UNIVERSIDADE ESTADUAL DE MARINGÁ
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ROBERTINO MENDES SANTIAGO JUNIOR

Modelos de paralelização de aplicações científicas estruturadas em árvores: geração de estimativas iniciais utilizando algoritmo de subdivisão

Maringá
2012

ROBERTINO MENDES SANTIAGO JUNIOR

Modelos de paralelização de aplicações científicas estruturadas em árvores: geração de estimativas iniciais utilizando algoritmo de subdivisão

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Departamento de Informática, Centro de Tecnologia da Universidade Estadual de Maringá, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Anderson Faustino da Silva

Maringá
2012

Dados Internacionais de Catalogação-na-Publicação (CIP)
(Biblioteca Central - UEM, Maringá - PR., Brasil)

Santiago Junior, Robertino Mendes
S235m Modelos de paralelização de aplicações científicas estruturadas em árvores: geração de estimativas iniciais utilizando algoritmo de subdivisão / Robertino Mendes Santiago Junior. - Maringá, 2012.
91f. : figs., grafs., tabs.

Orientador: Prof° Dr° Anderson Faustino da Silva.
Dissertação (mestrado) - Universidade Estadual de Maringá, Departamento de Informática, Programa de Pós-Graduação em Ciência da Computação, 2012.

1. Paralelização - Modelos. 2. Plataforma MPI. 3. Clusters - Computação. 4. Estrutura de dados em árvore. 5. Equações não lineares. I. Silva, Anderson Faustino da, orient. II. Universidade Estadual de Maringá. Departamento de Informática, Programa de Pós-Graduação em Ciência da Computação. III. Título.

CCD 22. ed. 005.275

masa-000764

FOLHA DE APROVAÇÃO

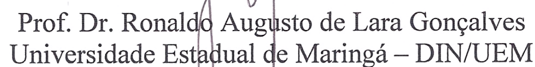
ROBERTINO MENDES SANTIAGO JUNIOR

Modelos de paralelização de aplicações científicas estruturadas em árvores: geração de estimativas iniciais utilizando algoritmo de subdivisão

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Departamento de Informática, Centro de Tecnologia da Universidade Estadual de Maringá, como requisito parcial para obtenção do título de Mestre em Ciência da Computação pela Banca Examinadora composta pelos membros:

BANCA EXAMINADORA

Prof. Dr. Anderson Faustino da Silva
Universidade Estadual de Maringá – DIN/UEM



Prof. Dr. Ronaldo Augusto de Lara Gonçalves
Universidade Estadual de Maringá – DIN/UEM

Prof. Dr. Márcio Augusto de Souza
Universidade Estadual de Ponta Grossa – DEINFO/UEPG

Aprovada em: 05 de março de 2012.
Local da defesa: Sala 101, Bloco C56, *campus* da Universidade Estadual de Maringá

DEDICATÓRIA(S)

Ao meu pai Robertino, por tornar o impossível uma realidade.
À minha mãe Marina, pelo eterno carinho e apoio incondicional.
À minha esposa Marcia, pelo amor e companheirismo.

AGRADECIMENTO(S)

Agradeço primeiramente à Deus por todas as graças concedidas à mim.

Agradeço aos meus pais Robertino Mendes Santiago e Marina Lopes das Silva Santiago por serem o alicerce da minha vida.

Agradeço à minha amada esposa Marcia Maria Carlos por todo amor, compreensão e companheirismo.

Agradeço às minhas filhas Layane Carlos Soletti e Marcella Carlos Soletti pelo carinho e afeto.

Agradeço à minha irmã Thaís Mendes Santiago e à Vanessa Souza Santos por sempre estarem com sua casa de portas abertas quando precisei.

Agradeço ao professor Dr. Ronaldo Augusto de Lara Gonçalves por toda a paciência, pelos aconselhamentos e ensinamentos e pela amizade, os quais contribuíram para minha vida profissional e pessoal.

Agradeço ao professor Dr. Anderson Faustino da Silva pelos ensinamentos, pela colaboração e contribuições à este estudo.

Agradeço aos professores do Departamento de Informática que contribuíram de forma direta ou indireta com este trabalho.

Agradeço à secretária do PCC Maria Inês Davanço por toda sua prestatividade e colaboração.

Agradeço aos colegas de mestrado pela amizade.

EPÍGRAFE

Ainda que eu ande pelo vale da sombra da morte, não temerei mal algum, porque Tu estás comigo;
[...]
Salmo 23:4

Modelos de paralelização de aplicações científicas estruturadas em árvores: geração de estimativas iniciais utilizando algoritmo de subdivisão

RESUMO

Computação paralela tem sido muito utilizada em várias áreas de conhecimento para resolver problemas de alta complexidade, os quais normalmente exigem tempo de processamento elevado quando solucionados sequencialmente. A solução iterativa de sistemas de equações não lineares é considerada um desses problemas para um grande número de aplicações, principalmente quando o sistema possui muitas dimensões, o que justifica sua paralelização. O desenvolvimento de modelos de paralelização para a solução iterativa de sistemas de equações não lineares baseadas em árvores n-árias é o foco deste estudo, sendo vinculado nesta pesquisa ao problema da geração de estimativas iniciais utilizado em simulação computacional de colunas de destilação reativa. Foram desenvolvidos 4 modelos de paralelização, um após o outro, sendo o modelo seguinte criado a partir de melhorias no modelo anterior. A implementação dos modelos paralelos foi escrita em linguagem de programação C, fazendo uso da biblioteca MPI, a partir de uma versão melhorada do algoritmo sequencial, em que foi otimizado o uso de memória, por meio de uma metodologia mais eficiente de lógica de programação. Nos experimentos realizados em um *cluster*, foram utilizados dados de casos reais conhecidos na literatura e os resultados comprovam a corretude dos modelos implementados. Todos os modelos proporcionaram vantagens sobre a aplicação original, em diferentes situações aqui analisadas, mas os resultados foram mais significativos para os modelos 03 e 04, quando executados sobre o caso real de maior tamanho, atingindo valores de *speedup* próximos ao linear, com picos da ordem de 3.94, 7.74 e 13.77 para 4, 8 e 16 nós, respectivamente.

Palavras-chave: Modelos de paralelização, Plataforma MPI, Clusters de computadores, Estrutura em árvore, Sistemas de equações não lineares.

Models of parallelization of scientific applications structured in tree: generation of initial estimates using algorithm of subdivision

ABSTRACT

Parallel computing has been widely used in various fields of knowledge to solve highly complex problems, which typically require high processing time when you solved sequentially. The iterative solution of systems of nonlinear equations is considered one of these problems for a large number of applications, particularly when the system has many dimensions, which justifies their parallelization. The development of models of parallelization for the iterative solution of systems of nonlinear equations based on n-ary trees is the focus of this study, this research is linked to the problem of generation of initial estimates used in computer simulation of reactive distillation columns. We developed four models of parallelization, one after another, with the following model created from improvements in the previous model. The implementation of the parallel model is written in the C programming language, using the MPI library from an enhanced version of the algorithm, which is optimized memory usage, by a more efficient method of programming logic. In experiments conducted in a cluster, we used data from actual cases known in literature and the results prove the correctness of the models implemented. All models have provided advantages over the original application in different situations analyzed here, but the results were more significant for the models 03 and 04, when run on the real case of largest size, reaching values close to linear speedup, with peaks of order of 3.94, 7.74 and 13.77 for 4, 8 and 16 nodes, respectively.

***Keywords*:** Models of parallelization, Plataform MPI, Clusters of computers, Tree structure, Systems of nonlinear equations.

LISTA DE FIGURAS

Figura 2.1 Categoria SISD 22
Figura 2.2 Categoria SIMD 22
Figura 2.3 Categoria MISD 22
Figura 2.4 Categoria MIMD 23
Figura 2.5 Taxonomia de arquiteturas paralelas 24
Figura 2.6 Sistema de memória compartilhada 24
Figura 2.7 Sistema de memória distribuída 25
Figura 2.8 Paralelismo funcional 25
Figura 2.9 Paralelismo de dados 26
Figura 2.10 Cluster Beowulf 27

Figura 3.1 Representação esquemática do processo convencional (Machado, 2009) 36
Figura 3.2 Representação esquemática do processo por coluna de destilação reativa (Machado, 2009) 37
Figura 3.3 Teste de cobertura (Corazza et al., 2008) 39
Figura 3.4 Representação gráfica do processo de subdivisão do retângulo inicial de duas dimensões após duas subdivisões 40

Figura 4.1 Políticas de escalonamento 42
Figura 4.2 Árvore de estimativas com elementos enumerados globalmente . . . 43
Figura 4.3 Identificação do nível de subdivisão 44
Figura 4.4 Árvore de estimativas com elementos enumerados localmente 46

Figura 5.1 Arquivo de configuração do modelo 01 50
Figura 5.2 Representação da árvore de estimativas gerada por meio do processo de subdivisão de um retângulo com 2 dimensões 56

Figura 6.1 Tempos de execução do problema com 5 dimensões utilizando a política de escalonamento de nós adjacentes (em segundos) 66
Figura 6.2 Tempos de execução do problema com 5 dimensões utilizando a política de escalonamento de nós equidistantes (em segundos) . . . 67
Figura 6.3 Tempos de execução do problema com 7 dimensões utilizando a política de escalonamento de nós adjacentes (em minutos) 68
Figura 6.4 Tempos de execução do problema com 7 dimensões utilizando a política de escalonamento de nós equidistantes (em minutos) 68

Figura 6.5 Speedups obtidos com problema de 5 dimensões utilizando a política de escalonamento de nós adjacentes 69
Figura 6.6 Speedups obtidos com problema de 5 dimensões utilizando a política de escalonamento de nós equidistantes 70
Figura 6.7 Speedups obtidos com problema de 7 dimensões utilizando a política de escalonamento de nós adjacentes 70
Figura 6.8 Speedups obtidos com problema de 7 dimensões utilizando a política de escalonamento de nós equidistantes 71
Figura 6.9 Eficiências obtidas com problema de 5 dimensões utilizando a política de escalonamento de nós adjacentes 74
Figura 6.10 Eficiências obtidos com problema de 5 dimensões utilizando a política de escalonamento de nós equidistantes 75
Figura 6.11 Eficiências obtidas com problema de 7 dimensões utilizando a política de escalonamento de nós adjacentes 75
Figura 6.12 Eficiências obtidas com problema de 7 dimensões utilizando a política de escalonamento de nós equidistantes 76

LISTA DE TABELAS

Tabela 5.1 Funções básicas do Open MPI 52
Tabela 5.2 Rotinas de envio Open MPI 52
Tabela 5.3 Funções de comunicação coletiva do Open MPI 53
Tabela 5.4 Rotinas de entrada e saída paralelas 53

Tabela 6.1 Relação de ganho de desempenho conforme evolução dos modelos utilizando a política de escalonamento de nós adjacentes para o problema com 5 dimensões 72
Tabela 6.2 Relação de ganho de desempenho conforme evolução dos modelos utilizando a política de escalonamento de nós equidistantes para o problema com 5 dimensões 72
Tabela 6.3 Relação de ganho de desempenho conforme evolução dos modelos utilizando a política de escalonamento de nós adjacentes para o problema com 7 dimensões 73
Tabela 6.4 Relação de ganho de desempenho conforme evolução dos modelos utilizando a política de escalonamento de nós equidistantes para o problema com 7 dimensões 73

Tabela A.1 Tempos de execução do problema com 5 dimensões utilizando a política de escalonamento de nós adjacentes (em segundos) 85
Tabela A.2 Tempos de execução do problema com 5 dimensões utilizando a política de escalonamento de nós equidistantes (em segundos) . . . 85
Tabela A.3 Tempos de execução do problema com 7 dimensões utilizando a política de escalonamento de nós adjacentes (em minutos) 86
Tabela A.4 Tempos de execução do problema com 7 dimensões utilizando a política de escalonamento de nós equidistantes (em minutos) 86

Tabela B.1 Speedups obtidos com problema de 5 dimensões utilizando a política de escalonamento de nós adjacentes 87
Tabela B.2 Speedups obtidos com problema de 5 dimensões utilizando a política de escalonamento de nós equidistantes 87
Tabela B.3 Speedups obtidos com problema de 7 dimensões utilizando a política de escalonamento de nós adjacentes 88
Tabela B.4 Speedups obtidos com problema de 7 dimensões utilizando a política de escalonamento de nós equidistantes 88

Tabela C.1 Eficiências obtidas com problema de 5 dimensões utilizando a política de escalonamento de nós adjacentes 89
Tabela C.2 Eficiências obtidas com problema de 5 dimensões utilizando a política de escalonamento de nós equidistantes 89
Tabela C.3 Eficiências obtidas com problema de 7 dimensões utilizando a política de escalonamento de nós adjacentes 90
Tabela C.4 Eficiências obtidas com problema de 7 dimensões utilizando a política de escalonamento de nós equidistantes 90

Tabela D.1 Relação das soluções estimadas e relações obtidas para o problema com 5 dimensões 91
Tabela D.2 Relação das soluções estimadas e relações obtidas para o problema com 7 dimensões 91

LISTA DE ALGORITMOS

1 Algoritmo da aplicação 38
2 Algoritmo principal do modelo 01 54
3 Função SubdivideRecursivo do modelo 01 55
4 Algoritmo principal do modelo 02 57
5 Função SubdivideRecursivo do modelo 02 58
6 Algoritmo principal do modelo 03 59
7 Função SubdivideRecursivo do modelo 03 61
8 Algoritmo principal do modelo 04 63
9 Função SubdivideRecursivo do modelo 04 64

LISTA DE ABREVIATURAS E SIGLAS

B&B	Branch-and-bound
B&P	Branch-and-prune
CESDIS	Center of Excellence in Space Sciences
LECAD	Laboratório Experimental de Computação de Alto Desempenho
MIMD	Multiple Instruction Multiple Data
MISD	Multiple Instruction Single Data
MPI	Message Passing Interface
NUMA	Non-Uniform Memory Access
SIMD	Single Instruction Multiple Data
SISD	Single Instruction Single Data
SMP	Symmetric Multi-Processing

SUMÁRIO

1 Introdução **17**

1.1 Contexto e Motivação 18

1.2 Objetivo e Metas 18

1.3 Organização do Trabalho 19

2 Computação Paralela **20**

2.1 Arquiteturas Paralelas 21

2.1.1 Taxonomia de Flynn 21

2.1.2 Arquitetura MIMD 23

2.2 Computação em *clusters* 24

2.2.1 Paralelismo funcional e de dados 25

2.2.2 Granulosidade 25

2.2.3 Cluster 26

2.2.4 MPI - Message Passing Interface 28

2.2.5 Balanceamento de carga 30

2.2.6 Métricas de desempenho 31

2.3 Trabalhos relacionados 31

3 Algoritmo de geração de estimativas iniciais **35**

3.1 Produção de Biodiesel 36

3.2 O algoritmo 37

4 Modelos de Paralelização **41**

4.1 Modelo 01: Paralelização com Balanceamento Estático de Carga 43

4.2 Modelo 02: Paralelização com Balanceamento Dinâmico de Carga por Ordem de Chegada 44

4.3 Modelo 03: Paralelização Síncrona com Balanceamento Dinâmico de Carga por Ordem das Maiores Cargas 45

4.4 Modelo 04: Paralelização Assíncrona com Balanceamento Dinâmico de Carga por Ordem das Maiores Cargas 46

5 Implementação dos modelos **48**

5.1 Otimizando o uso da memória 48

5.2 Plataforma operacional 50

5.2.1 Open MPI 51

5.3 Modelo 01 54
5.4 Modelo 02 56
5.5 Modelo 03 59
5.6 Modelo 04 60

6 Experimentos e resultados **65**
6.1 Tempos de execução 65
6.2 Speedup 69
6.3 Eficiência 73

7 Conclusões e trabalhos futuros **77**
7.1 Trabalhos futuros 78

Referências **80**

A Tempos de execução **85**

B Speedups **87**

C Eficiências **89**

D Soluções estimadas e obtidas **91**

Capítulo 1

Introdução

Os constantes avanços tecnológicos permitiram que nas últimas décadas o desempenho dos computadores aumentasse exponencialmente, seguindo a Lei de Moore, uma teoria evolucionista a respeito de hardware formulada por Gordon Moore na década de 60. Em sua teoria, Moore afirmou que o número de transistores em um circuito integrado iria dobrar a cada 24 meses e consequentemente o potencial de processamento.

Entretanto, essa tendência exponencial não pode continuar indefinidamente devido aos limites fundamentais impostos pela física básica (Krishnan et al., 2007). Quanto maior a velocidade do hardware, maior será sua temperatura (Polloni e Fedeli, 2003). A dissipação do calor gerado pelo hardware tem sido reconhecida como um problema em potencial que pode limitar o tratamento da informação (Krishnan et al., 2007).

Diante disso, uma alternativa utilizada pelos projetistas tem sido aumentar os número de processadores, permitindo assim que o poder de processamento aumente por meio do uso de processamento paralelo. Essa abordagem permite que sejam utilizados vários processadores, convencionais ou não, barateando assim o custo total do equipamento, sendo esses agrupados em um único computador ou dispersos em computadores individuais trabalhando em conjunto, formando assim os *clusters* de computadores (Novaes e Gregores, 2007).

Devido ao seu baixo custo em relação aos tradicionais supercomputadores, o número de *clusters* tem aumentado durante os anos, podendo ser utilizados tanto para fins científicos como comerciais. Atualmente, segundo a lista dos 500 maiores supercomputadores do mundo (TOP500.org, 2011), 82% são *clusters* de computadores.

Sobretudo, apenas aglomerar processadores não garante alcançar um processamento paralelo eficiente. É necessário a existência de uma plataforma de software para permitir a programação, execução e gerenciamento dos processos paralelos, bem como a aplicação de modelos de programação paralela e a utilização de técnicas de paralelização de algoritmos de tal forma que se aproveite todo o poder de processamento disponível.

Diversas áreas científicas podem ser beneficiadas com a utilização de *clusters*, simulando computacionalmente experimentos que, em alguns casos, podem demandar grande quantidade de poder de processamento, permitindo assim estimar resultados e comportamentos desses experimentos de forma mais rápida.

1.1 Contexto e Motivação

A solução iterativa de sistemas de equações não lineares é um dos problemas conhecidos que requer poder computacional elevado. A Engenharia Química é uma das áreas de conhecimento que possui muitos sistemas iterativos de equações não lineares. A simulação da produção de ésteres de ácidos graxos (biodiesel) em colunas de destilação reativa utiliza um modelo matemático baseado em um sistema iterativo de equações não lineares (Machado, 2009).

A convergência da solução desse sistema depende das estimativas iniciais para as suas incógnitas, o que demanda muito esforço e tempo para serem encontradas. Entretanto, esse conjunto de estimativas iniciais pode ser gerado por meio de um algoritmo de subdivisão (Corazza et al., 2008).

De acordo com o número de incógnitas (dimensões) envolvidos no sistema, a geração de estimativas iniciais pelo algoritmo de subdivisão pode ser um processo muito demorado devido ao fato de utilizar o próprio sistema iterativo durante sua execução. Além disso, o algoritmo de subdivisão gera uma estrutura de dados em forma de árvore n-ária, em que n equivale ao número de dimensões do problema envolvido. Esta árvore possui um custo de processamento muito alto para ser analisada, principalmente para a aplicação de coluna de destilação reativa aqui mencionada, a qual pode envolver centenas de incógnitas.

1.2 Objetivo e Metas

O objetivo deste trabalho foi desenvolver modelos de paralelização para a solução iterativa de sistemas de equações não lineares baseadas em árvores n-árias, os quais são neste

trabalho vinculados ao problema da geração de estimativas iniciais, mas que podem ser aproveitados em outros problemas da mesma natureza.

Para alcançar o objetivo deste estudo, as seguintes metas foram realizadas:

- Análise do algoritmo sequencial original, para a identificação das suas principais características;
- Otimização da versão sequencial do algoritmo, por meio do uso de uma metodologia mais eficiente de lógica de programação;
- Projeto e implementação de 4 modelos de paralelização, sendo cada modelo desenvolvido por meio de melhorias no modelo anterior;
- Experimentação e análise dos modelos de paralelização, identificando as vantagens e desvantagens de cada um.

1.3 Organização do Trabalho

No Capítulo 2 é apresentado uma revisão bibliográfica sobre assuntos relacionados à arquiteturas paralelas, detalhando as classificações existentes, à computação em *cluster* e são descritos alguns trabalhos relacionados ao presente estudo. No capítulo 3 é descrito o funcionamento do algoritmo de geração de estimativas.

Os quatro modelos paralelos elaborados são apresentados no Capítulo 4. No Capítulo 5 é abordado a implementação de cada modelo proposto, bem como as otimizações feitas no algoritmo original e o ambiente de execução dos experimentos. Os resultados dos experimentos são apresentados no Capítulo 6. As conclusões e trabalhos futuros são descritos no Capítulo 7.

No Apêndice A são apresentadas as tabelas contendo os valores do tempos de execução dos modelos desenvolvidos. Os valores dos *speedups* são apresentados em tabelas presentes no Apêndice B. No Apêndice C são apresentadas as tabelas contendo os valores de eficiência dos modelos. Os resultados estimados e obtidos na execução da aplicação são apresentados no Apêndice D.

Capítulo 2

Computação Paralela

A arquitetura básica para os computadores modernos foi concebida nos anos 50 pelo matemático John von Neumann, denominada de Arquitetura de von Neumann, que ainda serve de base para os computadores atuais (Dale e Lewis, 2010). Sua arquitetura era formada basicamente por cinco partes fundamentais: unidade de controle, unidade aritmética, memória e mecanismos de entrada/saída.

Tradicionalmente, os computadores são vistos como máquinas sequenciais que executam instruções específicas contidas em algoritmos, de forma sequencial, uma de cada vez (Stallings, 2009). Cada instrução executa uma sequência de operações. Embora a velocidade com que computadores executam essas instruções tenha sido aumentada constantemente nos últimos anos, existem barreiras físicas que podem impedir uma evolução ainda maior, como a tecnologia de concepção, limitações e conflito de recursos, o aquecimento do circuito e a organização arquitetural, entre outras (Tanenbaum, 2007).

Em meio a essas preocupações, tem-se buscado formas de aumentar o desempenho dos computadores sem esbarrar nessas barreiras. Uma solução é organizar os computadores de forma que, múltiplas unidades de processamento possam cooperar para que o trabalho seja feito em paralelo.

Uma solução é utilizar um aglomerado de computadores interligados por meio de uma rede de interconexão, os quais funcionam de forma cooperativa, para que o trabalho seja feito em paralelo, formando assim um cluster de computadores. A possibilidade de expandir o poder computacional utilizando um aglomerado de computadores fez com que, nos últimos anos, o uso de *clusters* de computadores aumentasse, sendo utilizados

para fins científicos e comerciais. Estudos relacionados à arquiteturas paralelas e *clusters* possibilitam o desenvolvimento de novas tecnologias e técnicas de processamento paralelo.

2.1 Arquiteturas Paralelas

Durante muitos anos os computadores paralelos têm sido utilizados para diferentes propósitos, e diferentes arquiteturas tem sido propostas e utilizadas (Rauber e Rünger, 2010). De maneira geral, um computador paralelo pode ser definido como um conjunto de elementos processadores cooperando para a resolução de problemas de forma rápida. Nessa seção é aborado assuntos relacionados à arquiteturas paralelas.

2.1.1 Taxonomia de Flynn

A taxonomia de arquitetura de computadores mais popular foi definida em 1966 pelo pesquisador Michael J. Flynn, denominada taxonomia de Flynn (Flynn, 1966). Flynn classificou os computadores de acordo com o número de fluxos de instruções e o número de fluxos de dados. O fluxo de instruções pode ser definido como a sequência de instruções que é executada por uma unidade de processamento e o fluxo de dados pode ser definido como sendo a sequência de trocas de dados entre a memória e a unidade de processamento (El-Rewini e Abd-El-Barr, 2005).

Flynn definiu em sua taxonomia quatro categorias distintas de arquitetura de computadores, sendo:

- SISD - *Single Instruction Single Data Stream*
- SIMD - *Single Instruction Multiple Data Stream*
- MISD - *Multiple Instruction Single Data Stream*
- MIMD - *Multiple Instruction Multiple Data Stream*

Na categoria SISD (Figura 2.1), um único elemento processador executa um fluxo de instruções (FI) que opera sobre um único fluxo de dados (FD). A máquina de von Neumann é considerada um exemplo clássico desta categoria, além dos computadores atuais que possuem um único processador de núcleo único (*single-core*). Na classe SIMD (Figura 2.2), a unidade de controle transmite simultaneamente o mesmo fluxo de instruções para todos os elementos processadores, o qual é executado sobre fluxos de dados diferentes. Neste modelo, cada unidade de processamento trabalha em dados de sua própria memória.

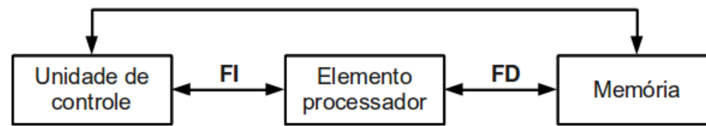


Figura 2.1: Categoria SISD

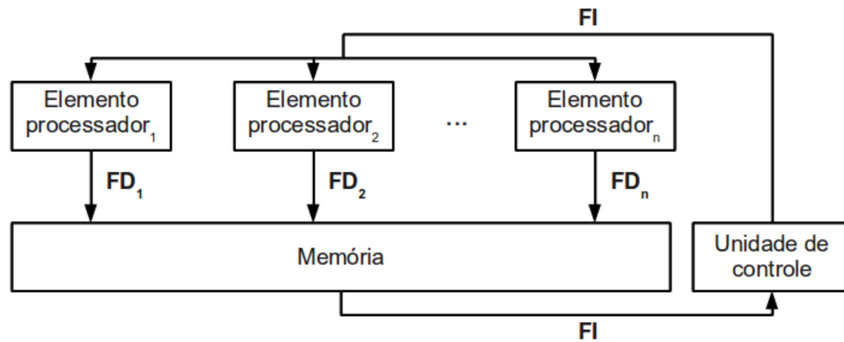


Figura 2.2: Categoria SIMD

Na classe MISD (Figura 2.3) o mesmo fluxo de dados é transmitido para um conjunto de processadores, em que cada processador executa fluxos de instruções diferentes. Entretanto, essa arquitetura não é comercialmente implementada (Stallings, 2009). A classe de arquitetura paralela mais familiar é a MIMD (Figura 2.4) e possibilita a forma mais básica de processamento paralelo (Flynn e Rudd, 1996). Consiste de múltiplos processadores interconectados. Ao contrário da classe SIMD, cada elemento processador executa, de forma independente, diferentes fluxos de instruções sobre diferentes fluxos de dados.

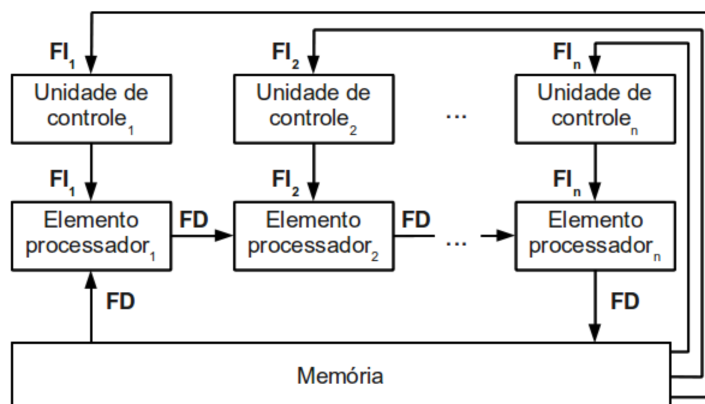


Figura 2.3: Categoria MISD

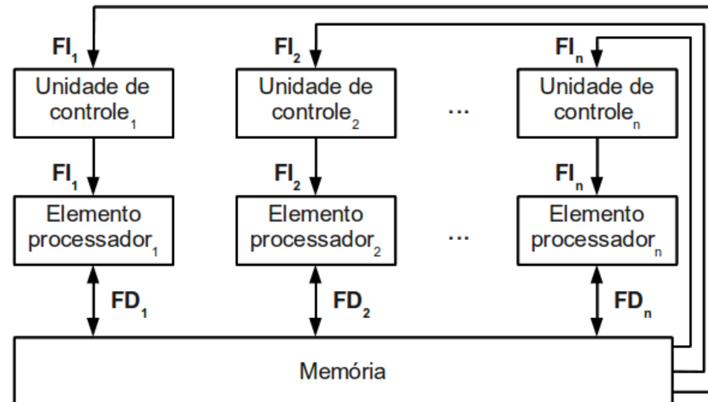


Figura 2.4: Categoria MIMD

2.1.2 Arquitetura MIMD

Apesar da taxonomia de Flynn ser bem aceita, determinadas classes de computadores paralelos não são enquadradas nesta classificação (Duncan, 1990). Conforme Fountain (2006), várias taxonomias foram elaboradas a fim de englobar o maior número possível de arquiteturas: (Shore, 1973), (Hockney e Jesshope, 1981), (Fountain e Shute, 1990) e (Duncan, 1990).

Segundo Stallings (2009), a categoria MIMD pode ser divida de acordo com a comunicação entre os processadores, como pode ser observado na Figura 2.5. Stallings subdividiu a categoria MIMD em Memória Compartilhada e Memória Distribuída. Na categoria de memória compartilhada, a comunicação entre os processadores ocorre por meio de um mesmo espaço de endereçamento de memória acessível a todos os processadores (Figura 2.6). Nesta categoria estão enquadradas as arquiteturas: SMP (Multiprocessador simétrico) e NUMA (Acesso não uniforme à memória).

Segundo Gruber e Keller (2010), na arquitetura SMP, todos os processadores visualizam a memória de uma forma totalmente simétrica, tendo aproximadamente o mesmo tempo de acesso à memória para todos os processadores. Na arquitetura NUMA, a memória está distribuída entre os processadores. Entretanto, o sistema cria uma abstração de um único espaço de endereçamento e caso a memória de outro processador seja acessada, a latência aumenta. Desta forma, não é possível prever com precisão o tempo de computação.

Na categoria de memória distribuída, cada processador possui sua própria memória, sendo essa acessível somente a ele mesmo. A comunicação entre dois processadores ocorre

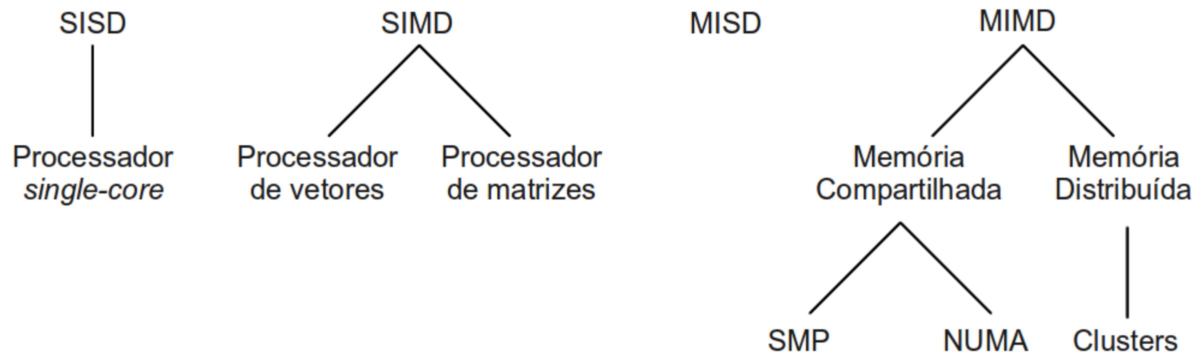


Figura 2.5: Taxonomia de arquiteturas paralelas

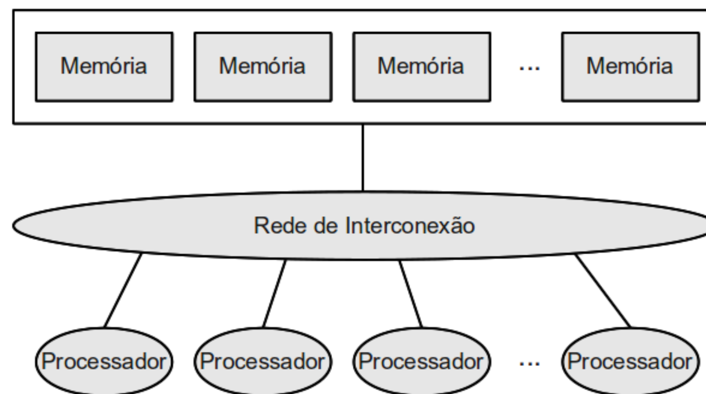


Figura 2.6: Sistema de memória compartilhada

por meio da rede de interconexão (Figura 2.7). No contexto de memória distribuída, aparecem os *clusters*, muitas vezes contento elementos SMP.

2.2 Computação em *clusters*

Diversas pesquisas tem sido realizadas sobre linguagens e ambientes de programação paralelas com o objetivo de facilitar a programação paralela. Atualmente existem muitas técnicas e ambientes já disponíveis (Rauber e Rünger, 2010). Entretanto, indiferentemente do ambiente o sistema específico a ser utilizado, há vários aspectos que devem ser considerados no desenvolvimento de programas paralelos, os quais são discutidos nessa seção.

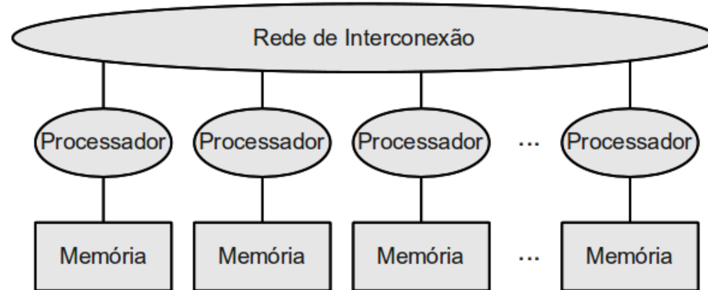


Figura 2.7: Sistema de memória distribuída

2.2.1 Paralelismo funcional e de dados

Existem essencialmente dois tipos de paralelismo (Yero, 2003): funcional e de dados. O paralelismo funcional ocorre quando uma aplicação é formada por várias seções distintas de códigos paralelizáveis. Neste tipo, os processadores executam seções distintas do código, atuando sobre o mesmo conjunto de dados ou sobre conjuntos distintos, conforme pode ser observado no exemplo da Figura 2.8 (Gomes, 2009).

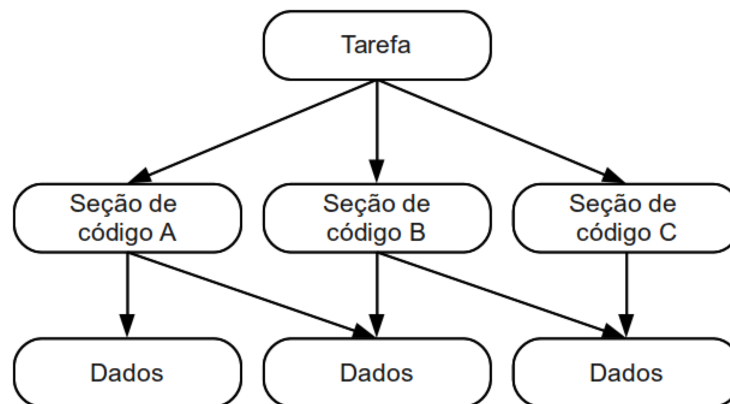


Figura 2.8: Paralelismo funcional

Segundo Foster (1995), o paralelismo de dados pode ser definido como sendo a execução de uma mesma seção de código sobre partes distintas de um estrutura de dados composta sobre processadores distintos (Figura 2.9).

2.2.2 Granulosidade

O termo mais frequente que define o nível conceitual de paralelismo é a granulosidade, também conhecido como granularidade. Esse termo está relacionado as quantidades de

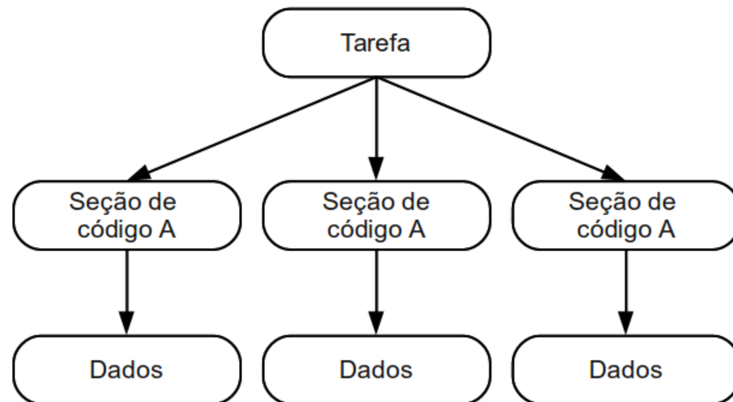


Figura 2.9: Paralelismo de dados

processamento e de comunicação das partes de uma aplicação quando executadas em paralelo. Segundo Grama et al. (2003), o número e o tamanho das tarefas em que um problema é dividido determinam a granulosidade do paralelismo. Normalmente, quanto maior é a taxa de comunicação ou sincronização entre os componentes por porção de código, menor é a granulosidade.

Assim, intuitivamente, pode-se dizer que granulosidade é uma característica relativa e os tipos básicos são (El-Rewini e Abd-El-Barr, 2005):

- **Granulosidade grossa:** cada componente paralelo pode executar uma grande quantidade de instruções até que surja a necessidade de comunicação ou sincronização.
- **Granulosidade fina:** cada componente paralelo pode executar uma pequena quantidade de instruções até que surja a necessidade de comunicação ou sincronização.
- **Granulosidade média:** cada componente paralelo pode executar uma quantidade mediana de instruções até que surja a necessidade de comunicação ou sincronização.

2.2.3 Cluster

Um *cluster* pode ser definido como um conjunto de computadores, com cada computador contendo um ou mais elementos de processamento, conhecidos como nós, interligados por meio de uma rede de interconexão, possuindo software de suporte acessível ao usuário para organizar e controlar as tarefas de computação concorrente que colaboram no processamento de uma aplicação (Sterling, 2001).

Em 1994, os pesquisadores Thomas Sterling e Don Becker participavam de um projeto do CESDIS (*Center of Excellence in Space Sciences*) e construíram um *cluster* com 16 computadores equipados com processadores Intel DX4 conectados por meio de uma rede *Ethernet*, dando origem então à classe de *clusters* hoje conhecida como *Beowulf* (Figura 2.10). Em um *cluster Beowulf* o único computador que possui equipamentos como mouse, teclado, monitor é o nó servidor (mestre). Todo o acesso aos nós clientes (escravos) é realizado via conexão remota por meio do nó servidor, sendo este último o único interligado à rede externa (Morrisson, 2003)(El-Rewini e Abd-El-Barr, 2005).

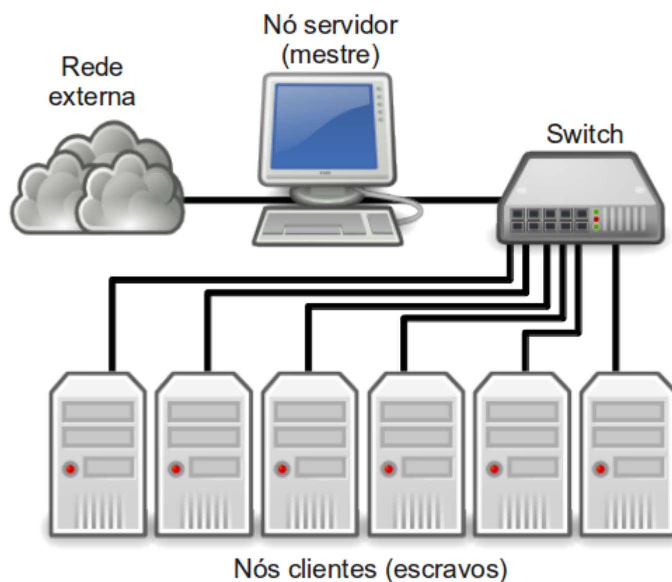


Figura 2.10: Cluster Beowulf

Devido à sua crescente popularidade, *clusters* de computadores têm se mostrado uma excelente alternativa de baixo custo para os tradicionais supercomputadores em laboratórios científicos, universidades e centros de supercomputação, além de serem empregados para fins comerciais. Atualmente, conforme a lista dos 500 maiores supercomputadores do mundo (TOP500.org, 2011), 82% são *clusters* de computadores.

Uma das limitações dos *clusters* é o hardware de rede. A interface de rede utilizada dependerá das necessidades de largura de banda, requisitos de latência, limitações de distância e restrições orçamentárias. A *Fast-Ethernet* é a rede mais comumente utilizada e transmite dados a uma velocidade de 10/100 Mbits por segundo, porém, alguns grupos têm utilizado tecnologias mais caras, como Gigabit Ethernet ou Myrinet, capazes de superar as limitações de velocidade da *Fast-Ethernet* (Freisleben e Kielmann, 1995).

Segundo El-Rewini e Abd-El-Barr (2005), um *cluster* é dito homogêneo quando todos os nós possuem a mesma arquitetura e executam o mesmo sistema operacional. Quando

seus nós possuem arquiteturas diferentes ou executam sistemas operacionais diferentes, é dito heterogêneo. Essa característica deve ser levada em consideração, pois possui implicações importantes no balanceamento de carga das aplicações. Comumente, a quantidade de carga recebida por um nó do *cluster* é proporcional à sua capacidade de processamento, mas existem diferentes métodos de distribuição de carga de trabalho, os quais fazem uso particular das especificidades de cada nó de processamento.

2.2.4 MPI - Message Passing Interface

Em um *cluster* de computadores a memória está distribuída fisicamente em cada computador do *cluster* e o acesso aos dados da memória de um nó por outro nó é feito por meio da rede de comunicação, seja de forma explícita ou implícita. Para realizar a comunicação de forma explícita, a passagem de mensagem, também conhecida como troca de mensagens, é comumente utilizada.

Nesse contexto, uma mensagem pode ser definida como uma unidade lógica para a comunicação entre os processos, sendo considerada como uma coleção de informações relacionadas que são transmitidas como uma única entidade (El-Rewini e Abd-El-Barr, 2005).

MPI (*Message Passing Interface*) é um padrão que especifica a semântica e a sintaxe de funções de comunicação, definido pelo Fórum MPI, com o propósito de atender às necessidades de aplicações paralelas típicas (Navaux et al., 2001). O Fórum MPI (MPI, 2011) é um grupo aberto com mais de 40 organizações participantes, entre elas, fabricantes, pesquisadores, desenvolvedores e usuários.

É importante ressaltar que MPI é uma especificação destinada a desenvolvedores e usuários de bibliotecas de passagem de mensagem. MPI não é uma biblioteca, mas define como uma biblioteca MPI deve ser. Em MPI, todo o paralelismo é explícito, sendo o desenvolvedor responsável em especificar como e quando o paralelismo será utilizado. As primitivas de comunicação do MPI podem ser classificadas de acordo com a quantidade de processos envolvidos na comunicação e o modo como as mensagens são transmitidas entre os processos.

De acordo com a quantidade de processos envolvidos na comunicação, as operações de comunicação podem ser classificados em dois grandes grupos: ponto-a-ponto e coletiva (Rauber e Rünger, 2010). A comunicação ponto-a-ponto é realizada entre pares de processos, um emissor e um receptor, por meio do uso de primitivas do tipo *send* e *receive*. A comunicação coletiva, também conhecida como global, ocorre entre n-processos. A

operação mais comumente utilizada é a de *broadcast*, em que um processo envia a mesma mensagem para todos os outros processos.

Conforme Sterling (1999) o padrão MPI possui quatro diferentes modos de comunicação ponto-a-ponto:

- **Padrão (*Standard*):** a decisão da utilização ou não de um *buffer* interno fica a critério do MPI. Se for utilizado, o processo emissor pode continuar seu processamento após os dados serem armazenados no *buffer*. Caso contrário, o processo emissor fica bloqueado enquanto o processo receptor correspondente não sinalize o recebimento da mensagem.
- ***Buffered*:** o MPI sempre irá utilizar um *buffer* para o envio das mensagens, sendo de responsabilidade do desenvolvedor a alocação de memória. O processo emissor fica bloqueado até que os dados sejam transferidos para o *buffer*.
- ***Ready*:** requer que o processo receptor esteja aguardando quando o processo emissor enviar a mensagem.
- **Síncrono (*Synchronous*):** o processo emissor permanece bloqueado enquanto o processo receptor não for iniciado.

As rotinas de comunicação coletiva são utilizadas quando é necessário realizar operações coordenadas entre vários processos. Por padrão, o MPI seleciona todos os processos existentes. Entretanto, caso haja necessidade, o desenvolvedor pode especificar um subgrupo de processos. As operações de comunicação coletivas são (Ignácio e Ferreira Filho, 2002):

- **Barreira**: utilizada para sincronizar todos os processos.
- **Difusão**: um processo envia dados para os demais processos.
- **Junção**: todos os processos enviam dados para um único processo.
- **Espalhamento**: um processo distribui um conjunto de dados entre todos os processos.
- **Redução**: realiza operações como soma, multiplicação, etc., de dados distribuídos.

Assim, de uma forma geral, a comunicação pode ser bloqueante ou não-bloqueante (Rauber e Rünger, 2010). Na comunicação bloqueante, o processo irá continuar sua

execução somente quando todos os recursos, tais como *buffers*, especificados na chamada puderem ser reutilizados, enquanto que, na comunicação não-bloqueante, o processo continua sua execução após a chamada da ação solicitada. Entretanto, não é possível definir o momento em que a comunicação será realizada.

A versão mais recente do MPI é a 2.2, lançada em 4 de setembro de 2009 e possui suporte oficial para as linguagens de programação Fortran e C/C++. Entretanto, já está em votação a proposta para a versão 3.0, que até o término deste estudo, não havia sido finalizada. Atualmente, o padrão MPI é implementado por várias bibliotecas, dentre elas, podemos citar o LAM/MPI, MPICH, GridMPI e Open MPI.

2.2.5 Balanceamento de carga

Segundo Barney (2009), balanceamento de carga refere-se à distribuição das tarefas entre os processadores de modo que todos sejam mantidos ocupados durante a execução de uma aplicação paralela. O balanceamento de carga é necessário para que aplicações paralelas possam obter melhor desempenho. Entretanto, a utilização de barreiras de sincronização, faz com que o desempenho global nos trechos sincronizados seja determinado pela tarefa mais lenta.

O balanceamento de carga envolve atribuir as tarefas para cada processador, de forma dependente da sua carga atual, e essa atribuição pode ser feita de forma estática, em que a divisão e a distribuição das tarefas são realizadas no início da aplicação, e dinâmica, sendo realizado quando existir a presença de nós saturados, quando o número de tarefas criadas no início da execução a serem distribuídas não é conhecido ou quando o número de tarefas a serem executadas ultrapassa a quantidade de processadores disponíveis no sistema (Palin, 2007).

Conforme Willebeek-LeMair e Reeves (1993), o balanceamento de carga dinâmico é essencial para o uso eficiente de sistemas altamente paralelos na resolução de problemas não-uniformes com as estimativas imprevisíveis de carga.

As estratégias de balanceamento, a fim de reduzir o número de transferências de carga entre processadores, devem considerar (Silva, 2006):

- a estrutura hierárquica de comunicação do ambiente com a finalidade de reduzir a sobrecarga (*overhead*) de comunicação entre os processos;

- as informações sobre o índice de carga externa e poder computacional dos processadores, sendo possível definir de forma dinâmica o desempenho dos processadores participantes do ambiente.

2.2.6 Métricas de desempenho

Métricas são comumente utilizadas para mensurar o desempenho da paralelização aplicada em relação às diferentes metodologias utilizadas e à versão sequencial. Segundo Eager et al. (1989), duas medidas de desempenho para a avaliação de um sistema paralelo merecem um foco especial, o *speedup* e a eficiência.

Speedup pode ser definido como a relação entre o tempo gasto na execução de uma determinada tarefa utilizando um único processador e o tempo gasto com N processadores, indicando assim quantas vezes o programa paralelo é mais rápido que a versão sequencial (Cortes e Mendez, 1999).

O *speedup* é calculado utilizando a fórmula 2.1:

$$S = \frac{T_s}{T_p} \tag{2.1}$$

Onde T_s é o tempo de execução do programa na versão sequencial e T_p o tempo na versão paralela.

Quando o *speedup* obtido é igual ao número de processadores, diz-se que o *speedup* é linear, porém, isso dificilmente ocorre devido ao fato da existência de alguns fatores restritivos, como a sobrecarga imposta pela infra-estrutura de paralelização, de comunicação e de criação de tarefas envolvidas no processamento. Quando o *speedup* obtido é maior que o número de processadores, diz-se que o *speedup* é superlinear.

O cálculo de eficiência mostra se os recursos foram bem aproveitados. A eficiência é calculada utilizando a fórmula 2.2:

$$E = \frac{S}{P} \tag{2.2}$$

Onde S representa o *speedup* e P o número de nós utilizados no processamento.

2.3 Trabalhos relacionados

Muitas pesquisas têm sido feitas sobre o uso de processamento paralelo para produção de resultados de forma rápida, eficiente e econômica em diferentes áreas científicas. Na pesquisa realizada por Mullenix e Povitsky (2009), desenvolvida no *Ohio Supercomputer Center*, foi utilizado um *cluster* para simular o processo de ablação. A ablação é um processo de rápida remoção de material de uma superfície sólida por meio de reações químicas, sublimação e de outros processos erosivos. A paralelização da aplicação permitiu

reduzir o tempo da simulação utilizada no estudo do problema, que necessitava de 59.95 horas, para apenas 3.2 horas quando executado em paralelo utilizando 32 processadores.

O trabalho realizado por Gomes (2009) discute questões relacionadas à paralelização de um algoritmo utilizado na solução de problemas científicos da área de Engenharia Química, o qual simula o processo de adsorção de moléculas em superfícies heterogêneas bidimensionais usando um *cluster* de computadores. Esse algoritmo utiliza o método de Monte Carlo para calcular o estado de energia do sistema após movimentos das moléculas e os resultados são utilizados para a obtenção de gráficos de isotérmicas, comparando-os com os experimentos reais e dados conhecidos. Segundo o autor, a execução sequencial do referido algoritmo consome muitas horas de processamento. Nesse trabalho foram obtidos redução do tempo de processamento em até 83.17%.

O estudo produzido por Costa (2010) aborda a análise e validade de metodologias numéricas de paralelização previamente implementadas na plataforma numérica COMMA3D, que consiste em um programa desenvolvido em Fortran 90, que constitui a base de uma plataforma paralelizada de modelação numérica do comportamento termomecânico de materiais compósitos[1] .

A utilização da plataforma COMMA3D permite realizar a modelação numérica por elementos finitos do comportamento elástico de componentes estruturais como vigas encastradas e placas de alumínio entalhadas, recorrendo a técnicas de cálculo paralelo e distribuído. Segundo o autor, para este estudo foi implementado o método do gradiente conjugado precondicionado, utilizando quatro precondicionadores distintos, sendo experimentado em um cluster Beowulf com 12 processadores. Nos experimentos realizados foram obtidos valores de *speedup* e eficiência relevantes à validação do modelo proposto. Para dois modelos de precondicionadores os valores de *speedup* permaneceram próximos ao linear utilizando uma malha de elementos finitos com 23 mil elementos. Utilizando uma malha com 49.951 elementos, foi possível atingir *speedup* superlinear.

No estudo realizado por Pinto (2011), foi desenvolvida uma estratégia de paralelização aplicada ao problema de planejamento da operação de sistemas hidrotérmicos. Basicamente, o objetivo do planejamento da operação energética de um sistema hidrotérmico é estabelecer, para cada etapa do período de planejamento, as metas de geração para cada usina que atenda a demanda e reduzir o valor esperado do curso de operação ao longo do período. Conforme o autor, a determinação das metas de geração das usinas que minimizam o custo total de operação no problema de planejamento do Sistema Elétrico

[1] Os materiais compósitos são formados a partir de dois ou mais materiais distintos, em que cada um contribui para as propriedades do material formado (Shigley e Mischke, 2005)

Brasileiro é um processo que requer processamento computacional intensivo, tendo em vista a quantidade de usinas térmicas e hidráulicas interligadas.

Um *cluster* de computadores foi utilizado para avaliar o desempenho da estratégia de paralelização. Utilizou-se um caso que possui características baseadas em períodos mensais, séries hidrológicas que definem os valores das afluências no período, chamadas de cenários de afluências, e ciclos, que estabelecem como o problema é resolvido: do início para o final (*forward*) ou do final para o início (*backward*) do período. Por meio deste trabalho, foi possível reduzir o tempo de processamento, que necessitava de 15 horas quando executado sequencialmente, para apenas 17 minutos utilizando 128 processadores, atingindo desta forma a eficiência de 41.10%. Entretanto, apesar da eficiência relativamente baixa para um número elevado de processadores, utilizando 8 processadores, o modelo atingiu 94,85% e 88.96% utilizando 16 processadores.

Estudos sobre o benefício da paralelização de problemas que utilizam a estrutura de árvore também têm sido feitos. O trabalho de Oliveira (2010) descreve estratégias para a paralelização de algoritmos do tipo *branch-and-prune* (B&P) e *branch-and-bound* (B&B) em ambientes distribuídos compartilhados e dinâmicos. Segundo a autora, *branch-and-pune* são algoritmos utilizados para resolver problemas de satisfação de restrição e tais problemas utilizam algum tipo de busca exaustiva, baseada em ramificação (*branch*) e poda de ramos inviáveis (*prune*), para encontrar soluções viáveis, podendo fazer o uso de um conjunto de restrições para selecionar as soluções corretas.

Tais algoritmos buscam pela solução ótima do problema percorrendo uma estrutura de árvore, em que cada nó indica uma solução parcial do problema da aplicação e cada ramo da árvore representa um conjunto de possíveis soluções viáveis. Basicamente a diferença entre os dois algoritmos está no objetivo da busca e na forma como a poda de ramos é feita. Os algoritmos B&B avaliam somente soluções viáveis dentro de um limite, como o custo da melhor solução já encontrada, enquanto os algoritmos B&P avaliam todas as soluções viáveis considerando as restrições do problema. Os resultados com o intuito de medir os *speedups* em relação a algoritmos sequenciais eficientes mostraram que, dependendo dos valores de níveis de corte realizado na árvore, pode-se obter uma boa paralelização, obtendo valores próximos ao linear.

Na pesquisa realizada por Modenesi (2008) foi proposto o uso de paralelismo para tratar o problema da análise de agrupamentos no processamento de grandes volumes de dados, utilizando as estratégias de paralelismo pela divisão do conjunto de dados e

pela divisão do conjunto de partições[2]. Segundo o autor, a análise de agrupamentos é uma abordagem de mineração de dados, e pode ser definida como a classificação não supervisionada dos dados em grupos, em que requer alto poder computacional.

O agrupamento hierárquico gera uma estrutura em forma de árvore onde os pontos são aninhados em grupos de acordo com alguma métrica. Em testes realizados em um *cluster* de computadores, os melhores *speedups* foram obtidos no processamento do arquivo com maior número de variáveis, alcançando valores próximos ao linear.

[2]O conceito de partições, também conhecidas como nebulosas (*fuzzy*), permite que os mesmos dados possam pertencer a mais de um agrupamento, mas com diferentes valores de pertinência (Modenesi, 2008).

Capítulo

3

Algoritmo de geração de estimativas iniciais

Nas diversas áreas científicas existem experimentos que são muito difíceis de serem realizados, muitos dos quais são caros e até perigosos, requerendo equipamentos especiais e procedimentos rígidos para evitar acidentes. Com o uso de simulação computacional, é possível estimar resultados e comportamentos desses experimentos evitando desta forma danos e gastos desnecessários.

Nesse contexto, um dos problemas que demanda grande quantidade de poder computacional é a solução iterativa de sistemas de equações não lineares. O nível de exigência do poder de processamento aumenta em função do tamanho dos sistemas envolvidos. Uma das áreas de conhecimento que possui muitos sistemas iterativos de equações não lineares é a Engenharia Química, em que podemos citar a simulação da produção de ésteres de ácidos graxos (biodiesel) em colunas de destilação reativa, a qual utiliza um modelo matemático baseado em um sistema iterativo de equações não lineares (Machado, 2009). Sabe-se que a convergência da solução desse sistema depende das estimativas iniciais para as suas incógnitas, o que demanda muito esforço e tempo para serem encontradas. Entretanto, esse conjunto de estimativas iniciais pode ser gerado por meio de um algoritmo de subdivisão (Corazza et al., 2008).

Porém, a geração de estimativas iniciais pelo algoritmo de subdivisão também é um processo muito demorado quando envolve sistemas com um elevado número de incógnitas

(dimensões), porque utiliza o próprio sistema iterativo durante sua execução, justificando assim a sua paralelização, que é alvo no presente trabalho.

3.1 Produção de Biodiesel

O governo brasileiro, por meio da Lei 11.097 de 13/01/2005, autorizou o uso comercial do biodiesel introduzindo-o na matriz energética brasileira. No artigo 4°, biodiesel é definido como "biocombustível derivado de biomassa renovável para uso em motores a combustão interna com ignição por compressão ou, conforme regulamento, para geração de outro tipo de energia, que possa substituir parcial ou totalmente combustíveis de origem fóssil".

O método convencional de produção do biodiesel envolve a transesterificação de óleos vegetais e tal processo é realizado por meio de duas etapas distintas: a reação química dos óleos vegetais com o álcool comum, estimulada por um catalizador homogêneo, e a separação dos produtos em que ocorre a purificação para a obtenção do biodiesel final (Machado, 2009). A destilação reativa é uma operação em que ocorre simultaneamente a reação química e a separação dos produtos no mesmo equipamento.

O uso da coluna de destilação reativa (Stankiewicz & Moulijin, 2002 *apud* Machado, 2009) permite reduzir em aproximadamente 80% os custos do processo por meio da eliminação de etapas e equipamentos envolvidos, de 28 equipamentos e 11 etapas diferentes no processo convencional (Figura 3.1) para apenas uma coluna de destilação reativa (Figura 3.2).

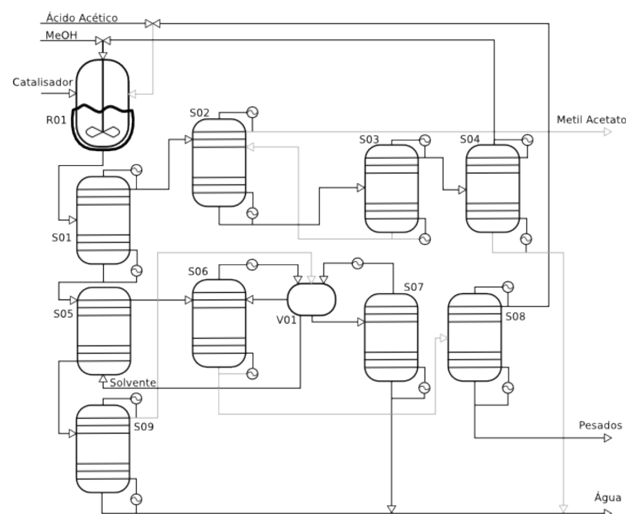


Figura 3.1: Representação esquemática do processo convencional (Machado, 2009)

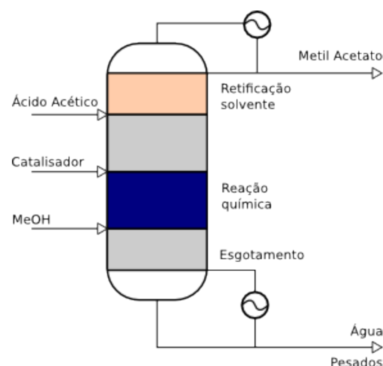


Figura 3.2: Representação esquemática do processo por coluna de destilação reativa (Machado, 2009)

A simulação computacional da coluna de destilação reativa utiliza sistemas de equações não lineares para a obtenção dos resultados simulados. Sobretudo, para a obtenção desses resultados é necessário o conhecimento de estimativas iniciais para esses sistemas.

3.2 O algoritmo

Em 2001, conforme Corazza et al. (2008), Michael W. Smiley e Changbum Chun, da Universidade do Estado de Iowa, nos Estados Unidos apresentaram um algoritmo que por meio de subdivisões poderia localizar simultaneamente todas as raízes de sistemas de equações algébricas não lineares, a partir de determinados intervalos de variáveis[1].

Em geral, o algoritmo de subdivisão consiste em, a partir de um sistema de variáveis não lineares, subdividir cada intervalo de variável em subintervalos de mesmo tamanho (congruentes), os quais são submetidos a um teste de avaliação para verificar se os mesmos continuam sendo intervalos possíveis, ou seja, se podem conter a solução para a respectiva variável (Corazza et al., 2008).

O subintervalo submetido ao teste de avaliação é mantido para novas subdivisões somente se o teste retornar um resultado positivo, caso contrário, é descartado. Após a execução de um número finito de subdivisões, os subintervalos serão tão pequenos que se aproximarão das raízes das respectivas variáveis. Nesse momento, é necessário utilizar um método de convergência local para que sejam encontradas as raízes. Deste modo, a solução (ou soluções) estará nos intervalos que foram mantidos, fornecendo assim convergência local eficaz e garantida do método empregado.

[1]Intervalo de variável: intervalo na reta real no qual o valor de uma variável incógnita pode ser encontrado. Por exemplo, o intervalo real [1,5] é um intervalo de variável para uma variável de valor 2

O algoritmo (Corazza et al., 2008) aqui utilizado, é capaz de encontrar estimativas iniciais para qualquer sistema de equações lineares e não lineares. Este algoritmo utiliza o método de Newton-Raphson para resolver um sistema fortemente não linear, como no caso da coluna de destilação reativa que, para convergência do método, depende de um grande número de estimativas iniciais. O funcionamento do algoritmo sequencial pode ser observado no Algoritmo 1.

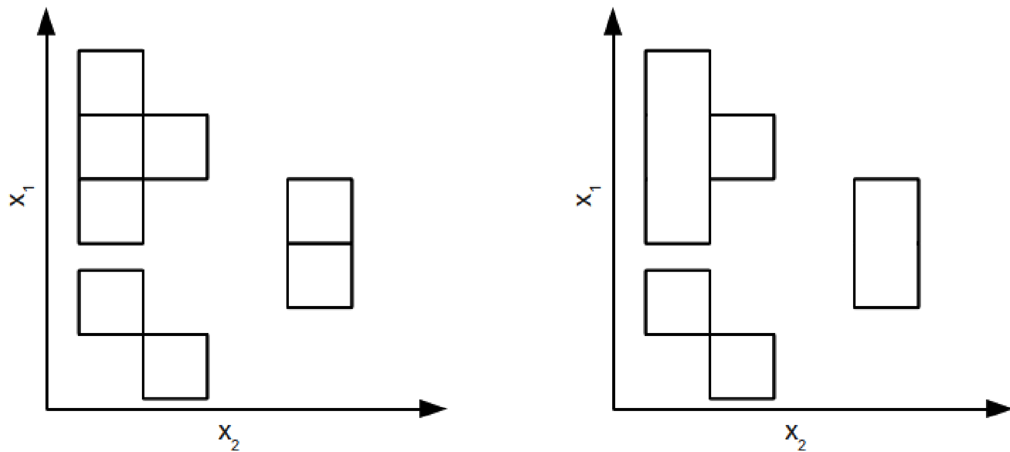
```
1  Entrada: Arquivo de configuração contendo o intervalo inicial e a quantidade de
            subdivisões
2  Saída: Estimativas que convergiram
3  Inicializa M ;
4  para l ← 1 até iCov faça
5  |  para i ← 1 até maxSub faça
6  |  |  Inicializa m;
7  |  |  para j ← 1 até M * 2^d faça
8  |  |  |  Obtém coordenadas;
9  |  |  |  Obtém números aleatórios;
10 |  |  |  Aplica f(x);
11 |  |  |  Avalia retângulo;
12 |  |  |  se retângulo aprovado então
13 |  |  |  |  Retêm retângulo;
14 |  |  |  |  Incrementa m;
15 |  |  |  senão
16 |  |  |  |  Descarta retângulo;
17 |  |  |  fim
18 |  |  fim
19 |  |  M ← m;
20 |  fim
21 |  Avalia retângulos adjacentes;
22 |  Redefine M;
23 fim
24 Aplica o método de Newton;
25 Verifica soluções encontradas;
26 Exibe soluções;
```

Algoritmo 1: Algoritmo da aplicação

O algoritmo inicialmente faz a leitura de um arquivo de configurações e inicializa as variáveis associadas. A variável *iCov* controla a quantidade de vezes que o teste de cobertura é aplicado. No teste de cobertura (Figura 3.3), os retângulos[2] que possuem

[2] Neste contexto, um retângulo pode ser definido como sendo uma estrutura de dados que possui um conjunto de intervalos (valor inicial e final) para cada dimensão do problema, formando assim uma estimativa.

as mesmas coordenadas (adjacentes) em pelo menos uma dimensão (Figura 3.3(a)) são usados para definir um novo retângulo após a aplicação do teste (Figura 3.3(b)). A variável $maxSub$ controla a quantidade de subdivisões. A cada subdivisão, o contador de retângulos mantidos na subdivisão atual (m) é inicializado.



(a) Retângulos mantidos após i subdivisões

(b) Retângulos apresentado em (a) após o teste de cobertura ser aplicado

Figura 3.3: Teste de cobertura (Corazza et al., 2008)

Cada retângulo pode gerar até 2^d sub-retângulos, sendo d o número de variáveis (dimensão). Desta forma, na linha 7 é feito um laço de repetição com base no produto da quantidade de retângulos mantidos nas subdivisões até o momento pela quantidade de sub-retângulos que cada retângulo pode gerar.

Entre as linhas 8 e 18 são obtidas as coordenadas de cada novo sub-retângulo e é aplicado a função do sistema de equações não lineares no ponto central de cada sub-retângulo. Cada sub-retângulo então é avaliado com o auxílio dos números aleatórios, que realizam ajustes nos valores do ponto central do sub-retângulo para prever a próxima iteração. Se o sub-retângulo for aprovado pelo critério de seleção será mantido e o contador de retângulos mantidos é incrementado na atual subdivisão, caso contrário, o sub-retângulo é descartado.

O exemplo ilustrado na Figura 3.4 demonstra como o processo de subdivisão de um retângulo ocorre. Para este exemplo, consideramos um problema com duas dimensões em que são aplicadas duas subdivisões. No nível 0, são obtidas as coordenadas dos sub-retângulos em que o retângulo inicial poderá ser subdividido, sendo cada sub-retângulo aplicado à função do sistema de equações e submetido ao teste de avaliação. Neste exemplo, todos os sub-retângulos que o retângulo inicial gerou foram aprovados e

mantidos e então, o algoritmo continua o processamento na próxima subdivisão. No nível 1, o processo de subdivisão ocorre novamente, entretanto, somente os sub-retângulos etiquetados com os valores 3 e 4 são mantidos. No nível 2, os sub-retângulos descartados no nível anterior, representado pelas áreas escuras maiores, não são utilizados na continuidade da aplicação. As áreas escuras menores são os sub-retângulos descartados no processo de subdivisão do nível 2.

Após avaliar todas as possibilidades (linha 19) o contador da quantidade total de retângulos mantidos é atualizado. Após o máximo de subdivisões ser atingido, é aplicado um teste de cobertura, caso tenha sido definido, para verificar a existência de retângulos adjacentes. Após ter sido estimado todos os sub-retângulos possíveis do retângulo inicial, é aplicado o método de Newton nos retângulos mantidos para verificar a existência de raízes (convergência). Qualquer convergência de algum retângulo é exibida na tela.

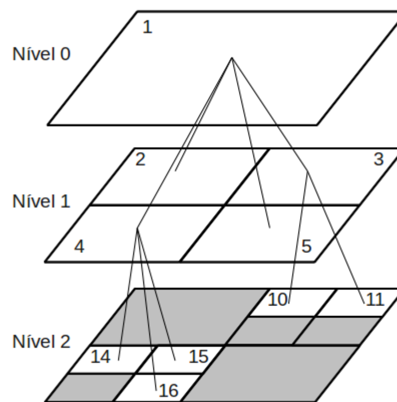


Figura 3.4: Representação gráfica do processo de subdivisão do retângulo inicial de duas dimensões após duas subdivisões

Uma estrutura em árvore, denominada neste estudo como árvore de estimativas é gerada para manter os sub-retângulos gerados, em que o nó raiz da árvore representa o retângulo inicial e cada nó filho da árvore representa um único sub-retângulo possível. Para encontrar os retângulos a serem mantidos, a árvore de estimativas é percorrida segundo a técnica "Primeiro em Largura" (*Breadth First*[3]).

O retângulo inicial, em sua primeira subdivisão, produz o primeiro nível da árvore. Todos os sub-retângulos presentes no primeiro nível são submetidos ao teste de avaliação. Somente os sub-retângulos que podem conter a solução são mantidos. O próximo nível é gerado a partir dos sub-retângulos retidos.

[3]*Breadth First*: visita todos os sucessores de um nó visitado antes de visitar quaisquer sucessores de qualquer um desses sucessores (Tenenbaum et al., 1995).

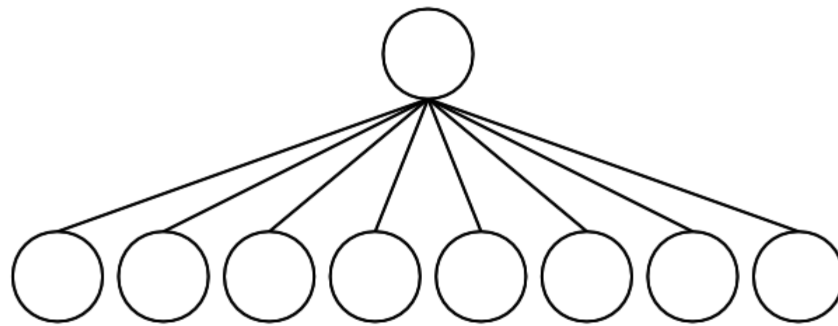
Capítulo 4

Modelos de Paralelização

No presente trabalho, quatro modelos de paralelização baseados na otimização do algoritmo de subdivisão foram desenvolvidos. Cada modelo possui duas versões, sendo que cada versão utiliza uma política de escalonamento entre duas possíveis, as quais são denominadas "escalonamento de nós adjacentes" e "escalonamento de nós equidistantes" (Figura 4.1), conforme explicadas mais adiante neste capítulo. Convém ressaltar que, apesar dos modelos serem baseados no algoritmo de subdivisão aqui apresentado, os mesmo foram propostos utilizando apenas as características da estrutura de armazenamento e método de percurso em árvores n-ária, com o objetivo de poderem ser aproveitados em quaisquer algoritmos da mesma classe.

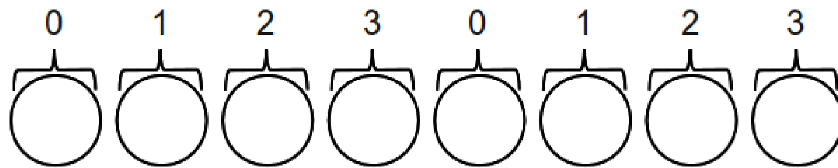
Na política de escalonamento de nós adjacentes (Figura 4.1(b)) todos os processadores possuem o retângulo inicial, que representa a raiz da árvore. Cada processador então gera somente os $\frac{2^d}{n}$ sub-retângulos subsequentes (adjacentes), onde d é a dimensão e n é o número de processadores envolvidos na computação. Na política de escalonamento de nós equidistantes (Figura 4.1(c)), o retângulo inicial também é conhecido porém, cada processador gera $\frac{2^d}{n}$ sub-retângulos equidistantes a um intervalo predefinido de n sub-retângulos.

O exemplo da Figura 4.1 ilustra uma árvore de estimativas gerada a partir de um retângulo inicial (nó raiz da árvore) com três dimensões após a primeira subdivisão na versão sequencial do algoritmo (Figura 4.1(a)). Na política de escalonamento de nós adjacentes, neste exemplo sendo executado por quatro processadores (indexados de 0 a 3), cada processador, de posse do retângulo inicial, gera somente os sub-retângulos



(a) Árvore gerada

(b) Política de escalonamento de nós adjacentes



(c) Política de escalonamento de nós equidistantes

Figura 4.1: Políticas de escalonamento

subsequentes que lhe compete: o processador 0 gera os dois primeiros, o processador 1 gera o terceiro e quarto sub-retângulos, o processador 2 gera o quinto e sexto sub-retângulos e finalmente o processador 3 gera os dois últimos sub-retângulos.

Na política de escalonamento de nós equidistantes, também sendo executado por quatro processadores (Figura 4.1(c)), o processador 0 gera o primeiro e o quinto sub-retângulos (obedecendo o intervalo de n), o processador 1 gera o segundo e o sexto sub-retângulos, o processador 2 gera o terceiro e sétimo sub-retângulos e o processador 3 gera o quarto e oitavo sub-retângulos.

Cada nó, após atingir uma quantidade de subdivisões pré-definida, aplica o método de convergência somente para o sub-retângulo que está no último nível da árvore (folhas), não sendo necessário armazenar na memória todos os sub-retângulos testados, tendo em vista que a aplicação do método de convergência só deve ser aplicado após um número finito de subdivisões.

Em ambos modelos, todos os retângulos são etiquetados com um valor inteiro, identificado-os individualmente dentro da estrutura independentemente como os sub-retângulos são distribuídos, conforme pode ser observado na Figura 4.2. Esses valores visam garantir o mesmo ambiente para todas as execuções das implementações, sendo utilizados como semente na geração de números aleatórios a cada sub-retângulo gerado.

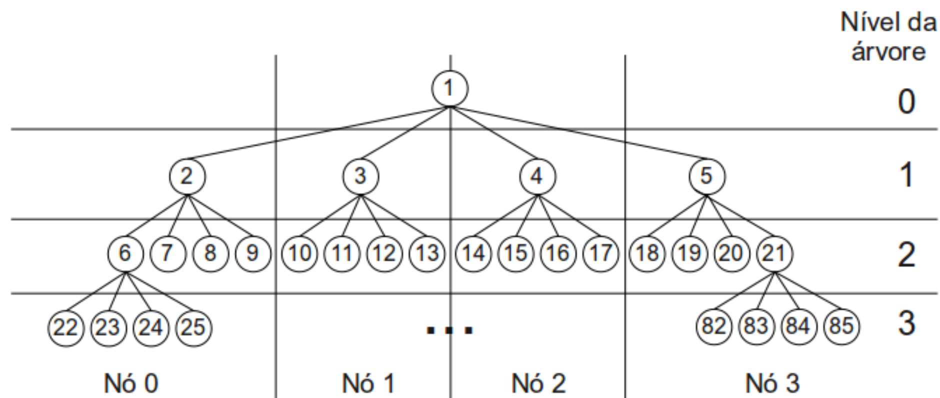


Figura 4.2: Árvore de estimativas com elementos enumerados globalmente

4.1 Modelo 01: Paralelização com Balanceamento Estático de Carga

O primeiro modelo é o mais simples dos quatro modelos desenvolvidos. Cada nó do *cluster* (processador), inclusive o nó mestre, gera na primeira subdivisão do retângulo inicial os sub-retângulos de sua competência, obedecendo a política de escalonamento. Note que o balanceamento estático de carga é de fato uma distribuição inicial de carga do tipo igualitária, que se tenta distribuir a mesma quantidade de sub-retângulos para cada nó do *cluster*.

A partir da segunda subdivisão, cada sub-retângulo poderá também gerar até 2^d novos sub-retângulos e assim sucessivamente, onde d representa a dimensão do problema. Entretanto, um sub-retângulo somente irá gerar novos sub-retângulos caso seja aprovado no teste de avaliação que verifica se este sub-retângulo tem potencial para ser uma boa estimativa inicial.

Neste modelo, não há troca de mensagem entre os nós do *cluster*. Após avaliar todos os sub-retângulos possíveis, cada nó permanece aguardando em uma barreira de sincronização até que todos os demais finalizem sua parcela de processamento. Quando todos os nós alcançarem a barreira, cada nó envia ao nó mestre os sub-retângulos

convergidos, caso possuam. A quantidade de sub-retângulos testados também é enviado a fim de verificar a acurácia do modelo. O nó mestre aglomera os dados recebidos e finaliza a aplicação.

Convém ressaltar que a barreira de sincronização utilizada neste modelo serve apenas para finalizar a execução paralela, permitindo agrupar os resultados parciais e enviá-los em um único momento ao nó mestre.

4.2 Modelo 02: Paralelização com Balanceamento Dinâmico de Carga por Ordem de Chegada

O segundo modelo desenvolvido faz uso de uma política de balanceamento dinâmico de carga em determinado nível da árvore, do tipo "receptor inicia", não sendo este nível alterado durante a execução da aplicação. A estratégia "receptor inicia" permite que os processadores ociosos procurem por processadores mais sobrecarregados.

No exemplo da Figura 4.3, uma árvore de estimativas com três níveis de subdivisões é gerada a partir de um problema de duas dimensões. O índice que controla a subdivisão decresce conforme a profundidade da árvore aumenta. Quando este índice atinge o valor de 1 (um), não ocorre mais subdivisões.

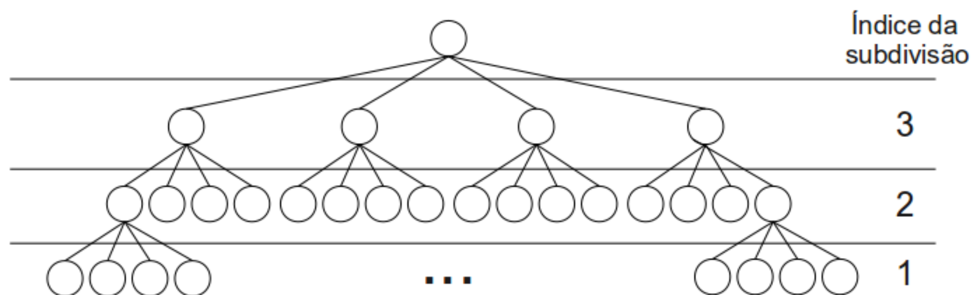


Figura 4.3: Identificação do nível de subdivisão

Quando um nó termina sua parcela de processamento, ele sinaliza para os demais que está ocioso e fica aguardando carga de trabalho. Quando qualquer nó, que ainda possuir retângulos a processar, passa pelo nível em que ocorre a verificação de balanceamento de carga, ele fica ciente se existe algum nó ocioso, por meio de um sinal. O nível em que ocorre a verificação de balanceamento refere-se em que momento da subdivisão a verificação é feita, sendo definido pelo valor do índice da subdivisão. Caso exista, o nó ciente seleciona um sub-retângulo ainda não processado e envia-o para o nó ocioso, removendo então o sinal que o bloqueava, colocando-o novamente para execução. Note que o atendimento

aos nós ociosos é feito por ordem de chegada ao ponto de verificação. O primeiro nó a chegar é o primeiro a atender um nó ocioso.

O nó ocioso recebe o sub-retângulo e os demais dados necessários para o processamento. Ao finalizar o processamento do sub-retângulo, sinaliza aos demais nós que está ocioso e permanece aguardando o recebimento de mais sub-retângulos. Caso ocorra de dois ou mais nós passem pelo nível de balanceamento ao mesmo tempo e selecionem o mesmo nó ocioso, é criado uma fila de sub-retângulos para o nó ocioso. Após ler e processar todos os elementos de sua fila, o nó pode novamente sinalizar que está ocioso.

O processo de balanceamento encerra quando todos os nós envolvidos no processamento sinalizarem que estão ociosos. A partir de então, cada nó envia ao nó mestre a quantidade de sub-retângulos processados e os convergidos. O nó mestre então aglomera os dados recebidos e encerra a aplicação. O início do processo de execução do mestre e a distribuição inicial do nó raiz da árvore de estimativas ocorrem da mesma forma que no modelo 01. Além disso, este modelo também faz uso de sincronização por barreira na finalização do paralelismo.

4.3 Modelo 03: Paralelização Síncrona com Balanceamento Dinâmico de Carga por Ordem das Maiores Cargas

O terceiro modelo funciona de maneira semelhante ao modelo anterior. Entretanto, quando um nó sinaliza que terminou sua parcela de processamento e que está ocioso, todos os nós ao passarem pelo nível de verificação de balanceamento de carga, ficam aguardando em uma barreira de sincronização para que todos informem a quantidade estimada de sub-retângulos a serem processados. Com uma finalidade diferente da barreira usada na finalização da paralelização, este modelo utiliza barreira intermediária para produzir oportunidades de checagem global de carga, em diversos momentos da execução, permitindo assim realizar o balanceamento de forma mais eficiente.

Todos os sub-retângulos, além de serem etiquetados de forma global (Figura 4.2), também são etiquetados de forma local (Figura 4.4). Essa enumeração é utilizada para realizar o cálculo da quantidade total de carga de cada nó.

Ao informarem a quantidade de carga restante, os dados são classificados em ordem decrescente, ou seja, da maior carga para a menor carga. Somente os nós que possuírem as maiores cargas deverão enviar aos nós ociosos. Se existir somente um nó ocioso, somente o nó que possuir a maior carga irá enviar sub-retângulos ao nó ocioso; se existirem dois nós

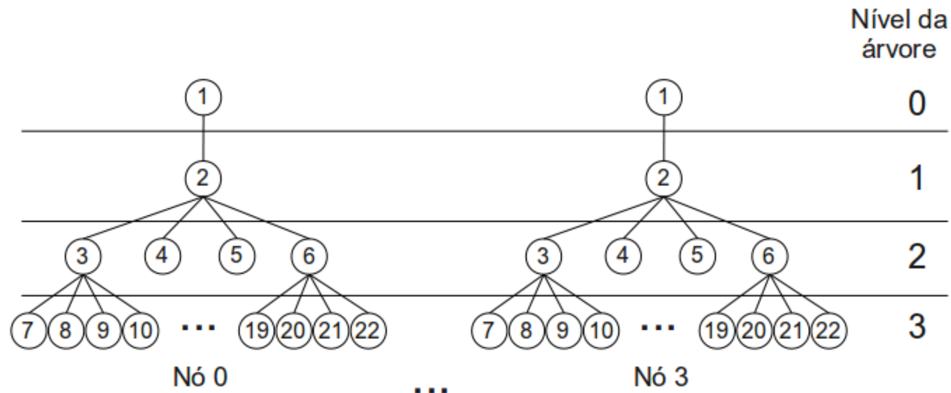


Figura 4.4: Árvore de estimativas com elementos enumerados localmente

ociosos, os dois nós que possuírem as maiores cargas restantes irão enviar sub-retângulos, e assim sucessivamente.

O nó ocupado que deve enviar carga de trabalho para um nó ocioso calcula quantos retângulos irmãos (retângulos filhos de um mesmo retângulo pai na árvore) faltam ser testados no nível onde ocorre a verificação de balanceamento carga. A quantidade restante é dividida de forma igual entre o nó que está enviando e o nó ocioso selecionado. Caso essa divisão resulte em um número fracionado (não inteiro), o nó que distribui fica com a maior carga.

Quando todos os nós sinalizarem que estão ociosos, o processo de balanceamento de carga é encerrado e então, o nó mestre recebe de cada nó envolvido no processamento a quantidade de sub-retângulos processados e os convergidos, aglomerando-os e encerrando a aplicação, assim como em todos os modelos.

4.4 Modelo 04: Paralelização Assíncrona com Balanceamento Dinâmico de Carga por Ordem das Maiores Cargas

O quarto modelo desenvolvido, assim como o terceiro modelo, utiliza o conceito de maior carga restante para definir qual nó irá distribuir carga para os nós ociosos. Entretanto, diferentemente do terceiro modelo, a partir do momento que existir um ou mais nós ociosos, os demais nós ocupados, ao passarem pelo nível de verificação de balanceamento de carga, não aguardam em uma barreira de sincronização para calcular qual o nó que possui a maior carga de trabalho restante.

Cada nó informa sua carga de trabalho e continua seu processamento. Assim que todos os nós ocupados informarem o valor de sua carga, um *flag* global é marcado. Assim que um nó passar novamente no nível de verificação de balanceamento e perceber que todos já se manifestaram sobre sua quantidade de carga de trabalho, ele verifica sua posição na classificação das cargas. Caso seja o mais ocupado (ou um dos mais ocupados, na existência de mais de um nó ocioso), ele calcula a quantidade restante de retângulos no nível e envia-os ao nó ocioso, caso contrário continua seu processamento normalmente.

O último nó que distribuir carga de trabalho desativa o *flag*, permitindo assim que todos os nós informem seus novos valores de carga de trabalho restante, possibilitando desta forma que os valores das cargas restantes sejam atualizados e uma nova classificação seja feita.

Após todos os nós sinalizarem que estão ociosos, o processo de balanceamento de carga é finalizado. Então, todos os nós enviam ao nó mestre a quantidade de sub-retângulos processados e os convergidos. As informações recebidas são aglomeradas pelo nó mestre e a aplicação é encerrada.

Capítulo

5

Implementação dos modelos

A aplicação alvo, originalmente escrita em linguagem Fortran, foi reescrita em linguagem C Ansi, sendo esta uma decisão de projeto que visa permitir que no futuro ela possa ser mais facilmente mantida pelo pessoal de computação. A escolha da linguagem C Ansi também foi motivada por permitir a integração de bibliotecas de manipulação de *thread* como Pthreads ou OpenMP, visando o desenvolvimento de modelos de paralelização híbridos. Desta forma, essa mudança de linguagem facilitou identificar uma ineficiência do algoritmo original, no uso da memória, causada pelo algoritmo de percurso da árvore, a qual foi então resolvida. O objetivo do trabalho é o de criar modelos de paralelização que sejam eficientes sobre uma boa implementação sequencial e não somente sobre uma programação sequencial não otimizada. A próxima seção apresenta essa melhoria.

5.1 Otimizando o uso da memória

Conforme visto na Seção 3.2, o algoritmo básico da aplicação utiliza o percurso “Primeiro em Largura” para percorrer a árvore de estimativas gerada e, utilizando este percurso, todos os retângulos observados são armazenados, pois os mesmos são usados na descida da árvore até encontrar todos os retângulos da última iteração (folhas), os quais são, no final, submetidos aos critérios de seleção e convergência. Com isso, a utilização de memória aumenta rapidamente, devido ao crescimento acelerado da árvore com o avanço das iterações, promovendo a ocorrência intensiva de *swap*. Para grandes sistemas, em vários testes realizados, quase sempre a execução foi abortada devido ao estouro da capacidade

de armazenamento. O tempo de processamento também cresce demasiadamente com essa abordagem original.

Para otimizar o uso da memória, optou-se por trocar a política de percurso para "Primeiro em Profundidade"(*Depth First*[1]) para não precisar armazenar todos os retângulos observados. Com essa nova abordagem, somente os retângulos observados durante a descida rumo a um retângulo folha são armazenados. Além disso, sempre que um retângulo folha é encontrado, o mesmo já é submetido ao critério de seleção e caso seja aprovado, já é submetido ao teste de convergência final, caso contrário, ele já é descartado. Durante o retorno do caminho descido, os retângulos intermediários que não forem mais descidos em outros filhos são descartados. Dessa forma, somente os nós aprovados no teste de convergência final permanecem armazenados.

A paralelização foi realizada a partir do modelo otimizado, que utiliza o percurso "Primeiro em Profundidade". A paralelização do algoritmo foi baseada no modelo mestre-escravo, onde o nó mestre efetua a leitura do arquivo de configuração e distribui para os nós escravos as variáveis de configuração bem como o retângulo inicial, e posteriormente recebe as soluções encontradas pelos nós escravos. Sobretudo, é válido salientar que o nó mestre também é utilizado no processamento dos sub-retângulos.

Cada retângulo recebe um identificador único (etiqueta). Esse valor é utilizado como semente, por meio da função *srand* da linguagem C, para a geração de números aleatórios. Essa abordagem se fez necessário para estabelecer uma igualdade entre os modelos no sentido de que todos sejam executados nas mesmas condições.

Todos os modelos implementados utilizam um arquivo de configuração que é lido pelo nó mestre (Figura 5.1). No arquivo de configuração são definidos os seguintes valores:

- Problema: identifica o código do problema dentro da aplicação. Este código serve para executar o sistema de equação não linear adequado ao problema;
- Dimensão: identifica a quantidade de dimensões do sistema de equação;
- ColunasMatriz: identifica a quantidade de colunas existentes no retângulo inicial;
- MaxSubdivisões: especifica a quantidade de subdivisões aplicadas no retângulo inicial;
- MaxIterações: especifica a quantidade de vezes que serão avaliados os retângulos adjacentes

[1] *Depth First*: visita todos os sucessores de um nó visitado antes de visitar qualquer um de seus "irmãos"(Tenenbaum et al., 1995).

- MetodoIterativo: identifica qual método de convergência será aplicado;
- Valores do intervalo de cada dimensão do retângulo inicial.

```
Problema:   71
Dimensão:   7
ColunasMatriz:  2
MaxSubdivisões: 3
MaxIterações:   0
MetodoIterativo:    1
0.0E0       100.0E0
0.0E0       10.0E0
0.0E0       60.0E0
0.0E0       10.0E0
0.0E0       1.0E0
3.0E0       5.0E0
0.0E0       1.0E0
```

Figura 5.1: Arquivo de configuração do modelo 01

5.2 Plataforma operacional

Os experimentos paralelos foram executados no *cluster* existente no Laboratório Experimental de Computação de Alto Desempenho (LECAD) do Departamento de Informática da Universidade Estadual de Maringá. O *cluster* do LECAD é composto por 10 computadores sendo: 1 servidor com processador Intel Core 2 Quad 2.4 Ghz, 4 núcleos, 2 Gb de memória RAM, 6 computadores com, cada um, processador AMD Opteron 1218 2.6 Ghz, 2 núcleos, 4 Gb de memória RAM, e 3 computadores com, cada um, 2 processadores Intel Xeon E5620 2.4 Ghz, 4 núcleos *Hyper-Threading*[2], 8 Gb de memória RAM. Todos os computadores possuem interface de rede *gigabit* conectados por um *switch gigabit*. A implementação da plataforma MPI utilizada é a OpenMPI 1.3.2. A distribuição Linux utilizada é a Rocks Cluster, com Kernel 2.6.18, que é uma distribuição gratuita que possui facilidades de instalação, controle de versão, gerenciamento e integração (Sacerdoti et al., 2004).

Clusters também precisam ser monitorados, como por exemplo, na realização de auditorias, contabilidade, avaliação de desempenho, *feedback*, entre outros. Com o passar do tempo, um *cluster* tende a crescer com o acréscimo de novos nós. Porém, nem

[2] *Hyper-Threading*: permite que cada núcleo execute dois processos ao mesmo tempo. (Tanenbaum, 2007)

sempre é possível conservar a homogeneidade, assim como a confiabilidade dos nós. Desta forma, o monitoramento do *cluster* torna-se um tópico importante. Para realizar esse monitoramento são necessários dois passos (Sacerdoti et al., 2003): a) um sistema que identifique falhas no *cluster*; b) um mecanismo de acompanhamento detalhado que possibilite aos usuários ajustar a execução das aplicações paralelas.

Ganglia é um sistema de monitoramento distribuído escalável utilizado em sistemas de computação de alto desempenho como *clusters* e *grids* (Massie et al., 2004). Ganglia é distribuído gratuitamente junto com a distribuição Rocks Clusters. Porém, é compatível com um extenso conjunto de arquiteturas e sistemas operacionais, sendo utilizado por milhares de instituições no mundo, onde podemos citar o Twitter, Wikipedia, UOL, entre outros.

Ganglia utiliza algoritmos e estruturas de dados cuidadosamente projetados para produzir baixas sobrecargas por nó de processamento, não prejudicando assim o desempenho do *cluster* (Ganglia, 2010).

5.2.1 Open MPI

O objetivo do projeto Open MPI é proporcionar um ambiente de execução paralela para uma extensa variedade de sistemas computacionais, com robustez e alto desempenho (Graham et al., 2006). Embora o projeto Open MPI seja uma união dos projetos LAM/MPI, LA-MPI e FT-MPI, aproveitando o que existia de melhor nesses projetos, o Open MPI é uma implementação totalmente nova, obedecendo por completo as especificações do MPI-1 e MPI-2, permitindo ainda um completo suporte a aplicações concorrentes e *multithreads* (Gabriel et al., 2004).

Embora o Open MPI possua um amplo conjunto de funções do padrão MPI implementadas, a grande maioria dos problemas pode ser resolvida utilizando somente as funções descritas na Tabela 5.1 (Foster, 1995). É importante salientar que, apesar do Open MPI possuir suporte para Fortran e C/C++, neste estudo foi utilizada somente a linguagem C. Para que o pré-processador C encontre as referências das funções MPI é necessário incluir a diretiva *#include"mpi.h"*. Neste trabalho, os códigos apresentados mostram somente a chamada da função, não sendo discutido sobre os parâmetros a serem utilizados. Maiores informações podem ser obtidas em (OpenMPI, 2010).

Uma aplicação MPI compõe-se de um conjunto de processos MPI, sendo cada um executado em um nó de processamento do *cluster*. Pelo fato de haver apenas um processo MPI em cada nó de processamento, muitas vezes dizemos apenas "nó mestre" ou "nó escravo" para nos referirmos a "processo mestre" ou "processo escravo", respectivamente.

Conforme já mencionado, processo mestre é aquele que coordena a distribuição de tarefas e agrupamento dos resultados na computação mestre-escravo, usada no presente trabalho. Processo escravo é aquele que recebe uma tarefa, a executa e devolve os resultados ao processo mestre.

Segundo El-Rewini e Abd-El-Barr (2005), um requisito importante em um sistema de passagem de mensagem é o de garantir uma comunicação segura entre os processos. O Open MPI, assim como outras bibliotecas MPI, utilizam o conceito de comunicadores a fim que garantir esse requisito de comunicação segura. Comunicadores pode ser definidos como uma coleção de processos que podem enviar mensagens uns aos outros (Pacheco, 2011). Quando o ambiente MPI é inicializado, um comunicador padrão contendo todos os processos é definido, denominado MPI_COMM_WORLD. Outro comunicador pré-definido é o MPI_COMM_SELF que inclui somente o processo que realiza a chamada.

Função	Descrição
MPI_Init	Inicializa o ambiente de execução MPI
MPI_Comm_size	Determina o tamanho do grupo associado ao comunicador
MPI_Comm_rank	Determina a identificação do processo associado ao comunicador
MPI_Send	Envia uma mensagem
MPI_Recv	Recebe uma mensagem
MPI_Finalize	Finaliza o ambiente de execução MPI

Tabela 5.1: Funções básicas do Open MPI

Processos MPI se comunicam por meio de rotinas de comunicação MPI. A rotina de comunicação ponto-a-ponto *MPI_Send* utiliza o modo de comunicação padrão (*Standard*), em que a decisão de utilizar o *buffer* interno fica a cargo da implementação. Entretanto, o Open MPI possui rotinas implementadas que utilizam os demais modos de comunicação listados na Seção 2.2.4, conforme pode ser observado na Tabela 5.2 que também relaciona a variação não-bloqueante de cada rotina.

Modo de comunicação	Rotinas Bloqueantes	Rotinas Não Bloqueantes
Standard	*MPI_Send*	*MPI_Isend*
Buffered	*MPI_Bsend*	*MPI_Ibsend*
Ready	*MPI_Rsend*	*MPI_Irsend*
Syncronous	*MPI_Ssend*	*MPI_Issend*

Tabela 5.2: Rotinas de envio Open MPI

Entretanto, segundo Grama et al. (2003), ao utilizar as rotinas de comunicação não bloqueantes é necessário certificar-se que estas operações foram concluídas antes

de prosseguir o processamento. Para verificar a conclusão de rotinas não-bloqueantes, utiliza-se as rotinas *MPI_Test* ou *MPI_Wait*. Enquanto a primeira rotina realiza um teste para verificar se a operação terminou, a outra aguarda até que a operação seja concluída.

O Open MPI também implementada as rotinas de comunicação coletivas descritas na Seção 2.2.4. Todos os processos participantes na comunicação devem chamar a rotina de comunicação coletiva (Foster, 1995). As rotinas implementadas podem ser observadas na Tabela 5.3. Até a versão utilizada neste trabalho, não existia a variação não-bloqueante das rotinas de comunicação coletiva.

Operação	Rotina
Barreira	*MPI_Barrier*
Difusão	*MPI_Bcast*
Juntar	*MPI_Gather*
Espalhar	*MPI_Scatter*
Redução	*MPI_Reduce*

Tabela 5.3: Funções de comunicação coletiva do Open MPI

O Open MPI também possui rotinas de entrada e saída (E/S) paralela implementadas para acesso a arquivos e dispositivos (Gabriel et al., 2004). Segundo Sterling (2001), a ideia fundamental na abordagem MPI para E/S paralela é que um arquivo pode ser aberto por um conjunto de processos, permitindo que operações, individuais ou coletivas, possam ser realizadas sobre um arquivo. As principais rotinas podem ser observadas na Tabela 5.4:

Rotina	Descrição
MPI_File_open	Abre um arquivo
MPI_File_set_view	Cria uma visão do arquivo para cada processo, permitindo que vários processos acessem o mesmo arquivo
MPI_File_write	Efetua a gravação dos dados no arquivo
MPI_File_read	Efetua a leitura do arquivo
MPI_File_get_size	Recupera o tamanho total do arquivo
MPI_File_close	Fecha um arquivo

Tabela 5.4: Rotinas de entrada e saída paralelas

5.3 Modelo 01

O funcionamento do primeiro modelo pode ser observado no Algoritmo 2. O ambiente MPI é inicializado, sendo atribuído um identificador único para cada processo MPI, sendo representado no algoritmo pela variável $myId$. O valor da variável $myId$ varia entre 0, que é atribuído ao nó mestre, e o total de processos envolvidos no processamento menos um.

O nó mestre efetua a leitura do arquivo de configuração e inicializa as variáveis associadas e principalmente aloca a estrutura inicial da árvore, ou seja, o nó raiz da árvore que irá conter o retângulo inicial. Então, o mestre distribui o retângulo inicial e demais dados necessários para os demais nós do *cluster* por meio do uso da primitiva MPI *Bcast*.

Ao utilizar o percurso "Primeiro em Profundidade" foi possível utilizar recursividade para gerar os sub-retângulos. Desta forma, não é necessário armazenar todos os sub-retângulos testados, somente os que convergiram. A função recursiva desenvolvida, denominada $SubdivideRecursivo()$, pode ser observada no Algoritmo 3.

```
1  Inicializa ambiente MPI;
2  se myId == 0 então
3  |  Lê arquivo de configuração e aloca estruturas;
4  fim
5  Distribui o retângulo inicial e demais valores;
6  SubdivideRecursivo(maximoSubdivisao, retângulo inicial, etiqueta global,
   etiqueta local);
7  Barreira;
8  se myId == 0 então
9  |  Recebe valores dos nós escravos;
10 senão
11 |  Envia valores para o nó mestre;
12 fim
13 se myId == 0 então
14 |  Grava dados no arquivo de saída;
15 |  Imprime dados na tela;
16 fim
17 Finaliza ambiente MPI;
```

Algoritmo 2: Algoritmo principal do modelo 01

Ao realizar a primeira subdivisão, que é controlada pela variável *subDivisaoAtual*, os sub-retângulos são gerados de acordo com a política de escalonamento utilizada. A partir

da segunda subdivisão, a quantidade de sub-retângulos que serão testados será 2^d, em que d representa a dimensão do problema. Na linha 6 inicia o laço de repetição que controla a navegabilidade dentro da árvore.

Cada sub-retângulo recebe dois identificadores, um global e outro local. O identificador global é utilizado como semente para a geração de números aleatórios e o identificador local é utilizado para calcular a quantidade de carga restante de cada nó, ambos utilizam os valores dos identificadores do retângulo "pai".

Então, o ponto central da estimativa (intervalos de variáveis) armazenada em cada novo sub-retângulo é submetido à função do sistema de equações não lineares. Esse sub-retângulo então é avaliado e se for aprovado pelo critério de seleção, será utilizado como parâmetro para a chamada recursiva da função *SubdivideRecursivo*(), caso contrário, será descartado. O processo recursivo encerra somente quando a variável *subDivisaoAtual* atingir o valor de 1 (um). Neste momento, o sub-retângulo é submetido ao método de Newton para verificar a existência de raiz e, caso haja convergência, será armazenado.

```
1  se subDivisaoAtual == maximoSubDivisao então
2  |   tamanhoTrabalho ← quantidade de subretângulos de acordo com a política de
   |   escalonamento;
3  senão
4  |   tamanhoTrabalho ← 2^d;
5  fim
6  para l ← 0 até tamanhoTrabalho faça
7  |   Calcula a etiqueta do retângulo de forma global e local;
8  |   Obtém coordenadas;
9  |   Obtém números aleatórios;
10 |   Aplica f(x);
11 |   Avalia retângulo;
12 |   se retângulo aprovado então
13 |   |   se subDivisaoAtual == 1 então
14 |   |   |   Aplica o método de Newton;
15 |   |   |   Verifica solução encontrada;
16 |   |   senão
17 |   |   |   SubdivideRecursivo(subDivisaoAtual -1, retângulo, etiqueta global,
   |   |   |   etiqueta local);
18 |   |   fim
19 |   fim
20 fim
```

Algoritmo 3: Função SubdivideRecursivo do modelo 01

Após um número finito de subdivisões dos sub-retângulos que foram mantidos, pode ocorrer da árvore estar desbalanceada, conforme pode ser observado na Figura 5.2. Neste exemplo, foram realizadas 5 subdivisões em um retângulo de 2 dimensões. Os nós das árvores marcados com "X" foram descartados no teste de avaliação. Após realizar todas as subdivisões, a possível solução está presente apenas em um conjunto restrito de sub-retângulos da árvore, tornando-a assim desbalanceada, o que implica no surgimento de nós ociosos de forma muito mais rápida, sugerindo desta forma a utilização de balanceamento de carga dinâmico.

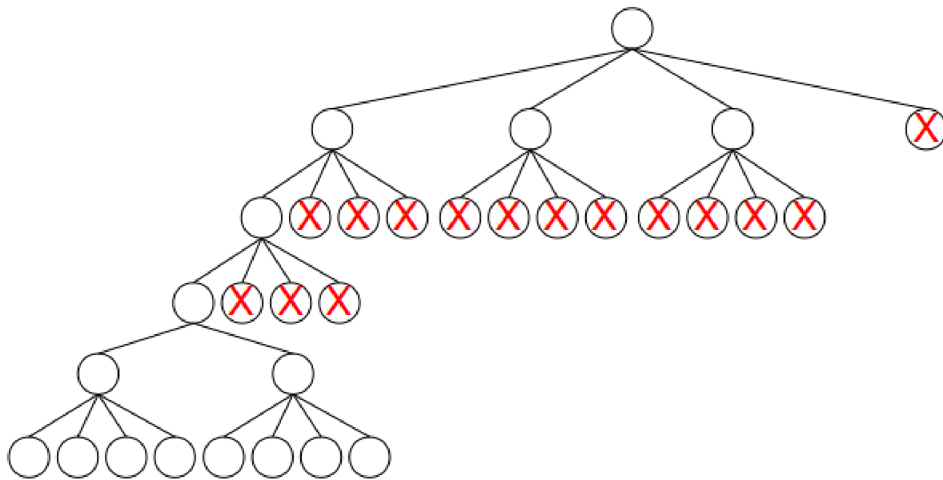


Figura 5.2: Representação da árvore de estimativas gerada por meio do processo de subdivisão de um retângulo com 2 dimensões

Após verificar todos os sub-retângulos, cada nó fica aguardando em uma barreira até que todos os demais nós atinjam o mesmo ponto. Após passarem pela barreira, todos os nós enviam seus dados para o nó mestre para que sejam gravados no arquivo de saída e informados na tela o usuário. Antes de finalizar o programa, o ambiente MPI deve ser finalizado.

5.4 Modelo 02

O algoritmo principal (Algoritmo 4) desenvolvido no segundo modelo possui funcionamento semelhante ao primeiro modelo. O nó mestre efetua a leitura do arquivo de configuração, aloca as estruturas necessárias para receber os dados lidos e as distribui para os demais nós envolvidos no processamento.

No arquivo de configuração deste modelo foi inserido um novo parâmetro: Balanceamento. Esse parâmetro identifica em qual nível da subdivisão ocorrerá o balanceamento de carga. Quando um nó termina sua parcela de processamento, sinaliza aos demais nós sua disponibilidade (linha 7), e aguarda até receber novos retângulos a serem testados.

Cada vez que um nó atinge o nível de balanceamento de carga, como pode ser observado no Algoritmo 5, é verificado a existência de nós disponíveis. Caso haja, o primeiro nó a atingir este ponto irá selecionar um nó disponível e enviará o retângulo atual para este nó disponível e retira-o da fila de nós disponíveis.

```
1  Inicializa ambiente MPI;
2  se myId == 0 então
3  |  Lê arquivo de configuração e aloca estruturas;
4  fim
5  Distribui o retângulo inicial e demais valores;
6  SubdivideRecursivo(maximoSubdivisao, retângulo inicial, etiqueta global,
   etiqueta local);
7  Sinaliza disponibilidade do nó;
8  repita
9  |  Aguarda recebimento de retângulos;
10 |  repita
11 |  |  SubdivideRecursivo (subdivisaoRecebida -1, retângulo, etiqueta
   |  |  global, etiqueta local);
12 |  até existe fila de retângulos ;
13 |  Sinaliza disponibilidade do nó;
14 até quantidadeTrabalhadores != quantidadeNosDisponiveis ;
15 Barreira;
16 se myId == 0 então
17 |  Recebe valores dos nós escravos;
18 senão
19 |  Envia valores para o nó mestre;
20 fim
21 se myId == 0 então
22 |  Grava dados no arquivo de saída;
23 |  Imprime dados na tela;
24 fim
25 Finaliza ambiente MPI;
```

Algoritmo 4: Algoritmo principal do modelo 02

Caso ocorra de dois ou mais nós atingirem o nível de balanceamento ao mesmo tempo e selecionem o mesmo nó disponível, é criado uma fila de sub-retângulos a serem processados pelo nó disponível. Ao enviar o sub-retângulo atual, o algoritmo realiza um salto para o

próximo sub-retângulo "irmão" através da instrução *continue* da linguagem C. Se o nó, ao passar pelo nível de balanceamento de carga, verificar que não existe nós disponíveis, continua o processo de recursividade.

O processo de balanceamento de carga encerra quando todos os nós sinalizarem que estão disponíveis. Após todos os nós passarem pela barreira de sincronização, o nó mestre recebe de todos os demais os dados para serem gravados no arquivo de saída e exibidos na tela.

```
1  se subDivisaoAtual == maximoSubDivisao então
2  |  tamanhoTrabalho ← quantidade de subretângulos de acordo com a política de
   |  escalonamento;
3  senão
4  |  tamanhoTrabalho ← 2^d;
5  fim
6  para l ← 0 até tamanhoTrabalho faça
7  |  Calcula a etiqueta do retângulo de forma global e local;
8  |  Obtém coordenadas;
9  |  Obtém números aleatórios;
10 |  Aplica f(x);
11 |  Avalia retângulo;
12 |  se retângulo aprovado então
13 |  |  se subDivisaoAtual == nivelBalanceamento então
14 |  |  |  se existe nós disponíveis então
15 |  |  |  |  noDestino ← nó disponível;
16 |  |  |  |  envia retângulo ao noDestino;
17 |  |  |  |  continua;
18 |  |  |  fim
19 |  |  fim
20 |  |  se subDivisaoAtual == 1 então
21 |  |  |  Aplica o método de Newton;
22 |  |  |  Verifica solução encontrada;
23 |  |  senão
24 |  |  |  SubdivideRecursivo(subDivisaoAtual -1, retângulo, etiqueta global,
   |  |  |  etiqueta local);
25 |  |  fim
26 |  fim
27 fim
```

Algoritmo 5: Função SubdivideRecursivo do modelo 02

5.5 Modelo 03

O algoritmo principal que implementa o terceiro modelo também possui o mecanismo de balanceamento de carga e possui o mesmo arquivo de configuração do segundo modelo. Entretanto, diferentemente do segundo modelo, conforme pode ser observado na linha 9 do Algoritmo 6 e linha 19 do Algoritmo 7, esse processo ocorre de maneira síncrona.

```
1  Inicializa ambiente MPI;
2  se myId == 0 então
3  |   Lê arquivo de configuração e aloca estruturas;
4  fim
5  Distribui o retângulo inicial e demais valores;
6  SubdivideRecursivo(maximoSubdivisao, retângulo inicial, etiqueta global,
   etiqueta local);
7  Sinaliza disponibilidade do nó;
8  repita
9  |   Coleta carga dos processos de maneira síncrona;
10 |   se todas a carga de todos os processos == 0 então
11 |   |   Encerra Repita;
12 |   fim
13 |   Aguarda recebimento de retângulos;
14 |   repita
15 |   |   SubdivideRecursivo (subdivisaoRecebida -1, retângulo, etiqueta
   |   |   global, etiqueta local);
16 |   até existe retângulos no vetor recebido ;
17 |   Sinaliza disponibilidade do nó;
18 até quantidadeTrabalhadores != quantidadeNosDisponiveis ;
19 Barreira;
20 se myId == 0 então
21 |   Recebe valores dos nós escravos;
22 senão
23 |   Envia valores para o nó mestre;
24 fim
25 se myId == 0 então
26 |   Grava dados no arquivo de saída;
27 |   Imprime dados na tela;
28 fim
29 Finaliza ambiente MPI;
```

Algoritmo 6: Algoritmo principal do modelo 03

Quando um nó termina sua parcela de processamento e sinaliza ao demais que está disponível, realiza uma chamada à função MPI *Allgather*[3], informando que sua carga de trabalho é igual a 0 (zero). Um nó ocupado, ao passar pelo ponto de verificação de balanceamento, também realiza uma chamada à função *Allgather* informando sua carga de trabalho. Desta forma, a informação contendo a carga de trabalho restante de cada nó será difundida entre todos os nós envolvidos no processamento.

Após realizado a coleta da carga de trabalho restante, a estrutura onde estão armazenados esses valores é ordenada de forma decrescente, permitindo assim identificar quais os nós que possivelmente consumirão mais tempo de processamento. Somente o nó mais ocupado (ou os mais ocupados, na existência de mais de um nó disponível) irá enviar parte da carga de trabalho.

Diferentemente do algoritmo desenvolvido no modelo 02, entre as linhas 6 e 14 do Algoritmo 7, os sub-retângulos são testados de forma antecipada e somente aqueles que forem aprovados no teste serão armazenados no vetor. Essa alteração foi necessária para permitir que o nó ocupado divida a quantidade de sub-retângulos retidos na iteração com o nó (ou nós) disponível ao invés de enviar somente um único sub-retângulo.

Após enviar a parcela de sub-retângulos retidos para o nó ocupado, o valor do índice que controla a iteração é atualizado para que o processamento dos sub-retângulos continue e não processe os sub-retângulos enviados. O processo de balanceamento de carga encerra quando todos os nós realizarem a chamada à função *Allgather* enviando o valor de 0, sinalizando assim sua disponibilidade. O nó mestre recebe os dados dos demais nós e grava-os no arquivo de saída, além de exibi-los na tela.

5.6 Modelo 04

O algoritmo principal desenvolvido a partir do modelo 04 possui basicamente as mesmas apresentadas no Algoritmo 6 e utiliza o mesmo arquivo de configuração do segundo modelo. Entretanto, como pode ser observado nos Algoritmos 8 e 9, o balanceamento de carga ocorre de maneira assíncrona. Para sinalizar sua disponibilidade, o nó que acabou sua parcela de processamento, grava em um arquivo, utilizando as funções não-bloqueantes do MPI-IO, o valor 0 e aguarda o recebimento de retângulos a serem processados.

O balanceamento para os nós ocupados ocorre em duas etapas. Na primeira etapa, na existência de nós disponíveis, o nó grava em um arquivo, denominado arquivo de cargas, sua carga de trabalho restante e continua seu processamento. Quando a quantidade de nós

[3] A função Allgather tem o objetivo de reunir dados de todos os processos e difundir esses dados para todos os processos (OpenMPI, 2010).

```
1  se subDivisaoAtual == maximoSubDivisao então
2  |  tamanhoTrabalho ← quantidade de subretângulos de acordo com a política de
   |  escalonamento;
3  senão
4  |  tamanhoTrabalho ← 2^d;
5  fim
6  para l ← 0 até tamanhoTrabalho faça
7  |  Calcula a etiqueta do retângulo de forma global e local;
8  |  Obtém coordenadas;
9  |  Obtém números aleatórios;
10 |  Aplica f(x);
11 |  Avalia retângulo;
12 |  se retângulo aprovado então
13 |  |  vetorRetangulosAprovados[] ← retângulo aprovado;
14 |  fim
15 fim
16 para l ← 0 até quantidadeRetangulosAprovados faça
17 |  se subDivisaoAtual == nivelBalanceamento então
18 |  |  se existe nós disponíveis então
19 |  |  |  Coleta carga dos processos de maneira síncrona;
20 |  |  |  Ordena estrutura com o tamanho das cargas;
21 |  |  |  se minhaPosicao < quantidade de nós disponíveis então
22 |  |  |  |  Calcula meusNos ;
23 |  |  |  |  Calcula a parcela de retângulos retidos para cada nó ;
24 |  |  |  |  para i ← 0 até meusNos faça
25 |  |  |  |  |  noDestino ← nó disponível;
26 |  |  |  |  |  envia parcela de retângulos retidos ao noDestino;
27 |  |  |  |  |  atualiza l;
28 |  |  |  |  |  continua;
29 |  |  |  |  fim
30 |  |  |  fim
31 |  |  fim
32 |  fim
33 |  se subDivisaoAtual == 1 então
34 |  |  Aplica o método de Newton;
35 |  |  Verifica solução encontrada;
36 |  senão
37 |  |  SubdivideRecursivo(subDivisaoAtual -1, retângulo, etiqueta global,
   |  |  etiqueta local);
38 |  fim
39 fim
```

Algoritmo 7: Função SubdivideRecursivo do modelo 03

disponíveis somada à quantidade de nós que informaram sua carga for igual à quantidade de nós envolvidos na paralelização, um *flag* é definido. Um nó ocupado irá somente informar sua carga de trabalho novamente após a *flag* de verificação ter sido removida.

Após a definição da *flag*, a segunda etapa do balanceamento de carga é habilitada. Nesta etapa, os valores das cargas de trabalho são lidos do arquivo de cargas e transportados para uma estrutura que é ordenada de forma decrescente. Somente o nó (ou nós) que possuir a maior carga de trabalho restante irá enviar sua parcela de retângulos retidos para o nó disponível.

Assim como o algoritmo do modelo anterior, o balanceamento de carga é finalizado assim que todos os nós informarem o valor de 0 (zero). Após, os nós escravos enviam os dados para o nó mestre para que este grave-os no arquivo de saída e exiba-os na tela.

```
1  Inicializa ambiente MPI;
2  se myId == 0 então
3  |  Lê arquivo de configuração e aloca estruturas;
4  fim
5  Distribui o retângulo inicial e demais valores;
6  SubdivideRecursivo(maximoSubdivisao, retângulo inicial, etiqueta global,
   etiqueta local);
7  Sinaliza disponibilidade do nó;
8  repita
9  |  se todas a carga de todos os processos == 0 então
10 |  |  Encerra Repita;
11 |  fim
12 |  Aguarda recebimento de retângulos;
13 |  repita
14 |  |  SubdivideRecursivo (subdivisaoRecebida -1, retângulo, etiqueta
   |  |  global, etiqueta local);
15 |  até existe retângulos no vetor recebido ;
16 |  Sinaliza disponibilidade do nó;
17 até quantidadeTrabalhadores != quantidadeNosDisponiveis ;
18 Barreira;
19 se myId == 0 então
20 |  Recebe valores dos nós escravos;
21 senão
22 |  Envia valores para o nó mestre;
23 fim
24 se myId == 0 então
25 |  Grava dados no arquivo de saída;
26 |  Imprime dados na tela;
27 fim
28 Finaliza ambiente MPI;
```

Algoritmo 8: Algoritmo principal do modelo 04

```
1  se subDivisaoAtual == maximoSubDivisao então tamanhoTrabalho ←
   quantidade de subretângulos de acordo com a política de escalonamento;
2  senão tamanhoTrabalho ← 2^d;
3  para l ← 0 até tamanhoTrabalho faça
4  |   Calcula a etiqueta do retângulo de forma global e local;
5  |   Obtém coordenadas; Obtém números aleatórios;
6  |   Aplica f(x);
7  |   Avalia retângulo;
8  |   se retângulo aprovado então
9  |   |   vetorRetangulosAprovados[] ← retângulo aprovado;
10 |   fim
11 fim
12 para l ← 0 até quantidadeRetangulosAprovados faça
13 |   se subDivisaoAtual == nivelBalanceamento então
14 |   |   se existe nós disponíveis então
15 |   |   |   se não existe flag de verificação E não informou tamanho da carga
   |   |   |   então Informa tamanho da carga restante;
16 |   |   |   se quantidade de nós disponíveis + quantidade de nós que informaram
   |   |   |   carga == quantidade de trabalhadores então Define flag verificação;
17 |   |   fim
18 |   |   se existe flag de verificação então
19 |   |   |   Leitura e Ordenação estrutura com o tamanho das cargas;
20 |   |   |   se minhaPosicao < quantidade de nós disponíveis então
21 |   |   |   |   Calcula meusNos ;
22 |   |   |   |   Calcula a parcela de retângulos retidos para cada nó ;
23 |   |   |   |   para i ← 0 até meusNos faça
24 |   |   |   |   |   noDestino ← nó disponível;
25 |   |   |   |   |   envia parcela de retângulos retidos ao noDestino;
26 |   |   |   |   |   atualiza l;
27 |   |   |   |   |   se Último nó ocupado a enviar dados então Remove flag de
   |   |   |   |   |   verificação ;
28 |   |   |   |   |   continua;
29 |   |   |   |   fim
30 |   |   |   fim
31 |   |   fim
32 |   fim
33 |   se subDivisaoAtual == 1 então
34 |   |   Aplica o método de Newton;
35 |   |   Verifica solução encontrada;
36 |   senão
37 |   |   SubdivideRecursivo(subDivisaoAtual -1, retângulo, etiqueta global,
   |   |   etiqueta local);
38 |   fim
39 fim
```

Algoritmo 9: Função SubdivideRecursivo do modelo 04

Capítulo

6

Experimentos e resultados

Nos experimentos realizados, os tempos coletados foram contabilizados do início ao fim da execução da aplicação, não sendo discriminados tempos de comunicação, tempo de processamento e tempo de ociosidade. É importante salientar que o nó mestre além de distribuir o retângulo inicial, também realiza o processamento (subdivisões), sendo portanto incluído nos cálculos de *speedup* e eficiência.

Dados de casos reais conhecidos pertencentes a dois sistemas de equações não-lineares, um com 5 dimensões e outro com 7 dimensões, disponíveis em (Polymath-Software, 2011) juntamente com suas respectivas soluções, foram utilizados como entrada nos modelos implementados. Para o sistema com 5 dimensões foram aplicadas 6 níveis de subdivisões (iterações), e para o sistema com 7 dimensões foram aplicadas 3 subdivisões. Todos os modelos implementados empregam o paralelismo de dados e foram executados sobre 1, 2, 4, 8, 16 e 32 processadores.

No Apêndice D são apresentados os valores das soluções conhecidas disponíveis na literatura e os valores obtidos por meio da execução da aplicação, a fim de validar a qualidade numérica dos resultados obtidos.

6.1 Tempos de execução

Cada modelo paralelo desenvolvido utilizou duas políticas de escalonamento distintas já discutidas no Capítulo 4. Desta forma, os comparativos entre cada modelo serão feitos obedecendo a política utilizada. Para a obtenção dos tempos de execução, foram realizados

3 execuções para cada conjunto de nós envolvidos no processamento e calculado a média aritmética entre elas. Os gráficos das médias dos tempos de execução obtidos podem ser observados nas Figuras 6.1 e 6.2 para o problema com 5 dimensões e Figuras 6.3 e 6.4 para o problema com 7 dimensões. Os valores podem ser observados, respectivamente, nas Tabelas A.1, A.2, A.3 e A.4 contidas no Apêndice A.

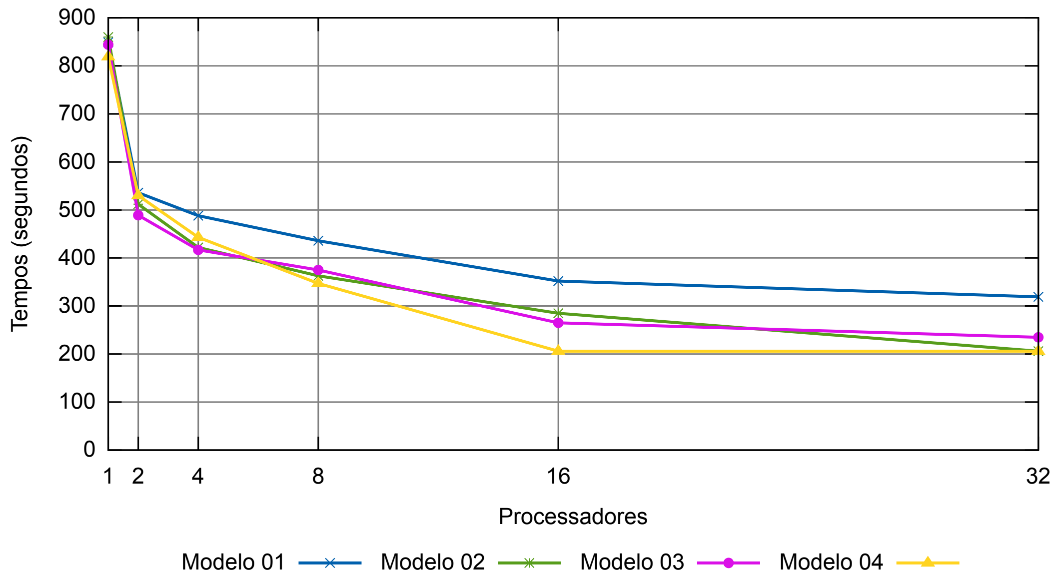


Figura 6.1: Tempos de execução do problema com 5 dimensões utilizando a política de escalonamento de nós adjacentes (em segundos)

De forma geral, observando as curvas representadas nos gráficos, nota-se que o tempo total de execução da aplicação diminui consideravelmente em função do aumento do número total de processadores envolvidos na computação paralela, satisfazendo assim um dos seus principais objetivos. Executando o caso com 5 dimensões, a redução de tempo chegou à ordem de 74% utilizando a política de escalonamento de nós adjacentes e 77% utilizando a política de escalonamento de nós equidistantes, enquanto que, executando o problema com 7 dimensões a redução chegou à ordem de 95% para ambas as políticas de escalonamento.

Entretanto, essa redução não é linear, tendendo a se estabilizar conforme o número de processadores envolvidos aumenta. Analisando os gráficos de tempo de execução para o caso com 5 dimensões (Figuras 6.1 e 6.2) observa-se que, para problemas em que o tempo de execução é curto, a redução de tempo de execução foi relativamente menor.

Os modelos implementados utilizando a política de escalonamento de nós equidistantes obtiveram uma estabilização de redução de tempo mais antecipada, a partir de 4 processadores, comparados aos modelos utilizando a política de escalonamento de nós

adjacentes, considerando a execução do caso com 5 dimensões. Considerando a execução do problema com 7 dimensões (Figuras 6.3 e 6.4) essa estabilização ocorreu a partir de 16 processadores.

Com o objetivo de medir a sobrecarga da produzida pela adição da infraestrutura de paralelização no algoritmo sequencial, foi incluído nos gráficos a execução dos modelos paralelos utilizando apenas um único processador. É possível observar nos gráficos que houve sobrecarga para todos os modelos propostos, utilizando ambas políticas de escalonamento.

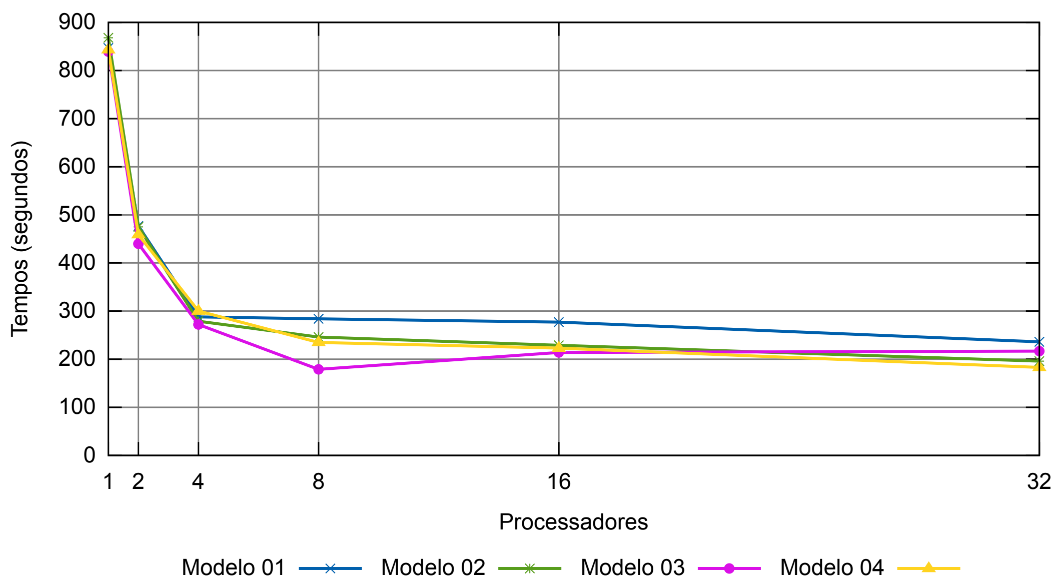


Figura 6.2: Tempos de execução do problema com 5 dimensões utilizando a política de escalonamento de nós equidistantes (em segundos)

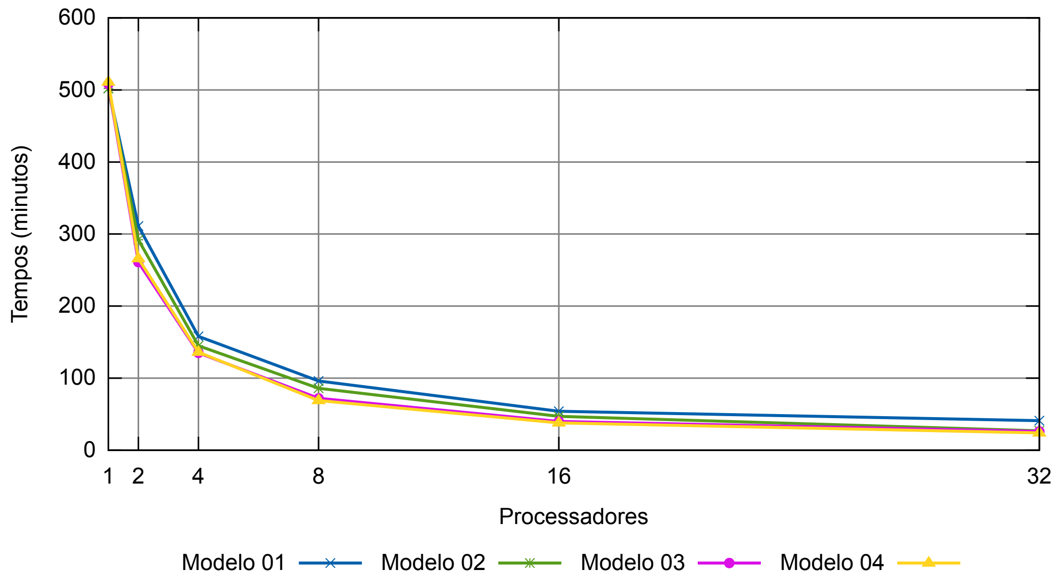


Figura 6.3: Tempos de execução do problema com 7 dimensões utilizando a política de escalonamento de nós adjacentes (em minutos)

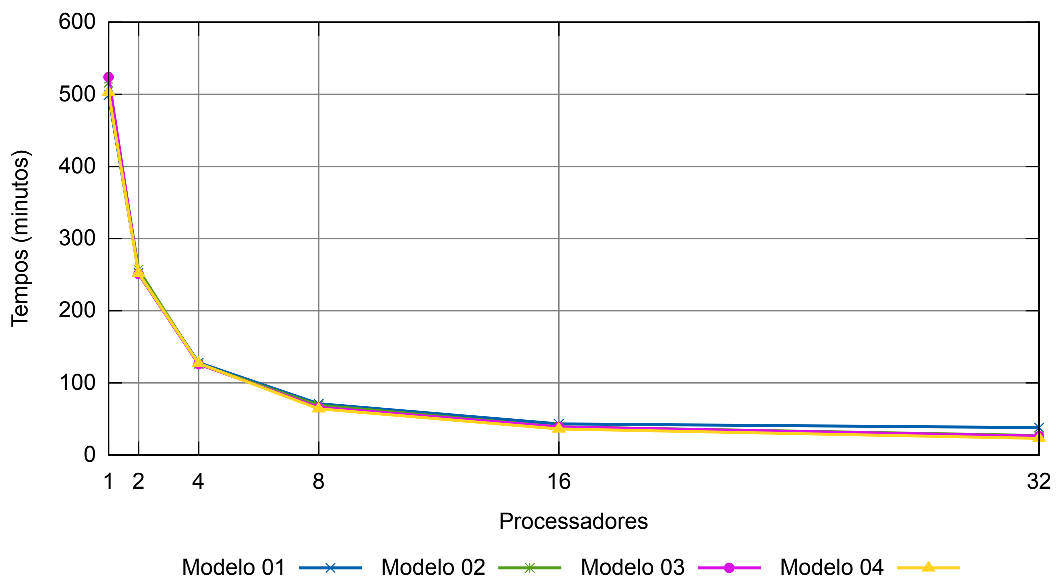


Figura 6.4: Tempos de execução do problema com 7 dimensões utilizando a política de escalonamento de nós equidistantes (em minutos)

Desta forma, aumentar o número de processadores não implica necessariamente em reduzir o tempo de execução. Possivelmente, a partir de determinado número de processadores envolvidos, a tendência é que os algoritmos paralelos consumam mais tempo, principalmente devido ao custo da comunicação entre os processadores ou à

ociosidade dos processadores causado pelo desbalanceamento da árvore de estimativas no caso do modelo que utiliza o balanceamento estático, sendo o grau deste consumo dependente do modelo implementado.

6.2 Speedup

O *speedup* de cada modelo paralelo foi obtido dividindo o tempo de execução gasto pelo programa na versão sequencial pelo tempo gasto na versão paralela. Os gráficos dos *speedups* obtidos podem ser observados nas Figuras 6.5 e 6.6 para 5 dimensões e Figuras 6.7 e 6.8 para 7 dimensões, onde os valores intermediários foram interpolados e uma curva contendo os valores do *speedup* ideal teórico foi adicionada para ser utilizada como referência. Os valores dos *speedups* podem ser visualizados nas Tabelas B.1 e B.2 para o caso com 5 dimensões e Tabelas B.3 e B.4 para o caso com 7 dimensões contidos nos Apêndice B.

Analisando os gráficos de *speedup* para a execução utilizando o caso com 5 dimensões (Figuras 6.5 e 6.6), é possível observar que o valores de *speedup* se mantiveram estáveis, não sendo beneficiados pelo aumento de processadores envolvidos. Essa característica ocorre devido ao fato do problema necessitar de pouco tempo de processamento, sendo saturado pela sobrecarga causada pela ociosidade dos processadores, no caso do Modelo 01 ou pela política de balanceamento implementada nos Modelos 02, 03 e 04.

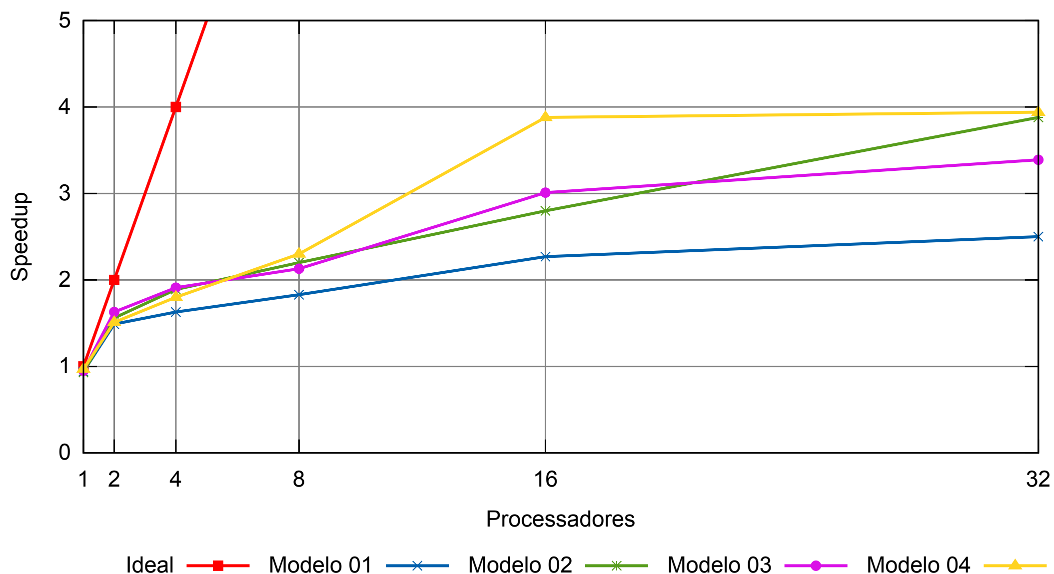


Figura 6.5: Speedups obtidos com problema de 5 dimensões utilizando a política de escalonamento de nós adjacentes

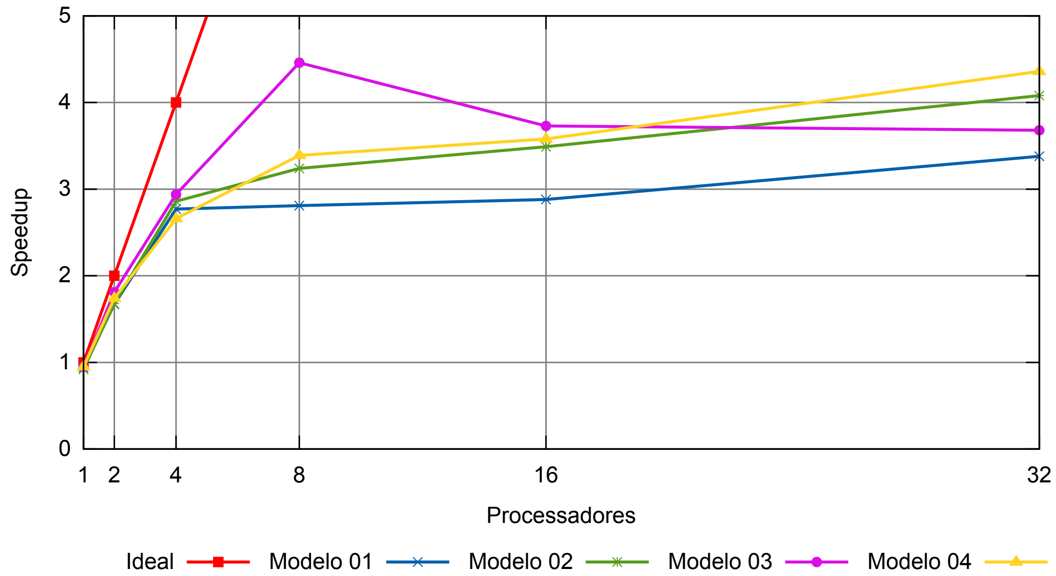


Figura 6.6: Speedups obtidos com problema de 5 dimensões utilizando a política de escalonamento de nós equidistantes

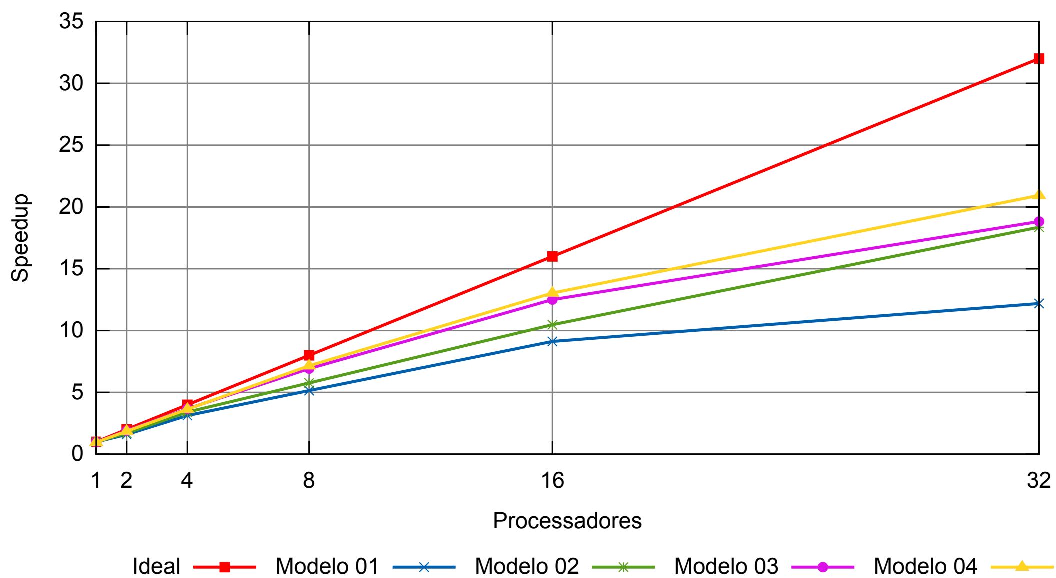


Figura 6.7: Speedups obtidos com problema de 7 dimensões utilizando a política de escalonamento de nós adjacentes

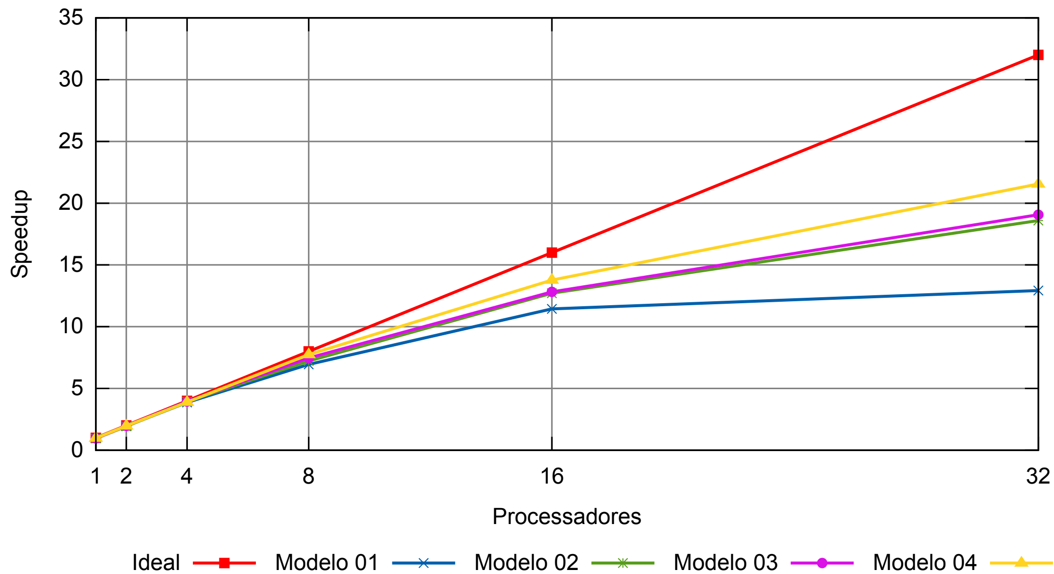


Figura 6.8: Speedups obtidos com problema de 7 dimensões utilizando a política de escalonamento de nós equidistantes

No entanto, observa-se nas Tabelas 6.1 e 6.2, as quais apresentam o percentual de ganho (ou perda) de desempenho conforme a evolução dos modelos desenvolvidos, que os modelos que possuem mecanismo de balanceamento de carga obtiveram um melhor desempenho em relação ao modelo que não possui essa característica, obtendo picos de mais de 50%. Sobretudo, para este problema, não é possível determinar qual o modelo mais indicado a ser utilizado em qualquer quantidade de processadores. Para até 4 processadores, o modelo que obteve um melhor desempenho foi o Modelo 03, enquanto que acima de 8 processadores, o Modelo 04 atingiu níveis melhores desempenho, ambos utilizando a política de escalonamento de nós adjacentes, enquanto que utilizando a política de nós equidistantes, o Modelo 03 obteve os melhores resultados até 16 processadores. Para 32 processadores, o Modelo 04 atingiu o melhor nível.

Utilizando o problema com 7 dimensões, é possível observar nos gráficos ilustrados nas Figuras 6.7 e 6.8 que os modelos implementados executados em paralelo proporcionaram um ganho de desempenho significativo sobre a versão sequencial, chegando próximo ao linear com até 16 processadores.

Todas as curvas dos gráficos tiveram uma redução no ganho de desempenho ao utilizar 32 processadores. Isso se deve ao fato de que, ao realizar os experimentos, foram selecionados no *cluster* apenas os computadores homogêneos que proporcionavam um maior número de processadores possíveis, sendo 3 computadores com 2 processadores

Nr. de Nós	$\frac{Modelo02}{Modelo01}$	$\frac{Modelo03}{Modelo02}$	$\frac{Modelo04}{Modelo03}$
01	-1,06%	2,15%	2,11%
02	4,70%	4,49%	-7,36%
04	15,95%	1,06%	-5,76%
08	20,22%	-3,18%	7,98%
16	23,35%	7,50%	28,90%
32	55,20%	-12,63%	16,22%

Tabela 6.1: Relação de ganho de desempenho conforme evolução dos modelos utilizando a política de escalonamento de nós adjacentes para o problema com 5 dimensões

Nr. de Nós	$\frac{Modelo02}{Modelo01}$	$\frac{Modelo03}{Modelo02}$	$\frac{Modelo04}{Modelo03}$
01	-2,13%	3,26%	0,00%
02	0,60%	7,74%	-3,87%
04	3,25%	2,80%	-9,52%
08	15,30%	37,65%	-23,99%
16	21,18%	6,88%	-4,02%
32	20,71%	-9,80%	18,48%

Tabela 6.2: Relação de ganho de desempenho conforme evolução dos modelos utilizando a política de escalonamento de nós equidistantes para o problema com 5 dimensões

Quad-Core cada (a descrição completa pode ser vista na Seção 5.2). Totalizando o número de núcleos (*cores*), obtém-se o valor de 24.

Entretanto, essa arquitetura possui o recurso de *Hyper-Threading*, permitindo o paralelismo em nível de *threads*, em que cada núcleo executa simultaneamente dois processos compartilhando alguns recursos. Dessa forma, nestes 3 computadores, é possível executar até 48 processos simultaneamente, porém, com ligeira queda de desempenho uma vez que alguns recursos de hardware são compartilhados. É importante ressaltar que durante os experimentos realizados não foi possível garantir o uso exclusivo do *cluster* para a execução das aplicações, não sendo descartadas interferências referentes ao escalonamento de processo de outros usuários.

Contudo, é possível afirmar, por meio dos valores contidos nas Tabelas 6.3 e 6.4, que o modelo ideal para a execução deste problema é o Modelo 04 utilizando a política de escalonamento de nós equidistantes, em que obteve ganho de desempenho melhor em relação aos demais modelos. Utilizando a política de escalonamento de nós adjacentes, com até 4 processadores, o modelo mais indicado é o Modelo 03, enquanto que a partir de 8 processadores o mais indicado é Modelo 04.

Nr. de Nós	$\frac{Modelo02}{Modelo01}$	$\frac{Modelo03}{Modelo02}$	$\frac{Modelo04}{Modelo03}$
01	0,00%	-1,01%	-1,02%
02	6,92%	11,76%	-2,11%
04	8,95%	7,62%	-0,82%
08	11,84%	20,14%	3,32%
16	14,80%	19,39%	4,32%
32	50,62%	2,51%	11,26%

Tabela 6.3: Relação de ganho de desempenho conforme evolução dos modelos utilizando a política de escalonamento de nós adjacentes para o problema com 7 dimensões

Nr. de Nós	$\frac{Modelo02}{Modelo01}$	$\frac{Modelo03}{Modelo02}$	$\frac{Modelo04}{Modelo03}$
01	-3,03%	-1,04%	3,16%
02	-1,53%	2,07%	0,00%
04	1,81%	0,00%	-1,27%
08	3,88%	3,46%	3,61%
16	11,10%	0,87%	7,41%
32	43,77%	2,53%	13,06%

Tabela 6.4: Relação de ganho de desempenho conforme evolução dos modelos utilizando a política de escalonamento de nós equidistantes para o problema com 7 dimensões

6.3 Eficiência

Os gráficos apresentados nas Figuras 6.9 e 6.10 exibem a eficiência obtida pelos modelos paralelos sobre o caso com 5 dimensões, enquanto as Figuras 6.11 e 6.12 exibem os gráficos referentes à eficiência obtida pelos modelos paralelos sobre o caso com 7 dimensões. Assim como nos gráficos de *speedup*, os valores intermediários foram interpolados e uma curva com os valores da eficiência ideal teórica foi adicionada para ser utilizada como referência. Os valores das curvas dos gráficos podem ser visualizados nas Tabelas C.1, C.2, C.3 e C.4, consecutivamente, contidas no Apêndice C.

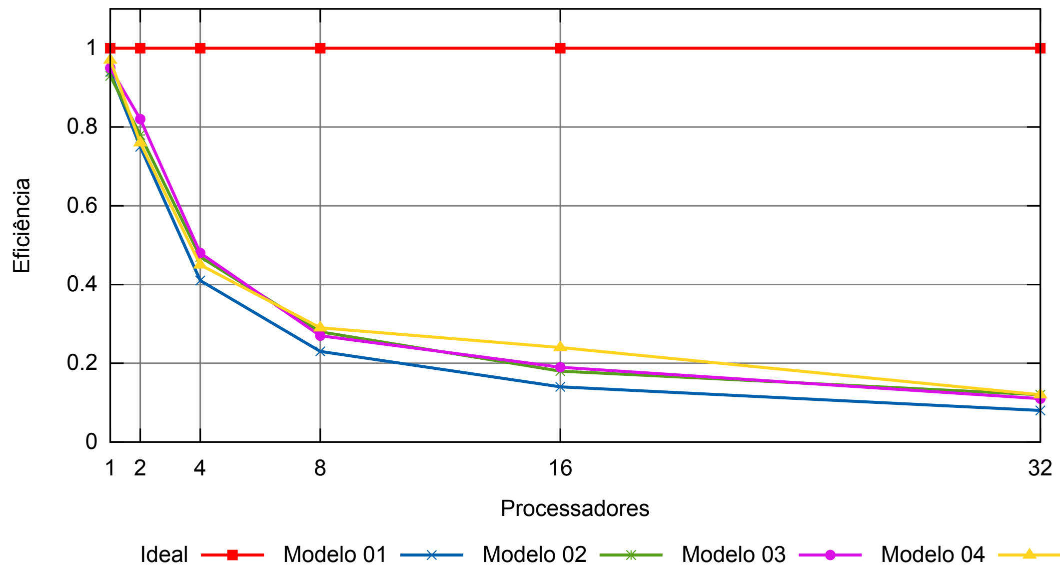


Figura 6.9: Eficiências obtidas com problema de 5 dimensões utilizando a política de escalonamento de nós adjacentes

Apesar dos modelos desenvolvidos apresentarem redução do tempo total de execução sobre o caso com 5 dimensões, é possível observar através das Figuras 6.9 e 6.10 que houve uma queda de eficiência dos modelos em ambas as políticas de escalonamento, sendo diretamente relacionada ao aumento do número de processadores envolvidos. Essa queda ocorre devido ao fato da granulosidade da aplicação ser fina, incorrendo em uma maior sobrecarga gerada pelas políticas de balanceamento de carga, no caso dos modelos que implementam este recurso, ou pela ociosidade dos processadores, no caso do Modelo 01.

Analisando os gráficos de eficiência dos modelos sobre o caso com 7 dimensões (Figuras 6.11 e 6.12) é possível avaliar que os recursos computacionais foram utilizados de forma mais satisfatória, tendo maior destaque os Modelos 03 e 04 que obtiveram bons índices, na ordem de valores maiores ou iguais a 0,8 com até 16 processadores. Utilizando 32 processadores houve uma pequena queda de eficiência, conforme discutido na Seção 6.2, relacionada ao fato dos processos serem executadas em *Hyper-Threading*.

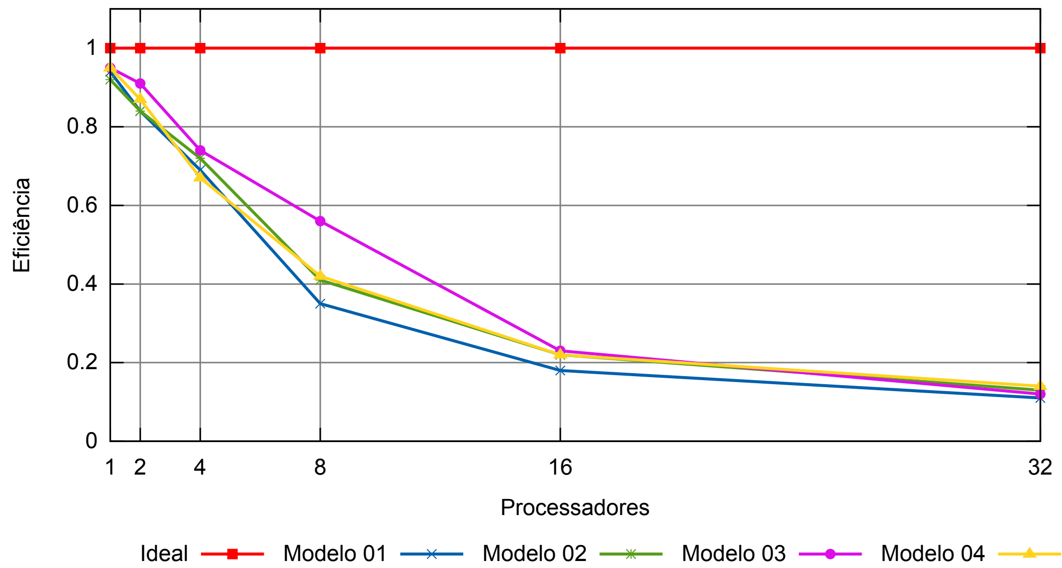


Figura 6.10: Eficiências obtidos com problema de 5 dimensões utilizando a política de escalonamento de nós equidistantes

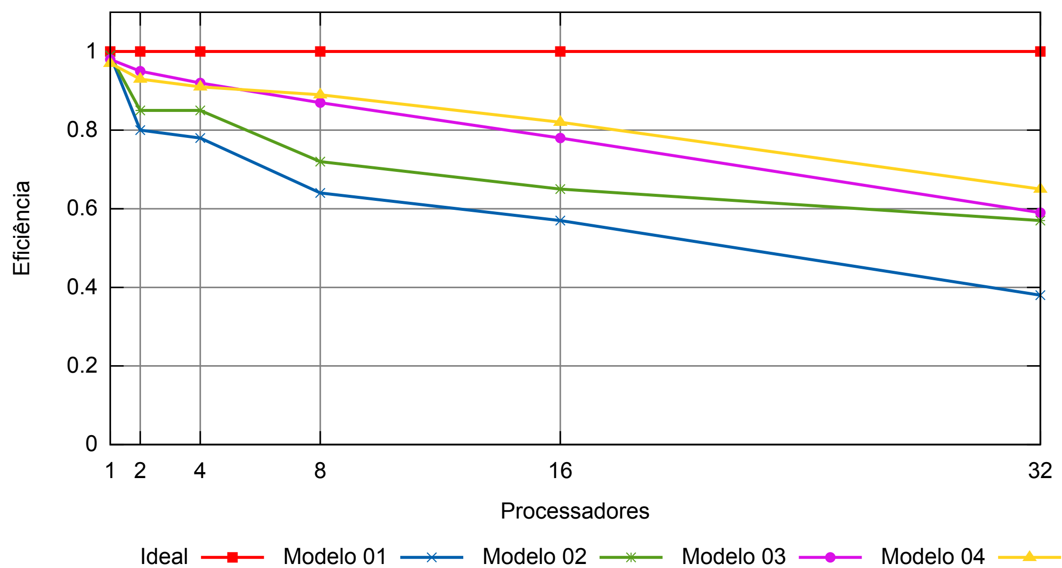


Figura 6.11: Eficiências obtidas com problema de 7 dimensões utilizando a política de escalonamento de nós adjacentes

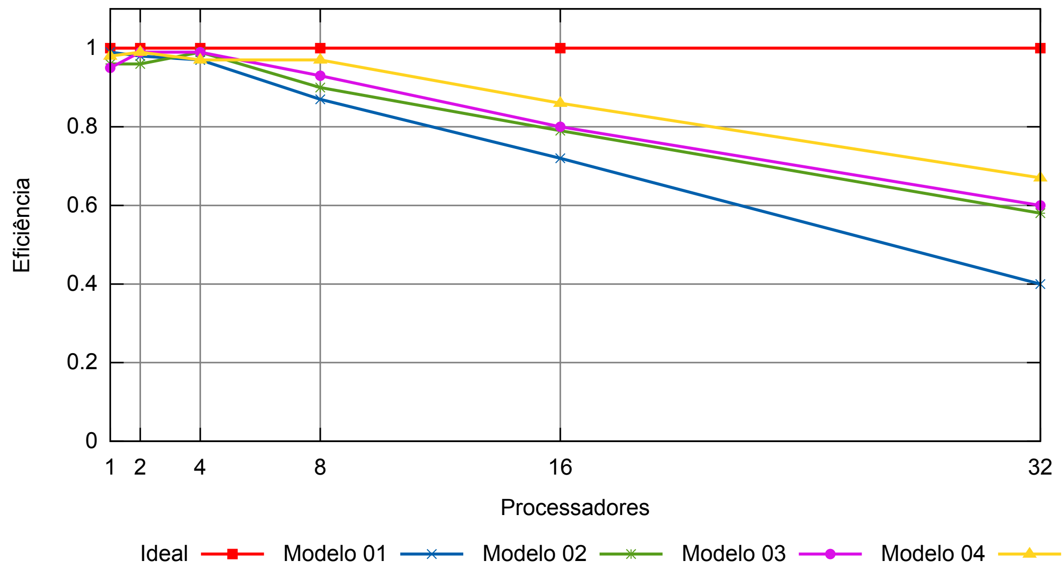


Figura 6.12: Eficiências obtidas com problema de 7 dimensões utilizando a política de escalonamento de nós equidistantes

Capítulo 7

Conclusões e trabalhos futuros

Nos últimos anos, *clusters* têm se mostrado uma excelente alternativa aos supercomputadores multiprocessados devido ao seu baixo custo associado à sua arquitetura. Nesse sentido, a utilização do paradigma de programação baseado em passagem de mensagens em ambientes de programação paralela tem sido favorecida.

Neste trabalho, foram apresentados quatro modelos de algoritmos paralelos para aplicações que são estruturadas em árvores. O problema alvo utilizado foi o algoritmo de subdivisão (Corazza et al., 2008) utilizado na geração de estimativas iniciais para sistemas de equações não-lineares que simulam o problema da coluna de destilação reativa.

A implementação dos modelos foi escrita em linguagem de programação C, utilizando a biblioteca de passagem de mensagens OpenMPI. Os experimentos foram executados no *cluster* existente no Laboratório Experimental de Computação de Alto Desempenho (LECAD) do Departamento de Informática da Universidade Estadual de Maringá. Apesar de o *cluster* possuir 10 computadores, os experimentos utilizaram somente os computadores que possibilitavam a utilização de um maior número de processadores homogêneos. Entretanto, se faz necessário estudos sobre a execução dos experimentos em *clusters* heterogêneos a fim de avaliar os impactos causados pela arquitetura e o desenvolvimento de políticas de balanceamento de carga que levem em consideração a heterogeneidade.

Preliminarmente, para otimizar o uso de memória, o algoritmo sequencial original foi alterado de forma que o percurso da árvore de estimativas gerada pela aplicação, que originalmente era “Primeiro em Largura”, foi alterado para “Primeiro em Profundidade”. Essa modificação foi necessária devido ao fato do algoritmo original produzir estouro de

memória à medida que o número de dimensões do sistema de equações não lineares e a quantidade de subdivisões aplicadas ao retângulo inicial aumentavam.

Outra otimização realizada, visando garantir o mesmo ambiente para todas as execuções, foi associar a geração da semente (*seed*) utilizada para a geração de números aleatórios a cada sub-retângulo (estimativa) gerado, conforme já explicado.

Todos os modelos desenvolvidos obedeceram duas políticas de escalonamento distintas para a divisão do retângulo inicial, nomeadas de "escalonamento de nós adjacentes" e "escalonamento de nós equidistantes", gerando assim duas versões para cada modelo. O primeiro modelo desenvolvido utilizou uma política de balanceamento de carga estático, enquanto que os três modelos subsequentes utilizaram políticas de balanceamento dinâmicas.

Dados obtidos em dois casos reais conhecidos foram utilizados para a realização dos experimentos, sendo um com 5 dimensões e outro com 7 dimensões. Os resultados gerados pelos algoritmos implementados foram comparados com os valores reais, o que validou a corretude dos algoritmos.

A partir dos resultados obtidos, observou-se uma redução no tempo total de execução para todos os cenários paralelos em todos os modelos propostos. Sobretudo, apesar dessa redução de tempo na execução do algoritmo sobre o caso com 5 dimensões, os valores de *speedup* e eficiência demonstram que os modelos desenvolvidos não utilizaram os recursos computacionais de maneira satisfatória, sendo recomendados preferencialmente para casos que possuem granulosidade grossa.

Sobre o caso com 7 dimensões, além da redução do tempo total de execução, os valores obtidos de *speedup* e eficiência foram satisfatórios, atingindo níveis próximos ao linear os Modelos 03 e 04 utilizando ambas as políticas de escalonamento, para a execução até 16 processadores. A partir de 16 processadores, apesar de atingir níveis aceitáveis, ocorreu uma queda na taxa de crescimento da curva de speedup. Tudo leva a crer que esse fato se deve à utilização do compartilhamento de recursos de hardware proporcionado pela arquitetura *hyper-thread*, a sobrecarga produzida pela infraestrutura da paralelização e pela sobrecarga do mecanismo de balanceamento de carga.

7.1 Trabalhos futuros

Como sugestões de trabalhos futuros, podemos citar:

- Aperfeiçoar os modelos desenvolvidos para obterem um melhor desempenho com problemas que possuam granulosidade fina;

- Desenvolver algoritmos paralelos que possam dividir o retângulo inicial independentemente da quantidade de processadores, permitindo assim que possam ser utilizados todos os processadores do *cluster*;
- Desenvolver e implementar modelos híbridos que utilizem os paradigmas de passagem de mensagem e memória compartilhada;
- Avaliar a execução dos modelos paralelos em clusters heterogêneos;
- Estudo e implementação dos modelos em outros problemas alvo.

Referências

BARNEY, B. Introduction to Parallel Computing. Acesso em: 28/11/2009, 2009.
Disponível em https://computing.llnl.gov/tutorials/parallel_comp/

CORAZZA, F. C.; OLIVEIRA, J. V.; CORAZZA, M. L. Application of a subdivision algorithm for solving nonlinear algebraic systems. *Acta Scientiarum*, v. 30, n. 1, 2008.

CORTES, O. A. C.; MENDEZ, O. R. S. Determinação do ranking de contingências em sistemas de energia elétrica utilizando mpi (message passing interface). *INFOCOMP Journal of Computer Science*, 1999.

COSTA, J. F. B. C. *Método dos elementos finitos: Análise de desempenho computacional paralelo.* Dissertação de Mestrado, Universidade de Aveiro, Aveiro, Portugal, 2010.

DALE, N.; LEWIS, J. *Computer science illuminated.* Jones and Bartlett Publishers, 2010.
Disponível em http://books.google.com.br/books?id=O3pHJF13Y8sC

DUNCAN, R. A survey of parallel computer architectures. *Computer*, v. 23, n. 2, p. 5 –16, 1990.

EAGER, D.; ZAHORJAN, J.; LAZOWSKA, E. Speedup versus efficiency in parallel systems. *Computers, IEEE Transactions on*, v. 38, n. 3, p. 408 –423, 1989.

EL-REWINI, H.; ABD-EL-BARR, M. *Advanced computer architecture and parallel processing.* Wiley series on parallel and distributed computing. Wiley, 2005.
Disponível em http://books.google.com.br/books?id=pDI9VxCBmncC

FLYNN, M. Very high-speed computing systems. *Proceedings of the IEEE*, v. 54, n. 12, p. 1901 – 1909, 1966.

FLYNN, M. J.; RUDD, K. W. Parallel architectures. *ACM Comput. Surv.*, v. 28, p. 67–70, 1996.
Disponível em http://doi.acm.org/10.1145/234313.234345

FOSTER, I. *Designing and building parallel programs.* Addison-Wesley, disponivel em: http://www.mcs.anl.gov/dbpp/, 1995.

FOUNTAIN, T. *Parallel computing: Principles and practice.* Cambridge University Press, 2006.
Disponível em http://books.google.com.br/books?id=d8sAGmfTx2gC

FOUNTAIN, T.; SHUTE, M. *Multiprocessor computer architectures.* North-Holland, 1990.
Disponível em http://books.google.com/books?id=4tUmAAAAMAAJ

FREISLEBEN, B.; KIELMANN, T. Approaches to support parallel programming on workstation clusters: A survey. *A Survey, Informatik Berichte, Fachgruppe Informatik, Universitat-GH Siegen*, v. 95, 1995.

GABRIEL, E.; FAGG, G. E.; BOSILCA, G.; ANGSKUN, T.; DONGARRA, J. J.; SQUYRES, J. M.; SAHAY, V.; KAMBADUR, P.; BARRETT, B.; LUMSDAINE, A.; CASTAIN, R. H.; DANIEL, D. J.; GRAHAM, R. L.; WOODALL, T. S. Open mpi: Goals, concept, and design of a next generation mpi implementation. In: *Proceedings, 11th European PVM/MPI Users Group Meeting*, 2004, p. 97–104.

GANGLIA Ganglia Monitoring System. Acesso em: 28/05/2010, 2010.
Disponível em http://ganglia.info

GOMES, J. L. *Paralelização de algoritmo de simulação de monte carlo para a adsorção em superfícies heterogêneas bidimensionais.* Dissertação de Mestrado, Universidade Estadual de Maringá, Maringá, PR, 2009.

GRAHAM, R. L.; SHIPMAN, G. M.; BARRETT, B. W.; CASTAIN, R. H.; BOSILCA, G.; LUMSDAINE, A. Open mpi: A high-performance, heterogeneous mpi. In: *Proceedings of the Fifth International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks*, 2006, p. 1–9.

GRAMA, A.; GUPTA, A.; KARYPIS, G.; KUMAR, V. *Introduction to parallel computing.* Addison-Wesley, 2003.

GRUBER, R.; KELLER, V. *HPC@Green it: Green high performance computing methods.* Springer, 2010.
Disponível em http://books.google.com.br/books?id=fDPdAYne6m8C

HOCKNEY, R.; JESSHOPE, C. *Parallel computers: architecture, programming and algorithms.* Adam Hilger, 1981.
Disponível em http://books.google.com/books?id=CP5gzotAXI4C

IGNÁCIO, A. A. V.; FERREIRA FILHO, V. J. M. MPI: uma ferramenta para implementação paralela. *Pesquisa Operacional*, v. 22, p. 105 – 116, 2002.
Disponível em http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0101-74382002000100007&nrm=iso

KRISHNAN, S.; GARIMELLA, S.; CHRYSLER, G.; MAHAJAN, R. Towards a thermal moore's law. *Advanced Packaging, IEEE Transactions on*, v. 30, n. 3, p. 462–474, 2007.

MACHADO, G. D. *Produção de biodiesel por esterificação em coluna de destilação reativa: modelagem matemática.* Dissertação de Mestrado, Universidade Estadual de Maringá, Maringá - PR, 2009.

MASSIE, M. L.; CHUN, B. N.; CULLER, D. E. The Ganglia Distributed Monitoring System: Design, Implementation, and Experience. *Parallel Computing*, v. 30, n. 7, 2004.

MODENESI, M. V. *Análise de agrupamentos FCM utilizando processamento paralelo.* Dissertação de Mestrado, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2008.

MORRISSON, R. S. Cluster Computing: Architectures, Operating Systems, Parallel Processing and Programming Languages. , n. 2.4, 2003.

MPI Message Passing Interface Forum. Acesso em: 17/11/2011, 2011.
Disponível em http://www.mpi-forum.org

MULLENIX, N.; POVITSKY, A. Parallel implementation of a tightly coupled ablation prediction code using mpi. In: *Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on*, 2009, p. 1 –4.

NAVAUX, P. O. A.; BARRETO, M. E.; ÁVILA, R. B.; DE OLIVEIRA, F. A. D. Execução de aplicações em ambientes concorrentes. *Escola Regional de Alto Desempenho*, v. 1, p. 179–193, 2001.

NOVAES, S. F.; GREGORES, E. M. *Da internet ao grid: a globalização do processamento.* São Paulo: Editora Unesp, 2007.
Disponível em http://books.google.com.br/books?id=nx3XIgizOHIC

OLIVEIRA, F. G. *Aplicações Autônomas para Computação em Larga Escala.* Dissertação de Mestrado, Universidade Federal Fluminense, Niterói, 2010.

OPENMPI Open MPI v1.3.4 documentation. Acesso em: 22/12/2010, 2010.
Disponível em http://www.open-mpi.org/doc/v1.3/

PACHECO, P. *An introduction to parallel programming.* Morgan Kaufmann. Elsevier Science, 2011.
Disponível em http://books.google.com.br/books?id=SEmfraJjvfwC

PALIN, M. F. *Tecnicas de Decomposição de Domínio em Computação Paralela para simulação de campos eletromagnéticos pelo Método dos Elementos Finitos.* Tese de Doutoramento, Escola Politécnica, Universidade de São Paulo, São Paulo, 2007.
Disponível em http://www.teses.usp.br/teses/disponiveis/3/3143/tde-08012008-122101

PINTO, R. J. *Aplicação de Processamento Paralelo ao Problema de Planejamento da Operação de Sistemas Hidrotérmicos Baseado em Cluster de Computadores.* Tese de Doutoramento, UFRJ/COPPE, Rio de Janeiro, 2011.

POLLONI, E.; FEDELI, R. *Introdução à ciência da computação.* THOMSON PIONEIRA, 2003.
Disponível em http://books.google.com.br/books?id=eTKb86SwCQYC

POLYMATH-SOFTWARE Numerical library problems involving simultaneous nonlinear equations. Disponível em: http://www.polymath-software.com/library/problemlist.shtml Acesso em: 20/01/2011, 2011.
Disponível em http://www.polymath-software.com/library/problemlist.shtml

RAUBER, T.; RÜNGER, G. *Parallel programming: For multicore and cluster systems.* Springer, 2010.
Disponível em http://books.google.com.br/books?id=wWogxOmA3wMC

SACERDOTI, F. D.; CHANDRA, S.; BHATIA, K. Grid systems deployment & management using rocks. In: *Proceedings of the 2004 IEEE International Conference on Cluster Computing*, Washington, DC, USA: IEEE Computer Society, 2004, p. 337–345.
Disponível em http://dl.acm.org/citation.cfm?id=1111682.1111735

SACERDOTI, F. D.; KATZ, M. J.; MASSIE, M. L.; CULLER, D. E. Wide area cluster monitoring with ganglia. *Cluster Computing, IEEE International Conference on*, v. 0, p. 289, 2003.

SHIGLEY, J.; MISCHKE, C. *Projeto de engenharia mecanica*. Bookman, 2005.
Disponível em `http://books.google.com.br/books?id=xZFaJHZfuTwC`

SHORE, J. E. Second thoughts on parallel processing. *Computers Electrical Engineering*, v. 1, n. 1, p. 95 – 109, 1973.
Disponível em `http://www.sciencedirect.com/science/article/pii/004579067390030X`

SILVA, J. M. N. *Estratégias de balanceamento de carga para um algoritmo branch-and-bound paralelo para executar em Grids computacionais*. Dissertação de Mestrado, Universidade Federal Fluminense, Niterói, RJ, 2006.

STALLINGS, W. *Computer organization and architecture: designing for performance*. Prentice Hall, 2009.
Disponível em `http://books.google.com.br/books?id=-7nM1DkWb1YC`

STERLING, T. *How to build a beowulf: a guide to the implementation and application of pc clusters*. Scientific and engineering computation. MIT Press, 1999.
Disponível em `http://books.google.com.br/books?id=xKM_GcCug78C`

STERLING, T. *Beouwlf cluster computing with linux*. MIT Press, 2001.

TANENBAUM, A. S. *Organização estruturada de computadores*. 2 ed. Prentice Hall, 2007.

TENENBAUM, A. A.; LANGSAM, Y.; AUGENSTEIN, M. J. *Estruturas de dados usando c*. Makron Books, 1995.

TOP500.ORG TOP500 Supercomputer Site. Acesso em: 16/11/2011, 2011.
Disponível em `http://i.top500.org/stats`

WILLEBEEK-LEMAIR, M.; REEVES, A. Strategies for dynamic load balancing on highly parallel computers. *Parallel and Distributed Systems, IEEE Transactions on*, v. 4, n. 9, p. 979 –993, 1993.

YERO, E. J. H. *Estudo sobre processamento maciçamente paralelo na internet*. Tese de Doutoramento, Universidade Estadual de Campinas, Campinas, SP, 2003.

Apêndice

A

Tempos de execução

Nr. de Nós	Sequencial	Modelo 01	Modelo 02	Modelo 03	Modelo 04
01	798	849	860	844	819
02	—	536	512	489	530
04	—	488	422	417	443
08	—	436	363	375	347
16	—	352	285	265	206
32	—	319	206	235	202

Tabela A.1: Tempos de execução do problema com 5 dimensões utilizando a política de escalonamento de nós adjacentes (em segundos)

Nr. de Nós	Sequencial	Modelo 01	Modelo 02	Modelo 03	Modelo 04
01	798	847	868	839	843
02	—	477	475	440	459
04	—	288	279	272	300
08	—	284	246	179	235
16	—	277	229	214	223
32	—	236	196	217	183

Tabela A.2: Tempos de execução do problema com 5 dimensões utilizando a política de escalonamento de nós equidistantes (em segundos)

Nr. de Nós	Sequencial	Modelo 01	Modelo 02	Modelo 03	Modelo 04
01	496	502	502	508	511
02	—	311	292	261	266
04	—	158	145	135	136
08	—	96	86	72	69
16	—	54	47	40	38
32	—	41	27	26	24

Tabela A.3: Tempos de execução do problema com 7 dimensões utilizando a política de escalonamento de nós adjacentes (em minutos)

Nr. de Nós	Sequencial	Modelo 01	Modelo 02	Modelo 03	Modelo 04
01	496	499	516	524	503
02	—	253	257	251	252
04	—	128	126	126	127
08	—	71	69	66	64
16	—	43	39	39	36
32	—	38	27	26	23

Tabela A.4: Tempos de execução do problema com 7 dimensões utilizando a política de escalonamento de nós equidistantes (em minutos)

Apêndice B

Speedups

Nr. de Nós	Modelo 01	Modelo 02	Modelo 03	Modelo 04
01	0,94	0,93	0,95	0,97
02	1,49	1,56	1,63	1,51
04	1,63	1,89	1,91	1,80
08	1,83	2,20	2,13	2,30
16	2,27	2,80	3,01	3,88
32	2,50	3,88	3,39	3,94

Tabela B.1: Speedups obtidos com problema de 5 dimensões utilizando a política de escalonamento de nós adjacentes

Nr. de Nós	Modelo 01	Modelo 02	Modelo 03	Modelo 04
01	0,94	0,92	0,95	0,95
02	1,67	1,68	1,81	1,74
04	2,77	2,86	2,94	2,66
08	2,81	3,24	4,46	3,39
16	2,88	3,49	3,73	3,58
32	3,38	4,08	3,68	4,36

Tabela B.2: Speedups obtidos com problema de 5 dimensões utilizando a política de escalonamento de nós equidistantes

Nr. de Nós	Modelo 01	Modelo 02	Modelo 03	Modelo 04
01	0,99	0,99	0,98	0,97
02	1,59	1,70	1,90	1,86
04	3,13	3,41	3,67	3,64
08	5,15	5,76	6,92	7,15
16	9,12	10,47	12,50	13,04
32	12,19	18,36	18,82	20,94

Tabela B.3: Speedups obtidos com problema de 7 dimensões utilizando a política de escalonamento de nós adjacentes

Nr. de Nós	Modelo 01	Modelo 02	Modelo 03	Modelo 04
01	0,99	0,96	0,95	0,98
02	1,96	1,93	1,97	1,97
04	3,87	3,94	3,94	3,89
08	6,95	7,22	7,47	7,74
16	11,44	12,71	12,82	13,77
32	12,93	18,59	19,06	21,55

Tabela B.4: Speedups obtidos com problema de 7 dimensões utilizando a política de escalonamento de nós equidistantes

Apêndice

C

Eficiências

Nr. de Nós	Modelo 01	Modelo 02	Modelo 03	Modelo 04
01	0,94	0,93	0,95	0,97
02	0,75	0,78	0,82	0,76
04	0,41	0,47	0,48	0,45
08	0,23	0,28	0,27	0,29
16	0,14	0,18	0,19	0,24
32	0,08	0,12	0,11	0,12

Tabela C.1: Eficiências obtidas com problema de 5 dimensões utilizando a política de escalonamento de nós adjacentes

Nr. de Nós	Modelo 01	Modelo 02	Modelo 03	Modelo 04
01	0,94	0,92	0,95	0,95
02	0,84	0,84	0,91	0,87
04	0,69	0,72	0,74	0,67
08	0,35	0,41	0,56	0,42
16	0,18	0,22	0,23	0,22
32	0,11	0,13	0,12	0,14

Tabela C.2: Eficiências obtidas com problema de 5 dimensões utilizando a política de escalonamento de nós equidistantes

Nr. de Nós	Modelo 01	Modelo 02	Modelo 03	Modelo 04
01	0,99	0,99	0,98	0,97
02	0,80	0,85	0,95	0,93
04	0,78	0,85	0,92	0,91
08	0,64	0,72	0,87	0,89
16	0,57	0,65	0,78	0,82
32	0,38	0,57	0,59	0,65

Tabela C.3: Eficiências obtidas com problema de 7 dimensões utilizando a política de escalonamento de nós adjacentes

Nr. de Nós	Modelo 01	Modelo 02	Modelo 03	Modelo 04
01	0,99	0,96	0,95	0,98
02	0,98	0,97	0,99	0,99
04	0,97	0,99	0,99	0,97
08	0,87	0,90	0,93	0,97
16	0,72	0,79	0,80	0,86
32	0,40	0,58	0,60	0,67

Tabela C.4: Eficiências obtidas com problema de 7 dimensões utilizando a política de escalonamento de nós equidistantes

Apêndice D

Soluções estimadas e obtidas

Soluções	Valores		
Solução estimada	ca=1,120613893	t1=90	tc=54,8512245
	b=0,5	y=0,31721198	
Solução obtida	ca=1,120613893	t1=90	tc=54,8512245
	b=0,5	y=0,31721199	

Tabela D.1: Relação das soluções estimadas e relações obtidas para o problema com 5 dimensões

Soluções	Valores			
Solução estimada	A=80,5340430737	B=6,9725465116	C=61,1190817583	D=7,2042004492
	E=0,5476701839	F=3,6532933311	G=0,0597027237	
Solução obtida 1	A=80,5340430752	B=6,9725465117	C=61,1190817594	D=7,2042004493
	E=0,5476701839	F=3,6532933311	G=0,0597027337	
Solução obtida 2	A=88,2940906995	B=7,6094182283	C=67,0424256548	D=7,8622514431
	E=12,4063758814	F=-16,937342646	G=0,0755409491	
Solução obtida 3	A=20,1461877628	B=28,5222960627	C=-33,2738512602	D=1,3487701304
	E=-27,1662884234	F=5,9736012722	G=-0,058428583	
Solução obtida 4	A=14,6776618415	B=20,8510314906	C=-24,3243042606	D=1,0391849472
	E=-47,4341416704	F=53,6556511124	G=-0,0584680274	

Tabela D.2: Relação das soluções estimadas e relações obtidas para o problema com 7 dimensões