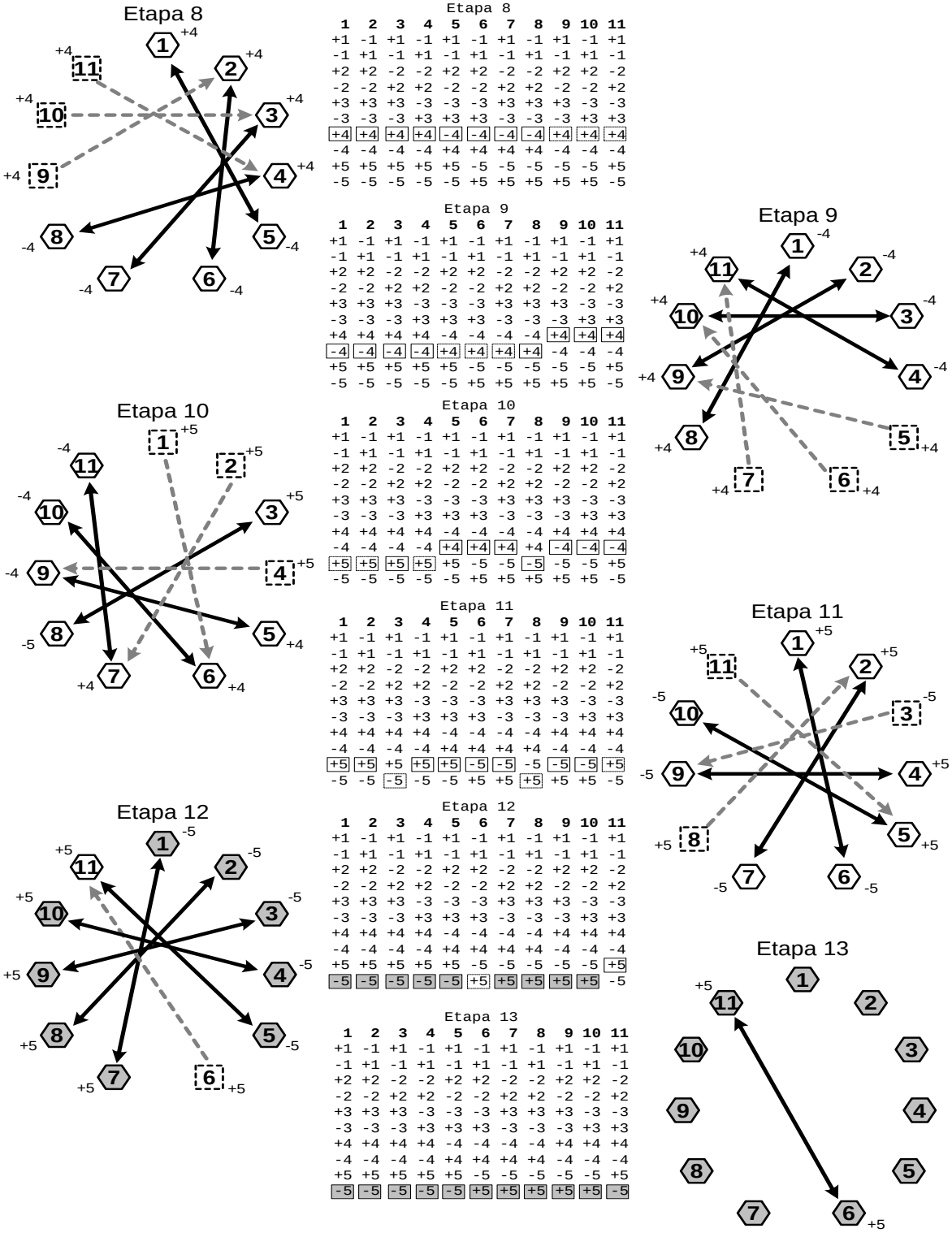


Figura 3.7: Comunicação entre 11 processadores em 13 etapas globais, configuração com *tempo de espera* e penalização de 3 etapas.

O tratamento nRN introduzido na seção 2.3 gera matrizes esparsas de comunicação entre processadores, mas em geral não há comunicação de cada processador com todos os outros. O passo IV do algoritmo prevê esses casos, e o número de etapas globais de comunicação

obtidos para os casos representados nas figuras 3.3 a 3.7 podem ser vistos como limites superiores para conjuntos de processadores representados. O algoritmo reproduz também situações como as mostradas na figura 3.2, em que somente há troca de dados entre cada processador e os processadores imediatamente anterior e posterior, típicas da aplicação do tratamento 1RN descrito em na seção 2.2.

4 ALGORITMOS SELECIONADOS PARA PARALELIZAÇÃO

Para a escolha dos algoritmos a serem paralelizados, dois critérios foram utilizados: cobrirem os aspectos com maior demanda de esforço computacional da simulação de problemas da Dinâmica dos Sólidos Computacional e da Dinâmica dos Fluídos Compuacional através do Método dos Elementos Finitos, e serem adequados à solução de problemas grandes.

4.1 MÉTODO DOS GRADIENTES CONJUGADOS

Em problemas de Mecânica dos Sólidos, a aplicação do Método dos Elementos Finitos resulta na solução do sistema de equações lineares indicado genericamente em (4.1). **A** em um bom número de problemas representa a matriz de rigidez global da estrutura sendo analizada e **b** o vetor de cargas nodais equivalentes.

$$\mathbf{Ax} = \mathbf{b} \tag{4.1}$$

A solução do sistema é a etapa que mais consome tempo de processamento à medida que o tamanho das malhas analisadas cresce.

Os algoritmos de solução ou *solvers* diretos, como os que utilizam a decomposição de Gauss ou de Cholesky, apesar de apresentarem excelente desempenho para sistemas pequenos, tornam-se inviáveis na solução de grandes sistemas de equações resultantes de malhas com muitos nós, pois dependem da montagem de uma matriz de rigidez global **A** que nesses sistemas facilmente esgota a disponibilidade de memória principal dos computadores de uso corrente. Apesar de que a cada ciclo tecnológico, a quantidade de memória principal disponível se torna maior e, comparativamente, mais barata, também a complexidade dos modelos torna-se maior, retro-alimentada pela disponibilidade de maior poder de processamento e de maior quantidade de memória, de modo que a demanda por memória nunca é inteiramente satisfeita.

As formas de organização das triangulações acima que permitem o uso da memória auxiliar, como a solução *Frontal* (IRONS, 1970, HINTON e OWEN, 1977), em que um conjunto mínimo de equações precisa permanecer na memória principal, e tão logo as transformações lineares sobre o sistema de equações (decomposições de Gauss ou Cholesky) envolvendo uma dada equação são completadas, a equação é transferida para a memória secundária (disco rígido ou outras formas de armazenamento), apesar de parcialmente resolverem o problema da memória principal disponível, sofrem em termos de desempenho, uma vez que o sub-sistema de armazenamento que passa a ser utilizado no lugar da memória principal é bastante mais lento que esta.

Além disso, os *solvers* diretos são de difícil paralelização, uma vez que há uma enorme dependência entre as operações envolvidas sobre as variáveis do sistema nos processos de triangulação. A divisão do sistema de equações em sub-conjuntos alocados a diversos processadores lógicos traz como conseqüências a troca de grandes quantidades de dados entre os mesmos (as equações completas da região de interface entre os sub-conjuntos) e necessidade de concentrar o processamento na etapa final da decomposição em um único processador.

Os métodos de solução iterativos, como o método de Gauss-Seidel e, em especial, o Método dos Gradientes Conjugados, em suas formulações elemento-por-elemento, são bastante mais simples de serem paralelizados, especialmente em configurações de memória distribuída, pois têm genericamente o formato da equação (2.1), de modo que, em cada iteração, as equações são desacopladas, podendo ser resolvidas em qualquer ordem, permitindo sua sub-divisão em grupos que podem ser alocados a diversos processadores lógicos. A demanda por memória principal desses métodos também é menor, uma vez que não é necessária a montagem da matriz de rigidez global.

O método de solução escolhido para ser paralelizado foi o Métodos dos Gradientes Conjugados. Seu emprego tende a ser vantajoso em termos de tempo de processamento comparado aos *solvers* diretos à medida que o tamanho do sistema de equações cresce, em especial quando utilizados com pré-condicionadores.

4.1.1 Método dos gradientes conjugados com pré-condicionador diagonal ou de Jacobi

O pré-condicionador diagonal ou de Jacobi é especialmente indicado nos casos em que as variáveis do sistema de equações são dimensionalmente bastante diferentes. É um dos pré-condicionadores com menor demanda por memória, pois a única estrutura extra de armazenamento necessária é um vetor com os termos da diagonal principal da matriz **A.**

O algoritmo correspondente à formulação elemento-por-elemento, em sua versão paralela otimizada para configurações de memória distribuída está baseado no apresentado por Duarte Filho (2002) e está mostrado na tabela 4.1, onde:

n: contador de iterações;

$\mathbf{x_0}$: vetor estimativa inicial;

b: vetor independente do sistema de equações (vetor de cargas nodais equivalentes);

bb: produto escalar $\mathbf{b} \cdot \mathbf{b}$ equivalente ao quadrado do módulo do vetor resíduo inicial;

r: vetor resíduo;

rr: produto escalar $\mathbf{r} \cdot \mathbf{r}$ equivalente ao quadrado do módulo do vetor resíduo;

z: vetor resíduo precondicionado;

rz: $\mathbf{r} \cdot \mathbf{z}$ **;**

p: vetor direção de busca global;

$\mathbf{p^e}$: vetor direção de busca para as equações nodais correspondentes aos nós do elemento *e*;

W: matriz de precondicionamento diagonal ou de Jacobi, correspondente aos termos da diagonal principal de **A.** Todos os termos não pertencentes à diagonal são nulos;

A: matriz de rigidez global;

$\mathbf{A^e}$: matriz de rigidez em coordenadas globais do elemento *e*;

ap: vetor resultante do produto **A p**;

pap: produto escalar $\mathbf{p} \cdot \mathbf{ap}$;

N_i: conjunto de equações nodais correspondentes aos nós alocados ao processador lógico *i*;

N_i^+: conjunto de equações nodais correspondentes aos nós dos elementos conectados aos nós alocados ao processador lógico *i*;

ΔN_{ij}: equações nodais alocadas ao processador lógico *i* correspondentes a nós pertencentes a elementos conectados aos nós alocados ao processador *j*;

E_i elementos conectados aos nós alocados ao processador lógico *i*.

Tabela 4.1: Algoritmo do Método dos Gradientes Conjugados, formulação elemento-por-elemento, paralelização otimizada para configurações de memória distribuída.

Etapa 1. Inicialização		
(a)	$n = 0$	
(b)	$bb = \mathbf{b} \cdot \mathbf{b}$	
(c)	$\mathbf{x_n} = 0$	para N_i
(d)	$\mathbf{r_n} = \mathbf{b}$	para N_i^+
(e)	$\mathbf{p_n} = \mathbf{z_n} = \mathbf{W}^{-1}\mathbf{r}_n$	para N_i^+
(f)	$rz_n = \mathbf{r}_n \cdot \mathbf{z}_n$	para N_i
Etapa 2. Atualização dos vetores estimativa e resíduo		
(g)	$\mathbf{ap}_n = \sum_{E_i} \mathbf{A}^e \mathbf{p}_n^e$	para N_i
(h)	$pap_n = \mathbf{p}_n \cdot \mathbf{ap}_n$	para N_i
(i)	Soma dos valores rz_n e pap_n entre todos os processadores lógicos	
(j)	Envia $\mathbf{ap}_n$ para processador lógico j	para ΔN_{ij}
(k)	Recebe $\mathbf{ap}_n$ do processador lógico j	para ΔN_{ji}
(l)	Repete (j) e (k) para todos os processadores lógicos *j* com os quais *i* troca dados	
(m)	$\alpha_n = \frac{rz_n}{pap_n}$	

Tabela 4.1 (*continuação*): Algoritmo do Método dos Gradientes Conjugados, formulação elemento-por-elemento, paralelização otimizada para configurações de memória distribuída.

Etapa 3. Atualização do vetor direção de busca		
(n)	$\mathbf{x}_{n+1} = \mathbf{x}_n + \alpha_n \mathbf{p}_n$	para N_i
(o)	$\mathbf{r}_{n+1} = \mathbf{r}_n - \alpha_n \mathbf{a}\mathbf{p}_n$	para N_i^+
(p)	$\mathbf{z}_{n+1} = \mathbf{W}^{-1}\mathbf{r}_{n+1}$	para N_i^+
(q)	$rz_{n+1} = \mathbf{r}_{n+1} \cdot \mathbf{z}_{n+1}$	para N_i
(r)	$rr_{n+1} = \mathbf{r}_{n+1} \cdot \mathbf{r}_{n+1}$	para N_i
(s)	Soma dos valores rz_{n+1} e rr_{n+1} entre todos os processadores lógicos	
(t)	$\beta_n = \dfrac{rz_{n+1}}{rz_n}$	
(u)	$\mathbf{p}_{n+1} = \mathbf{z}_{n+1} + \beta_n \mathbf{p}_n$	para N_i^+
Etapa 4. Teste de convergência		
(v)	Se $\sqrt{\dfrac{rr_{n+1}}{bb}}$ > tolerância, $n = n+1$, retorna à etapa 2	

Para que , no passo (g), o produto da matriz de rigidez do elemento em coordenadas globais $\mathbf{A}^e$ pelo vetor direção de busca $\mathbf{p}^e$ resulte em valores corretos para o conjunto N_i, é preciso que o vetor $\mathbf{p}^e$ em cada processador lógico contenha os valores corretos para todos os nós do elemento correspondente. Isso significa que cada processador deve ter atualizado em cada iteração os valores de $\mathbf{p}^e$ correspondentes aos nós dos elementos conectados ao grupo de nós (grupo de equações nodais) alocado ao processador, dando origem ao conjunto N_i^+, o qual define a abrangência das operações realizadas em (d), (e), (o), (p) e (u). Além disso, como a obtenção de $\mathbf{ap}_n$ no passo (g) é feita elemento por elemento (representado pelo somatório) de forma a evitar a montagem da matriz de rigidez global **A**, o grupo de elementos envolvido no processo é composto por todos os elementos conectados aos nós alocados ao processador (E_i).

O grupo E_i de elementos de um dado processador lógico é formado pelos elementos próprios, conectados exclusivamente aos nós alocados a esse processador (E_i^P), e pelos elementos conectados tanto a nós do grupo desse processador quanto a nós dos grupos de outros processadores (E_i^C), que são os *elementos comuns* definidos na seção 2.2. Estes últimos estão relacionados com a redundância de esforço computacional decorrente da divisão de tarefas empregada para paralelização.

Os passos (i), (j), (k) e (s) representam comunicação entre processadores lógicos. O passos (i) e (s) representam uma operação de comunicação na qual cada processador lógico fornece o seu valor local das variáveis a serem comunicadas (*rz, pap, rr*) e recebe a soma desse valor com os correspondentes de todos os outros processadores presentes no conjunto. Os passos (j) e (k) representam uma comunicação exclusiva e recíproca entre um par de processadores (*blocking send* e *blocking receive* do MPI).

O algoritmo apresentado pode ser implementado sem que nenhum processador assuma a coordenação do fluxo de dados entre os demais (*hostless* program), a não ser no final do processo iterativo para agrupar os resultados dos diversos processadores. Cada processador pode executar exatamente o mesmo código, caracterizando uma implementação do tipo Programa Único – Múltiplos Dados (*Single Program – Multiple Data*). As técnicas descritas nos capítulos 2 e 3 podem ser utilizadas para minimizar a redundância de processamento e o tempo de comunicação entre processadores lógicos.

4.1.2 Método dos gradientes conjugados com pré-condicionador de Cholesky

O pré-condicionador de Cholesky é bastante mais robusto que o pré-condicionador diagonal, sendo o mais adequado para sistemas mal condicionados. Seus requerimentos de memória também são maiores, exigindo o armazenamento das matrizes dos elementos fatoradas, ou seu recálculo a cada iteração, com conseqüente perda de velocidade de solução.

No Método dos Gradientes Conjugados com pré-condicionador de Cholesky, os passos (e) e (p) no algoritmo da tabela 5.1 são modificados para:

(e) $\mathbf{p_n} = \mathbf{z_n} = \mathbf{B}^{-1}\mathbf{r}_n$ para N_i^+

(p) $\mathbf{z}_{n+1} = \mathbf{B}^{-1}\mathbf{r}_{n+1}$ para N_i^+

onde **B** é a matriz de pré-condicionamento de Cholesky, definida pela expressão:

$$\mathbf{B} = \mathbf{W}^{1/2} \times \prod_{e=1}^{Nel} L\left[\mathbf{A}^e\right] \times \prod_{e=Nel}^{1} U\left[\mathbf{A}^e\right] \times \mathbf{W}^{1/2} \tag{4.2}$$

onde o símbolo $\prod$ representa produto, *Nel* é o número de elementos, **W** = diag(**A**) e *L*[.] e *U*[.] são os fatores triangular inferior e triangular superior da fatoração de Cholesky, respectivamente.

A implementação da equação (4.2) constitui-se nas seguintes etapas:

Antes do Método dos Gradientes Conjugados propriamente dito ser iniciado, as matrizes de rigidez fatoradas dos elementos são calculadas. O passo (e) do algoritmo da Tabela 5.1 fica explicitado pelos passos indicados na Tabela 4.2

Tabela 4.2: Alterações necessárias ao algoritmo do Método dos Gradientes Conjugados, formulação elemento-por-elemento, para contemplar o pré-condicionador de Cholesky.

Passo (e) substituído por:	
(1)	$\mathbf{z}_n = \mathbf{W}^{-1/2}\,\mathbf{r}_n$
(2)	Substituição à frente de $\mathbf{z}_n$ com a matriz fatorada triangular inferior de Cholesky, para cada elemento
(3)	Retrosubstituição de $\mathbf{z}_n$ com a matriz fatorada triangular superior de Cholesky, para cada elemento
(4)	$\mathbf{p}_n = \mathbf{z}_n = \mathbf{W}^{-1/2}\,\mathbf{z}_n$

As alterações para o passo (p) são similares.

Para a implementação de uma versão paralela otimizada para máquinas de memória distribuída, é preciso dividir o grupo de elementos E_i de cada processador lógico nos subgrupos E_i^P e E_i^C. As modificações necessárias são mostradas na Tabela 4.3.

Tabela 4.3: Alterações necessárias ao algoritmo do Método dos Gradientes Conjugados, formulação elemento-por-elemento, para contemplar o pré-condicionador de Cholesky em versão paralela otimizada para máquinas de memória distribuída.

Passo (e) substituído por:		
(1)	$\mathbf{z}_n = \mathbf{W}^{-1/2}\,\mathbf{r}_m$	para N_i^+
(2)	Substituição à frente de $\mathbf{z}_n$ em $L[\mathbf{A}^e]$	para E_i^C
(3)	Substituição à frente de $\mathbf{z}_n$ em $L[\mathbf{A}^e]$	para E_i^P
(4)	Retrosubstituição de $\mathbf{z}_n$ em $L[\mathbf{A}^e]$	para E_i^P
(5)	Envia $\mathbf{z}_n$ para processador lógico j	para ΔN_{ij}
(6)	Recebe $\mathbf{z}_n$ do processador lógico j	para ΔN_{ji}
(7)	Repete (5) e (6) para todos os processadores lógicos *j* com os quais *i* troca dados	
(8)	Retrosubstituição de $\mathbf{z}_n$ em $L[\mathbf{A}^e]$	para E_i^C
(9)	$\mathbf{p}_n = \mathbf{z}_n = \mathbf{D}^{-1/2}\,\mathbf{z}_n$	para N_i^+

A alterações para o passo (p) são similares. A ordem com que a retrosubstituição é feita nos passos (4) e (8), nos grupos E_i^C e E_i^P é a inversa da utilizada nos passos (2) e (3) para os mesmos grupos, conforme a equação (4.2), assegurando a simetria para a matriz de pré-condicionamento **B**.

As operações dos passos (2), (3), (4) e (8) não são desacopladas, uma vez que o valor de uma dada variável é dependente dos valores das demais variáveis calculados na mesma iteração. Como o pré-condicionamento é aplicado elemento-por-elemento, o resultado das operações de pré-condicionamento sobre o vetor **z** é dependente da ordem do elementos. Como o versão paralela altera a ordem dos elementos, os resultados obtidos com a versão paralela em cada iteração para cada variável do sistema somente serão absolutamente idênticos ao da versão

seqüencial se esta utilizar a mesma ordem empregada por todos os processadores lógicos, ou seja substituição à frente com os elementos comuns, substituição à frente com os elementos próprios de cada processador, retrosubstituição com os elementos próprios de cada processador e retrosubstituição com os elementos comuns. Contudo, a ordem dos elementos não afeta os resultados finais (após a convergência), causando apenas pequenas difererenças no número de iterações para a convergência ser alcançada e, conseqüentemente, sobre o tempo de processamento total.

Para assegurar que a matriz $\mathbf{A}^e$ seja positiva definida, é utilizada a regularização de Winget (Hughes et al., 1987).

Percebe-se que o uso do pré-condicionador de Cholesky adiciona, em relação ao pré-condicionador Diagonal, mais uma etapa de comunicação entre os processadores lógicos para cada iteração do processo de solução, além da substituição à frente e da retrosubstituição. O impacto sobre o tempo total de solução é, na maioria das vezes, compensado pela melhor taxa de convergência que, em geral, o pré-condicionador de Cholesky propicia.

Também nessa implementação, não há necessidade de um processador que coordene o fluxo de dados entre os processadores a não ser para consolidar os resultados no final do processo iterativo.

4.2 ESCOAMENTOS COMPRESSÍVEIS EM REGIME TRANSÔNICO E SUPERSÔNICO

Para simulação de escoamentos compressíveis em regimes transônico e supersônico foi utilizado o algoritmo empregado por Burbridge (1999), Burbridge e Awruch (2000) e Bono (2008) onde, para integração no tempo, é utilizado em esquema explícito de um passo com iterações.

4.2.1 As equações de conservação

Sejam $\Omega \subset R^3$ e (0,T) os domínios espacial e temporal, respectivamente, e seja Γ o contorno de Ω. As coordenadas espaciais e temporal são designadas por **x** e *t*, respectivamente. São

consideradas as equações de Navier-Stokes sem termos de fonte governando o fluxo compressível, escritas de forma adimensional por

$$\frac{\partial \mathbf{U}}{\partial t}+\frac{\partial \mathbf{F}_i}{\partial x_i}+\frac{\partial \mathbf{G}_i}{\partial x_i}=0 \tag{4.3}$$

onde **U** é o vetor de incógnitas das variáveis de conservação e $\mathbf{F}_i$ e $\mathbf{G}_i$ os vetores de fluxo convectivo e difusivo, respectivamente. Eles são definidos por

$$\mathbf{U}=\begin{Bmatrix}\rho\\ \rho v_i\\ \rho e\end{Bmatrix},\qquad \mathbf{F}_i=\begin{Bmatrix}\rho v_j\\ \rho v_i v_j+p\,\delta_{ij}\\ v_j\left(\rho e+p\right)\end{Bmatrix},\qquad \mathbf{G}_i=\begin{Bmatrix}0\\ -\tau_{ij}\\ -\tau_{ji}v_i-q_j\end{Bmatrix} \tag{4.4}$$

onde *i, j = 1,2,3*, v_i é a componente da velocidade na direção da coordenada x_i, ρ é a massa específica, p é a pressão termodinâmica, τ_{ij} são os componentes do tensor de tensões viscosas, q_j é o vetor de fluxo de calor, e é a energia total específica e δ_{ij} é a função delta de Kronecker. Para o escoamento compressível, as seguintes escalas adimensionais são utilizadas:

$$\begin{aligned} &x_i=\frac{\tilde{x}_i}{\tilde{L}_\infty} \qquad v_i=\frac{\tilde{v}_i}{\tilde{a}_\infty} \qquad \rho=\frac{\tilde{\rho}}{\tilde{\rho}_\infty} \qquad e=\frac{\tilde{e}}{\tilde{a}_\infty^2}\\ &p=\frac{\tilde{p}}{\tilde{\rho}_\infty\,\tilde{a}_\infty^2} \qquad u=\frac{\tilde{c}_v\,\tilde{T}}{\tilde{a}_\infty^2} \qquad t=\frac{\tilde{t}}{\tilde{L}_\infty/\tilde{a}_\infty}\end{aligned} \tag{4.5}$$

onde o símbolo sobrescrito ~ indica uma grandeza dimensional, o símbolo subscrito ∞ representa um valor de referência para um estado de corrente não perturbado, a é a velocidade do som, T é a temperatura, c_v é o calor especifico a volume constante, u é a energia interna específica e L é um comprimento de referência.

Considerando que o ar se comporta como um gás caloricamente perfeito, a pressão p (que é calculada pela equação de estado) e a energia interna específica u em termos de variáveis adimensionais são dadas pela equação:

$$p=\left(\gamma-1\right)\rho\,u\,,\qquad u=c_v\,T=e-\frac{1}{2}v_i\,v_i \tag{4.6}.$$

onde a relação entre calores específicos sob pressão e volume constantes $\gamma = c_p / c_v$ é uma constante do gás e considerada igual a 1.4. A viscosidade dinâmica e a condutividade térmica dependem da temperatura, sendo portanto modeladas pela lei de Sutherland. As equações de Euler são obtidas eliminando-se de (4.3) o vetor de fluxo difusivo. Condições iniciais e de contorno devem ser adicionadas a (4.3) de modo a definir o problema de forma única.

4.2.2 A formulação de Taylor-Galerkin com esquema de um passo iterativa.

O esquema explicito de um passo de Taylor-Galerkin utilizado para integração no tempo é similar aquele apresentado por Donea (1984) e está detalhado em (BONO, 2008). Expandindo as variáveis de conservação **U** em $t = t^{n+1}$ através da série de Taylor incluindo a primeira e a segunda derivadas, é obtida a expressão

$$\Delta \mathbf{U}^{n+1} = \Delta t \left(\frac{\partial \mathbf{U}}{\partial t}^{n} + \frac{1}{2} \frac{\partial \Delta \mathbf{U}}{\partial t}^{n+1} \right) + \frac{\Delta t^2}{2} \left(\frac{\partial^2 \mathbf{U}}{\partial t^2}^{n} + \frac{1}{2} \frac{\partial^2 \Delta \mathbf{U}}{\partial t^2}^{n+1} \right) \tag{4.7}$$

onde $\Delta \mathbf{U}^{n+1} = \mathbf{U}^{n+1} - \mathbf{U}^{n}$, $\Delta t = t^{n+1} - t^{n}$ é o passo de tempo, *n* e *n+1* indicam *t* e *t+Δt*, respectivamente. Substituindo a equação (4.3) e sua derivada na equação (4.7) e desprezando os termos de alta ordem, é obtida a expressão

$$\begin{aligned} \mathbf{U}_{J+1}^{n+1} = \mathbf{U}^{n} + \Delta t \left[-\frac{\partial \mathbf{F}_i}{\partial x_i}^{n} - \frac{\partial \mathbf{G}_i}{\partial x_i}^{n} + \frac{\Delta t}{2} \frac{\partial}{\partial x_k} \left(\mathbf{A}_k \frac{\partial \mathbf{F}_i}{\partial x_i}^{n} \right) \right] + \\ + \frac{\Delta t}{2} \left[-\frac{\partial \Delta \mathbf{F}_{iJ}}{\partial x_i}^{n+1} - \frac{\partial \Delta \mathbf{G}_{iJ}}{\partial x_i}^{n+1} + \frac{\Delta t}{2} \frac{\partial}{\partial x_k} \left(\mathbf{A}_k \frac{\partial \Delta \mathbf{F}_{iJ}}{\partial x_i}^{n+1} \right) \right] \end{aligned} \tag{4.8}$$

onde *J* é um contador de iteração, $\Delta \mathbf{F}_i^{n+1} = \mathbf{F}_i^{n+1} - \mathbf{F}_i^{n}$, $\Delta \mathbf{G}_i^{n+1} = \mathbf{G}_i^{n+1} - \mathbf{G}_i^{n}$ e $\mathbf{A}_i$ é o Jacobiano convectivo definido por $\mathbf{A}_i = \partial \mathbf{F}_i / \partial \mathbf{U}$ (Hughes e Tezduyar, 1984). Os incrementos das variáveis de campo $\Delta \mathbf{U}$ devem ser obtidos através de processo iterativo, uma vez que estão definidos para o mesmo tempo t^{n+1} que os incrementos do segundo termo do lado direito da equação (4.8).

A discretização espacial é obtida aplicando-se o método clássico do resíduo ponderado de Bubnov-Galerkin no contexto do Método dos Elementos Finitos na equação (4.8). A matriz de

massa consistente é substituída pela matriz de massa discreta, e a equação (4.8) é então resolvida em um esquema explícito que é condicionalmente estável.

A condição de estabilidade local segundo o critério de Courant-Friedrichs-Levy é dada por:

$$\Delta t_{ele} = CS \frac{L_{ele}}{a + \left(v_i \, v_i \right)^{1/2}} \tag{4.9}$$

onde *ele* indica um determinado elemento, L_{ele} é sua dimensão característica, a é a velocidade do som e *CS é um coeficiente de segurança* (o qual é sempre menor ou igual a 1,0).

Em velocidades transônicas e supersônicas, um amortecimento numérico adicional é necessário para capturar com precisão as ondas de choque e para suavizar oscilações locais na vizinhança das mesmas. O modelo de viscosidade artificial proposto por Argyris et al. (1990) foi adotado em função de sua simplicidade e eficiência em termos de tempo de processamento. O termo representando a viscosidade artificial é explicitamente adicionado à solução não suavizada:

$$\mathbf{U}_s^{n+1} = \mathbf{U}^{n+1} + \mathbf{M}_L^{-1} \mathrm{D} \tag{4.10}$$

onde $\mathbf{M}_L$ é a matriz de massa discreta, $\mathbf{U}_s^{n+1}$ e $\mathbf{U}^{n+1}$ são as soluções suavizada e não suavizada em $t + \Delta t$, respectivamente. O vetor D é dado por

$$\mathrm{D} = \sum_{\mathrm{ele}} \mathrm{CFL} \; \mathrm{CAF} \; S_{ele} \left[\mathbf{M} - \mathbf{M}_L \right]_{ele} \mathbf{U}_{ele}^n \tag{4.11}$$

onde *ele* é um índice de elementos, $\mathrm{CFL} = \Delta t / \Delta t_E$ é o número local de Courant-Friedrichs-Levy, CAF é um coeficiente de amortecimento artificial que deve ser especificado com cuidado para evitar interferência não desejada da viscosidade artificial sobre a viscosidade física, S_{ele} é um sensor de pressão em nível de elemento obtido como média dos valores nodais de S_i. Os valores de S_i são componentes do vetor global:

$$S_{\mathrm{k}} = \sum_{\mathrm{ele}} \frac{\left| \left(\mathbf{M} - \mathbf{M}_L \right)_{ele} \mathbf{p} \right|_k}{\left[\left| \mathbf{M} - \mathbf{M}_L \right|_{ele} \mathbf{p} \right]_k} \tag{4.12}$$

onde **p** é o vetor de pressão de um elemento específico, o símbolo $|\cdot|$ indica valor absoluto, **M** é a matriz de massa consistente em nível de elemento e *ele* refere-se aos elementos conectados ao nó *k*.

4.2.3 Técnica de integração multi-tempo usando sub-ciclos.

É comum o emprego de processos de refinamento de malha para que os altos gradientes das variáveis do problema possam ser adequadamente representados. Esse processos são aplicados localmente, somente nas regiões de altos gradientes, de modo a diminuir o impacto do tamanho da malha (em número de elementos e nós) sobre o tempo de processamento. Como conseqüência, as malhas resultantes podem apresentar elementos de tamanhos muito diferentes, especialmente em problemas tridimensionais nos quais a discretização é feita através de elementos tetraédricos.

A condição de estabilidade imposta pela equação (4.9) faz com que, em uma malha com elementos com tamanhos muito diferentes, o passo utilizado na integração no tempo esteja relacionado ao menor elemento da malha. Técnicas adaptativa para integração no tempo como a técnica de sub-ciclos proposta por Löhner et al. (1984) e empregada por Teixeira e Awruch (2001) e Bono (2008) podem ser utilizadas para aumentar consideravelmente a velocidade de processamento da simulação. O procedimento pode ser brevemente descrito pelos seguintes passos:

(I) O procedimento é iniciado determinando-se para cada elemento *k* o passo de tempo crítico (Δt_{Ek}) através da equação (4.9) e o valor mínimo para toda a malha Δt_{Emin} é determinado. Para cada elemento, é então atribuído um múltiplo inteiro n_{Ek}, correspondente à relação

$$n_{Ek} = \text{int}\left(\frac{\Delta t_{Ek}}{\Delta t_{E\min}}\right) \quad \text{, com } k = 1,...,nel \tag{4.13}$$

onde *nel* é o número total de elementos da malha.

(II) Depois de o passo de tempo local de cada elemento ter sido determinado, os elementos são agrupados em *g* grupos baseado no seu passo de tempo local

$$2^{(g-1)} \le n_{Ek} < 2^{g} \qquad \Delta t_g = 2^{(g-1)}\Delta t_{E\min} \qquad \Delta t_{Ek}^{g} = \Delta t_g \quad \text{, com } k = 1,...,nel \tag{4.14}$$

onde Δt_g é o passo de tempo do grupo *g* e Δt_{Ek}^{g} é o passo de tempo do elemento *k* pertencente ao grupo *g*.

O passo de tempo Δt_N para cada nó *N* é igualado ao menor passo de tempo de todos os elementos conectados a esse nó *N*. Dessa forma, o passo de tempo de cada nó é determinado por

$$\Delta t_N = \min_e \left(\Delta t_{Ee}^{g} \right) = 2^{g_N - 1} \Delta t_{E\min} , \qquad g_N = \min_e \left(g_e \right) \tag{4.15}$$

e pertence ao grupo nodal g_N o qual corresponde ao menor valor do grupo de elementos g_e de todos os elementos conectados ao nó *N*.

(III) Finalmente o passo de tempo Δt_{Ek}^{R} de cada elemento é reavaliado considerando o menor passo de tempo de todos os nós *m* pertencentes a esse elemento

$$\Delta t_{Ek}^{R} = \min_m \left(\Delta t_{Nm} \right) \tag{4.16}$$

Essa reavaliação leva cada elemento a um grupo g que é o menor grupo de todos os nós a ele conectados.

O método adaptativo para integração no tempo implementado tem três relógios: o primeiro está associado à marcha no tempo ou tempo corrente (*tempo*), o segundo está associado a cada grupo de nós (t_N) e o terceiro associado a cada grupo de elementos (t_G). Os dois últimos, quando comparados ao tempo corrente indicam quando um grupo de nós ou elementos está pronto para ser atualizado. O tempo corrente é atualizado com o passo de tempo mínimo Δt_{Emin} e cada grupo de nós ou elementos com seu próprio passo de tempo Δt_g. Cada relógio parte do mesmo valor inicial zero.

Quando o relógio do tempo corrente alcança o relógio de um grupo de nós ou de elementos, esses relógios são atualizados. Cada grupo de elementos cujo relógio é igual ao tempo corrente $t_G = tempo$ é atualizado. Depois que todos os grupos de elementos tenham sido atualizados, o laço de nós é executado. Cada grupo de nós cujo relógio é igual ao tempo corrente $t_N = tempo$ é atualizado. Os valores nodais das variáveis de todos os nós pertencentes a grupos cujo relógio $t_N > tempo$ são linearmente interpolados para obter valores nesse tempo, resultando em grande redução no esforço computacional e fazendo com que

todos os nós tenham valores para as variáveis nodais referentes a cada tempo do problema físico (*tempo*).

O passo de tempo de controle ou incremento de tempo mestre, considerado como sendo o maior passo de tempo de todos os grupos de elementos, determina quando um ciclo é completado. Nesse instante, todas as variáveis estão referenciadas ao mesmo tempo corrente, sem nenhuma interpolação.

Esse procedimento de atualização pode ser visualizado facilmente através do exemplo simplificado. Considere uma malha com 5 elementos unidimensionais em 3 grupos (g = 1, 2 e 3) com os passos de tempo determinados pela equação (5.9) como $1\Delta t$, $2\Delta t$, $2\Delta t$, $4\Delta t$ e $4\Delta t$. Aplicando os passos (I), (II) e (III) anteriormente descritos, os seguintes grupos de nós e de elementos são obtidos: $\Delta t_N = \{1, 1, 2, 2, 4, 4\}$ e $\Delta t_G = \{1, 1, 2, 2, 4\}$, respectivamente. O passo de tempo mestre do ciclo é $\Delta t_c = 4\Delta t$. Na figura 5.1, os nós são graficamente representados por diamantes, e os elementos por linhas conectando os respectivos nós. O tempo é representado pelo eixo vertical. O relógio dos grupos de nós é representado por círculos pretos, e círculos brancos são empregados para indicar os tempos nos quais uma interpolação linear foi utilizada para obter os valores das variáveis nodais.

No primeiro passo de integração no tempo, figura 4.1(a), o relógio do tempo corrente é t e os nós e elementos de todos os grupos são atualizados (é feita a integração no tempo) com seus respectivos passos de tempo. O relógio de cada grupo avança para diferentes posições no tempo. O relógio do tempo corrente avança para $t+\Delta t$ e todos os nós cujo grupo tem o relógio em uma posição à frente no tempo tem o valor das variáveis linearmente interpoladas para o valor corrente do tempo. No segundo passo de integração no tempo, figura 4.1(b), o tempo corrente é $t+\Delta t$, de modo que somente os grupos de nós e elementos cujo passo de tempo é Δt devem ser atualizados; para todos os demais nós, a interpolação é utilizada. Para atualizar o elemento 2, valores interpolados das variáveis para o nó 3 são necessários. No terceiro passo de integração, figura 4.1(c), o tempo corrente é $t+2\Delta t$, nós 1 a 5 e elementos 1 a 4 são atualizados. Como anteriormente, para atualizar o elemento 4 no grupo 2 valores interpolados das variáveis do nó 5 são necessários. O ciclo fica completo com o quarto e último sub-ciclo, figura 4.1(d). Somente os nós e elementos do grupo 1 são atualizados.

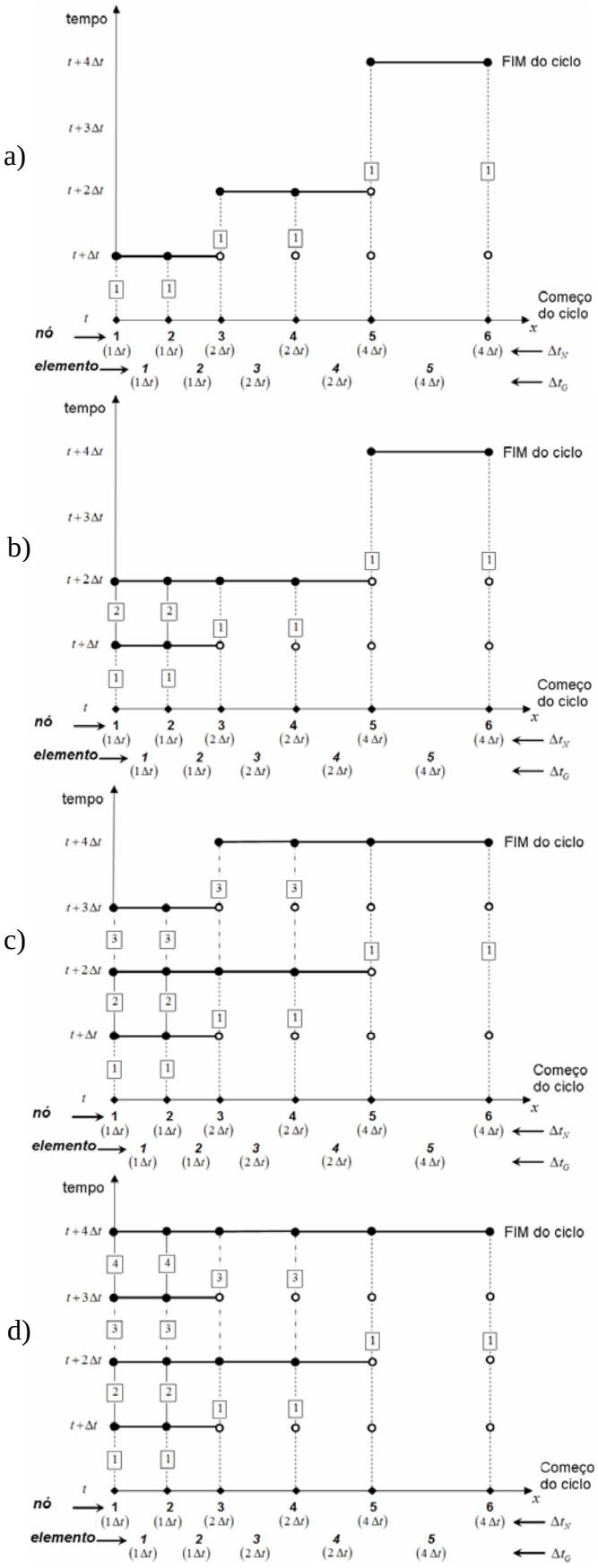


Figura 4.1: Técnica de sub-ciclos aplicada na integração no tempo.

O ganho no tempo de processamento (*speed-up*) usando sub-ciclos pode ser calculado através da expressão apresentada por Belytschko e Gilbertsen (1992)

$$speed-up = \frac{t^{nsc}}{t^{sc}} = \frac{NSC}{\sum_{k=1}^{NSC} PESC_k / 100} \tag{4.17}$$

onde t^{nsc} e t^{sc} são os tempos necessários para a solução com um passo de tempo global único (sem sub-ciclos) e usando múltiplos passos de tempo (com sub-ciclos), respectivamente, *NSC* é o número de sub-ciclos e *PESC*.é o percentual de elementos atualizado no subciclo *k*. No exemplo utilizado, são necessários 4 sub-ciclos para completar um ciclo (*NSC = 4*) e o percentuais de elementos nos grupos *g* = 1, 2 e 3 são 40%, 40% e 20%, respectivamente, resultando em um ganho de 1,54 no tempo de processamento (o processamento usando sub-ciclos é 1,54 vezes mais rápido que com um único passo de tempo global). O valor previsto pela equação (4.17) deve ser tomado como um valor teórico, uma vez que considera que todo o tempo de processamento está associado a laços de elementos, enquanto que numa implementação real há esforço computacional associado a laços de nós, interpolações e estruturas de controle. O valor de *speed-up* obtido é significativamente menor que o valor teórico.

4.2.4 A implementação paralela para o esquema de um passo com iterações

A equação (4.8) corresponde a um conjunto de equações relativas às variáveis nodais ρ, v_i e e, que são desacopladas dentro de cada iteração de cada passo de tempo, tendo o formato genérico da equação (2.1). Assim, se o valor dessas variáveis é conhecido em um tempo t, o novo valor de qualquer variável nodal, na primeira iteração do tempo $t+\Delta t$ pode ser avaliado de forma independente, e de forma similar nas iterações seguintes em função dos valores na iteração imediatamente anterior.

Para cada equação nodal, a informação necessária do passo de tempo anterior ou da iteração anterior é somente o valor das variáveis nos nós pertencentes aos elementos conectados ao nó no qual a equação nodal está sendo avaliada, tornando o conjunto representado pela equação (4.8) apto a ser paralelizado de forma eficiente para máquinas de memória distribuída.

No início do processo de integração no tempo, as equações nodais são divididas entre os diversos processadores lógicos presentes, conforme descrito no capítulo 2. Cada processador *i* precisa como informação inicial do valor das variáveis relativas a todos os nós N_i^+ dos elementos conectados ao grupo de nós alocado a esse processador. Ao longo de cada iteração de cada passo de tempo, os laços de elementos da implementação do algoritmo de solução resultante da equação (4.8) são tomados sobre o conjunto de elementos E_i, que inclui os elementos E_i^P conectados exclusivamente ao conjunto de nós N_i do processador e os *elementos comuns* E_i^C conectados tanto a nós do grupo do processador *i* quanto a nós alocados a outros processadores. Da mesma forma, todos os laços referentes a variáveis nodais que serão utilizadas pelos elementos de E_i são feitos sobre o conjunto N_i^+.

Ao final de cada iteração de cada passo de tempo, cada processador lógico *i* envia para um outro processador lógico *j* o valor de ρ, v_i e e referentes aos nós do conjunto ΔN_{ij} e recebe deste o valor das variáveis do conjunto ΔN_{ji} , repetindo o procedimento com tantos processadores quanto o processo de divisão de tarefas houver definido.

O uso de um modelo de viscosidade artificial, especificamente no que se refere ao cálculo dos valores nodais do sensor de pressão S_i na equação (4.12) e da média dos valores nodais do elemento S_{ele}, usado na equação (4.11) exige uma comunicação extra entre processadores. Cada processador *i* consegue calcular a partir dos valores nodais de pressão p_k do conjunto N_i^+ apenas os valores nodais de S_k do conjunto N_i. Contudo, para o cálculo do valor médio do sensor de pressão S_{ele} dos elementos do conjunto E_i , utilizados em (4.11), os valores nodais de S_k do conjunto N_i^+ são necessários.

Desta forma, o esquema explícito de Taylor-Galerkin empregado necessita, em sua forma paralela, para cada passo de tempo, de um processo de comunicação entre os processadores lógicos para a avaliação da viscosidade artificial e, para cada iteração de cada passo de tempo, de um processo de comunicação entre processadores para a atualização das variáveis nodais ρ, v_i e e. O algoritmo está mostrado na tabela 4.4.

Tabela 4.4: Algoritmo do esquema explícito de Taylor-Galerkin de um passo com iterações, paralelização otimizada para configurações de memória distribuída.

Etapa 1. Inicialização

(a) $n = 0 : t = 0$

(b) $\mathbf{U}^n = valores\ iniciais$

(c) $$\Delta t_{ele} = CS \frac{L_{ele}}{a + \left(v_i\, v_i \right)^{1/2}}$$

(d) $$\Delta t = \min\left(\Delta t_{ele} \right)$$

Etapa 2. Passo de tempo

(e) $$S_{\mathrm{k}} = \sum_{\mathrm{ele}} \frac{\left| \left(\mathbf{M} - \mathbf{M}_L \right)_{ele} \mathbf{p} \right|_k}{\left[\left| \mathbf{M} - \mathbf{M}_L \right|_{ele} \mathbf{p} \right]_k}$$ para N_i

(f) Envia S_k para processador lógico j para ΔN_{ij}

(g) Recebe S_k do processador lógico j para ΔN_{ji}

(h) Repete (f) e (g) para todos os processadores lógicos *j* com os quais *i* troca dados

(i) $S_{ele} = media(S_k) \quad nós\ k\ do\ elemento\ ele$ para E_i

(j) $$\mathrm{D} = \sum_{\mathrm{ele}} \mathrm{CFL\ CAF}\ S_{ele} \left[\mathbf{M} - \mathbf{M}_L \right]_{ele} \mathbf{U}^n_{ele}$$ para E_i

(k) $J = 0$

Etapa 3. Iteração

(l) $$\mathbf{U}^{n+1}_{J+1} = \mathbf{U}^n + \Delta t \left[-\frac{\partial \mathbf{F}_i}{\partial x_i}^n - \frac{\partial \mathbf{G}_i}{\partial x_i}^n + \frac{\Delta t}{2} \frac{\partial}{\partial x_k} \left(\mathbf{A}_k{}^n \frac{\partial \mathbf{F}_i}{\partial x_i}^n \right) \right] + \frac{\Delta t}{2} \left[-\frac{\partial \Delta \mathbf{F}_{iJ}}{\partial x_i}^{n+1} - \frac{\partial \Delta \mathbf{G}_{iJ}}{\partial x_i}^{n+1} + \frac{\Delta t}{2} \frac{\partial}{\partial x_k} \left(\mathbf{A}_k{}^n \frac{\partial \Delta \mathbf{F}_{iJ}}{\partial x_i}^{n+1} \right) \right] + \mathbf{M}_L{}^{-1} \mathrm{D}$$ para N_i

(m) Aplica as condições de contorno

Tabela 4.4 (*continuação*): Algoritmo do esquema explícito de Taylor-Galerkin de um passo com iterações, paralelização otimizada para configurações de memória distribuída.

(n)	Envia ρ, v_i e e para processador lógico j	para ΔN_{ij}
(o)	Recebe ρ, v_i e e do processador lógico j	para ΔN_{ji}
(p)	Repete (n) e (o) para todos os processadores lógicos *j* com os quais *i* troca dados	
(q)	Se a convergência das iterações não é satisfeita, *J= J+1, retorna ao passo (m)*	
Etapa 3. Atualização do tempo físico de análise		
(r)	$n = n+1 : t = t + \Delta t$	
(s)	Retorna ao passo (e)	

Uma vez que, em geral, os valores da variáveis são armazenados apenas para alguns valores de *t* (espaçados por um número muito grande de passos de tempo), não há necessidade de um processador lógico mestre, com cada processador trocando os dados necessários a cada iteração de cada passo de tempo diretamente com os demais processadores envolvidos de forma autônoma.

O uso da técnica de sub-ciclos não interfere na forma como o algoritmo foi paralelizado, pois, ao final de cada passo do relógio de tempo corrente, sempre estão disponíveis valores das variáveis nodais a serem comunicados, tenham eles sido calculados ou interpolados por um dado processador lógico.

O mesmo não pode ser dito a respeito da divisão de tarefas baseada em nós, que considera a premissa de que todos os nós exigem igual esforço computacional (e tem a eles conectado igual número de elementos). Com o uso da técnica de sub-ciclos, nós pertencentes a grupos de maior passo de tempo (conectados a elementos que também pertencem a grupos de maior passo de tempo) tem a eles associado um esforço computacional consideravelmente menor que os nós e elementos pertencentes a grupos de menor passo de tempo. Para nós e elementos do grupo 1Δt, todos os passos de tempo do relógio de tempo corrente exigem que o valor das variáveis nodais seja calculado, ao passo que para o grupo 256Δt, a cada 256 passos de tempo

do relógio do tempo corrente, somente um exige que o valor das variáveis nodais sejam calculadas, sendo nos demais 255 interpoladas.

Se a técnica de sub-ciclos for empregada para a solução de problemas estacionários (Bono, 2008), a divisão dos nós e elementos em grupos com diferentes passos de tempo é feita previamente ao início da integração no tempo, de modo que para cada ciclo completo de passos de tempo, o esforço computacional associado a cada elemento ou nó é constante ao longo dos ciclos. Desta forma, embora o método de divisão de tarefas possa alocar a cada processador cargas de trabalho consideravelmente diferentes, o processo de balanceamento permite uma redistribuição desses esforços. Se um dado processador lógico do conjunto tem um grupo de nós associado a elementos muito grandes (e, portanto, com passos de tempo grandes), o processamento desses nós e elementos será muito rápido, pois na maioria dos passos do relógio de tempo corrente somente são feitas interpolações, de modo que sua *velocidade relativa* em relação a outro processador cujo grupo de nós está associado a elementos pequenos (e, conseqüentemente, com pequenos passos de tempos, poucas interpolações, cálculo das variáveis nodais na maioria dos passos do relógio de tempo corrente) será maior. O processo de balanceamento irá retirar nós (e elementos) dos processadores mais lentos (de menor velocidade relativa) e alocá-los aos processadores mais rápidos (de maior velocidade relativa), até que haja equalização. Entenda-se *velocidade relativa* como a relação inversa entre os tempos que cada processador leva para processar o lote de dados a ele alocados, sem relação direta com a velocidade real (*clock)* do processador. Nessas condições, não há necessidade de um processador que coordene o fluxo de dados entre os processadores a não ser para consolidar os resultados em intervalos de tempo para os quais se queira o registro da solução obtida.

Se a técnica de sub-ciclos for empregada em problemas transientes, uma avaliação prévia ao início do processo de integração no tempo dos passos de tempo mínimos dos elementos não é suficiente. À medida que o escoamento vai se desenvolvendo, diferentes regiões da malha ficam sujeitas a altos gradientes das variáveis de campo, ou mesmo a campos de velocidades muito baixas ou quase estacionárias, exigindo que o passo de tempo empregado nessas regiões seja pequeno para manter a precisão do processo de simulação. Dessa forma, os nós e elementos da malha mudariam dinamicamente de grupos de sub-ciclos ao longo da análise, exigindo que também a redistribuição de tarefas fosse refeita em determinados pontos do avanço temporal da simulação. O uso de uma programação do tipo mestre-escravo se faz

necessária para, sempre que mudar a distribuição de tarefas, reunir os valores das variáveis nodais no processador mestre e redistribuí-los pelos processadores escravos. Esse procedimento não foi implementado neste trabalho.

4.3 ESCOAMENTOS INCOMPRESSÍVEIS EM REGIME SUBSÔNICO

Para simulação de escoamentos incompressíveis em regime subsônico foi utilizado o algoritmo empregado por Braun (2007) e Bono (2008) onde, para integração no tempo, é utilizado um esquema de dois passos.

4.3.1 A formulação de Taylor-Galerkin com esquema de dois passos.

O esquema explicito de dois passos de Taylor-Galerkin utilizado para integração no tempo foi proposto por Kawahara e Hirano (1983) empregado para problemas incompressíveis com a hipótese de pseudo-compressibilidade. Expandindo as variáveis de conservação **U** em $t = t^{n+1/2} = t^n + \Delta t/2$ através da série de Taylor incluindo a primeira e a segunda derivadas, é obtida a expressão

$$\Delta \mathbf{U}^{n+1/2} = \frac{\Delta t}{2}\frac{\partial}{\partial t}\left(\mathbf{U}^n + \frac{\Delta t}{4}\frac{\partial \Delta \mathbf{U}^n}{\partial t} \right) \tag{4.18}$$

onde $\Delta \mathbf{U}^{n+1/2} = \mathbf{U}^{n+1/2} - \mathbf{U}^n$, $\Delta t = t^{n+1} - t^n$ é o passo de tempo, *n* e *n+1/2* indicam *t* e *t+Δt/2*, respectivamente. Substituindo a equação (4.3) e sua derivada na equação (4.18) e desprezando os termos de ordem superior, é obtida a expressão

$$\mathbf{U}^{n+1/2} = \mathbf{U}^n + \frac{\Delta t}{2}\left[-\frac{\partial \mathbf{F}_i{}^n}{\partial x_i} - \frac{\partial \mathbf{G}_i{}^n}{\partial x_i} + \frac{\Delta t}{4}\frac{\partial}{\partial x_k}\left(\mathbf{A}_k{}^n \frac{\partial \mathbf{F}_i{}^n}{\partial x_i} \right) \right] \tag{4.19}$$

O campo das velocidades deve ser corrigido a partir do campo de pressões obtido em *t+Δt/2* através da expressão

$$v_i^{n+1/2} = v_i^{n+1/2} - \frac{1}{\rho}\frac{\Delta t^2}{8}\frac{\partial \Delta p^{n+1/2}}{\partial x_j}\delta_{ij} \tag{4.20}$$

Para obter os valores das variáveis de conservação **U** em *n+1* pode-se aplicar uma expansão em séries de Taylor, desprezando os termos de ordem superior, da forma

$$\Delta\mathbf{U}^{n+1} = \mathbf{U}^{n} + \Delta t \frac{\partial}{\partial t}\left(\mathbf{U}^{n} + \frac{\Delta t}{2}\frac{\partial \mathbf{U}^{n}}{\partial t} \right) \tag{4.21}$$

Substituindo a equação (4.3) e sua derivada temporal avaliadas em $t+\Delta t/2$ na equação (4.21), obtém-se:

$$\mathbf{U}^{n+1} = \mathbf{U}^{n+1/2} + \Delta t\left[-\frac{\partial \mathbf{F}_i}{\partial x_i}^{n+1/2} - \frac{\partial \mathbf{G}_i}{\partial x_i}^{n+1/2} + \frac{\Delta t}{2}\frac{\partial}{\partial x_k}\left(\mathbf{A}_k^{\,n+1/2}\frac{\partial \mathbf{F}_i}{\partial x_i}^{n+1/2} \right) \right] \tag{4.22}$$

onde *n+1* e *n+1/2* indicam $t+\Delta t$ e $t+\Delta t/2$, respectivamente.

4.3.2 A implementação paralela para o esquema de dois passos

As equações (4.19) e (4.22) correspondem a um conjunto de equações relativas às variáveis nodais ρ, v_i e e, que são desacopladas dentro de cada passo de tempo, tendo o formato genérico da equação (2.1). Assim, se o valor dessas variáveis é conhecido em um tempo t, o novo valor de qualquer variável nodal no tempo $t+\Delta t/2$ pode ser avaliado de forma independente, e de forma similar em $t+\Delta t$ em função dos valores em $t+\Delta t/2$.

Para cada equação nodal, a informação necessária do passo de tempo anterior é somente o valor das variáveis nos nós pertencentes aos elementos conectados ao nó no qual a equação nodal está sendo avaliada, tornando o conjunto representado pelas equações (4.19) e (4.22) apto a ser paralelizado de forma muito similar à que foi feita para os escoamentos compressíveis em regime transônico e subsônico. Como o escoamento é incompressível, o valor da massa específica ρ é constante, diminuindo de 5 para 4 o número de variáveis nodais a serem comunicadas entre processadores lógicos. A implementação foi feita substituindo-se a energia total específica e pela pressão termodinâmica p como variável nodal básica, sendo a energia total específica e a temperatura calculadas em função das demais.

Uma observação deve ser feita com relação à equação (4.20) que, sendo implementada elemento-por-elemento, permite ao processador lógico i, a partir das pressões do conjunto nodal N_i^+, calcular o valor corrigido da velocidade somente para o conjunto nodal N_i. Como

os valores da pressão e da velocidade são necessários no conjunto nodal N_i^+ em $t+\Delta t/2$ para permitir o cálculo das variáveis em $t+\Delta t$, o processo de comunicação das variáveis entre os processadores em $t+\Delta t/2$ deve ser dividido em duas etapas: primeiro há a comunicação do valor da pressão, o que permite a cada processador corrigir os valores da velocidade nos seus correspondentes grupos de nós, e então é feita a comunicação entre processadores do valor da velocidade.

Desta forma, o esquema explícito de Taylor-Galerkin empregado necessita, em sua forma paralela, para o primeiro meio passo de tempo $\Delta t/2$, de dois processos de comunicação entre os processadores lógicos para a atualização das variáveis nodais v_i e p, e para o segundo meio passo, um processo de comunicação. O algoritmo está mostrado na tabela 4.5.

Ao longo de cada passo de tempo, os laços de elementos utilizados na implementação do algoritmo de solução resultante das equações (4.19) e (4.22) são tomados sobre o conjunto de elementos E_i, que inclui os elementos E_i^P conectados exclusivamente ao conjunto de nós N_i do processador e os *elementos comuns* E_i^C conectados tanto a nós do grupo do processador *i* quanto a nós alocados a outros processadores. Da mesma forma, todos os laços referentes a variáveis nodais que serão utilizadas pelos elementos de E_i são feitos sobre o conjunto N_i^+.

Tabela 4.5: Algoritmo do esquema explícito de Taylor-Galerkin de dois passos sem iterações, paralelização otimizada para configurações de memória distribuída.

Etapa 1. Inicialização

(a) $n = 0 : t = 0$

(b) $\mathbf{U}^n = \textit{valores iniciais}$

(c) $$\Delta t_{ele} = CS \frac{L_{ele}}{a + \left(v_i\, v_i\right)^{1/2}}$$

(d) $$\Delta t = \min\left(\Delta t_{ele}\right)$$

Tabela 4.5 (*continuação*): Algoritmo do esquema explícito de Taylor-Galerkin de dois passos sem iterações, paralelização otimizada para configurações de memória distribuída.

Etapa 2. Passo de tempo $t = t + \Delta t/2$

(e) $\mathbf{U}^{n+1/2} = \mathbf{U}^{n} + \frac{\Delta t}{2}\left[-\frac{\partial \mathbf{F}_i}{\partial x_i}^{n} - \frac{\partial \mathbf{G}_i}{\partial x_i}^{n} + \frac{\Delta t}{4}\frac{\partial}{\partial x_k}\left(\mathbf{A}_k{}^{n} \frac{\partial \mathbf{F}_i}{\partial x_i}^{n} \right) \right]$ para N_i

(f) Aplica as condições de contorno

(g) Envia p para processador lógico j para ΔN_{ij}

(h) Recebe p do processador lógico j para ΔN_{ji}

(i) Repete (g) e (h) para todos os processadores lógicos j com os quais i troca dados

(j) $v_i^{n+1/2} = v_i^{n+1/2} - \frac{1}{\rho}\frac{\Delta t^2}{8}\frac{\partial \Delta p^{n+1/2}}{\partial x_j}\delta_{ij}$ para N_i

(k) Envia v_i para processador lógico j para ΔN_{ij}

(l) Recebe v_i do processador lógico j para ΔN_{ji}

(m) Repete (k) e (l) para todos os processadores lógicos j com os quais i troca dados

Etapa 3. . Passo de tempo $t = t + \Delta t$

(n) $\mathbf{U}^{n+1} = \mathbf{U}^{n+1/2} + \Delta t\left[-\frac{\partial \mathbf{F}_i}{\partial x_i}^{n+1/2} - \frac{\partial \mathbf{G}_i}{\partial x_i}^{n+1/2} + \frac{\Delta t}{2}\frac{\partial}{\partial x_k}\left(\mathbf{A}_k{}^{n+1/2} \frac{\partial \mathbf{F}_i}{\partial x_i}^{n+1/2} \right) \right]$ para N_i

(o) Aplica as condições de contorno

(p) Envia v_i e p para processador lógico j para ΔN_{ij}

(q) Recebe v_i e p do processador lógico j para ΔN_{ji}

(r) Repete (p) e (q) para todos os processadores lógicos j com os quais i troca dados

Tabela 4.5 (*continuação*): Algoritmo do esquema explícito de Taylor-Galerkin de dois passos sem iterações, paralelização otimizada para configurações de memória distribuída.

Etapa 4. Atualização do tempo físico de análise	
(s)	$n = n+1 : t = t + \Delta t$
(t)	Retorna ao passo (e)

Quanto à necessidade de um processador lógico mestre, valem os mesmos comentários feitos a respeito do esquema de um passo com iterações.

4.4 INTERAÇÃO FLUÍDO ESTRUTURA

Para simulação de problemas de Interação Fluído-Estrutura, foi utilizada a formulação arbitrária Lagrangeana-Euleriana (ALE) com acoplamento particionado entre o fluído e a estrutura e interfaces não coincidentes empregada por Braun (2007).

4.4.1 A análise conjunta do fluído e da estrutura.

Problemas de Dinâmica dos Fluídos tradicionalmente utilizam uma descrição Euleriana, na qual a malha é tratada como um referencial fixo através do qual o fluído escoa. Por outro lado, na Mecânica dos Sólidos, é utilizada a descrição Lagrangeana, na qual a malha move-se juntamente com os deslocamentos da matéria.

Para permitir que os movimentos apresentados pela estrutura ao vibrar sob a ação do escoamento sejam percebidos e computados adequadamente na análise do fluído, pode-se utilizar uma formulação ALE, cuja idéia básica está na introdução de um domínio de referência que se move arbitrariamente e de forma independente aos pontos espaciais e materiais, sendo que as equações da Mecânica do Contínuo passam a ser descritas a partir de pontos fixos definidos no domínio de referência (Braun, 2007).

Na implementação utilizada, a discretização espacial empregada para a simulação do escoamento é dividida em dois sub-domínios arbitrários: um sub-domínio puramente Euleriano (no qual os nós da malha utilizada na discretização permanecem fixos); e um sub-

domínio ALE (no qual as coordenadas dos nós da malha sofrem alterações ao longo do tempo) que engloba a estrutura imersa no escoamento. O sub-domínio ALE apresenta duas regiões de contorno, uma, denominada cotorno ALE, que faz interface com o sub-domínio Euleriano, de modo que os nós nela contidos permanecem fixos, e a outra, denominada de contorno sólido, que faz interface com a estrutura, e cujos nós reproduzem, conseqüentemente, os deslocamentos e a velocidade da estrutura a cada instante. As velocidades dos nós da malha pertencentes ao sub-domínio ALE e internos aos contornos são calculados a partir dos valores de velocidade da malha nas duas regiões de contorno, interpolando-se em função de uma potência do inverso da distância de cada nó aos contornos. As coordenadas dos nós da malha a cada instante são calculadas a partir das velocidades. Para o cálculo da velocidade do fluído em cada ponto do domínio, a velocidade da malha deve ser considerada.

A integração no tempo é feita utilizando-se o esquema explícito de dois tempos descrito na seção 4.3, com a condição de estabilidade dada pela equação (4.9) em função do tamanho dos elementos empregados na discretização espacial.

Para o acoplamento entre o fluído e a estrutura utiliza-se o tratamento particionado. A estrutura é tratada computacionalmente como uma entidade isolada, com uma malha independente que pode, inclusive, ser não coincidente na interface (contorno sólido) com a malha do fluído. Essa característica é extremamemente interessante porque permite o uso de uma malha bastante refinada para o fluído na região de interface, sem acarretar um esforço computacional extra na discretização da estrutura desnecessariamente.

A estrutura é considerada como um corpo deformável constituído de um material elástico linear com a presença de não linearidade geométrica. A equação de equilíbrio dinâmico é integrada no tempo utilizando-se o método implícito de Newmark no contexto do método de estabilização α-Generalizado, dando origem a um processo iterativo cujos sistemas de equações resultantes são resolvidos utilizando-se o Método dos Gradientes Conjugados. Maiores detalhes podem ser vistos em Braun (2007). A condição de estabilidade do método de Newmark empregado leva ao uso de um passo de tempo em geral maior que o passo de tempo necessário para a análise do fluído, de forma que a marcha no tempo da solução é feita através de sub-ciclos, com a estrutura avançando com o passo de tempo Δt e o fluído com passo $\Delta t/n$, sendo n um inteiro.

4.4.2 A implementação paralela para IFE

Em função da diferença entre os passos utilizados na marcha no tempo entre o fluído e a estrutura, a implementação é feita com a característica de sub-ciclos, onde um determinado número de passos é feito na integração no tempo para o fluído para cada passo na integração no tempo para a estrutura. Como os domínios do fluído e da estrutura não são resolvidos simultaneamente, mas alternadamente, e as malhas relativas a cada domínio são diferentes e independentes, com características geométricas próprias (formato geométrico, número de nós, número e tamanho dos elementos, etc), cada malha precisa ser dividida de forma independente entre os processadores lógicos presentes para garantir eficiência de paralelização.

Uma vez que as malhas para o fluído e para a estrutura são independentes, com distribuições de tarefas (particionamento de malhas) independentes, a transferência de dados do fluído para a estrutura e vice-versa de um processador diretamente para outro torna-se bastante complexa. Portanto, faz-se necessário um modelo de programação paralela do tipo mestre-escravo para coordenar a transferência de dados entre os diversos processadores ao se passar da análise do fluído para a estrutura , e desta novamente para o fluído, como pode ser visto na Tabela 4.6.

Tabela 4.6: Algoritmo de solução para problemas de interação fluído-estrutura, paralelização otimizada para configurações de memória distribuída.

Etapa 1. Inicialização	
(a) $t_{estrut} = 0 \quad : \quad t_{fluído} = 0$	
(b) $\Delta t_{estrut} = m.\Delta t_{fluído} \quad m = \text{int}$	
(c) Aplicação das condições iniciais para o sólido e para ao fluído	
(d) $n = 0$	
(e) $\dot{\mathbf{x}}_n = \dot{\mathbf{u}}_n$ no contorno sólido	mestre e escravos
(f) interporlar $\dot{\mathbf{x}}_n$ no sub-domínio ALE	mestre e escravos

Tabela 4.6 (*continuação*): Algoritmo de solução para problemas de interação fluído-estrutura, paralelização otimizada para configurações de memória distribuída.

Etapa 2. Integração no tempo do Fluído		
(g)	$\mathbf{x}_{n+1} = \mathbf{x}_n + \frac{n\Delta t_{fluido}}{\Delta t_{estrut}} \dot{\mathbf{x}}_n$ no sub-domínio ALE	mestre e escravos
(h)	Taylor-Galerkin, esquema de dois passos	mestre e escravos
(i)	$t_{fluido} = t_{fluido} + \Delta t_{fluido} \ : \ n = n+1$	
(j)	Caso $n < m$ retorna ao passo (g)	
Etapa 3. Transferência de dados do Fluído para Estrutura		
(k)	Comunicação das pressões no contorno sólido	escravos para mestre
(l)	Pressão no fluído transferida para cargas na estrutura	mestre
(m)	Comunicação das cargas na estrutura	mestre para escravos
Etapa 4. Integração no tempo da Estrutura		
(n)	Newmark no contexo α-Generalizado	mestre e escravos
(o)	$t_{estrut} = t_{estrutu} + \Delta t_{estrut}$	
Etapa 5. Transferência de dados da Estrutura para o Fluído		
(p)	Consolidação das velocidades $\dot{\mathbf{u}}_n$ do contorno sólido	escravos para mestre
(q)	Comunicação de $\dot{\mathbf{u}}_n$ de todo o contorno sólido	mestre para escravos
(r)	$\dot{\mathbf{x}}_n = \dot{\mathbf{u}}$ no contorno sólido	mestre e escravos
(s)	interporlar $\dot{\mathbf{x}}_n$ no sub-domínio ALE	mestre e escravos
(t)	$n = 0$	
(u)	retorna à Etapa 2	

Na tabela 4.6, $\dot{\mathbf{x}}_n$ são as coordenadas dos nós da malha de discretização espacial do fluído, e $\dot{\mathbf{u}}_n$ as velocidades nodais obtidos para a estrutura, sendo a região de interface Estrutura-Fluído chamada de contorno sólido indiscriminadamente para as duas malhas.

A paralelização da etapa (h) está descrita em detalhes na seção 4.3.2. A etapa (n) consiste em um processo iterativo que consiste numa série de etapas baseadas em laços de elementos (e portanto facilmente paralelizáveis) e da solução de um sistema de equações, feito através do Método dos Gradientes Conjugados cuja paralelização está detalhada na seção 4.1.

Quando a malha utilizada para a discretização da estrutura é muito pequena (poucos nós e elementos), pode ser vantajoso que a etapa da estrutura, passos (n) e (o), seja processada somente na máquina mestre. Nesse caso, os passos (m) e (p) não são realizados.

5 RESULTADOS OBTIDOS PARA CONFIGURAÇÕES DE MEMÓRIA DISTRIBUÍDA

As técnicas para a divisão de tarefas apresentadas no Capítulo 2 e de ordenamento da comunicação de dados entre processadores lógicos apresentadas no Capítulo 3 foram aplicadas para os algoritmos paralelos apresentados no capítulo 4.

Os códigos resultantes foram implementados em *FORTRAN 90* utilizando-se os compiladores Compaq Visual Fortran versão 6.6c e Intel Visual Fortran 9.1. Para comunicação de dados entre os processadores foi utilizada a biblioteca MPICH versão 1.2.5, uma versão de MPI desenvolvida pelo Argonne National Laboratory (2004), Estados Unidos. O sistema operacional utilizado foi o Windows XP Professional.

Antes de serem paralelizados, todos os códigos foram otimizados segundo os critérios contidos no Anexo A, a fim de se obter a maior velocidade de processamento possível e evidenciar possíveis problemas de saturação na comunicação entre computadores em função da taxa de transmissão e latência da rede de comunicação empregada.

A tabela 5.1 resume os testes efetuados com diversas configurações, indicando o tipo de problema (Problema), o número de computadores / processadores empregado nos testes (*CPUs*), a arquitetura e freqüência do núcleo dos processadores empregados, em GHz (*Classe*), o tipo de elemento finito utilizado na discretização espacial das malhas (*Malha*), o tamanho dos problemas testados em milhares de graus de liberdade (*GL x1000*), a taxa de transferência da rede Ethernet de comunicação entre computadores (*Rede*) e os tipos de tratamentos utilizados nas malhas para a divisão de tarefas (*Trat.*).

Tabela 5.1: Descrição dos testes de paralelismo realizados para configurações de memória distribuída.

Problema	CPUs	Classe	Malha	GL x1000	Rede	Trat.
Flexão de placa elástica linear Solver de Gradientes Conjugados	8	Athlon XP 1,3 a 2,1	Hexaedros	43 a 2120	100M	1RN
Escoamento compressível em torno de esfera	6	Athlon XP 1,3 a 1,5	Hexaedros	35 a 2826	100M	1RN
Escoamento compressível em torno de esfera	8	Athlon 64 2,3 a 2,5	Hexaedros	35 a 1193	100M	1RN
Escoamento compressível em torno de esfera	11	Athlon 64 2,0 a 2,2	Hexaedros	35 a 5129	1G	1RN
Escoamento compressível em torno de veículo espacial	20	Athlon 64 2,0 a 2,2	Hexaedros	1056 e 4746	1G	1RN nRN
Escoamento compressível em torno de esfera	20	Athlon64 2,0 a 2,2	Tetraedros	305	1G	1RN nRN
Escoamento compressível em torno de esfera	8	Athlon 64 2,0 a 2,2	Tetraedros	305	100M	1RN nRN
Escoamento compressível em torno de avião	20	Athlon 64 2,0 a 2,2	Tetraedros	229 a 8488	1G	1RN nRN
Escoamento compressível em torno de avião	8	Athlon 64 2,0 a 2,2	Tetraedros	1359	100M	1RN nRN
Escoamento compressível em torno de asa Uso de subciclos	20	Athlon 64 2,0 a 2,2	Tetraedros	554	1G	1RN
Escoamento incompressível em cavidade	20	Athlon 64 2,0 a 2,2	Hexaedros	21	1G	1RN nRN
Escoamento incompressível bidimensional em torno de edificação	20	Athlon 64 2,0 a 2,2	Hexaedros	182	1G	1RN nRN
Escoamento incompressível tridimensional em torno de edificação rígida	20	Athlon 64 2,0 a 2,2	Hexaedros	1534	1G	1RN nRN
Interação Fluído Estrutura Escoamento incompressível em cavidade com fundo flexível	20	Athlon 64 2,0 a 2,2	Hexaedros	21	1G	1RN nRN
Interação Fluído Estrutura Escoamento incompressível bidimensional em torno de edificação com teto flexível	20	Athlon 64 2,0 a 2,2	Hexaedros	182	1G	1RN nRN
Interação Fluído Estrutura Escoamento incompressível em torno de edificação tridimensional flexível	20	Athlon 64 2,0 a 2,2	Hexaedros	1534	1G	1RN nRN

5.1 MÉTODO DOS GRADIENTES CONJUGADOS

Para teste do algoritmo paralelo de solução de sistemas de equações utilizando o Método dos Gradientes Conjugados, o mesmo foi implementado em programa de análise estática elástica linear de sólidos tridimensionais usando elementos hexaédricos lineares de 8 nós com integração reduzida, livre de travamento volumétrico e travamento de cisalhamento e que não apresenta modos espúrios.

Na formulação do elemento, foi utilizado apenas um ponto de integração, de forma que a matriz de rigidez é calculada de forma explícita e o tempo computacional necessário para o seu processamento é bastante reduzido. Mais detalhes podem ser encontrados em Duarte Filho (2002).

Os testes de avaliação de desempenho do *solver* foram feitos para o problema de uma placa quadrada espessa, com lado igual a 4 unidades de comprimento e espessura igual a 0,8 unidades, submetida a uma carga de 10 unidades de força centrada na face superior, e apoiada em todo o contorno da face inferior, conforme figura 5.1. O módulo de elasticidade do material adotado foi de 10^7 unidades de força por unidade de superfície, com um coeficiente de Poisson de 0,3.

A fim de verificar a evolução do desempenho do *solver* paralelo usando Gradientes Conjugados com o aumento do tamanho do sistema de equações, diversas malhas foram utilizadas sobre a mesma geometria, com número crescente de elementos e nós, mantendo aproximadamente constante a razão de aspecto dos elementos de 1:1:1 e garantindo a existência de um nó central onde a carga é aplicada. O uso de elementos com tal razão de aspecto em um problema de placa espessa dá origem a sistemas extremamente bem condicionados e permite o máximo desempenho do *solver* de gradientes conjugados quanto ao número de iterações necessárias para ser alcançada a convergência. Em todos os testes a tolerância usada para determinar a convergência do Método dos Gradientes Conjugados foi de 10^{-6}, tanto com uso do pré-condicionador Diagonal quanto com o pré-condicionador de Cholesky.

A menor das malhas utilizadas está representada na figura 5.1 com 10x10x2 elementos. A família de malhas utilizada e suas características estão mostradas na tabela 5.2.

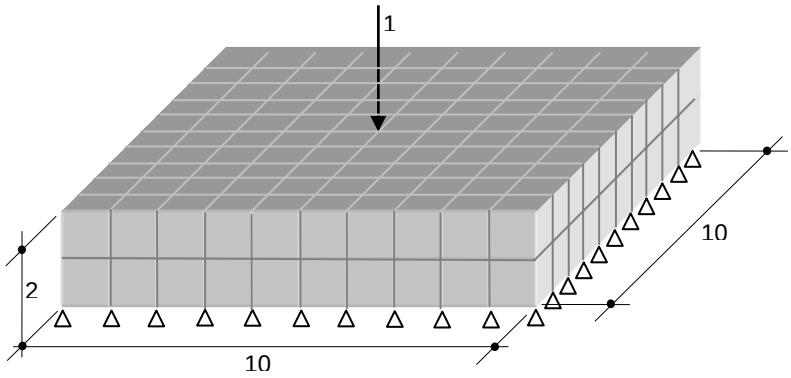


Figura 5.1: Placa quadrada espessa com carga centrada utilizada nos testes de desempenho do *solver* paralelo usando Gradientes Conjugados

Tabela 5.2: Família de malhas utilizada no teste de desempenho do *solver* paralelo usando Gradientes Conjugados

Malha	**Dimensões em elementos**	**Número de elementos**	**Número de nós**	**Número de equações (gl)**
PLACA010	10 x 10 x 2	200	363	1.089
PLACA016	16 x 16 x 3	768	1.156	3.468
PLACA020	20 x 20 x 4	1.600	2.205	6.615
PLACA026	26 x 26 x 5	3.380	4.374	13.122
PLACA030	30 x 30 x 6	5.400	6.727	20.181
PLACA036	36 x 36 x 7	9.072	10.952	32.856
PLACA040	40 x 40 x 8	12.800	15.129	45.387
PLACA046	46 x 46 x 9	19.044	22.090	66.270
PLACA050	50 x 50 x 10	25.000	28.611	85.833
PLACA060	60 x 60 x 12	43.200	48.373	145.119
PLACA070	70 x 70 x 14	68.600	75.615	226.845
PLACA080	80 x 80 x 16	102.400	111.537	334.611
PLACA090	90 x 90 x 18	145.800	157.339	472.017
PLACA100	100 x 100 x 20	200.000	214.221	642.663
PLACA110	110 x 110 x 22	266.200	283.383	850.149
PLACA120	120 x 120 x 24	345.600	366.025	1.098.075
PLACA130	130 x 130 x 26	439.400	463.347	1.390.041
PLACA140	140 x 140 x 28	548.800	576.549	1.729.647
PLACA150	150 x 150 x 30	675.000	706.831	2.120.493

Embora não seja objetivo deste trabalho comparar o desempenho computacional dos diversos *solvers*, as 8 primeiras malhas foram resolvidas por um código seqüencial, em um único processador, utilizando o Método dos Gradientes Conjugados com pré-condicionador Diagonal e com uma solução direta usando a eliminação de Gauss, com malha ordenada para banda mínima. Ambos códigos foram otimizados de acordo com a diretrizes mostradas no Anexo 1. A velocidade relativa do solver usando Gradientes Conjugados em relação ao solver direto pode ser vista na Figura 5.2 em função do número de equações do sistema. Até a malha PLACA020 (6615 equações) a solução direta de Gauss foi mais rápida, mas a partir daí a solução por Gradientes Conjugados se tornou mais eficiente, alcançando 12,74 vezes a velocidade de processamento da solução direta para 66 mil equações. O computador utilizado para o teste contava com 512 MB de memória RAM, e o exemplo com 66 mil equações foi o maior exemplo que podia ser resolvido com a solução direta de Gauss com essa quantidade de memória. Pelo fato de não ser preciso a montagem da matriz de rigidez global da estrutura, o mesmo computador foi capaz de resolver sistemas com até 470 mil equações com Gradientes Conjugados.

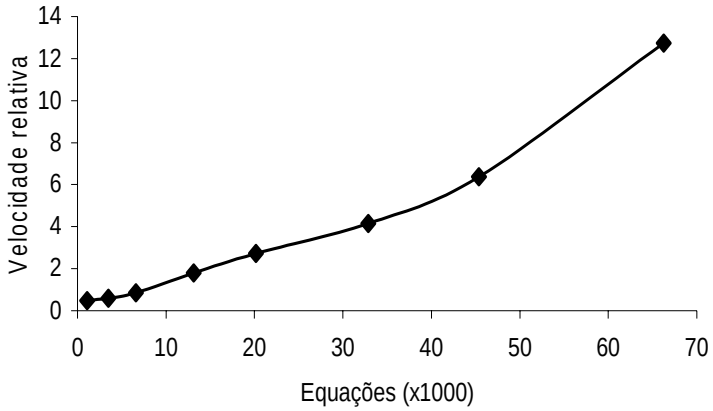


Figura 5.2: Velocidade relativa da solução de sistema de equações através do Método dos Gradientes Conjugados com pré-condicionador Diagonal em comparação com a solução do mesmo sistema pela decomposição direta de Gauss.

É preciso considerar que o sistema de equações utilizado é extremamente bem condicionado, favorecendo o desempenho de Gradientes Conjugados por exigir um número mínimo de iterações para a convergência, mas mesmo para sistemas mal condicionados a tendência é clara: a velocidade relativa de solução de Gradientes Conjugados vai se tornando exponencialmente maior com o aumento do tamanho do sistema. tornando sua escolha adequada para problemas de Elementos Finitos em que malhas muito refinadas se fazem

necessárias, seja pela complexidade da geometria a ser discretizada, seja pela complexidade do fenômeno físico sendo simulado. Em termos de demanda de memória, a vantagem é clara.

A implementação paralela foi testada com um conjunto de 8 computadores heterogêneos formando um cluster temporário, conectados entre si através de uma rede Fast Ethernet (100 Mbps). Cada computador tinha somente 1 processador com um núcleo e 512 MB de memória DDR. A especificação de cada processador está mostrada na tabela 5.3. Os computadores foram agrupados seguindo sempre o número de ordem indicado (4 computadores: do 1 ao 4, e assim por diante). A divisão de tarefas empregada utilizou o tratamento 1RN, e foi feito o balanceamento das cargas de trabalho atribuídas a cada computador.

Tabela 5.3: Conjunto de computadores utilizados em cluster temporário para teste da solução paralela usando Gradientes Conjugados

Computador	**Especificação**
1	AMD Athlon XP 2,08 GHz
2	AMD Athlon XP 2,0 GHz
3	AMD Athlon XP 2,0 GHz
4	AMD Athlon XP 1,53 GHz
5	AMD Athlon XP 1,53 GHz
6	AMD Athlon XP 1,53 GHz
7	AMD Athlon Thunderbird 1,33 GHz
8	AMD Athlon Thunderbird 1,33 GHz

5.1.1 Pré-condicionador diagonal ou de Jacobi

Para os testes com o pré-condicionador Diagonal ou de Jacobi, foram empregadas as malhas de PLACA040 a PLACA150. Em função da heterogeneidade dos computadores utilizados em termos de desempenho, o tempo de processamento de cada computador para cada problema foi avaliado, de modo a se poder quantificar os ganhos obtidos com a solução paralela. Em função da capacidade de memória, os computadores somente puderam processar de modo seqüencial as malhas de PLACA040 a PLACA090. Para as comparações com as malhas

maiores, foram feitas extrapolações dos tempos de processamento de cada máquina, como mostrado na figura 5.3, ajustando-se uma função do tipo $y = a.x^y$ com índice de determinação $R^2 = 0{,}9999$ ou superior (indicado por *extrap.* na legenda). Ainda assim, os resultados devem ser considerados com cuidado.

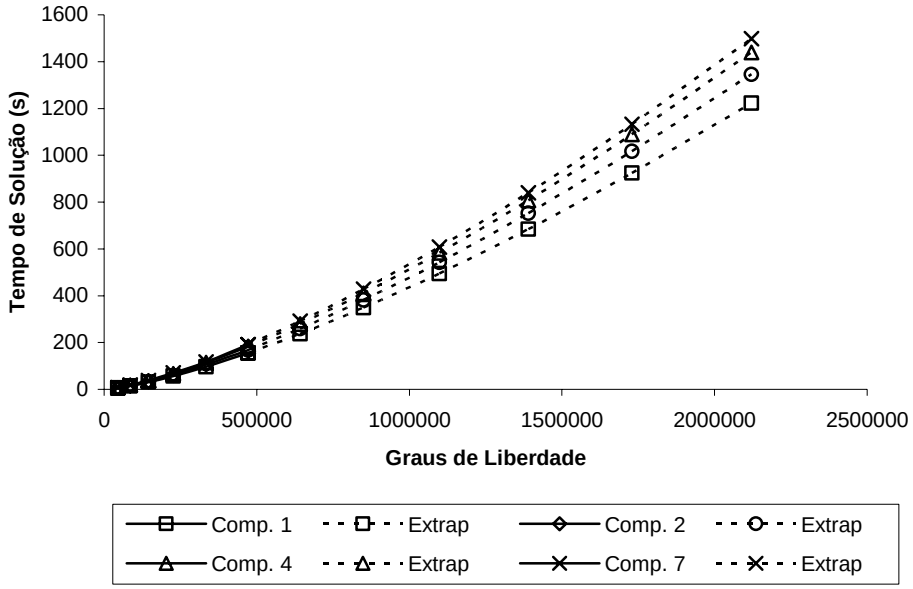


Figura 5.3: Tempo de solução utilizando Gradientes Conjugados com pré-condicionador Diagonal em função do tamanho do problema em graus de liberdade, para os diversos processadores utilizados.

O *speed-up* obtido para conjuntos de 2 a 8 computadores para cada tamanho de problema poder ser visto na figura 5.4. Mesmo os computadores sendo heterogêneos, o *speed-up* teórico é igual ao número de computadores empregado em função da normalização decorrente da equação (1.6). O ponto de término de cada curva indica o número máximo de equações que pode ser resolvido por aquele conjunto usando somente a memória principal (sem uso de memória virtual). Como esperado, quanto maior o número de computadores empregados, maior deve ser o tamanho do problema para que o *speed-up* obtido se aproxime do valor teórico.

Se as matrizes de rigidez das malhas utilizadas tiverem largura de banda aproximadamente constante, o tempo absoluto gasto com a troca de dados entre os computadores não sofre grandes alterações, pois o tratamento 1Rn empregado na divisão de tarefas faz com que o primeiro e o último computadores do conjunto troquem dados somente com um outro computador, e todos os demais com somente dois, e essa condição independe, dentro de certos

limites, do número de máquinas utilizadas. Assim, os computadores mais críticos em termos de tempo de comunicação são os centrais. Contudo, a relação entre o tempo de comunicação e o tempo de processamento vai se tornando cada vez maior com o aumento do número de computadores, limitando o ganho de velocidade, conforme a Lei da Amdhal. Contribui para isso a baixa velocidade da rede de comunicação, de apenas 100 Mbps.

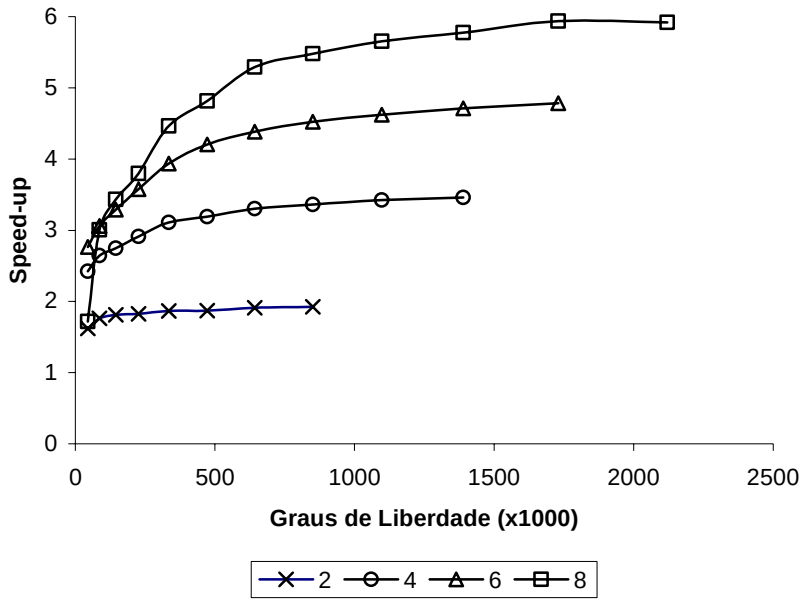


Figura 5.4: *Speed-up* da implementação paralela para Gradientes Conjugados com pré-condicionador Diagonal em função do tamanho do problema para diferentes tamanhos de *clusters* (2 a 8 computadores).

Além disso, com o aumento do número de máquinas, aumenta também a redundância de esforço computacional: a cada novo computador que é adicionado ao grupo, uma parcela maior da malha passa a ser processada por 2 computadores ao invés de um.

O resultado desses fatores é que, para cada tamanho de problema, existe um número ótimo de computadores atuando em paralelo a partir do qual a redundância no esforço computacional decorrente da inclusão de cada novo computador no conjunto não é compensada pela diminuição da carga de trabalho de cada computador, de modo que o *speed-up* passa a diminuir ao invés de aumentar.

Esse fenômeno é mostrado na figura 5.5, onde se pode ver a evolução do *speed-up* com o número de computadores utilizados em paralelo, para 4 tamanhos de problemas. Entre 45 e 227 mil graus de liberdade, o uso de 6 computadores proporciona o máximo *speed-up*, e a adição de mais computadores além desse número provoca diminuição da velocidade de

processamento do conjunto, um corportamento previsto pela lei da Amdahl. A diminuição do valor de speed-up representa o efeito da redundância do esforço computacional e da comunicação em rede sobre o processo. Para 334 mil graus de liberdade, o ponto de ótimo parece se deslocar para 8 computadores, e para além disso em problemas maiores, comportamento previsto pela lei de Gustafson.

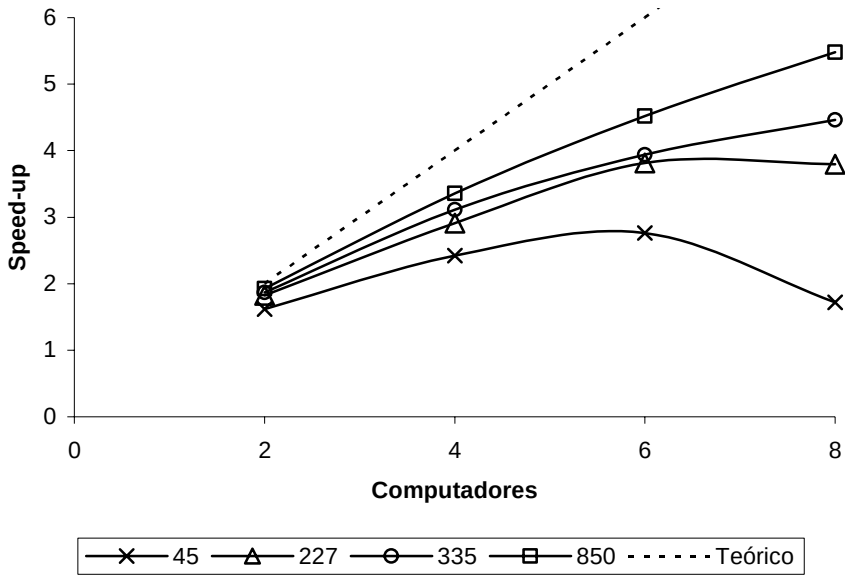
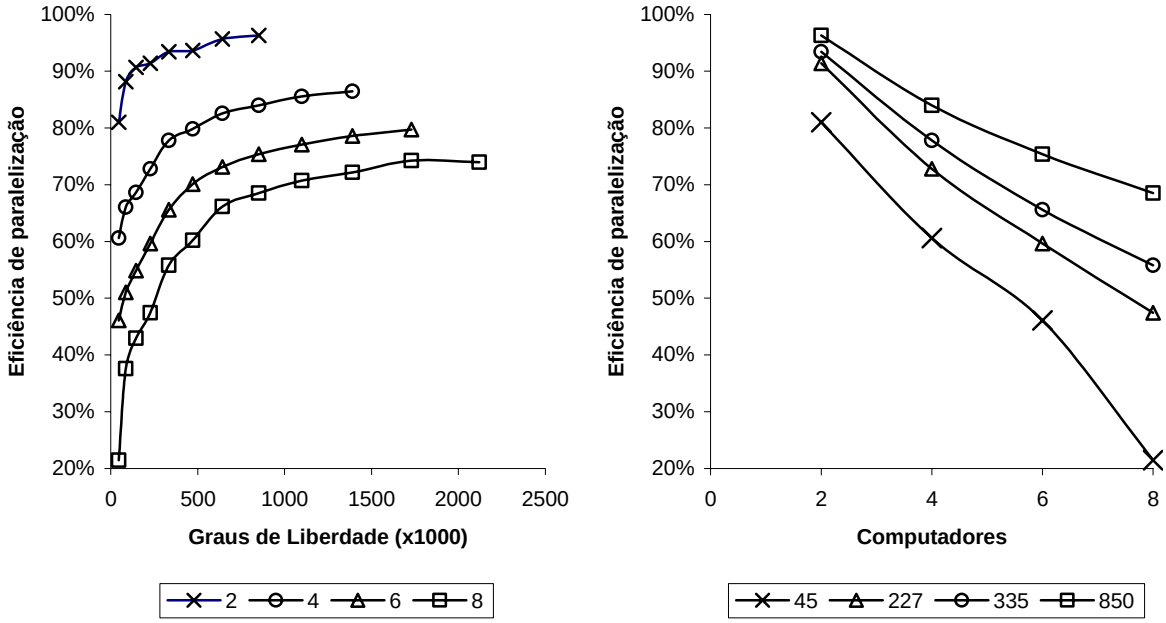


Figura 5.5: *Speed-up* de Gradientes Conjugados com pré-condicionador Diagonal em função número de computadores utilizados para alguns tamanhos de problemas em milhares de graus de liberdade.

A figura 5.6(a) mostra a eficiência de paralelização obtida para os diversos tamanhos de problema e a figura 5.6(b) em função do número de computadores empregados em paralelo.

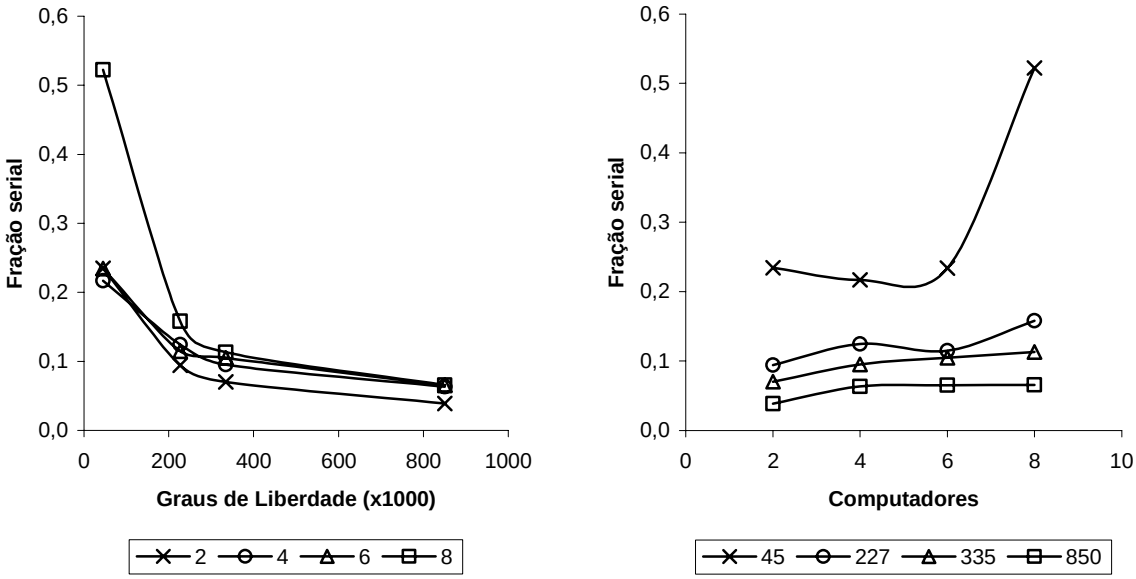
A rede relativamente lenta e a redundância do esforço computacional fazem com que a eficiência de conjuntos maiores de computadores atuando em problemas muito pequenos seja muito baixa.

A figura 5.7 mostra a fração serial de Karp-Flatt calculada em função dos valores de *speed-up* obtidos. Para problemas com até 227 mil graus de liberdade, a escalabilidade é ruim, uma vez que a fração serial cresce, especialmente com o uso de 8 computadores. A partir de 335 mil graus de liberdade, a fração serial tende a ficar constante, indicando boa escalabilidade. Fica bastante evidente por esse gráfico a perda de escalabilidade resultante da adição dos dois últimos computadores, os mais lentos, especialmente em problemas pequenos.



(a) (b)

Figura 5.6: Eficiência de paralelização de Gradientes Conjugados com pré-condicionador Diagonal (a) em função do tamanho do problema para diferentes tamanhos de *clusters* e (b) em função do tamanho de *cluster* para diferentes tamanhos de problemas em milhares de graus de liberdade



(a) (b)

Figura 5.7: Fração serial de Karp-Flatt para Gradientes Conjugados com pré-condicionador Diagonal (a) em função do tamanho do problema para diferentes tamanhos de *clusters* e (b) em função do tamanho de *cluster* para diferentes tamanhos de problemas em milhares de graus de liberdade.

Para verificar a influência da heterogeneidade dos computadores trabalhando em paralelo em termos de capacidade de processamento, um segundo conjunto de 4 máquinas foi utilizado para teste com as malhas PLACA040 a PLACA090. Todos os computadores possuíam 512 MB de memória DDR, e as especificações dos processadores estão mostradas na tabela 5.4. As eficiências de paralelização desse conjunto e do conjunto anterior utilizando os 4 primeiros computadores estão mostradas na figura 5.8.

Tabela 5.4: Conjunto de computadores heterogêneos utilizados em cluster temporário para teste da solução paralela usando Gradientes Conjugados

Computador	Especificação
1	AMD Athlon XP 2,35 GHz
2	AMD Athlon XP 2,10 GHz
3	AMD Athlon Thunderbird 1,35 GHz
4	AMD Athlon Pluto 0,8 GHz

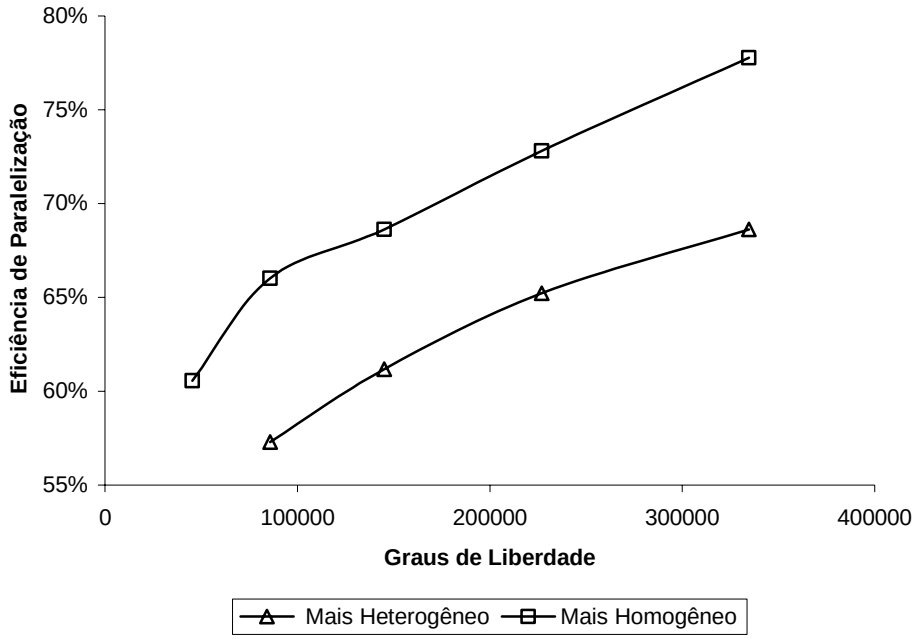


Figura 5.8: Eficiência de paralelização de Gradientes Conjugados com pré-condicionador Diagonal em função tamanho do problema em graus de liberdade para conjuntos de 4 computadores com diferentes graus de homogeneidade.

Apesar do processo de balanceamento da distribuição de tarefas compensar parcialmente a heterogeneidade do conjunto, é mais eficiente a utilização de computadores com um desempenho próximo entre si do que computadores de capacidades de processamento muito diferentes. Em função do balanceamento, um computador menos potente receberá um conjunto menor de nós e elementos para processar, mas se a malha apresentar uma largura de banda aproximadamente constante e for utilizado o tratamento 1RN, a quantidade de dados a ser comunicada através da rede e o número de elementos redundantes desse computador não é muito diferente dos correspondentes da máquina mais potente, levando à perda de eficiência.

5.1.2 Pré-condicionador de Cholesky

Para os testes com o pré-condicionador de Cholesky, foi empregado o mesmo conjunto de computadores indicado na tabela 5.3 e os mesmos procedimentos de avaliação do desempenho da solução paralela em relação à solução seqüencial utilizados para o pré-condicionador Diagonal. Em função da necessidade de armazenamento das matrizes fatoradas do pré-condicionador de Cholesky, cada computador individualmente conseguiu resolver apenas os problemas de PLACA040 a PLACA070 sem o uso de memória virtual. Com o conjunto de 8 computadores, foi possível a solução das malhas até PLACA130. Para as comparações com as malhas maiores (entre PLACA080 e PLACA130), foram feitas extrapolações dos tempos de processamento de cada máquina, como mostrado na figura 5.9, através de um ajuste perfeito de uma função do tipo $y = a.x^y$ (índice de determinação R^2 = 1,0). Ainda assim, os resultados devem ser considerados com cuidado.

Pela comparação das figuras 5.3 e 5.9, nota-se que apesar do pré-condicionador de Cholesky ter uma melhor taxa de convergência (menor número de iterações para a mesma tolerância), o tempo total de solução obtido foi maior, pois ele exige um esforço computacional consideravelmente maior que o pré-condicionador Diagonal. Contudo, é importante lembrar que os exemplos em análise são extremamente bem condicionados, e que sistemas mal condicionados exigem um número muito maior de iterações ou simplesmente não há convergência quando é empregado o pré-condicionador Diagonal.

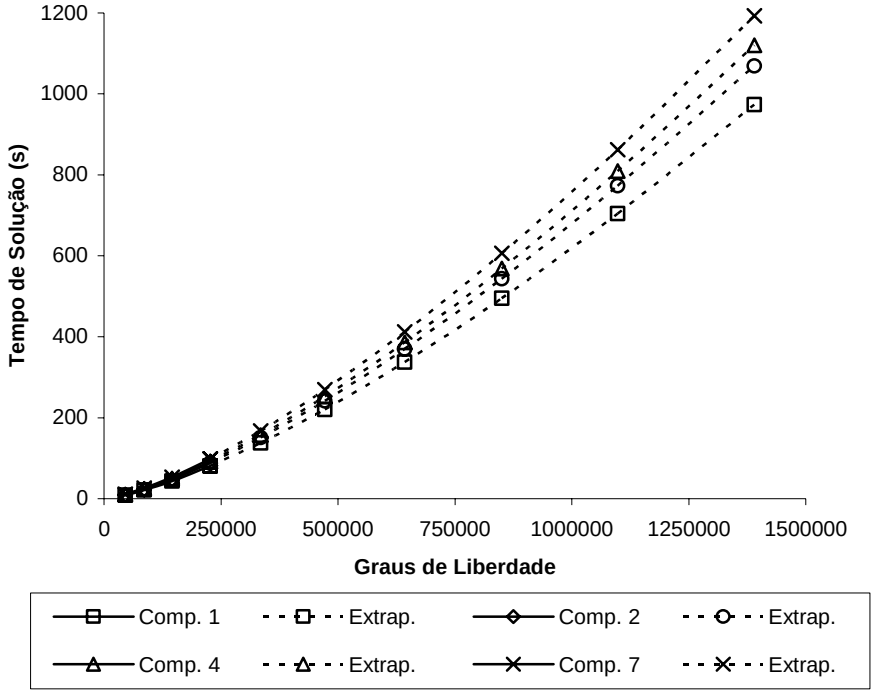


Figura 5.9: Tempo de solução utilizando Gradientes Conjugados com pré-condicionador de Cholesky em função do tamanho do problema em graus de liberdade, para os diversos processadores utilizados.

Como apontado no Capítulo 4, a taxa de convergência do Método dos Gradientes Conjugados é dependente da ordem dos elementos na etapa de aplicação da decomposição de Cholesky elemento-por-elemento. A implementação paralela tem, em geral uma taxa de convergência mais baixa que a versão seqüencial pela alteração da ordem dos elementos em cada computador do *cluster*, resultando em um número maior de iterações. O número de iterações até a convergência para as malhas de PLACA040 a PLACA090 obtidos na versão seqüencial e na versão paralela para 2 e 4 computadores, bem como a diferença percentual em entre elas estão mostrados na Tabela 5.5. Esse efeito está incluído nos valores de *speed-up* e eficiência de paralelização mostrados a seguir.

Os valores de *speed-up* obtidos com conjuntos de 2 a 8 computadores para diversos tamanhos de problemas podem ser vistos nas figuras 5.10 e 5.11.

O Método dos Gradientes Conjugados com pré-condicionador de Cholesky mostrou-se mais eficiente do ponto de vista de desempenho que com o pré-condicionador Diagonal, pois não somente o tempo de solução é menor como o *speed-up* alcançado pela versão paralela é maior, especialmente para os problemas maiores e com um maior número de computadores.

Para um problema com 1,39 milhões de graus de liberdade, um conjunto de 8 computadores foi 6,68 vezes mais rápido que um único computador (estimativa), e somente o menor de todos os problemas apresentou como tamanho ótimo do cluster 6 computadores. Para todos os demais tamanhos, o número máximo de computadores disponíveis foi a melhor alternativa. A desvantagem do pré-condicionador de Cholesky é o sua maior demanda por memória, não permitindo a solução de problemas tão grandes como o pré-condicionador Diagonal.

Tabela 5.5: Número de iterações até a convergência para o Método dos Gradientes Conjugados com pré-condicionador de Cholesky, versão seqüencial e paralela.

Malha	Seqüencial	2 Comp.	Diferença	4 Comp.	Diferença
PLACA040	93	99	6,5%	101	8,6%
PLACA050	116	123	6,0%	125	7,8%
PLACA060	138	148	7,2%	149	8,0%
PLACA070	161	171	6,2%	173	7,5%
PLACA080	184	195	6,0%	197	7,1%
PLACA090	206	219	6,3%	221	7,3%

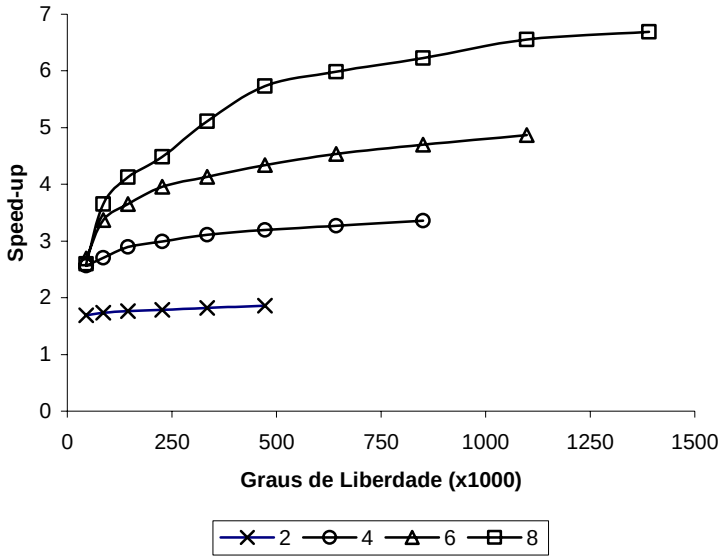


Figura 5.10: *Speed-up* de Gradientes Conjugados com pré-condicionador de Cholesky em função do tamanho do problema para diferentes tamanhos de *clusters*.

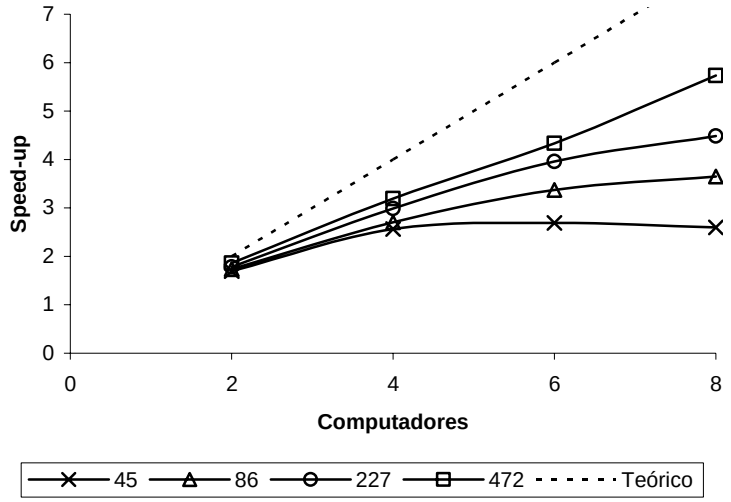
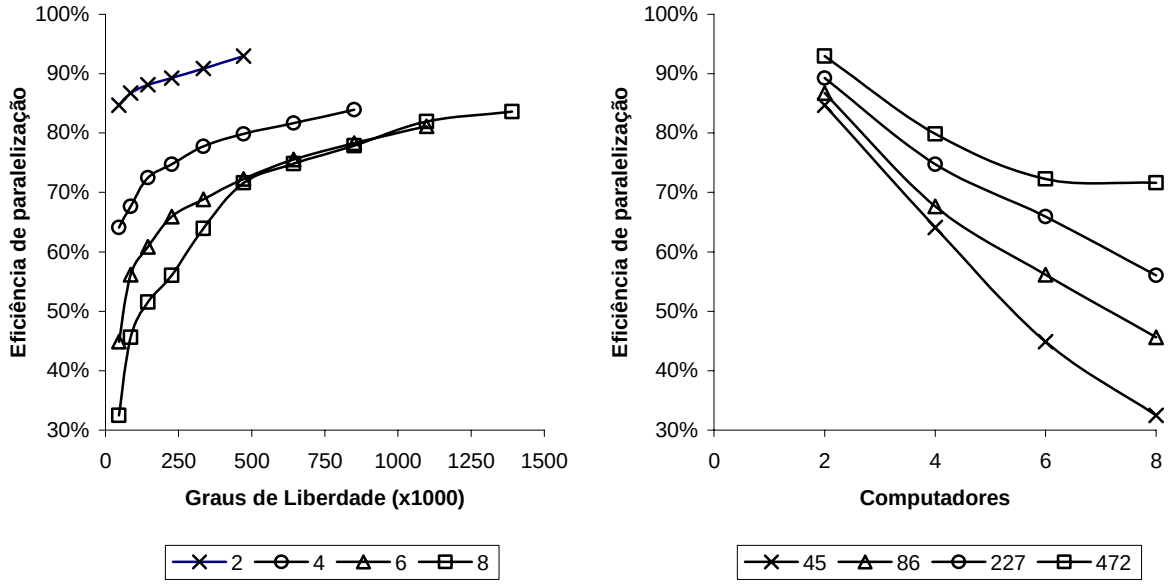


Figura 5.11: *Speed-up* de Gradientes Conjugados com pré-condicionador de Cholesky em função número de computadores utilizados para alguns tamanhos de problemas em milhares de graus de liberdade.

A eficiência de paralelização obtida está mostrada na figura 5.12.

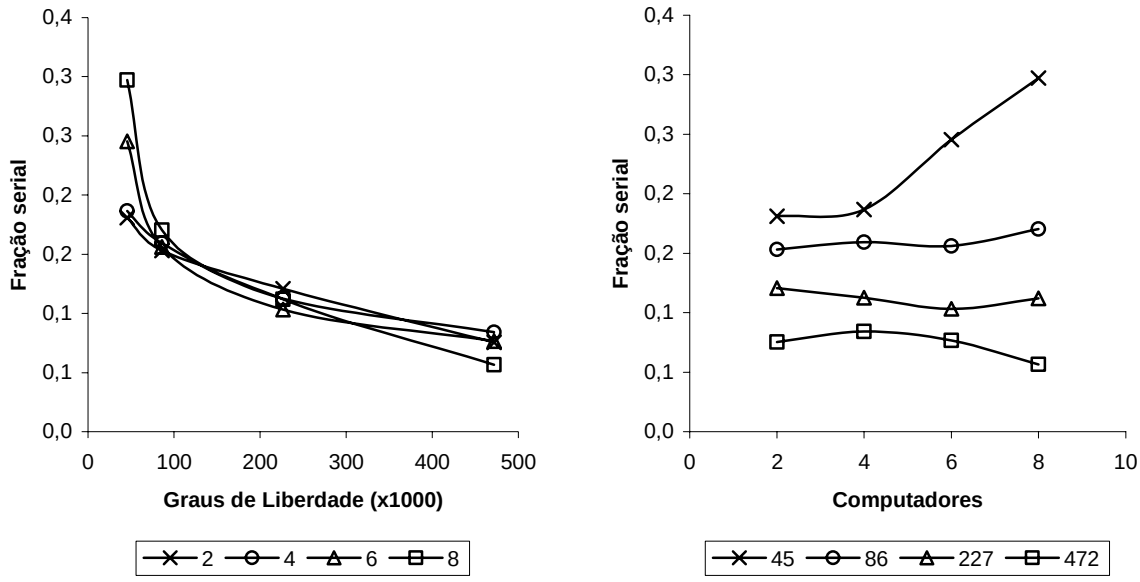


(a) (b)

Figura 5.12: Eficiência de paralelização de Gradientes Conjugados com pré-condicionador de Cholesky (a) em função do tamanho do problema para diferentes tamanhos de *clusters* e (b) em função do tamanho de *cluster* para diferentes tamanhos de problemas (x 1000 GL).

Os resultados para a fração serial de Karp-Flatt obtidos estão mostrados na figura 5.13. Para problemas com 86 mil ou mais graus de liberdade, há uma tendência de estabilização do

valor da fração serial, indicando boa escalabilidade. A fração serial diminui com o aumento do tamanho do problema, em concordância com a lei de Gustafson.



(a) (b)

Figura 5.13: Fração serial de Karp-Flatt para Gradientes Conjugados com pré-condicionador de Cholesky (a) em função do tamanho do problema para diferentes tamanhos de *clusters* e (b) em função do tamanho de *cluster* para diferentes tamanhos de problemas (x 1000 GL).

Para o conjunto mais heterogêneo de computadores, a eficiência de paralelização obtida foi mais baixa, como mostrado na figura 5.14 para um conjunto com os 4 computadores da tabela 5.4 e outro com os primeiros 4 computadores da tabela 5.3, reproduzindo o comportamento obtido com o pré-condicionador Diagonal.

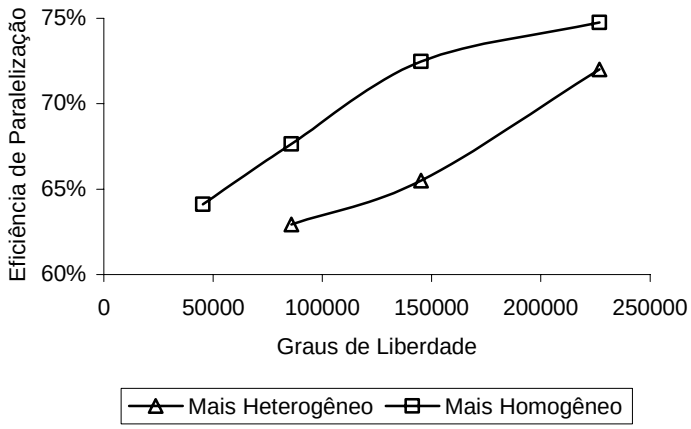


Figura 5.14: Eficiência de paralelização de Gradientes Conjugados com pré-condicionador de Cholesky em função tamanho do problema em graus de liberdade para conjuntos de 4 computadores com diferentes graus de homogeneidade.

5.2 ESCOAMENTOS COMPRESSÍVEIS EM REGIME TRANSÔNICO E SUPERSÔNICO

Para os testes do algoritmo paralelo explícito de Taylor-Galerkin de integração no tempo de um passo com iterações para a solução de problemas de escoamentos compressíveis tridimensionais, o mesmo foi implementado utilizando-se elementos hexaédricos lineares de 8 nós com integração das matrizes de elemento efetuada analiticamente (Gresho, 1984) considerando o elemento não distorcido e usando um único ponto de integração para avaliar a matriz Jacobiana, conforme apresentado por Burbridge (1999) e elementos tetraédricos lineares com as matrizes também avaliadas de forma analítica, sem a necessidade de integração numérica, conforme apresentado por Bono (2008).

5.2.1 Elementos Hexaédricos – Escoamento em torno de uma esfera

O exemplo utilizado para os testes iniciais com o esquema de um passo com interações consiste no escoamento de um fluído não viscoso compressível ao redor de uma esfera de raio R = 1,0, tendo-se como região de interesse a metade anterior. Considerando a simetria do problema, é suficiente definir o domínio correspondente a apenas um oitavo da esfera, como mostrado na figura 5.15. A velocidade de referência é Mach = 3,0 e a constante do gás γ foi considerada igual a 1,4. O modelo de escoamento utilizado é o de Euler.

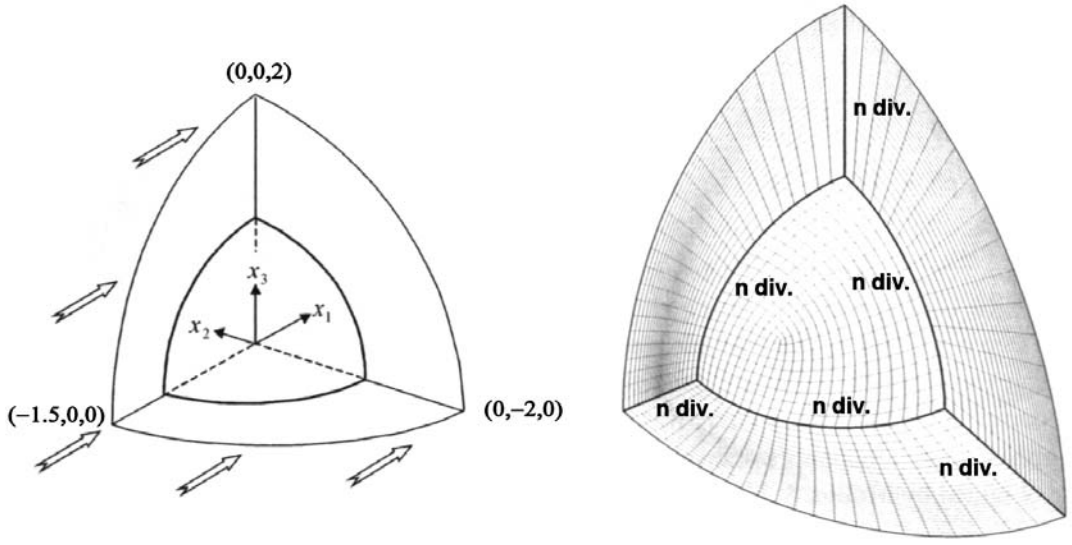


Figura 5.15: Geometria do domínio e discretização empregada para o problema de escoamento ao redor de uma esfera. *Fonte: Burbridge (1999)*

Condições de contorno foram impostas aos planos de simetria (x_1,x_2) e (x_1,x_3) para que através deles não exista fluxo de massa ou de qualquer outra quantidade envolvida na análise do problema. O domínio utilizado na análise está limitado pelo contorno sólido da esfera e por um elipsóide de raios (1,5; 2,0 ; 2,0) nas direções x_1, x_2 e x_3, respectivamente, adotado como *contorno de entrada* com as condições correspondentes à corrente não perturbada, ou seja, $v_{1\infty} = 3,0$, $v_{2\infty} = v_{3\infty} = 0,0$, $e_\infty = 6,2857$ e $\rho_\infty = 1,0$. O plano (x_2,x_3) é o *contorno de saída* sobre o qual não é aplicado nenhuma condição.

Sobre o domínio de análise foi gerada uma família de malhas, cada uma correspondendo a um número diferente de divisões (n div.) das linhas de definição do domínio, conforme inidicado na figura 5.15. As malhas obtidas foram designadas por ESF*xxx*, onde *xxx* representa o número de divisões empregado. As características das mesmas podem ser vistas na tabela 5.6.

Tabela 5.6: Família de malhas de elementos hexaédricos lineares usadas no problema de escoamento em torno de uma esfera.

Malha	**Número de elementos**	**Número de nós**	**Número de graus de liberdade**
ESF020	6.000	6.951	34.755
ESF030	20.250	22.351	111.755
ESF040	48.000	51.701	258.505
ESF050	93.750	99.501	497.505
ESF060	162.000	170.251	851.255
ESF070	257.250	268.451	1.342.255
ESF080	384.000	398.601	1.993.005
ESF090	546.750	565.201	2.826.005
ESF100	750.000	772.751	3.863.755
ESF110	998.250	1.025.751	5.128.755

O primeiro teste sobre a implementação paralela foi feito com um *cluster* temporário heterogêneo composto por 6 computadores conectados através de uma rede *Fast Ethernet* (100 Mbps). Cada computador dispunha de um processador com um único núcleo e de 512 MB de memória RAM. Os agrupamentos foram feitos seguindo o número de ordem indicado na tabela 5.7. A divisão de tarefas foi feita utilizando-se o tratamento 1RN com balanceamento da carga de trabalho computacional.

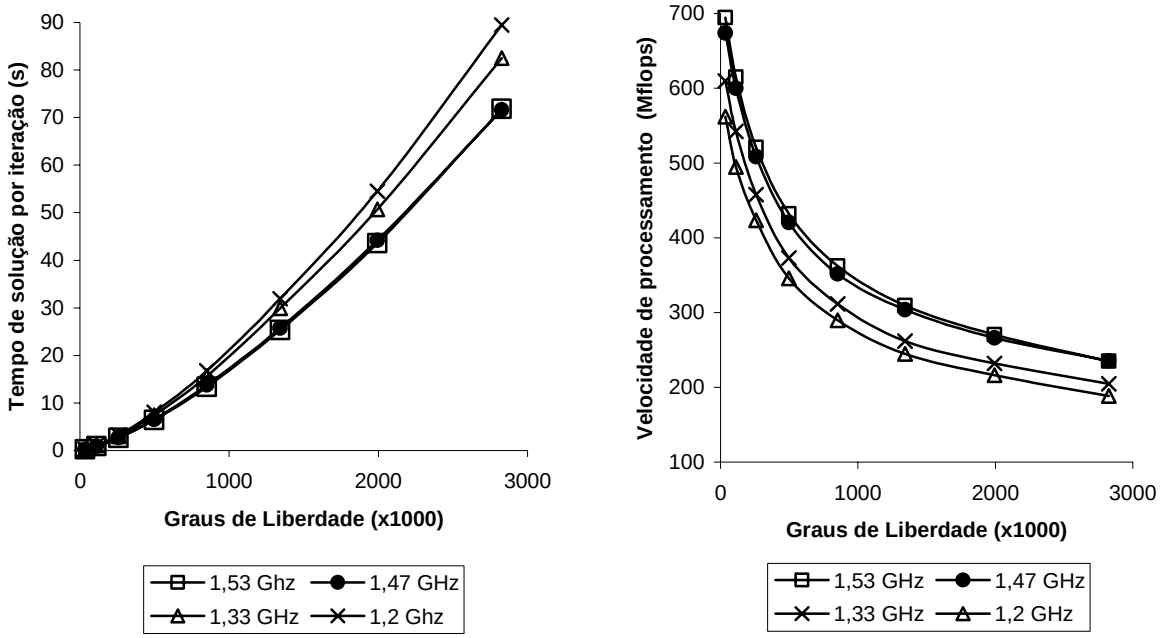
Tabela 5.7: Conjunto de computadores utilizados em cluster temporário para teste da implementação paralela do esquema explícito de 1 passo com iterações.

Computador	**Especificação**
1	AMD Athlon XP 1,53 GHz
2	AMD Athlon XP 1,53 GHz
3	AMD Athlon XP 1,47GHz
4	AMD Athlon XP 1,33 GHz
5	AMD Athlon XP 1,33 GHz
6	AMD Athlon XP 1,2 GHz

Para cada tamanho de problema, foi feita a simulação do escoamento limitando-se a análise a um número fixo de passos de tempo representativo da solução. Os passos de tempo utilizados correspondem à parte inicial de solução.

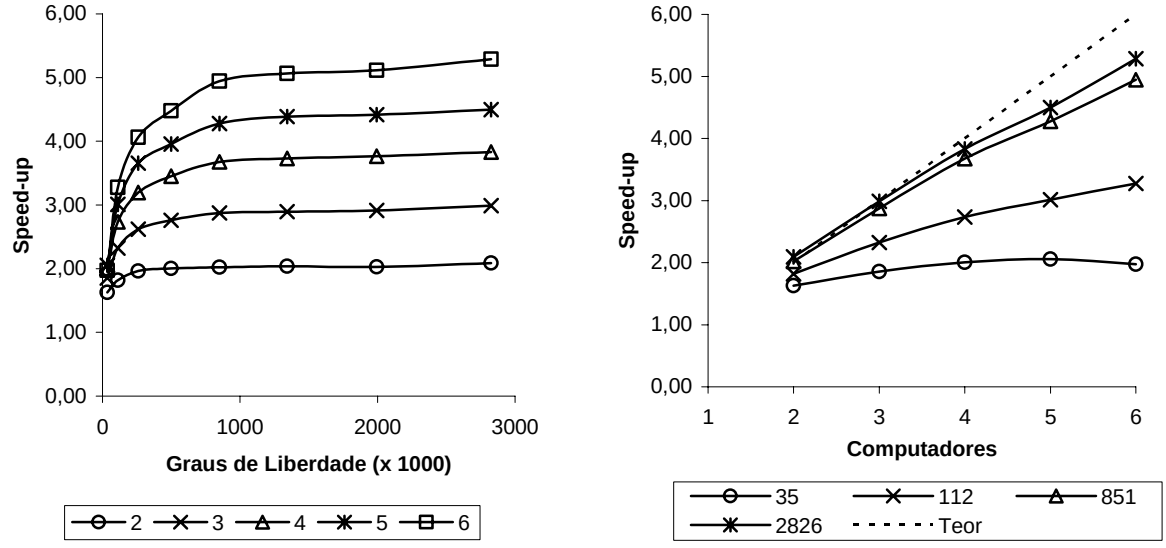
Os diversos tamanhos de problema foram analisados por cada computador do conjunto de forma seqüencial. Em função da memória RAM disponível, a análise se restringiu em cada computador aos problemas até o ESF090. Os tempos de solução obtidos, por iteração, estão mostrados na figura 5.16(a), mostrando que o desempenho do código aumenta quase linearmente com a velocidade do processador. Entre o computador mais rápido e o mais lento há uma diferença de 27,5% na freqüência do processador e de 24,6% na velocidade de execução média do código seqüencial implementado. As velocidades de processamento do código seqüencial em milhões de operações de ponto flutuante por segundo (Mflops) para cada processador foi avaliada pela comparação com o tempo de processamento do mesmo código em um supercomputador CRAY T-94, e podem ser vistas na figura 5.16(b). A diminuição da velocidade de processamento em Mflops com o aumento do tamanho do problema é decorrente do tamanho finito do *cache* do processador frente a crescente quantidades de dados a serem mapeados.

A partir da análise seqüencial, foi possível quantificar o desempenho da solução paralela agrupando-se os computadores em conjuntos de 2 a 6 máquinas. Os resultados de *speed-up* e eficiência de paralelização estão mostrados nas figuras 5.17 e 5.18.



(a) (b)

Figura 5.16: (a)Tempo de solução por iteração e (b) velocidade de processamento para o algoritmo seqüencial do esquema explicito de 1 passo com iterações em função do tamanho do problema em milhares de graus de liberdade, para processadores com freqüência entre 1,2 e 1,53 GHz.



(a) (b)

Figura 5.17: *Speed*-up do esquema explícito de um passo com iterações (a) em função tamanho do problema para diferentes tamanhos de *clusters* e (b) em função número de computadores utilizados para diferentes tamanhos de problemas em milhares de graus de liberdade.

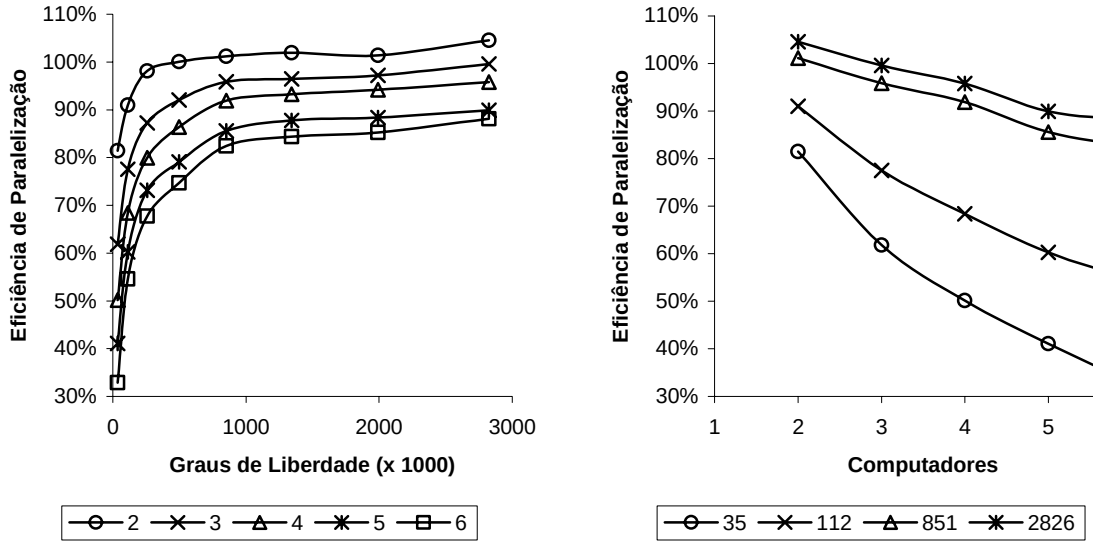


Figura 5.18: Eficiência de paralelização para o esquema explícito de um passo com iterações (a) em função tamanho do problema para diferentes tamanhos de *clusters* e (b) em função número de computadores utilizados para diferentes tamanhos de problemas em milhares de graus de liberdade.

Somente para o menor tamanho de problema (35 mil graus de liberdade ou equações) houve estagnação do *speed-up*, com o número ótimo de computadores sendo 4. Para problemas muito pequenos, ao se dividir as tarefas entre um número maior de computadores, a parcela de tempo gasta para o processamento propriamente dito torna-se pequena ao ser comparada com o tempo gasto na comunicação de dados através da rede, causando perda de desempenho.

Para todos os tamanhos de *clusters*, há um leve aumento da eficiência para a maior da malhas, a ESF090. Isso se deve não à execução do código paralelo ficar mais rápido, mas sim ao código seqüencial ser executado mais lentamente para essa malha. A quantidade de memória exigida por essa malha para uma execução seqüencial é bastante próxima da memória instalada em cada computador. Adicionando-se a parcela de memória destinada ao sistema operacional, o total provavelmente superou a memória RAM disponível, fazendo com que o código seqüencial fizesse uso, mesmo que pequeno, da memória virtual e, portanto, perdesse velocidade No código paralelo a memória utilizada é menor, de modo que o problema não acontece e o código é executado mais rapidamente, ganhando eficiência de paralelização. Percebe-se a ocorrência de um *speed-up* superlinear com o uso de 2 computadores para as 2 maiores malhas, decorrente de uma maior eficiência do *cache* dos processadores quando o conjunto de dados correspondente às malhas é dividido em 2.

Os valores de fração linear de Karp-Flatt obtidos estão mostrados na figura 5.19.

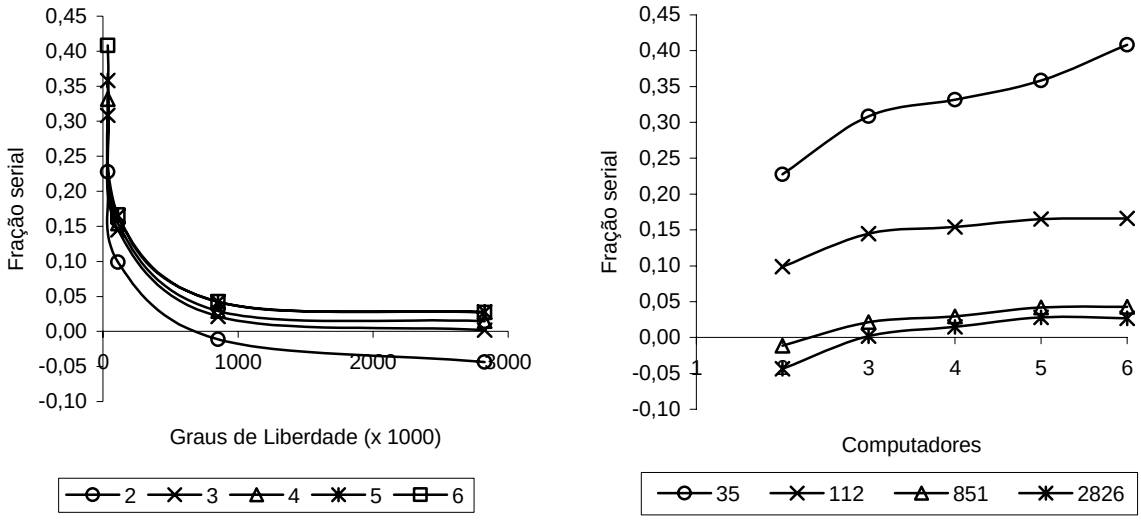


Figura 5.19: Fração serial para o esquema explícito de um passo com iterações (a) em função do tamanho do problema para diferentes tamanhos de *clusters* e (b) em função do tamanho do *cluster* para diferentes tamanhos de problemas em milhares de graus de liberdade.

A partir de 93 mil graus de liberdade (malha ESF050), a fração linear mostra-se praticamente constante e com valores bastante baixos, indicando excelente escalabilidade. A existência de *speed-up* superlinear é indicada pelos valores de fração serial negativos.

A implementação paralela foi testada também com um conjunto de 11 computadores em um cluster permanente heterogêneo, conectados através de uma rede *Fast Ethernet* (100 Mbps) ou Gigabit Ethernet (1 Gbps). Cada computador dispunha de um processador com um único núcleo AMD Athlon 64 e de 4 GB de memória RAM. Os agrupamentos foram feitos seguindo o número de ordem indicado na tabela 5.8. A divisão de tarefas foi feita utilizando-se o tratamento 1RN com balanceamento da carga de trabalho computacional. Esse conjunto é mais homogêneo que o primeiro em termos de desempenho dos computadores, sendo a diferença de freqüência entre o processador mais rápido e o mais lento de 8,7% e entre a velocidade média de execução do código seqüencial de 8,8%, conforme pode ser visto na figura 5.20. Os computadores do cluster permanente são consideravelmente mais potentes que os utilizados anteriormente, apresentando velocidade de processamento entre 1,8 e 2 vezes maior para o código seqüencial.

Tabela 5.8: Conjunto de computadores utilizados em cluster permanente para teste da implementação paralela do esquema explícito de 1 passo com iterações.

Computador	Freqüência do núcleo
1	2,5 GHz
2	2,3 GHz
3	2,4 GHz
4	2,4 GHz
5	2,3 GHz
6	2,3 GHz
7	2,4 GHz
8	2,4 GHz
9	2,5 GHz
10	2,4 GHz
11	2,5 GHz

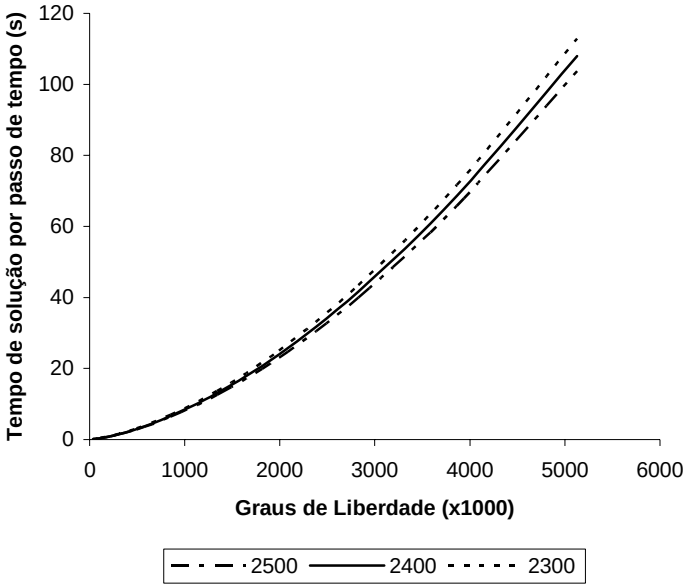
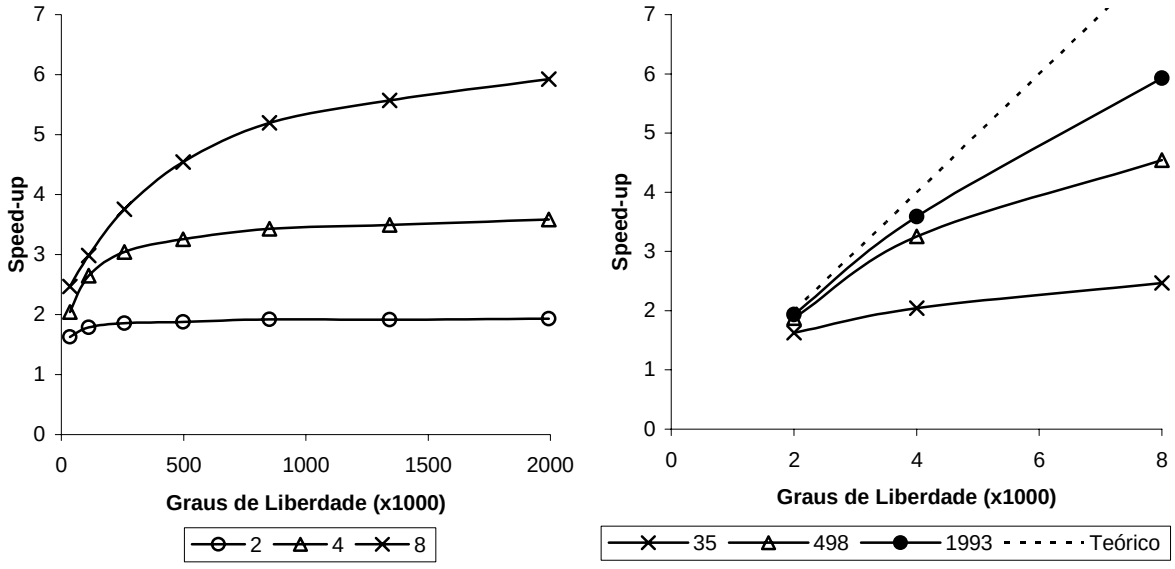


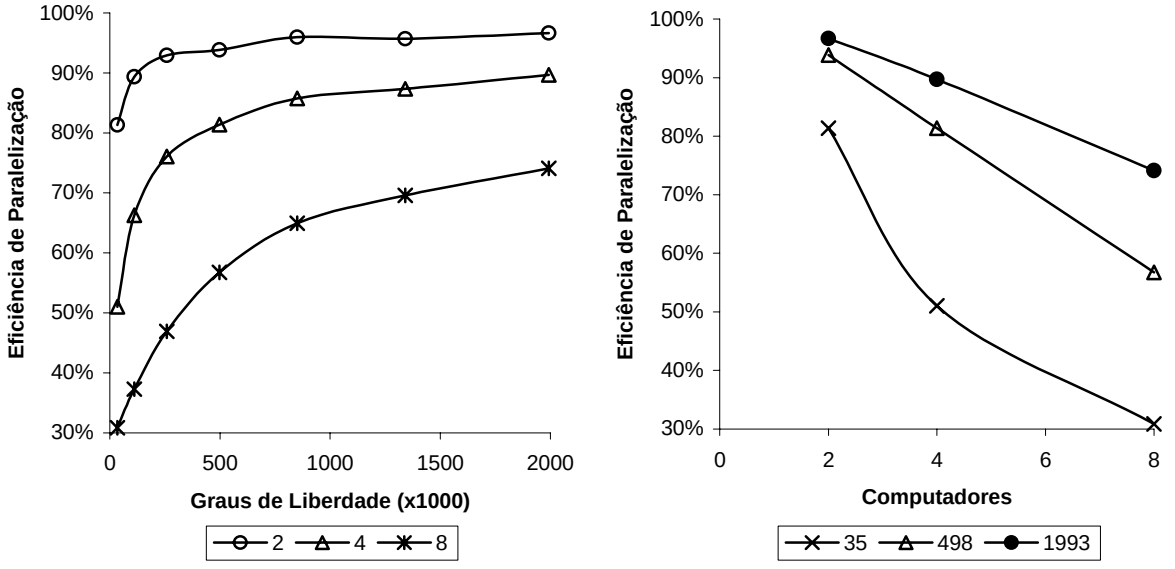
Figura 5.20: Tempo de solução por passo de tempo para o algoritmo paralelo do esquema explicito de 1 passo com iterações em função do tamanho do problema em milhares de graus de liberdade, para processadores com freqüência entre 2,3 e 2,5 GHz.

Com o uso da rede de 100 Mbps, os computadores foram agrupados em conjuntos de 2, 4 e 8 máquinas para os problemas de ESF020 a ESF080, sendo obtidos os valores de *speed-up* e de eficiência de paralelização mostrados nas figuras 5.21 e 5.22, respectivamente.



(a) (b)

Figura 5.21: *Speed*-up do esquema explícito de um passo com iterações (a) em função tamanho do problema para diferentes tamanhos de *clusters* e (b) em função número de computadores utilizados para diferentes tamanhos de problemas em milhares de graus de liberdade. *Cluster* permanente, rede de 100 Mbps.



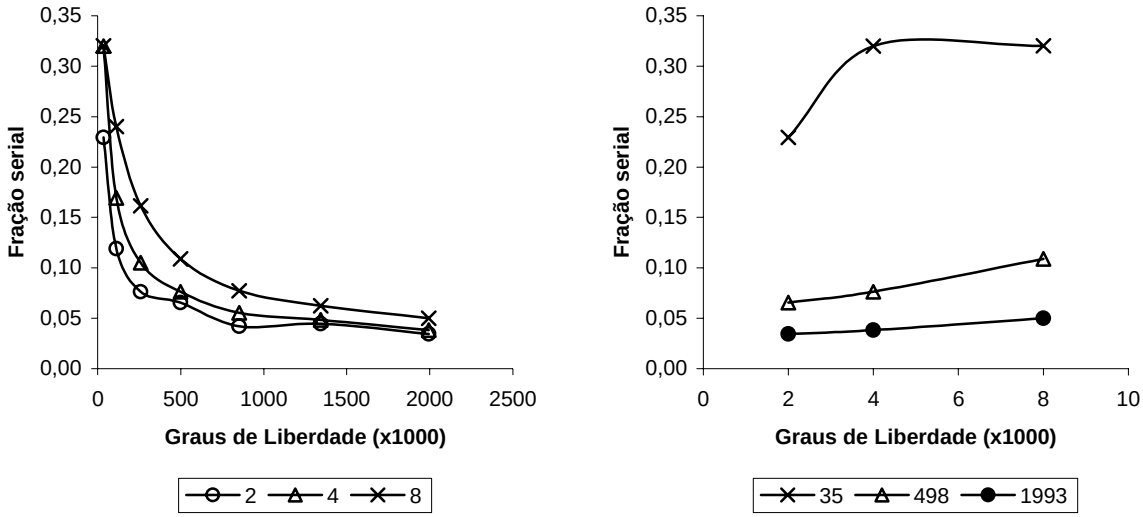
(a) (b)

Figura 5.22: Eficiência de paralelização para o esquema explícito de um passo com iterações (a) em função do tamanho do problema para diferentes tamanhos de *clusters* e (b) em função número de computadores utilizados para diferentes tamanhos de problemas em milhares de graus de liberdade. *Cluster* permanente, rede de 100 Mbps.

Pela comparação dos gráficos das figuras 5.21 e 5.17, percebe-se que com o *cluster* temporário de máquinas menos potentes foram obtidos valores de *speed-up* superiores aos obtidos com o *cluster* permanente. A diferença média entre os s*peed-ups* para cada tamanho de problema e cada tamanho de conjunto de computadores entre o *cluster* temporário e o *cluster* permanente foi de 4,5%.

O mesmo pode ser visto na comparação entre as figuras 5.22 e 5.18. Isso não significa que o *cluster* temporário com máquinas menos potentes processe os problemas de forma paralela mais rapidamente que o *cluster* permanente, apenas que o primeiro o faz de forma mais eficiente. Considerando que a velocidade relativa da comunicação de dados entre processadores em relação à velocidade de processamento para o *cluster* temporário é maior, o resultado obtido é coerente. Para manter a escalabilidade da solução paralela, é preciso que a velocidade da rede cresça de forma proporcional à velocidade de processamento dos computadores empregados.

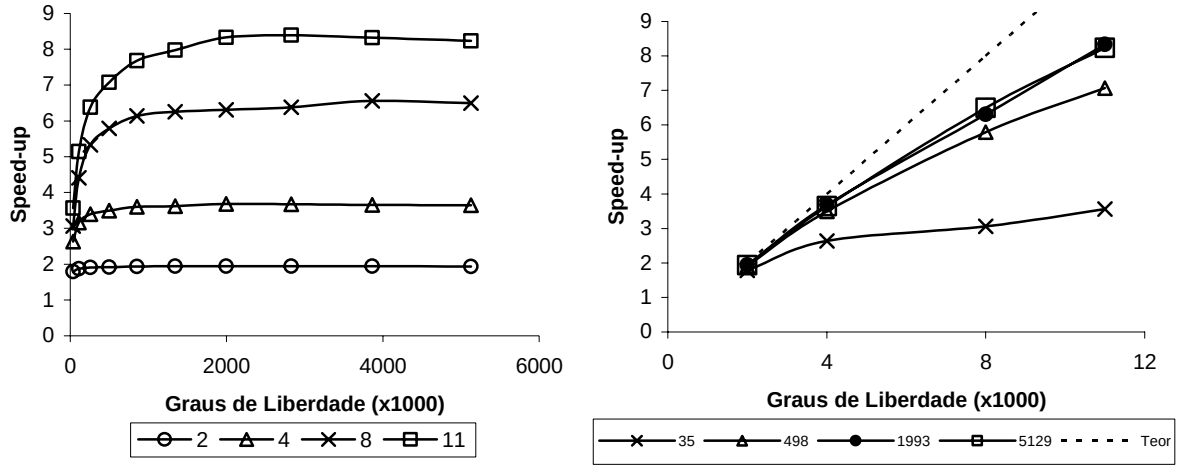
Os valores obtidos para a fração serial de Karp-Flatt estão mostrados na figura 5.23. Os valores obtidos são baixos apenas para problemas de maior tamanho, e as curvas são levemente ascendentes com o aumento do número de processadores, indicando uma escalabilidade razoável para os tamanhos de problema analisados.



(a) (b)

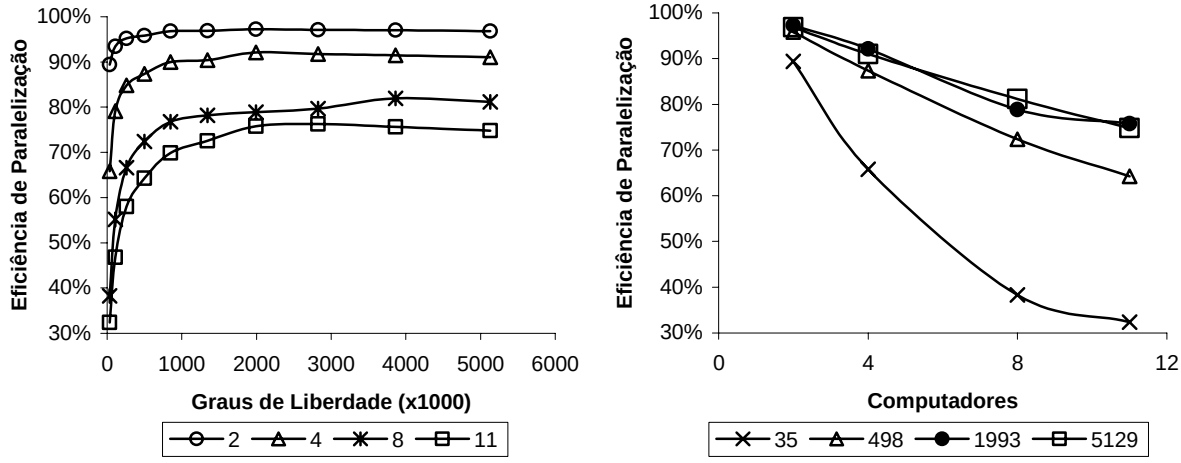
Figura 5.23: Fração serial para o esquema explícito de um passo com iterações (a) em função tamanho do problema para diferentes tamanhos de *clusters* e (b) em função número de computadores utilizados para diferentes tamanhos de problemas em milhares de graus de liberdade. *Cluster* permanente, rede de 100 Mbps.

Com o uso da rede de 1 Gbps, os computadores foram agrupados em conjuntos de 2, 4, 8 e 11 máquinas para os problemas de ESF020 a ESF110, sendo obtidos os valores de *speed-up* e de eficiência de paralelização mostrados nas figuras 5.24 e 5.25, respectivamente.



(a) (b)

Figura 5.24: *Speed*-up da implementação paralela para o esquema explícito de um passo com iterações (a) em função tamanho do problema para diferentes tamanhos de *clusters* e (b) em função número de computadores utilizados para diferentes tamanhos de problema. *Cluster* permanente, rede de 1 Gbps.



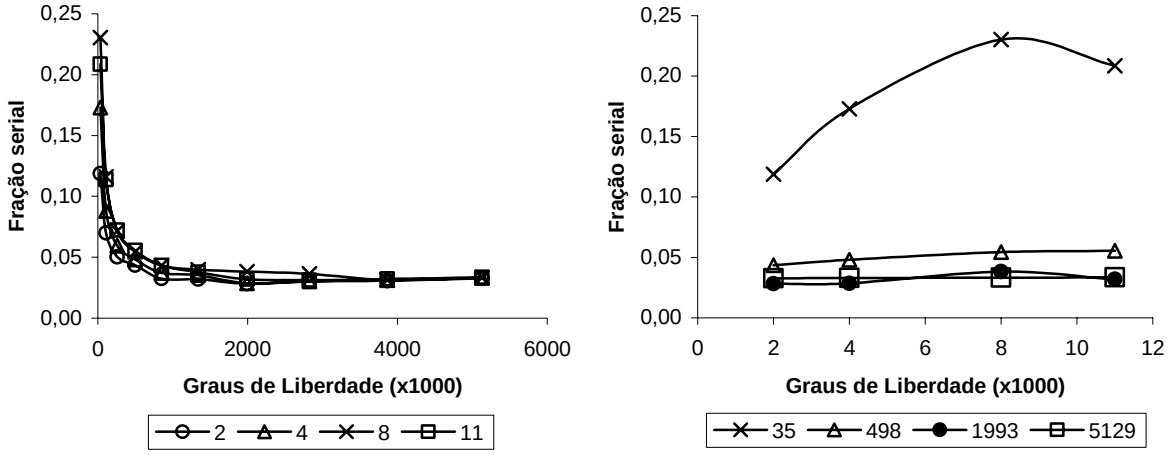
(a) (b)

Figura 5.25: Eficiência de paralelização para o esquema explícito de um passo com iterações (a) em função tamanho do problema para diferentes tamanhos de *clusters* e (b) em função número de computadores utilizados para diferentes tamanhos de problema. *Cluster* permanente, rede de 1 Gbps

A rede mais rápida trouxe ganho para todos os tamanhos de problemas e de grupos de processadores. A partir de 1 milhão de graus de liberdade, grupos de até 8 processadores

atingem o *speed-up* máximo, enquanto que com a rede de 100 Mbps isso acontece somente com grupos de até 4 processadores para esse tamanho de problema.

Os valores obtidos para a fração serial de Karp-Flatt estão mostrados na figura 5.26.



(a) (b)

Figura 5.26: Fração serial para o esquema explícito de um passo com iterações (a) em função tamanho do problema para diferentes tamanhos de *clusters* e (b) em função número de computadores utilizados para diferentes tamanhos de problema. *Cluster* permanente, rede de 1 Gbps

Em relação aos valores mostrados na figura 5.23, os resultados obtidos com a rede de 1Gbps tem valores mais baixos e comportamento mais constante, indicando uma melhor escalabilidade.

A figura 5.27 mostra o ganho no *speed*-up obtido para conjuntos de até 8 processadores ao se utilizar a rede de 1 Gbps em comparação à rede de 100 Mbps. Os maiores ganhos correspondem às configurações com maior número de computadores atuando em paralelo sobre os menores tamanhos de problemas, onde os tempos associados tanto à maior velocidade da rede de 1Gbps como sua menor latência têm maior importância em relação ao tempo de processamento propriamente dito de cada máquina. Por outro lado, em problemas muito grandes com um número pequeno de computadores, o tempo gasto em troca de dados entre os computadores através da rede é muito pequeno em relação ao tempo de processamento propriamente dito, fazendo com que não haja diferença em termos prático entre o uso de uma rede de 1 Gbps ou de 100 Mbps. O emprego de *clusters* temporários utilizando os computadores e redes disponíveis em um laboratório de desenvolvimento e

pesquisa torna-se viável para esse cenário sem que seja necessária a aquisição de uma infra-estrutura mais cara em termos de rede.

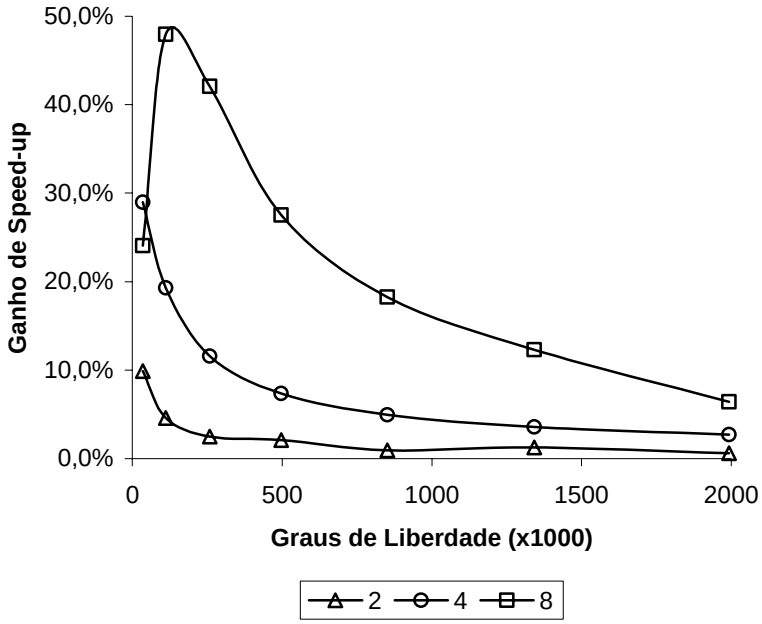


Figura 5.27: Ganho de *speed-up* com o uso de uma rede de 1 Gbps em comparação a uma rede de 100 Mbps para o esquema explícito de um passo com iterações em função tamanho do problema para diferentes tamanhos de *clusters*.

Considerando os valores estabilizados de *speed-up* do gráfico da figura 5.24 para cada tamanho de conjunto de computadores, foi possível fazer uma previsão aproximada do tamanho ótimo de c*luster*, ou seja, o número de computadores a partir do qual não haveria mais ganho na velocidade de processamento do problema com a adição de novos computadores. Os valores obtidos com os testes estão grafados na figura 5.28, e uma função parabólica foi ajustada sobre esse pontos, mostrada como uma linha pontilhada.

A previsão assim obtida deve ser encarada como uma aproximação, em função do limitado número de pontos disponível, mas indica que o tamanho ótimo do *cluster* para a aplicação paralela utilizada teria em torno de 24 computadores, com um *speed-up* máximo da ordem de 12, resultando em aproximadamente 50% de eficiência de paralelização. Apesar do uso de computadores e rede de comunicação com diferentes desempenhos em relação aos empregados levarem a modificações nos valores obtidos, o comportamento indicado pela figura 5.25 aponta com uma configuração composta por um número relativamente pequeno de computadores com o maior poder computacional possível como alternativa ótima para o

algoritmo empregado, ao invés de um grande número de computadores de desempenho médio (computação massivamente paralela).

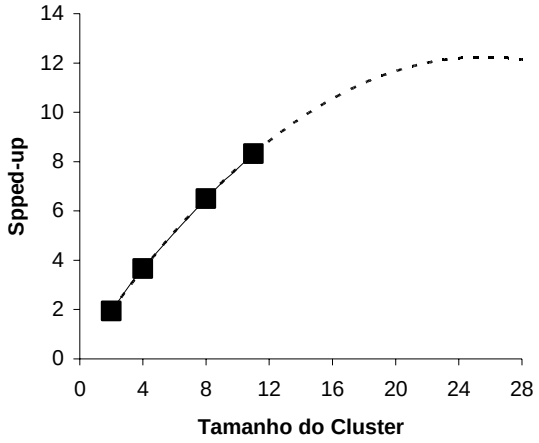
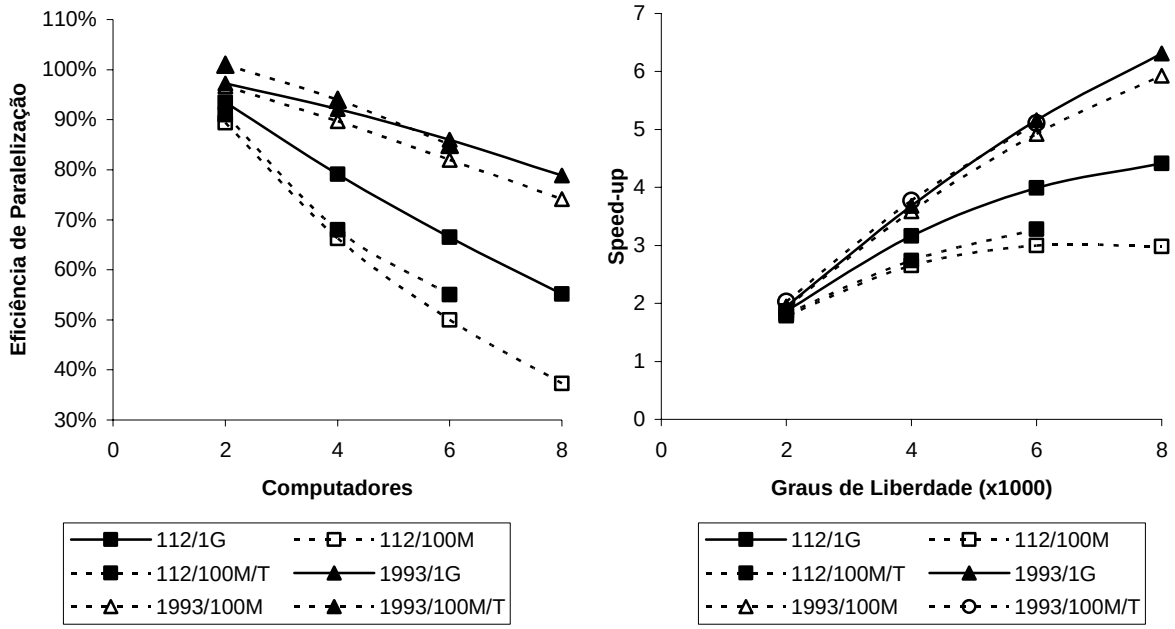


Figura 5.28: Previsão do tamanho ótimo do *cluster* para o esquema explícito de um passo com iterações.

A influência da velocidade relativa da rede em relação a velocidade de processamento dos processadores pode ser resumida na figura 5.29, onde o *speed-up* e a eficiência de paralelização obtidos para grupos de 2 a 8 computadores do *cluster* temporário usando rede de 100 Mbps e do *cluster* permanente usando redes de 100 Mbps e 1 Gbps estão mostrados para os problemas ESF030 (112 mil graus de liberdade) e ESF080 (1,993 milhões de graus de liberdade). Nos gráficos, o *cluster* temporário está indicado pela letra T.

Para problemas pequenos, a menor latência da rede de 1 Gbps levou aos melhores resultados. Com a rede de 100 Mbps, o conjunto com a maior velocidade relativa da rede em comparação à velocidade de processamento foi conjunto com maior eficiência de paralelização e *speed-up*. Para problemas grandes, onde a latência da comunicação em rede tem pouca influência sobre o desempenho, os resultados para as 3 configurações foram muito próximos. Os resultados com as máquinas rápidas e rede de 100 Mbps foi o que apresentou menor eficiência. Os resultados para o conjunto de máquinas mais lentas com rede de 100 Mbps e de máquinas mais rápidas com rede de 1Gbps foram praticamente equivalentes.

Aumentar a velocidade dos computadores utilizados sem aumentar a velocidade da rede aumenta a velocidade absoluta de processamento em paralelo mas diminui a eficiência e, conseqüentemente, o número ótimo de computadores que pode ser utilizado.



(a) (b)

Figura 5.29: (a) Eficiência de paralelização e (b) *speed-up* para o esquema explícito de um passo com iterações usando redes e conjuntos de computadores de diferentes velocidades para 2 tamanhos de problemas, em milhares de graus de liberdade.

5.2.2 Elementos Hexaédricos – Escoamento em torno de um veículo espacial

Para testes com uma geometria mais complexa foi utilizada a simulação do escoamento de um fluído não-viscoso com um número de Mach igual a $M_\infty = 2,95$ em torno de um corpo com geometria composta por uma meia esfera, um cilindro e um tronco de cone acoplados, com um ângulo de ataque de 10º em relação à direção do escoamento, representando, em termos gerais, o vôo de uma cápsula espacial na alta atmosfera. O problema é similar ao apresentado por Bono (2008), mas discretizado através de hexaedros lineares, ao invés de tetraedros. Foram utilizadas 2 malhas na discretização, cujas características estão mostradas na tabela 5.9. A malha VE1 está mostrada na figura 6.30, dando uma idéia geral do formato do veículo.

Tabela 5.9: Características das malhas utilizadas na discretização do escoamento em torno de um veículo espacial.

Malha	Número de Elementos	Número de Nós	Graus de Liberdade
VE1	198750	211146	1.055.730
VE2	917700	949212	4.746.060

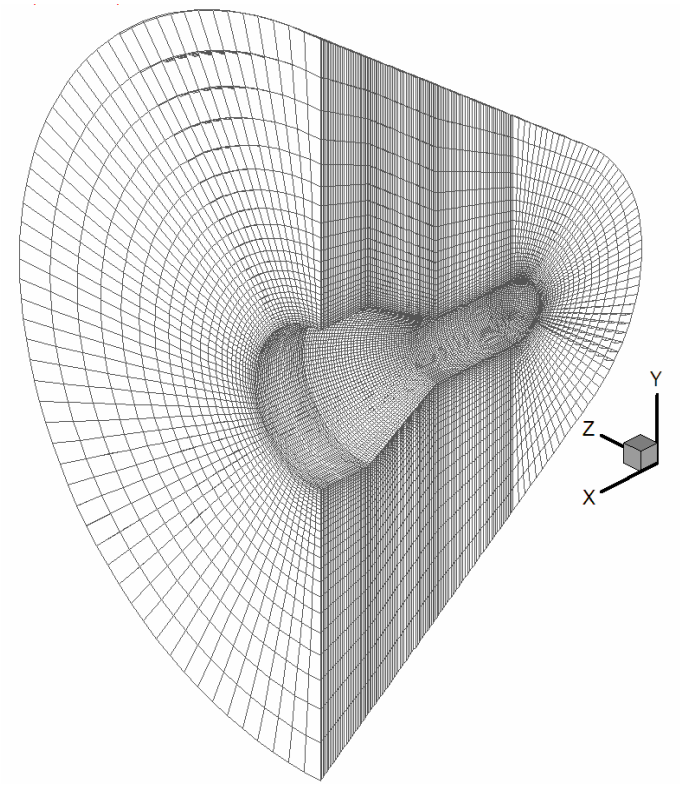


Figura 5.30: Malha VE1 utilizada na discretização do escoamento em torno de um veículo espacial.

A implementação paralela foi utilizada com um conjunto de 15 computadores em um cluster permanente heterogêneo. Cada computador dispunha de um processador com AMD Athlon 64 e de 4 GB de memória RAM. Os agrupamentos foram feitos seguindo o número de ordem indicado na tabela 5.10. Dos 15 computadores, 5 dispunham de um processador de núcleo duplo. Os primeiros núcleos desses computadores receberam os números de ordem 1 a 4 e 13, e os segundos 16 a 20. O objetivo desse ordenamento foi fazer com que problemas muito grandes não sobrecarregassem a memória das máquinas com processador de núcleo duplo, uma vez que os mesmos 4 GB das máquinas com processadores de núcleo simples deveriam atender 2 processos e o dobro da quantidade de dados. Com o ordenamento utilizado, grupos com até 15 processadores utilizariam somente um núcleo dessas máquinas, e com 16 ou mais processadores a quantidade de memória utilizada por cada núcleo seria pequena o suficiente para não forçar o uso de memória virtual. Além disso, também era objetivo tentar simular um *cluster* com 20 máquinas com processadores de núcleo simples, por ser uma configuração mais crítica, visto que todas as trocas de dados são feitas através da rede. Com o ordenamento utilizado, dificilmente um núcleo do processador precisará trocar dados com o outro, o que seria feito através da memória, e sim com núcleos de outras máquinas, o que é feito através da

rede. A presença de uma única interface de rede torna mais crítica ainda a configuração, fazendo com que a configuração utilizada seja uma aproximação conservativa de um *cluster* com 20 computadores com processadores de núcleo simples.

Tabela 5.10: Freqüência dos processadores utilizados na simulação do escoamento em torno de um veículo espacial.

Processador	**Freqüência do núcleo**
1 a 4	2,2 GHz
5 e 6	2,0 GHz
7	2,2 GHz
8 a 12	2,0 GHz
13	2,0 GHz
14 a 20	2,2 GHz

O padrão de rede utilizado foi o Gigabit Ethernet (1Gbps). As configurações das interfaces de rede foram otimizadas para utilizar pacotes de dados de 9 KB, o valor máximo para o padrão *Gigabit Ethernet Jumbo Frame*, uma vez que o *switch* de rede suportava essa característica. Além disso, o sistema operacional dos computadores foi reconfigurado para suportar um número maior de conexões simultâneas, e os tamanhos dos *buffers* de transmissão e recepção foram aumentados. Isso levou a uma melhora substancial de desempenho quanto à comunicação de dados.

As malhas utilizadas foram obtidas diretamente a partir de um gerador de malhas comercial, que já executava uma reordenação nodal para minimização da banda, de forma que o as malhas já continham o tratamento 1RN para divisão de tarefas. Foram executados testes em 3 condições:

a) com o tratamento 1RN aplicado, mas sem balanceamento das cargas de trabalho;

b) com o tratamento 1RN e balanceamento das cargas de trabalho;

c) com o tratamento nRN e balanceamento das cargas de trabalho.

Em todas as condições, a distribuição de tarefas inicial foi baseada na freqüência de cada processador lógico como índice de desempenho.

De forma similar ao que foi feito com o problema de escoamento em torno da esfera, apenas um determinado número de passos de tempo da solução foi utilizado como medida para a determinação do desempenho paralelo. Cada malha foi analisada com o código seqüencial individualmente por cada um dos computadores do conjunto, de modo a fornecer os tempos de solução utilizados na determinação do *speed-up* e da eficiência de paralelização, cujos resultados estão mostrados nas figuras 5.31 e 5.32, respectivamente.

O balanceamento prévio das cargas de trabalho mostrou-se tanto mais importante quanto maior o número de processadores lógicos utilizados. Quando o número de processadores é grande, o número de nós e de elementos alocados a cada processador não é muito grande, fazendo com que os nós e elementos associados a cargas ou condições de contorno possam ficar concentrados em um único processador, dada a distribuição não uniforme dessas características ao longo da malha. Processadores lógicos com um número idêntico de nós podem ter cargas de trabalho computacional bastante diferentes. Sem o balanceamento para equalizar a distribuição, a máquina mais lenta (aquela que é responsável pelos nós e elementos associados ao maior esforço computacional) governa o desempenho do conjunto. Esse efeito pode aparecer mais fortemente em algumas configurações quanto ao número de processadores que em outras, explicando o comportamento observado para conjuntos de 12 processadores nas figuras 5.31 e 5.32.

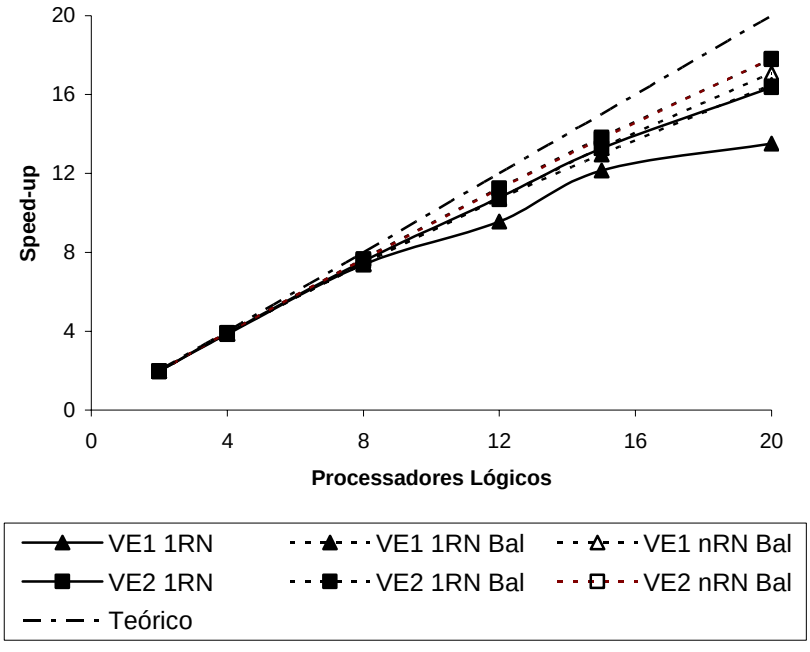


Figura 5.31: *Speed-up* do esquema explícito com 1 passo e iterações para as malhas VE1 e VE2 utilizadas na discretização do escoamento em torno de um veículo espacial.

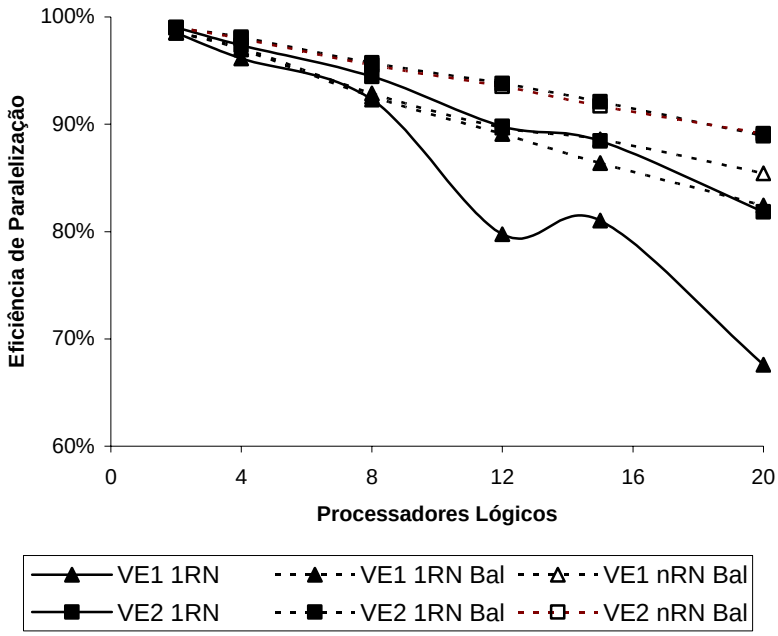


Figura 5.32: Eficiência de paralelização para o esquema explícito com 1 passo e iterações para as malhas VE1 e VE2 utilizadas na discretização do escoamento em torno de um veículo espacial.

Do ponto de vista prático não há diferença entre os resultados obtidos para os tratamentos 1RN e nRN balanceados com as malhas estruturadas utilizadas. O tratamento nRN apresentou ligeira vantagem (3% aproximadamente) para a malha de menor tamanho VE1 com o uso de conjuntos com maior número de processadores lógicos. Para a malha de maior tamanho VE2, os resultados são virtualmente idênticos. A eficiência de paralelização obtida para as duas malhas foi excelente, indicando a possibilidade de se utilizar um número bem maior de processadores lógicos com ganho de velocidade de processamento.

Os resultados para a fração serial de de Karp-Flatt estão mostrados na figura 5.33. As divisões de tarefas sem balanceamento das cargas de trabalho apresentaram uma escalabilidade bastante variável. O processo de balanceamento levou a frações seriais praticamente constantes e bastante baixas, comprovando a excelente escalabilidade do código paralelo para as malhas utilizadas.

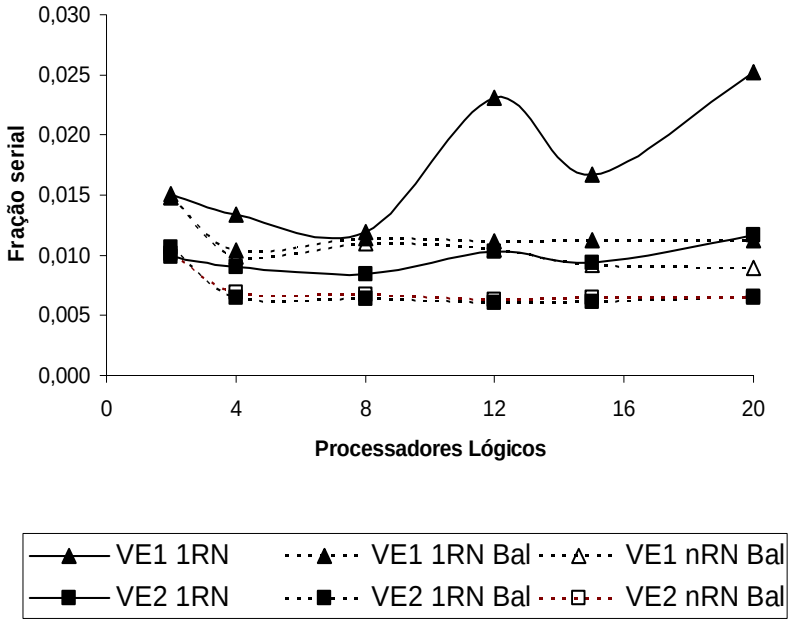


Figura 5.33: Fração serial para o esquema explícito com 1 passo e iterações para as malhas VE1 e VE2 utilizadas na discretização do escoamento em torno de um veículo espacial.

5.2.3 Elementos Tetraédricos – Escoamento supersônico em torno de uma esfera

O mesmo problema da seção 5.2.1 foi utilizado para teste da implementação do algoritmo paralelo para o esquema explícito de 1 passo com iterações com o emprego de elementos tetraédricos. A malha utilizada é formada por 331.799 elementos tetraédricos lineares e 61.095 nós, totalizando 305.475 graus de liberdade, e está mostrada na Figura 5.34.

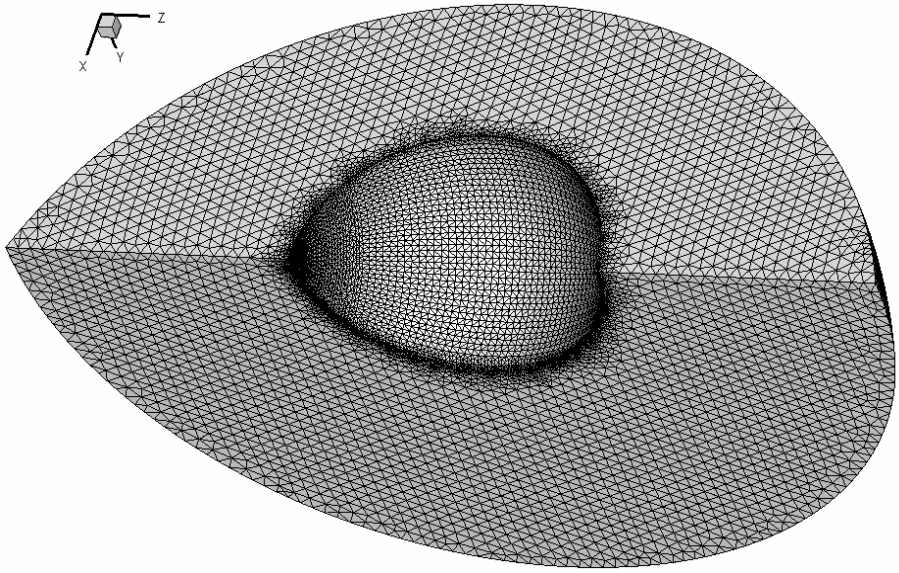


Figura 5.34: Malha de elementos tetraédricos utilizada para o problema de escoamento supersônico em torno de uma esfera.

Os mesmos computadores utilizados na simulação do escoamento em torno do veículo espacial foram empregados, mas com a ordem dos núcleos alterada. Como o tamanho da malha utilizada é relativamente pequeno, não havia o problema da malha inteira sendo processada pelos 2 núcleos de um mesmo processador esgotar a memória disponível, de modo que os núcleos dos processadores de núcleo duplo foram colocados seqüencialmente no ordenamento do cluster. Grupos de 2 a 20 computadores foram estabelecidos seguindo a ordem indicada na tabela 5.11.

Tabela 5.11: Freqüência dos processadores utilizados na simulação do escoamento supersônico em torno de uma esfera.

Processador	**Freqüência do núcleo**
1 a 8	2,2 GHz
7 a 18	2,0 GHz
19	2,2 GHz
20	2,0 GHz

A divisão de tarefas foi feita considerando a malha original sem nenhum tratamento e com os tratamentos 1RN e nRN. Os resultados em termos de speed-up e eficiência de paralelização estão mostrados nas figuras 6.35 e 6.36, respectivamente.

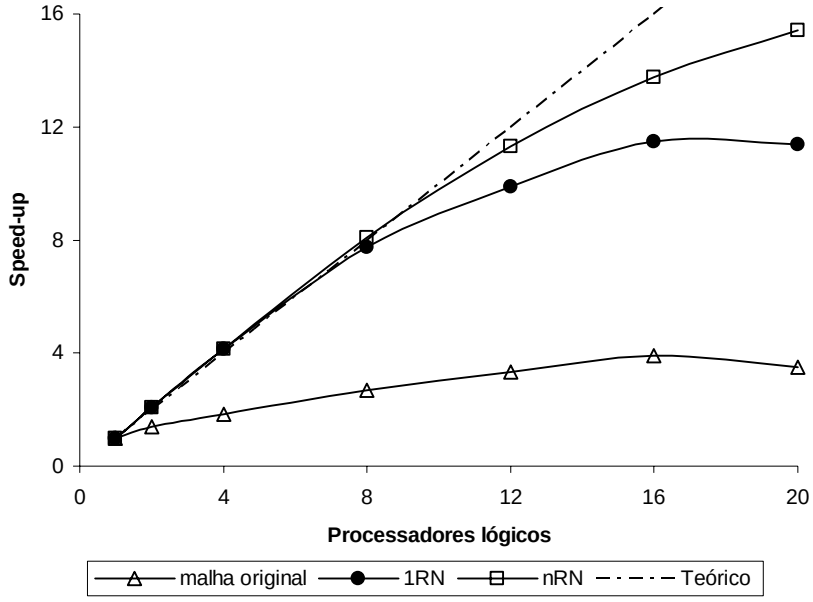


Figura 5.35: Speed-up obtido para o problema de escoamento supersônico em torno de uma esfera com diversos tratamentos para a divisão de tarefas, malha de tetraedros, .

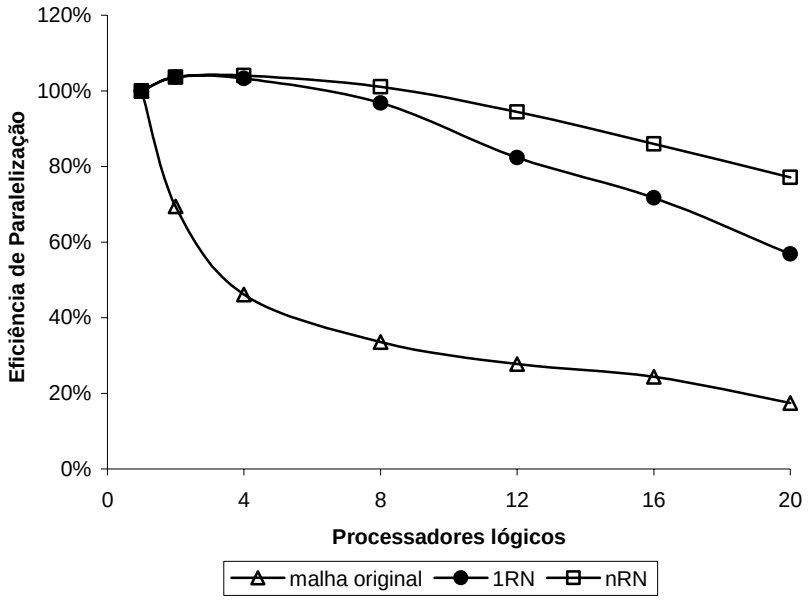


Figura 5.36: Eficiência de paralelização obtida para o problema de escomanento supersônico em torno de uma esfera com diversos tratamentos para a divisão de tarefas, malha de tetraedros.

Para a malha original ou com o tratamento 1RN, o tamanho ótimo de *cluster* é de 16 processadores lógicos. O aumento do número de núcleos de processamento a partir deste ponto causa estagnação ou diminuição da velocidade de processamento do conjunto, pois o poder de processamento adicionado com cada núcleo não é suficiente para compensar a perda de eficiência de paralelização decorrente do aumento da comunicação de dados pela rede e da redundância de esforço computacional inerentes à divisão de tarefas entre os núcleos.

O uso do tratamento nRN apresentou, para qualquer número de processadores lógicos, melhor eficiência de paralelização, permitindo o crescimento da velocidade com o acréscimo de processadores até o número máximo de 20 processadores. Para conjuntos com 16 processadores, a solução com o tratamento 1RN é aproximadamente 2,95 vezes mais rápida que a solução sem tratamento. O uso do tratamento nRN permitiu uma solução 3,44 vezes mais rápida que sem tratamento, e 16,5% mais rápida que com o tratamento 1RN. O speed-up máximo com o tratamento nRN é 33,9% maior que o com o 1RN (15,4 com 20 núcleos contra 11,5 com 16 núcleos), e 294,8% maior que o *speed-up* máximo para os dados sem tratamento (3,9 com 16 núcleos).

As eficiências de paralelização acima de 100% obtidas com conjuntos de até 4 processadores com os tratamentos 1RN e nRN (*speed-up* superlinear) são explicadas pelo uso mais eficiente

do *cache* do processador (independente para cada núcleo, no caso do AMD AThlon 64 X2 utilizado) que precisa mapear uma quantidade menor de dados na configuração paralela que quando um único núcleo é responsável pela processamento da totalidade dos dados.

Para verificar a influência da velocidade da rede no desempenho da implementação paralela, fez-se um teste com os oito primeiros processadores conectados através de uma rede de 100 Mbps. Os resultados de *speed-up* obtidos, bem como o desempenho relativo com a rede de 100Mbps em comparação ao com a rede de 1Gbps estão mostrados na tabela 5.12.

Tabela 5.12: Desempenho relativo para rede de 100Mbps, escoamento em torno de uma esfera, malha de tetraedros.

Tratamento	**Número de Processadores**	**Speed-up 1 Gbps**	**Speed-up 100 Mbps**	**Desempenho Relativo 100Mbps / 1 Gbps**
1RN	4	4,1	3,47	85%
	8	7,7	6,43	84%
nRN	4	4,2	3,43	82%
	8	8,1	6,77	84%

A velocidade mais baixa da rede de conexão tem um impacto sobre o desempenho um pouco maior com o tratamento nRN que com o 1RN, explicado pelo maior número de processos de comunicação entre processadores que o primeiro exige.

O speed-up médio com uma rede de 100 Mbps é 84% do speed-up obtido com a rede de 1Gbps para as configurações testadas, mostrando a validade de se formar um *cluster temporário* com os computadores disponíveis em um laboratório para a obtenção de maior desempenho computacional mesmo quando não há disponibilidade de uma estrutura de rede de alta velocidade.

5.2.4 Elementos Tetraédricos – Escoamento supersônico em torno de um modelo de avião

O escoamento supersônico em torno de um modelo de avião com a configuração canard-asa-estabilizador-fuselagem baseado no problema apresentado por Bono (2008) foi utilizado

como base para os testes de desempenho do algoritmo paralelo. A geometria básica do problema pode ser vista na figura 5.37.

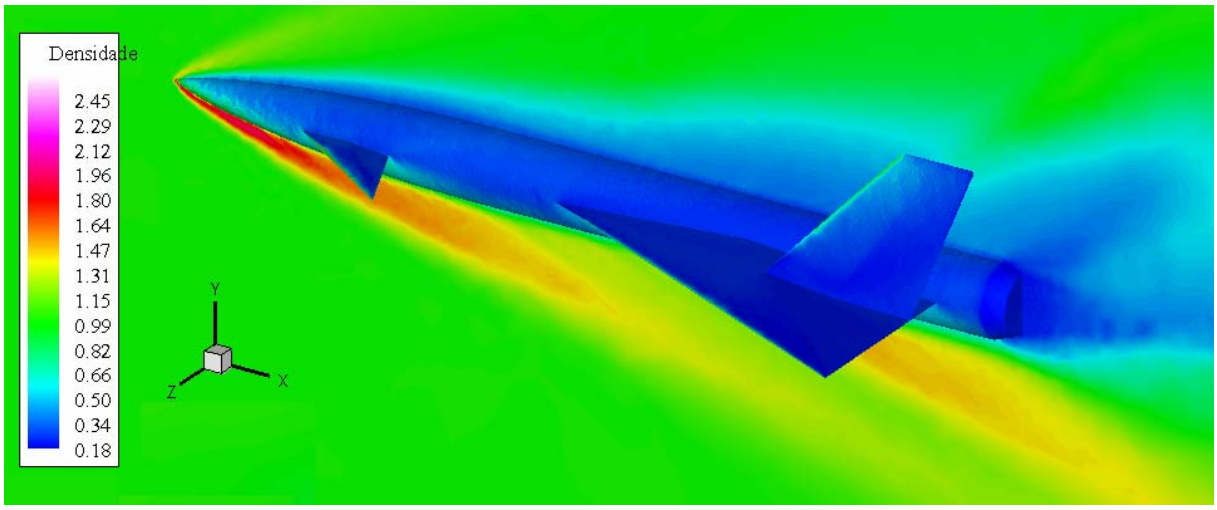


Figura 5.37: Visão geral do problema de escoamento supersônico em torno de um modelo de avião. Distribuição da densidade.

A partir de uma malha original com 242979 elementos tetraédricos lineares e 45823 nós foram obtidas duas famílias de malhas de complexidade e tamanho crescente através do processo de refinamento de malha desenvolvido por Popiolek (2006): uma para o modelo de escoamento de Euler e outra para o modelo de Navier-Stokes. Para a obtenção das malhas, não foi utilizado um processo de refinamento uniforme, no qual todos os elementos são sub-divididos da mesma forma, mas local, fazendo com que um número maior de nós e elementos se concentrem nas regiões do domínio com maiores gradientes das variáveis do problema. Uma visão geral das malhas utilizadas está mostrada na figura 5.38 e suas características estão mostradas na tabela 5.13.

A mesma metodologia empregada anteriormente foi utilizada para estimar o desempenho da solução paralela. Foi utilizado o mesmo conjunto de computadores do exemplo 5.2.2. Como as malhas empregadas são bastante grandes, a ordenação dos núcleos dos processadores da tabela 5.10 foi adotada.

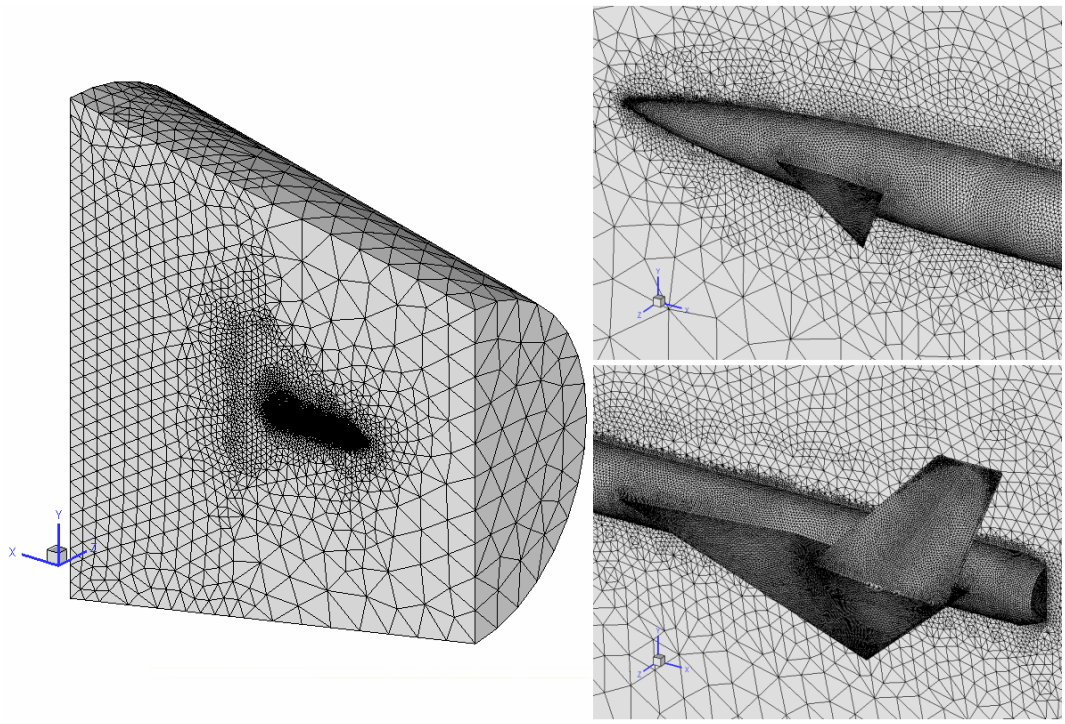


Figura 5.38: Domínio do problema de escoamento em torno de um modelo de avião e discretização utilizando elementos tetraédricos lineares.

Tabela 5.13: Características das malhas utilizadas na discretização do escoamento em torno de um modelo de avião.

Malha	Número de Elementos	Número de Nós	Graus de Liberdade
EU1 / NS1	242.979	45.823	229.115
EU2	2.093.370	364.996	1.824.980
EU3	7.397.968	1.274.645	6.373.225
NS2	1.501.912	271.842	1.359.210
NS3	7.400.765	1.322.254	6.611.270
NS4	9.654.369	1.697.705	8.488.525

As malhas referentes ao modelo de escoamento de Navier-Stokes foram executadas com o algoritmo paralelo sem nenhum tratamento, com os tratamentos 1RN e nRN. Os resultados obtidos para o *speed-up* estão mostrados na figura 5.39, e para a eficiência de paralelização na figura 5.40.

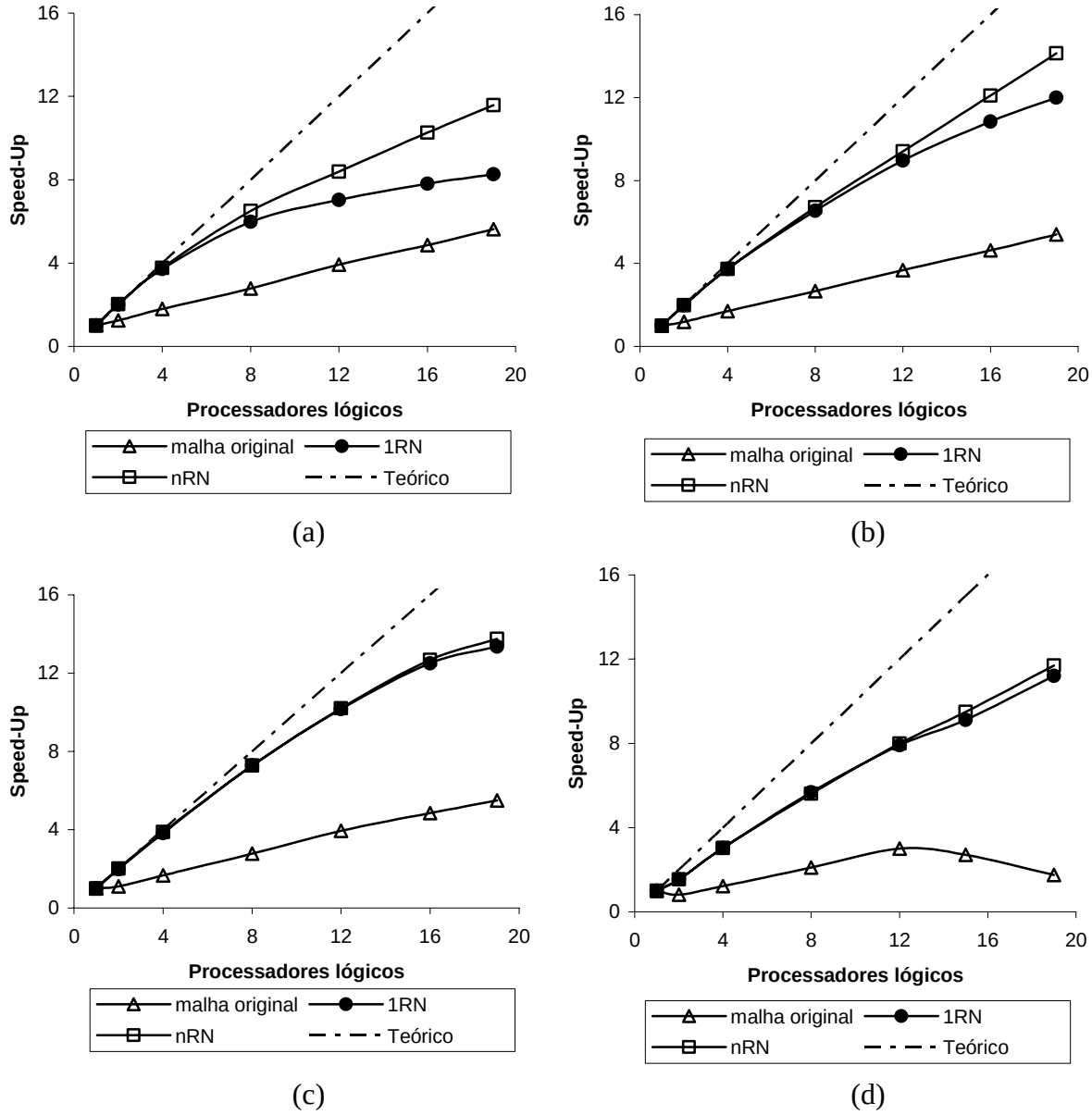


Figura 5.39: *Speed-up* obtido para o escoamento supersônico em torno de um modelo de avião, modelo de escoamento de Navier-Stokes, malhas (a) NS1, (b) NS2, (c) NS3 e (d) NS4.

Para todos os tamanhos de problema, aplicar o algoritmo paralelo sobre as malhas sem nenhum tratamento leva a um desempenho bastante baixo, com um *speed-up* máximo com o uso de 19 processadores de apenas 5,6 (30% de eficiência). O uso do tratamento nRN levou ao maior desempenho para todos os casos, em especial para problemas de tamanho pequeno a médio analisados com um número grande de processadores lógicos. Quanto maior o tamanho do problema, maior deve ser o número de processadores empregado para que o tratamento nRN apresente vantagem significativa sobre o tratamento 1RN. Para as malhas NS3 e NS4, os tratamentos foram equivalentes, aparecendo uma pequena diferença apenas com o uso de 20

processadores. O maior desempenho obtido foi com a malha NS2 empregando 20 processadores, onde foi obtido um *speed-up* de 14,1 (74% de eficiência).

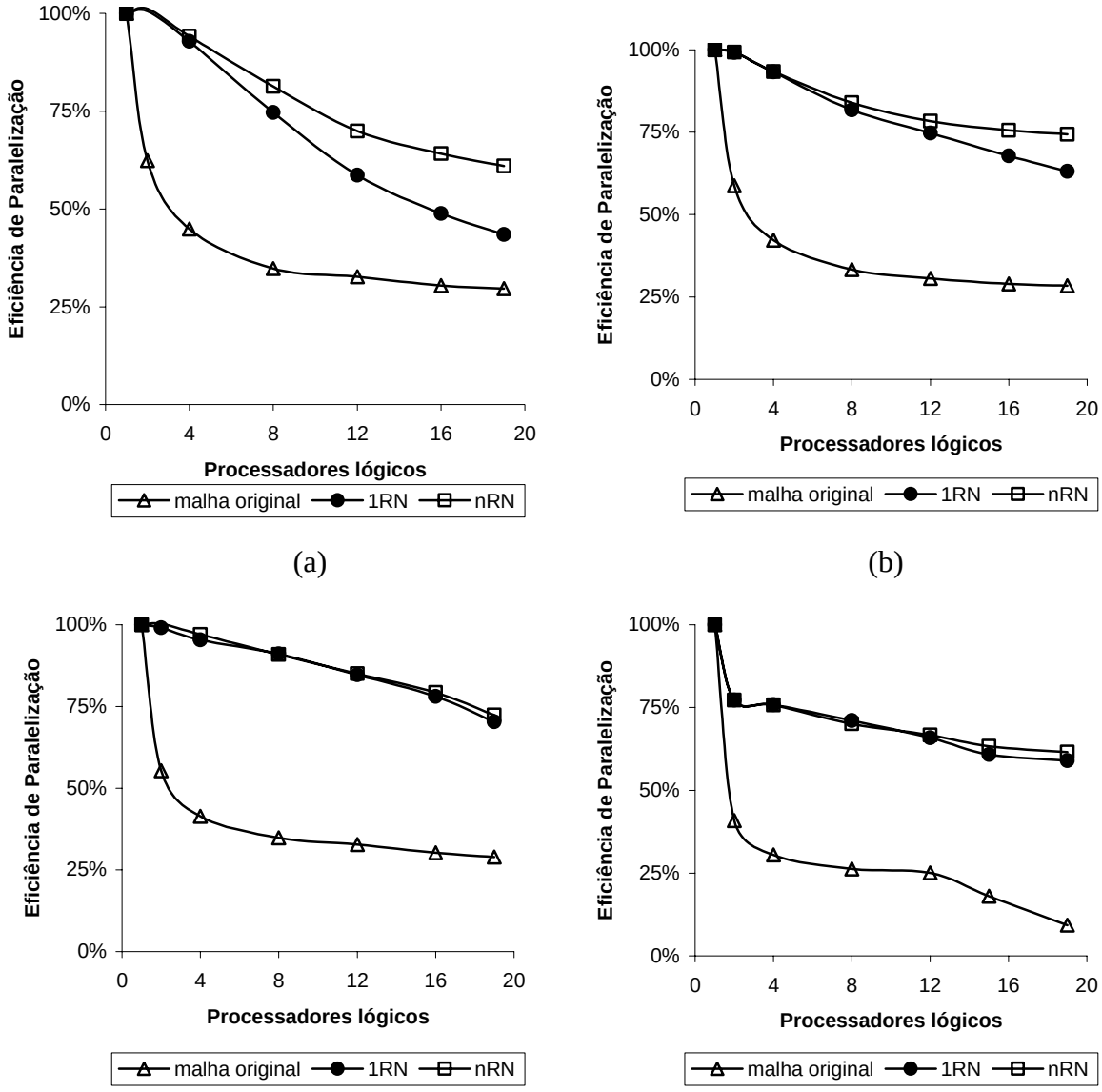


Figura 5.40: Eficiência de paralelização obtido para o escoamento supersônico em torno de um modelo de avião, modelo de escoamento de Navier-Stokes, malhas (a) NS1, (b) NS2, (c) NS3 e (d) NS4.

Os resultados para a fração serial de de Karp-Flatt estão mostrados na figura 5.41, mostrando como os tratamentos 1RN e nRN são importantes para que uma boa escalabilidade do problema seja obtida. Os valores de fração linear negativa indicam a ocorrência de *speed-up* superlinear.

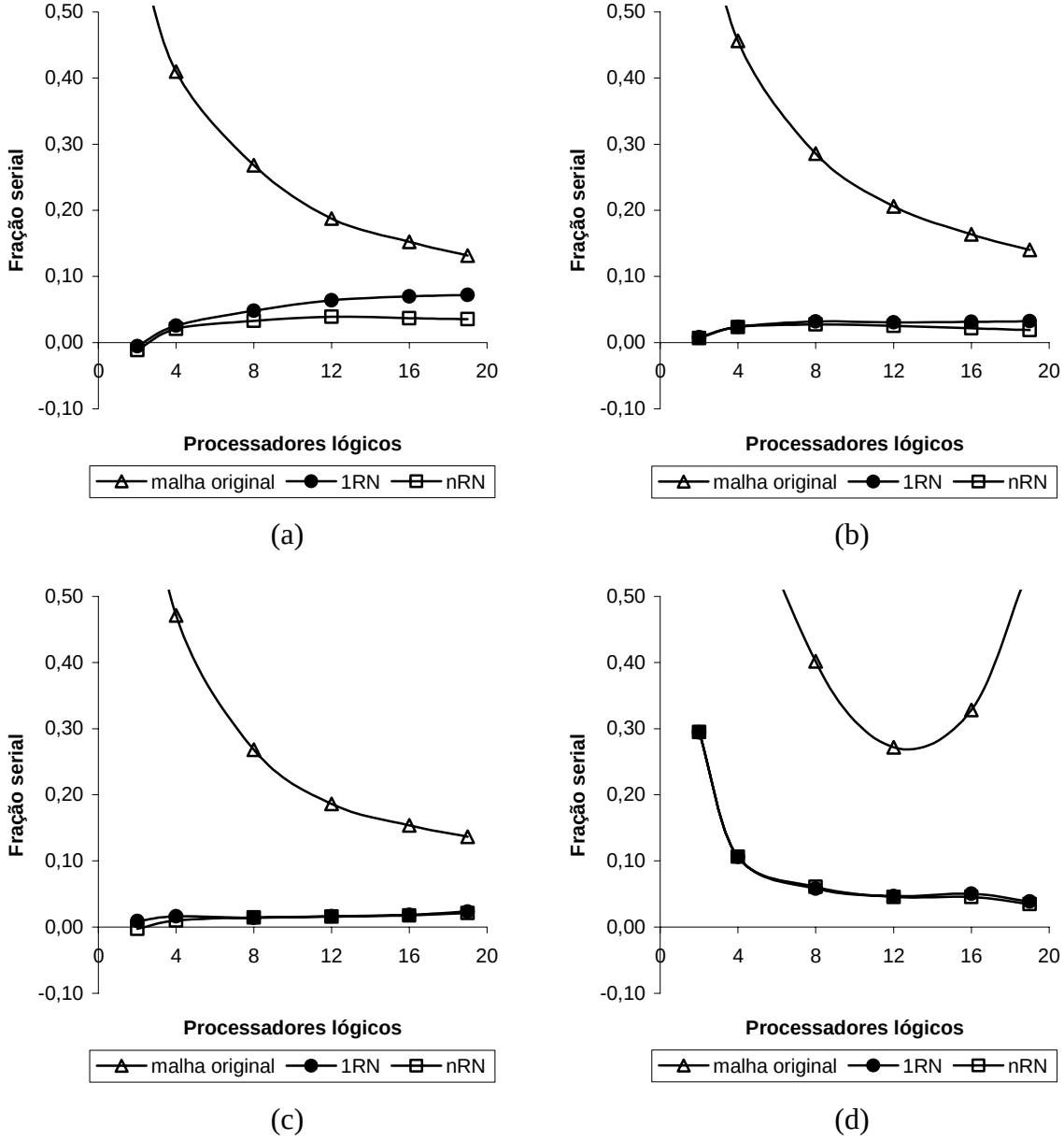


Figura 5.41: Fração serial para o escoamento supersônico em torno de um modelo de avião, modelo de escoamento de Navier-Stokes, malhas (a) NS1, (b) NS2, (c) NS3 e (d) NS4.

Para a maior das malhas (NS4) houve uma queda no desempenho para todos os tamanhos de conjuntos de processadores em relação à malha de tamanho imediatamente inferior (NS3), quando a expectativa seria de que o desempenho crescesse ou se estabilizasse com o aumento do tamanho do problema. Contudo, a malha NS4 é formada por tetraedros e obtida por refinamento de uma malha de tamanho menor. Malhas com essas características apresentam um número muito grande de elementos conectados a um mesmo nó, chegando a mais de uma centena. A divisão de tarefas nessas condições faz com que haja um número muito grande de

elementos comuns, com a conseqüente redundância do esforço computacional, afetando negativamente o desempenho do conjunto.

Para verificar a influência da velocidade da rede no desempenho da implementação paralela, fez-se um teste com a malha NS2 e os oito primeiros processadores conectados através de uma rede de 100 Mbps. Os resultados de *speed-up* obtidos, bem como o desempenho relativo com a rede de 100Mbps em comparação ao com a rede de 1Gbps estão mostrados na tabela 5.14.

Tabela 5.14: Desempenho relativo para rede de 100Mbps, escoamento em torno de um modelo de avião, malha NS2.

Tratamento	**Número de Processadores**	**Speed-up 1 Gbps**	**Speed-up 100 Mbps**	**Desempenho Relativo 100Mbps / 1 Gbps**
1RN	4	3,73	3,06	82%
	8	6,54	5,27	81%
nRN	4	3,74	2,94	79%
	8	6,71	4,84	72%

Em função da complexidade da geometria do problema, o impacto da rede mais lenta sobre o desempenho é nitidamente maior para o tratamento nRN que para o tratamento 1RN, indicando que a latência da comunicação em rede assume um papel preponderante no desempenho. Para a rede mais lenta, o tratamento 1RN parece ser o mais adequado

As malhas referentes ao modelo de escoamento de Euler foram executadas com o algoritmo paralelo sem nenhum tratamento e sem balanceamento das cargas de trabalho, com os tratamentos 1RN e nRN mas sem balanceamento, e com os dois tratamentos com balanceamento. Os resultados obtidos para o *speed-up* obtido estão mostrados na figura 5.42, e para a eficiência de paralelização na figura 5.43.

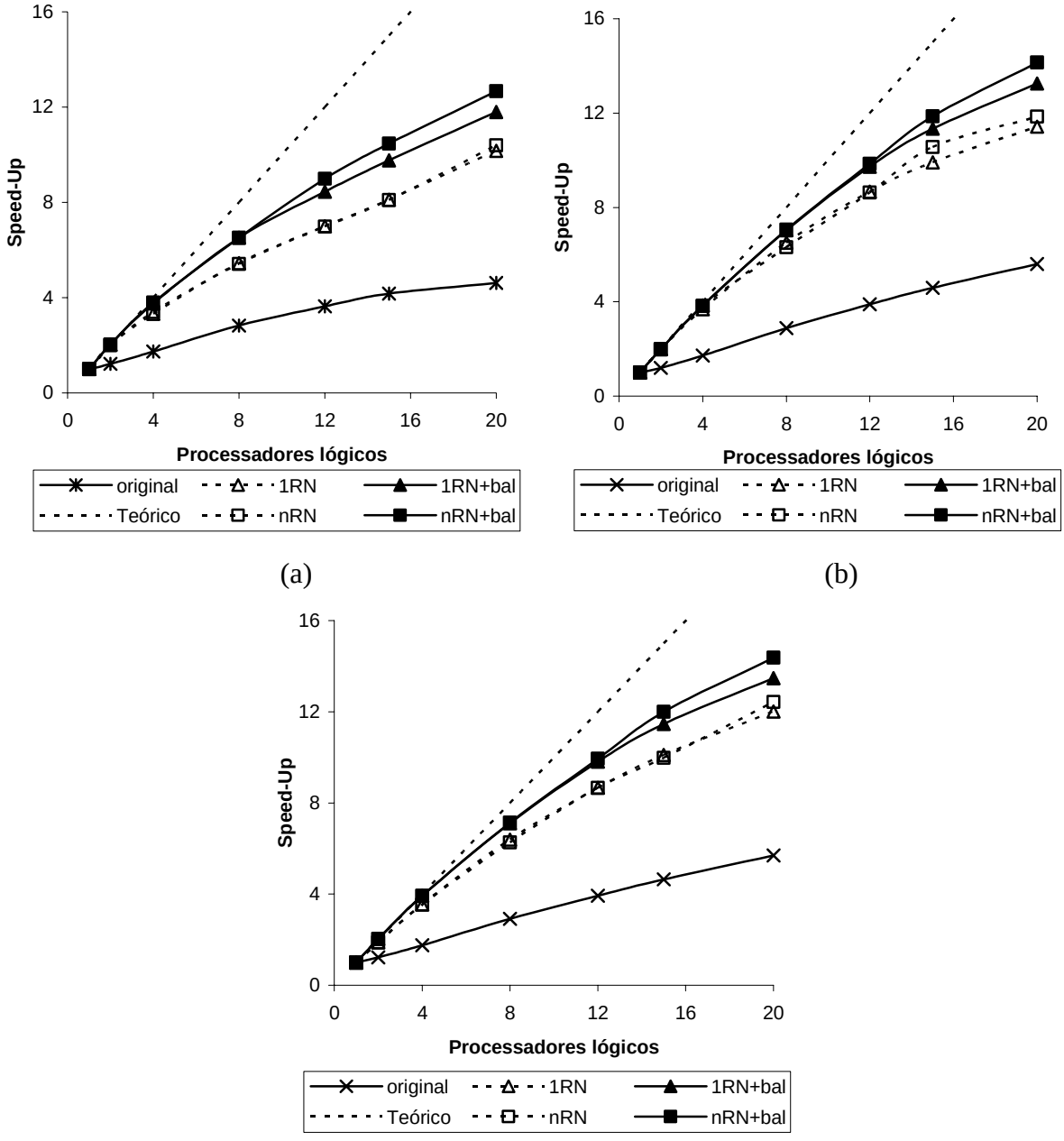


Figura 5.42: *Speed-up* obtido para o escoamento supersônico em torno de um modelo de avião, modelo de escoamento de Euler, malhas (a) EU1, (b) EU2 e (c) EU3.

Mesmo com as cargas de trabalho iniciais sendo distribuídas proporcionalmente à freqüência de cada processador lógico, o balanceamento das mesmas representou ganhos de até 22% no *speed-up* obtido, ressaltando tanto a não uniformidade espacial da aplicação das condições de contorno e de cargas como a não uniformidade do graus de conexão dos nós (número de elementos a ele conectados).

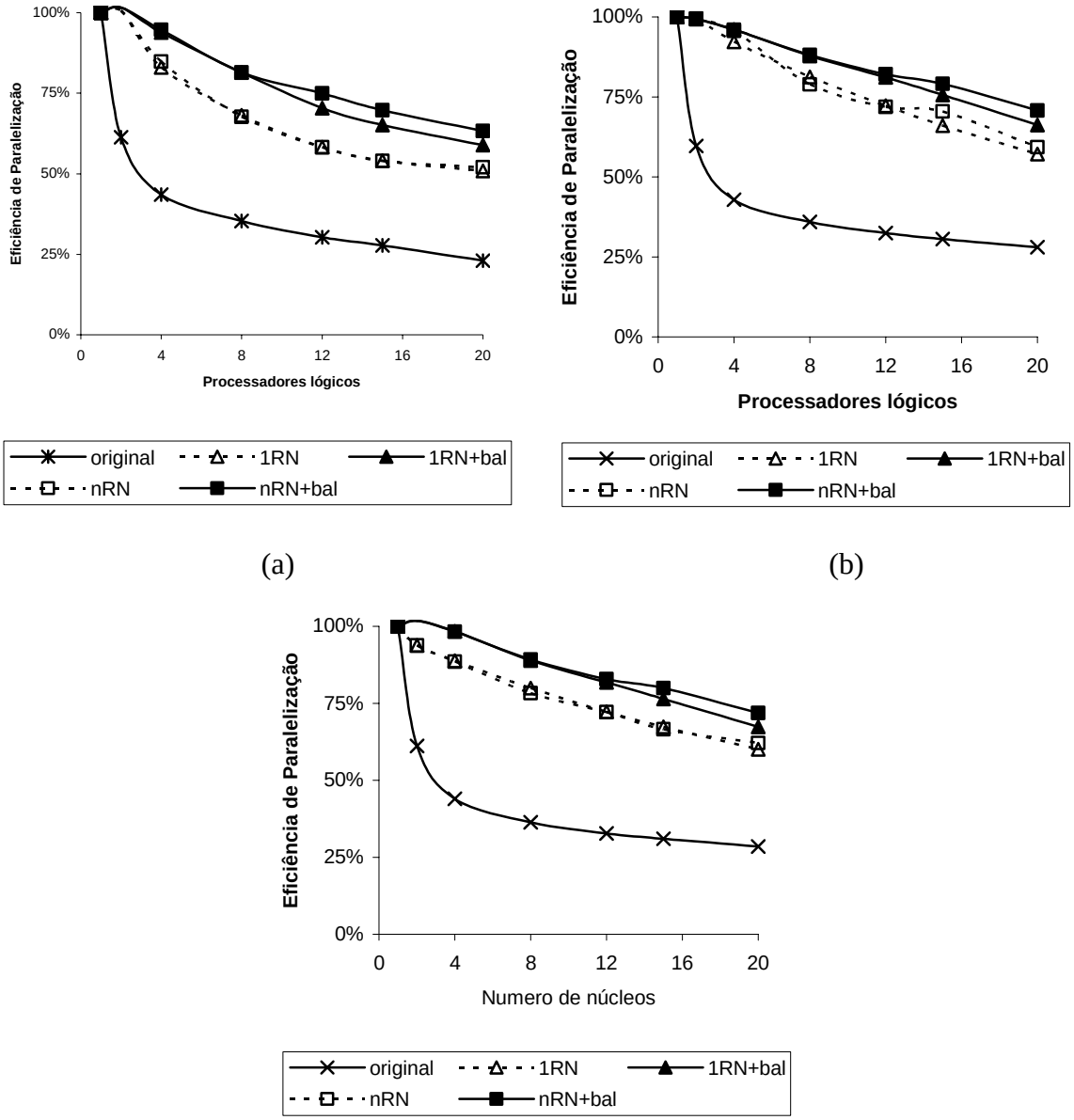


Figura 5.43: Eficiência de paralelização obtido para o escoamento supersônico em torno de um modelo de avião, modelo de escoamento de Euler, malhas (a) EU1, (b) EU2 e (c) EU3.

Também para as malhas com o modelo de escoamento de Euler, a aplicação do algoritmo paralelo sobre as malhas sem nenhum tratamento resultou num desempenho bastante baixo, com um *speed-up* máximo de 5,7 com 20 processadores (28% de eficiência). Para todos os tamanhos de problema, os melhores resultados foram obtidos com o tratamento nRN em conjunto com o balanceamento das cargas de trabalho, em especial quando eram utilizados mais de 12 processadores. Com o uso de 20 processadores, o emprego do tratamento nRN

com balanceamento resultou em um desempenho 7% superior ao do tratamento 1RN com balanceamento, atingindo um *speed-up* máximo de 14,4 (eficiência de 72%).

Os valores obtidos para a eficiência de paralelização do modelo de escoamento de Euler são muito próximos dos obtidos para o modelo de escoamento de Navier-Stokes, com uma leve vantagem para o último, uma vez que o modelo de Navier-Stokes exige mais esforço computacional para a avaliação das matrizes, mas o mesmo tempo de comunicação em rede.

Os resultados para a fração serial de Karp-Flatt estão mostrados na figura 5.44, mostrando novamente como os tratamentos 1RN e nRN são importantes para que uma boa escalabilidade do problema seja obtida. Os valores de fração linear negativa indicam a ocorrência de *speed-up* superlinear.

Considerando os problemas com geometria mais complexa (veículo espacial para hexaedros, modelo de avião para tetraedros), uma comparação entre o desempenho do algoritmo paralelo para os dois tipos de elementos pode ser feita tomando-se, para cada malha empregada, qual o maior valor de *speed-up* obtido e a eficiência de paralelização correspondente. Os dados correspondentes estão mostrados na tabela 5.15, onde GL indica o número de graus de liberdade do problema, NP o número de processadores empregados para a obtenção do *speed-up* máximo, *tetra* refere-se a elementos tetraédricos e *hexa* a hexaédricos.

Comparando-se os valores obtidos na tabela 5.15, percebe-se a nítida diferença existente entre a eficiência do algoritmo aplicado a malhas de hexaedros e a de tetraedros. As implementações são idênticas, diferindo apenas na forma de avaliação das matrizes. O fato de uma malha de hexaedros ter de ser estruturada faz com que haja uma constância muito maior quanto ao número de elementos conectados a cada nó ao longo da malha do que em uma malha não estruturada de tetraedros. Isso faz com que a premissa de que a carga de trabalho computacional associada aos nós é constante, utilizada no processo de divisão de tarefas, aproxime-se muito mais da condição existente nas malhas de hexaedros, tornando a divisão de tarefas mais eficiente e menos dependente do balanceamento das cargas.

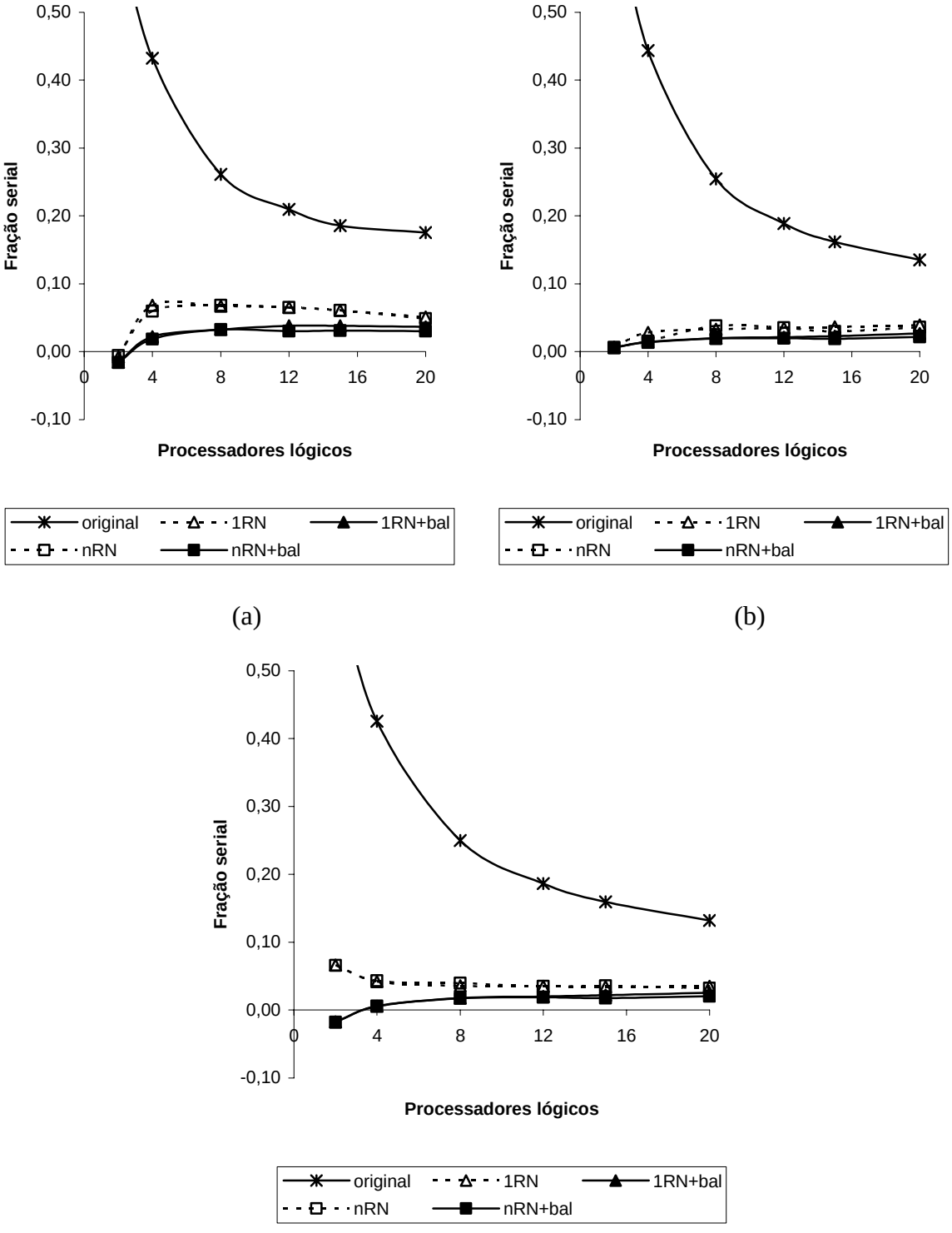


Figura 5.44: Fração serial obtida para o escoamento supersônico em torno de um modelo de avião, modelo de escoamento de Euler, malhas (a) EU1, (b) EU2 e (c) EU3.

Tabela 5.15: Valores máximos de *speed*-up e a eficiência computacional correspondente para as malhas de hexaedros e tetraedros utilizadas na simulação de escoamentos em torno de um veículo espacial e de um modelo de avião.

Malha	**Tipo**	**GL**	**Elementos**	***Speed-up* máximo**	**NP**	**Eficiência**
VE1	*hexa*	1.055.730	198.750	17,08	20	85%
VE2	*hexa*	4.746.060	917.700	17,82	20	89%
EU1	*tetra*	229.115	242.979	12,70	20	61%
EU2	*tetra*	1.824.980	2.093.370	14,10	20	74%
EU3	*tetra*	6.373.225	7.397.968	14,40	20	72%
NS1	*tetra*	229.115	242.979	11,60	19	62%
NS2	*tetra*	1.359.210	1.501.912	14,10	19	63%
NS3	*tetra*	6.611.270	7.400.765	13,70	19	71%
NS4	*tetra*	8.488.525	9.654.369	11,70	19	72%

Para as malhas não estruturadas de tetraedros, o grande número de elementos conectados a um nó faz com que a redundância no esforço computacional quando da divisão de tarefas seja grande, além de aumentar a quantidade de dados a ser transmitida pela rede em comparação com as malhas de hexaedros. A premissa de que a carga de trabalho computacional associada a cada nó é constante ao longo da malha é, em muitos casos, bastante distante da condição existente, fazendo com que o balanceamento das cargas de trabalho sejam de grande importância para obtenção do desempenho ótimo. Além disso, a grande quantidade de elementos conectados a um mesmo nó faz com que a informação de um grande número de nós seja necessária para que as equações nodais desse nó sejam resolvidas em um dado passo de tempo ou iteração. Divisões de tarefas com um tratamento como o nRN faz com que seja comum a troca de dados de cada processador presente no conjunto com todos os outros, tornando mais importante o correto ordenamento das comunicações entre processadores para que não haja perda de eficiência na velocidade de processamento do conjunto.

5.2.5 Elementos Tetraédricos – Escoamento transônico em torno de uma asa delta usando a técnica de sub-ciclos

Para teste do algoritmo paralelo para escoamentos compressíveis de um passo com iterações utilizando a técnica de sub-ciclos para integração no tempo, foi utilizado o problema de escoamento transônico ao redor de uma asa delta apresentada por Bono (2008) e representada na figura 5.45. A malha utilizada tem 613.077 elementos tetraédricos, 110.725 nós e 553.625 graus de liberdade.

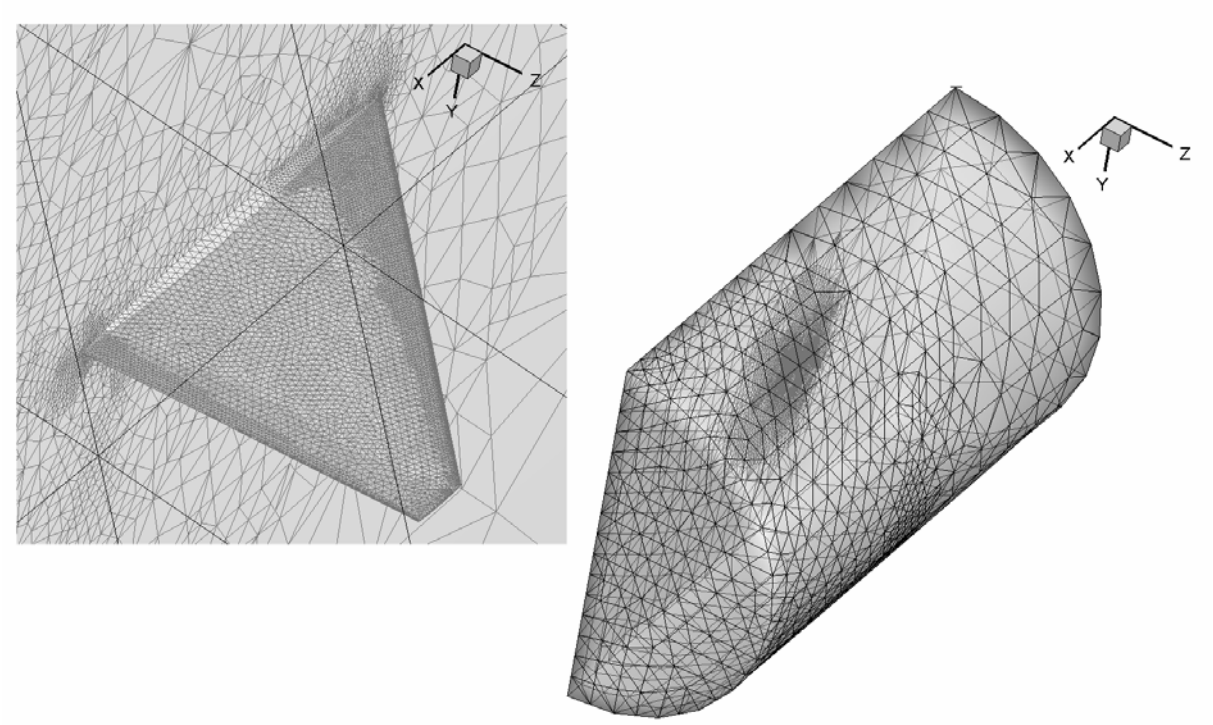


Figura 5.45: Malha utilizada na análise paralela do escoamento transônico ao redor de uma asa delta com a técnica de sub-ciclos.

Para a aplicação do algoritmo com a técnica de sub-ciclos, os elementos da malha são divididos em grupos cujo valor do passo de tempo empregado é um múltiplo 2^n do valor do menor passo de tempo, correspondente ao menor elemento da malha. Na tabela 5.16 estão mostrados os grupos de passos de tempo utilizados, o percentual de elementos em cada grupo e o valor do *speed-up* teórico da técnica de sub-ciclos.

Tabela 5.16: Distribuição dos elementos da malha em função do passo de tempo mínimo a ser empregado Escoamento transônico em torno de uma asa delta.

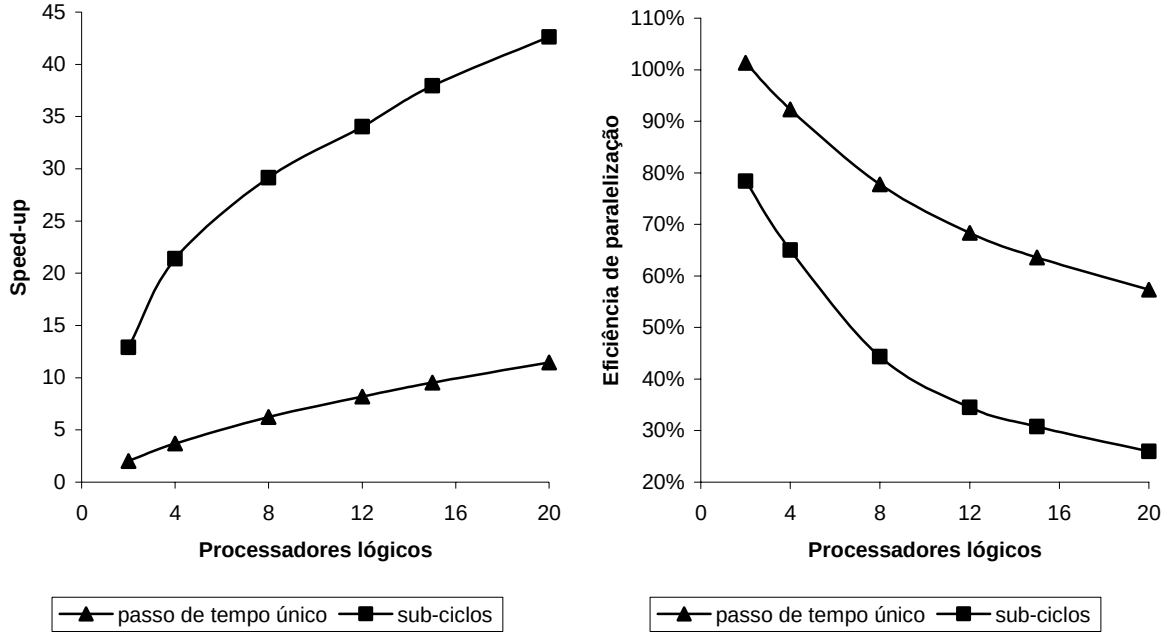
Grupo	**% de elementos**	**Grupo**	**% de elementos**
1 Δt	0,06%	2 Δt	0,04%
4 Δt	2,13%	8 Δt	21,71%
16 Δt	30,91%	32 Δt	17,71%
64 Δt	12,40%	128 Δt	4,45%
256 Δt	10,53%		
***Speed-up* de sub-ciclos teórico**			**16,4**
***Speed-up* de sub-ciclos obtido**			**8,22**

Para que o ganho de desempenho do algoritmo paralelo possa ser avaliado, o código seqüencial foi utilizado com a malha do problema com um único passo de tempo e com sub-ciclos. A partir dos tempos de processamento obtidos, é possível avaliar o desempenho da versão paralela. O valor de *speed*-up efetivamente obtido para a técnica de sub-ciclos com o algoritmo seqüencial foi de 8,22. Os valores de *speed-up* e eficiência de paralelização para o algoritmo paralelo são mostrados na figura 5.46(a).

Se o ganho de velocidade de processamento obtido com a técnica de sub-ciclos foi de 8,22 vezes, o *speed-up* teórico da versão paralela com sub-ciclos é dada pelo produto do *speed-up* teórico de paralelização (igual ao número de processadores utilizado) com o *speed-up* obtido com sub-ciclos, ou seja 20 x 8,22 = 164,4 para 20 processadores. O valor obtido para cada número de processadores foi utilizado como base para o cálculo da eficiência de paralelização mostrada na figura 5.46(b).

Apesar do *speed-up* obtido ser bastante alto em termos absolutos (até 43 vezes mais rápido que o código original) a eficiência de paralelização obtida com o uso de sub-ciclos é bastante mais baixa que com o uso de um passo de tempo único. Como com a técnica de sub-ciclos durante uma parcela considerável da análise os valores das variáveis nodais de vários nós são apenas interpolados, o esforço computacional é consideravelmente mais baixo que com o algoritmo original (8,22 vezes), mas a quantidade de dados a ser comunicada entre os diversos processadores continua a mesma. Isso faz com que, proporcionalmente ao tempo gasto com

processamento, o tempo de comunicação de dados pela rede se torne maior, baixando a eficiência do conjunto.



(a) (b)

Figura 5.46: (a) *Speed-up* e (b) eficiência de paralelização para o algoritmo paralelo com a técnica de sub-ciclos para na análise paralela do escoamento transônico ao redor de uma asa delta.

Além disso, a divisão de tarefas nunca é ótima, pois o esforço computacional varia ao longo dos passos de tempo de um ciclo, de modo que uma divisão de tarefas inicial baseada na premissa de que o esforço computacional é constante para todos os nós deixa de ser verdadeira espacialmente e temporalmente, pois para cada passo de tempo o esforço é diferente. Mesmo o balanceamento das cargas de trabalho não consegue equilibrar perfeitamente o esforço, pois ele é feito sobre o tempo de processamento gasto para um determinado número de passos de tempo (pelo menos 1 ciclo completo). Na média há um equilíbrio, mas isso não significa que em um determinado passo de tempo um processador tenha uma grande carga de trabalho enquanto outro fica esperando que o processamento do primeiro termine para que haja a comunicação de dados, e que no passo seguinte a situação não se inverta.

5.3 ESCOAMENTOS INCOMPRESSÍVEIS EM REGIME SUBSÔNICO

Para os testes do algoritmo paralelo explícito de Taylor-Galerkin de integração no tempo de dois passos para a solução de problemas de escoamentos incompressíveis tridimensionais, empregou-se uma implementação com os mesmos elementos hexaédricos lineares com integração reduzida utilizado no esquema de um passo com iterações. Maiores detalhes podem ser encontrados em Burbridge (1999).

Os computadores utilizados nas simulações foram os mesmos do exemplo 5.2.2, seguindo a ordem indicada na tabela 5.10. A fim de verificar a influência dos tratamentos da malha e do balanceamento das cargas de trabalho sobre o desempenho, em todos os exemplos a implementação paralela do algoritmo de solução foi empregada com 4 configurações:

a) a malha original sem nenhum tratamento e sem realizar o balanceamento das cargas de trabalho, usando apenas a divisão inicial de tarefas baseada na freqüência dos processadores;

b) a malha original sem nenhum tratamento, mas com o balanceamento das cargas de trabalho;

c) o tratamento 1RN com balanceamento das cargas de trabalho;

d) o tratamento nRN com balanceamento das cargas de trabalho.

5.3.1 Cavidade de fundo flexível

Este exemplo simula o escoamento bidimensional em uma cavidade com paredes laterais rígidas e um fundo flexível. O escoamento do fluído é perturbado pelo movimento do fundo. Maiores detalhes podem ser encontrados em Foster et al. (2007) e em Braun (2009). Para os testes do esquema explícito de escoamento, o fundo foi considerado rígido. No exemplo 6.4.1 o problema completo, com o fundo flexível, será utilizado para os testes de desempenho do algoritmo paralelo para interação fluído-estrutura.

Foi utilizada a malha da figura 5.47 com 2.500 elementos hexaédricos lineares e 5.202 nós, correspondendo a 20.808 graus de liberdade. Como o escoamento é bidimensional, foi considerado apenas um elemento na direção da espessura (eixo z). Os resultados obtidos para o *speed-up* e para a eficiência de paralelização podem ser vistos na figura 5.48.

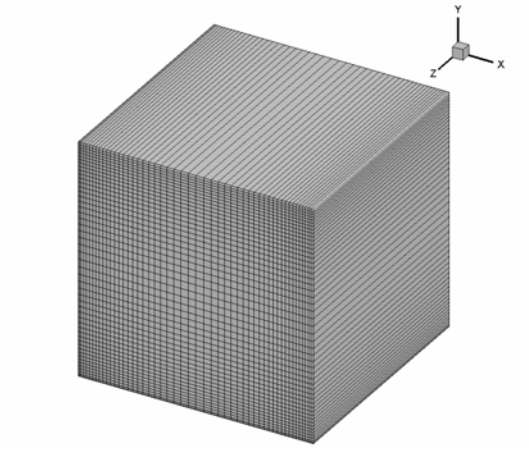
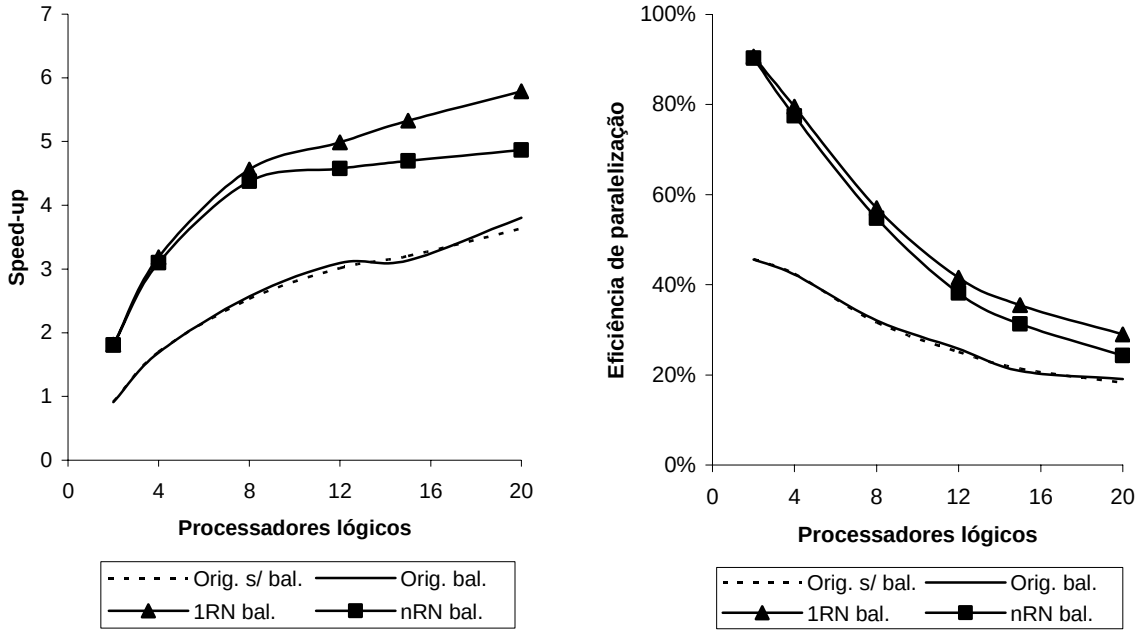


Figura 5.47: Malha utilizada na análise do escoamento em cavidade.



(a) (b)

Figura 5.48: (a) *Speed-up* e (b) eficiência de paralelização para o algoritmo paralelo do esquema explícito de dois passos na análise do escoamento em cavidade.

Em função da malha ser constituída por um número muito pequeno de nós e elementos, o *speed –up* obtido para um número grande de processadores é relativamente baixo, atingindo 5,8 para 20 processadores no melhor dos casos.

A malha apresenta uma distribuição de esforço computacional por nó praticamente uniforme, uma vez que o processo de balanceamento não trouxe diferenças visíveis para o desempenho. Na comunicação de dados entre os processadores, a latência da rede teve uma influência maior que a taxa de transferência. Em função disso, o tratamento nRN, que reduz a

quantidade de dados a serem comunicados entre processadores mas implica em um número maior de trocas de dados que o tratamento 1RN, apresentou um desempenho pior que o último.

Como termo de comparação de desempenho, Petry (2002) resolve um problema semelhante de escoamento bidimensional em uma cavidade quadrada, com uma malha em torno de 6 vezes maior, e obteve uma velocidade de processamento de 8,69 x 10^{-6} s por passo de tempo por iteração, obtendo uma taxa de aproximadamente 700 Mflops sustentada em um supercomputador Cray T94 utilizando um único processador. O melhor resultado obtido com o *cluster* empregado foi de 3,63 x 10-7 s por passo de tempo por iteração para 20 computadores com tratamento de malha 1RN e carga balanceada. Apesar dos códigos serem diferentes, com implementações totalmente distintas (o Cray usa vetorização, o *cluster* usa paralelização), e o tamanho do problema ser diferente (uma malha maior levaria a melhores resultados de *speed*-up mas também a uma maior carga atribuída ao *cache* do processador), a comparação pode ser utilizada como ordem de grandeza do desempenho relativo. Se a comparação direta pudesse ser realizada, o *cluster* seria aproximadamente 24 vezes mais rápido que o Cray, com uma taxa equivalente de 16,8 Gflops sustentada.

5.3.2 Escoamento bidimensional em torno de uma edificação prismática

Este exemplo simula o escoamento do vento natural sobre uma edificação prismática de parede e tetos rígidos, havendo desenvolvimento de camada limite do terreno e geração e desprendimento de vórtices. Maiores detalhes são apresentados por Braun (2007).

A malha emprega elementos tridimensionais para simular um escoamento bidimensional, tendo somente um elemento na direção da espessura (eixo z). Uma visão geral da malha utilizada pode ser vista na figura 5.49. Um total de 22.368 elementos e 45.558 nós foram utilizados na discretização, correspondendo a 182.232 graus de liberdade. Os resultados obtidos para o *speed-up* e para a eficiência de paralelização podem ser vistos na figura 5.50.

Sendo uma malha maior que a do exemplo 5.3.1, o tempo de comunicação de dados pela rede não impõe uma perda de desempenho tão grande quanto naquele exemplo. O balanceamento das cargas de trabalho representa um ganho de aproximadamente 5% no *speed-up* máximo obtido. Em função do esquema de geração utilizado, a malha original apresentava uma ordenação de nós bastante ruim para ser base para divisão de tarefas. Os ordenamentos 1RN e

nRN produziram melhoras significativas no desempenho, resultando para o *speed-up* máximo ganhos de 71% e 85%, respectivamente.

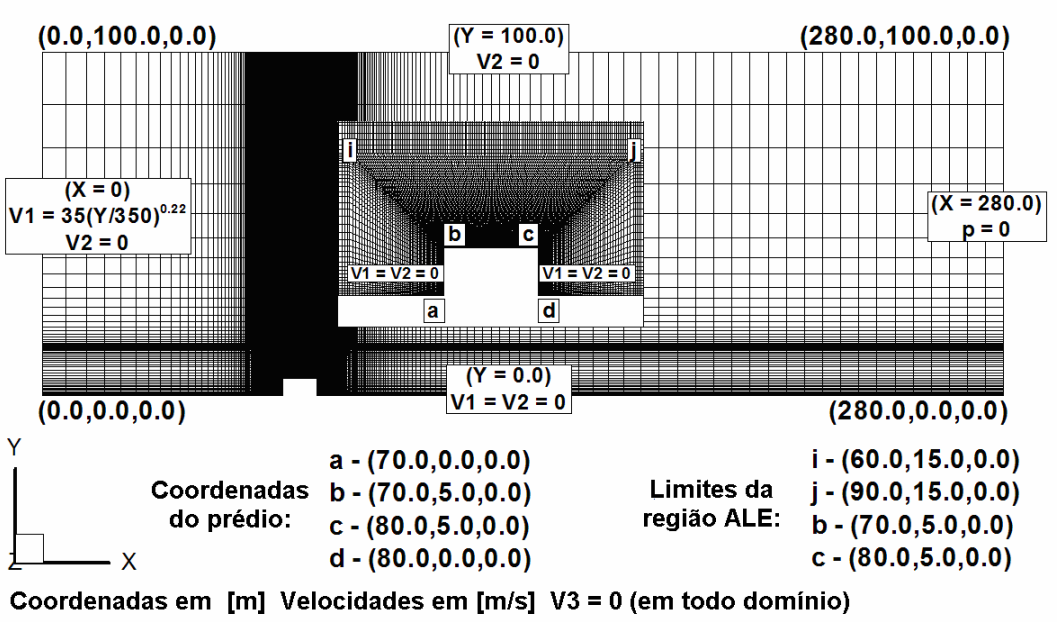
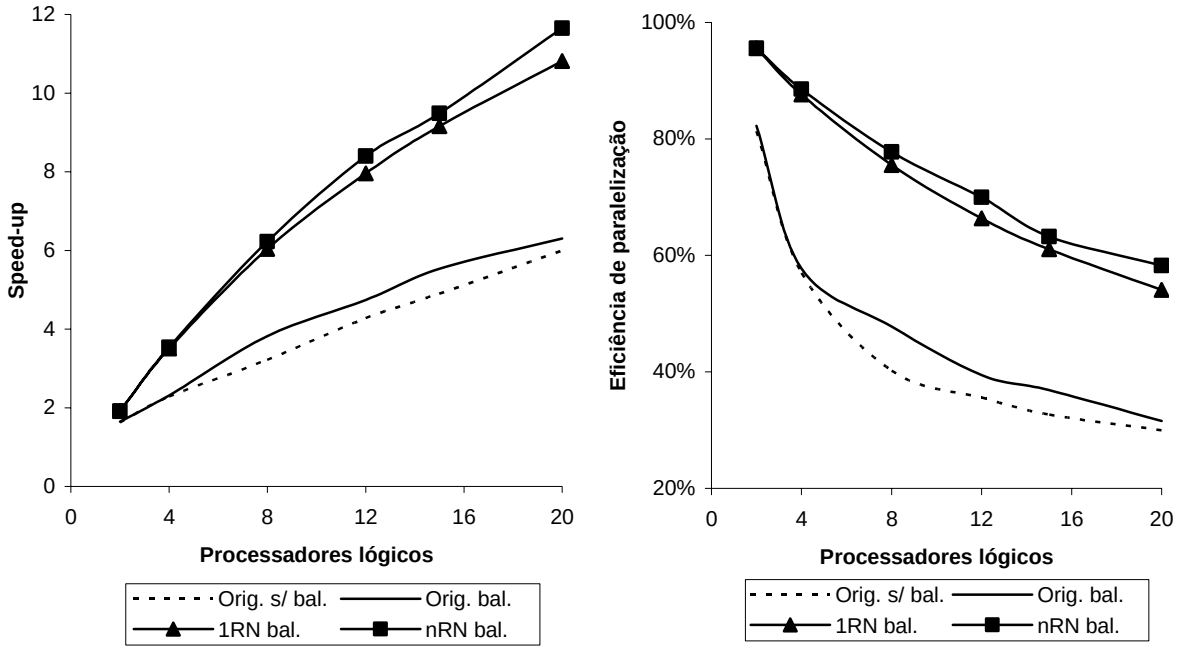


Figura 6.49: Malha utilizada na análise do escoamento bidimensional em torno de uma edificação prismática. *Fonte: Braun, 2007.*



(a) (b)

Figura 5.50: (a) *Speed-up* e (b) eficiência de paralelização para o algoritmo paralelo do esquema explícito de dois passos na análise do escoamento em torno de uma edificação prismática.

O uso do tratamento nRN com balanceamento das cargas de trabalho foi o mais vantajoso. Pelo tamanho da malha, a menor quantidade de dados a serem comunicados através da rede pelos processadores lógicos compensou, em tempo de processamento total, a necessidade de um número maior de comunicações entre os processadores que o que é necessário com o uso do tratamento 1RN.

5.3.3 Escoamento tridimensional em torno de uma edificação prismática

Este exemplo simula os estudos experimentais em túnel de vento conduzidos por Akins et al. (1977) no Laboratório de Dinâmica dos Fluídos e Difusão da Universidade do Estado do Colorado (EUA). O modelo é caracterizado por uma seção retangular com uma razão altura / largura igual a dois, submetido a um escoamento de camada limite atmosférica com ângulo de ataque igual a zero para obtenção dos coeficientes aerodinâmicos de um edifício prismático imerso no escoamento. Maiores detalhes podem ser encontrados em Braun (2007).

Uma visão geral do domínio computacional e das condições de contorno utilizadas na análise estão mostradas na figura 6.51. A malha empregada pode ser vista na figura 5.52, sendo composta por 368.800 elementos e 383.613 nós, totalizando 1.534.452 graus de liberdade.

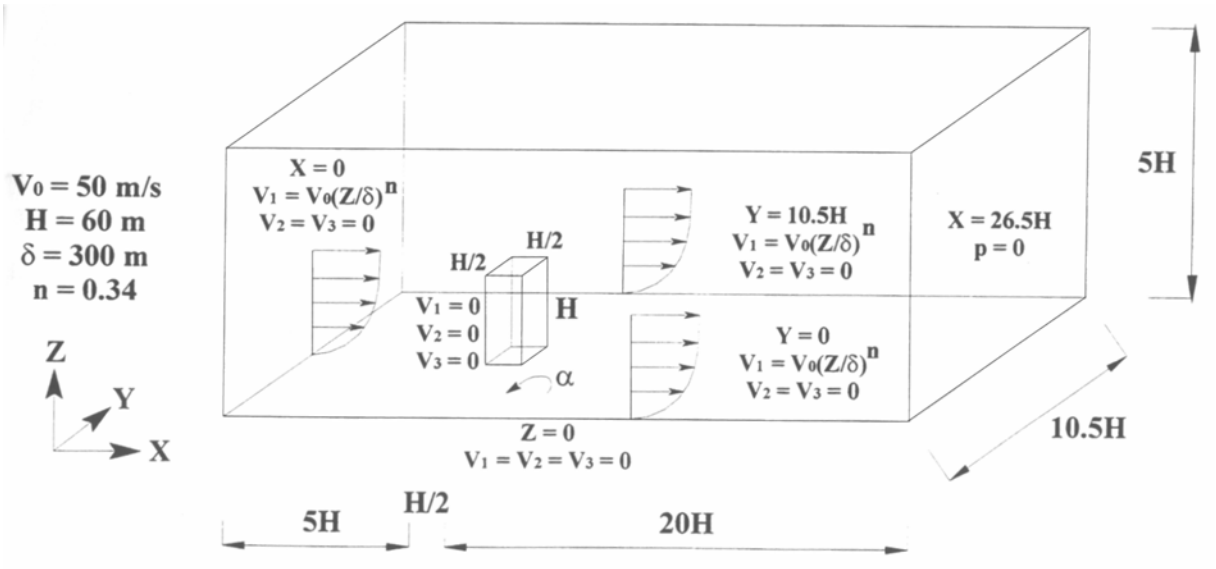


Figura 5.51: Domínio computacional e condições de contorno do escoamento tridimensional em torno de uma edificação prismática. *Fonte: Braun (2007)*

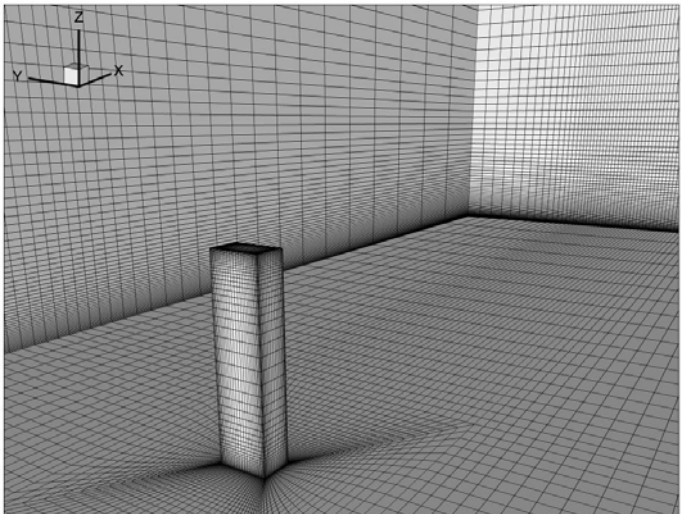
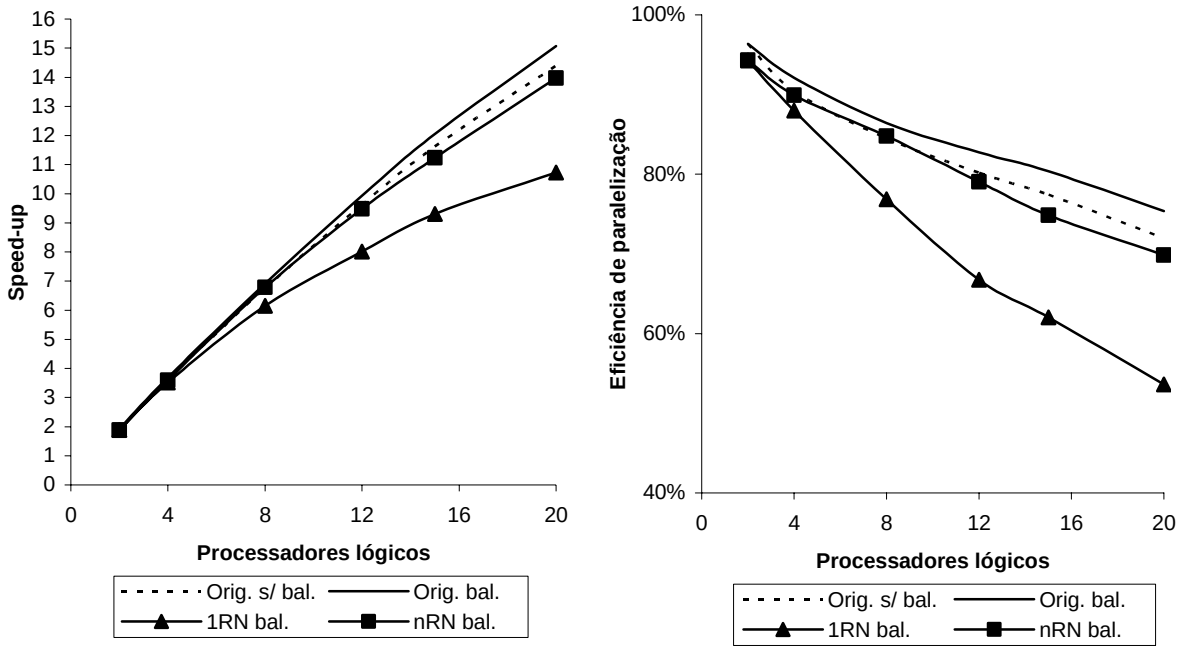


Figura 5.52: Malha utilizada na análise do escoamento tridimensional em torno de uma edificação prismática. *Fonte: Braun (2007)*

Os resultados de *speed-up* e eficiência de paralelização obtidos com o algoritmo paralelo estão mostrados na figura 5.53. O tamanho da malha utilizada permitiu que se obtivesse valores de *speed-up* bastante altos, alcançando 15,1 com o uso de 20 processadores lógicos. O balanceamento das cargas de trabalho foi responsável por uma ganho de aproximadamente 5% no desempenho.



(a) (b)

Figura 5.53: (a) *Speed-up* e (b) eficiência de paralelização para o algoritmo paralelo do esquema explícito de dosi passos para na análise do escoamento em cavidade.

Para este exemplo, os tratamentos 1RN e nRN resultaram em diminuição da velocidade de processamento. De todos os exemplos analisados para os vários algoritmos, foi o único em que isso ocorreu. O tratamento nRN resultou em um desempenho bastante próximo do obtido com a malha sem reordenação, mais ainda assim com um desempenho inferior. Uma provável explicação é que o gerador de malha utilizado empregue um reordenador nodal de minimização de banda mais eficiente que o empregado nos tratamentos 1RN e nRN.

5.4 INTERAÇÃO FLUÍDO-ESTRUTURA

No algoritmo paralelo para análise de problemas de interação fluído estrutura, utilizou-se, para a parte do fluído, o esquema explícito de dois passos, e para a solução do sistema de equações resultante da parte da estrutura, o Método dos Gradientes Conjugados. A análise dinâmica foi feita pelo método de Newmark no contexto α-generalizado. A implementação foi feita para problemas tridimensionais empregando os elementos hexaédricos lineares com integração reduzida utilizados nos testes anteriores para o Método dos Gradientes Conjugados e para escoamentos subsônicos incompressíveis.

Os exemplos utilizados nos testes foram os mesmos empregados no escoamento subsônico, mas com uma estrutura deformável imersa no escoamento ao invés de um contorno sólido.

O primeiro exemplo analisado é o de escoamento em uma cavidade de fundo flexível, substituindo-se o fundo da cavidade do exemplo 5.3.2 por uma membrana deformável, como mostra a figura 5.54. A malha para a estrutura é formada por 108 elementos e 296 nós (888 graus de liberdade).

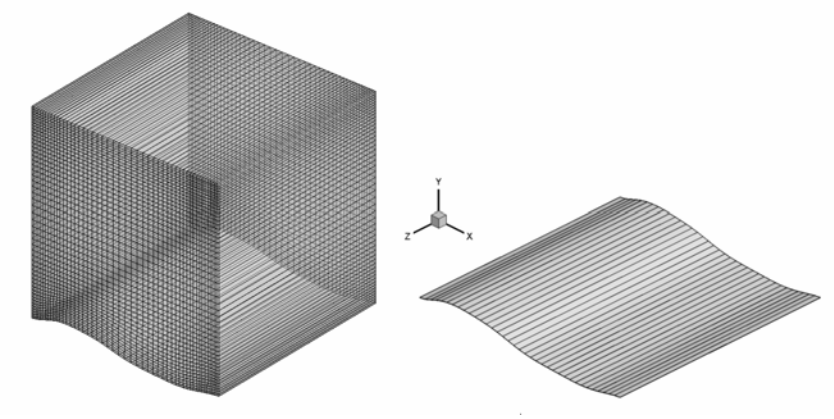


Figura 5.54: Malhas para o fluído e para a estrutura usadas na simulação do problema de escoamento em cavidade com fundo flexível. *Fonte: Braun (2009)*

O segundo exemplo é o de um escoamento bidimensional em uma edificação prismática, substituindo-se o teto da edificação, considerado rígido no exemplo 6.3.2, por uma membrana flexível discretizada por uma malha com 600 elementos e 1510 nós (4530 graus de liberdade), mostrada na figura 5.55.

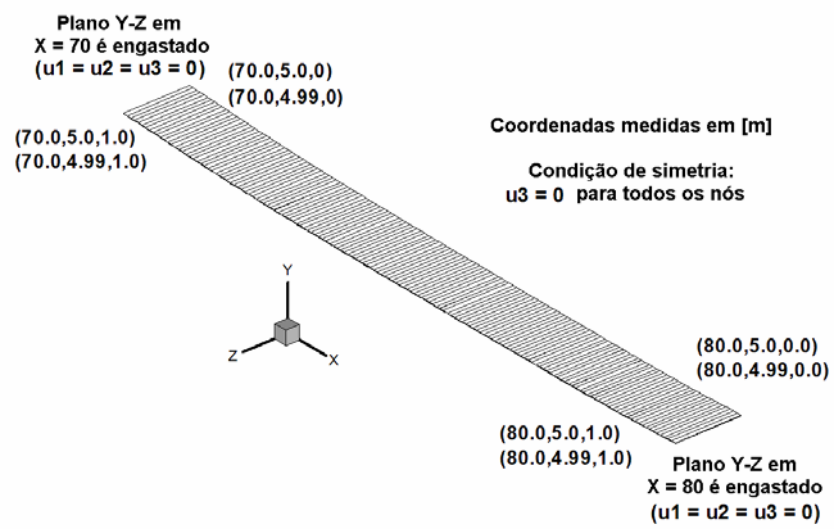


Figura 5.55: Malha utilizada na discretização do teto da edificação sob escoamento bidimensional. *Fonte: Braun (2007).*

No último exemplo, o prédio prismático imerso em um escoamento tridimensional deixa de ser um objeto rígido e passa a ser flexível, sendo discretizado por uma malha com 1000 elementos e 1404 nos (4212 graus de liberdade), como mostrado na figura 5.56.

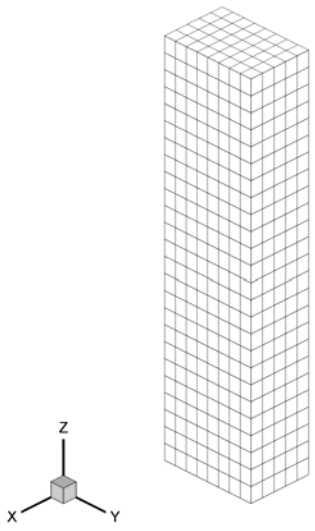


Figura 5.56: Malha utilizada na discretização do edifício imerso em um escoamento tridimensional. *Fonte: Braun (2007).*

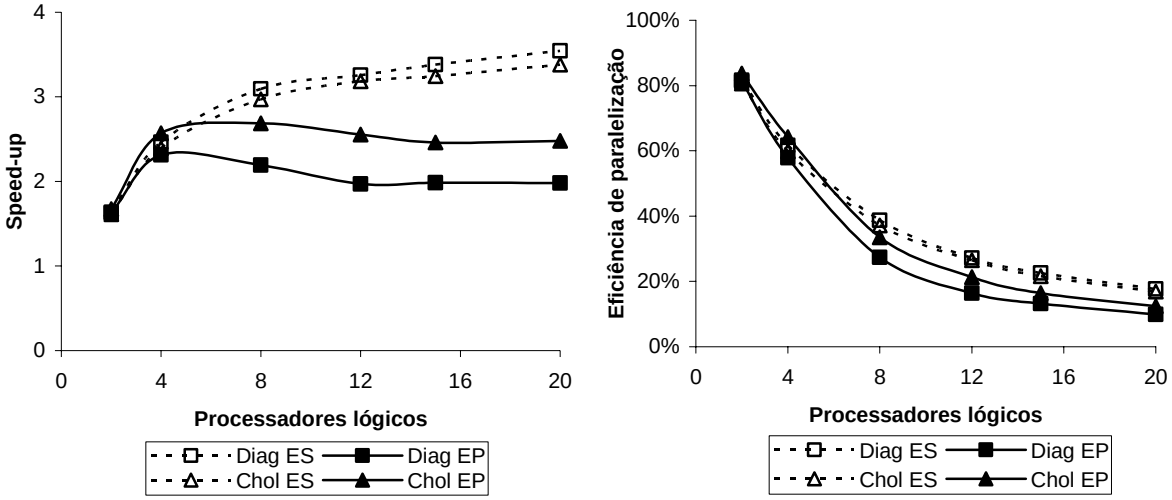
No algoritmo utilizado, as malhas para o fluído e para a estrutura são independentes entre si. Da mesma forma, na versão paralela cada malha tem seu processo de divisão de tarefas com ou sem tratamento de forma independente. Para cada processador lógico envolvido na análise é designado um grupo de nós e elementos da malha do fluído e da malha da estrutura. Para o balanceamento das cargas de trabalho, somente se considera que uma situação equilibrada foi encontrada quando os tempos de processamento dos diversos processadores é o mesmo, dentro de uma certa tolerância, tanto no que se refere ao fluído quanto à estrutura.

Em todos os exemplos utilizados a estrutura de dados correspondente à estrutura é muito pequena, sendo esperadas eficiências de paralelização muito baixas e, eventualmente, *speed-ups* inferiores à unidade. Para contornar parcialmente o problema, foi implementada a possibilidade da estrutura ser processada somente no processador mestre (processamento seqüencial) ao invés de em todos os processadores.

Os computadores e a ordem de uso dos processadores foi a mesma empregada nos testes com o algoritmo paralelo para escoamento subsônico incompressível.

Como indicado no algoritmo utilizado para a análise de problemas de interação fluido estrutura na seção 4.4.2, há a necessidade a cada determinado número de passos de tempo (da análise do fluído) de se transferir os dados de pressão no contorno sólido da estrutura de dados do fluído para a estrutura de dados da estrutura, e, após a correspondente integração no tempo da estrutura, passar os dados de velocidade da malha desta última para a estrutura de dados do fluído. Na implementação paralela, isso significa que os processadores lógicos envolvidos devem transmitir pela rede seus dados sobre pressão no contorno sólido para o processador mestre. Uma vez consolidados, esses dados são distribuídos novamente para os diversos processadores segundo a divisão de tarefas da estrutura. O processamento da estrutura é feito de forma paralela, com a solução do sistema de equações resultante sendo feito pelo Método dos Gradientes Conjugados (em paralelo). Obtidas as velocidades nos nós da malha para o grupo de cada processador, os valores são agrupados no processador mestre e distribuídos novamente aos diversos processadores segundo a divisão de tarefas do fluído. Ou seja, há uma quantidade muito maior de dados trafegando na rede que em qualquer dos algoritmos anteriormente testados, pois o algoritmo para interação fluído-estrutura é o único que exige um modelo de programação do tipo mestre-escravo.

Os resultados obtidos para o *speed-up* e para a eficiência de paralelização do exemplo da cavidade com fundo flexível estão mostrados na figura 5.57. Tanto o pré-condicionador Diagonal como o pré-condicionador de Cholesky foram utilizados. As curvas indicadas por "ES" correspondem a soluções nas quais a estrutura foi analisada de forma seqüencial, somente pelo processador mestre. A indicação "EP" corresponde à estrutura analisada de forma paralela por todos os processadores. Em todos os testes foi utilizado o tratamento 1RN com balanceamento das cargas de trabalho, tanto para o fluído como para a estrutura.



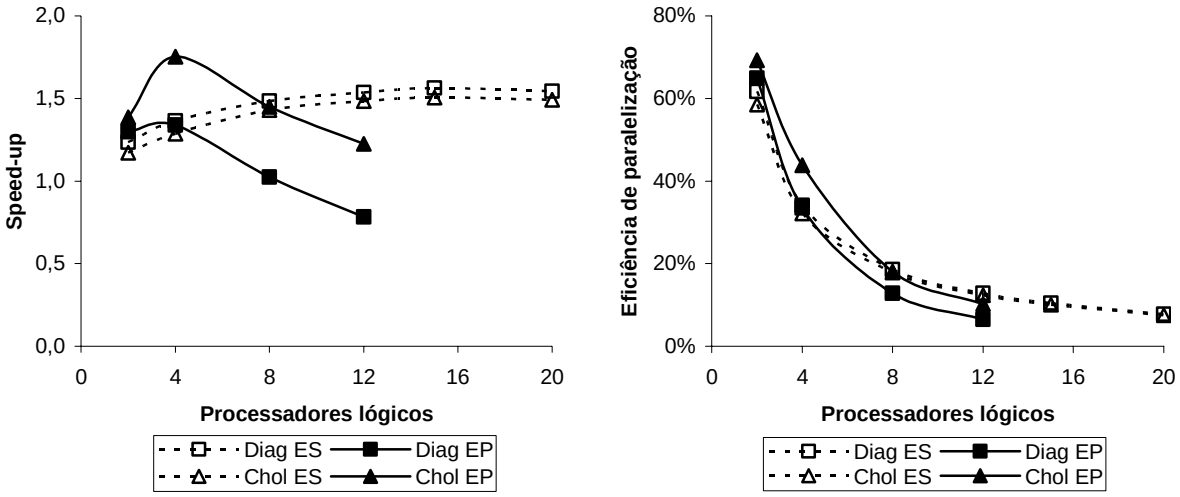
(a) (b)

Figura 5.57: (a) *Speed-up* e (b) eficiência de paralelização para o algoritmo paralelo de interação fluído-estrutura na análise do escoamento em cavidade com fundo flexível.

Os resultados mostram que o pré-condicionador Diagonal levou a um tempo de processamento ligeiramente menor que o de Cholesky. Além disso, a estrutura de dados correspondente à estrutura é tão pequena que é preferível, em termos de desempenho, processar a estrutura somente no processador mestre, utilizando os demais em paralelo apenas para a análise do fluído.

Considerando que a análise somente do fluído para este problema resultou em um *speed-up* máximo de 5,79 com 20 processadores, o valor máximo obtido com a interação fluído-estrutura de 3,54 é consideravelmente menor, decorrência da maior quantidade de dados trafegando na rede entre os processadores durante o processo de solução.

Os resultados obtidos para o exemplo do escoamento bidimensional em torno da edificação com o teto composto por uma membrana flexível estão mostrados na figura 5.58.



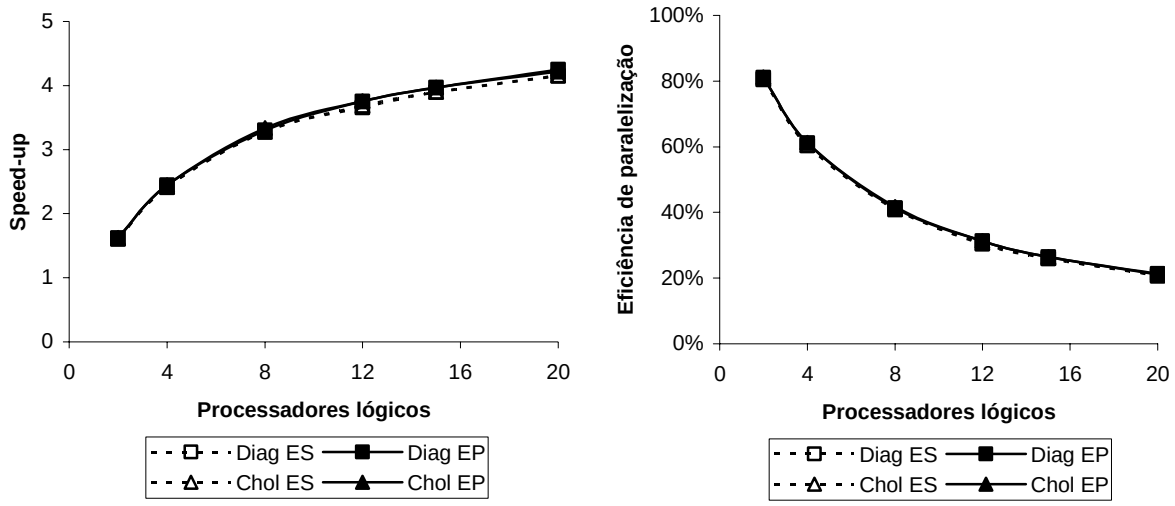
(a) (b)

Figura 5.58: (a) *Speed-up* e (b) eficiência de paralelização para o algoritmo paralelo de interação fluído-estrutura na análise do escoamento bidimensional em torno de edificação com teto composto por membrana flexível.

Novamente a estrutura de dados correspondente é tão pequena que somente é compensador a análise da estrutura em paralelo com um conjunto máximo de 4 processadores, onde se obteve o *speed-up* máximo de 1,54 com o pré-condicionador Diagonal. Além disso, o sistema correspondente à estrutura é extremamente mal condicionado, decorrente do formato bastante achatado dos elementos utilizados para discretizar a membrana do teto, exigindo mais de 1000 iterações para que o Método dos Gradientes Conjugados alcançasse a convergência. Nesse mesmo exemplo, a análise em paralelo somente do escoamento conseguiu um valor máximo de *speed-up* de 10,81 com o tratamento utilizado.

Finalmente, os dados correspondentes a análise da interação fluído-estrutura do escoamento tridimensional em torno do edifício flexível estão mostrados na figura 5.59. Neste exemplo, os pré-condicionadores de Gaus e Cholesky levaram a um mesmo tempo de solução. O tempo de processamento correspondente à estrutura foi bastante pequeno, de modo que não houve diferença prática entre o processamento paralelo e seqüencial da mesma.

Mesmo o fluido sendo preponderante em termos de tempo de processamento no problema, a da necessidade constante de transferência de dados entre o processador mestre e os escravos resultou em um *speed-up* máximo de 4,25, bastante inferior ao valor 10,32 obtido para o mesmo problema com a estrutura rígida.



(a) (b)

Figura 5.59: (a) *Speed-up* e (b) eficiência de paralelização para o algoritmo paralelo de interação fluído-estrutura na análise do escoamento tridimensional em torno de edifício flexível.

Os resultados mostram que, em problemas de interação fluído estrutura, se a malha correspondente à estrutura é muito pequena, muitas vezes não vale a pena realizar o processo de solução da parte da estrutura de forma paralela. Talvez ainda mais vantajoso que processar a parte da estrutura apenas no processador mestre seria fazê-lo em todos os processadores, de forma seqüencial, reduzindo bastante a comunicação mestre-escravos.

Se uma solução seqüencial do sistema de equações correspondente à estrutura vai ser utilizada, um *solver* direto pode ser considerado, especialmente para sistemas mal condicionados como o da edificação com teto em membrana flexível.

Em todos os exemplos analisados, a solução seqüencial da estrutura seria mais rápida utilizando o *solver* direto de Gauss, como indicado na figura 5.2. A opção pela utilização da versão paralela do Método dos Gradientes Conjugados é para que seja possível resolver de forma mais eficiente estruturas de dados bem maiores que as aqui analisadas.

5.5 SISTEMATIZAÇÃO DOS RESULTADOS

Com base nos resultados obtidos, diversas características foram identificadas. Os valores apresentados a seguir devem ser interpretados como um comportamento médio entre diversas

aplicações, e representam mais um indicativo de tendência que um valor que efetivamente será alcançado por alguma aplicação específica.

Em programas de simulação computacional baseados no Método dos Elementos Finitos cuja discretização espacial é feita através de elementos hexaédricos é possível obter ganhos de desempenho consideráveis através de implementações paralelas baseadas em trocas de mensagens entre os computadores através da rede (MPI). Computadores comuns ou estações de trabalho de um laboratório de pesquisa, mesmo quando não especificamente configurados para computação de alto desempenho, podem ser reunidos temporariamente em pequenos *clusters* temporários e interligados através de uma rede de baixa velocidade (*Fast Ethernet* ou 100 Mbps) proporcionando um desempenho bastante superior ao que poderia ser obtido por computadores individuais. A tabela 5.17 mostra os valores médios de *speed-up* obtidos para as aplicações utilizando malhas de hexaedros (*solver* interativo de Gradientes Conjugados, escoamentos compressíveis) com interconexão através de uma rede *Ethernet* de 100 Mbps, com processadores com freqüências de núcleo variando de 1,3 a 2,5 GHz para diversos tamanhos de problemas. O relativamente pequeno número de elementos conectados a um mesmo nó nas malhas de hexaedros faz que, na divisão de tarefas, não seja necessária a comunicação de grande quantidade de dados entre os processadores e nem haja uma redundância considerável no esforço computacional. Essas características, aliadas a uma distribuição de tarefas que minimize o número de operações de comunicação entre processadores, tal como a divisão de tarefas baseada em nós com o tratamento 1RN, permite o emprego de uma rede de baixa velocidade e latência relativamente alta sem que a eficiência de paralelização seja muito baixa.

Tabela 5.17: Valores médios de *speed-up* para aplicações com elementos hexaédricos e rede de 100Mbps.

	Tamanho do problemas (x 1000 GL)				
Número de Processadores	**50**	**250**	**500**	**1400**	**2000**
2	1,6	1,8	1,9	1,9	1,9
4	2,3	3	3,2	3,5	3,5
8	2,3	4	5	6	6,0

Nos problemas cuja discretização espacial é feita por malhas não estruturadas de tetraedros, o número de elementos conectados a ume mesmo nó é, em geral, bastante superior ao obtido em malhas de tetraedros, resultando em um maior número de operações de comunicação entre computadores, uma maior quantidade de dados a ser transmitida entre eles e uma maior redundância no esforço computacional. Apesar disso, os dois exemplos testados de malhas de tetraedros com rede em 100Mbps (escoamento compressível em torno de uma esfera e escoamento compressível em torno de um avião) apresentaram valores de *speed-up* bastante semelhantes aos valores médios obtidos com malhas de hexaedros., conforme mostra a tabela 5.18.

Tabela 5.18: Valores consolidados de *speed-up* para aplicações com elementos tetraédricos e rede de 100Mbps.

	Tamanho do problemas (x 1000 GL)	
Número de Processadores	**300**	**1400**
4	3,4	3,1
8	6,8	5,3

O uso de uma rede mais rápida (1 Gbps) permite alcançar valores de *speed-up* de 20 a 40% maiores especialmente em problemas pequenos com o uso de um número grande de processadores. A maior taxa de transferência da rede *Gigabit Ethernet* em relação à *Fast Ethernet* e, em especial, sua menor latência é particularmente importante para problemas discretizados por malhas de tetraedros, resultando em grandes aumentos de eficiência mesmo em problemas grandes, o que não ocorre com as malhas de tetraedros.

Para malhas de hexaedros com divisão de tarefas empregando o tratamento 1RN e com rede de 1Gbps, os valores médios de *speed-up* considerando as diversas aplicações testadas (escoamento compressível em torno de uma esfera, escoamento compressível em torno de um veículo espacial, escoamento incompressível em cavidade, escoamento incompressível bidimensional em torno de edificação, escoamento incompressível tridimensional em torno de edificação rígida) estão mostrados na Tabela 5.19. Os valores consolidados para o tratamento nRN estão mostrados na tabela 5.20. O tratamento nRN é o que resulta em melhor desempenho quando utilizado com malhas estruturadas.

Tabela 5.19: Valores médios de *speed-up* para aplicações com elementos hexaédricos, rede de 1 Gbps e tratamento 1RN.

	Tamanho do problemas (x 1000 GL)				
Número de Processadores	**30**	**200**	**1000**	**1500**	**5000**
4	2,9	3,4	3,7	3,7	3,8
8	3,8	5,7	6,8	7,0	7,1
12	4,3	7,2	9,2	9,5	9,7
16	5,3	9,2	13	13,5	13,8
20	5,8	10,8	16,5	17,2	17,8

Tabela 5.20: Valores consolidados de *speed-up* para aplicações com elementos hexaédricos, rede de 1 Gbps e tratamento nRN.

	Tamanho do problemas (x 1000 GL)				
Número de Processadores	**30**	**200**	**1000**	**1500**	**5000**
4	3,1	3,5	3,9	3,9	3,9
8	4,4	6,2	7,4	7,5	7,6
12	4,6	8,4	10,8	11,0	11,2
16	4,7	9,5	13,3	13,6	13,8
20	4,9	11,6	17,1	17,5	17,8

Para malhas de tetraedros, com divisão de tarefas empregando o tratamento 1RN e com rede de 1Gbps, os valores médios de *speed-up* considerando as diversas aplicações testadas (escoamento compressível em torno de esfera, escoamento incompressível em torno de avião) estão mostrados na Tabela 5.21. Os valores médios para o tratamento nRN estão mostrados na tabela 5.22. Também para esse tipo de malha o tratamento nRN é o que resulta em melhor desempenho, apresentando uma diferença maior em relação ao tratamento 1RN que o observado para malhas de hexaedros, especialmente quando um grande número de processadores é empregado.

Tabela 5.21: Valores médios de *speed-up* para aplicações com elementos tetraédricos, rede de 1 Gbps e tratamento 1RN.

	Tamanho do problemas (x 1000 GL)		
Número de Processadores	**250**	**1500**	**6500**
4	3,9	3,8	3,9
8	6,7	6,8	7,2
12	8,5	9,3	10
16	9,7	11,1	12
20	10,5	12,6	13,4

Tabela 5.22: Valores médios de *speed-up* para aplicações com elementos tetraédricos, rede de 1 Gbps e tratamento nRN.

	Tamanho do problemas (x 1000 GL)		
Número de Processadores	**250**	**1500**	**6500**
4	3,9	3,8	3,9
8	7	6,9	7,2
12	9,6	9,6	10,1
16	11,5	12	12,3
20	13,2	14,1	14,1

O modelo de programação paralela de programa sem mestre (*hostless program)* empregado em todas as aplicações com exceção das para simulação de interação fluído estrutura mostraram-se bastante eficientes para todos os tipos de malha, alcançando valores de *speed-up* respeitáveis. Por outro lado, o modelo *mestre-escravo* empregado nos problemas de interação fluído-estrutura mostraram-se altamente ineficientes, principalmente devido à enorme quantidade de dados que precisa trafegar na rede dos escravos para o mestre e vice versa. Considerando os resultados obtidos, sempre que possível esse modelo deve ser preterido em favor do primeiro.

6 RESULTADOS OBTIDOS PARA CONFIGURAÇÕES DE MEMÓRIA COMPARTILHADA E HÍBRIDAS MEMÓRIA COMPARTILHADA – MEMÓRIA DISTRIBUÍDA.

6.1 CONSIDERAÇÕES GERAIS SOBRE A IMPLEMENTAÇÃO

As técnicas desenvolvidas nos capítulos 2 e 3 e a forma de paralelização empregada nos algoritmos mostrados no capítulo 4 foram concebidas e otimizadas para configurações de memória distribuída, em que a comunicação entre processadores é feita através de uma rede lógica e a sincronização e troca de dados entre os processadores são feitas através de uma biblioteca de troca de mensagens. Contudo, esses mesmos algoritmos podem ser utilizados em configurações de memória compartilhada, tais como computadores pessoais com 1 único processador com diversos núcleos, funcionando na maioria dos aspectos como um computador com diversos processadores.

Os processadores de uso geral para computadores pessoais e *notebooks* com múltiplos núcleos de processamento têm se tornado o padrão de mercado nos últimos 3 anos, com o número de núcleos físicos variando de 2 a 4 e de núcleos lógicos variando de 2 a 8. Nesses processadores, cada núcleo funciona como um processador completo, acessando a totalidade da memória, justificando a nomenclatura *memória compartilhada*. Em tais configurações, não há a necessidade de haver uma troca de mensagens entre núcleos, pois as variáveis estão na memória e todo o conteúdo da memória pode ser mapeado por qualquer dos núcleos. Há formas, portanto, bem mais simples de se implementar um algoritmo paralelo do que trocando mensagens entre os núcleos. Para isso, existem bibliotecas específicas de comandos que permitem atribuir tarefas aos diversos núcleos, determinar que variáveis são compartilhadas entre eles e que variáveis são proprietárias de cada um, que trechos do código precisam ser executados de forma seqüencial (por um único núcleo) e de que forma uma variável pode ser atualizada por mais de núcleo.

Para testar a eficiência da forma de paralelização dos algoritmos propostos em configurações de memória compartilhada, o algoritmo para o esquema explícito de um passo com interação para simulação de escoamentos compressíveis em regime transônico e supersônico foi paralelizado de forma otimizada para emprego nessas configurações. Para tanto, utilizou-se a biblioteca OpenMP, e a implementação foi feita com cada laço do programa não relacionado à entrada e saída de dados tornado paralelo, e com somente as variáveis locais específicas de cada laço tornadas privadas, mantendo as demais compartilhadas para otimizar o uso da memória. Toda a estrutura para uso da biblioteca MPI para comunicação em rede foi mantida, podendo ser configurada através de parâmetros nos dados de entrada. Cuidados foram tomados para que a estrutura de comunicação em rede não causasse impacto sobre a velocidade de processamento quando desativada.

Como o compilador Compaq Visual Fortran v.6.6 utilizado anteriormente não tem suporte para OpenMP, foi empregado também o compilador Intel Visual Fortran 9.1. Como em muitas situações o código gerado pelo compilador Intel é mais lento que o gerado pelo compilador Compaq, nos testes com configurações utilizando somente MPI ambos foram empregados para um comparativo de desempenho.

Os testes foram feitos com um *cluster* permanente e homogêneo composto por 8 computadores com processadores de núcleo quádruplo Intel Core 2 Quad Q9550 com freqüência de núcleo de 3,75 GHz e 12 MB de cache L2, conectados através de uma rede *Gigabit Ethernet* (1 Gbps). A memória instalada em cada computador foi de 8 GB DDR2 800.

Duas malhas de diferentes tamanhos foram escolhidas para cada tipo de elemento (hexaedros lineares e tetraedros lineares), empregadas anteriormente nos testes com configurações de memória distribuída nos exemplos 5.2.2. e 5.2.4, e indicadas na tabela 6.1.

Tabela 6.1: Características das malhas utilizadas nos testes com configurações de memória compartilhada.

Malha	**Elemento**	**Número de Elementos**	**Número de Nós**	**Graus de Liberdade**
VE1	Hexaedro	198.750	211.146	1.055.730
VE2	Hexaedro	917.700	949.212	4.746.060
NS1	Tetraedro	242.979	45.823	229.115
NS2	Tetraedro	1.501.912	271.842	1.359.210

6.2 IMPACTO SOBRE O DESEMPENHO DO USO PLENO DOS NÚCLEOS DE UM PROCESSADOR

Uma parte considerável do desempenho de um processador está relacionada ao acesso à memória, e forma como ela é feita para os vários núcleos é uma característica chave para um bom desempenho com o uso simultâneo de todos os núcleos.

O processador Core 2 de 4 núcleos da Intel não tem controladora de memória interna (no próprio processador), de modo que o acesso à memória pelos vários núcleos fica a cargo do *chipset* da placa-mãe, que desta forma é quase tão importante para o desempenho quanto o processador. Nos testes realizados foi utilizado o chipset Intel P45. O processador conta com um *cache* L2 de 6MB unificado para cada 2 núcleos, compensando, pelo tamanho, a maior latência da controladora de memória externa, e garantindo coerência de *cache* em processamento paralelo com 2 núcleos sem necessidade de acesso à memória principal. Contudo, como o processador de 4 núcleos da Intel é obtido pelo encapsulamento em um único *chip* de 2 processadores de núcleo duplo, sem um *cache* unificado para todos os núcleos, quando todos eles estão em uso em processamento paralelo há a necessidade de acesso à memória para coerência de *cache* entre cada dupla de núcleos.

Averiguar a influência da arquitetura dos processadores utilizados sobre o desempenho foi o objetivo do primeiro teste, de forma a quantificar a perda de desempenho de um código seqüencial quando mais de um núcleo do processador estivesse em uso.

Inicialmente uma única cópia da versão seqüencial de cada um dos códigos empregados foi executada, de modo que somente um núcleo do processador estivesse em uso. Os tempos de processamento obtidos foram utilizados como tempos de referência para uma comparação com 2 ou mais cópias do código sendo executadas simultaneamente, de modo a colocar 2 ou mais núcleos do processador em uso pleno. O procedimento foi feito com as versões dos códigos geradas por ambos compiladores empregados, de forma a se verificar a existência de alguma influência do compilador. Com os tempos de processamento obtidos, foi possível estimar a velocidade relativa ao caso com somente um núcleo em uso. Os resultados obtidos estão mostrados na tabela 6.2.

A diferença entre o desempenho relativo dos códigos gerados pelos dois compiladores é bastante pequena, podendo-se considerar os resultados equivalentes. Em média, com o uso

simultâneo de 2 núcleos há uma perda de aproximadamente 3% no desempenho, e com o uso dos 4 núcleos, 10% aproximadamente. As perdas de desempenho verificadas são características da arquitetura do processador (e do *chipset* da placa-mãe), não podendo ser generalizadas para outras arquiteturas de processadores.

Tabela 6.2: Velocidade relativa de processamento quando mais de um núcleo do processador é utilizado de forma plena – malhas NS1, NS2, VE1 e VE2.

		Compaq 6.6		**Intel 9.1**	
Processador	**Malha**	**2 núcleos**	**4 núcleos**	**2 núcleos**	**4 núcleos**
Intel Q9550	NS1	0,943	0,808	0,959	0,874
	NS2	0,971	0,928	0,978	0,912
	VE1	0,977	0,916	0,970	0,878
	VE2	0,981	0,924	0,978	0,927
Média		0,968	0,894	0,971	0,898

A perda de desempenho observada na execução de códigos seqüenciais simultâneos é esperada de forma análoga na execução de um código paralelo da forma como foi implementado para configurações de memória distribuída utilizando a biblioteca MPI em um processador de núcleos múltiplos, pois apesar dos códigos estarem sendo executados pelos vários núcleos de um mesmo processador, eles são independentes entre si, utilizando áreas de memória principal diferentes e independentes e concorrendo entre si pelo acesso à memória, de forma muito similar à que acontece com a execução simultânea de códigos seqüenciais independentes. Às perdas pelo uso simultâneo dos vários núcleos do processador (associadas principalmente ao acesso à memória) se somam as perdas de desempenho inerentes à implementação paralela.

6.3 PARALELISMO SEM O USO DA REDE

Utilizar uma implementação paralela para memória distribuída em uma configuração de memória compartilhada, especificamente em um computador com um único processador com

múltiplos núcleos, faz com que cada núcleo se comporte como se fosse um computador independente, com seu próprio código e seu conjunto de dados.

Quando uma implementação paralela é executada em diversos computadores com um único processador com um único núcleo, a controladora de memória somente precisa atender a um processo ou um código. A comunicação entre os processos ou códigos é feita obrigatoriamente através da rede lógica de conexão dos computadores.

Em um computador com um único processador de núcleos múltiplos, a controladora de memória deve atender a tantos processos ou códigos quanto o número de núcleos. O impacto sobre o desempenho deve, de certa forma, ser minimizado pela configuração de *cache* escolhida para o processador. Ao se executar uma implementação paralela otimizada para configurações de memória compartilhada, boa parte das variáveis utilizadas nos processos em cada núcleo são as mesmas, podendo ser armazenadas de forma comum, otimizando tanto o uso do *cache* quanto o acesso à memória. Por outro lado, ao se executar uma implementação paralela otimizada para memória distribuída, cada processo em cada núcleo tem suas próprias variáveis, concorrendo uns com os outros pelo uso de *caches* comuns e do acesso à memória. A vantagem é que a comunicação entre os núcleos ou processos se dá inteiramente na memória, e não mais através da rede, pois o sistema operacional é capaz de identificar pacotes cuja origem e destino sejam a mesma interface de rede e os mantém internos ao computador.

Para verificar o desempenho das implementações paralelas utilizando MPI empregadas no capítulo 5 em uma configuração de memória compartilhada, os códigos para tetraedros e hexaedros foram testados em um dos computadores usando de 1 ao número total de núcleos disponíveis como se fossem computadores independentes. Além disso, versões com o uso de OpenMP dos mesmos códigos foram testadas nas mesmas condições.

Como uma estimativa de eficiência, os resultados de *speed-up* foram obtidos considerando-se como referência o tempo de processamento do código seqüencial para o mesmo compilador. A mesma versão seqüencial é base para as configurações usando o compilador Intel com MPI ou com OpenMP. Contudo, como os compiladores geram códigos que podem apresentar diferenças significativas na velocidade de processamento, as *velocidade relativa*s foram calculadas tendo como referência o menor tempo de processamento para a versão seqüencial, independentemente do compilador utilizado.

Os tratamentos 1RN e nRN usados em conjunto com MPI para 2 computadores ou núcleos são equivalentes (1 única reordenação e 1 única divisão) e os resultados obtidos para 4 núcleos são praticamente idênticos, de forma que não será feita diferenciação entre eles.

Os resultados de *speed-up* obtidos para as malhas VE1 e VE2 (hexaedros) com a biblioteca MPI utilizada em conjunto com o compiladores Compaq v.6.6 e Intel v.9.1, bem como os obtidos com OpenMP em conjunto com o compilador Intel v. 9.1 estão mostrados na figura 6.1.

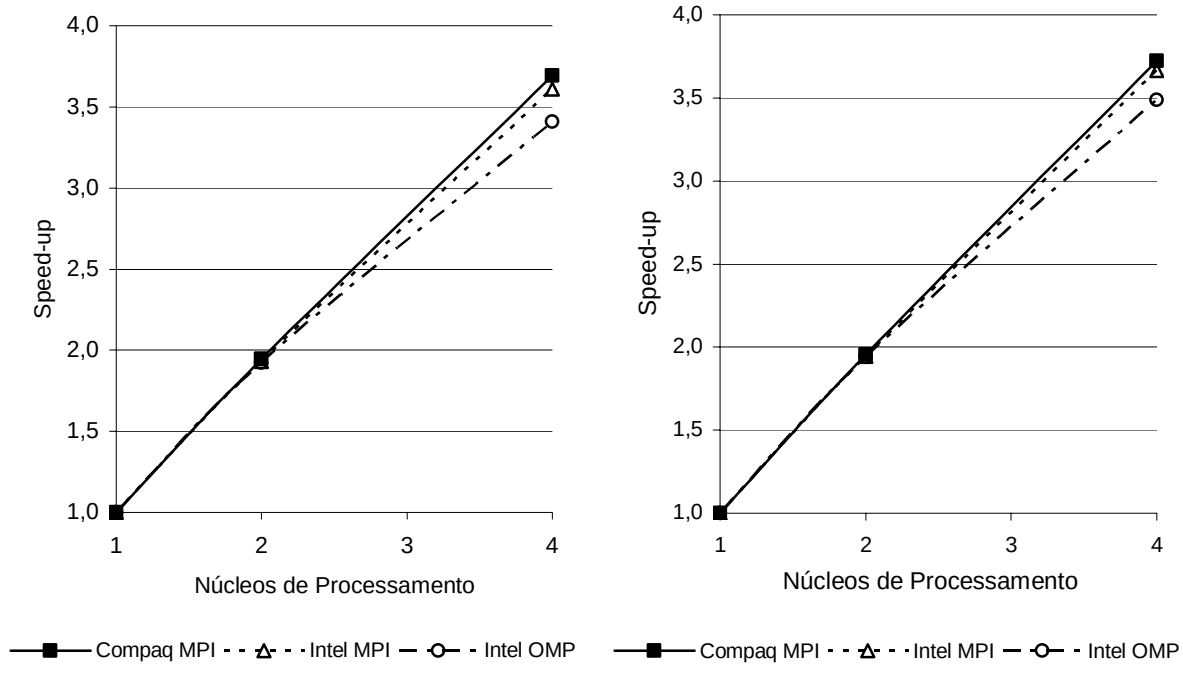
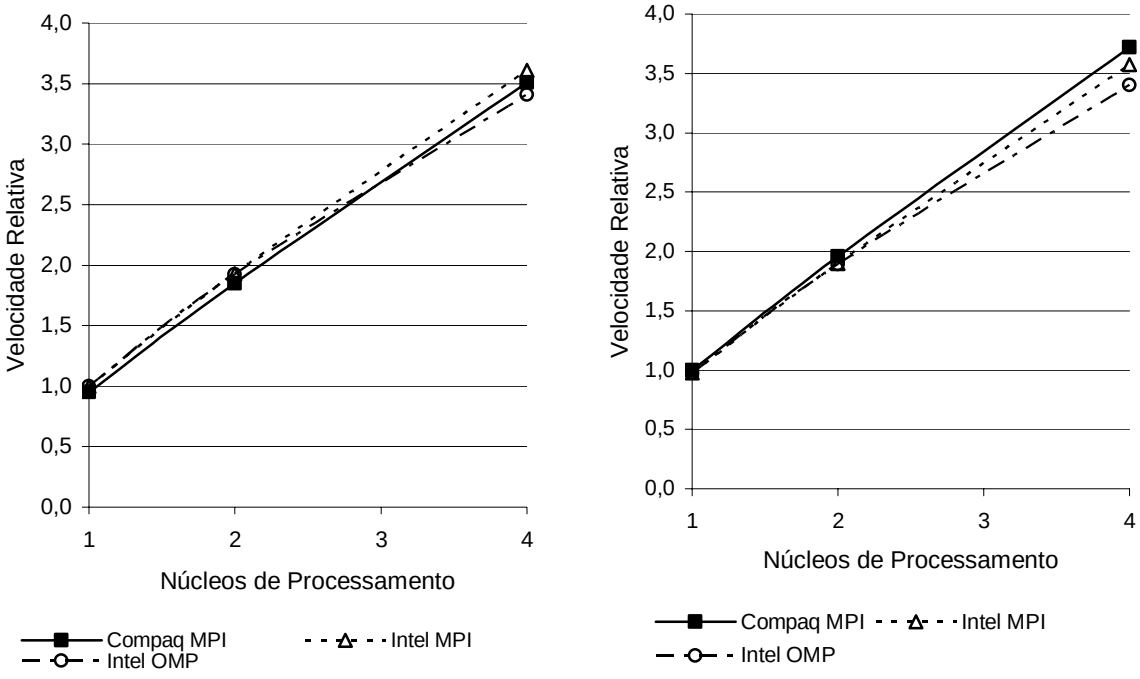


Figura 6.1: *Speed-ups* obtidos para as malhas VE1 (a) e VE2 (b) em um processador multi-núcleos com MPI empregada em conjunto com os compiladores Compaq v.6.6 e Intel v.9.1, e com OpenMP em conjunto com o compilador Intel v.9.1.

Com o uso de apenas 2 núcleos do processador, os valores de *speed-up* obtidos nas três implementações foram muito próximos, alcançando aproximadamente 1,93 para a malha VE1 e 1,95 no VE2. Com o uso de 4 núcleos, as duas implementações com MPI foram praticamente equivalentes (*spped-up* médio de 3,65 para VE1 e 3,69 para VE2), e a implementação empregando-se OpenMP mostrou-se a menos eficiente (aproximadamente 7% mais lenta que as demais), contrariando a expectativa inicial, uma vez que não há redundância do esforço computacional nessa forma de implementação.

Para fins de comparação, o código gerado pelo compilador Compaq com MPI obteve *speed-ups* de 1,97 e 3,88 para 2 e 4 computadores com um único processador de um único núcleo da classe Athlon 64 conectados por uma rede de 1 Gbps para a malha VE1, e 1,98 e 3,92 para a malha VE2, todos maiores que os obtidos com o processador de múltiplos núcleos. Se por um lado há neste último uma maior velocidade de comunicação entre os processos (a troca de dados ocorre internamente na memória sem o uso da rede), por outro há uma maior sobrecarga na controladora de memória em função dos processos em execução em cada núcleo concorrendo entre si pelo acesso à memória, anulando a vantagem da comunicação mais rápida.

Os valores de *velocidade relativa* obtidos estão mostrados na figura 6.2.



(a) (b)

Figura 6.2: *Velocidades relativas* obtidas para as malhas VE1 (a) e VE2 (b) em um processador multi-núcleos com MPI empregada em conjunto com os compiladores Compaq v.6.6 e Intel v.9.1, e com OpenMP em conjunto com o compilador Intel v.9.1.

Apesar de apresentarem o mesmo valor de *speed-up* e, conseqüentemente, mesma eficiência de paralelização, o código obtido com o compilador Compaq / MPI é ligeiramente mais lento que o correspondente com o compilador Intel / MPI para a malha menor (VE1), e ligeiramente mais rápido para a malha maior (VE2). Para os dois problemas o código

utilizando OpenMP foi o mais lento, com diferenças em relação à implementação mais rápida de 6% e 9% para as malhas VE1 e VE2, respectivamente.

Os *speed-ups* obtidos de para as malhas de tetraedros estão mostrados na figura 6.3.

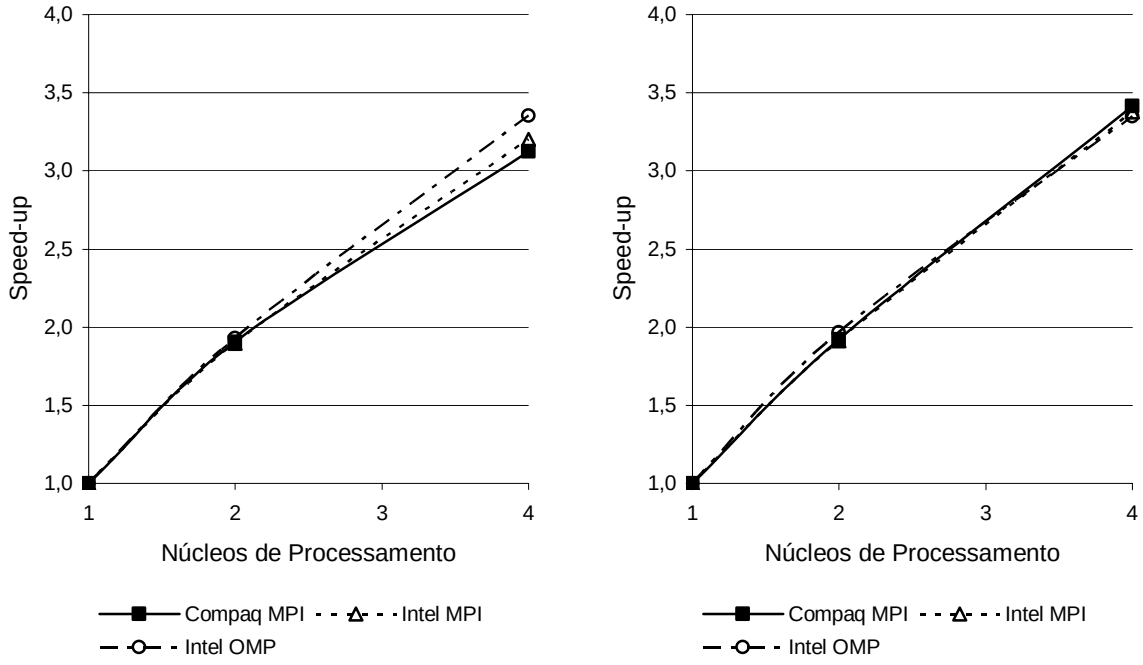


Figura 6.3: *Speed-ups* obtidos para as malhas NS1 (a) e NS2 (b) em um processador multi-núcleos com MPI empregada em conjunto com os compiladores Compaq v.6.6 e Intel v.9.1, e com OpenMP em conjunto com o compilador Intel v.9.1.

O *speed-up* obtidos com MPI para a menor malha de tetraedros (NS1) foi aproximadamente 15% menor que o obtido para a menor malha de hexaedros, e a diferença para as malhas maiores foi de aproximadamente 8%. Esses resultados são conseqüência direta das características geométricas dos dois tipos de malhas. As malhas de tetraedros tendem a ter um número muito maior de elementos conectados a cada nó, fazendo com que a comunicação de dados entre os processadores lógicos (núcleos) seja consideravelmente maior. Essa característica não afeta as implementações empregando OpenMP, pois não há comunicação de dados entre processos, mas acesso compartilhado à memória, fazendo com que os resultados obtidos sejam praticamente equivalentes para os dois tipos de malhas. Para a malha menor (NS1), a implementação empregando OpenMP apresentou o maior *speed-up* (5% maior que a melhor solução com MPI), e para a malha maior (NS2) todas as implementações foram praticamente equivalentes.

Para fins de comparação, o código gerado pelo compilador Compaq com MPI obteve *speed-ups* de 2,02 e 3,59 para 2 e 4 computadores com um único processador de um único núcleo da classe Athlon 64 conectados por uma rede de 1 Gbps para a malha VE1, e 1,99 e 3,72 para a malha VE2. Os melhores resultados obtidos para a configuração de múltiplos núcleos foi de 1,93 e 3,35 para a malha NS1 e 1,97 e 3,35 para a malha NS2, mostrando também neste caso o impacto negativo sobre o desempenho decorrente da sobrecarga da controladora de memória ao atender diversos núcleos simultaneamente.

Os valores de *velocidade relativa* obtidos estão mostrados na figura 6.4.

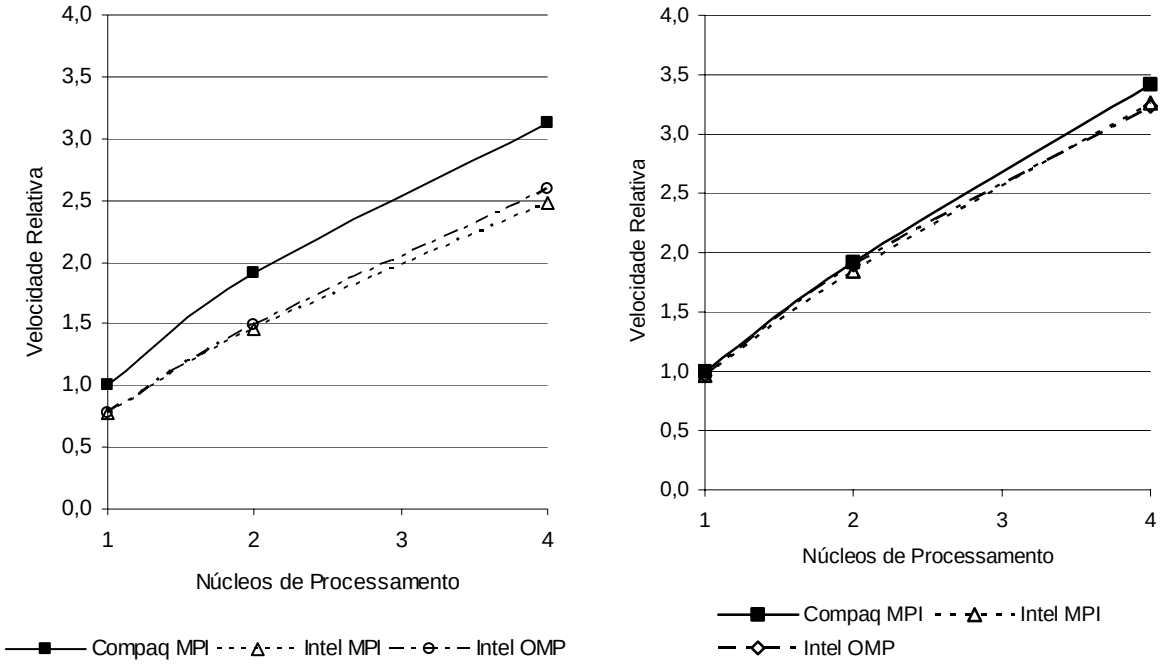


Figura 6.4: *Velocidades relativas* obtidas para as malhas NS1 (a) e NS2 (b) em um processador multi-núcleos com MPI empregada em conjunto com os compiladores Compaq v.6.6 e Intel v.9.1, e com OpenMP em conjunto com o compilador Intel v.9.1.

Para a malha menor (NS1) a implementação Compaq com MPI obteve uma velocidade de processamento 21% maior que as obtida pela implementação mais rápida com o compilador Intel (OpenMP), mesmo tendo o mais baixo *speed-up* dentre todas as implementações para essa malha, um resultado surpreendente considerando-se que o compilador Intel v.9.1 é uma versão melhorada e otimizada para processadores Intel do compilador Compaq. Para a malha de maior tamanho (NS2), a vantagem em termos de velocidade de processamento para a implementação Compaq / MPI é cerca de 5% mais rápida que o as implementações com o

compilador Intel, embora essa diferença seja pequena o suficiente para que as soluções possam ser consideradas equivalentes.

6.4 PARALELISMO HÍBRIDO COM E SEM O USO DA REDE

Quando um código paralelo empregando MPI é utilizado em um computador com um único processador com múltiplos núcleos, a comunicação entre os processos ou núcleos ocorre sem o uso da rede, permanecendo interna ao computador. Contudo, quando é formado um *cluster* de computadores com esse tipo de processador, as comunicações entre os núcleos de um processador e os de outro é efetivamente feita através da rede. Ao contrário de *clusters* compostos por computadores com um único processador de núcleo único, como os utilizados nos exemplos do capítulo 5, que dispõem de uma interface de rede para cada núcleo, em um *cluster* de computadores com um processador de múltiplos núcleos normalmente existe apenas uma interface de rede para atender à comunicação de todos os núcleos.

Esquemas de divisões de tarefas que impliquem na comunicação de cada núcleo com todos os outros, como é o caso do tratamento nRN, tendem a ser menos eficientes em *clusters* de computadores com processadores de múltiplos núcleos, pois há um grande número de comunicações simultâneas e externas a cada computador saturando a interface de rede e dividindo a largura de banda disponível nos *switches* de rede.

Por outro lado, tratamentos como o 1RN, que fazem com que em um processador de 4 núcleos somente o primeiro e o último núcleos precisem de uma comunicação de dados externa ao computador cada um, com as comunicações entre os núcleos (1-2), (2-3) e (3-4) permanecendo internas, tendem a ser mais vantajosas. Um tratamento do tipo nRN somente se tornaria mais vantajoso quando a quantidade de dados a serem comunicados entre núcleos decorrente da divisão de tarefas a ele associada fosse significativamente menor que a correspondente quantidade de dados para um tratamento do tipo 1RN, compensando assim a perda de desempenho associada à latência de um número muito maior de processos de comunicação em rede e à saturação da interface de rede.

Em uma divisão de tarefas baseada em nós, o uso de uma biblioteca de paralelismo de memória compartilhada como a OpenMP para dividir tarefas entre os múltiplos núcleos do processador de um computador, associado ao uso de um biblioteca como MPI para a

comunicação de dados entre computadores (os nodos do *cluster*) leva ao mesmo número de comunicações pela rede e à mesma quantidade de dados a serem comunicados que com o uso unicamente de MPI quando a malha sofre um tratamento do tipo 1RN (visto que parte das comunicações permanecem internas). Quando um tratamento mais complexo como o nRN é utilizado, uma implementação OpenMP-MPI resulta em um número bastante menor de processos de comunicação através da rede que uma implementação MPI pura, embora a quantidade de dados por processo de comunicação seja maior.

Desta forma, uma implementação OpenMP-MPI torna-se atrativa quando:

a) o emprego da biblioteca OpenMP representa a alternativa de melhor desempenho para a paralelização de tarefas entre os núcleos de um mesmo processador;

b) um número grande de processadores com múltiplos núcleos é empregado. Neste caso, a comunicação direta entre núcleos utilizando MPI somente e empregando o tratamento 1RN levaria a uma divisão da malha ineficiente, e o tratamento nRN levaria a um número muito grande de processos de comunicação simultâneos que se tornam críticos com um número reduzido de interfaces de rede.

A fim de verificar o desempenho de todas as configurações, a implementação paralela com os oito computadores disponíveis foi feita de 2 formas:

a) os 32 núcleos se comunicam diretamente uns com os outros com o uso da biblioteca MPI, independentemente de pertencerem a um mesmo processador ou a processadores diferentes.

b) a divisão de tarefas entre os núcleos de um mesmo processador é feita utilizando-se a biblioteca OpenMP. A comunicação entre computadores é feita através da biblioteca MPI.

Com o uso de OpenMP, a distribuição de tarefas é feita dividindo-se cada laço do programa em tantas partes quantos forem os núcleos do processador, uma vez que nos processadores utilizados os correspondentes núcleos tem o mesmo poder computacional. Com o uso de MPI, a divisão inicial de tarefas entre núcleos é feita baseada na freqüência de cada núcleo, e balanceada até que o equilíbrio seja obtido. Os tratamentos 1RNe nRN são utilizados. Com o uso simultâneo de OpenMP e MPI, a divisão inicial de tarefas entre computadores é feita

utilizando-se como parâmetro de desempenho relativo de cada processador o produto da freqüência do processador pelo número de núcleos disponíveis. Essa divisão inicial entre computadores é balanceada até que o equilíbrio seja atingido e os tratamentos 1RN e nRN são empregados na divisão da malha entre computadores. A divisão do esforço computacional pelo diversos núcleos internamente ao processador é feita através do OpenMP dividindo-se os laços entre eles.

Os resultados de *speed-up* obtidos para as malhas de hexaedros estão mostrados na figura 6.5. Nas implementações utilizando apenas MPI há, em geral, equivalência entre o desempenho dos códigos gerados pelos compiladores Compaq e Intel. O tratamento 1RN foi, em geral, mais eficiente que o nRN, embora a diferença de desempenho não seja significativa na maioria dos casos. A única configuração em que o tratamento 1RN resultou em *speed-up* significativamente maior (11% aproximadamente) que o tratamento nRN foi com a malha VE2 com o compilador Intel.

As implementações utilizando somente MPI foram mais eficientes que as utilizando MPI e OpenMP em conjunto. Com o tratamento 1RN, as comunicações entre os núcleos de um mesmo processador ocorrem internamente, de modo que as únicas comunicações com o uso da rede são entre o primeiro núcleo do processador e o último do processador anterior, e entre o último núcleo do processador e o primeiro do processador seguinte, de modo que o número de comunicações em rede e a quantidade de dados envolvida são aproximadamente os mesmos dos correspondentes a uma implementação usando OpenMP e MPI com o tratamento 1RN. No entanto, a divisão de tarefas entre processadores inerente ao emprego de MPI resulta em redundância do esforço computacional. Essa redundância é menor na configuração que emprega MPI e OpenMP em conjunto, em comparação com as configurações utilizando somente MPI pois a divisão de tarefas é feita em função do número de computadores, e não de núcleos, resultando em um número de divisões 4 vezes menor (para processadores de 4 núcleos) e, conseqüentemente, um número consideravelmente menor de fronteiras. Esse efeito pode ser percebido na malha de menor tamanho (VE1) quando o número de processadores utilizados torna-se grande. As configurações utilizando apenas MPI apresentam uma perceptível perda de escalabilidade, caracterizada pela redução da inclinação dos gráficos de *speed-up versus* número de processadores, o que não é percebido na configuração empregando OpenMP em conjunto com MPI até o número máximo de processadores disponíveis. Se essa tendência se mantiver com o aumento do número de processadores, o

speed-up obtido com o uso conjunto de OpenMP e MPI provavelmente será maior que o obtido com apenas MPI.

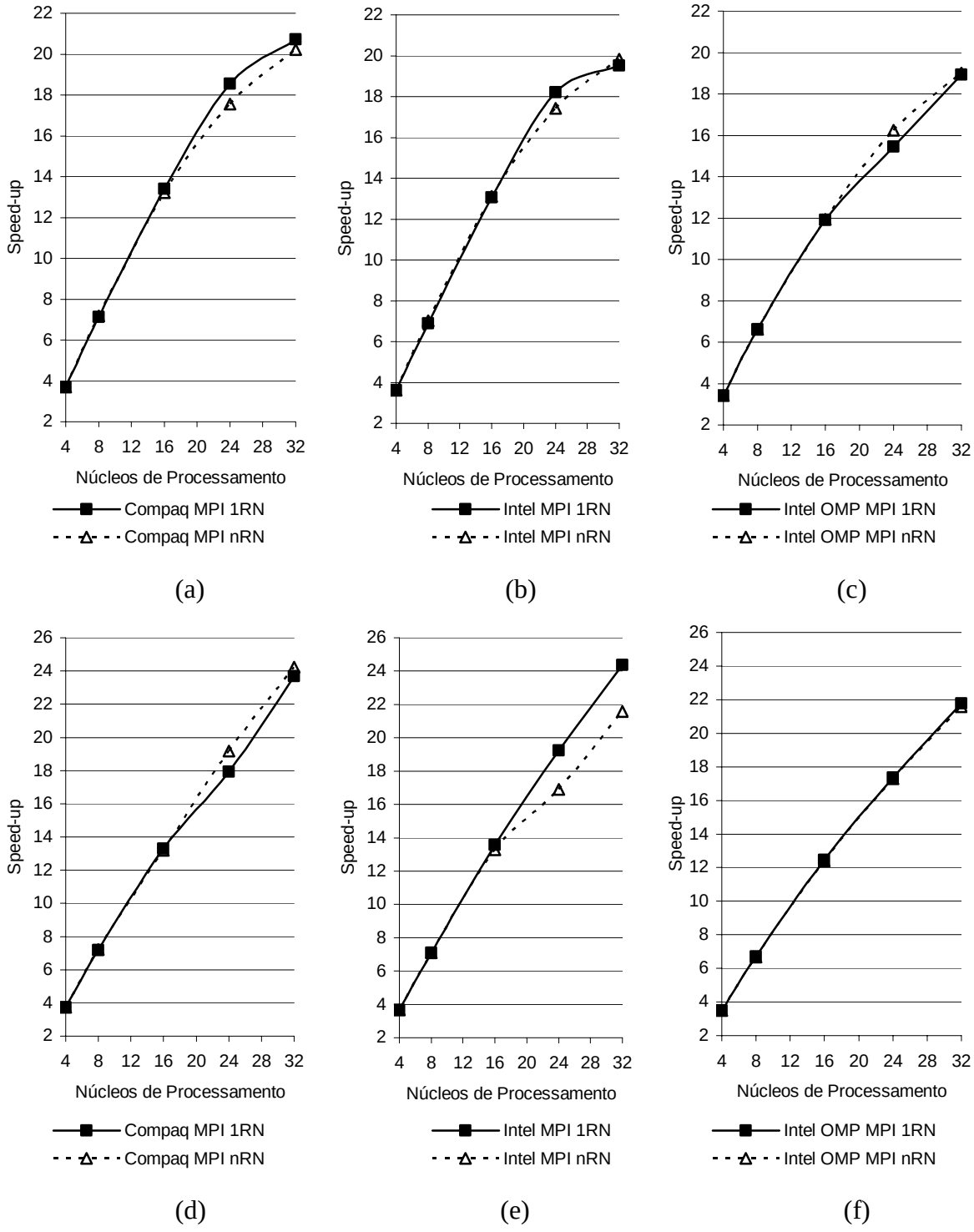


Figura 6.5 *Speed-ups* obtidos em *cluster* permanente com processadores multi-núcleos para o problema VE1 com (a) compilador Compaq v.6.6 / MPI, (b) compilador Intel v.9.1 / MPI, (c) compilador Intel v.9.1 / OpenMP + MPI e para o problema VE2 com (d) compilador Compaq v.6.6 / MPI, (e) compilador Intel v.9.1 / MPI, (f) compilador Intel v.9.1 / OpenMP + MPI.

Nas implementações utilizando somente MPI com o tratamento nRN, cada núcleo de cada processador pode estabelecer comunicações com todos os outros núcleos dos outros processadores, de modo que a interface de rede única para cada 4 núcleos tende a ser muito solicitada e constituir-se em um gargalo importante para o desempenho. O tratamento nRN reduz a quantidade de dados a serem comunicados através da rede em relação ao tratamento 1RN, mas aumenta o número de operações de comunicação, o que é particularmente crítico quando a quantidade de dados a ser comunicada é pequena, tornando a solução mais dependente da latência da comunicação em rede que da taxa de transferência, situação agravada pela presença de apenas uma interface de rede para atender a vários núcleos simultaneamente. Esses fatores explicam o menor desempenho das soluções com esse tratamento em 3 das 4 configurações.

Os valores máximos de *speed*-up utilizando o compilador Compaq / MPI para 16 processadores foram de 13,4 e 13,3 para as malhas VE1 e VE2, respectivamente, ambas para o tratamento 1RN. Para comparação, os valores obtidos com o *cluster* de 15 computadores com processadores de núcleo único e uma rede de 1Gbps foram 13,3 e 13,8 para as mesmas malhas. A maior eficiência do conjunto com processadores de núcleo único deve-se tanto à controladora de memória e interface de rede serem dedicadas a apenas um núcleo ou processo, quanto à velocidade relativa da rede em relação à velocidade de processamento ser maior (rede de 1Gbps para 2,2 GHz por núcleo contra 3,75 GHz).

Em função do número relativamente pequeno de elementos conectados a cada nós da malha, o que é favorável ao emprego de MPI tanto no que se refere à redundância do esforço computacional quanto ao número de operações de comunicação entre processadores e à quantidade de dados a ser comunicada através da rede, os algoritmos paralelos desenvolvidos mostraram-se particularmente eficientes em problemas com malhas de hexaedros, alcançando uma eficiência de paralelização entre 65% e 75% com 32 núcleos de processamento para a malha de maior tamanho (VE2).

Os resultados de velocidade reativa obtidos estão mostrados na figura 6.6. O comportamento geral é o mesmo verificado para o *speed-up*, com as implementações utilizando apenas MPI praticamente equivalentes, ambas apresentando maior velocidade de processamento que a solução empregando OpenMP combinado com MPI. Para efeitos de comparação, a velocidade de processamento para 16 núcleos de 3,75GHz no c*luster* com processadores de núcleos

múltiplos é 1,76 vezes maior que a obtida para 15 núcleos de 2,2 GHz com o cluster com processadores de núcleo único para a malha VE1, e 1,85 vezes maior para a malha VE2.

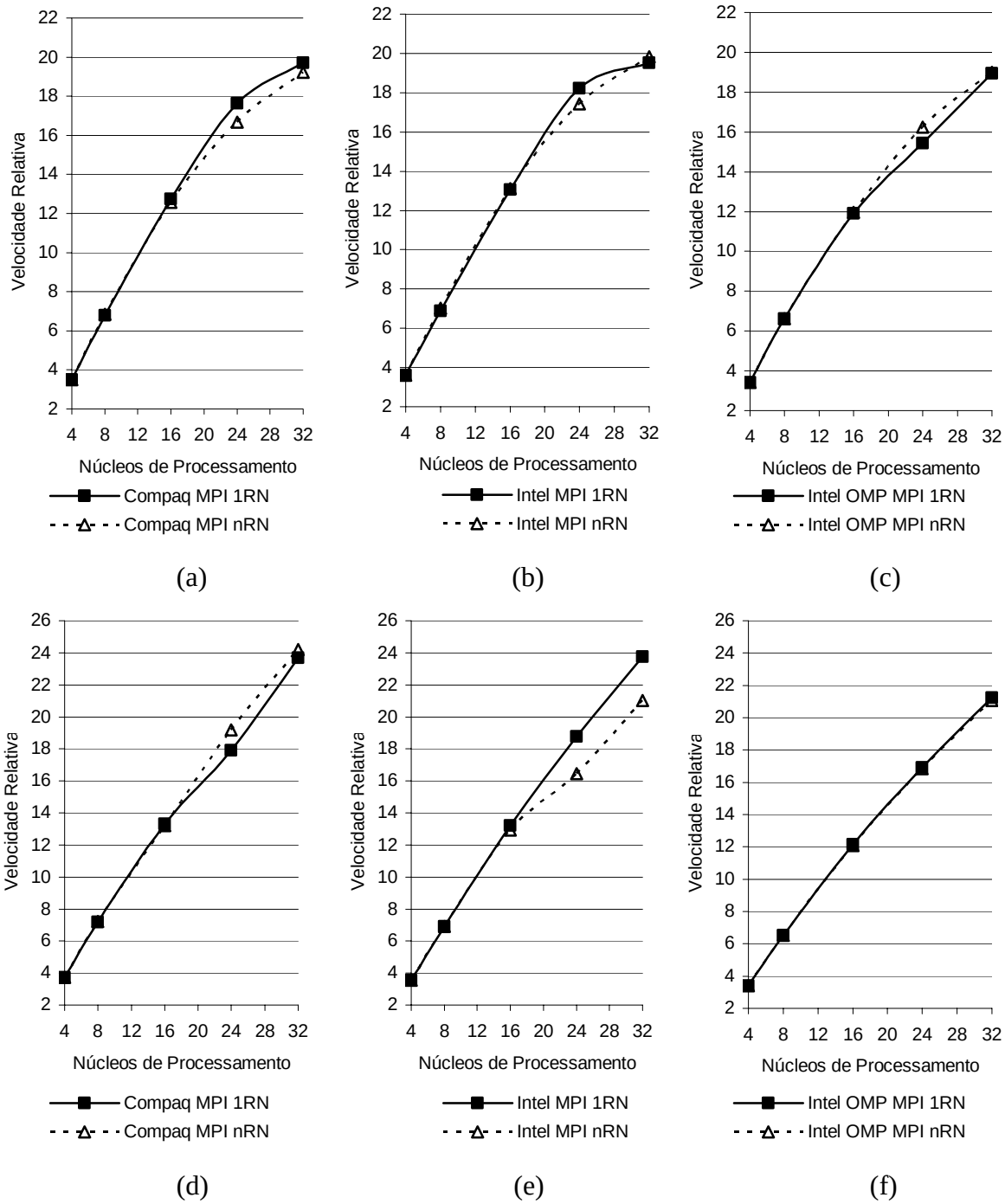


Figura 6.6: Velocidades relativas obtidas em *cluster* permanente com processadores multi-núcleos para o problema VE1 com (a) compilador Compaq v.6.6 / MPI, (b) compilador Intel v.9.1 / MPI, (c) compilador Intel v.9.1 / OpenMP + MPI e para o problema VE2 com (d) compilador Compaq v.6.6 / MPI, (e) compilador Intel v.9.1 / MPI, (f) compilador Intel v.9.1 / OpenMP + MPI.

Os resultados para a fração serial de Karp-Flatt estão mostrados na figura 6.7 para a malha VE1 e 6.8 para a malha VE2, mostrando uma escalabilidade levemente pior para a implementação utilizando OpenMP em conjunto com MPI em relação às implementações empregando somente MPI, especialmente quando um número pequeno de núcleos é empregado. Os resultados utilizando apenas MPI para os compiladores Compaq e Intel são equivalentes.

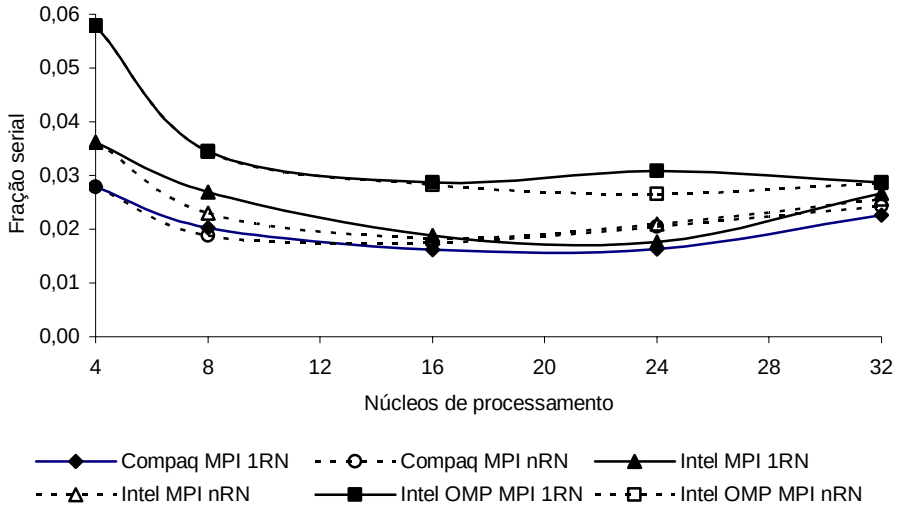


Figura 6.7 Frações seriais obtidas em *cluster* permanente com processadores multi-núcleos para o problema VE1 – malha de hexaedros.

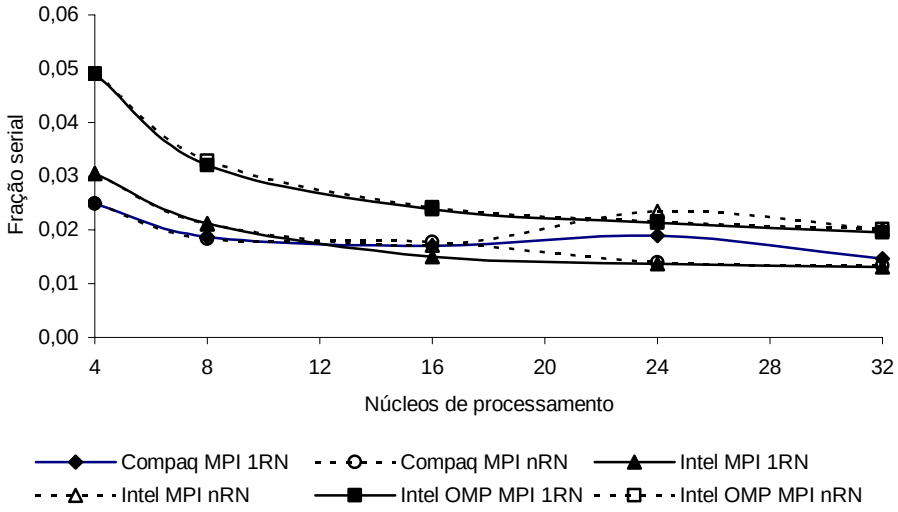


Figura 6.8 Frações seriais obtidas em *cluster* permanente com processadores multi-núcleos para o problema VE2 – malha de hexaedros.

Para as malhas de tetraedros, os valores de *speed-up* obtidos estão mostrados na figura 6.10, e os de velocidade relativa estão mostrados na figura 6.11.

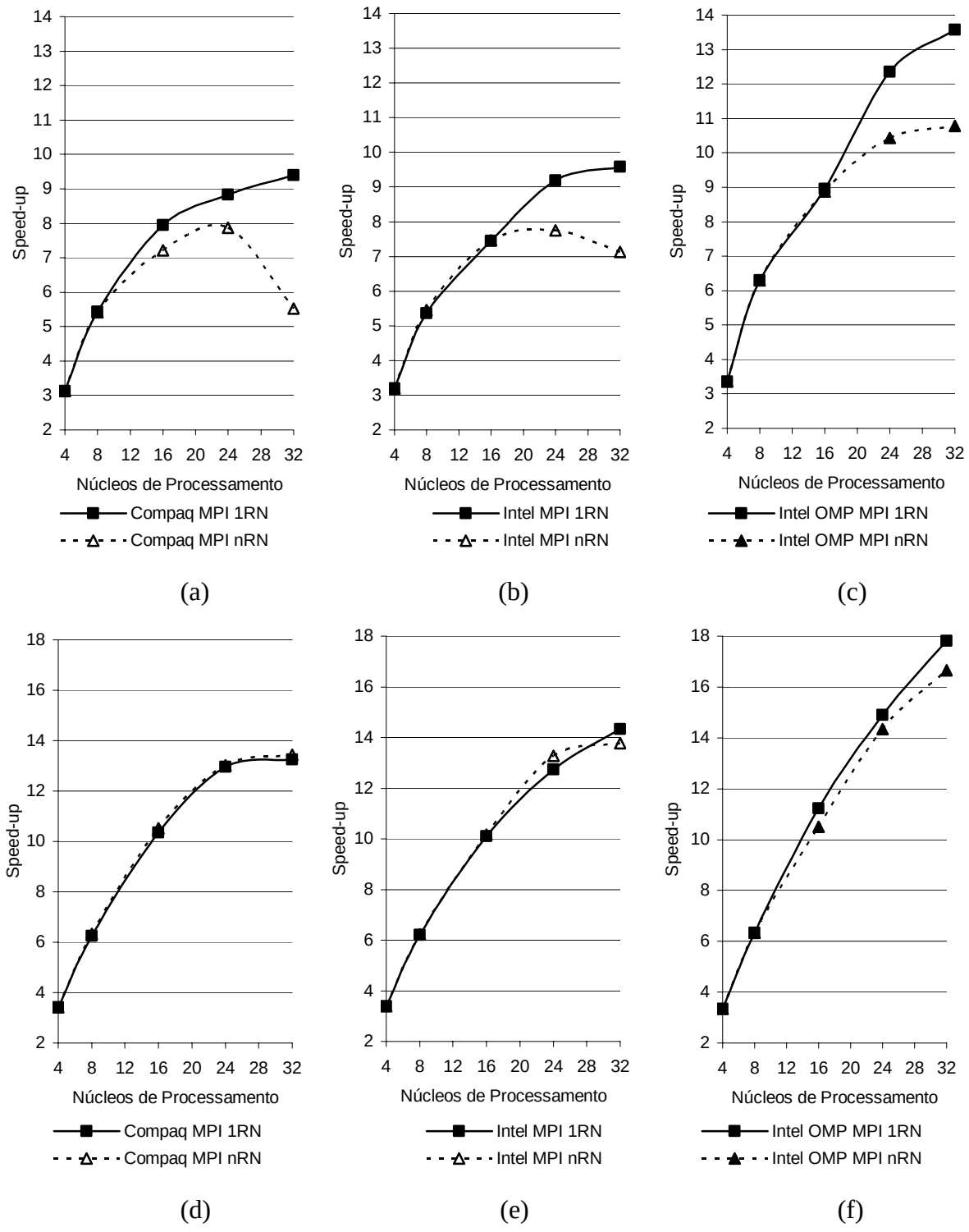


Figura 6.9: *Speed-ups* obtidos em *cluster* permanente com processadores multi-núcleos para o problema NS1 com (a) compilador Compaq v.6.6 / MPI, (b) compilador Intel v.9.1 / MPI, (c) compilador Intel v.9.1 / OpenMP + MPI e para o problema NS2 com (d) compilador Compaq v.6.6 / MPI, (e) compilador Intel v.9.1 / MPI, (f) compilador Intel v.9.1 / OpenMP + MPI.

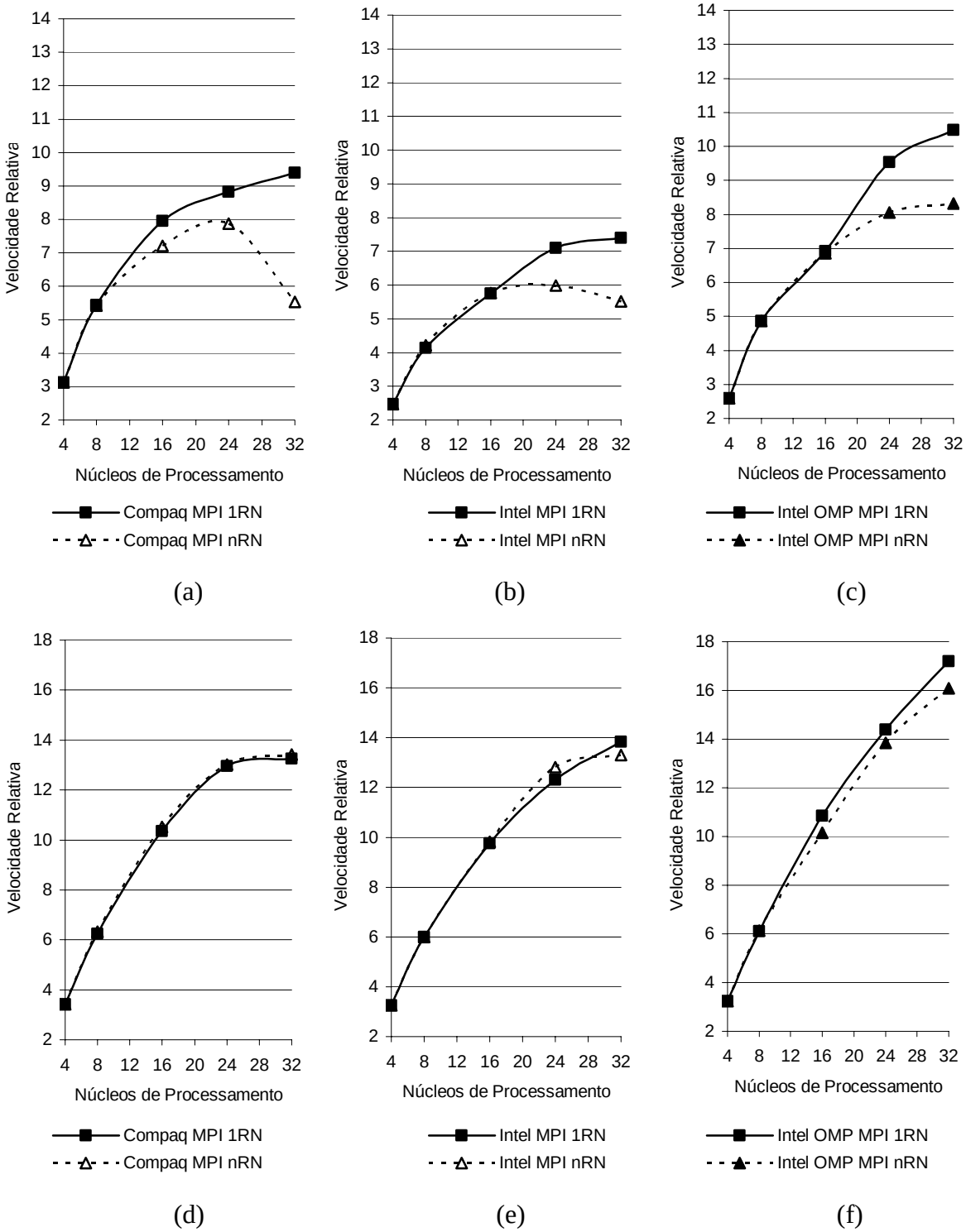


Figura 6.10: Velocidades relativas obtidas em *cluster* permanente com processadores multi-núcleos para o problema NS1 com (a) compilador Compaq v.6.6 / MPI, (b) compilador Intel v.9.1 / MPI, (c) compilador Intel v.9.1 / OpenMP + MPI e para o problema NS2 com (d) compilador Compaq v.6.6 / MPI, (e) compilador Intel v.9.1 / MPI, (f) compilador Intel v.9.1 / OpenMP + MPI.

Os valores máximos de *speed-up* obtidos para o código com o compilador Compaq e tratamento 1RN foram de 7,9 e 10,4 para as malhas NS1 e NS2, respectivamente, e 7,2 e 10,5 para o tratamento nRN. Para comparação, os valores obtidos com o *cluster* de 16 computadores com processadores de núcleo único e uma rede de 1Gbps foram 7,8 e 10,8 para o tratamento 1RN e 10,3 e 12,1 para o tratamento nRN. A existência de uma interface de rede única para 4 núcleos realizando as operações de comunicação através dela de forma simultânea e concorrente levou a uma séria penalização em termos de latência na comunicação através da rede com o *cluster* com processadores de núcleos múltiplos, fator agravado pela grande quantidade de processos de comunicação decorrentes do grande número de elementos conectados em cada nó nas malhas de tetraedro. Esses fatores fizeram com que os resultados obtidos com o tratamento nRN no *cluster* com processadores de núcleos múltiplos fossem piores que os obtidos com o tratamento 1RN, especialmente na malha menor (NS1). O tratamento 1RN, por acarretar um número mínimo de operações de comunicação em rede, impediu que a interface única de rede para atender a vários núcleos se constituísse em um grande gargalo para o desempenho. Os valores de *speed-up* obtidos no *cluster* com processadores com múltiplos núcleos fosse equivalente ao obtido no *cluster* com processadores de núcleo único.

O uso de OpenMP em conjunto com MPI resultou nos maiores valores de *speed-u*p, mostrando ser a alternativa mais eficiente para problemas discretizados por malhas não estruturadas em c*lustesr* cujos nós são computadores com processadores de múltiplos núcleos. O número de operações de comunicação de dados em rede é bastante reduzida em ralação às implementações utilizando somente MPI, o que foi altamente benéfico para o desempenho em malhas de tetraedros onde a grande quantidade de elementos conectados a um mesmo nó é critica, seja quanto ao tempo de comunicação em rede, seja na redundância do esforço computacional.

A implementação com o compilador Compaq apresentou velocidades de processamento bastante superior às obtidas com o compilador Intel para a malha menor (NS1), e equivalente para a malha maior (NS2). Contudo, a muito maior eficiência de paralelização da implementação empregando OpenMP e MPI em conjunto mais do que compensou o código bastante mais lento do compilador Intel para a malha menor, tornando-se a alternativa mais rápida para os dois tamanhos de malha testados.

Os resultados para a fração serial de Karp-Flatt estão mostrados nas figuras 6.11 e 6.12 para as malhas NS1 e NS2, respectivamente. Para a malha menor (NS1), todas as soluções com o tratamento nRN apresentaram crescimento da fração serial com o número de processadores, indicando o efeito negativo de de um grande número de processos de comunicação sobre a escalabilidade. As soluções com o tratamento 1RN apresentaram fração serial aproximadamente constante, com valores médios inferiores aos obtidos com o tratamento nRN. Para a malha maior, os resultado obtidos para todas as implementações são equivalentes, e indicam boa escalabilidade.

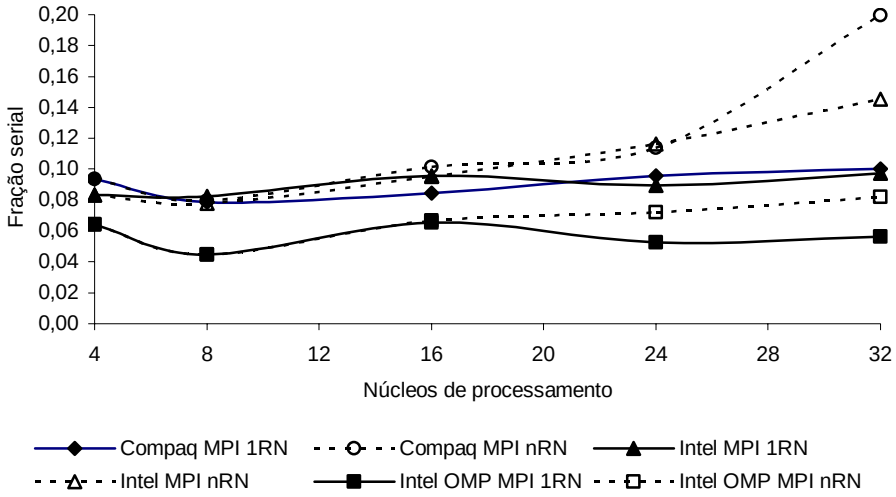


Figura 6.11: Frações seriais obtidas em *cluster* permanente com processadores multi-núcleos para o problema NS1.

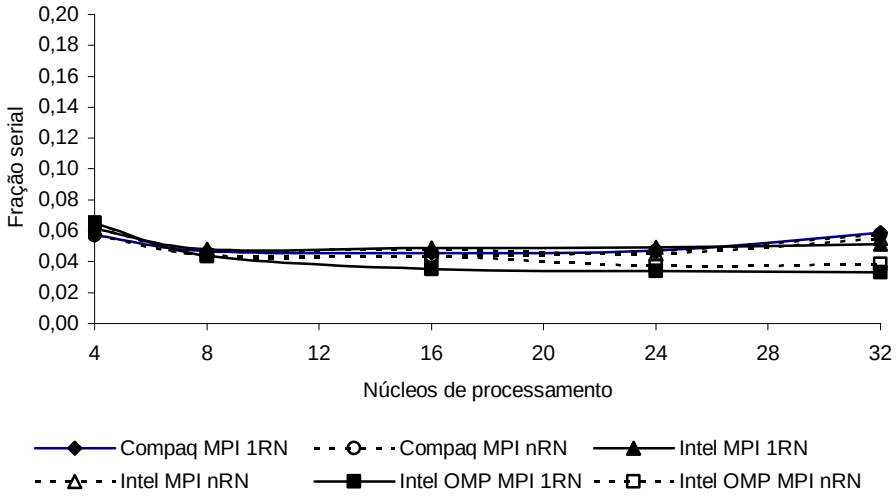


Figura 6.12: Frações seriais obtidas em *cluster* permanente com processadores multi-núcleos para o problema NS2.

6.5 SISTEMATIZAÇÃO DOS RESULTADOS

Com base nos resultados obtidos, pode-se identificar as seguintes tendências de comportamento das implementações paralelas testadas:

a) Em computadores com configurações de memória compartilhada, especificamente com um único processador com múltiplos núcleos, uma abordagem paralela utilizando somente MPI, ou seja, tratando cada núcleo do processador como um nodo de *cluster* ou computador independente, é a alternativa que apresenta maior eficiência e maior velocidade de processamento. Sua implementação, contudo, é bastante mais trabalhosa que uma equivalente utilizando OpenMP, pela exigência de maior organização dos dados, sincronização e divisão de tarefas inerente a um paralelismo baseado na troca de mensagens entre os processos.

b) Em *clusters* compostos por computadores com processadores multi-núcleos, a abordagem paralela ideal depende do tipo de malha de elementos finitos utilizada na discretização espacial do problema. Em malhas estruturadas de hexaedros, o número de elementos conectados a um mesmo nó é relativamente baixo, resultando em pequena quantidade de dados a ser comunicada entre processadores e baixa redundância do esforço computacional. Essas características associadas a um tratamento que minimize o número de processos de comunicação entre núcleos, deixando o maior número de operações internas ao computador (sem o uso da rede) conseguem anular o efeito negativo sobre o desempenho da existência de uma única interface de rede para atender aos múltiplos núcleos do processador e às suas operações de comunicação simultâneas e concorrentes, de modo que a melhor abordagem paralela é aquela baseada inteiramente baseada na troca de mensagens entre os núcleos presentes no *cluster* (MPI), sem o emprego de uma biblioteca de paralelismo de memória compartilhada (OpenMP), e em uma divisão de tarefas com um mínimo de operações de comunicação entre os núcleos (1RN).

c) No mesmo tipo de *cluster*, para malhas não estruturadas de tetraedros, onde o número de elementos conectados a um mesmo nó é muito grande com conseqüências negativas para o desempenho relacionadas à redundância do esforço computacional, ao número de operações de comunicação entre processadores e à quantidade de dados envolvidos nessas operações, a abordagem paralela mais indicada é a que utiliza

memória compartilhada (OperMP) para a mobilização dos vários núcleos dentro de um nodo do *cluster*, e troca de mensagens através da rede (MPI) para a comunicação entre nodos do *cluster*.

d) Pela existência de uma interface de rede para cada núcleo, *clusters* formados por computadores com processadores de núcleo único tendem a ser mais eficientes que os formados por computadores com processadores de núcleos múltiplos nas configurações adequadas ao paralelismo que utiliza somente a troca de mensagens (MPI). Contudo, é importante lembra que, para o mesmo número de núcleos de processamento, uma configuração de hardware baseada em processadores de núcleos múltiplos é bastante mais barata que a com processadores de núcleo único, por evitar a duplicação de componentes como fontes, placas-mãe, unidades de armazenamento, etc. Essa diferença de custo poderia ser investida em processadores mais potentes, resultando em um conjunto com maior velocidade de processamento em paralelo, apesar de menos eficiente.

7 CONCLUSÕES E SUGESTÕES

7.1 CONCLUSÕES

Neste trabalho foi desenvolvida uma metodologia de paralelização para processos de solução iterativos de problemas de mecânica dos sólidos, dinâmica dos fluídos e interação fluído-estrutura utilizando o Método dos Elementos Finitos em configurações de memória distribuída, memória compartilhada ou híbridas que pode ser aplicada para *clusters* temporários e permanentes, homogêneos ou não. Diversos testes foram realizados com diferentes configurações de tamanho de cluster, poder computacional dos computadores envolvidos, velocidade da rede de conexão e tamanho e tipo da estrutura de dados envolvida. Apesar de os testes não esgotarem todas as possíveis combinações, os resultados obtidos permitiram estabelecer as seguintes conclusões:

- Os algoritmos iterativos de solução mostraram-se adequados para a paralelização, particularmente para configurações de memória distribuída. A divisão de tarefas entre os processadores lógicos envolvidos é relativamente simples de ser feita, bem como a identificação das variáveis envolvidas na comunicação entre processadores. Enquadram-se nessa classificação o Método dos Gradientes Conjugados para a solução de sistemas de equações lineares e os esquemas explícitos de Taylor-Galerkin de dois passos e de um passo com iterações para a solução de problemas de dinâmica dos fluídos. As características desses algoritmos permitiram que fosse adotada uma estratégia de paralelização sem processador ou computador mestre (*Hostless program*), resultando em baixa comunicação de dados entre os processadores.

- As escolha de um processo de divisão de tarefas baseada em nós mostrou-se adequada para as estruturas de dados decorrentes do Método dos Elementos Finitos, uma vez que o método resulta na solução de sistemas de equações envolvendo variáveis nodais. A identificação dos dados necessários para a comunicação entre processadores lógicos

envolvidos na solução paralela bem como da redundância de esforço computacional intrínseca a essa divisão é diretamente relacionada com os dados da malha utilizada na discretização do problema.

- A divisão de tarefas baseada em nós fundamenta-se na suposição de que o esforço computacional associado aos nós da malha é constante ao longo de toda a estrutura de dados. Características como cargas, condições de contorno, não linearidades físicas e processos adaptativos espaciais e temporais não são uniformes para toda a malha, a real distribuição do esforço computacional ao longo dos nós não é constante. Além disso, boa parte do esforço computacional está associado ao cálculo das matrizes dos elementos, de modo que a variabilidade do número de elementos conectados a cada nó também contribui para a não homogeneidade da distribuição do esforço computacional, especialmente em malhas não estruturadas. O balanceamento das cargas de trabalho empregado mostrou-se eficaz o suficiente para minimizar os efeitos da não uniformidade do esforço computacional e para permitir o ajuste das cargas de trabalho alocadas a cada processador quando um indicador simples e imperfeito como a freqüência do processador é utilizado como base para uma divisão inicial de tarefas em *clusters* heterogêneos.

- Os tratamentos de malha empregados para a minimização tanto da quantidade de dados comunicados entre processadores lógicos a cada iteração do processo de solução quanto da redundância do esforço computacional apresentaram um bom desempenho quanto à eficiência de paralelização obtida. O tratamento 1RN mostrou-se mais adequado quando o número de processadores lógicos não é muito grande em uma configuração de memória distribuída, quando a latência de comunicação em rede é grande para essa mesma configuração ou em configurações híbridas de memória distribuída e memória compartilhada, como em *clusters* cujos nós possuem processadores de múltiplos núcleos. O tratamento nRN mostrou-se particularmente interessante em *clusters* com um número maior de nodos cujos processadores são de núcleo único.

- A forma de ordenamento das comunicações em rede empregada foi necessária para que, com o tratamento nRN, fossem obtidos tempos de comunicação em rede relativamente baixos.

- Embora otimizada para configurações de memória distribuída, a forma de paralelização escolhida e as técnicas empregadas para aumentar sua eficiência fizeram com que os códigos gerados sejam adequados para uso em configurações de memória compartilhada e em configurações híbridas (*clusters* de computadores com processadores de múltiplos núcleos). Os resultados obtidos em termos de *speed-up*, eficiência de paralelização e velocidade de processamento nessas configurações, para problemas que usem uma malha estruturada de elementos hexaédricos na discretização espacial foram melhores com o uso somente de MPI que com o emprego de OpenMP ou OpenMP em conjunto com MPI. Quando a discretização espacial utiliza malhas não estruturadas de tetraedros, o emprego de OpenMP, sozinho ou em conjunto com MPI, levou à maior eficiência de paralelização. A velocidade de processamento é altamente dependente do compilador utilizado, de modo que compiladores que oferecem suporte a OpenMP nem sempre são os que geram o código de maior velocidade de processamento.

- A solução paralela de problemas utilizando malhas não estruturadas é bastante menos eficiente que a de problemas de tamanho semelhante utilizando malhas estruturadas. Com o uso de tetraedros, o número de elementos conectados a um mesmo nó é, em geral, muito maior e muito mais variável do que quando são utilizados elementos hexaédricos, levando a uma maior redundância do esforço computacional na divisão de tarefas entre processadores em uma configuração de memória distribuída e a uma maior heterogeneidade na distribuição do esforço computacional de solução ao longo da malha. Quanto maior o número de elementos conectados em um nó, maior o número de outros nós relacionados a ele, maior o número de variáveis nodais que são necessárias para o cálculo do valor atualizado das variáveis desse nó em um processo iterativo, maior a quantidade de dados que precisam ser trocados entre processadores a cada passo iterativo. Conseqüentemente, mais importantes se tornam procedimentos eficientes para minimizar a comunicação de dados entre processadores, a redundância do esforço computacional e proporcionar uma divisão de tarefas equilibrada.

- A técnica de sub-ciclos empregada para acelerar a integração no tempo nos esquemas explícitos de Taylor-Galerkin empregados em Dinâmica dos Fluídos adiciona não uniformidade temporal à já existente não uniformidade espacial da distribuição do esforço computacional associado a cada nó da malha utilizada na discretização do

domínio do problema. O fato do esforço computacional associado a grupo de nós ou elementos da malha mudar a cada passo de tempo torna difícil a obtenção de uma distribuição de cargas de trabalho equilibrada em todos os passo de tempo. A divisão de tarefas fixa ao longo do tempo utilizada levou a valores de *speed-up* e eficiência de paralelização baixos, embora a combinação do *speed-ups* proporcionado pela paralelização com aquele proporcionado pela técnica de sub-ciclos tenha produzido ganhos de velocidade de processamento notáveis.

- A escolha de um algoritmo com tratamento particionado para a solução de problemas de interação fluído-estrutura levou a uma baixa eficiência de paralelização em função do fluído e da estrutura serem considerados como entidades isoladas, com estrutura de dados e distribuição de tarefas próprias. Embora os algoritmos utilizados para a solução do fluído e da estrutura individualmente tenham apresentado boa escalabilidade na paralelização, a junção dos mesmos com um tratamento particionado deu origem a uma grande quantidade de dados trafegando entre os processadores. A eficiência obtida no processo foi considerada insatisfatória.

- A implementação utilizada foi totalmente baseada nas capacidades apresentadas pela linguagem de programação utilizada (Fortran 90) e nas bibliotecas de comunicação empregadas (MPI e OpenMP). Nenhuma característica especial do sistema operacional foi utilizada além do suporte à comunicação em rede. Os códigos assim desenvolvidos podem ser utilizados em versões normais do sistema operacional, sem a necessidade de versões específicas para *clusters*, tanto em *clusters permanentes* especialmente configurados para o processamento puro como em *clusters temporários,* utilizando as estações de trabalho e a infra-estrutura de rede normalmente disponíveis nos laboratórios de pesquisa e desenvolvimento.

- A configuração que garante a maior eficiência de paralelização é aquela na qual os processadores envolvidos têm o mesmo poder computacional. Configurações heterogêneas apresentam menor rendimento, mas as técnicas empregadas para divisão e balanceamento das cargas de trabalho em tais configurações proporcionaram uma boa eficiência para a as diferenças de poder computacional típicas das várias gerações de um processador dentro de uma mesma arquitetura.

7.2 SUGESTÕES PARA TRABALHOS FUTUROS

Com base nos resultados obtidos, algumas sugestões para trabalhos futuros puderam ser formuladas para esclarecer alguns tópicos que não puderam ser melhor estudados neste trabalho ou para estender sua aplicação a outros campos. São elas:

- O estudo sobre a paralelização de problemas de Mecânica dos Sólidos foi bastante limitado e restringiu-se a problemas estáticos de elasticidade linear, onde o cálculo das matrizes de rigidez e de cargas e o posterior cálculo de tensões é naturalmente feito elemento a elemento e portanto, totalmente desacoplados, permitindo facilmente a paralelização, e onde a solução do sistema de equações é a responsável pela maior parte do tempo de processamento necessário. Problemas dinâmicos e com não linearidade física e geométrica necessitam de grande esforço computacional e demandam grandes tempos de processamento, sendo áreas de grande interesse. Problemas com efeitos de plasticidade e dano contínuo alteram a distribuição espacial do esforço computacional, exigindo cuidados com a distribuição de tarefas para manter a eficiência de paralelização. O tratamento dado a problemas de Dinâmica dos Sólidos Computacional dentro do estudo de Interação Fluído-Estrutura pode ser considerado preliminar e precisa ser bastante aprofundado.

- Problemas com não uniformidade espacial e temporal do esforço computacional relacionado aos elementos da malha trazem desafios para que a eficiência de paralelização seja obtida e mantida ao longo do processo de solução. O emprego da técnica de sub-ciclos em problemas transientes de dinâmica dos fluídos onde o passo de tempo empregado em cada elemento depende não somente do tamanho do elemento mas também dos gradientes das variáveis nodais é um exemplo desse tipo de problema e exige uma abordagem dinâmica para o problema de divisão de tarefas e balanceamento das cargas de trabalho.

- A Utilização de uma divisão de tarefas baseada em particionamento dual (por elementos), para comparação de desempenho com o método empregado.

- O processo de divisão de tarefas entre processadores utilizado pode ser melhorado pela adição de um custo computacional associado aos nós que leve em consideração o número de elementos conectados a cada nó, ponderando a quantidade de nós alocada a

cada processador. Permitir um certo desbalanceamento de cargas no particionamento da malha pela busca na região provável de divisão do sistema de equações dos pontos onde o sistema apresenta largura de banda mínima. A diminuição na quantidade de dados associados às fronteiras dos subdomínios e do número de subdomínios vizinhos podem mais do que compensar o desbalanceamento de cargas em termos de tempo de execução.

- O ordenamento da comunicação entre processadores pode ser melhorado pela identificação de etapas em que 2 ou mais processadores estão em estado de espera, mas a comunicação entre eles ainda não ocorreu. Estas situações podem ser identificadas *a priori* pela análise da tabela de incrementos, a qual pode ser modificada alterando a ordem dos incrementos de alguns processadores, levando a possíveis reduções no número total de etapas de comunicação do conjunto.

- A abordagem utilizada na comunicação de dados entre processadores no algoritmo com tratamento particionado para a análise de problemas de interação fluído-estrtura mostrou-se particularmente ineficiente. Um melhor estudo da estrutura de dados para evitar o máximo possível uma configuração do tipo mestre-escravo faz-se necessário.

- A aplicação dos algoritmos desenvolvidos em *clusters* com um número maior de computadores com processadores de múltiplos núcleos precisa ser melhor estudada para se identificar possíveis gargalos ao desempenho e formas de corrigi-los.

- Desenvolvimento de rotinas que permitam a detecção de falha de um processador lógico, de modo a realizar de modo automático a exclusão do processador do processo e a reconfiguração do *cluster* com nova divisão de tarefas e balanceamento.

- A verificação da adequabilidade da forma de paralelização desenvolvida em processadores com um número de núcleos lógicos maior que o de núcleos físicos, e com poderes computacionais diferentes por núcleo precisa ser estudada para a identificação de possíveis gargalos ao desempenho e formas de corrigi-los.

- Um grande ganho na velocidade de processamento pode ser obtida em certos algoritmos ao utilizarem-se os processadores gráficos atualmente disponíveis para o processamento de códigos numéricos (GPGPU). Verificar a adequabilidade dos algoritmos de solução empregados para os problemas de Mecânica dos Sólidos e de

Dinâmica dos Fluídos nesse tipo de processadores, bem como estudar a forma de paralelização para *clusters* compostos por computadores com processadores de múltiplos núcleos e múltiplos processadores gráficos parece ser o caminho a ser seguido para a obtenção de computação de alto desempenho com baixos custos.

- Formas de baixo custo de melhorar a velocidade de comunicação entre computadores podem ser testadas, como a utilização de mais de uma interface de rede em computadores de múltiplos núcleos e o desenvolvimento de técnicas de comunicação entre os núcleos ou processadores que delas façam uso.

REFERÊNCIAS BIBLIOGRÁFICAS

AMD Família de Processadores AMD Opteron. Disponível em < http://www.amd.com/br-pt/processors/productInformation/0,,30_118_8825,00.html > Acesso em: dezembro de 2008.

AMDAHL, G. Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities", AFIPS CONFERENCE **Proceedings..**, (30), pp. 483-485, 1967.

ARGONE NATIONAL LABORATORY. MPICH – A portable implementation of MPI. Disponível em < http://www-unix.mcs.anl.gov/mpi/mpich1/index.htm> . Acesso em setembro de 2005.

ARGYRIS, J.; DOLTSINIS, I. S.; FRIZ, H. Study on computational reentry aerodynamics. **Computer Methods in Applied Mechanics and Engineering**. v.81, p. 257–289. 1990.

AZEVEDO, R.L. **Análise de problemas de interação fluído-estrutura usando o método dos elementos finitos com um acoplamento monolítico.** Porto Alegre, RS. 1999. Tese (Doutorado). Programa de Pós-Graduação em Engenharia Civil. Universidade Federal do Rio Grande do Sul.

BARNEY, B. Introduction to parallel computing. Disponível em < https://computing.llnl.gov/ tutorials/parallel_comp/ >.*Lawrence Livermore National Laboratory*. Acesso em maio e 2007.

BATHE, K.J. **Finite element procedures.** Englewood Cliffs, New Jersey¨Prentice Hall, 1996.

BELYTSCHKO, T.; GILBERTSEN, N. D. Implementation of Mixed Time Integration Techniques on a Vectorized Computer with Shared Memory. **International Journal of Numerical Methods in Engineering**. v.35, p. 1803-1828. 1992.

BELYTSCHKO, T.; LU, Y. Y. Explicit Multi-time step integration for first and second order finite element semidiscretizations. **Computational. Methods in Applied. Mechanics and Engineering**. v.108. p. 353-383. 1993.

BELYTSCHKO, T.; YEN, H. J.; MULLEN, R. Mixed Method for Time Integration. **Computational. Methods in Applied Mechanics and Engineering**. v. 17/18. p. 259-275. 1979.

BENNETT, R. M.; WALKER, C. E. Computational Test Cases for a Clipped Delta Wing With Pitching and Trailing-Edge Control Surface Oscillations, *TM-209104, NASA*, 1999.

BENZI, M. Preconditioning techniques for large linear systems: a survey. **Journal of Computational Physics**, v.182, p. 418-477. 2002.

BONO, G. **Simulação numérica de escoamentos em diferentes regimes utilizando o método dos elementos finitos**. Porto Alegre, RS. 2008, 186p. Tese (Doutorado). Programa de Pós-Graduação em Engenharia Mecânica. Universidade Federal do Rio Grande do Sul.

BRAUN, A. L. B. **Simulação numérica na engenharia do vento incluindo efeitos de interação fluído-estrutura.** Porto Alegre, RS. 2007, 283p. Tese (Doutorado). Programa de Pós-Graduação em Engenharia Civil. Universidade Federal do Rio Grande do Sul.

. A Partitioned model for fluid-structure interaction problems using hexahedral finite elements with one-point quadrature. **International Journal for Numerical Methods in Engineering**, 2009 (accepted for publication).

BRAUN, A.L. Um modelo para simulação numérica da ação do vento sobre seções de pontes. Porto Alegre, RS. 2002.. Dissertação (Mestrado). Programa de Pós-Graduação em Engenharia Civil. Universidade Federal do Rio Grande do Sul.

BURBRIDGE, H. P. A finite element Taylor-Galerkin scheme for three-dimensional numerical simulation of high compressible flows with analytical evaluation of element matrices**. Hybrid methods in engineering**, [S.l.] v. 2, n. 4, p. 485-506, 2000.

BURBRIDGE, H. P. **O esquema explícito de Taylor-Galerkin na simulação numérica de escoamentos compressíveis tridimensionais utilizando elementos finitos hexaédricos de oito nós**. Porto Alegre, RS. 1999, 140p. Dissertação (Mestrado). Programa de Pós-Graduação em Engenharia Civil. Universidade Federal do Rio Grande do Sul.

CATALYUREK, U., BOMAN, E., DEVINE, K., BOZDAG, D., HEALPHY, R., RIESEN, L.A. Hypergraph-based Dynamic Load Balancing for Adaptive Scientific Computations. **Proceedings of IPDPS'07**, Best Algorithms Paper Award, March 2007.

CHANG, H. J.; BASS, J. M.; TWORZYDLO, W.; ODEN, J. T. H-P Adaptive Methods for Finite Element Analysis of Aerothermal Loads in High-speed Flows. *CR-189739, NASA*, 1993.

CHORIN, A.J. A numeical method for solving incompressible viscous flow problems. **Journal of Computational physics,** v. 105, p.207-223. 1967

COURANT, R., FRIEDRICHS, K. e LEWY, H. Über die partiellen Differenzengleichungen der mathematischen Physik, *Mathematische Annalen*, vol. 100, no.1, pages32–74, 1928. – Tradução para o inglês em COURANT, R., FRIEDRICHS, K. e LEWY, H. "On the partial difference equations of mathematical physics", *IBM Journal*, March 1967, pp. 215-234. Disponível em < http://www.stanford.edu/class/cme324/classics/courant-friedrichs-lewy.pdf > Acesso em março de 2007.

CUMINATO, J.A., CASTELO FILHO, A., BOAVENTURA, M. TOMÉ, M.F. Simulation.of free surface floews in a distributed memory environment. **Journal of Computational and Applied Mathematics,** 1999; **103:**77-92.

DEMKOWICZ, L., KURTZ, J., PARDO, D., PASZYNSKI, M., RACHOWICZ, W. E ZDUNEK A. **Computing with hp-Adaptative Finite Elements Vol.2 Frontiers: three dimensional elliptc and Maxwell problems with Applications**. Chapman & Hall / CRC, 2007. 417p.

DEMMEL, J. CS267 Applications off Parallel Computers. Lecture 23: Load Balancing and Scheduling. Disponível em < http://www.cs.berkeley.edu/~demmel/cs267_Spr99/ Lectures/Lect_23_1999.pdf > Acesso em março de 2007.

DONEA, J. A. Taylor-Galerkin for convective transport problems. **International Journal for Numerical Methods in Engineering.** v.20, p. 101–119. 1984.

DONEA, J., HUERTA, ª, PONTHOT, J. Ph., RODRÍGUEZ-FERRAN, A. Arbitrary Lagrangian–Eulerian Methods. In: *Encyclopedia of Computational Mechanics*, Edited by Erwin Stein, Ren´e de Borst and Thomas J.R. Hughes. Volume 1: *Fundamentals*. 2004 John Wiley & Sons, Ltd. ISBN: 0-470-84699-2. Chapter 14. Disponível em < http://www.wiley.co.uk/ecm/pdfs/Volume_1_Chapter_14.pdf > Acessado em março de 2008.

DORNELES, R. V. **Particionamento de Domínio e Balanceamento de Cargo no modelo HIDRA.** Porto Alegre, RS. 2003, 136p. Tese (Doutorado). Programa de Pós-Graduação em Computação. Universidade Federal do Rio Grande do Sul.

DUARTE FILHO, L. A. **Análise Estática e dinâmica, linear e não-linear geométrica, através de elementos hexaédricos de oito nós com um ponto de integração**. Porto Alegre, RS. 2002, 111p. Dissertação (Mestrado). Programa de Pós-Graduação em Engenharia Civil. Universidade Federal do Rio Grande do Sul.

ELGELMAM, M.; SANI, R. L.; GRESHO, P. M.; BERCOVIER, M. Consistent vs. Reduced Integration Penalty Methods for Imcompressible media using general old ans new elements. **J. Numer. Methods in Fluids**, v. 2, p. 25-42. 1982.

ELIAS, R.; MARTINS, M. E.; COUTINHO, A. Paralelismo de Programas com Malhas não estruturadas. Núcleo de Atendimento em Computação de Alto Desempenho (NACAD), UFRJ. Disponível em < http://arquivosevt.lncc.br/pdfs/M15.pdf> Acesso em julho de 2008.

FÖRSTER, C.; WALL, W. A.; RAMM, E. Artificial added mass instabilities in sequential staggered coupling of nonlinear structures and incompressible viscous flows. **Computer Methods in Applied Mechanics and Engineering**. v. 196, p. 1278-1293. 2007.

FOX, G. C., WILLIAMS, R. D.; MESSINA, P.C. **Parallel Computing Works** [S.I.], Morgan Kaufmann Publishers, San Francisco, USA, 1994. ISBN 1-55860-253-4. 977p.

GONZÁLEZ, L.A.S. Análise de escoamentos de fluídos quase incompressíveis e das vibrações induzidas em objetos imersos. Porto Alegre, RS. 1993.. Dissertação (Mestrado). Programa de Pós-Graduação em Engenharia Civil. Universidade Federal do Rio Grande do Sul.

GRECHO, P. M.; CHAN, S. T.; LEE, R. L.; UPSON, G. D. A modfied finite element method for solving the time dependent, imcompressible..Navier-Stokes Equations. PartI: Theory. **International Journal for Numerical Methods in Fluides.** v.4, p. 557–598. 1984

GRECHO, P.M. e SANI, R.L. **Incompressible flow and the finite element method**. Sussex, UK: John Wuiley & Sons Ltd., 1999

GUSTAFSON, J. Reevaluating Amdahl's Law. **Communications of the ACM**. v.31, issue 5, p. 532-533. May, 1988.

HENDRICKSON, B. DEVINE, K. Dynamic load balancing in computational mechanics, **Comp. Meth. Applied Mechanics & Engineering** , v.184(2-4), p. 485-500. 2000.

HENDRIKSON, B.; LELAND, R. An improved spectral graph partitionaing algorithm for mapping parallel computations. **SIAM Journal Scientific Computing**. v. 16, n. 2, p. 452-469. 1995.

HENDRIKSON, B.; LELAND, R. Chaco: software for partitioning graphs. Disponível em < http://www.cs.sandia.gov/~bahendr/chaco.html > Acesso em junho de 2008.

HINTON, E.E.; OWEN, D.R.J. **Finite Element Programming**. Academic Press, London, 1977. 305p.

HOIT, M., WILSON, F. L. An equation numbering algorithm based on a minimum FRONT criteria. **International Journal for Numerical Methods in Engineering**. v.16, p. 225-239. 1983.

HOOKER, J. R.; BATINA; J. T.; WILLIAMS, M. H. Spatial and temporal adaptive procedures for the unsteady aerodynamic analysis of airfoils using unstructured meshes. *Technical Memorandum TM-107635*, *NASA*, 1992.

HUGHES, J. R.; FERENCZ, R. M. Large-scale vectorized implicit calculation in solid mechanics on a CRAY X-MP/48 utilizing EBE preconditioned conjugate gradients**. Computer Methods in Applied Mecghanics and Engineering**. v. 61, p. 215-248. 1987.

HUGHES, T. J. R.; LIU, W. K. Implicit-explicit Finite Element in Transient Analysis: Stability Theory. **Journal of Applied. Mechanics.** v. 45, p. 371-374. 1978.

HUGHES, T. J. R.; TEZDUYAR, T. E. Finite element methods for first-order hyperbolic systems with particular emphasis on the compressible Euler equations. **Computer Methods in Applied Mechanics and Engineering**. v. 45, p. 217–284. 1984.

INTEL Processadores para servidor Intel. Disponível em < http://www.intel.com/portugues/products/server/processors/index.htm > Acesso em dezembro de 2008.

IRONS, B.M. A frontal solution scheme for finite element analysis. **International Journal for Numerical Methods in Engineering**. v. 2, n.1, p. 5-32. 1970.

JERNELL, L. S. Comparisons of theoretical and experimental pressure distributions over a wing-body model at high supersonic speeds. *TN D-6480*, NASA, 1971.

KALLINDERIS, Y.; VIJAYAN, P. Adaptive refinement-coarsening scheme for three-dimensional unstructured meshes. **AIAA Journal**. v. 38, p. 1440-1447. 1993.

KARP, A. H.; FLATT, H. P. Measuring Parallel Processor Performance, **Communications of the ACM. v.** 33, n. 5, p. 539-543. May 1990.

KARYPIS LAB. Metis – Serial graph partitioning and fill-reducing matrix ordering. Disponível em < http://glaros.dtc.umn.edu/gkhome/metis/metis/overview >. Acesso em novembro de 2008.

KARYPIS, G., KUMAR, V. METIS – Family of multilevel partitioning algorithms. Disponível em < http://glaros.dtc.umn.edu/gkhome/views/metis > Acesso em junho de 2008.

KAWAHARA, M.; HIRANO, H. A Finite Element Method for High Reynolds Number Viscous Fluid Flow Using Two Step Explicit Scheme. **International Journal of Numerical Methods in Fluids.** v.3, p.137-163. 1983

KESSLER, M.P. **Simulação numérica de escoamentos hipersônicos em não-equilíbrio termo-químico através do método dos elementos finitos**. Porto Alegre, RS. 2002, 122p. Tese (Doutorado). Programa de Pós-Graduação em Engenharia Mecânica. Universidade Federal do Rio Grande do Sul.

KWAK, D., KIRIS, C., KIM, C.S. Computational challenges of viscous imcompressible flows, **Computer & Fluids**, v. 34, p.283-299. 2005.

KWOK, K.; LAM, F.; HAMDI, M.; PAN, Y.; HUI, C. *Application-Specific Load Balancing on Heterogeneous Systems.* In: Buyya, R. (Ed.). **High performance Cluster Computing. Vol. 2 Programming and Applications.** New Jersey, USA: Prentice Hall, p.350-374. 1999.

LÖHNER, R. **Applied computational fluid dynamics techniques. an introduction based on finite element methods.** John Wiley & Sons Ltd., England, 2001. 366p.

LÖHNER, R.; MORGAN, K.; ZIENKIEWICZ, O. C. The Solution of Non-linear Hyperbolic Equation Systems by the Finite Element Method**. International Journal for Numerical Methods in Fluids.** v. 4, p. 1043-1063. 1984.

MAURITS, N. M.; VAN DER VEM, H.; VELDMAN, A. E. P. Explicit multi-time stepping for convection-dominated flow problems. **Computational Methods in Applied Mechanics and Engineering.** v. 157, p. 133-150. 1998.

MURARO JR., A.; PRETO, A. J.; STEPHANY, S.; PASSARO, A.; LIMA, O.F.; ABE, N.M.; TANAKA, R.Y. "Implementação Paralela do Método dos Gradientes Conjugados para Solução de Sistemas Esparsos de Equações Lineares. Disponível em< http://hermes2.dpi.inpe.br:1905/archive.cgi/lac.inpe.br/worcap/2004/10.05.16.08> Acesso em: setembro de 2008.

OLIVEIRA JR., J. A. A. **Desenvolvimento de um sistema de dinâmica dos fluídos computacional empregando o método de elementos finitos e técnicas de ato desempenho**. Porto Alegre, RS. 2006, 99p. Dissertação (Mestrado). Programa de Pós-Graduação em Engenharia Mecânica. Universidade Federal do Rio Grande do Sul.

OPENMP Arquitechture Review Board.. The OpenMP API specification for parallel programming. Disponível em < http://openmp.org/wp/ > . Acesso em março de 2008.

PELLEGRINI, F. A parallelisable multi-level banded diffusion scheme for computing balanced partitions with smooth boundaries. **EuroPar,** Rennes. Spring Verlag, publicado em LNCS series, LNCS 4641, pp 191-200. Ago 2007.

PELLEGRINI, F. Scotch & PT-Scotch: Software package and libraries for sequential and parallel graph partitioning, static mapping, and sparse matrix block ordering, and sequential mesh and hypergraph partitioning. Disponível em < http://www.labri.u-bordeaux.fr/perso/pelegrin/scotch/ > Acessado em junho de 2008.

PETRY, A. P. **Análise numérica de escoamentos turbulentos tridimensionais empregando o método dos elementos finitos e simulação de grandes escalas**. Porto Alegre, RS. 2002, 135p. Tese (Doutorado). Programa de Pós-Graduação em Engenharia Mecânica. Universidade Federal do Rio Grande do Sul.

PETRY, A.P. Análise numérica de interação fluído-estrutura através do método dos elementos finitos. Porto Alegre, RS. 1993.. Dissertação (Mestrado). Programa de Pós-Graduação em Engenharia Civil. Universidade Federal do Rio Grande do Sul.

PEZZI, G.P. **Escalonamento *Work-Stealing* de programas Divisão-e-Conquista com MPI-2.** Porto Alegre, RS. 2006, 69p. Dissertação (Mestrado). Programa de Pós-Graduação em Computação. Universidade Federal do Rio Grande do Sul.

PHON, K.K., TOH, K.C., CHAN, S.H. e LEE, F.H. An efficient diagonal preconditioner for finite element solution of Biot's consolidation equations. **International Journal for Numerical Methods in Engineering,** v. 55, p. 377-400. 2002

POPIOLEK; T.L.; AWRUCH, A.M. Numerical Simulation of Incompressible Flows using Adaptive Unstructured Meshes and the Pseudo-compressibility Hypothesis. **Advances in Engineering Software. v. 37, p.** 260-274. 2006.

PREISS, R., DIEKMANN, R. PARTY partitioning library. Disponível em < http://wwwcs.uni-paderborn.de/fachbereich/AG/monien/RESEARCH/PART/party.html > Acesso em junho de 2008.

REDDY, J.N. e GARTLING, D.K. **The finite element method in the heat transfer and fluid dynamics.** Boca Raton: CRC Press, 1994.

SAAD, Y. **Iterative methods for sparse linear systems**, PWS Publishing Company, 20 Park Plaza Boston, MA 02116. 1996

SAAD, Y.; VANDERVORST, H. A. Iterative solution of linear systems in the 20th century, *J. Computational and Applied Mathematics* v.122: p.1–33. 2000

SGI. SGI NUMAlink interconnect fabric.. Disponível em < http://openmp.org/wp/ > . Acesso em abril de 2008.

SHEWCHUK, J.R. An introdution to the conjugate gradient method without the agonizing pain. Disponível em < http://reference.kfupm.edu.sa/content/i/n/ an_introduction_to_the_ conjugate_gradien_55475.pdf >. Acessado em maio de 2004.

SHULTZ, S.L. 1997. **Elementos Finitos Tri-Lineares com Integração reduzida e controle de modos espúrios na Análise Linear de Placas e Cascas.** Porto Alegre, RS. 1997. Dissertação (Mestrado). Programa de Pós-Graduação em Engenharia Mecânica. Universidade Federal do Rio Grande do Sul.

SILVESTER, P. P.; AUDA, H.A. A memory economic frontwidth reduction algorithm. **International Journal for Numerical Methods in Engineering**. v. 20, p. 733-743. 1984.

TEIXEIRA, F. G. Sistema de Reordenação Nodal para Soluções Tipo Banda. Porto Alegre, RS. 1991, 98p. Dissertação (Mestrado). Programa de Pós-Graduação em Engenharia Civil. Universidade Federal do Rio Grande do Sul.

TEIXEIRA, P. R. F.; AWRUCH, A. M. Three-dimensional Simulation of High Compressible Flows using a Multi-time-step Integration Technique with Sub-cycles. **Applied Mathematical Modeling**. v. 25, p. 613-627. 2001.

TEIXEIRA, P.R.F. **Simulação numérica de escoamentos tridimensionais de fluídos compressíveis e incompressíveis e estruturas deformáveis usando o método dos elementos finitos**. Porto Alegre, RS. 2001, 237p. Tese (Doutorado). Programa de Pós-Graduação em Engenharia Civil. Universidade Federal do Rio Grande do Sul.

TOP500. Lists November 2008. Disponível em < http://www.top500.org/lists/2008/11 > Acesso em: dezembro de 2008.

TURKEL, E., RADESPIEL, R., KROLL, N. Assement of preconditioning methods for multidimensional aerodynamics, **Computer & Fluids,** v. 26(6), p. 613-634. 1997.

VAN DER VEM, H.; NIEMANN-TUITMAN, B. E.; VELDMAN, A. E. P. An explicit multi-time-stepping algorithm for aerodynamic flows. **Journal of Computational and Applied Mathematics**. v. 82, p. 423-431. 1997.

VANDERVORST, H. A. (2000). Krylov subspace iteration, **Computing in Science and Engineering.** v. 2, p. 32–37. 2000.

WACKERS, J.; KOREN, B. A simple and efficient space-time adaptive grid technique for unsteady compressible flows. In: 16th AIAA COMPUTATIONAL FLUID DYNAMICS CONFERENCE. 2003. **Proceedings..** AIAA-paper 2003-3825, USA.

WALSHAW,C., CROSS, M. JOSTLE – graph partitioning software. Disponível em < http://staffweb.cms.gre.ac.uk/~c.walshaw/jostle/ > Acesso em junho de 2008.

WILKINSON, B.; ALLEN, M. **Parallel Programing: Techniques and Applications Using Networked Workstations and Parallel Computers**. ISBN 0136717101, 1st Ed., Prentice Hall, Upper Saddle River, N.J. 1999. 467p.

WU, X., GOLUB, G.H., CUMINATO, J.A., YUAN, J.Y. Symmetric-triangular decomposition and its app´lications Part II: preconditioners for indefinite systems. **BIT Numerical Mathematics,** 2008, **48:** 139-162.

XAVIER, C. M. **Análise de Modelos Submalha em Elementos Finitos.** Porto Alegre, RS. 2008, 92p. Dissertação (Mestrado). Programa de Pós-Graduação em Engenharia Mecânica. Universidade Federal do Rio Grande do Sul.

ZIENCKIEWICZ, O.C., TAYLOR, R.L. e NITHIARASU, P., **The finite element method for fluid dynamics.** Oxford: Elsevier Butterworth-Heinemann, 6ª ed., 2005.

A1 OTIMIZAÇÃO DE CÓDIGOS FORTRAN VISANDO MAIOR VELOCIDADE DE EXECUÇÃO

Uma série de cuidados na implementação de um algoritmo podem ser tomados de modo a se obter maiores velocidades de processamento em códigos seqüenciais (executados por um único processador), e que é vantajoso também para implementações paralelas, uma vez que cada processador lógico envolvido pode executar mais rapidamente a tarefa a ele alocada. Alguns desses cuidados estão descritos a seguir, e, apesar de estarem voltados especificamente para implementação de códigos em *FORTRAN*, podem ser aproveitados para implementações em outras linguagens com algumas adaptações.

A1.1 ESCOLHA DO COMPILADOR

A grande maioria dos processadores disponíveis para microcomputadores pessoais, freqüentemente utilizados para processamento científico em instituições de ensino e pesquisa, são capazes de processar o conjunto de instruções x86 da arquitetura IA-32 de 32 bits, estabelecido pela Intel a partir do processador 80386. Isso permite que um mesmo código possa ser executado em qualquer processador posterior ao 80386, seja ele da Intel ou de outro fabricante, como AMD e VIA. Dessa forma, não são necessárias versões específicas de programas comerciais ou sistemas operacionais para cada geração de processador de cada fabricante.

Contudo, a evolução dos processadores desde o 80386 têm se caracterizado por não se limitar a um simples aumento de freqüência de operação em cada família de processadores, modificando também a tecnologia empregada, sempre visando um maior desempenho. Assim, a quantidade de memória *cache*, o número de unidades de processamento de ponto flutuante (*FPU*) e unidades aritméticas e lógicas (*ALU*) tem aumentado, arquitetura interna tem se modificado, e instruções novas tem sido adicionadas ao conjunto x86 para acelerar o processamento de aplicações multimídia (MMX da Intel, 3DNow!, 3DNow!+ da AMD), para

permitir o processamento de múltiplos dados com uma única instrução (*SIMD* como as instruções SSE, SSE2,SSE3 da Intel) e para o processamento também em ambiente de 64 bits (EMT64 da Intel, x86-64 ou AMD64 da AMD). Essas extensões visam otimizar o desempenho do processador para as aplicações mais comuns, que vão se alterando com o passar do tempo. Alguns processadores são otimizados inteiramente para essas instruções adicionais, como por exemplo o Intel Pentium IV, otimizado para a execução de instruções SSE2 ao invés de x86. Em operações de ponto flutuante, um Pentium IV conseguia, usando apenas instruções x86, metade do desempenho obtido com uso de SSE2.

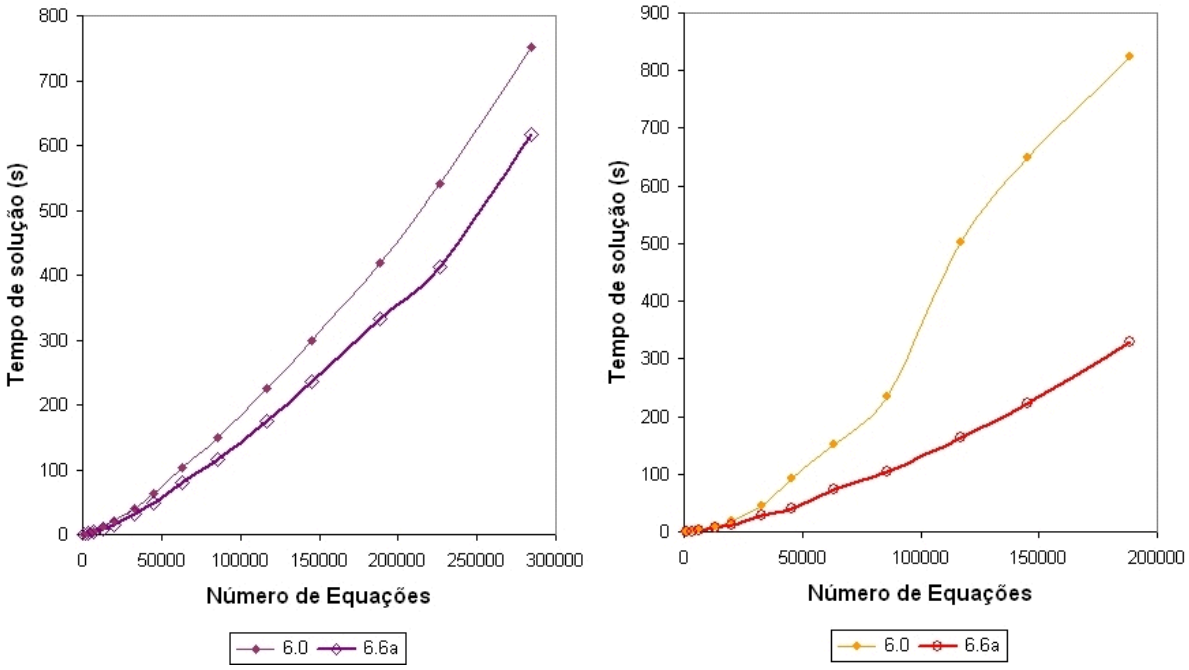
Para que o código gerado por um compilador seja rápido, é preciso que o compilador esteja otimizado para a arquitetura de cada processador, e das extensões (MMX, SSE, 3DNow!, etc) por eles suportadas. Há perda de generalidade do código, uma vez que um dado processador pode não ser capaz de executar um código otimizado para uma arquitetura diferente da sua ao não suportar as extensões dessa arquitetura, mas por outro lado há ganhos significativos no desempenho quando se sabe em qual arquitetura de processadores será utilizado o código.

Pata exemplificar a importância do compilador, o código de análise estática elástica linear com *solver* utilizando o Método dos Gradientes Conjugados foi utilizado para fazer uma análise de Elementos Finitos de um problema de flexão de placa grossa. A mesma geometria foi discretizada através de malhas com elementos cada vez menores, resultando em um número crescente de elementos e equações no sistema.

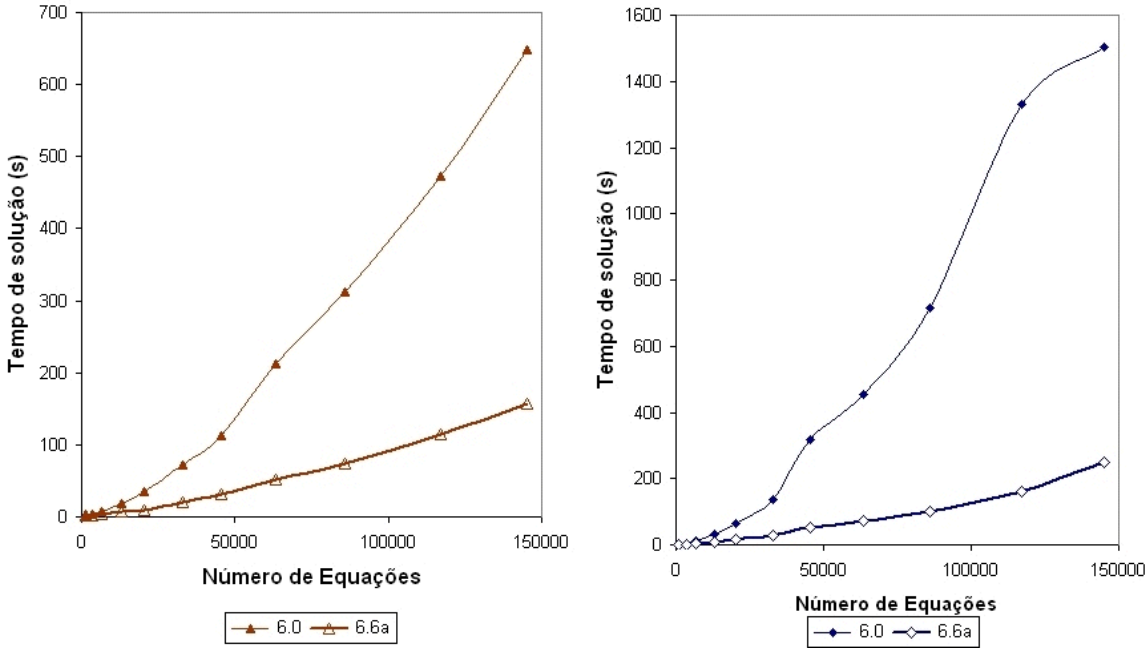
O código foi compilado com o compilador Compaq Visual Fortran versão 6.0, que contava com otimizações apenas até a extensões SSE para arquitetura Intel Pentium III, e com a versão 6.6a, que disponibilizava otimizações para os processadores AMD Athlon e Intel Pentium IV. A evolução do tempo de processamento com o tamanho do problema (número de equações resultantes do tamanho da malha utilizado) está mostrada nas figuras A1.1 para (a) um Intel Pentium III de 1GHz com memória SDR, (b) um Intel Pentium IV *Willamette*de 1,4 GHz com memória RAMBUS, (c) um AMD Athlon Thunderbird de 1,33 GHz com memória SDR e (d) um AMD Athlon Pluto de 0,9 GHz com memória SDR.

Como esperado, os maiores ganhos de velocidade de processamento foram obtidos nos processadores para cuja arquitetura a versão 6.0 não estava otimizada. Contudo, mesmo o Intel Pentium III, cuja arquitetura já era contemplada pela versão 6.0, teve ganho de

desempenho com a versão 6.6a, indicando que o compilador como um todo foi melhorado entre essas versões.



(a) (b)



(c) (d)

Figura A1.1: Diferenças nos tempos de processamento para um mesmo código compilado pelas versões 6.0 e 6.6a do compilador Compaq Visual Fortran para (a) Pentium III 1GHz, (b) Pentium IV 1,4 GHz, (c) Athlon 1,33 GHz e (d) Athlon 0,9 GHz.

Como uma regra geral, as versões mais recentes dos compiladores são as que oferecem o melhor desempenho para as arquiteturas de processadores mais recentes. Contudo, nem todos os compiladores são otimizados para todas as arquiteturas, e nem todas as implementações podem fazer uso das extensões das instruções x86, exigindo modificações na filosofia de programação empregada. Somente o teste permite determinar com certeza o melhor compilador / versão para cada aplicação.

A1.2 DIMENSIONAMENTO DE MATRIZES E USO DO CACHE

Em vetores multi-dimensionais (matrizes) com acesso não-seqüencial na memória, deve-se evitar dimensões que sejam múltiplos de potências de 2, tais como 16, 32, 64, 128, 256, 512, em especial na dimensão mais à esquerda. Essa situação tende a forçar um evento conhecido como erro do *cache* (*cache miss)*, e deriva do fato do tamanho do *cache* do processador ser uma potência de 2, fazendo com que elementos de vetores cujas dimensões são também potências de 2 (ou múltiplos de potências de 2), acessados de forma não contígua (acesso seqüencial de elementos que não estejam em áreas contíguas da memória) tornam o uso do cache ineficiente.

Considere-se um processador imaginário que disponha de uma memória *cache* de 8 posições (0 a 7) com mapeamento direto, ou seja, uma dada posição da memória principal vai ocupar a posição no *cache* dada pelo resto da divisão inteira por 8 (o tamanho do *cache*). Se forem feitas operações com acesso seqüencial à memória, as primeiras 8 posições da memória (0 a 7) ocuparão as 8 posições do *cache*, e, a partir das 8 seguintes, (8 a 15) as posições do *cache* deverão ser desocupadas para que os novos dados ocupem seu lugar. Se as operações envolverem os dados das posições 0 a 7 da memória principal, esses dados serão trazidos da memória principal (mais lenta) para o *cache* (mais rápido) e serão processadas com grande velocidade. Se as operações envolverem somente os dados que ocupam as posições (8 a 15) ou (16 a 23), etc, o mesmo acontece. Se as operações envolverem os dados que ocupam as posições (0; 9; 66; 35; 20;53,38,95) da memória principal, também serão realizadas com grande velocidade, pois essas posições são mapeadas para posições diferentes no cache, sendo acessadas apenas a primeira vez na memória principal, e seguintes no *cache*. Contudo, se os dados a serem processados ocupam na memória principal as posições (0; 8; 16; 24; 32; 40; 48; 56) ou então (3; 19; 11; 59; 43; 27; 35; 51), o processamento será lento, pois as posições da

memória principal de cada conjunto são mapeadas pela mesma posição do *cache* (0 no primeiro conjunto, 3 no segundo). Dessa forma,para o primeiro conjunto, ao ser necessário o dado que está na posição 0 da memória principal, o mesmo é trazido para a posição 0 do *cache*. Ao ser necessário o segundo dado, na posição 8, ele será colocada na mesma posição 0 do *cache.* Se o primeiro dado for novamente necessário, ele deverá ser buscado na memória principal mais uma vez, pois não se encontra mais no *cache*, tornando o processo bastante mais lento.

Matrizes com a primeira dimensão múltipla de uma potência de 2, com armazenamento por coluna (como é o padrão do *FORTRAN*) fazem com que dados sendo acessados seqüencialmente na segunda dimensão estejam separados entre si na memória principal de tal forma que há grande probabilidade dos mesmos serem mapeados nas mesmas posições do *cache,* forçando o acesso repetido à memória principal e causando perda na velocidade de processamento. Apesar da maioria dos processadores modernos usarem um cache associativo por grupos ao invés de um *cache* de mapeamento direto para minimizar o problema, quando são empregadas matrizes com um número muito grande de elementos o problema persiste. A degradação da performance é tanto maior quanto maior a potência de 2 da qual a dimensão do vetor é múltipla. Isso vale tanto para alocação dinâmica de vetores quanto alocação estática (dimensionamento fixo). Arranjos bidimensionais que são implementados como vetores unidimensionais podem apresentar o mesmo problema se o passo de acesso às posições empregado for também múltiplo de potência de 2.

Uma forma simples e prática de contornar o problema é aumentar as dimensões das matrizes em alguns elementos, tornando-as não múltiplas de potências de 2. Para alocação dinâmica de vetores, uma abordagem eficiente é sempre dimensionar os vetores com número impar de posições.

Para exemplificar essa problema, a Figura A1.2 mostra resultados de tempo de solução do código empregado anteriormente. Vários vetores eram dependentes do número de elementos da malha, e as malhas utilizadas eram prismáticas tridimensionais com dimensões (20 x 20 x 4), (40 x 40 x 8), em elementos, gerando matrizes com dimensões múltiplas de potências de 2. O sufixo*"L"* indica dimensionamento livre do vetores e matrizes, ou seja, aquele exigido pelos dados do problema, e portanto passível de ser múltiplo de 2^n, e o *"i"* indica uso somente de vetores com dimensões de tamanho ímpar. Dois processadores foram utilizados: um Intel Pentium III de 1GHz com 32 Kbytes de cache L1256 Kbytes de cache L2 (i1000S) e um

AMD Duron de 1,2 GHz com 128 Kbytes de cache L1 e 64 Kbytes de cache L2 (D1200S), ambos com memória SDR. Para cada tamanho de problema (número de equações) os vetores e matrizes dependentes do número de elementos tinham dimensões com um número de posições que era múltiplo da potência de 2 indicada sobre as curvas.

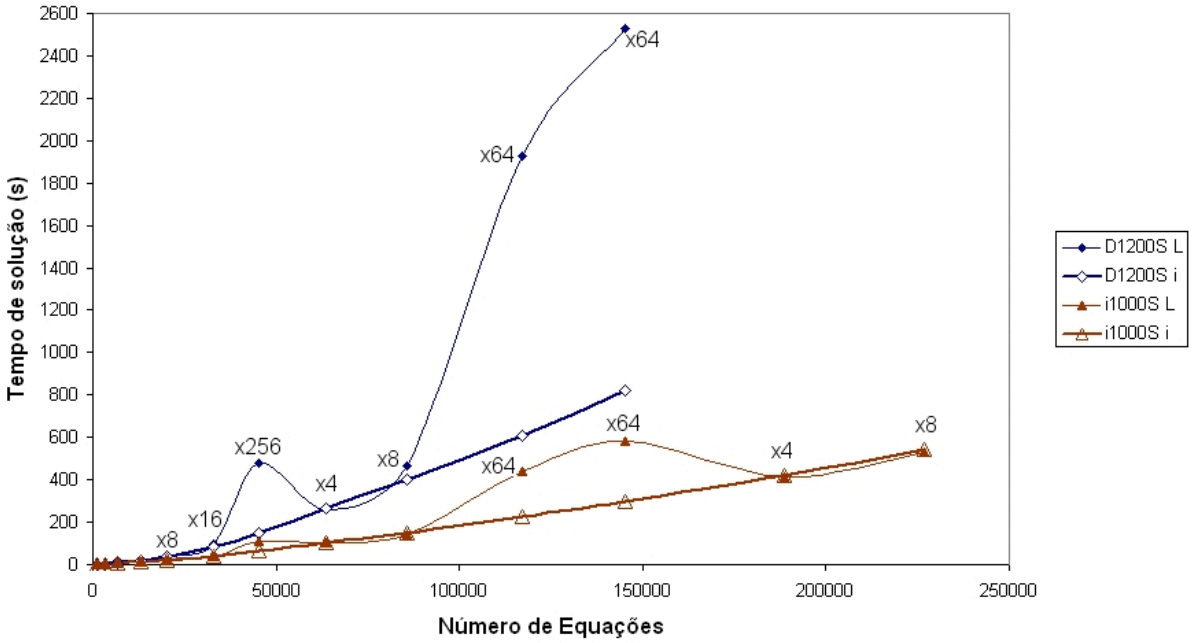


Figura A1.2: Diferenças nos tempos de processamento para um mesmo código com matrizes e vetores com dimensões com um número de posições múltiplas de potências de 2 (L) e com um número ímpar de posições (i).

Percebe-se que para os dois computadores utilizados (D1200S e i1000S), problemas com número de elementos (e, conseqüentemente, a dimensão da maior parte das matrizes e vetores) múltiplo de potências de 2 de alta ordem (64 ou 256) fazem com que o tempo de processamento seja muito grande, chegando a superar o tempo de processamento de problemas maiores mas com dimensões dos vetores múltiplos de potências de 2 de baixa ordem (O processador D1200S levou 477 segundos para resolver 45000 equações, e 260 segundos para 64000 equações). O uso de vetores cujas dimensões tem um número impar de posições resolveu completamente o problema. Percebe-se que o tipo e tamanho do cache do processador têm influência sofre o efeito adverso dos tamanhos das dimensões dos vetores. O D1200S, que tem 192 Kbytes de cache total e cache L2 de apenas 64 Kbytes, apresentou maior perda de desempenho (3 vezes) que o Pentium III, que possui cache L1 de 32 Kbytes e L2 de 256 Kbytes (2 vezes).

A1.3 ACESSANDO MATRIZES DE FORMA EFICIENTE

O FORTRAN armazena os elementos de vetores muti-dimensionais (matrizes) em espaços contíguos da memória por colunas (ao contrário do C e da intuição geral de utilizar armazenamento por linha). Isso significa que em um matriz A(10,10), os elementos são armazenados da seguinte forma:

A(1,1)...A(10,1), A(1,2)...A(10,2), ...A(1,i)...A(10,i), ...A(1,10)...A(10,10)

Assim, a forma de acesso aos elementos ao se usar estruturas de laços deve respeitar essa ordem de armazenamento, uma vez que o acesso a dados armazenados em regiões próximas da memória é feito com velocidade consideravelmente maior que a dados armazenados em posições muito distantes.

O código abaixo

```
INTEGER  X(3,5), Y(3,5), I, J
  Y = 0
  DO I=1,3                  ! laço externo I varia lentamente
    DO J=1,5                ! laço interno J varia rapidamente
      X (I,J) = Y(I,J) + 1  ! Elemento das matrizes sendo acessados
    END DO                  ! seqüencialmente em linha
  END DO
.
  END PROGRAM
```

é altamente ineficiente, pois acessa os elementos das matrizes por linha e não por coluna. O laço mais interno corresponde à dimensão mais à direita, quando, para obter o maior desempenho, os laços mais internos devem corresponder à dimensão mais à esquerda, a assim sucessivamente em um vetor multi-dimensional. O mesmo código, alterado para máximo desempenho, ficaria

```
INTEGER  X(3,5), Y(3,5), I, J
  Y = 0
  DO J=1,5                  ! laço externo J varia lentamente
    DO I=1,3                ! laço interno I varia rapidamente
      X (I,J) = Y(I,J) + 1  ! Elemento das matrizes sendo acessados
    END DO                  ! seqüencialmente em coluna
  END DO
.
  END PROGRAM
```

O cuidado na forma como os elementos de vetores multi-dimensionais devem ser acessados aplicado no código dos exemplos anteriores (o qual acessava as matrizes por linha) levou aos ganhos de desempenho mostrados na Figura A1.3, onde o sufixo 6.6A refere-se à versão do

compilador (Compaq Visual Fortran 6.6A) e *"Loop"* indica a versão do código no qual a ordem dos laços foi alterada para que os acessos às matrizes ocorressem por coluna. O Ganho no desempenho de todas máquinas foi, em média, de 170%.

Os processadores empregados foram:

T1400S: AMD Athlon Thunderbird 1.4 GHz memória SDR
T1470D: AMD Athlon Thunderbird 1.47 GHz memória DDR
T1333S: AMD Athlon Thunderbird 1.33 GHz memória SDR
T1333D: AMD Athlon Thunderbird 1.33 GHz memória DDR
i1000S: intel Pentium III 1.0 GHz memória SDR

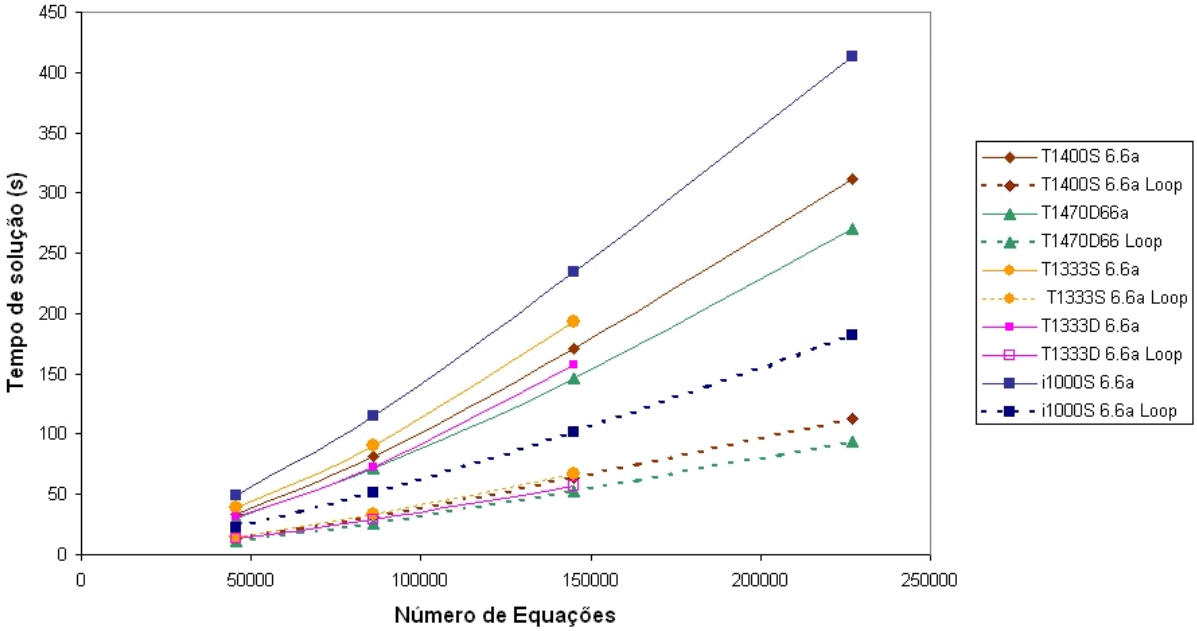


Figura A1.3: Diferenças nos tempos de processamento para um mesmo código com matrizes sendo acessadas de forma otimizada e não otimizada.

A1.4 USO CONJUNTO DAS TÉCNICAS DE OTIMIZAÇÃO

O uso conjunto das técnicas de otimização descritas anteriormente levam a um ganho significativo na velocidade de processamento de um mesmo código. O algoritmo de solução empregado não é alterado, apenas a forma de implementação.

As figuras A1.3 e A1.4 mostram, para os processadores AMD Athlon Thunderbird 1,33GHz e Intel Pentium III 1,0 GHz, respectivamente, os tempos de solução para o programa original

compilado com a versão 6.0 do compilador Compaq Visual Fortran (indicado por 6.0), com o uso de matrizes com dimensões de tamanho ímpar (6.0 ímpar), com o uso de um compilador otimizado para a arquitetura do processador, no caso a versão 6.6a do mesmo compilador (6.6a) e com o acesso às matrizes sendo feito com ordem dos laços respeitando a forma de armazenamento (6.6a Loop).

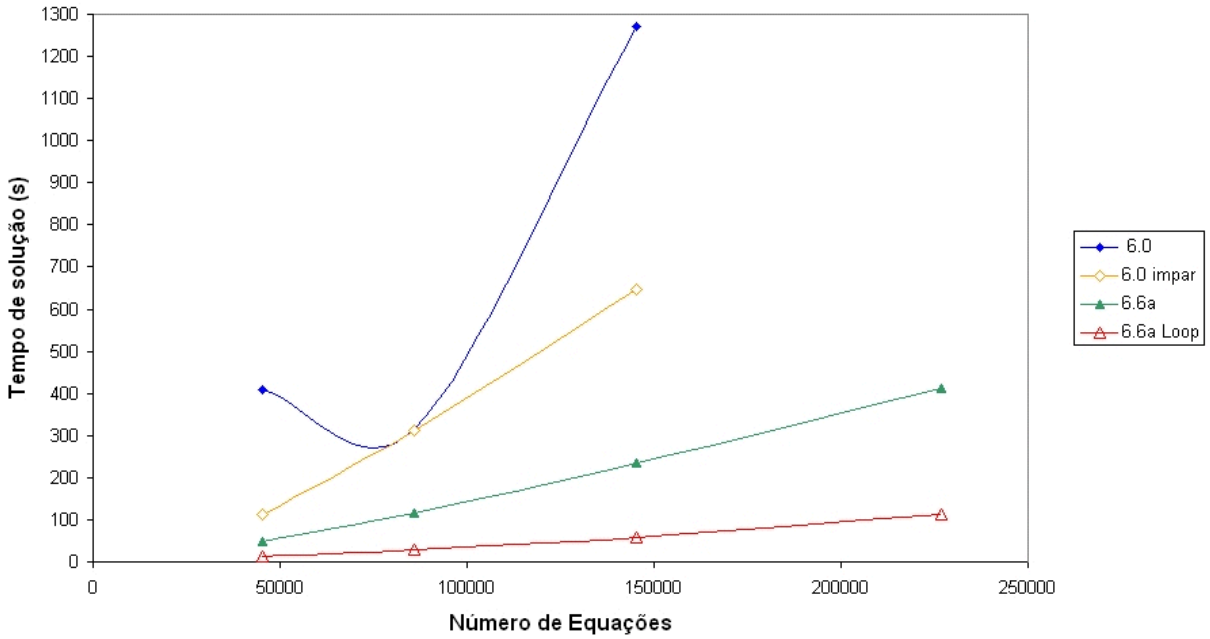


Figura A1.4: Diferenças nos tempos de processamento para um mesmo código com diferentes graus de otimização, processador AMD Athlon 1,33 GHz.

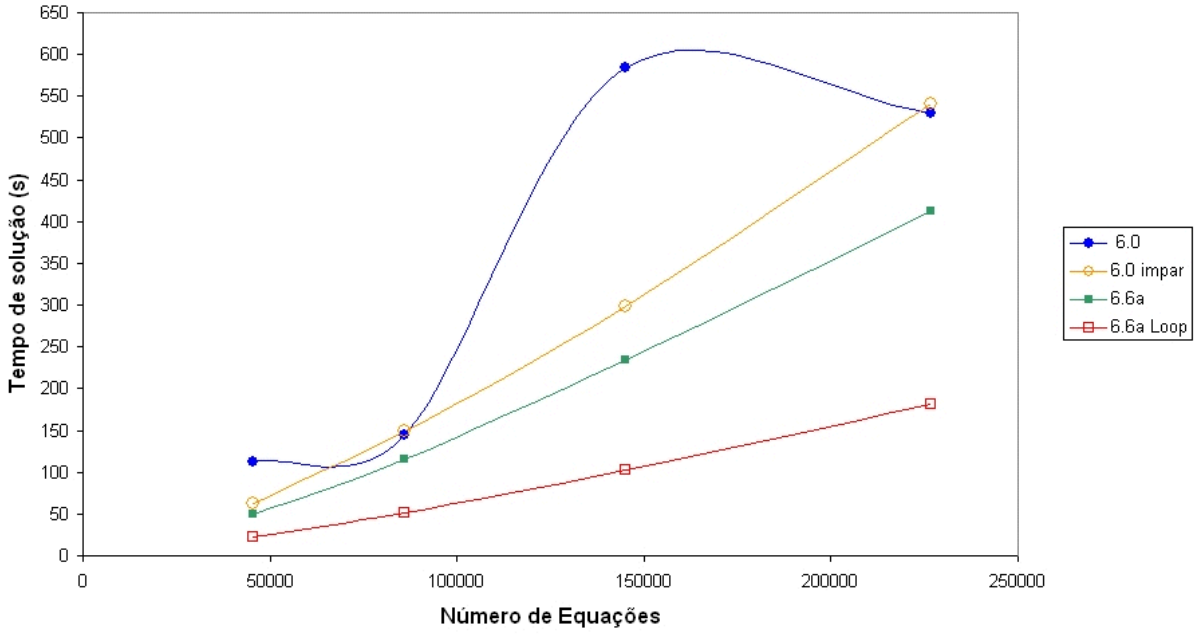


Figura A1.5: Diferenças nos tempos de processamento para um mesmo código com diferentes graus de otimização, processador Intel Pentium III 1,0 GHz.

Cada etapa de otimização inclui as anteriores, de modo que nas curvas designadas por (6.6a Loop) todas as otimizações foram empregadas. Os ganhos obtidos em velocidade de processamento foram significativos. Para o melhor dos casos, o problema com 145000 equações, o código final otimizado foi executado 22 vezes mais rápido que o código original no AMD Athlon, e 6 vezes no Intel Pentium III.

A1.5 ALOCAÇÃO DE VETORES E MATRIZES

Considerando unicamente o desempenho, a forma mais eficiente de se trabalhar em Fortran com vetores e matrizes é utilizando o dimensionamento estático, no qual o código do programa contém a informação do tamanho a ser utilizado em cada vetor. A desvantagem dessa abordagem é a necessidade da compilação de uma versão do código para cada conjunto de dados a ser processado. Em uma implementação paralela, na qual a quantidade de dados alocada a cada processador depende do número de processadores utilizado, o dimensionamento estático pode exigir um número muito grande de versões executáveis do mesmo código.

Uma forma de contornar esse problema é dimensionar os vetores com um número bastante grande de posições (*Estático Maj*), de forma a ser compatível com um maior número de conjuntos de dados. Essa abordagem deve ser utilizada com cuidado para não esgotar os recursos de memória (real e virtual) e, em problemas pequenos, pode ser altamente ineficiente quando comparado a um dimensionamento com o tamanho necessário e suficiente dos vetores.

O uso dos comandos ALLOCATE para dimensionamentos dinâmicos em Fortran gera códigos executáveis mais lentos para problemas grandes quando comparado ao dimensionamento estático. O uso da função DEALLOCATE para liberar memória de vetores que não mais são utilizados melhora o desempenho, mas não aos níveis do dimensionamento estático.

Uma solução de compromisso é o dimensionamento estático de um único vetor inteiro e um real no limite da memória real do computador e o uso de ponteiros para mapear a posição dos diversos vetores necessários dentro desses vetores estáticos (*Ponteiros*). Isso pode ser programado, criando-se variáveis inteiras que são os ponteiros para a posição de cada vetor

(com sérios prejuízos à inteligibilidade do código), ou utilizando-se o comando POINTER, que permite mapear um vetor (de nome qualquer) em outro, o que torna mais claro o código uma vez que se pode utilizar um nome diferente para cada vetor do programa, de tamanho dinâmico, utilizando os dados contidos no vetor de dimensionamento estático mapeado. Contudo, essa técnica impõe a limitação de que todos os dados devam ser organizados em arranjos unidimensionais (vetores), uma vez que os mesmos são ponteiros de vetores unidimensionais.

A figura A1.6 mostra os tempos de processamento por iteração para o programa de análise de fluídos compressíveis utilizando elementos finitos tridimensionais hexaédricos com um ponto de integração empregados nos exemplos desta tese. Percebe-se que, a medida que o tamanho do problema se torna maior, o dimensionamento estático (Estático e Estático Maj) se torna a opção mais vantajosa em desempenho, o dimensionamento dinâmico (ALLOCATE e ALLOC+DEALLOC) a mais desvantajosa, e o uso de ponteiros (Ponteiros e POINTER) um meio termo.

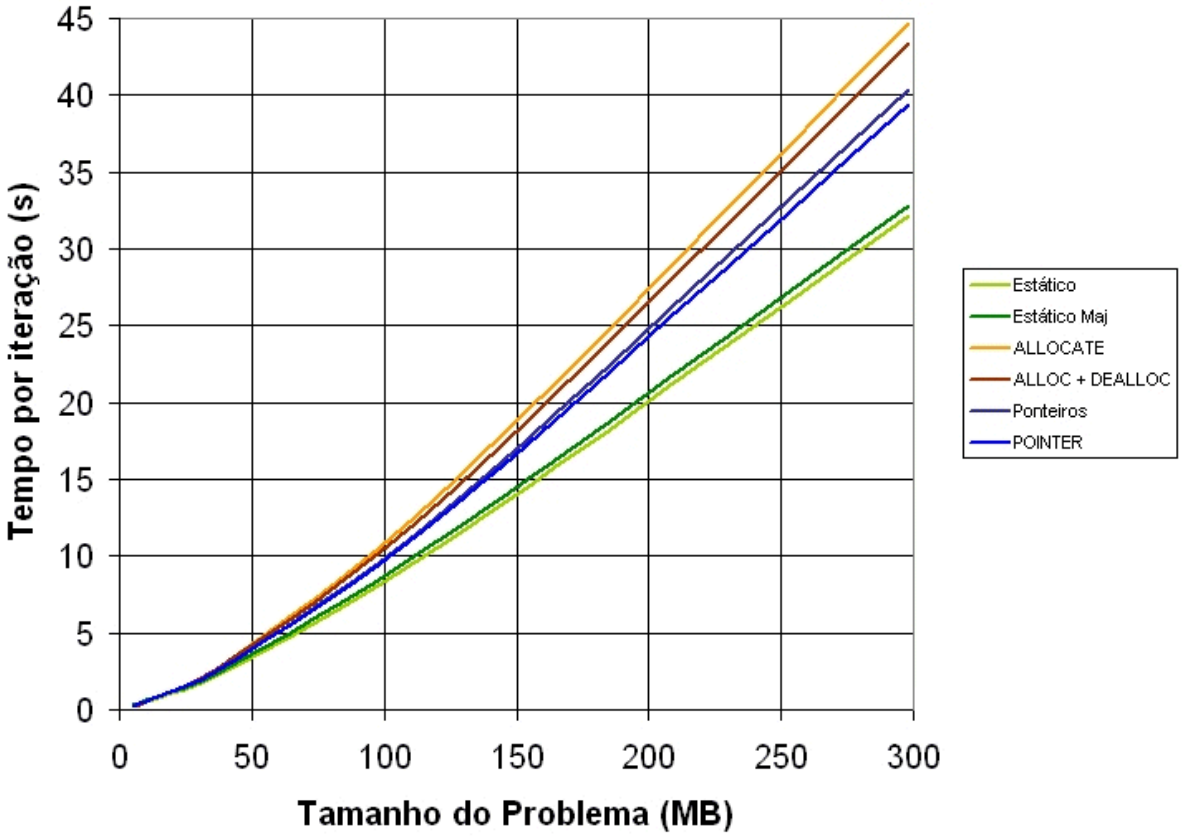


Figura A1.6: Diferenças nos tempos de processamento para um mesmo código com diferentes formas de dimensionamento de vetores e matrizes.