Universidade Federal Fluminense

MAURICIO JOSÉ MACHADO GUEDES

PARALELIZAÇÃO DA RESOLUÇÃO DE EDPs PELO MÉTODO HOPSCOTCH UTILIZANDO REFINAMENTO ADAPTATIVO E BALANCEAMENTO DINÂMICO DE CARGA

Niterói, 2009

MAURICIO JOSÉ MACHADO GUEDES

PARALELIZAÇÃO DA RESOLUÇÃO DE EDPs PELO MÉTODO HOPSCOTCH UTILIZANDO REFINAMENTO ADAPTATIVO E BALANCEAMENTO DINÂMICO DE CARGA

Tese de Doutorado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Doutor.

Área de concentração: Modelagem Computacional

Orientador:

Mauricio Kischinhevsky

Universidade Federal Fluminense

NITERÓI

2009

PARALELIZAÇÃO DA RESOLUÇÃO DE EDPs PELO MÉTODO HOPSCOTCH UTILIZANDO REFINAMENTO ADAPTATIVO E BALANCEAMENTO DINÂMICO DE CARGA

Mauricio José Machado Guedes

Tese de Doutorado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Doutor.

Aprovada por:

_____

Prof. Mauricio Kischinhevsky, D.Sc / IC-UFF (Presidente)

_____

Profa. Regina Célia Paula Leal Toledo, D.Sc. / IC-UFF

_____

Prof. Luiz Nélio Henderson Guedes de Oliveira, D.Sc. / IPRJ-UERJ

_____

Capitão-de-Fragata Dilson Godoi Espenchitt, D.Sc. / CASNAV - Comando da Marinha

_____

Prof. José Henrique Carneiro de Araújo, D.Sc. / IC-UFF

Niterói, 8 de maio de 2009

Agradecimentos

Primeiro a meus pais, que tornaram esse trabalho possível ao me proporcionarem uma sólida base educacional e tranqüilidade para me formar Engenheiro Eletricista aqui na UFF, o principal degrau para a realização deste trabalho.

Agradeço ao meu orientador, Prof. Mauricio Kischinhevsky, pela orientação segura, bom senso e principalmente por acreditar em mim.

Agradeço àquelas pessoas que tornaram o desenvolvimento deste trabalho mais fácil, pela competência e amizade, Jacques, Carlos e Rafael.

Agradeço à Sandra, pela amizade e pelo trabalho que teve em formatar este trabalho.

À Marinha do Brasil e ao CASNAV, pela concessão da dedicação exclusiva ao Doutorado, o que possibilitou o término deste trabalho.

À Márcia, Débora e Danielle, por serem fonte de amenos pensamentos quando tudo parecia que não ia dar certo.

Resumo

Este trabalho propõe estudar a resolução numérica paralela eficiente de Equações Diferenciais Parciais (EDPs) discretizadas por diferenças finitas utilizando o método Hopscotch. Apesar das várias implementações do método já realizadas, algumas utilizando processamento paralelo, em nenhuma delas foi usado um refinamento adaptativo do domínio. Por este motivo, este trabalho desenvolve um resolutor de EDPs pelo método Hopscotch, discretizadas usando diferenças finitas, com refinamento adaptativo do domínio com o conseqüente balanceamento dinâmico da carga. O domínio é subdividido em subdomínios, e esses subdomínios são refinados e desrefinados durante a resolução. Os valores gatilho do processo de alteração do refinamento são introduzidos pelo usuário. O usuário introduz também os valores de gatilho de refinamento e desrefinamento. É proposto um método de particionamento que não despenda tempo procurando minimizar o perímetro dos subdomínios, buscando assim balancear a carga de modo rápido, sendo por isso indicado para uso em computadores paralelos e problemas com freqüentes mudanças das condições de refinamento. Testes são realizados com os três tipos de EDP, parabólica, elíptica e hiperbólica. Os testes iniciam com a versão seqüencial, continuam com a versão paralela e culminam com a versão paralela com balanceamento dinâmico de carga. Métricas são definidas em cada um desses testes para medir o desempenho do resolutor desenvolvido.

Abstract

In this work efficient parallel numerical solution of finite-difference discretizations of partial differential equations (PDEs) with the Hopscotch method is studied. A new technique for adaptive mesh refinement and dynamic load balance is proposed, and tests are presented on a distributed computing environment. Despite the many implementations already presented for the method, some of which employ parallel processing, none of them has used adaptive mesh refinement with load balance. Thus, in this work a package is built to solve PDEs discretized by finite differences, with adaptive mesh refinement and load balance. The domain is divided in subdomains, that are refined and coarsened during the execution. The process of subdomain refinement change can be chosen by the user, which provides the trigger values. A partition method that does not seek to minimize the communications is proposed. This method achieves the balance quickly, resulting in an easy-to-use technique for parallel computers and problems that frequently change the refinement. Tests are performed using three PDE types, elliptic, parabolic and hyperbolic. The tests start with a sequential version, proceed with the parallel version and end-up with the load balanced parallel version. Metrics are defined to evaluate the package performance.

Palavras-chave

1. Métodos Numéricos
2. Solução numérica de EDPs
3. Método das Diferenças Finitas
4. Refinamento Adaptativo de Malhas
5. Método Hopscotch
6. Processamento Paralelo
7. Balanceamento de carga

GLOSSÁRIO

ACO:	Ant Colony Optimization
AMR:	(Adaptative Mesh Refinement) – Refinamento Adaptativo de Malhas
DM:	Distributed Memory, memória distribuída
IMD:	Índice Médio de Desbalanceamento
KL:	Kernighan-Lin
LAM:	Local Area Multicomputer
MIMD:	Multiple Instruction, Multiple Data
MPI:	Message Passing Interface, protocolo de passagem de dados
OEH:	Odd-even Hopscotch, hopscotch par-ímpar
PC:	Personal Computer, computador pessoal
RCB:	Recursive Coordinated Bisection
RB:	Recursive Bisection
SA:	Simulated Annealing
SFC:	Space-filling Curve
SIMD:	Single Instruction, Multiple Data
SM:	Shared Memory, memória compartilhada
2D:	Bidimensional
u:	Variável dependente
a:	Coeficiente de difusão
c:	Velocidade da onda
R:	Região do domínio
∂R:	Contorno do domínio
n:	Quantidade de pontos em uma dimensão do domínio
N:	Quantidade de pontos do domínio = n x n
m:	Quantidade de pontos em uma dimensão de um subdomínio
p:	Quantidade de processadores
q:	Quantidade de processos

SUMÁRIO

CAPÍTULO 1 INTRODUÇÃO 1
1.1 - MOTIVAÇÃO 1
1.2 - PARTICIONAMENTO DO DOMÍNIO 2
CAPÍTULO 2 EQUAÇÕES DIFERENCIAIS PARCIAIS E O MÉTODO HOPSCOTCH 3
2.1 - EQUAÇÕES DIFERENCIAIS PARCIAIS (EDPs) 3
2.2 - CONDIÇÕES DE CONTORNO 4
2.3 - MÉTODOS NUMÉRICOS 5
2.4 - MALHAS DE DISCRETIZAÇÃO 6
2.5 - MÉTRICAS 9
2.5.1 - Normas de Erro 9
2.5.2 - Tempo 9
2.6 - EQUAÇÕES DE DIFERENÇAS FINITAS 9
2.6.1 - Discretização das equações usadas como exemplo 12
2.6.1.1 - Equação parabólica 12
2.6.1.1.1 - Formulação Explícita 12
2.6.1.1.2 - Formulação Implícita 12
2.6.1.2 - Equação elíptica 13
2.6.1.2.1 - Formulação Explícita 13
2.6.1.2.2 - Formulação Implícita 14
2.6.1.3 - Equação hiperbólica 14
2.6.1.3.1 - Formulação explícita 14
2.6.1.3.2 - Formulação implícita 15
2.7 - APLICAÇÃO DO MÉTODO ODD-EVEN HOPSCOTH (OEH) 15
2.7.1 - Consistência, Estabilidade e Convergência do Método 17
2.8 - CASOS TESTE 18
2.8.1 - Equação Parabólica 18
2.8.2 - Equação Elíptica 21
2.8.3 - Equação Hiperbólica 23
CAPÍTULO 3 APLICAÇÕES PARALELAS 27
3.1 - MÉTRICAS 28
3.1.1 - “Speedup” 28
3.1.2 - Eficiência 28
3.1.3 - Escalabilidade 29
3.2 - MONITORAMENTO 29
3.3 - CONCEITOS DE PARALELIZAÇÃO 29
3.3.1 - Paradigmas 29
3.3.2 - Granularidade 29
3.4 - SUPERCOMPUTADORES 30
3.5 - TOPOLOGIA DA REDE DE PROCESSADORES 31
3.6 - PARALELIZAÇÃO DO MÉTODO HOPSCOTCH 33
3.6.1 - Implementação 33
3.6.2 - Tipos de Subdomínio com Relação ao Contorno 34
3.6.3 - Comunicação entre os Processadores 36
3.7 - O LAM/MPI 38
3.7.1 - Rotinas Básicas 38
3.7.2 - Rotinas de Buferização 38

3.7.3 - Rotinas de Broadcast e Sincronização 39
3.8 - CASOS TESTE 39
3.8.1 - Equação Parabólica 39
3.8.2 - Equação Elíptica 42
3.8.3 - Equação Hiperbólica 44
CAPÍTULO 4 O REFINAMENTO ADAPTATIVO 47
4.1 - TIPOS DE REFINAMENTO 47
4.1.1 - Refinamento-*h* 47
4.1.2 - Refinamento-*p* 48
4.1.3 - Refinamento-*r* 48
4.2 - O REFINAMENTO-*h* UTILIZADO 48
4.3 - COMUNICAÇÃO ENTRE SUBDOMÍNIOS COM REFINAMENTOS DIFERENTES 52
4.4 - ROTINAS USADAS PARA O REFINAMENTO E O DESREFINAMENTO 53
4.5 - CONSIDERAÇÕES SOBRE A CONVERGÊNCIA 53
CAPÍTULO 5 PARTICIONAMENTO DO DOMÍNIO E BALANCEAMENTO DE CARGA 54
5.1 - PARTICIONAMENTO DE MALHAS IRREGULARES 54
5.1.1 - Métodos Geométricos 54
5.1.1.1 - Bisecção Recursiva 55
5.1.1.2 - “Space-Filling Curve” (SFC) 56
5.1.2 - Método Multinível 56
5.1.3 - Heurísticas 57
5.1.3.1 - Kernighan-Lin (KL) 58
5.1.3.2 - Método Fidducia-Mattheyses 58
5.1.3.3 - “Helpful-sets” 58
5.1.4 - Método Espectral 58
5.1.5 - Metaheurísticas 59
5.1.5.1 - Colônia de formigas (ACO – Ant Colony Optimization)) 59
5.1.5.2 - Recozimento simulado (SA -Simulated annealing) 59
5.1.5.3 - Algoritmos genéticos 60
5.1.6 - Pacotes Disponíveis 60
5.2 - PARTICIONAMENTO DE MALHAS REGULARES 62
5.3 - TIPOS DE BALANCEAMENTO 63
5.4 - IMPLEMENTAÇÃO DO REFINAMENTO ADAPTATIVO 65
5.4.1 - O Programa 65
5.4.2 - Balanceamento de Carga 65
5.5 - MÉTRICAS 66
5.5.1 - Índice de desbalanceamento (ID) 66
5.5.2 - Índice médio de desbalanceamento (IMD) 66
5.5.3 - Ganho obtido com o balanceamento 66
5.5.4 - O “speedup” e a eficiência da versão com balanceamento de carga 67
CAPÍTULO 6 BALANCEAMENTO PROPOSTO 68
6.1 - O PARTICIONADOR 68
6.2 - EXEMPLO 70
6.2.1 - Subdivisão do Domínio 70
6.2.2 - Tráfego de Carga 71
6.2.3 - Subdomínios Pertencentes a Cada Processador 72
6.3 - TRÁFEGO DAS FRANJAS 73
6.4 - IMPLEMENTAÇÃO 74
6.5 - O PROGRAMA 74

6.6 - DESEMPENHO DO PARTICIONADOR *PREPARA* E SUA COMPARAÇÃO COM OUTROS PROGRAMAS 76
6.6.1 - Desempenho do particionador *Prepara* 77
6.6.2 - Comparação com outros Particionadores 78
6.7 - DESCRIÇÃO DAS ROTINAS QUE IMPLEMENTAM O BALANCEAMENTO DE CARGA 79
6.8 - Casos Teste 85
6.8.1 - Equação Parabólica 86
6.8.2 - Equação Elíptica 91
6.8.3 - Equação Hiperbólica 95
CAPÍTULO 7 CONCLUSÕES 101
7.1 - Trabalhos Futuros 102

LISTA DE FIGURAS

Figura 1 - Discretização de um domínio contínuo, formando uma malha [32].....6
Figura 2 - Exemplo de malha estruturada.....7
Figura 3 - Exemplo de malha elíptica [13].....7
Figura 4 - Exemplo do grafo *4elt*, com elementos triangulares [95].....8
Figura 5 - Posicionamento das variáveis.....8
Figura 6 - Molécula computacional.....11
Figura 7 - Aplicação do OEH.....16
Figura 8 - Consistência, estabilidade e convergência para problemas lineares.....18
Figura 9 - Discretização.....20
Figura 10 - Curvas de nível da condição inicial.....20
Figura 11 - Soluções da equação.....20
Figura 12 - Gráfico da solução da equação.....22
Figura 13 - Gráfico e curvas de nível da solução da equação.....22
Figura 14 - Visualização da equação da onda.....24
Figura 15 - Vistas de topo da evolução da onda conforme as iterações avançam, podendo-se observar a presença de difusão numérica.....25
Figura 16 - Algumas topologias usadas em um computador de memória compartilhada.....31
Figura 17 - Topologia em hipercubos.....32
Figura 18 - Exemplo de uma "fat-tree" quaternária.....33
Figura 19 - Divisão do domínio.....34
Figura 20 - Exemplo de particionamento.....35
Figura 21 - Tipos de subdomínio.....35
Figura 22 - Tipos de subdomínio em um domínio particionado em 36 subdomínios.....36
Figura 23 - Posição relativa dos subdomínios.....36
Figura 24 - Subdomínio com as franjas enviadas. Os pontos pretos são pontos do subdomínio, e os pontos brancos são pontos das franjas.....37
Figura 25 - Subdomínio com as franjas recebidas. Os pontos pretos são pontos do subdomínio, e os pontos brancos são pontos das franjas.....37
Figura 26 - "Speedup" da resolução da equação parabólica.....40
Figura 27 - Eficiência da resolução da equação parabólica.....41
Figura 28 - Escalabilidade da resolução da equação parabólica.....42
Figura 29 - "Speedup" da resolução da equação elíptica.....43
Figura 30 - Eficiência da resolução da equação elíptica.....43
Figura 31 - Escalabilidade da resolução da equação elíptica.....44
Figura 32 - "Speedup" da resolução da equação da onda.....45
Figura 33 - Eficiência da resolução da equação da onda.....46
Figura 34 - Escalabilidade da resolução da equação da onda.....46
Figura 35 - Malha adaptativa de Berger e Oliger [9].....48
Figura 36 - Particionamento e refinamento de um domínio com refinamento inicial 12 x 12.....49
Figura 37 - Exemplo esquemático de uma spline cúbica construída nos intervalos $[x_{j-1},x_j]$, cada função S_j sendo uma cúbica [43].....50
Figura 38 - Um domínio dividido em 5x5 células, representado pelas linhas grossas. Um fator de refinamento 2 passa o domínio para 10x10 células. As células adicionais foram criadas pela adição das linhas finas.....51
Figura 39 - Correspondência entre os pontos da malhas refinada e da malha origem.....51
Figura 40 - Processo de refinamento de um subdomínio – pontos auxiliares.....52

Figura 41 - Processo de refinamento de um subdomínio – pontos refinados 52
Figura 42 - Comunicação entre franjas com diferentes níveis de refinamento 53
Figura 43 - Planos de corte (esq) a árvore de corte associada para bisecção recursiva. Pontos são os objetos a serem balanceados; os cortes são mostrados com linhas e nós na árvore. 55
Figura 44 - Particionamento SFC [67]. 56
Figura 45 - Método multinível [78] 57
Figura 46 - Cálculo da quantidade de colunas que ficam no processador 69
Figura 47 - Exemplo de balanceamento de carga 70
Figura 48 - Exemplo de subdivisão do domínio 71
Figura 49 - Exemplo de refinamento 71
Figura 50 - Tráfego de carga 72
Figura 51 - Subdomínios pertencentes a cada processador 72
Figura 52 - Exemplo do tráfego de comunicação das franjas 73
Figura 53 - Fluxograma de funcionamento do programa. A linha pontilhada envolve os procedimentos do laço de resolução do método numérico 75
Figura 54 - Pseudo-código da implementação do método numérico 76
Figura 55 - Desempenho do Particionador com relação à quantidade de subdomínios 77
Figura 56 - Desempenho do particionador com relação à quantidade de nós de um subdomínio 78
Figura 57 - Despacho das franjas 81
Figura 58 - Índices usados 82
Figura 59 - Tráfego de dados 82
Figura 60 - Armazenamento no processador *p* 83
Figura 61 - Legenda para as figuras 62, 67 e 72 85
Figura 62 - Evolução do refinamento durante a resolução 87
Figura 63 - “Speedup” do programa com balanceamento de carga 89
Figura 64 - Eficiência do programa com balanceamento de carga 89
Figura 65 - Ganho do programa com balanceamento de carga quando comparado com a versão sem balanceamento de carga 90
Figura 66 - Índice de desbalanceamento 91
Figura 67 - Evolução do grau de refinamento durante a resolução da equação 92
Figura 68 - “Speedup” do programa com balanceamento de carga 93
Figura 69 - Eficiência do programa com balanceamento de carga 94
Figura 70 - Ganho do programa com balanceamento de carga quando comparado com a versão sem balanceamento de carga 94
Figura 71 - Índice médio de desbalanceamento 95
Figura 72 - Evolução do refinamento adaptativo 96
Figura 73 - “Speedup” do programa com balanceamento de carga 98
Figura 74 - Eficiência do programa com balanceamento de carga 98
Figura 75 - Ganho do programa com balanceamento de carga quando comparado com a versão sem balanceamento de carga 99
Figura 76 - Índice médio de desbalanceamento 100

LISTA DE TABELAS

Tabela 1 - Tipos de EDPs 4
Tabela 2 - Variação de *K* 21
Tabela 3 - Resultados da precisão e tempo, com *K*=1,0 21
Tabela 4 - Variação de *K* 23
Tabela 5 - Resultados da precisão e tempo, com K=1,5 23
Tabela 6 - Variação de *K* 25
Tabela 7 - Resultados da precisão e tempo, com *K*=0,15 26
Tabela 8 - Métricas da paralelização do Hopscotch 39
Tabela 9 - Métricas da paralelização do Hopscotch 42
Tabela 10 - Métricas da paralelização do Hopscotch 44
Tabela 11 - Critérios de Classificação 63
Tabela 12 - Comparação entre os particionadores para $\approx$ 15.600 nós 79
Tabela 13 - Matriz *envia* do exemplo 6.2 79
Tabela 14 - Erro e tempo de execução em função da freqüência de verificação 86
Tabela 15 - “Speedup” e eficiência do balanceamento de carga 88
Tabela 16 - Ganho para o balanceamento de carga 88
Tabela 17 - “Speedup” e eficiência do balanceamento de carga 92
Tabela 18 - Ganho para o balanceamento de carga 93
Tabela 19 - “Speedup” e eficiência do balanceamento de carga 97
Tabela 20 - Ganho para o balanceamento de carga 97

CAPÍTULO 1

INTRODUÇÃO

Equações Diferenciais Parciais (EDPs) são utilizadas para modelar uma vasta gama de eventos da natureza, nas mais diversas áreas de aplicação. As EDPs são definidas em regiões contínuas, o que implica sua validade para infinitos pontos. Deste modo, não podemos tratá-las computacionalmente antes de realizar o que se chama de discretização do domínio, com a criação de uma malha de pontos, onde a equação será calculada. As aplicações das EDPs são diversificadas, e dentre outras podemos citar a previsão meteorológica, o estudo da poluição nos oceanos, rios e na atmosfera (por ex, efeitos na camada de ozônio), aplicações na termodinâmica, eletromagnetismo, aerodinâmica e na prospecção e extração de petróleo. Em situações reais, a malha resultante da discretização do domínio contém muitos pontos, em número que pode ser da ordem de milhões e, além disso, para estudar efeitos de longo prazo, as EDPs devem descrever longos intervalos de tempo. Tais modelos requerem uma quantidade de memória e uma velocidade de processamento que recomendam a utilização de modernos supercomputadores paralelos.

Assim, as pesquisas no campo de métodos numéricos eficientes de resolução das EDPs e da ciência da computação aparecem como ferramentas indispensáveis no tratamento desses problemas.

1.1 - MOTIVAÇÃO

Foi observado do estudo da literatura pertinente que, apesar das várias implementações paralelas do Hopscotch realizadas, nenhuma delas fazia uso de refinamento adaptativo da malha (AMR – Adaptative Mesh Refinement) por blocos em um ambiente de processamento paralelo com balanceamento de carga para otimizar o desempenho da aplicação. O AMR por blocos foi primeiro desenvolvido por Berger e Oliger [9] para equações hiperbólicas, e estendido por Berger e Colella para problemas de hidrodinâmica de choques, por Almgren et alli para as equações de Navier-Stokes e por Martin e Cartwright [58] para resolver problemas elípticos. O trabalho desenvolvido aqui faz uma abordagem do AMR em blocos de uma maneira um pouco diferente, para melhor adaptá-lo ao método Hopscotch e ao balanceamento dinâmico de carga, isto é, este trabalho foca o emprego do método Hopscotch em um ambiente paralelo com refinamento adaptativo e dinâmico da malha utilizando balanceamento de carga, em problemas ocasionados por fenômenos difusivos e convectivos, que afetam

determinadas regiões do domínio sendo estudado. Exemplos típicos destes fenômenos são o estudo da poluição em rios e mares causada por despejos de esgoto ou químicos ou acidentes envolvendo produtos químicos no solo, rios ou oceanos, estudos de meteorologia, indústria petrolífera, propagação eletromagnética e acústica, dentre outros.

1.2 - PARTICIONAMENTO DO DOMÍNIO

Para que seja implementado o processamento paralelo, o domínio precisa ser dividido em subdomínios, idealmente em tantos subdomínios quantos sejam os processadores disponíveis. Para isso, é necessário um algoritmo que meça a carga total e a carga em cada processador para detectar quando um desbalanceamento de carga acontece e para calcular a quantidade de carga que precisa migrar de um processador para outro. Apesar dos diversos particionadores disponíveis existentes (Chaco, Metis, Scotch, entre outros), eles não são focados no particionamento de malhas regulares para utilização do método Hopscotch, apesar de serem capazes de lidar com o problema. Por isso foi decidido construir um balanceador dinâmico de carga, de funcionamento simples e se aproveitando da geometria de uma malha estruturada, para aplicações resultantes de discretizações por diferenças finitas da resolução pelo método Hopscotch.

CAPÍTULO 2

EQUAÇÕES DIFERENCIAIS PARCIAIS E O MÉTODO HOPSCOTCH

O método Hopscotch foi proposto por Gordon em 1965 e detalhadamente analisado, a partir de 1970 por uma série de artigos de Gourlay [35, 36, 37] e seus colaboradores. É uma técnica de cálculo implícito de diferenças finitas, que alterna o emprego das formulações explícita e implícita, pertencente ao grupo dos "splitting methods".

O método consiste em aplicar as equações explícita e implícita para calcular o valor em cada ponto do domínio, em modo alternado, daí o nome *Hopscotch*. Os valores de aproximação nos pontos a terem seu valor calculado pela equação explícita serão sempre obtidos primeiro. Assim, quando os pontos a terem seu valor calculado pelo método implícito forem resolvidos, todos os pontos adjacentes necessários para o cálculo já terão valor conhecido. Deste modo, não é necessário resolver um sistema de equações. Uma etapa adicional simetriza o processo alternando os conjuntos de pontos.

O método Hopscotch constitui um exemplo de método implícito em que os valores das aproximações numéricas que compõem a estimativa em um novo instante de tempo são calculadas de modo explícito, vale dizer, sem requerer a resolução de sistemas de equações lineares.

2.1 - EQUAÇÕES DIFERENCIAIS PARCIAIS (EDPs)

Os problemas de engenharia normalmente envolvem a resolução de modelos matemáticos dos fenômenos físicos e uma grande parte desses modelos é definida por equações diferenciais em que são conhecidas as condições de contorno e/ou as condições iniciais. As equações diferenciais são obtidas através da aplicação das leis e princípios fundamentais da natureza para um sistema, representando o balanço de massa, força e energia. Em um dado problema de engenharia, existe a divisão dos parâmetros em dois grupos que deverão influenciar na forma como o sistema se comporta. Estes parâmetros incluem as propriedades tais como o módulo de elasticidade, condutividade hidráulica e viscosidade [32].

Por outro lado, existem os parâmetros que produzem uma perturbação em um sistema. Exemplos destes parâmetros incluem as forças externas, momentos, diferença de carga hidráulica ao longo de um meio poroso e a diferença no fluxo do fluido.

Uma EDP pode ser classificada sob diferentes aspectos. A ordem de uma equação diferencial parcial é a maior ordem de derivada parcial presente na equação. Convenciona-se

definir a dimensionalidade da EDP como o número de direções espaciais que nela aparecem. Por exemplo, problemas evolutivos bidimensionais possuem, além de uma coordenada temporal, duas coordenadas espaciais.

Considere-se a equação diferencial de segunda ordem em duas variáveis x e y, que não necessariamente representam coordenadas espaciais:

$$a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2} + d\frac{\partial u}{\partial x} + e\frac{\partial u}{\partial y} + fu = g$$

Quando a, b, c, d, e, f e g forem constantes ou funções de x e y apenas, a equação acima é dita linear. Caso contrário, é considerada não-linear. Equações não-lineares nas quais a derivada de maior ordem aparece linearmente são denominadas quase-lineares.

Em função dos valores de a, b e c, podemos classificar a equação acima em:

Tabela 1 - Tipos de EDPs

Discriminante	Tipo	Exemplo	Equação
$ac - b^2 > 0$	elíptica	Eq. de Laplace	$u_{xx} + u_{yy} = 0$
$ac - b^2 < 0$	hiperbólica	Eq. da onda	$c^2(u_{xx} + u_{yy}) = u_{tt}$
$ac - b^2 = 0$	parabólica	Eq. do calor	$u_{xx} + u_{yy} = u_t$

As EDPs elípticas estão freqüentemente associadas aos problemas de equilíbrio, que são aqueles nos quais a propriedade de interesse não se altera com o passar do tempo. As EDPs parabólicas estão associadas a problemas evolutivos, de difusão com presença do fenômeno dissipativo. As EDPs hiperbólicas estão associadas a problemas evolutivos de vibração ou de convecção.

2.2 - CONDIÇÕES DE CONTORNO

Uma equação diferencial pode ter solução única quando são especificadas condições sobre a variável dependente na fronteira ∂R da região R, em que se quer resolver o problema.

De forma resumida, têm-se os seguintes tipos básicos de condições de contorno [101]:

a) $u = g$ (condição de Dirichlet, valor estipulado no contorno)

b) $\frac{\partial u}{\partial n} = g$ (condição de Neumann, fluxo estipulado no contorno)

c) $au + b\frac{\partial u}{\partial n} = g$ (condição de Robin, combinação das anteriores)

onde u é a solução, g é uma função definida em ∂R, $\partial u / \partial n$ é a derivada normal no contorno, a e b são constantes diferentes de zero.

2.3 - MÉTODOS NUMÉRICOS

Existem várias limitações à utilização dos métodos analíticos, tendo em vista a grande variabilidade dos parâmetros, das propriedades dos materiais, das condições de contorno e de condições iniciais diversificadas. Métodos numéricos são usados quando não é possível obter uma solução analítica, ou a forma dela é tão complicada que seu uso não é prático. Por sua vez, métodos numéricos apropriados permitem a solução das equações diferenciais em qualquer distribuição espacial, com propriedades dos materiais bastante variáveis, em qualquer geometria e variando com o tempo, ou seja, em condições evolutivas.

Nas modelagens numéricas a solução obtida é definida por um procedimento aproximado, que em muitas situações se aproxima de forma significativa do resultado real esperado [74].

Para uma EDP ser resolvida numericamente é necessário discretizar a equação, isto é, trazê-la para um subspaço finito de pontos. Isto pode feito por, entre outras, uma das três técnicas de discretização [32, 60] a seguir, que reduzem o problema à solução de equações algébricas:

a) Método dos elementos finitos
b) Método das diferenças finitas
c) Método dos volumes finitos

Uma das principais técnicas discretização utilizadas é o Método das Diferenças Finitas. No método das diferenças finitas, as funções são representadas por seus valores nos pontos de uma malha e as derivadas são aproximadas por diferenças destes valores. Esse método tem um importante papel na análise numérica, pois é uma das maneiras mais simples de se discretizar uma EDP, sendo considerado o mais simples de aprender e de usar.

O erro entre a solução aproximada e a solução exata é determinado pelo erro que é cometido quando se transforma o operador diferencial em um operador de diferenças. Esse erro é chamado de erro de discretização, ou erro de truncamento, já que esse termo reflete o fato de que o operador de diferenças ser uma parte finita de uma série de Taylor e, portanto, truncado em sua capacidade de representação do operador diferencial. Ademais, os erros decorrentes das operações aritméticas em uma máquina com capacidade de representação finita, os erros de arredondamento, são acrescidos aos de truncamento.

A resolução de uma EDP evolutiva usando o método das diferenças finitas tem duas formulações principais: a explícita e a implícita. Formulações explícitas calculam o estado adiante do sistema a partir do estado atual do sistema, enquanto a formulação implícita encontra o estado adiante resolvendo um sistema de equações envolvendo os estados adiante e atual do sistema. Matematicamente, se *Y(t)* é o estado atual do sistema e *Y(t+? t)* é o estado adiante no tempo, então, pela formulação explícita, é necessário resolver $Y(t + ?t) = F(Y(t))$, enquanto que, pela formulação implícita, é necessário resolver $G(Y(t), Y(t + ?t)) = 0$ para encontrar $Y(t + ?t)$ [32, 34].

As formulações implícitas requerem mais esforço computacional, além de serem mais difíceis de implementar, mas são freqüentemente incondicionalmente estáveis .

Para se utilizarem técnicas numéricas de solução, não é possível tratar a região *R* como contínua, pois a solução é calculada somente para um conjunto discreto de pontos do domínio, dando origem a uma malha de pontos, nos quais o valor da solução aproximada da EDP será calculado.

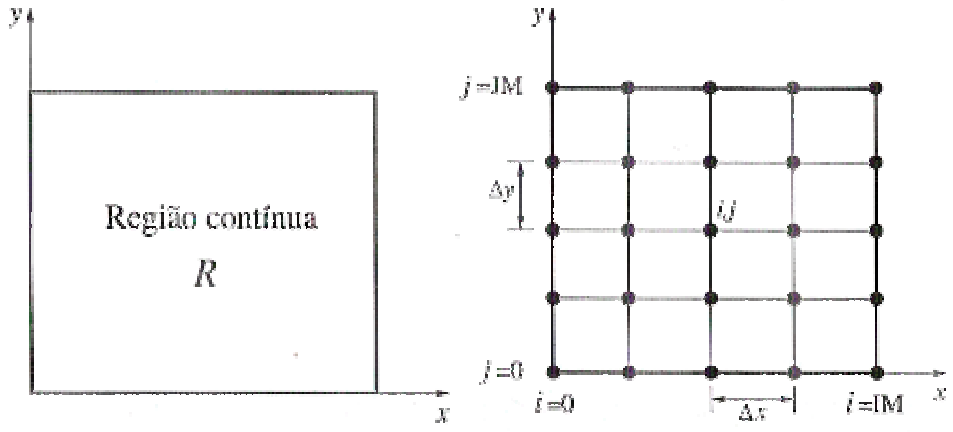


Figura 1 - Discretização de um domínio contínuo, formando uma malha [32]

Os domínios considerados neste trabalho terão, no contexto da Figura 1, IM+1 = JM+1 = *n*, isto é, serão domínios quadrados. A quantidade de pontos em um domínio será *n* x *n* = n^2 = N.

2.4 - MALHAS DE DISCRETIZAÇÃO

O processo de discretização conduz à criação de uma malha de pontos onde o valor de interesse será calculado. Uma malha é uma estrutura geométrica formada por nós e arcos ou arestas.

Normalmente refinar uniformemente malhas regulares demanda mais recursos computacionais, pois é necesssário um grande número de pontos para fornecer uma precisão

aceitável em todo o domínio. Técnicas que refinam, isto é, aumentam a quantidade de pontos da malha, somente nas regiões onde uma maior precisão é requerida, exigem menos recursos computacionais. Assim, o refinamento adaptativo, isto é, somente onde ele é necessário, é uma solução para tratar problemas com geometria complexa, presença de fortes gradientes e outras particularidades do problema. Para problemas que evoluem no tempo, o refinamento deve ser dinâmico para acompanhar as necessidades de precisão na solução, aumentando ou diminuindo o refinamento conforme a necessidade.

Existem malhas regulares, irregulares, estruturadas e não-estruturadas [46]. As malhas regulares são basicamente de dois tipos, as malhas baseadas no desenvolvimento planar de um retângulo (malhas retangulares) e as malhas baseadas no desenvolvimento planar de um triângulo (malhas triangulares). Todas as outras malhas regulares planares são derivações destas [46].

As malhas estruturadas são aquelas que têm todos os seus nós com a mesma quantidade de vizinhos [46].

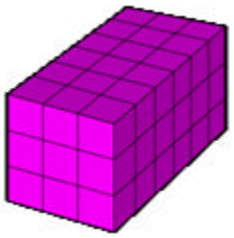


Figura 2 - Exemplo de malha estruturada

A seguir estão apresentadas outras malhas que comumente aparecem em discretizações do domínio. Essas malhas normalmente estão relacionadas ao método de elementos finitos.

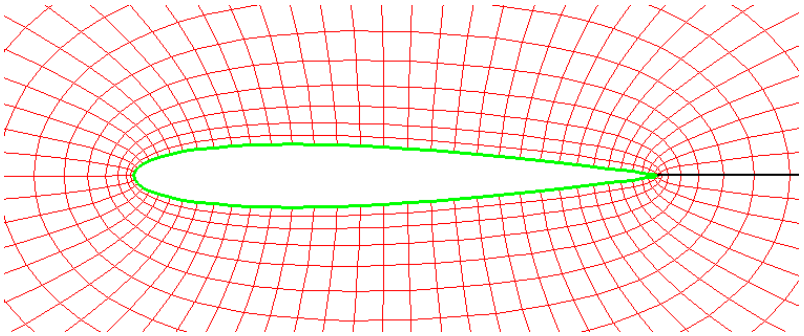


Figura 3 - Exemplo de malha elíptica [13]

Malhas irregulares são mais difíceis de particionar do que malhas regulares. São encontradas tipicamente na resolução de problemas utilizando elementos e volumes finitos. Malhas regulares são muito empregadas na resolução de problemas por diferenças finitas.

A malha considerada neste trabalho é a malha regular 2D. Ela é inicialmente estruturada, mas devido ao refinamento desigual do domínio ela perde a estruturação, voltando a tornar-se estruturada apenas quando o refinamento torna a ficar uniforme.

Uma malha pode ser representada por um grafo, formado por arestas e nós. Existem disponíveis arquivos de entrada de grafos com mais de um milhão de nós, usados na comparação de desempenho entre os vários pacotes de particionamento disponíveis e por novas propostas de particionadores. Um exemplo destas malhas é dado no arquivo *4elt*, que contém uma malha usada para estudar o escoamento em torno de uma asa de quatro elementos, como mostrado na figura 4 a seguir.

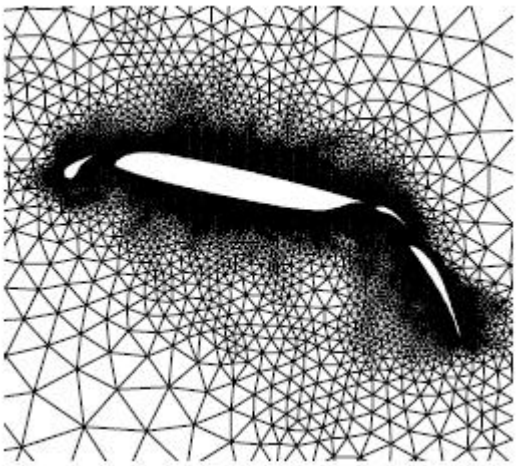


Figura 4 - Exemplo do grafo *4elt*, com elementos triangulares [95]

Já nas malhas "staggered" [85]**,** em vez de posicionar todas as variáveis em um ponto da malha, diferentes variáveis são colocadas em posições diferentes na malha, que são deslocadas meia célula. A figura 5 a seguir mostra o posicionamento das variáveis da equação de continuidade

$$\frac{\partial r}{\partial t} = -\nabla \cdot \left(r \vec{u} \right),$$

considerada em duas dimensões. A variável r está centrada no interior da célula, e ru_x e ru_y estão centrados no centro das arestas.

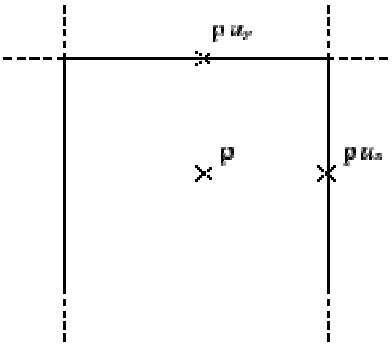


Figura 5 - Posicionamento das variáveis.

2.5 - MÉTRICAS

A seguir estão definidas as métricas usadas para medir a precisão da resolução numérica e o tempo gasto para chegar à solução.

2.5.1 - Normas de Erro

O erro é a diferença entre o resultado, em cada ponto do domínio discretizado, conseguido pelo método numérico, *u*, e o resultado exato da EDP, $\hat{u}$.

$$e_i = u_i - \hat{u}_i$$

Existem várias maneiras de se calcular o erro, e a escolha de qual método é o melhor vai depender do problema sendo resolvido. Dentre esses diversos métodos, os mais comuns são [23]:

Norma L_1 : É a média aritmética percentual do erro.

$$\|e\|_{L_1} = \frac{1}{mu_{max}} \sum_{i=1}^{m} |e_i|$$

Norma L_2: É a média quadrática do erro, também chamada norma RMS.

$$\|e\|_{L_2} = \left[\frac{1}{m} \left(\frac{1}{u_{\max}^2} \sum_{i=1}^{m} |e_i| \right) \right]^{1/2}$$

Norma L_∞ : Também chamada de norma infinita, é o maior erro percentual da solução.

$$\|e\|_{L_\infty} = \frac{1}{|u_{\max}|} \max |e_i|$$

2.5.2 - Tempo

O tempo de execução ("elapsed time") que a aplicação requer para resolver a EDP será calculado, para cada processo, pela função *MPI_Wtime*, chamada uma vez imediatamente após a função *MPI_Init* e outra vez imediatamente antes da função *MPI_Finalize*. O MPI é apresentado na seção 3.7.

2.6 - EQUAÇÕES DE DIFERENÇAS FINITAS

Realizar uma aproximação numérica de uma EDP significa obter valores em uma quantidade limitada de pontos do domínio, que são relacionados por expressões algébricas representando aproximações do operador diferencial.

Os métodos de diferenças finitas [60] consistem em substituir as derivadas parciais presentes na equação diferencial por aproximações por diferenças finitas, cujo resultado final é uma equação algébrica, a equação de diferenças finitas.

As aproximações por diferenças finitas têm como base a expansão em série de Taylor de uma função *f*. Supondo que *f* seja contínua no intervalo [a,b] e que possua derivadas até a ordem *N* contínuas nesse intervalo, o Teorema de Taylor [30, 32] nos permite escrever, para todo *x* pertencente ao intervalo de interesse,

$$f(x) = f(x_0) + (\Delta x)\frac{\partial f}{\partial x}\Big|_{x_0} + \frac{(\Delta x)^2}{2!}\frac{\partial^2 f}{\partial x^2}\Big|_{x_0} + \frac{(\Delta x)^3}{3!}\frac{\partial^3 f}{\partial x^3}\Big|_{x_0} + \ldots + r$$

<u>Notação</u>:

$f(x_i) \rightarrow f_i$

$f(x_i + \Delta x) \rightarrow f_{i+1}$

$f_{i,j}^{t} = f(t, x, y)$

$f_{i+1,j+1}^{t+1} = f(t + \Delta t, x + \Delta x, y + \Delta y)$

em que $?x = x - x_0$ e *r* é o resto, definido como

$$r_N = \frac{(\Delta x)^N}{N!}\frac{\partial^N f}{\partial x^N}\Big|_{?}, ?\in [a,b]$$

O erro local de truncamento (ELT) é o conjunto dos termos da série de termos que não são usados na aproximação por diferenças finitas. Os termos de um $ELT = \left[-\frac{(\Delta x)}{2!}\frac{\partial^2 f}{\partial x^2}\Big|_i - \frac{(\Delta x)^2}{3!}\frac{\partial^3 f}{\partial x^3}\Big|_i - \cdots\right]$ serão representados por $O(\Delta x)$, o que significa que a aproximação é de primeira ordem. Neste caso, para Δx suficientemente pequeno, o erro na aproximação numérica da derivada de *f* é reduzido de forma linearmente proporcional ao espaçamento utilizado. Para uma aproximação de segunda ordem, indicada por $O(\Delta x)^2$, o erro na aproximação numérica da derivada de *f* é reduzido de forma quadrática. Deve-se notar que a ordem da aproximação só indica como o ELT varia com o refinamento da malha, e não o valor do erro.

Para as equações usadas nos testes deste trabalho, a derivada no tempo será discretizada em 1ª ordem e as derivadas espaciais em 2ª ordem, conforme a figura 6 a seguir.

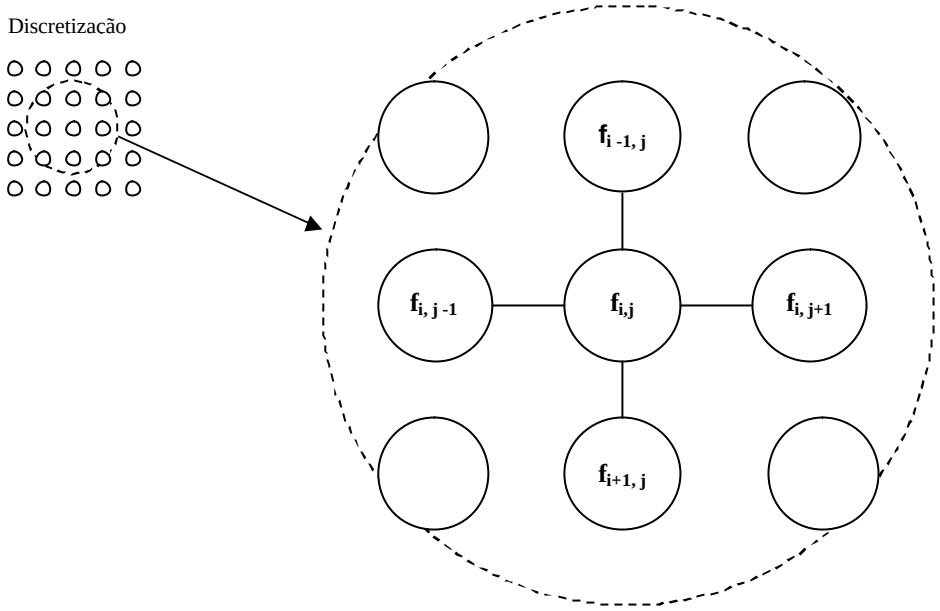


Figura 6 - Molécula computacional

Partindo das expansões de Taylor:

$$f(x_i + \Delta x) = f(x_i) + \Delta x \left.\frac{\partial f}{\partial x}\right|_i + \frac{(\Delta x)^2}{2!} \left.\frac{\partial^2 f}{\partial x^2}\right|_i + O(\Delta x)^3 \tag{1}$$

$$f(x_i - \Delta x) = f(x_i) - \Delta x \left.\frac{\partial f}{\partial x}\right|_i + \frac{(\Delta x)^2}{2!} \left.\frac{\partial^2 f}{\partial x^2}\right|_i + O(\Delta x)^3 \tag{2}$$

Para obtermos uma expressão para a derivada primeira por diferenças progressivas, já que ela será usada para discretizar o tempo, subtraímos $f(x_i)$ de (1), e obtemos:

$$f(x_i + \Delta x) - f(x_i) = (\Delta x) \left.\frac{\partial f}{\partial x}\right|_i + O(\Delta x)^2$$

$$\left.\frac{\partial f}{\partial x}\right|_i = \frac{f_{i+1} - f_i}{\Delta x} + O(\Delta x) \tag{3}$$

Para obtermos uma expressão por diferenças centrais na discretização do espaço, para a derivada segunda, somamos (1) e (2), e obtemos:

$$f(x_i + \Delta x) + f(x_i - \Delta x) = 2 f(x_i) + (\Delta x)^2 \left.\frac{\partial^2 f}{\partial x^2}\right|_i + O(\Delta x)^4$$

$$\left.\frac{\partial^2 f}{\partial x^2}\right|_i = \frac{f_{i+1} - 2 f_i + f_{i-1}}{(\Delta x)^2} + O(\Delta x)^2, \tag{4}$$

que é uma aproximação de segunda ordem.

2.6.1 - Discretização das equações usadas como exemplo

A seguir estão discretizadas as equações usadas como exemplo neste trabalho. Serão usadas uma equação de cada tipo, uma elíptica, uma parabólica e uma hiperbólica.

2.6.1.1 - Equação parabólica

Este tipo de EDP descreve problemas que evoluem no tempo e no espaço relacionados com processos de difusão/dispersão.

$$\frac{\partial u}{\partial t} = a\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right)$$

Onde u é a variável dependente e a é o coeficiente de difusão, e será considerado igual a 1 neste trabalho.

Notação:

$$u_{i,j}^{t} = u(t, x, y)$$

$$u_{i+1,j+1}^{t+1} = u(t+\Delta t, x+\Delta x, y+\Delta y)$$

A equação acima será discretizada usando as formulações explícita e implícita. O método Hopscotch é uma combinação dessas duas formulações.

2.6.1.1.1 - Formulação Explícita

Na formulação explícita, as equações são independentes, permitindo, portanto, solução direta. Usando (3) e (4), vem:

$$\frac{u_{i.j}^{t+1} - u_{i,j}^{t}}{\Delta t} = a\left(\frac{u_{i+1,j}^{t} - 2u_{i,j}^{t} + u_{i-1,j}^{t}}{(\Delta x)^2} + \frac{u_{i,j+1}^{t} - 2u_{i,j}^{t} + u_{i,j-1}^{t}}{(\Delta y)^2}\right) + O(\Delta t) + O\left[(\Delta x)^2\right] + O\left[(\Delta y)^2\right], \text{ e}$$

$$u_{i,j}^{t+1} = \left(1 - 2a\,(\Delta t)\left(\frac{1}{(\Delta x)^2} + \frac{1}{(\Delta y)^2}\right)\right)u_{i,j}^{t} + a\,(\Delta t)\left(\frac{u_{i+1,j}^{t} + u_{i-1,j}^{t}}{(\Delta x)^2} + \frac{u_{i,j+1}^{t} + u_{i,j-1}^{t}}{(\Delta y)^2}\right)$$

Fazendo ?x = ?y = h e definindo

$$K = a\,\frac{\Delta t}{h^2}, \tag{5}$$

vem:

$$u_{i,j}^{t+1} = (1-4k)u_{i,j}^{t} + K\left(u_{i+1,j}^{t} + u_{i-1,j}^{t} + u_{i,j+1}^{t} + u_{i,j-1}^{t}\right). \tag{6}$$

2.6.1.1.2 - Formulação Implícita

Na formulação implícita, as equações resultantes são acopladas, o que exige a resolução de um sistema de equações a cada passo de tempo. Usando (3) e (4), vem:

$$\frac{u_{i.j}^{t+1} - u_{i,j}^{t}}{\Delta t} = a\left(\frac{u_{i+1,j}^{t+1} - 2u_{i,j}^{t+1} + u_{i-1,j}^{t+1}}{(\Delta x)^2} + \frac{u_{i,j+1}^{t+1} - 2u_{i,j}^{t+1} + u_{i,j-1}^{t+1}}{(\Delta y)^2}\right) + O(\Delta t) + O\left[(\Delta x)^2\right] + O\left[(\Delta y)^2\right], \text{ e}$$

$$u_{i,j}^{t} = \left(1 + 2a\left(\Delta t\right)\left(\frac{1}{\left(\Delta x\right)^2} + \frac{1}{\left(\Delta y\right)^2}\right)\right)u_{i,j}^{t+1} - a\left(\Delta t\right)\left(\frac{u_{i+1,j}^{t+1} + u_{i-1,j}^{t+1}}{\left(\Delta x\right)^2} + \frac{u_{i,j+1}^{t+1} + u_{i,j-1}^{t+1}}{\left(\Delta y\right)^2}\right).$$

Fazendo $?x = ?y = h$ e definindo $K = a\frac{?t}{h^2}$, vem:

$$u_{i,j}^{t} = (1+4k)u_{i,j}^{t+1} - K\left(u_{i+1,j}^{t+1} + u_{i-1,j}^{t+1} + u_{i,j+1}^{t+1} + u_{i,j-1}^{t+1}\right), \text{ e}$$

$$u_{i,j}^{t+1} = \left(\frac{1}{1+4k}\right)\left(u_{i,j}^{t} + K\left(u_{i+1,j}^{t+1} + u_{i-1,j}^{t+1} + u_{i,j+1}^{t+1} + u_{i,j-1}^{t+1}\right)\right) \quad . \qquad (7)$$

2.6.1.2 - Equação elíptica

Este tipo de EDP descreve problemas de equilíbrio, por exemplo, térmico. A discretização da equação elíptica se faz de forma semelhante à da equação parabólica. O tempo associado ao método hopscotch, neste caso, é artificialmente adicionado, sendo chamado de tempo numérico.

$$\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = 0$$

Onde u é a variável dependente e a é o coeficiente de difusão, e será considerado igual a 1 neste trabalho.

Notação:

$u_{i,j}^{t} = u(t, x, y)$

$u_{i+1,j+1}^{t+1} = u(t + \Delta t, x + \Delta x, y + \Delta y)$

A equação acima será discretizada usando as formulações explícita e implícita. O método Hopscotch é uma combinação dessas duas formulações.

2.6.1.2.1 - Formulação Explícita

Na formulação explícita, as equações são independentes, permitindo, portanto, solução direta. Usando (3) e (4), vem:

$$\frac{u_{i.j}^{t+1} - u_{i,j}^{t}}{\Delta t} = a\left(\frac{u_{i+1,j}^{t} - 2u_{i,j}^{t} + u_{i-1,j}^{t}}{\left(\Delta x\right)^2} + \frac{u_{i,j+1}^{t} - 2u_{i,j}^{t} + u_{i,j-1}^{t}}{\left(\Delta y\right)^2}\right) + O(\Delta t) + O\left[(\Delta x)^2\right] + O\left[(\Delta y)^2\right], \text{ e}$$

$$u_{i,j}^{t+1} = \left(1 - 2a\left(\Delta t\right)\left(\frac{1}{\left(\Delta x\right)^2} + \frac{1}{\left(\Delta y\right)^2}\right)\right)u_{i,j}^{t} + a\left(\Delta t\right)\left(\frac{u_{i+1,j}^{t} + u_{i-1,j}^{t}}{\left(\Delta x\right)^2} + \frac{u_{i,j+1}^{t} + u_{i,j-1}^{t}}{\left(\Delta y\right)^2}\right)$$

Fazendo $?x = ?y = h$ e definindo

$$K = a\frac{\Delta t}{h^2}, \qquad (5)$$

vem:

$$u_{i,j}^{t+1} = (1-4k)u_{i,j}^{t} + K\left(u_{i+1,j}^{t} + u_{i-1,j}^{t} + u_{i,j+1}^{t} + u_{i,j-1}^{t}\right). \qquad (6)$$

2.6.1.2.2 - Formulação Implícita

Na formulação implícita, as equações resultantes são acopladas, o que exige a resolução de um sistema de equações a cada passo de tempo. Usando (3) e (4), vem:

$$\frac{u_{i.j}^{t+1} - u_{i,j}^{t}}{\Delta t} = a\left(\frac{u_{i+1,j}^{t+1} - 2u_{i,j}^{t+1} + u_{i-1,j}^{t+1}}{(\Delta x)^2} + \frac{u_{i,j+1}^{t+1} - 2u_{i,j}^{t+1} + u_{i,j-1}^{t+1}}{(\Delta y)^2}\right) + O(\Delta t) + O\left[(\Delta x)^2\right] + O\left[(\Delta y)^2\right], \text{ e}$$

$$u_{i,j}^{t} = \left(1 + 2a(\Delta t)\left(\frac{1}{(\Delta x)^2} + \frac{1}{(\Delta y)^2}\right)\right)u_{i,j}^{t+1} - a(\Delta t)\left(\frac{u_{i+1,j}^{t+1} + u_{i-1,j}^{t+1}}{(\Delta x)^2} + \frac{u_{i,j+1}^{t+1} + u_{i,j-1}^{t+1}}{(\Delta y)^2}\right).$$

Fazendo $?x = ?y = h$ e definindo $K = a\dfrac{?t}{h^2}$, vem:

$$u_{i,j}^{t} = (1+4k)u_{i,j}^{t+1} - K\left(u_{i+1,j}^{t+1} + u_{i-1,j}^{t+1} + u_{i,j+1}^{t+1} + u_{i,j-1}^{t+1}\right), \text{ e}$$

$$u_{i,j}^{t+1} = \left(\frac{1}{1+4k}\right)\left(u_{i,j}^{t} + K\left(u_{i+1,j}^{t+1} + u_{i-1,j}^{t+1} + u_{i,j+1}^{t+1} + u_{i,j-1}^{t+1}\right)\right) \quad . \qquad (7)$$

2.6.1.3 - Equação hiperbólica

Este tipo de EDP descreve problemas que envolvem propagação, transporte. Será agora discretizada a equação da onda linear de primeira ordem bidimensional. A equação é da forma

$$u_t + \nabla.(uc) = 0, \text{ onde } c = (c_1, c_2)^t.$$

Assim, $\dfrac{\partial u}{\partial t} = -\left[c_1\dfrac{\partial u}{\partial t} + c_2\dfrac{\partial u}{\partial y}\right]$.

2.6.1.3.1 - Formulação explícita

Usando (3) e (4), vem

$$\frac{u_{i,j}^{t+1} - u_{i,j}^{t}}{\Delta t} = -\left[c_1\frac{u_{i+1,j}^{t} - u_{i-1,j}^{t}}{2\Delta x} + c_2\frac{u_{i,j+1}^{t} - u_{i,j-1}^{t}}{2\Delta y}\right],$$

fazendo $\Delta x = \Delta y = h$ e $c_1 = c_2 = c$, vem:

$$u_{i,j}^{t+1} = u_{i,j}^{t} - c\frac{\Delta t}{2h}\left(u_{i+1,j}^{t} + u_{i-1,j}^{t} + u_{i,j+1}^{t} + u_{i,j-1}^{t}\right). \qquad (8)$$

2.6.1.3.2 - Formulação implícita

Usando (3) e (4), vem:

$$\frac{u_{i,j}^{t+1} - u_{i,j}^{t}}{\Delta t} = -\left[c_1 \frac{u_{i+1,j}^{t+1} - u_{i-1,j}^{t+1}}{2\Delta x} + c_2 \frac{u_{i,j+1}^{t+1} - u_{i,j-1}^{t+1}}{2\Delta y}\right],$$

fazendo $\Delta x = \Delta y = h$ e $c_1 = c_2 = c$, vem:

$$u_{i,j}^{t+1} = u_{i,j}^{t} - c\frac{\Delta t}{2h}\left(u_{i+1,j}^{t+1} + u_{i-1,j}^{t+1} + u_{i,j+1}^{t+1} + u_{i,j-1}^{t+1}\right) \qquad (9)$$

2.7 - APLICAÇÃO DO MÉTODO ODD-EVEN HOPSCOTH (OEH)

O método é eficiente e tem boa precisão, conforme [33, 55]. As equações explícitas e implícitas são aplicadas em quatro varreduras. Considere *iter* como o número da iteração. Primeiro, a equação explícita é aplicada quando *i* + *j* + *iter* é par, e a equação implícita é aplicada quando *i* + *j* + *iter* é ímpar. Depois, a equação explícita é aplicada quando *i* + *j* + *iter* é ímpar e a equação implícita é aplicada quando *i* + *j* + *iter* é par. Essas quatro varreduras formam um passo do método Hopscotch.

A figura 7 a seguir ilustra a aplicação do método. Os valores aproximados da solução nos pontos no instante de tempo t estão ilustrados com o uso do preenchimento em branco dos círculos que envolvem os pontos da discretização, na figura 7a. Primeiro, aplica-se a equação explícita em pontos alternados, começando no ponto acima e à esquerda, o que provê aproximações para os pontos da discretização identificáveis com o preenchimento em cinza dos círculos à sua volta. Após a aplicação da equação explícita, o domínio fica como na figura 7b. Em seguida, tendo em vista que os pontos envoltos por círculos cinza correspondem a informações no instante de tempo atualizado, t + ½, pode-se aplicar a fórmula implícita para obter aproximações neste instante de tempo, para os pontos que ainda não dispõem de aproximação para t + ½, ou seja, aqueles envoltos por círculos brancos (figura 7c). Note que todos os pontos necessários ao cálculo da equação implícita já estavam um instante de tempo à frente, e assim o cálculo da equação implícita se torna explícito, dispensando a resolução de um sistema de equações. Ao final desse procedimento o tempo será t + ½. Posteriormente, o passo se completa com a repetição do procedimento, invertendo a utilização da equação utilizada em cada ponto do domínio. A cada meio-passo torna-se necessário comunicar Agora, a equação explícita é aplicada aos pontos envoltos por círculos com preenchimento em

preto na figura 7d, e a aplicação da equação implícita ao restante dos pontos completa o passo (figura 7e).

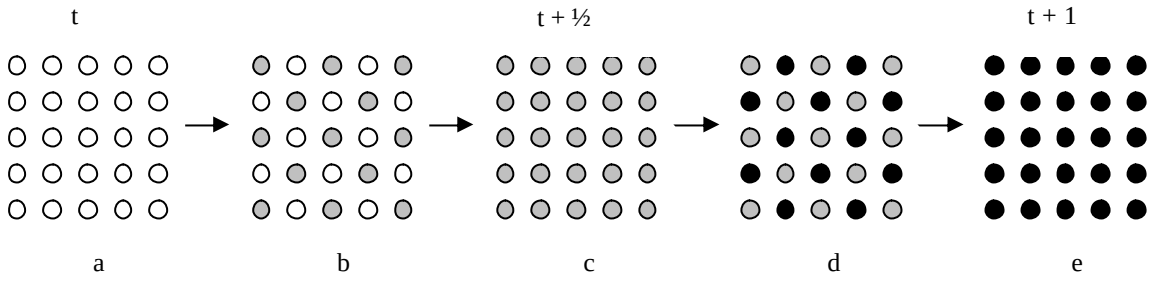


Figura 7 - Aplicação do OEH

O resultado do esquema OEH é uma seqüência de cálculos explícitos. Nas atualizações que seriam implícitas, todos os valores vizinhos envolvidos já estão conhecidos pela aplicação do método explícito, logo não existe a necessidade de resolução de sistemas lineares.

O método tem precisão de segunda ordem no espaço e é incondicionalmente estável para determinados tipos de problemas [93]. Sendo explícito, é rápido por não ter que resolver sistemas de equações e é adequado a paralelização porque os cálculos são feitos independentemente em cada região.

Para a execução de todos os cálculos necessários ao passo mostrado na figura 7, é necessária uma superposição de domínio de 2 fileiras. Isto é, cada partição do domínio contém uma parte dos subdomínios adjacentes. Isto permite que um meio-passo seja feito sem que seja preciso haver comunicação entre os processadores. Neste trabalho, um meio-passo corresponderá a uma iteração. Isto é, a exigência de comunicação é baixa em relação à quantidade de cálculo realizada nos processadores. Mais, a parcela $O(N)$ do algoritmo pode ser feita totalmente em paralelo. A parte do algoritmo que não é paralelizável é $O(1)$. Além disso, esse método permite um fácil particionamento do domínio. O sobretrabalho devido à paralelização é a tarefa de organizar os dados a serem transferidos, as franjas, em áreas reservadas à comunicação, e a de copiá-los após seu recebimento pelo processador destino da mensagem transmitida, para a estrutura local de dados (tarefas denominadas respectivamente empacotamento e desempacotamento). Neste trabalho, em que os subdomínios são quadrados, essa é uma tarefa $O(\sqrt{n})$.

Por esses motivos, o método Hopscotch é um algoritmo eficiente para a solução de EDPs usando processamento paralelo. Em [5, 8, 27, 28, 33, 54, 55, 77, 84, 86, 87, 91] apresentam-se aplicações do método Hopscotch a diversos ramos da engenharia, com bons resultados. Em [5] os autores resolvem numericamente as equações de águas rasas usando o

método Hopscotch com processamento paralelo. Em [8] os autores empregam o método Hopscotch para resolver as EDPs que modelam a transferência de íons em determinados componentes químicos. Em [27], o método Hopscotch é empregado na solução de equações elípticas em ambiente paralelo. Em [28] os autores aplicam o método Hopscotch ao problema de derretimento de gelo, onde a fronteira entre as partes líquida e sólida muda constantemente e precisa ser determinada como parte da solução do problema. Em [33] os autores empregam o método Hopscotch para resolver o modelo de um biosensor amperimétrico, que é um dispositivo capaz de detectar determinados componentes químicos. Em [54], o método Hopscotch é aplicado ao modelo de ondas viajantes de FitsHugh-Nagomo, que é um modelo usado para simular a propagação de ondas em meios excitáveis, tais como o tecido do coração e fibras nervosas. Em [55] os autores comparam diversos métodos numéricos de resolução das equações de Schrödinger, que descreve como o estado quântico de um sistema físico evolui no tempo, e tem aplicações em dinâmica dos fluidos, física de plasma e ótica não-linear. Em [77] os autores aplicam o método Hopscotch para resolver equações de propagação "laser" em semicondutores. Em [84] os autores utilizam o método Hopscotch para resolver as equações de Navier-Stokes. Em [86], os autores aplicam o método Hopscotch, em ambiente paralelo, a problemas de hidrodinâmica de transporte em águas rasas para estudo de ambientes para a vida marinha. Em [87] os autores empregam o método Hopscotch, em ambiente paralelo, como resolutor das equações que modelam o transporte de substâncias dissolvidas na água, para estudo de poluição em rios.

2.7.1 - Consistência, Estabilidade e Convergência do Método

Se a solução aproximada obtida pelo método converge para a solução real quando o espaçamento da malha e o intervalo de tempo tendem a zero, o método é convergente. Um método é estável se os erros provenientes da computação, tais como arredondamento e truncamento, diminuem à medida que as iterações avançam. Um método é consistente se os erros de truncamento local obtidos na discretização da série de Taylor tendem a zero quando h, K e o intervalo de tempo tendem a zero. A figura 8 a seguir sintetiza essas considerações para problemas lineares.

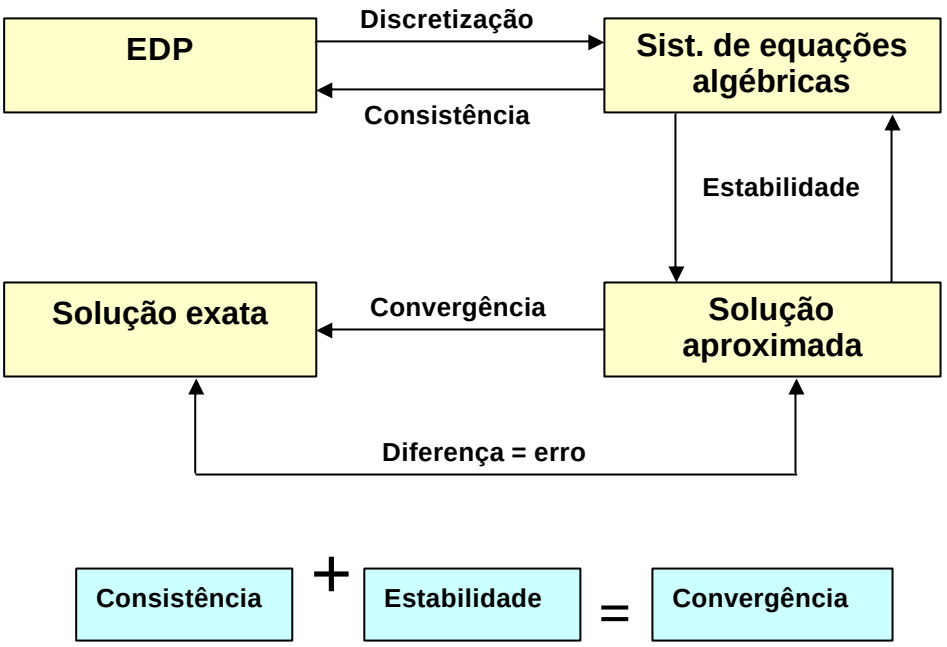


Figura 8 - Consistência, estabilidade e convergência para problemas lineares

Os procedimentos apresentados em [10, 66, 93] discutem condições de estabilidade do OEH para problemas de advecção-difusão unidimensionais e multidimensionais. Fazendo uso de analogias com o método leapfrog-Du Fort-Frankel, foi mostrado que a estabilidade do OEH não depende do coeficiente de difusão. Entretanto, existe uma limitação do intervalo de tempo devido à parte convectiva.

2.8 - CASOS TESTE

Para a validação da implementação computacional do método Hopscotch serão resolvidas equações de solução analítica conhecida para servir de padrão de comparação. Serão usadas três equações: uma elíptica, a equação de Poisson; uma parabólica, a equação do calor; e uma hiperbólica, a equação da onda de primeira ordem 2D. Essas equações serão usadas por conterem elementos presentes em modelos de fenômenos difusivos, convectivos e de equilíbrio. A equação de Poisson é uma equação de equilíbrio, enquanto as outras evoluem no tempo. Primeiramente essas equações serão implementadas usando processamento seqüencial. Foi usada uma máquina do ambiente computacional descrito no Apêndice D.

2.8.1 - Equação Parabólica

Será utilizada a equação de difusão parabólica

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x_1^2} + \frac{\partial^2 u}{\partial x_2^2}$$

no quadrado unitário $[0 \le x_1, x_2 \le 1] \times [t \ge 0]$, sujeita à condição inicial

$$u = sinpx_1 sinpx_2 \qquad \left(0 \le x_1, x_2 \le 1, \quad t = 0\right),$$

e às condições de contorno de Robin:

$$\left.\begin{aligned} \frac{\partial u}{\partial x_1} - 2u = pe^{-2p^2 t} sinpx_2, \quad x_1 = 0 \\ \frac{\partial u}{\partial x_1} + 2u = -pe^{-2p^2 t} sinpx_2, \quad x_1 = 1 \end{aligned}\right\} \quad 0 \le x_2 \le 1, \ t > 0$$

e

$$\left.\begin{aligned} \frac{\partial u}{\partial x_2} - 2u = pe^{-2p^2 t} sinpx_1, \quad x_2 = 0 \\ \frac{\partial u}{\partial x_2} + 2u = -pe^{-2p^2 t} sinpx_1, \quad x_2 = 1 \end{aligned}\right\} \quad 0 \le x_1 \le 1, \ t > 0$$

A solução deste problema é dada por

$$u = e^{-2p^2 t} sinpx_1 sinpx_2$$

Aplicação da Condições de Contorno:

$$\frac{u_{i+1,j} - u_{i-1,j}}{2\Delta x} + 2u_{i,j} = -pe^{-2p^2 t} sinpx_2$$

$$\frac{u_{i+1,j}}{2\Delta x} = \frac{u_{i-1,j}}{2\Delta x} - 2u_{i,j} - pe^{-2p^2 t} sinpx_2$$

$$u_{i+1,j} = \frac{u_{i-1,j}}{2\Delta x} 2\Delta x - 2u_{i,j} 2\Delta x - 2\Delta x pe^{-2p^2 t} sinpx_2$$

$$u_{i+1,j} = u_{i-1,j} - 2\Delta x(2u_{i,j} + 2\Delta x pe^{-2p^2 t} sinpx_2)$$

O mesmo procedimento é aplicado às outras três condições de contorno, resultando em:

$$u_{i-1,j} = u_{i+1,j} - 2\Delta x(2u_{i,j} + 2\Delta x pe^{-2p^2 t} \sin px_2)$$

$$u_{i,j+1} = u_{i,j-1} - 2\Delta y(2u_{i,j} + 2\Delta y pe^{-2p^2 t} \sin px_1)$$

$$u_{i,j-1} = u_{i,j+1} - 2\Delta y(2u_{i,j} + 2\Delta y pe^{-2p^2 t} \sin px_1)$$

A figura 9 a seguir mostra a orientação da discretização.

$$
\begin{array}{ccccc}
 & & u_{i,j+1} & & \\
 & & | & & \\
u_{i-1,j} & — & u_{i,j} & — & u_{i+1,j} \\
 & & | & & \\
 & & u_{i,j-1} & &
\end{array}
$$

Figura 9 - Discretização

Na figura 10 a seguir está apresentada a curva de nível da função no instante inicial, para uma discretização de 30 x 30.

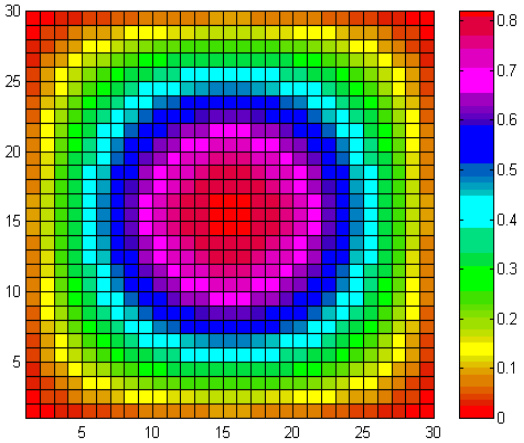


Figura 10 - Curvas de nível da condição inicial

Na figura 11 a seguir estão apresentados gráficos da função nos instantes t = 0,0s (condição inicial) e t = 0,01s.

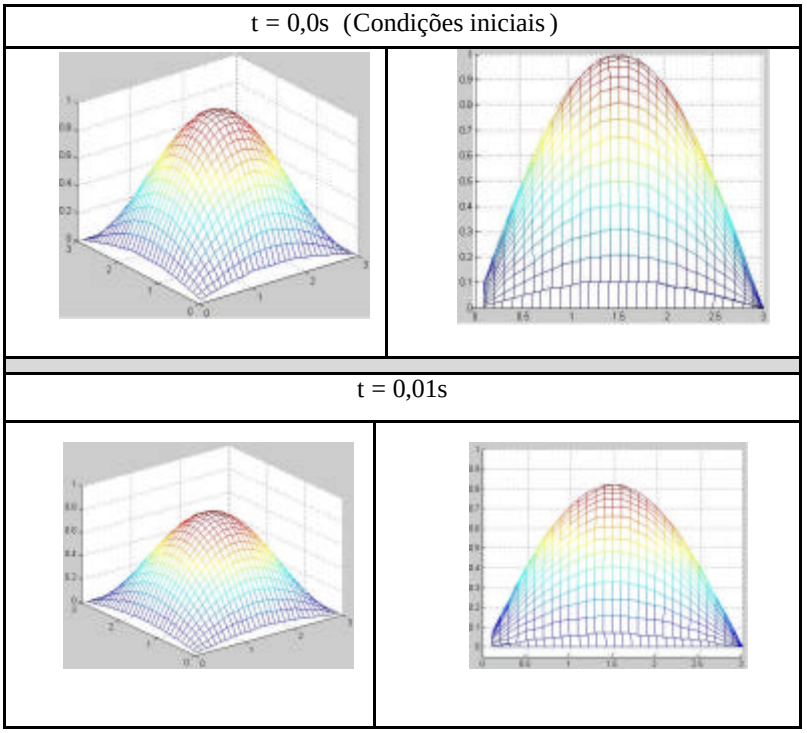


Figura 11 - Soluções da equação

Com o intuito de pesquisar qual o melhor valor de K usar, a tabela 2 foi construída. Ela apresenta resultados de erro máximo e tempo de execução para diferentes valores de K, com a quantidade de iterações calculada pela fórmula

$$qiter = \frac{t_{final}(n+1)^2}{K},$$

onde t_{final} = 0,001, $n=\sqrt{N}$ é uma dimensão do domínio e $qiter$ é a quantidade de iterações.

Tabela 2 - Variação de *K*

K **(60 x 60)**	**Quant. de iterações**	**Erro L_∞**	**Tempo (s)**
0,50	745	0,0643	1,7876
1,00	372	0,0699	0,8966
1,5	248	0,0816	0,6037
2,00	187	0,1003	0,4724
2,5	145	0,1300	0,4137

Pode-se ver que o erro se torna muito grande, mais de 10%, para valores de K acima de 2,0. Pode-se ver, também, que o erro se estabiliza para valores de K menores que 1. Logo, será usado um valor de K = 1,0 nos testes envolvendo essa equação. A tabela 3 a seguir apresenta o erro e o tempo para cinco granularidades da malha para tempo final de 10^{-4} e K = 1,0.

Tabela 3 - Resultados da precisão e tempo, com *K*=1,0

Refinamento *n* x *n*	**Quant. de iterações**	**Erro L_∞**	**Tempo(s)**
60 x 60	372	0.06989	1,641
120 x 120	1.464	0,03235	29,31
240 x 240	5.808	0,01363	427,5
480 x 480	23.136	0,006405	6.888,3
960 x 960	92.160	0,003490	111.862,5

2.8.2 - Equação Elíptica

Considere agora o problema de Poisson em $R = (0,1)^2$ definido por

$$-\nabla^2 u = f(x,y)$$

com condição de contorno de Dirichlet u = 0 sobre ∂R.

O termo fonte é dado por

$$f(x,y) = -90x^8y^{10}(1-x)(1-y) + 20x^9y^{10}(1-x)(1-y) - 90x^{10}y^8(1-x)(1-y) + 20y^9x^{10}(1-x)$$

A solução analítica desse problema é dada por $u(x,y) = x^{10}y^{10}(1-x)(1-y)$ para a condição inicial $u(x,y) = 0$ em *R*, e está apresentada na figura 12 a seguir.

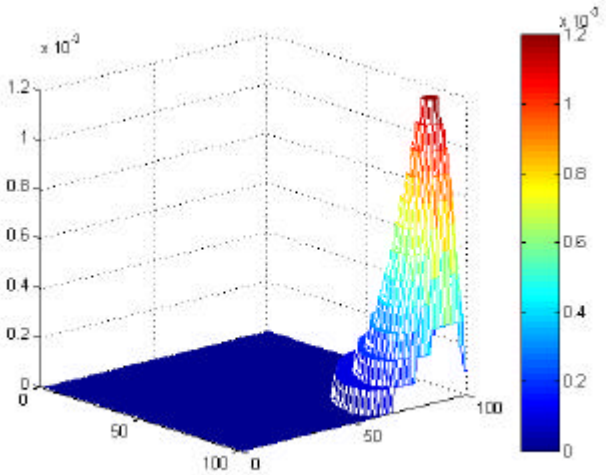


Figura 12 - Gráfico da solução da equação

A figura 13 a seguir apresenta as curvas de nível da solução da equação.

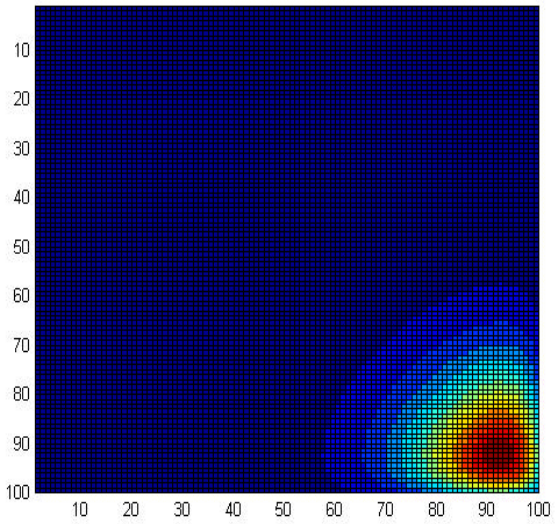


Figura 13 - Gráfico e curvas de nível da solução da equação

A tabela 4 apresenta resultados de erro máximo e tempo de execução para diferentes valores de *K*, com a quantidade de iterações calculada pela fórmula

$$qiter = \frac{t_{final}(n+1)^2}{K},$$

onde $t_{final} = 0{,}1$, $n = \sqrt{N}$ é uma dimensão do domínio e *qiter* é a quantidade de iterações. Deve-se lembrar que, nesse caso, o t_{final} é um tempo artificial.

Tabela 4 - Variação de *K*

K **(60 x 60)**	**Quant. de iterações**	**Erro L_∞**	**Tempo (s)**
0,5	745	0,00796758	1,599710
1,0	373	0,00791667	0.822446
1,5	248	0,00783927	0,425437
2,0	187	0,00775261	0,442356

Foi escolhido um valor de K igual a 1,5, por ser interno à região de melhor precisão e já apresentar uma queda acentuada no tempo de processamento. A tabela 5 a seguir apresenta a precisão da solução e o tempo de execução para cinco granularidades da malha, com a quantidade de passos e o Δt sendo escolhidos para obter um K de cerca de 1,5

Tabela 5 - Resultados da precisão e tempo, com K=1,5

Refinamento *n* x *n*	**Quant. de iterações**	**Erro L_∞**	**Tempo (s)**
60 x 60	248	0,007839	1,328
120 x 120	976	0,005332	21,19
240 x 240	3.872	0,005185	336,6
480 x 480	15.424	0,005095	5.362,7
960 x 960	61.568	0,005084	82.325,0

2.8.3 - Equação Hiperbólica

A equação da onda de primeira ordem no caso bidimensional é da forma

$$u_t + \nabla.(uc) = 0, \text{ onde } c = (c_1, c_2)^t.$$

Considere uma onda caminhando ao longo de uma direção não coincidente com os eixos coordenados. Em particular considere $u = u(x,y,t)$ Assim,

$$\frac{\partial u}{\partial t} + c_1 \frac{\partial u}{\partial x} + c_2 \frac{\partial u}{\partial y} = 0$$

com condição inicial $u(x,y,0)$ dada por

$$u(x,y,0) = 1 - 16\left[(x-0{,}25)^2 + (y-0{,}25)^2\right]$$

$$u(x,y,0) = 0 \qquad \text{se } u(x,y,0) < 0 \ ,$$

Considerando que $c_1 = c_2 = c$, a solução analítica desse problema é dada por

$u(x,y) = 1 - 16\left[(x-(1-c)0{,}25)^2 + (y-(1-c)0{,}25)^2\right]$, $u(x,y,0) = 0$ se $u(x,y,0) < 0$,

para a condição inicial $u(x,y) = 0$ em R, e está apresentada na figura 14 a seguir.

A onda tem a forma mostrada na figura 14 a seguir.

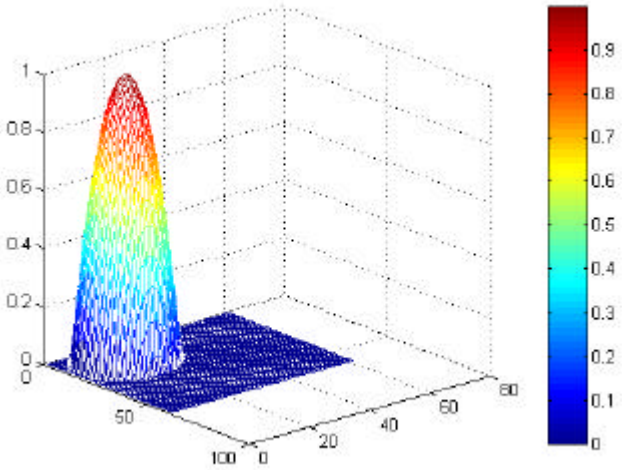
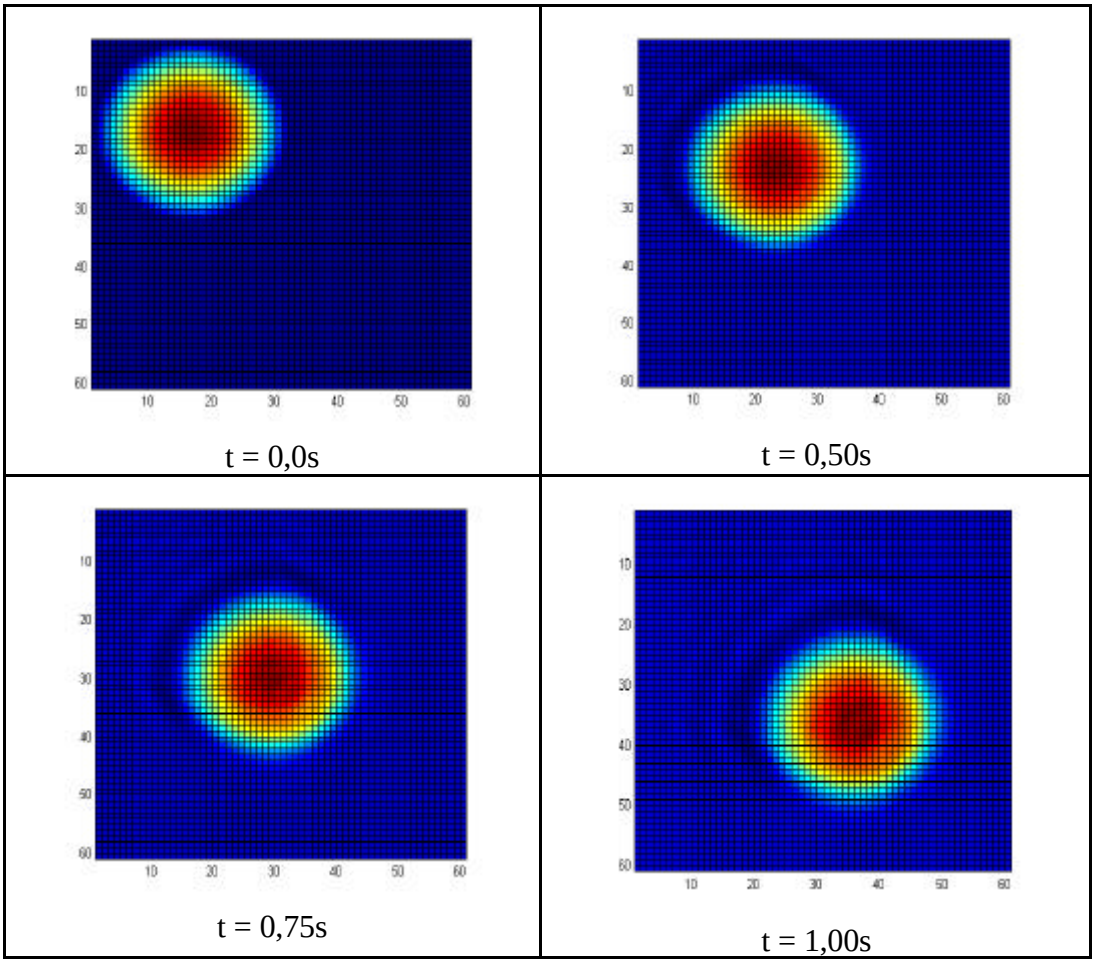


Figura 14 - Visualização da equação da onda

A figura 15 a seguir apresenta seis fotografias da evolução da onda à medida que as iterações avançam.



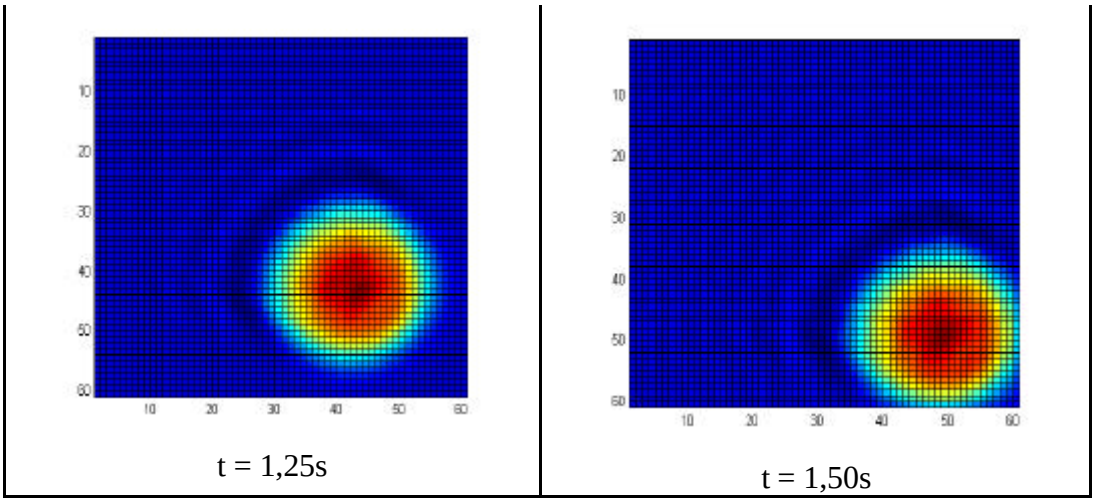


Figura 15 - Vistas de topo da evolução da onda conforme as iterações avançam, podendo-se observar a presença de difusão numérica

A tabela 6 a seguir apresenta o erro e o tempo para cinco granularidades da malha para K = 0,15. A quantidade de repetições foi calculada pela fórmula

$$qiter = \frac{c(n+1)t_{final}}{2K} \quad ,$$

onde t_{final} = 0,5, $c = 1{,}5\sqrt{2}$, $n=\sqrt{N}$ é uma dimensão do domínio e *qiter* é a quantidade de iterações.

Tabela 6 - Variação de *K*

K **(60 x 60)**	**Quant. de iterações**	**Erro** L_∞	**Tempo (s)**
0,05	647	0,146097	0,042719
0,10	323	0,144651	0,02320
0,15	215	0,142916	0,017313
0,20	161	0,140656	0,014018
0,25	129	0,149189	0,012487

A solução não converge para valores de K acima de 0,25. Foi escolhido um valor de K igual a 0,15, por ser interno à região de estabilidade e já apresentar uma queda acentuada no tempo de processamento. A tabela 7 a seguir apresenta a precisão da solução e o tempo de execução para cinco granularidades da malha, com a quantidade de passos e o $?t$ sendo escolhidos para obter um K de cerca de 0,15.

Tabela 7 - Resultados da precisão e tempo, com K=0,15

Refinamento n x n	**Quant. de iterações**	**Erro L_∞**	**Tempo (s)**
60 x 60	215	0,142916	0,03677
120 x 120	428	0,13108712	0,3647
240 x 240	852	0,15722607	3,503
480 x 480	1701	0,22640871	27,95
960 x 960	3398	0,21456814	204,7

CAPÍTULO 3

APLICAÇÕES PARALELAS

Uma aplicação paralela tem por objetivo diminuir o tempo de execução de um trabalho pela sua divisão em vários pedaços ou unidades de processamento, chamadas de tarefas. Estas são executadas em vários processadores, tirando proveito da maior capacidade de processamento e da maior quantidade de memória disponível.

Um ambiente paralelo pode ser formado por um supercomputador ou por várias estações de trabalho ou PCs, que compõem um "cluster". Uma das vantagens do "cluster" é a possibilidade de alocar vários processadores a um custo muito menor que o de um supercomputador. A ligação de vários "clusters" dispersos geograficamente é chamada de "grid". A desvantagem é que a comunicação entre os processadores é dependente da latência da rede, sendo bem mais lenta que no caso de um supercomputador, cujos processadores estão ligados a um barramento de dados. Há uma tendência em chamar de processamento distribuído o ambiente formado por várias máquinas independentes ligadas em rede, não necessariamente ocupando o mesmo local físico. Quando o ambiente é composto de um supercomputador paralelo, o processamento é chamado de processamento paralelo.

O tempo de execução é reduzido pela distribuição da carga pelos diversos processadores disponíveis. Idealmente, o tempo de execução de um código paralelo é proporcional a $1/N$, onde N é a quantidade de processadores. No entanto, as comunicações e o desbalanceamento da carga aumentam o tempo de processamento. Esse aumento estabelece uma perda de eficiência.

Talvez a maior diferença para o programador em relação à aplicação seqüencial é a possibilidade de perda do determinismo. Isto é, uma aplicação paralela pode nem sempre se comportar de modo determinístico. O comportamento da rede pode alterar a ordem de obtenção de recursos pelos processadores, e condições de "deadlock" podem ocorrer, alterando o comportamento do programa paralelo sem que o código tenha sido alterado (o "deadlock" ocorre com um conjunto de processos e recursos não-preemptíveis, onde um ou mais processos desse conjunto está aguardando a liberação de um recurso por um outro processo que, por sua vez, aguarda a liberação de outro recurso alocado ou dependente do primeiro processo). Isto é, para a mesma entrada, com mesmo programa e mesmos dados, a mesma saída não é garantidamente obtida, o que caracteriza um comportamento probabilístico. Por isso, o programador precisa tomar cuidados para garantir que este tipo de comportamento não ocorra.

3.1 - MÉTRICAS

A seguir serão definidas algumas métricas, além daquelas já definidas em 2.5, que são importantes para quantificar especificamente o desempenho de uma aplicação paralela.

3.1.1 - "Speedup"

Fornece a medida de quanto um programa rodando em paralelo é mais rápido do que esse mesmo programa rodando seqüencialmente. É definido como $S(p) = \frac{t_S}{t_p}$ [45], onde p é a quantidade de processadores, t_1 é o tempo de execução do programa seqüencial e t_p é o tempo de execução do programa paralelo com p processadores.

A lei de Amdahl [3] estabelece um limite superior para o "speedup", que é dependente da parte do programa que não é paralelizável. Se P é a proporção do programa que pode ser tornado paralelo, e (1-P) for a proporção que não pode ser paralelizado, então o máximo "speedup" que pode ser obtido usando p processadores é

$$\frac{1}{(1-P)+\frac{P}{p}}$$

No limite, quando p tende a infinito, o "speedup" máximo tende para $1/(1-P)$. Por esta razão, computação paralela torna-se mais vantajosa quando P torna-se maior. Para uma aplicação cujo P fosse 1 (ideal), o "speedup" por usar p processadores seria p, chamado de "speedup" linear. Entretanto, não é incomum a obtenção de "speedup" superlinear, que é a obtenção de um "speedup" superior a p em uma máquina com p processadores. Uma razão para isso é que a memória cache aumenta com o número de processadores, tornando possível carregar maiores quantidades de dados em uma cache com tamanho agregado maior, evitando assim operações de "swapping", reduzindo com isso a quantidade de acessos à memória.

Em [39] mostra-se que a lei de Amdahl não deve ser encarada como uma restrição ao uso de processamento paralelo massivo.

3.1.2 - Eficiência

Para programas paralelos, a *eficiência* é definida como $E(p) = \frac{t_S}{t_p p}$, onde t_S é o tempo de execução do programa seqüencial e t_p é o tempo de execução do programa paralelo usando p processadores.

3.1.3 - Escalabilidade

A escalabilidade é a medida da capacidade de um sistema aumentar o "speedup" em proporção linear ao aumento do número de processadores. O conceito de escalabilidade [57, 76] em sistemas paralelos é que, dado um nível de desempenho em um determinado problema, manter esse nível de desempenho com um aumento de carga pode ser resolvido com um aumento dos recursos computacionais, isto é, aumentando-se a quantidade de processadores.

3.2 - MONITORAMENTO

A depuração de uma aplicação paralela é uma tarefa difícil e trabalhosa. Foi usada análise "on-line" e "off-line" para monitorar o desempenho da aplicação. No monitoramento "on-line", as informações são analisadas durante a execução do programa. No monitoramento "off-line", são gravados arquivos, um por processo, para posterior análise. Como a aplicação deste trabalho é naturalmente síncrona, não houve necessidade de sincronização dos relógios das máquinas. Em [92] os autores descrevem alguns tipos de erros relacionados com o desenvolvimento de aplicações paralelas.

3.3 - CONCEITOS DE PARALELIZAÇÃO

A seguir são apresentados mais alguns conceitos de paralelização.

3.3.1 - Paradigmas

Existem dois paradigmas de programação, segundo [40]:

a) SPMD ("single program", "multiple data"): existe apenas um código, que roda em todos os processadores; e
b) MPMD ("multiple programs", "multiple data"): existe mais de um código.

O programa desenvolvido aqui usa o paradigma SPMD, ou seja, existe apenas um código que roda em todos os processadores.

3.3.2 - Granularidade

É a medida do tamanho da tarefa [76]. Grão computacional é o tamanho da tarefa. No caso do método Hopscotch, o grão computacional é a quantidade de operações aritméticas a serem realizadas em um subdomínio, em um meio-passo. A granularidade é uma medida da razão entre computação e comunicação. Programas com granularidade fina implicam em pequena quantidade de computação entre comunicações sucessivas e são pouco beneficiados pela paralelização, enquanto aqueles com granularidade grossa se beneficiam desta [94].

3.4 - SUPERCOMPUTADORES

São máquinas que estão entre as mais rápidas do mundo em um dado momento. Esta é a conceituação de supercomputador. As principais classes de arquiteturas serão definidas a seguir, usando a taxonomia de *Flynn* (1966) [88, 90], que há muitos anos é usada para a classificação dos supercomputadores. Existem dois tipos principais de supercomputador:

a) **Sistemas de memória compartilhada**: Esses sistemas têm múltiplas CPU (Central Processing Unit, Unidade Central de Processamento) que dividem o mesmo espaço de endereçamento de memória. Isto significa que é transparente para o usuário aonde os dados estão armazenados, já que existe apenas uma memória acessada por todas as CPUs. Máquinas com um processador vetorial podem ser consideradas como um exemplo de sistemas SIMD de memória compartilhada (SM-SIMD), enquanto modelos com vários processadores são exemplos de máquinas MIMD (SM-MIMD).

b) **Sistemas de memória distribuída**: Neste caso cada CPU tem sua própria memória. As CPUs são conectadas por uma rede e trocam dados entre memórias quando é necessário. Ao contrário das máquinas com memória compartilhada, o usuário precisa saber a localização dos dados e precisa explicitamente manipulá-los. Sistemas MIMD com memória distribuída (DM-MIMD) exibem uma grande variedade na topologia de sua rede de interconexão das memórias e CPUs, e que são, em geral, transparentes ao usuário.

Esses dois tipos de sistemas podem ser usados nas máquinas apresentadas a seguir, cuja classificação é baseada na maneira de manipular as instruções e os dados. Podem ter memória compartilhada ou distribuída. Possuem duas principais classes.

a) **Máquinas SIMD**: Esses sistemas têm uma grande quantidade de processadores, as mais modernas contendo mais de 16.384 processadores, todas podendo executar a mesma instrução em massas de dados diferentes. Uma subclasse dos sistemas SIMD são os processadores vetoriais. Processadores vetoriais agem em vetores em vez de agir em um único dado, usando para isso CPUs especialmente estruturadas. Quando os dados são adequados à utilização dessas unidades vetoriais, os resultados podem ser obtidos na razão de um, dois ou três por ciclo de relógio do sistema. Assim, os processadores vetoriais atuam como se fossem em paralelo, mas só quando executando em modo vetorial, sendo então, nesse caso, várias vezes mais rápidos do que quando executando em modo escalar. Assim, para fins práticos, processadores vetoriais são considerados como máquinas SIMD.

b) **Máquinas MIMD**: Essas máquinas executam vários grupos de instruções em paralelo em diferentes massas de dados.A diferença para as máquinas multi-processador SISD é o fato de que na máquinas MIMD as instruções e os dados são relacionados porque eles representam pedaços diferentes da mesma tarefa sendo executada. Assim, sistemas MIMD podem executar sub-tarefas em paralelo de modo a diminuir o tempo de execução da tarefa principal sendo executada.

3.5 - TOPOLOGIA DA REDE DE PROCESSADORES

O conhecimento da topologia da rede de processadores é importante na otimização do particionamento de um grafo, que pode ser feito de modo a tirar o máximo proveito da topologia do supercomputador A maioria dos particionadores existentes considera a topologia da rede em seus algoritmos de particionamento.

A figura 16 a seguir apresenta exemplos das topologias de rede mais usadas em um computador de memória compartilhada, sendo *n* a quantidade de processadores..

a) Uma *"crossbar"* usa n^2 conexões;

b) Uma rede-Ω usa $n\log_2 n$ conexões; e

c) Uma rede com barra central usa somente uma conexão.

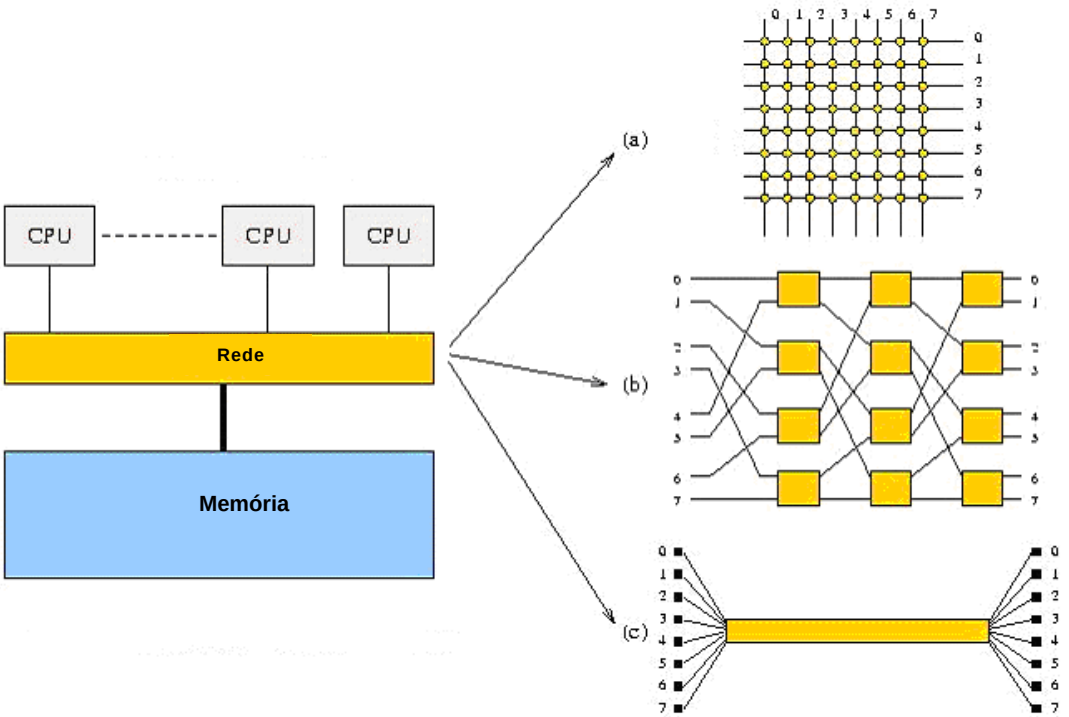


Figura 16 - Algumas topologias usadas em um computador de memória compartilhada

Uma das topologias mais usadas é a assim chamada "hypercube topology", apresentada na figura 17 a seguir.

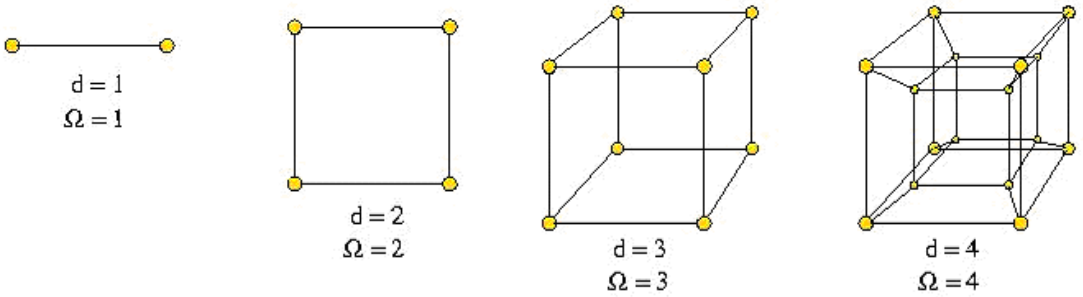


Figura 17 - Topologia em hipercubos

Arquiteturas baseadas em hipercubos têm sido extensivamente usada no projeto de computadores paralelos e na construção de algoritmos paralelos. Um hipercubo tem quantidade de nós igual a 2^n, sendo n a dimensão do hipercubo. Assim, por exemplo, um supercomputador com 2^n = 1024 processadores tem a topologia da rede de ligação dos processadores baseada em um hipercubo de dimensão 10. Uma vantagem da topologia em hipercubos é que para um hipercubo com 2^d nós, a quantidade de passos entre dois quaisquer nós é no máximo d. Assim, a dimensão da rede cresce somente logaritmicamente com a quantidade de nós. Na prática, hoje em dia, a exata topologia de uma rede hipercúbica não é mais tão importante, porque todos os sistemas no mercado atualmente usam a tecnologia do "wormhole routing". Para uma mensagem enviada de i para j, um sinal é enviado de i para j, resultando em uma ligação direta entre i e j. Assim que esta conexão é estabelecida, os dados propriamente ditos são enviados sem perturbar a operação dos nós intermediários.

Outra maneira eficiente de conectar uma grande quantidade de processadores é por meio de uma "fat tree". A figura 18 a seguir apresenta um exemplo de uma "fat tree" quaternária. Na prática, nos nós próximos à raiz acontece um congestionamento, resolvido por disponibilizar mais capacidade de comunicação nesses nós.

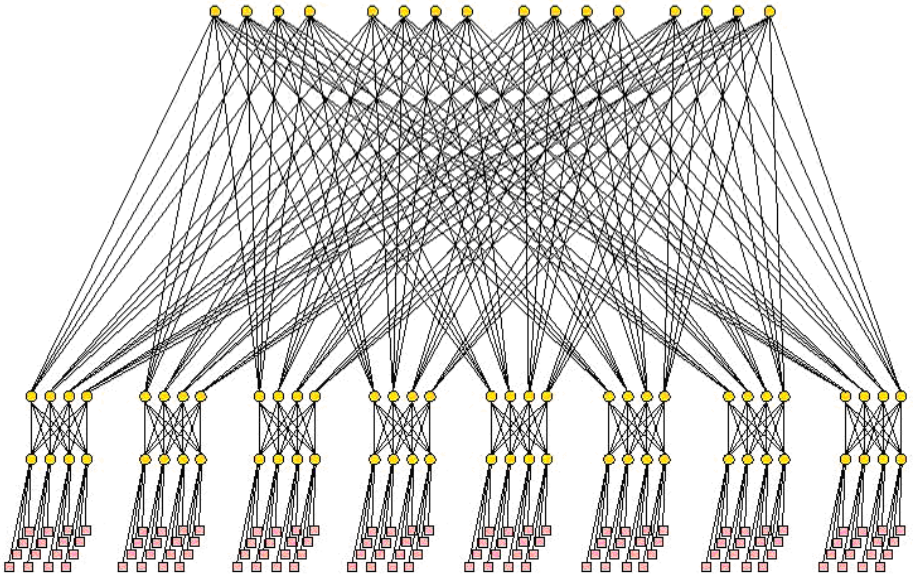


Figura 18 - Exemplo de uma "fat-tree" quaternária

3.6 - PARALELIZAÇÃO DO MÉTODO HOPSCOTCH

A solução numérica de EDPs envolve a discretização do domínio em pontos, formando uma malha. Para resolver o problema em paralelo, a malha precisa ser decomposta em subdomínios, idealmente tantos quantos forem os processadores disponíveis, de modo que os cálculos referentes a cada subdomínio sejam realizados por um dos processadores alocados.

Se dividirmos o domínio R em tantos subdomínios quantos forem os processadores disponíveis, cada subdomínio poderá ser calculado simultaneamente com os outros. Considerando que o esquema de diferenças finitas adotado neste trabalho usa a tradicional discretização de cinco pontos (ver figura 9), cada ponto precisa somente acessar seus quatro vizinhos, e somente ao final de cada meio passo será necessário comunicar as soluções das bordas entre os processadores.

Uma versão paralela do método Hopscotch foi implementada. Ela usa "Message-Passing Interface" (MPI) para a distribuição inicial da carga e a comunicação entre os processadores.

3.6.1 - Implementação

A paralelização da resolução pelo método Hopscotch será explicada através de um exemplo. Seja uma subdivisão utilizando 9 processadores [81], cada um deles com um subdomínio 5 x 5 a título de exemplo. O programa permite que se use qualquer quantidade de processadores e qualquer tamanho de subdomínio, com a restrição de que exista sempre o

mesmo número de linhas e colunas, isto é, o número de subdomínios deve ser um quadrado perfeito, 9, 16, 25, 36, 49, 64 e assim por diante. A figura 19 a seguir mostra como fica um domínio 15 x 15 dividido em 9 subdomínios.

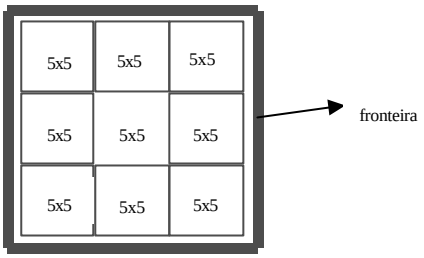


Figura 19 - Divisão do domínio

Independentemente do número de subdomínios definidos, existirão no máximo nove tipos de subdomínios, de acordo com a existência e a posição do contorno.

3.6.2 - Tipos de Subdomínio com Relação ao Contorno

Com relação a posição do contorno, existem no máximo nove tipos de subdomínio que devem ser tratados diferentemente no código. A figura 20 seguir apresenta um exemplo de um domínio com n = 15 (15 x 15), particionado em 9 subdomínios com m = 5, cada um deles de um tipo diferente. Os valores nos pontos amarelos são atualizados explicitamente, os valores nos pontos pretos são atualizados implicitamente e os pontos vermelhos são o contorno. Deve-se notar que na camada da franja mais próxima do subdomínio é necessária uma atualização explícita dos valores de alguns pontos. Essa atualização explícita usa a camada mais externa da franja, daí a necessidade de uma franja de duas camadas.

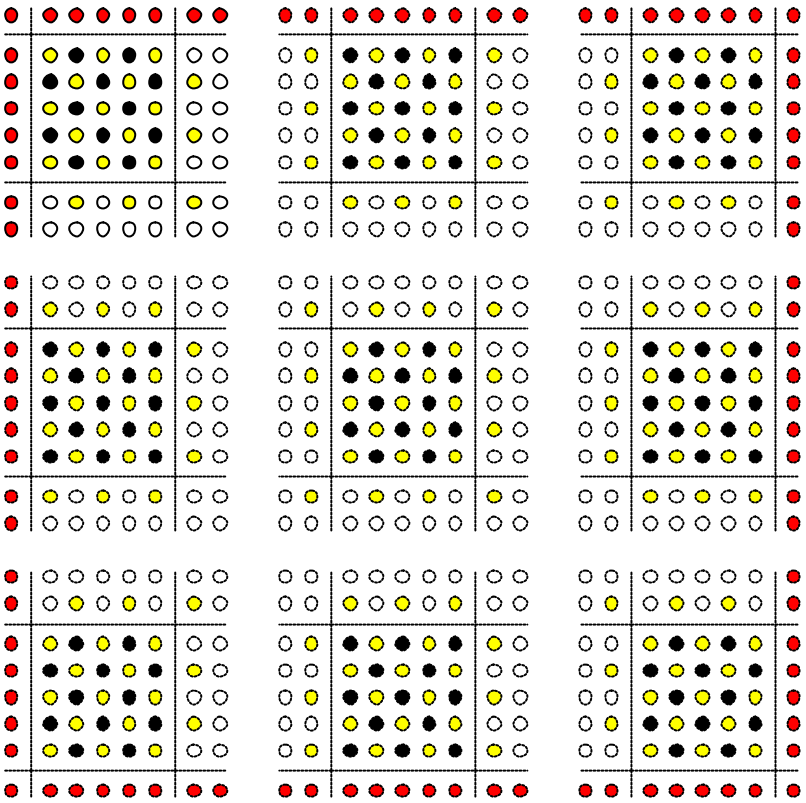


Figura 20 - Exemplo de particionamento

Os tipos de subdomínio são mostrados a seguir.

Tipo 1	**Tipo 4**	**Tipo 7**
Tipo 2	**Tipo 5**	**Tipo 8**
Tipo 3	**Tipo 6**	**Tipo 9**

Figura 21 - Tipos de subdomínio

A figura 22 a seguir mostra um exemplo de um domínio dividido em 36 subdomínios, podendo-se ver que existirão, para particionamentos a partir de 9 subdomínios de um domínio n x n:

a) um subdomínio do tipo 1, tipo 3, tipo 7 e tipo 9;

b) n-2 subdomínios do tipo 2, tipo 4 tipo 6 e tipo 8.; e

c) (n-2) x (n-2) subdomínios do tipo 5.

Figura 22 - Tipos de subdomínio em um domínio particionado em 36 subdomínios

3.6.3 - Comunicação entre os Processadores

Um meio passo do método é o cálculo alternado nos pontos pelas equações explícita e implícita. Com todos os pontos calculados, a complementação do passo será usar a fórmula explícita para calcular os pontos que foram calculados com a fórmula implícita, e vice-versa. O cálculo dos valores dos pontos localizados nas bordas de um subdomínio depende dos valores de pontos localizados nos subdomínios adjacentes.

A posição relativa dos subdomínios é mostrada na figura 23 a seguir.

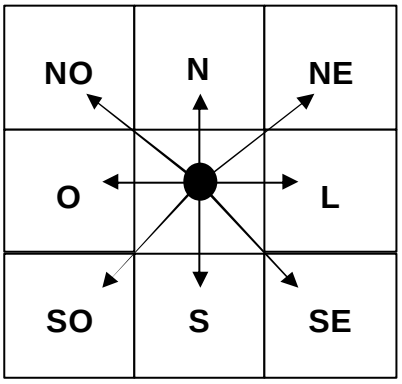


Figura 23 - Posição relativa dos subdomínios

Essa comunicação consistirá de valores dos pontos de um subdomínio que devem ser mandados para os seus vizinhos. Assim, cada subdomínio terá cópias de valores de pontos dos subdomínios adjacentes. Esses pontos são chamados de franja. As franjas são enviadas por cada processador conforme a figura 24 a seguir.

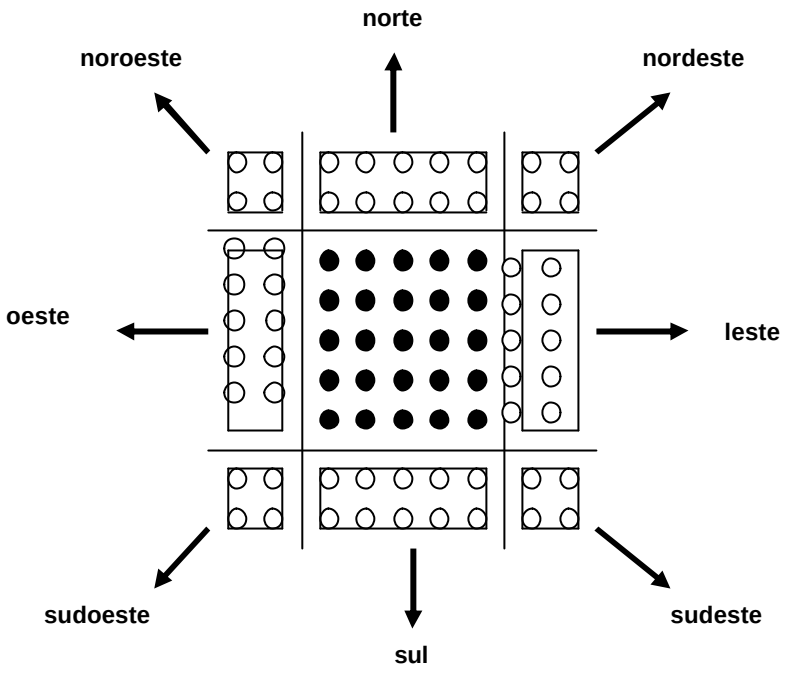


Figura 24 - Subdomínio com as franjas enviadas. Os pontos pretos são pontos do subdomínio, e os pontos brancos são pontos das franjas

As franjas são recebidas de cada vizinho conforme figura 25 a seguir.

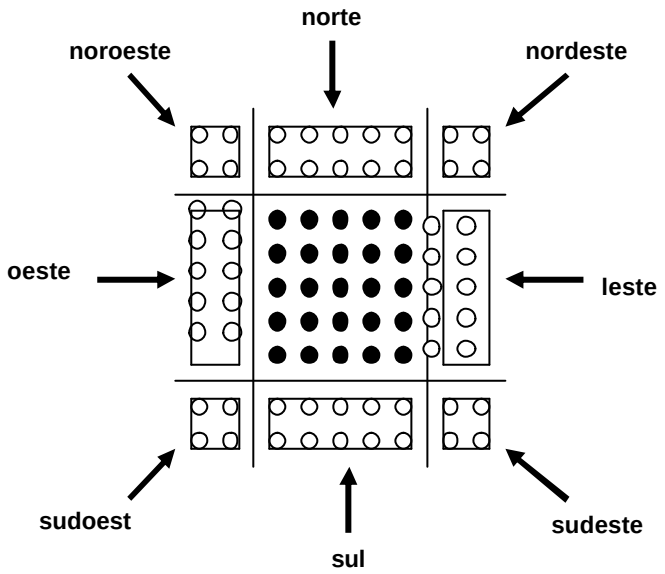


Figura 25 - Subdomínio com as franjas recebidas. Os pontos pretos são pontos do subdomínio, e os pontos brancos são pontos das franjas

O Método Hopscotch exige uma franja de duas camadas, fazendo com que cada subdomínio tenha dimensão (m + 2) x (m + 2), pois no primeiro meio-passo é necessário que se calcule o valor da equação nos pontos m + 1, usando, para isso, os pontos m + 2, devido ao esquema dos cinco pontos (ver figura 9).

A comunicação é síncrona, pois a cada meio passo é necessária a comunicação entre os subdomínios, isto é, cada subdomínio necessita dos outros para poder prosseguir.

3.7 - O LAM/MPI

LAM (Local Area Multicomputer) é um ambiente de programação e desenvolvimento de sistemas MPI para computadores heterogêneos ligados a uma rede. O modelo utilizado aqui pressupõe memória distribuída, o que torna necessária a troca de mensagens entre os processos localizados em processadores diferentes. Foi usado o MPI para a comunicação entre os processadores. O MPI é uma biblioteca de troca de mensagens desenvolvida para ambientes de memória distribuída, máquinas paralelas massivas, "clusters" e redes heterogêneas. O programador é responsável pela distribuição das comunicações entre os processadores. No desenvolvimento dessa aplicação foram usadas as funções do MPI descritas a seguir.

3.7.1 - Rotinas Básicas

Essas são as rotinas básicas de inicialização, identificação, troca de mensagens e finalização de uma aplicação MPI.

MPI_Init: Inicializa um processo MPI. Portanto, deve ser a primeira rotina a ser chamada por cada processo, pois estabelece o ambiente necessário para executar o MPI. Também sincroniza todos os processos na inicialização da aplicação.

MPI_Comm_rank: Identifica um processo dentro de um determinado grupo. Retorna sempre um valor inteiro entre 0 e p-1, onde p é a quantidade de processos.

MPI_Comm_size: Retorna a quantidade de processos dentro de um grupo.

MPI_Send: Rotina básica para envio de mensagens.

MPI_Recv: Rotina básica para o recebimento de mensagens. Bloqueia até ser recebida a mensagem.

MPI_Finalize: É chamada para encerrar a aplicação MPI. É usada para liberar memória.

3.7.2 - Rotinas de Buferização

As rotinas a seguir foram necessárias porque a aplicação congelava durante as comunicações entre os processadores. Apesar de não apresentar erro, a aplicação estava em uma situação de "deadlock". Um "deadlock" é uma situação que aparece quando um processso não pode prosseguir porque está esperando um outro processo que, por sua vez, está esperando pelo primeiro processo [59, 65].

MPI_Buffer_attach: Cria um “buffer” de tamanho definido pelo programador para operações de envio de dados.

MPI_Buffer_detach: Remove um “buffer” criado.

MPI_Bsend: Rotina para envio de mensagens usando um “buffer”.

3.7.3 - Rotinas de Broadcast e Sincronização

Essas rotinas foram usadas para sincronizar a aplicação e para distribuir uma mesma informação para todos os processos.

MPI_Alltoall: Envia dados a partir de cada processo para todos os processos.

MPI_Barrier: Bloqueia até que todos os processos tenham atingido esta rotina.

3.8 - CASOS TESTE

Uma versão paralela do método Hopscotch foi implementada [38]. As mesmas equações usadas para o caso seqüencial (ver item 2.8) serão usadas agora no programa paralelizado. O ambiente computacional usado nos testes está apresentado no Apêndice D.

Foram usadas diversas quantidades de processadores na resolução das EDPs para obter estimativas para as métricas estabelecidas em 3.1. Cada subdomínio foi enviado a um processador, isto é, $p = q$.

3.8.1 - Equação Parabólica

Usando agora a equação parabólica de 2.8.1 com o programa paralelo, dividindo-se o domínio em 4, 9, 16 e 25 subdomínios, obtêm-se os seguintes resultados.

Tabela 8 - Métricas da paralelização do Hopscotch

Tamanho domínio	*q*	Tamanho subdom.	Tempo (s)	“Speedup”	Eficiência	erro
60 x 60	4	30 x 30	0,6603	2,485	0,6213	0,05757
	9	20 x 20	0,5648	2,905	0,7264	
	16	15 x 15	0,6593	2,490	0,6222	
	25	12 x 12	1,214	1,352	0,3379	
120 x 120	4	60 x 60	8,141	3,600	0,4000	0,02510
	9	40 x 40	5,351	5,477	0,6086	
	16	30 x 30	5,000	5,862	0,3664	
	25	24 x 24	5,334	5,495	0,2198	
240 x 240	4	120 x 120	110,1	3,880	0,9707	0,01089
	9	80 x 80	60,85	7,025	0,7806	
	16	60 x 60	43,79	9,762	0,6102	
	25	48 x 48	36,06	11,85	0,4742	

Tamanho domínio	*q*	Tamanho subdom.	Tempo (s)	"Speedup"	Eficiência	erro
480 x 480	4	240 x 240	1868,8	3,686	0,9215	0,002463
	9	160 x 160	840,2	8,198	0.9109	
	16	120 x 120	552,7	12,46	0,7789	
	25	96 x 96	365,2	18,86	0,7545	
960 x 960	4	480 x 480	30.041,2	3,723	0,9307	0,001443
	9	320 x 320	14.191,4	7,882	0,8758	
	16	240 x 240	8.148,2	13,73	0,8581	
	25	192 x 192	5.457,5	20,50	0,8200	

As figuras 26 e 27 a seguir apresentam os gráficos dos valores do "speedup" e da eficiência em função da quantidade de processadores utilizados no processamento. O domínio foi particionado na mesma quantidade dos processadores utilizados no processamento, isto é, cada subdomínio foi enviado para um processador.

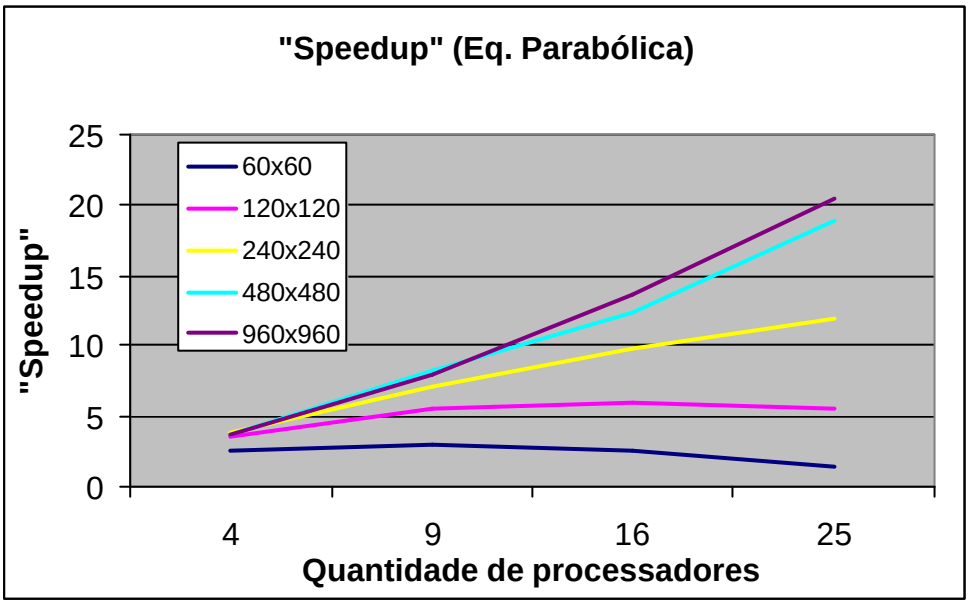


Figura 26 - "Speedup" da resolução da equação parabólica

Pode-se ver que os subdomínios que contêm mais pontos, de 240x240 para cima, apresentam "speedup" próximo do linear. Os subdomínios que contêm menos pontos que este apresentaram inclusive saturação em seu "speedup".

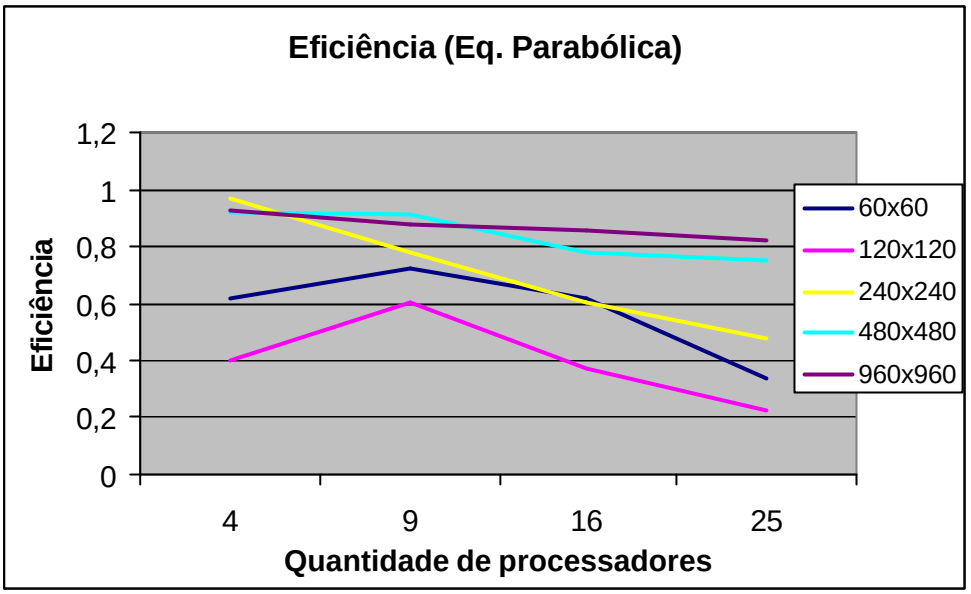


Figura 27 - Eficiência da resolução da equação parabólica

O gráfico da figura 27 mostra que a eficiência tende a cair quando se aumenta a quantidade de processadores, com a conseqüente diminuição dos subdomínios. O resolutor obteve melhor desempenho com os subdomínios maiores, obtendo valores acima de 80% de eficiência para o domínio 960 x 960.

O gráfico a seguir mostra a escalabilidade. A série 1 mostra como o "speedup" aumentou quando a carga passou de 60 x 60 para 120 x 120, e a quantidade de processadores passou de 4 para 16, isto é, um aumento quadrático na carga e na quantidade de processadores. A série 2 mostra o "speedup" quando o tamanho do domínio passou de 120 x 120 para 240 x 240, e a quantidade de processadores passou de 4 para 16. A série 3 mostra o "speedup" quando o tamanho do domínio passou de 240 x 240 para 480 x 480, e a quantidade de processadores passou de 4 para 16. A série 4 mostra o "speedup" quando o tamanho do domínio passou de 480 x 480 para 960 x 960, e a quantidade de processadores passou de 4 para 16.

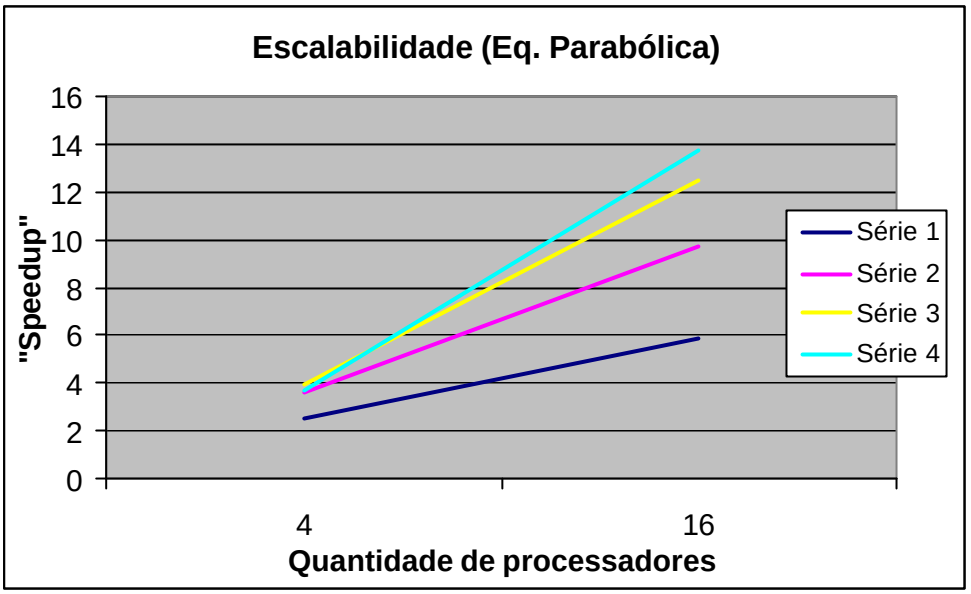


Figura 28 - Escalabilidade da resolução da equação parabólica

3.8.2 - Equação Elíptica

Usando agora a equação de Poisson de 2.8.2 com o programa paralelo, dividindo-se o domínio em 4, 9, 16 e 25 subdomínios, obtêm-se os seguintes resultados.

Tabela 9 - Métricas da paralelização do Hopscotch

Tamanho domínio	*q*	Tamanho subdom.	Tempo (s)	"Speedup"	Eficiência	erro
60 x 60	4	30 x 30	1,05	1,265	0,3162	0,006926
	9	20 x 20	0,667	1,991	0,2212	
	16	15 x 15	0,616	2,156	0,1347	
	25	12 x 12	0,645	2,059	0,08236	
120 x 120	4	60 x 60	12,5	1,695	0,4237	0,001788
	9	40 x 40	7,04	3,010	0,3344	
	16	30 x 30	5,02	4,221	0,2638	
	25	24 x 24	4,74	4,470	0,1788	
240 x 240	4	120 x 120	185,9	1,811	0,4527	0,0006534
	9	80 x 80	89,02	3,781	0,4201	
	16	60 x 60	55,51	6,064	0,3790	
	25	48 x 48	40,23	8,367	0,3347	
480 x 480	4	240 x 240	2897,6	1,851	0,4627	0,0005835
	9	160 x 160	1321,1	4,059	0,4910	
	16	120 x 120	778,4	6,889	0,4306	
	25	96 x 96	510,3	10,51	0,4204	
960 x 960	4	480 x 480	46.167,5	1,783	0,4460	0,0005986
	9	320 x 320	20.203,6	4,075	0,4528	
	16	240 x 240	11.728,0	7,019	0,4387	
	25	192 x 192	7.882,1	10,44	0,4176	

As figuras 29 e 30 a seguir apresentam os gráficos dos valores do “speedup” e da eficiência em função da quantidade de processadores utilizados no processamento. O domínio foi particionado na mesma quantidade dos processadores utilizados no processamento, isto é, cada subdomínio foi enviado para um processador.

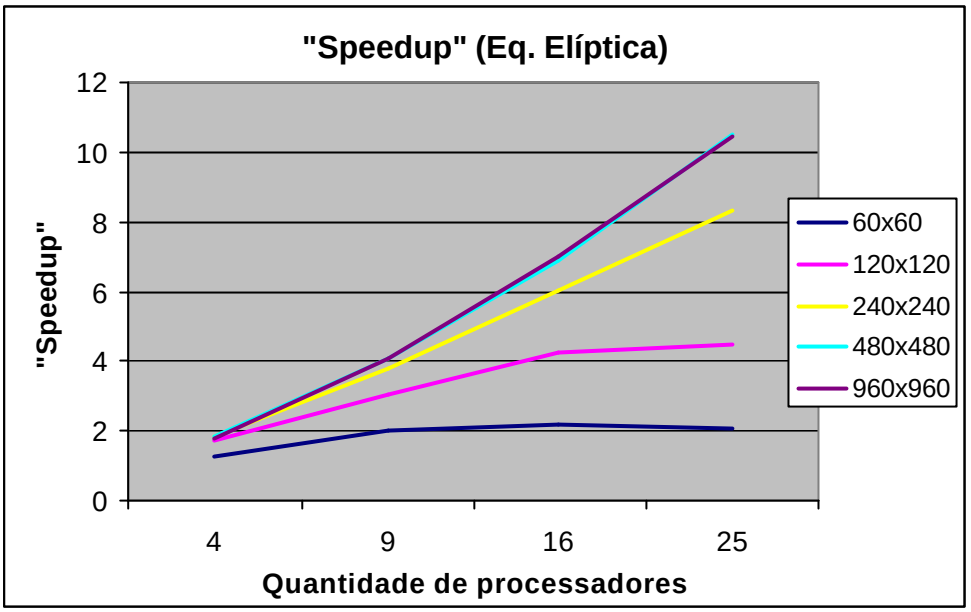


Figura 29 - “Speedup” da resolução da equação elíptica

Pode-se ver que subdomínios com mais pontos apresentam “speedup” próximo do linear. Os subdomínios com menos pontos apresentaram inclusive saturação em seu “speedup”.

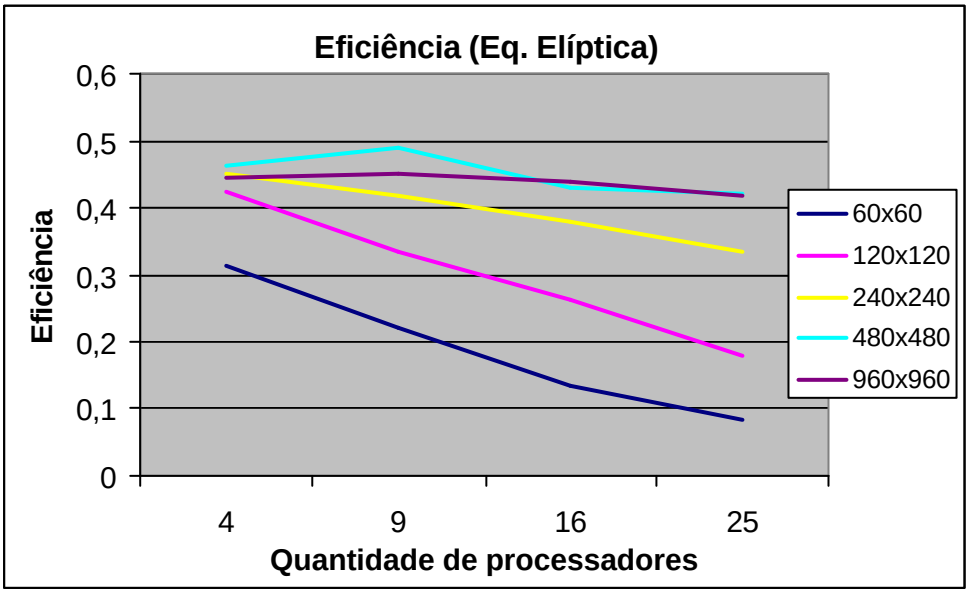


Figura 30 - Eficiência da resolução da equação elíptica

O gráfico da figura 30 mostra que a eficiência tende a cair quando se aumenta a quantidade de processadores, com a conseqüente diminuição dos subdomínios, e que o resolutor obteve melhor desempenho com os subdomínios maiores.

O gráfico a seguir mostra a escalabilidade. A série 1 mostra como o "speedup" aumentou quando a carga passou de 60 x 60 para 120 x 120, e a quantidade de processadores passou de 4 para 16, isto é, um aumento quadrático na carga e na quantidade de processadores. A série 2 mostra o "speedup" quando passou de 120 x 120 para 240 x 240, e a quantidade de processadores passou de 4 para 16.

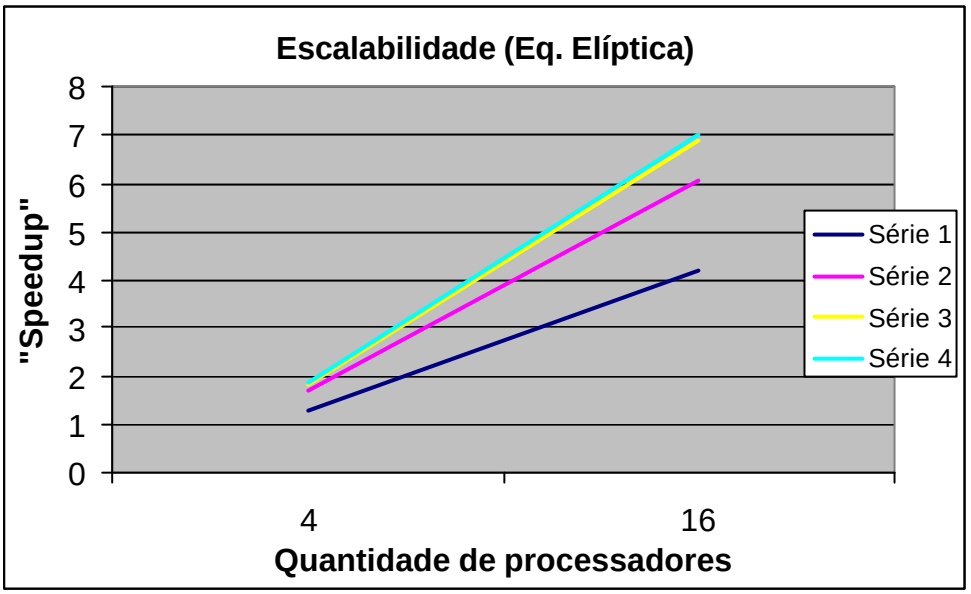


Figura 31 - Escalabilidade da resolução da equação elíptica

3.8.3 - Equação Hiperbólica

Usando agora a equação da onda de 2.8.3 com o programa paralelo, dividindo-se o domínio em 9, 16, 25 e 36 subdomínios, obtêm-se os seguintes resultados.

Tabela 10 - Métricas da paralelização do Hopscotch

Tamanho domínio	*q*	Tamanho subdom.	Tempo (s)	"Speedup"	Eficiência	erro
60 x 60	4	30 x 30	0,1728	0,2128	0,05320	0,1429
	9	20 x 20	0,2192	0,1677	0,01863	
	16	15 x 15	0,3279	0,1121	0,0007006	
	25	12 x 12	0,8241	0,04462	0,0001785	
120 x 120	4	60 x 60	0,6217	0,5866	0,1466	0,1311
	9	40 x 40	0,6009	0,6069	0,06743	
	16	30 x 30	0,6457	0,5648	0,03530	
	25	24 x 24	1,054	0,3460	0,01384	

Tamanho domínio	*q*	Tamanho subdom.	Tempo (s)	"Speedup"	Eficiência	erro
240 x 240	4	120 x 120	3,323	1,054	0,2635	0,1573
	9	80 x 80	2,324	1,507	0,1674	
	16	60 x 60	2,205	1,589	0,09931	
	25	48 x 48	2,300	1,523	0,06092	
480 x 480	4	240 x 240	21,53	1,298	0,3245	0,2264
	9	160 x 160	13,31	2,010	0,2233	
	16	120 x 120	10,64	2,627	0,1642	
	25	96 x 96	8,484	3,294	0,1318	
960 x 960	4	480 x 480	149,9	1,365	0,3412	0,2146
	9	320 x 320	78,25	2,616	0,2907	
	16	240 x 240	54,34	3,767	0,2354	
	25	192 x 192	41,01	4,991	0,1996	

As figuras 32 e 33 a seguir apresentam os gráficos dos valores do "speedup" e da eficiência em função da quantidade de processadores utilizados no processamento. O domínio foi particionado na mesma quantidade dos processadores utilizados no processamento, isto é, cada subdomínio foi enviado para um processador, $p = q$.

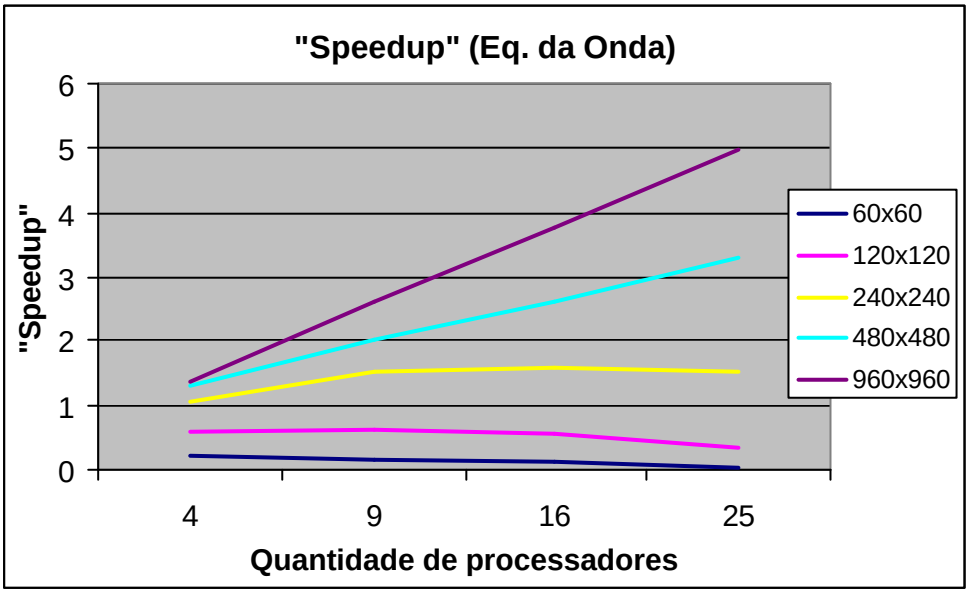


Figura 32 - "Speedup" da resolução da equação da onda

Pode-se ver que subdomínios os domínios que contêm mais pontos, 480 x 480 e 960 x 960, apresentam "speedup" próximo do linear. Os subdomínios que contêm menos pontos apresentaram inclusive saturação em sua "speedup".

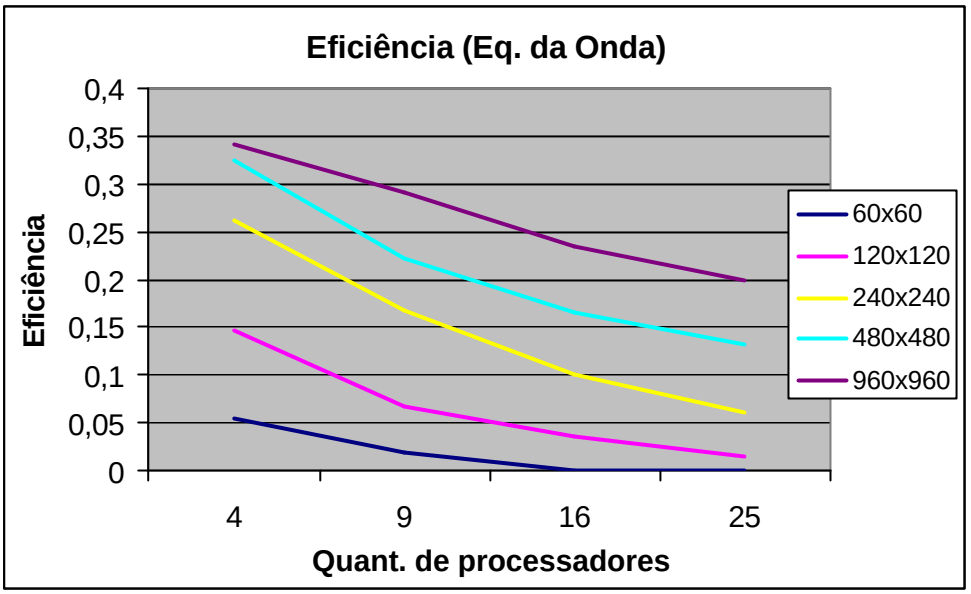


Figura 33 - Eficiência da resolução da equação da onda

O gráfico da figura 33 mostra que a eficiência tende a cair quando se aumenta a quantidade de processadores, com a conseqüente diminuição do tamanho dos subdomínios, e que o resolutor obteve melhor desempenho com os subdomínios maiores.

O gráfico a seguir mostra a escalabilidade. A série 1 mostra como o "speedup" aumentou quando a carga passou de 60 x 60 para 120 x 120, e a quantidade de processadores passou de 4 para 16, isto é, um aumento quadrático na carga e na quantidade de processadores. A série 2 mostra o "speedup" quando passou de 120 x 120 para 240 x 240, e a quantidade de processadores passou de 4 para 16.

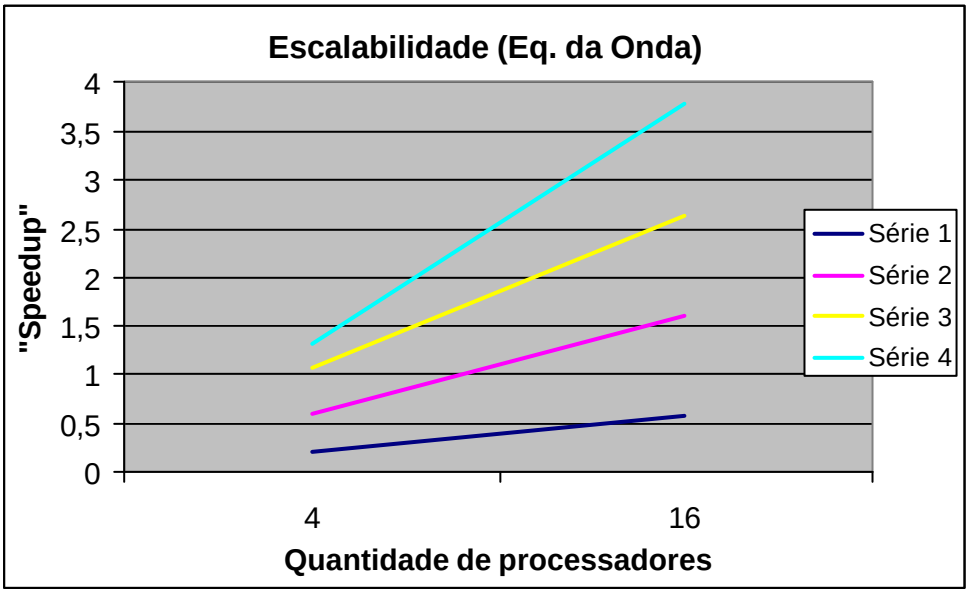


Figura 34 - Escalabilidade da resolução da equação da onda

CAPÍTULO 4

O REFINAMENTO ADAPTATIVO

O refinamento é realizado para aumentar a precisão dos cálculos quando do uso de métodos numéricos.

4.1 - TIPOS DE REFINAMENTO

Existem alguns tipos de refinamento normalmente usados, o refinamento-*h*, o refinamento-*p* e o refinamento-*r*.

4.1.1 - Refinamento-*h*

O refinamento-*h* aumenta a precisão aumentando o número de pontos do domínio, diminuindo com isso o espaçamento entre os pontos. Isto pode ser feito em todo o domínio ou só nas regiões que apresentem níveis de erro superiores ao calculado por algum indicador.

Para atingir melhores níveis de precisão durante o processo de solução numérica, usa-se o refinamento do domínio. Mas refinar todo o domínio pode ser muito custoso, pois aumenta muito o número de pontos da malha. O refinamento adaptativo aumenta a quantidade de pontos somente onde é necessário. Esta situação acontece quando a solução muda rapidamente em pequenas regiões do domínio, permanecendo estável no restante.

Desde o artigo de Berger e Oliger [9], o refinamento adaptativo (*adaptive mesh refinement*-AMR) para malhas estruturadas vem se tornando popular. Suas áreas de aplicação são numerosas e sua eficiência tem sido demonstrada. Todavia, lidar com AMR é uma tarefa difícil, o que normalmente mantém as pessoas afastadas dessa técnica. Isso é verdade, pois AMR é intrinsicamente um método dinâmico que exige gerenciamento dinâmico da memória. A figura 35 a seguir ilustra o AMR.

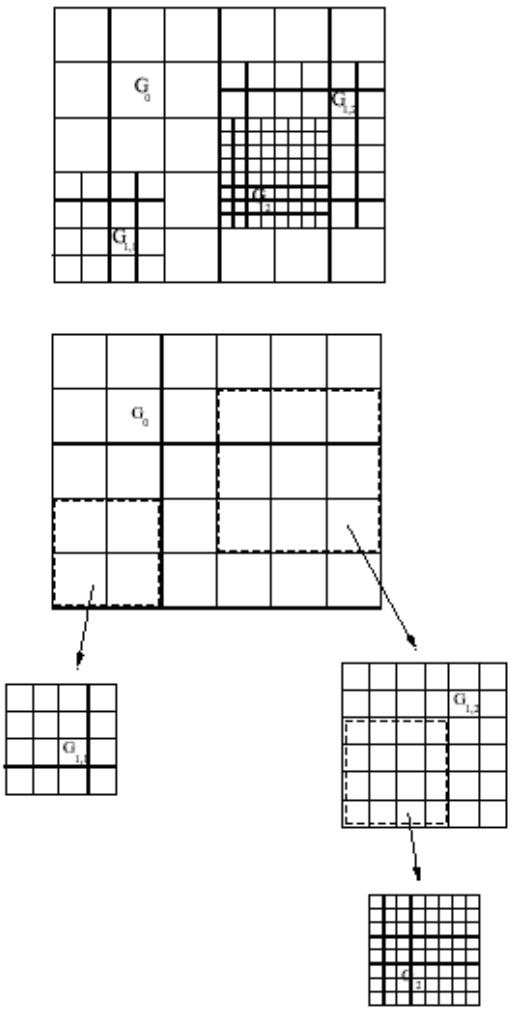


Figura 35 - Malha adaptativa de Berger e Oliger [9]

Existem técnicas que combinam o refinamento da malha (refinamento-*h*) com variação da ordem de discretização (refinamento-*p*) [21].

4.1.2 - Refinamento-*p*

O refinamento-*p* usa funções de mais alto grau para representar os objetos para obter maior precisão, sem alterar a quantidade de pontos da malha. Comparado com o refinamento-*h*, o refinamento-*p* tem as vantagens de ter convergência mais rápida e melhor precisão.

4.1.3 - Refinamento-*r*

O refinamento-*r* muda o posicionamento dos pontos, sem acrescentar novos pontos, de acordo com o nível de erro apresentado em regiões do domínio.

4.2 - O REFINAMENTO-*h* UTILIZADO

O refinamento utilizado aqui difere daquele apresentado por Berger e Oliger [9]. Primeiramente o domínio é dividido em regiões, com um refinamento inicial. Partindo desse

refinamento inicial, as regiões podem ser refinadas, até um máximo definido pelo usuário dependendo da aplicação, e desrefinadas até alcançar o grau de refinamento inicial. A figura 36 seguir apresenta um domínio com refinamento inicial 12 x 12 dividido em 16 regiões 3 x 3. Neste exemplo, três regiões estão refinadas 4x e quatro regiões estão refinadas 2x, ficando as demais com seu refinamento inicial.

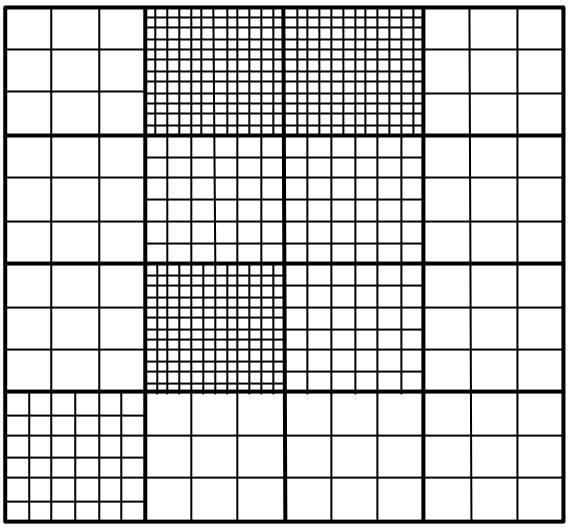


Figura 36 - Particionamento e refinamento de um domínio com refinamento inicial 12 x 12

O refinamento de uma região é feito usando *splines* cúbicas, que foi o método que melhor resultado produziu. Foram experimentados a interpolação bi-linear, que produziu resultados insatisfatórios, e a interpolação utilizando polinômios de Lagrange, que funciona bem para poucos pontos, mas exibe comportamento oscilatório quando o número de pontos aumenta, inviabilizando seu uso. Normalmente a aproximação obtida com seis pontos já é inadequada.

O método para a interpolação foi usar splines cúbicas, que conectam pontos sucessivos por curvas cúbicas unidas de tal modo que a continuidade da função, assim como a continuidade das duas primeiras derivadas, é assegurada. Assim, a função interpoladora é seccionalmente cúbica.

A idéia é ajustar trechos de funções nos intervalos, da forma:

$$S(x) = \begin{cases} s_1(x) \;\; se \;\; x_1 \le x < x \\ s_2(x) \;\; se \;\; x_2 \le x < x_3 \\ \vdots \\ s_{n-1}(x) \;\; x_{n-1} \le x < x_n \end{cases}$$

onde s_i é uma função polinomial de grau três, construída para cada intervalo, definida por

$$s_i(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d \qquad \text{para } i = 1, 2, \ldots, n-1 \qquad (1)$$

A figura 37 a seguir ilustra a construção de spline cúbica [43].

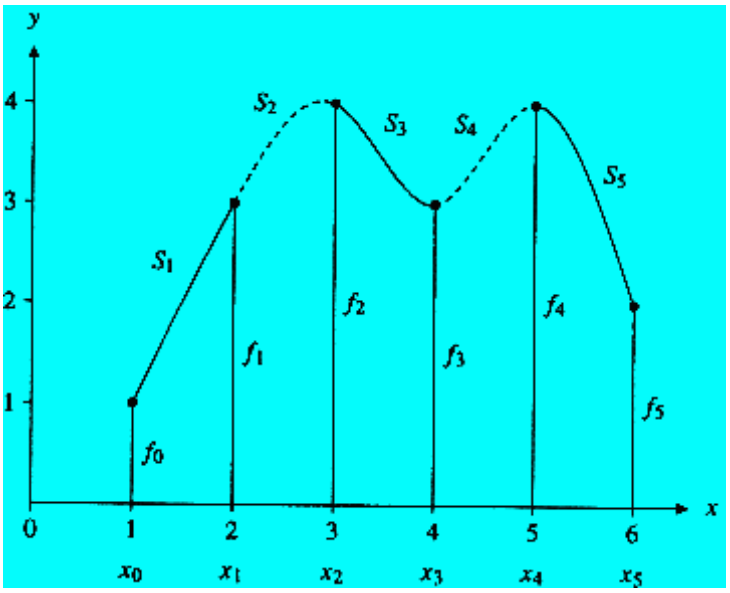


Figura 37 - Exemplo esquemático de uma spline cúbica construída nos intervalos [x_{j-1},x_j], cada função S_j sendo uma cúbica [43]

Para garantir uma colagem suave e contínua nas junções entre dois subintervalos, garantindo a continuidade em todo o intervalo, algumas condições devem ser seguidas pelos s_i :

$$s_i(x_i) = s_{i+1}(x_i)$$

$$s_1'(x_i) = s_{i+1}'(x_i)$$

$$s_1''(x_i) = s_{i+1}''(x_i)$$

Para cada s_i é preciso determinar seus quatro coeficientes a_i, b_i, c_i e d_i . Com a condição (1) mais essas três condições sobre os polinômios $s_i(x)$, obtemos um sistema linear de n equações com $n + 2$ incógnitas. Esse sistema é tridiagonal, o que permite uma solução rápida. Devido à falta de equações para determinar as condições em s_0 e em s_n nos extremos, é necessário fixar duas incógnitas. A maneira de fixar implica em uma determinada forma de *spline*. Neste trabalho é usada a *spline* cúbica com condição "not-a knot" nas extremidades, o que requer que a terceira derivada da *spline* seja contínua em x_1 e x_{n-1}. Esta condição mantém a curva cúbica nas extremidades.

Essa última consideração é importante porque, no caso de refinamento, a camada de pontos mais próxima aos limites da região será extrapolada. Na figura 38 a seguir, os pontos a serem interpolados estão dentro da área definida pela linha pontilhada, e os pontos a serem extrapolados são aqueles situados fora da área definida pela linha pontilhada.

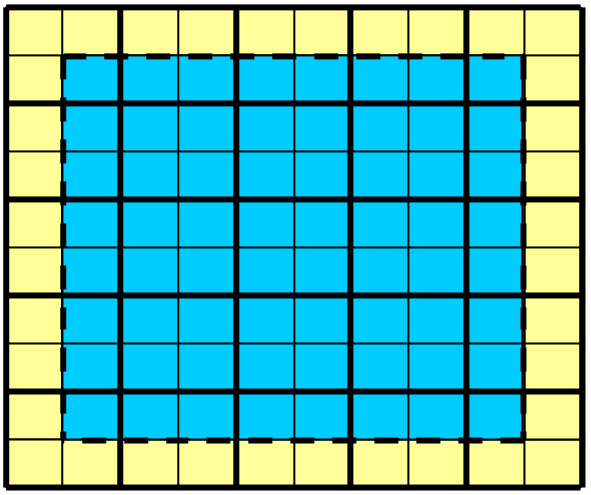


Figura 38 - Um domínio dividido em 5x5 células, representado pelas linhas grossas. Um fator de refinamento 2 passa o domínio para 10x10 células. As células adicionais foram criadas pela adição das linhas finas.

A correspondência entre os pontos da malha refinada e da malha antes do refinamento é a seguinte:

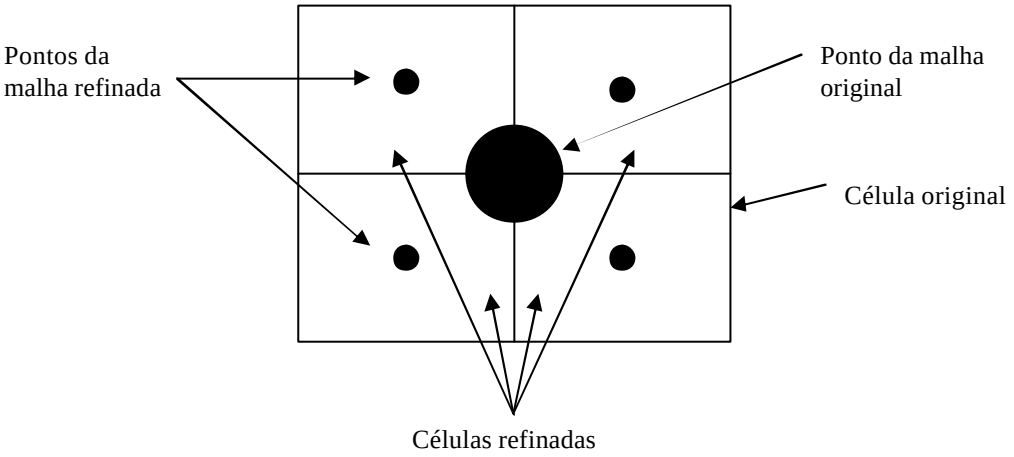


Figura 39 - Correspondência entre os pontos da malhas refinada e da malha origem

As figuras 40 e 41 a seguir ilustram como é realizado o refinamento dos subdomínios. Nessas figuras, os ponto amarelos são os pontos da malha original (x_{antigo},y_{antigo}), e os pontos verdes são os pontos refinados (x_{novo},y_{novo}). Os pontos vermelhos são pontos auxiliares.

Primeiramente são calculadas as posições (x_{novo},y_{novo}) dos pontos refinados. Depois, a partir do valor da EDP em (x_{antigo},y_{antigo}), o valor da EDP é calculado em cada (x_{novo},y_{antigo}), que são os pontos auxiliares, em vermelho, usando spline, para cada linha y_{antigo}. Após esse passo, o processo fica como ilustrado na figura 37.

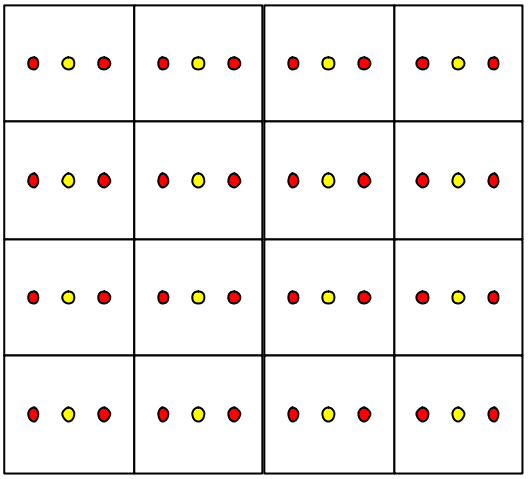


Figura 40 - Processo de refinamento de um subdomínio – pontos auxiliares

Finalmente, a partir do valor da EDP nos pontos auxiliares (x_{novo},y_{antigo}), usando todos os pontos x_{novo} de cada coluna y_{novo} para encontrar, usando spline, o valor da EDP nos pontos (x_{novo},y_{novo}), que são os pontos em verde na figura 38.

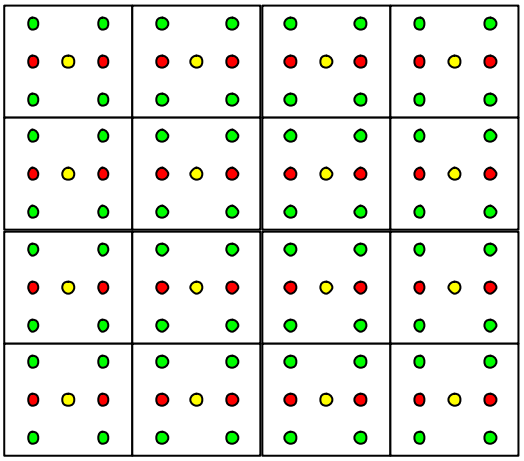


Figura 41 - Processo de refinamento de um subdomínio – pontos refinados

4.3 - COMUNICAÇÃO ENTRE SUBDOMÍNIOS COM REFINAMENTOS DIFERENTES

O subdomínio menos refinado apenas envia e recebe as franjas. O subdomínio mais refinado executa as operações de desrefinamento na franja enviada ou de refinamento na franja recebida, usando *spline*, conforme ilustra a figura 42 a seguir.

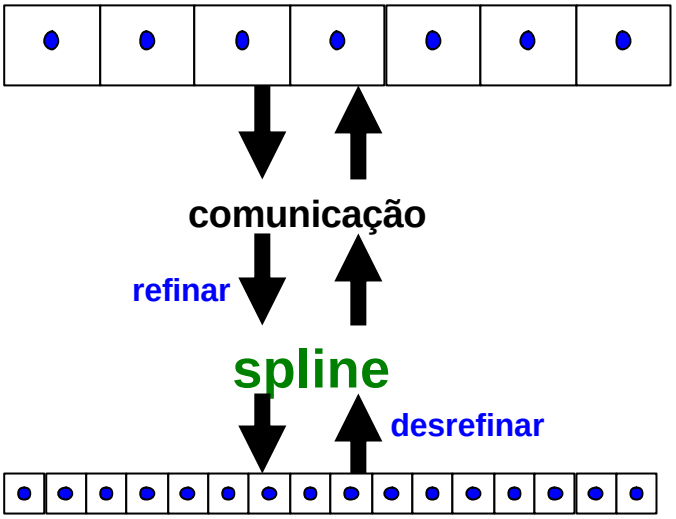


Figura 42 - Comunicação entre franjas com diferentes níveis de refinamento

4.4 - ROTINAS USADAS PARA O REFINAMENTO E O DESREFINAMENTO

As rotinas para efetuar o refinamento e o desrefinamento foram obtidas de [12]. São elas:

a) **Rotina *tridiagonal*** : resolve o sistema cuja matriz é tridiagonal;

b) **Rotina *cubic_nak***: determina os coeficientes para uma *spline* cúbica "not-a-knot" a partir de um conjunto dado de pontos; e

c) **otina *spline_eval***: Calcula uma *spline* cúbica em um único valor da variável independente para os coeficientes obtidos na rotina *cubic_nak*.

4.5 - CONSIDERAÇÕES SOBRE A CONVERGÊNCIA

Ao ser refinado, o espaçamento h entre os pontos diminui. Por isso, o $?t$ deve ser diminuído para que a relação $\frac{?t}{h^2}$ não se altere, isto é, para que a convergência continue a ser verificada em todas as regiões do domínio.

CAPÍTULO 5

PARTICIONAMENTO DO DOMÍNIO E BALANCEAMENTO DE CARGA

O processamento paralelo requer que o domínio seja dividido em tantas partes quantos sejam os processadores disponíveis para a execução. Para maximizar o desempenho da aplicação paralela, é de vital importância que a carga entre os processadores esteja balanceada.

O balanceamento de carga é a equalização da carga destinada aos processadores. Ele maximiza o desempenho da aplicação mantendo o tempo ocioso e a comunicação entre os processadores os mais baixos que seja possível. Em aplicações com carga constante, um balanceamento de carga estático pode ser realizado como pré-processamento. Outras aplicações, tais como resolução de EDP usando malhas adaptativas, têm distribuição de carga imprevisível durante o processamento, requerendo assim um balanceamento de carga dinâmico, que equalize a decomposição do domínio entre os processadores à medida que o processamento avança.

Um importante fato a ser levado em conta pelos códigos de particionamento é que a tarefa de balancear igualmente a carga entre os processadores enquanto minimiza as comunicações entre eles é um problema NP-completo [2, 99]. Assim, um código de particionamento com boa qualidade deve ter as seguintes características:

a) Obter um balanceamento aproximado da carga entre os processadores; e

b) Manter as comunicações em nível o mais baixo possível.

Para o caso do balanceamento dinâmico, caso em que o balanceamento é feito sempre que necessário durante a execução de uma aplicação, mais uma característica é desejável: a rapidez no cálculo do balanceamento.

5.1 - PARTICIONAMENTO DE MALHAS IRREGULARES

O particionamento de corte mínimo de malhas irregulares é um problema NP-completo [28], exaustivamente estudado nos últimos anos. Existem vários métodos de particionamento, e vários pacotes disponíveis na internet para utilização.

5.1.1 - Métodos Geométricos

O particionamento geométrico [20] é um método antigo, conceitualmente simples, para gerar decomposições de modo rápido e econômico. Esses métodos movimentam regiões

do domínio entre os processadores considerando apenas a posição geométrica dos nós da malha.

Os métodos geométricos têm vantagens e desvantagens. Eles são efetivos quando somente a localização geométrica é importante ou quando a conectividade do grafo a ser particionado não está disponível. Como a comunicação não é minimizada, os particionadores geométricos podem provocar um alto custo de comunicação se comparados aos particionadores mais modernos.

Uma das razões para a relativamente baixa qualidade das partições geradas por algoritmos baseados nos métodos geométricos é que eles não consideram qualquer informação de conectividade do grafo a ser particionado.

5.1.1.1 - Bisecção Recursiva

Desde que na prática todas as malhas estão dentro de um domínio físico e têm coordenadas associadas, é usada a divisão do domínio pela localização física dos pontos [82, 89].

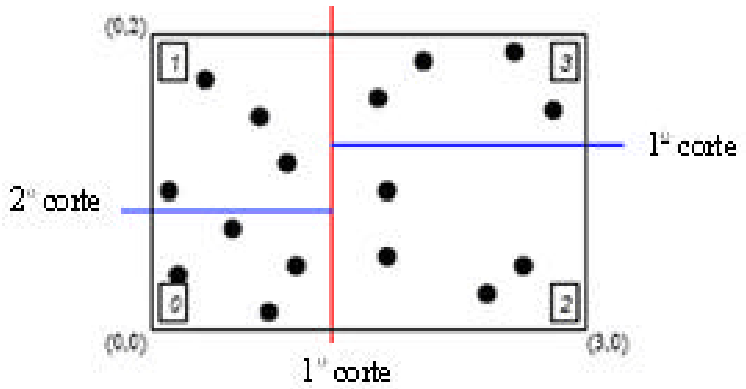


Figura 43 - Planos de corte (esq) a árvore de corte associada para bisecção recursiva. Pontos são os objetos a serem balanceados; os cortes são mostrados com linhas e nós na árvore.

O método de Bisecção Recursiva (RB) calcula um plano de corte que divide uma região em duas subregiões, cada uma com metade do esforço computacional. Recursivamente vai dividindo os subdomínios resultantes até uma condição de parada. Dois métodos derivam diretamente do RB. Um é o método de Bisecção Recursiva Coordenada (RCB), que usa planos de corte que são ortogonais aos eixos coordenados

Vantagens:

a) É simples, rápido e não consome muitos recursos computacionais;

b) Cria subdomínios regulares;

c) Pode ser usado em aplicações estruturadas ou não; e

d) Todos os processadores podem facilmente conhecer toda a decomposição.

Desvantagens:

a) Não permite o controle dos custos de comunicação;

b) Pode gerar subdomínios desconectados;

c) A qualidade da partição é mediana; e

d) Necessita das coordenadas geométricas.

O RCB é usado em AMR, em simulação de partículas, simulações de choque e detecção de contato e na apresentação gráfica paralela.

O outro é o método de Bisecção Recursiva Inercial (RIB), que calcula cortes ortogonais aos principais eixos de inércia [82, 89].

5.1.1.2 - "Space-Filling Curve" (SFC)

No particionamento SFC, as coordenadas de um objeto são convertidas para uma chave SFC representando a posição desse objeto ao longo da SFC. A ordenação das chaves fornece uma ordenação linear dos elementos, que preenche completamente o domínio. A linha ordenada é então dividida em pedaços balanceados para serem enviados aos processadores.

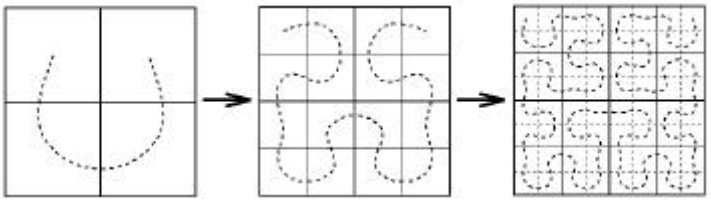


Figura 44 - Particionamento SFC [67].

Vantagens:

a) É simples, rápido e barato;

b) Guarda a localização geométrica dos elementos; e

c) A ordenação linear dos elementos pode melhorar o desempenho da memória cache.

Desvantagens:

a) Não controla explicitamente os custos de comunicação;

b) Pode gerar subdomínios desconectados;

c) Freqüentemente gera partições de pior qualidade do que o RCB; e

d) Necessita das coordenadas geométricas.

Aplicações do SFC podem ser encontradas em [31, 61, 68, 72,100].

5.1.2 - Método Multinível

Esses métodos são considerados os mais efetivos para simulações envolvendo EDP em uma malha irregular. Os nós representam os dados a serem particionados, por exemplo,

elementos, pontos ou linha de matrizes. Os arcos representam relacionamentos entre os nós, A quantidade de arcos cortados por uma fronteira de subdomínio dá a idéia do volume de comunicação necessária durante a execução. [41, 47, 48]

O objetivo do particionador é alocar cargas iguais às partições e ao mesmo tempo minimizar o custo de comunicação.

A idéia dessa abordagem é captar a informação sobre a conectividade de um grafo muito grande (> 1 milhão de nós) usando uma série de grafos com menos nós. Um algoritmo baseado na abordagem multinível tem normalmente 3 fases: desrefinamento, particionamento e refinamento.

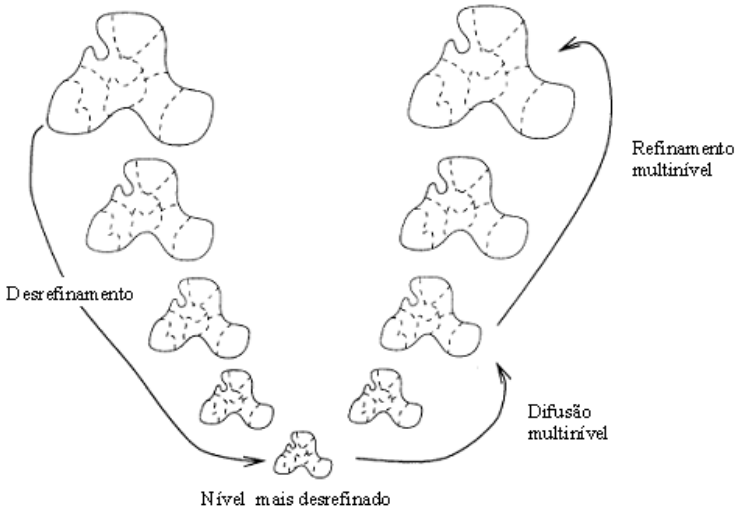


Figura 45 - Método multinível [78]

Usa estratégia difusiva, que é um agoritmo iterativo [49] que, em cada passo, troca fração fixa da diferença de carga entre dois processadores vizinhos. Quando essas operações locais são usadas, a distribuição de carga convergirá para uma distribuição globalmente balanceada da carga. A eficiência desse algoritmo depende do parâmetro de difusão a que determina o tamanho da fração da carga a ser trocada em cada passo.

Vantagens do Método Multinível:

a) Cria partições de boa qualidade na maioria das aplicações;
b) Controla explicitamente os custos de comunicação; e
c) É largamente utilizado para particionamento estático.

Desvantagens:

Demanda mais recursos computacionais que os métodos geométricos; e

5.1.3 - Heurísticas

A seguir estão algumas heurísticas empregadas no particionamento de grafos.

5.1.3.1 - Kernighan-Lin (KL)

Seu objetivo é particionar um grafo de uma maneira que minimize as conexões entre os subdomínios criados.

Ele funciona trocando nós entre as duas bisecções que diminuam o valor do corte. O algoritmo pára quando não for capaz de achar nenhum par que diminua o valor de corte [22].

Tem as seguintes características [53]:

a) Sua complexidade é $O(n^3)$;

b) O resultado do algoritmo é usado como solução inicial para uma nova chamada do algoritmo. Isto é feito até que duas chamadas consecutivas produzam a mesma solução. Normalmente a quantidade de chamadas para que isto ocorra varia de 4 a 6 ; e

c) È uma heurística iterativa. A solução depende dos valores iniciais.

5.1.3.2 - Método Fidducia-Mattheyses

O algoritmo Fidducia-Mattheyses é uma heurística de biparticionamento de grafos. Ele é um aperfeiçoamento do algoritmo KL. Seu objetivo é minimizar o corte entre dois conjuntos de nós. Em vez de realizar a troca de par de nós, ele só move os nós que ocasionem uma máxima redução no valor de corte, mantendo o balanceamento abaixo de uma constante estabelecida [22, 56].

5.1.3.3 - "Helpful-sets"

É um algoritmo de busca local. Ele inicia com uma dada bisecção A, composta das bisecções V_1 e V_2 e busca melhorar o tamanho do corte realizando mudanças locais. O algoritmo inicia em determinados nós para procurar por *h*-helpul sets, isto é, subconjuntos de nós em V_1 ou em V_2 que diminuem o valor do corte por *h* se movidos para o outro lado. Se tal conjunto for encontrado, digamos, em V_1, ele é movido para V_2. O algoritmo então inicia a busca por um conjunto de nós em um V_2 sobrecarregado que seja grande o suficiente para rebalancear a bisecção mas que aumente o valor do corte por não mais que (*h*-1) arcos. Se tal conjunto for achado, ele é movido para V_1 e o processo é iniciado novamente [22].

5.1.4 - Método Espectral

Um método espectral de particionamento usa os autovetores da matriz de adjacência ou da matriz laplaciana do grafo para construir uma representação geométrica, que é então heuristicamente particionada [2, 73, 82].

5.1.5 - Metaheurísticas

Uma metaheurística é um método heurístico para resolver de forma genérica problemas de otimização que não disponham de algoritmo que os resolvam em tempo polinomial. A seguir são apresentadas as metaheurísticas usadas para resolver o problema de particionamento.

5.1.5.1 - Colônia de formigas (ACO – Ant Colony Optimization))

Formigas são capazes de encontrar a rota mais curta entre uma fonte de alimento e o seu ninho sem o uso de informações visuais, sendo capazes também de se adaptar a mudanças no meio. Foi descoberto que, para trocar informação sobre qual caminho deve ser seguido, as formigas se comunicam umas com as outras por meio de trilhas de feromônio. Quanto mais formigas seguirem uma trilha, mais atrativa esta trilha se tomará para ser seguida por outros indivíduos. Este processo pode ser descrito como um laço de realimentação positiva, onde a probabilidade de uma formiga escolher um caminho, aumenta com o número de formigas que passaram por aquele caminho anteriormente. Visto que o feromônio se acumula mais rapidamente nos caminhos mais curtos, e que as formigas preferem seguir trilhas com grandes quantidades de feromônio, após algum tempo todas as formigas convergirão para o caminho mais curto.

Utilizando esse princípio combinado com a evaporação do feromônio para evitar uma convergência prococe a uma solução ruim, foi proposto um método de otimização a ser usado em problemas NP, como é o caso de particionamento de grafos.

Em [56] os autores fazem uso do algoritmo de Colônia de Formigas aplicado à estratégia de particionamento multinível, onde em cada nível de expansão do grafo é executado o algoritmo de colônia de formigas. Em [29] o autor usa ACO em um problema de particionamento e clusterização de grafos.

5.1.5.2 - Recozimento simulado (SA -Simulated annealing)

SA é uma metaheurística para otimização que consiste numa técnica de busca local probabilística, e se fundamenta numa analogia com a termodinâmica. O processo consiste de duas etapas: na primeira a temperatura do sólido é aumentada para um valor máximo no qual ele se funde; na segunda o resfriamento deve ser realizado lentamente até que o material se solidifique, sendo acompanhado e controlado esse arrefecimento de temperatura. Nesta segunda fase, executada lentamente, os átomos que compõem o material organizam-se numa estrutura uniforme com energia mínima. Este processo é chamado recozimento (“annealing”).

De forma análoga, o algoritmo de recozimento simulado substitui a solução atual por uma solução próxima (i.e., na sua vizinhança no espaço de soluções), escolhida de acordo com uma função objetivo e com uma variável T (dita *Temperatura*, por analogia). Quanto maior for T, maior a componente aleatória que será incluída na próxima solução escolhida. À medida que o algoritmo progride, o valor de T é decrementado, começando o algoritmo a convergir para a solução ótima ou para uma solução próxima dela.

Uma das principais vantagens deste algoritmo é permitir testar soluções mais distantes da solução atual e dar mais independência do ponto inicial da pesquisa.

Em [22] os autores mostram uma aplicação do método AS no particionamento de grafos.

5.1.5.3 - Algoritmos genéticos

Algoritmos genéticos (AG) são implementados como uma simulação de computador em que uma população de representações abstratas do universo de soluções, os cromossomos (ou indivíduos), compostos por genes, é selecionada usando uma função objetivo, em busca de soluções melhores. A evolução geralmente se inicia a partir de um conjunto de cromossomos criados aleatoriamente e é realizada por meio de cruzamentos entre eles. Após o cruzamento, alguns poucos indivíduos sofrem mutação. A adaptação de cada cromossomo é avaliada por uma função objetivo, e alguns indivíduos com os maiores valores calculados pela função objetivo são selecionados para a próxima geração. A nova população então é utilizada como entrada para a próxima iteração do algoritmo.

A idéia é simples, e sua implementação não é complicada. A principal arte é a codificação da solução no cromossomo e garantir que o cruzamento produza novos cromossomos que façam parte do universo de soluções do problema.

Em [14] é apresentada uma aplicação do AG ao problema de particionamento de grafos. O cromossomo tem quantidade de genes igual ao número de nós. Cada gene corresponde a um vértice do grafo. Um gene tem valor 0 se o vértice correspondente está, por exemplo, no lado esquerdo da bisecção, e tem valor 1 caso contrário. Assim, existirá em cada cromossomo um igual número de 0s e 1s, que devem ser mantidos após o cruzamento.

5.1.6 - Pacotes Disponíveis

Existem disponíveis vários aplicativos para o particionamento de domínio. De todos os pacotes pesquisados, apenas o AGRIF foi projetado para particionar malhas regulares resultantes da discretização para a resolução de problemas por diferenças finitas. Em [7] são

apresentados vários projetos que têm por objetivo o particionamento de grafos e o balanceamento de carga. Os aplicativos pesquisados estão relacionados a seguir:

a) METIS: É um pacote para particionamento de grafos e malhas irregulares desenvolvido na Universidade de Minnesota pelo Karypis Lab, com o patrocínio do Exército dos EUA através do Army Research Laboratory. Os algoritmos de particionamento usados em Metis são baseados na técnica de particionamento multinível. [48, 49, 50, 51, 78, 79]. O pacote, juntamente com o manual [52], pode ser baixado do sítio abaixo.

Endereço do sítio: http://glaros.dtc.umn.edu/gkhome/views/metis

b) **PARTY:** A biblioteca de particionamento PARTY foi desenvolvida na Universidade de Paderborn. Usa vários métodos heurísticos de particionamento. Uma comparação de PARTY com outros pacotes de particionamento pode ser encontrada em [62]. Uma cópia do pacote [74] pode ser obtida no sítio abaixo.

Endereço do sítio: http://wwwcs.uni-paderborn.de/fachbereich/AG/monien/RESEARCH/PART/party.html

c) ZOLTAN: A biblioteca Zoltan [11, 19] é um conjunto de rotinas para aplicações em malhas irregulares, usando processamento paralelo com balanceamento dinâmico de carga. É principalmente um particionador de grafos e hipergrafos. Foi desenvolvido pelo Sandia National Laboratories. Sandia é uma agência do governo dos EUA. A Sandia Corporation, uma companhia da Lockheed Martin, gerencia a Sandia para o Departamento de Energia Nuclear do governo dos EUA. No sítio da Sandia existe uma lista com os artigos publicados sobre o desenvolvimento e uso do Zoltan. O pacote, juntamente com o manual [19], pode ser obtido do sítio abaixo.

Endereço do sítio: http://www.cs.sandia.gov/Zoltan/

d) JOSTLE: JOSTLE é um pacote para particionamento de grafos e malhas irregulares que aparecem em resolução de problemas usando elementos e volumes finitos. Foi desenvolvido no Old Royal Naval College da Universidade de Greenwich, Londres. O algoritmo usado é baseado no particionamento multinível de grafos, e o método de difusão é usado para reparticionamento [97, 98]. JOSTLE é comercializado sob o nome de NetWorks, mas uma cópia do executável e manual [96] para propósitos acadêmicos pode ser baixada do sítio abaixo.

Endereço do sítio: http://staffweb.cms.gre.ac.uk/~wc06/jostle/

e) SCOTCH: Software e bibliotecas [69, 70, 71] para particionamento de grafos, malhas e hipergrafos. Sua meta é o mapeamento estático de grafos, usando a abordagem de dividir e conquistar. Foi desenvolvido no Laboratoire Bordelais

de Recherche em Informatique, na França, que faz parte do Institut National de Recherche em Informatique et em Automatique (INRIA). Uma cópia pode ser obtida no sítio abaixo.

Endereço do sítio: http://www.labri.fr/perso/pelegrin/scotch/

f) CHACO: É um software projetado para particionar grafos [42]. Os algoritmos usados incluem particionamento inercial, espectral, Kernighan Lin (KL) e KL multinível. Antes que uma aplicação seja executada em modo paralelo, primeiro ela precisa ser decomposta em tarefas para serem atribuídas aos diversos processadores. Também foi desenvolvido no Sandia National Laboratories. Uma cópia pode ser baixada do sítio abaixo.

Endereço do sítio: http://www.sandia.gov/~bahendr/chaco.html

g) AGRIF (Adaptative Grid Refinement in Fortran): É um pacote escrito em FORTRAN 90 para a integração de um AMR dentro de um modelo de diferenças finitas. Sua característica principal é a implementação do método de AMR de *Berger-Oliger's AMR* sobre a integração do tempo em função do refinamento do grid (partes menos refinadas são integradas antes) e o procedimento de refinamento da malha. Trabalha com malhas "staggered", possibilitando que as variáveis sejam posicionadas no centro das células ou nos cantos. Foi projetado para ser usado com programas já existentes de modo a prover a eles as potencialidades do refinamento de malhas. Seu desenvolvedor é o Laboratório LMC-IMAG, em Grenoble, na França, que é parte do Institut National de Recherche em Informatique et em Automatique (INRIA). Uma cópia pode ser obtida do sítio abaixo. Necessita o compilador FORTRAN 90.

Endereço do sítio: http://www-lmc.imag.fr/IDOPT/AGRIF/

h) **OVERTURE**: É um pacote para resolver, em geometrias complexas, PDEs discretizadas por diferenças finitas e volumes finitos [13]. Usa malhas estruturadas e refinamento adaptativo. É um desenvolvimento do Lawrence Livermore National Laboratory da Universidade da Califórnia sob os auspícios do U.S. Department of Energy. Pode ser obtido a partir do sítio abaixo.

Endereço do sítio: https://computation.llnl.gov/casc/Overture/

5.2 - PARTICIONAMENTO DE MALHAS REGULARES

O particionamento da malhas regulares foi estudado recentemente por Donaldson [24, 25, 26], Christou [17], Meyer [17, 25, 26] e Yackel [24]. O particionamento ótimo, isto é, aquele que fornece balanceamento perfeito com mínimo corte, é um problema NP-completo [24]. Christou e Meyer usaram a heurística de algoritmos genéticos para encontrar bons particionamentos. Yackel transforma o problema no Problema da Mochila. Esses métodos

buscam minimizar o perímetro das partições, minimizando com isso as comunicações entre os processadores. De qualquer maneira, encontrar um particionamento satisfatório não é uma tarefa trivial, por isso demandando tempo. Se as condições do problema exigirem mudanças freqüentes de refinamento, encontrar o particionamento por métodos heurísticos pode não ser atraente, ainda mais se o processamento não for distribuído, e sim paralelo. Máquinas paralelas cujos processadores se comuniquem pelo barramento interno têm tempos de comunicação várias ordens de grandeza menores do que processadores ligados utilizando uma rede externa. Por causa disso, este trabalho propõe um método de partição que não dispenda tempo procurando minimizar o perímetro, buscando balancear a carga de modo rápido, sendo por isso indicado para uso em computadores paralelos e problemas com freqüentes mudanças das condições de refinamento.

5.3 - TIPOS DE BALANCEAMENTO

As técnicas de balanceamento mencionadas na literatura diferem de acordo com o ambiente e em como e quando o balanceamento será realizado. Um conjunto de critérios de classificação para avaliar estratégias de balanceamento é sugerido em [16]. Esses critérios são utilizados neste trabalho para classificar a estratégia de balanceamento utilizada. A tabela a seguir resume os critérios mais relevantes propostos em [16].

Tabela 11 - Critérios de Classificação

CRITÉRIO	CLASSIFICAÇÃO	
Instante do balanceamento	**Estático**: quando a distribuição de carga ocorre apenas no início da execução.	**Dinâmico**: a distribuição ocorre quando ela é necessária
Forma de distribuição	**Sob demanda**: a distribuição acontece ao longo da execução. Um coordenador distribui um bloco de tarefas para cada processador e este, quando termina a execução, recebe outro bloco de tarefas.	**Por transferência**: o balanceamento é realizado através do envio de carga de um processador sobrecarregado para um com folga.
Utilização do índice de carga externa	**Integrado**: utiliza algum índice de carga externa.	**Isolado**: não utiliza índice de carga externa.

CRITÉRIO	CLASSIFICAÇÃO	
Escopo do balanceamento de carga	**Global**: a transferência de carga pode ocorrer entre qualquer processador do sistema.	**Local**: a transferência de carga ocorre entre processadores de um mesmo grupo, disjuntos ou não.
Localização da execução do algoritmo	**Centralizado**: é executado em um único processador que é responsável por informar aos processadores envolvidos sobre a transferência de carga.	**Distribuído**: é executado localmente em cada processador.
Sincronismo	**Síncrono**: participação, ao mesmo tempo, de todos os processadores.	**Assíncrono**: o algoritmo é executado independentemente em cada processador.
Ativação do algoritmo	**Periódico**: ativado em intervalos pré-determinados de tempo	**Por evento**: ativado quando um determinado evento acontece.
Política de localização	**Não cego**: utiliza índice de carga interna na transferência	**Cego**: não utiliza índice de carga interna.

De acordo com a tabela 11, o balanceamento proposto neste trabalho é:

Dinâmico: O balanceamento é realizado durante a execução da aplicação sempre que houver alteração no refinamento dos subdomínios;

Por transferência: a carga flui de um processador com carga superior a média para um processador menos carregado;

Isolado: Não verifica a carga externa do processador nos cálculos do balanceamento;

Global: Todos os processadores trocam carga com todos os outros;

Distribuído: Cada processador executa o mesmo procedimento para o cálculo do balanceamento de carga;

Síncrono: O fluxo de carga ocorre em todos os processadores ao mesmo tempo;

Por evento: O balanceamento de carga é ativado sempre que houver uma alteração no refinamento; e

Não cego: O balanceamento é realizado observando a carga de todos os processadores participantes do processamento.

5.4 - IMPLEMENTAÇÃO DO REFINAMENTO ADAPTATIVO

As seções a seguir descrevem o refinamento adaptativo proposto neste trabalho, que foca na utilização do método Hopscotch.

5.4.1 - O Programa

O programa permite que os subdomínios tenham refinamentos diferentes conforme figura 36 da seção 4.2. Na presença de gradientes elevados na solução, a linha de ação mais comum é refinar a malha onde esses gradientes ocorrem, e deixar a malha menos refinada nas regiões cuja solução apresente gradientes pequenos [1]. No caso de uma descontinuidade, é necessário que a malha seja mais refinada nas proximidades dela, para que seja obtida a precisão requerida. Como o refinamento da malha inteira é computacionalmente dispendioso, o programa permite refinamentos diferentes em cada subdomínio.

5.4.2 - Balanceamento de Carga

O balanceamento de carga será feito retirando-se pontos de um processador sobrecarregado, chamado de fornecedor, e enviando-os para um processador ocioso, chamado de receptor. Assim, um processador pode estar lidando com mais de um subdomínio se for receptor ou pode estar lidando com apenas um subdomínio (o próprio) se for fornecedor.

Essa carga é constituída de pontos do domínio que, apesar de estarem logicamente pertencentes a um determinado subdomínio que está em um determinado processador, na verdade parte desse subdomínio estará fisicamente em outro processador. A transferência de pontos será feito por faixas do subdomínio, onde uma dimensão será mantida constante e a outra variará de modo a se ajustar à quantidade de pontos a serem transferidos.

A diferença no nível de refinamento dos subdomínios gera um desbalanceamento de carga, acarretando que o tempo total de processamento seja dado pelo processador mais carregado. Para prevenir isso, foi implementado um balanceamento de carga entre os processadores.

Esse balanceamento de carga deve levar em conta a quantidade de pontos a ser distribuída entre os processadores e o grau de refinamento do subdomínio, por causa da equação $k = a\dfrac{\Delta t}{h^2}$, que indica que ao ser reduzido h (malha mais refinada), Δt tem que ser reduzido na proporção do quadrado da redução em h, isto é, do aumento do refinamento.

Quando o refinamento adaptativo é usado, após um intervalo de execução, a malha pode ser refinada (ou desrefinada) em algum local, normalmente baseada em uma estimativa de erro. O refinamento (ou desrefinamento) pode acarretar uma grande variação na quantidade

de pontos que está atribuída a cada processador, provocando um desbalanceamento da carga, gerando, com isso, a necessidade de se realizar balanceamento dinâmico. O desbalanceamento da carga pode ser também provocada por diferentes valores de passo no tempo, diferentes ordens de discretização ou por características não-lineares do material sendo simulado.

Em geral, os algoritmos de balanceamento dinâmico devem satisfazer os seguintes objetivos [45]:

a) Rebalancear a carga de cada processador com rapidez e escalabilidade; e

b) Minimizar o custo de comunicação após o rebalanceamento.

5.5 - MÉTRICAS

Ao lado das métricas já definidas em 2.5.5 em 3.4, será definida a seguir o indicador que está relacionado com o balanceamento dinâmico.

5.5.1 - Índice de desbalanceamento (ID)

O índice de desbalanceamento (ID) será medido pela relação:

$$ID = 1 - \frac{car_{min}}{car_{max}}$$

onde $car_{\min}$ é a menor carga presente em um processador e $car_{\max}$ é a maior carga presente em um processador. Quando ID = 0, o balanceamento é o ideal.

5.5.2 - Índice médio de desbalanceamento (IMD)

O índice médio de desbalanceamento é uma média do desbalanceamento existente ao longo do processamento da EDP. É medido pela expressão

$$IMD = \frac{\sum_{i=1}^{b} ID_i}{b}$$

onde b é a quantidade de balanceamentos executados durante a resolução da EDP.

5.5.3 - Ganho obtido com o balanceamento

O ganho obtido com o balanceamento é definido para aplicações síncronas, e é definido pela expressão

$$G = \frac{t_{ps}}{t_{pc}},$$

onde G é o ganho obtido por balancear a carga. t_{ps} é o tempo de execução do programa paralelo sem balanceamento de carga e t_{pc} é o tempo de execução do programa paralelo com balanceamento de carga.

5.5.4 - O "speedup" e a eficiência da versão com balanceamento de carga

As fórmulas de cálculo do "speedup" e da eficiência foram definidas em 3.1.1 e 3.1.2 respectivamente, e para a versão com balanceamento de carga são, respectivamente,

$$S = \frac{t_S}{t_{pc}} \quad , \quad EBD = \frac{t_S}{t_{pc}\, p},$$

onde t_s é o tempo de execução do programa seqüencial, t_{pc} é o tempo de execução do programa paralelo com balanceamento de carga e p é a quantidade de processadores.

O valor de t_S a ser usado deve ser obtido a partir de um domínio de tamanho equivalente àquele que seria obtido considerando-se o subdomínio mais refinado. Por exemplo, para um domínio 30 x 30 dividido em 9 processadores, cada subdomínio terá o tamanho 10 x 10, e um domínio 60 x 60 dividido em 9 processadores teria subdomínios de tamanho 20 x 20. Se um dos subdomínios do domínio 30 x 30 for refinado um nível, isto é, 2x, ele passará a ter tamanho 20 x 20 e por isso será comparado com a versão seqüencial de tamanho de domínio 60 x 60 para o cálculo do "speedup" e da eficiência.

CAPÍTULO 6

BALANCEAMENTO PROPOSTO

O balanceamento proposto neste trabalho visa principalmente ao método Hopscotch, utilizando malhas regulares. O balanceamento será realizado movendo pontos de um subdomínio atribuído a um processador mais carregado para um processador com capacidade ociosa. O mapeamento do balanceamento é feito pela rotina *Prepara*.

A carga a ser balanceada constitui-se de quantidade de operações aritméticas que os processadores realizam. Deve-se notar que quando se refina um subdomínio por um fator 2, a quantidade de pontos no subdomínio refinado se tornará 4 vezes maior. Mais ainda, para manter o valor de K constante, (ver equação (5)), o Δt terá que ser reduzido por um fator 4, o que aumentará o número de iterações em um fator 4. Assim, um subdomínio refinado por um fator 2 terá uma carga computacional 16 vezes maior que o subdomínio não refinado.

O programa trata o domínio logicamente como se ele não estivesse particionado. Mesmo após o balanceamento de carga, quando pedaços do domínio são movimentados entre os processadores, logicamente é como se estivessem unidos. Isto é, fisicamente o domínio está espalhado pelos processadores, mas logicamente ele está como se não estivesse dividido.

6.1 - O PARTICIONADOR

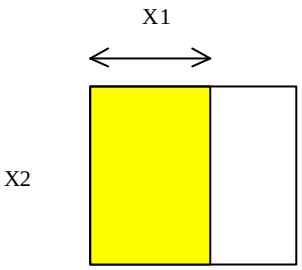
O particionador proposto neste trabalho, chamado *Prepara*, não é dependente da quantidade de nós da malha, mas sim da quantidade de subdomínios q em que o domínio é dividido. Sua complexidade de tempo é $O(q^2)$, onde q é a quantidade de subdomínios em que o domínio foi dividido, pois o particionador tem que percorrer uma matriz q x q. Ele é chamado por cada processo criado, e não depende da quantidade de nós que o subdomínio tenha, isto é, independe do grau de refinamento.

O particionador funciona segundo os passos a seguir.

a) É calculada a quantidade de pontos que cada processador deve possuir para o balanceamento ideal (ou próximo). Essa quantidade é a média aritmética de pontos por processador, $\overline{m}$.

$$\overline{m} = \frac{\sum_{i=1}^{q} q_i}{q}$$, onde q é a quantidade de partições.

b) É calculado, usando a fórmula (11) a seguir, quantas colunas vão ficar no processador (X1). A quantidade de linhas (X2) sempre permanece constante. Ver figura 46 a seguir.



onde $\lceil X1 \rceil = \frac{\overline{m}}{X2}$

Figura 46 - Cálculo da quantidade de colunas que ficam no processador

c) É encontrado quais processadores enviam (fornecedores) e quais recebem (recebedores). Aqueles processadores com quantidade de pontos acima da média serão os fornecedores e os processadores com quantidade de pontos abaixo da média serão os receptores.

d) É calculado quantos pontos cada processador identificado como recebedor deve receber para atingir a média.

e) É calculado quantas colunas essa quantidade de pontos significa no processador fornecedor. O mapa da distribuição fica armazenado na matriz *envia*.

f) As colunas são enviadas para um processador recebedor. Este procedimento se repete até que o processador fornecedor tenha atingido a média de pontos.

g) O mesmo procedimento é realizado para todos os outros processadores fornecedores.

A figura 47 a seguir ilustra um exemplo do procedimento de particionamento para dois processadores fornecedores e dois processadores receptores.

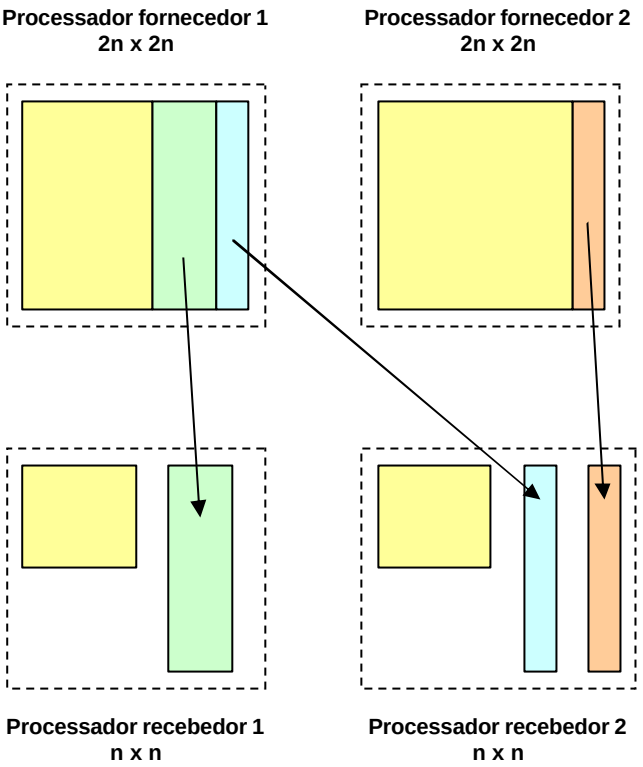


Figura 47 - Exemplo de balanceamento de carga

Esse procedimento será melhor compreendido usando um exemplo de balanceamento.

6.2 - EXEMPLO

Será usado, como exemplo ilustrativo para explicação do procedimento de balanceamento, um domínio 300 x 300, com 90.000 pontos, resolvido em um "cluster" de 9 processadores. Os processos são numerados de 0 a 8, e os subdomínios 1, 3 e 4 terão refinamento nível 2 e o subdomínio 0 terá refinamento nível 3.

6.2.1 - Subdivisão do Domínio

A divisão em 9 processadores levou à criação de 9 subdomínios de 100 x 100 pontos. A figura 48 a seguir mostra a divisão do domínio e o nível de refinamento dos subdomínios para este exemplo.

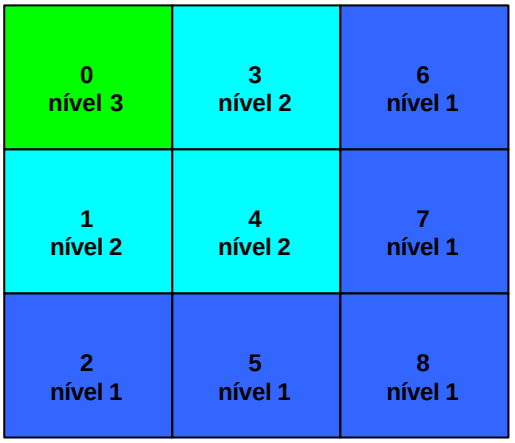


Figura 48 - Exemplo de subdivisão do domínio

A figura 49 a seguir mostra o refinamento usado em cada subdomínio juntamente com as faixas que devem ser enviadas aos processadores receptores.

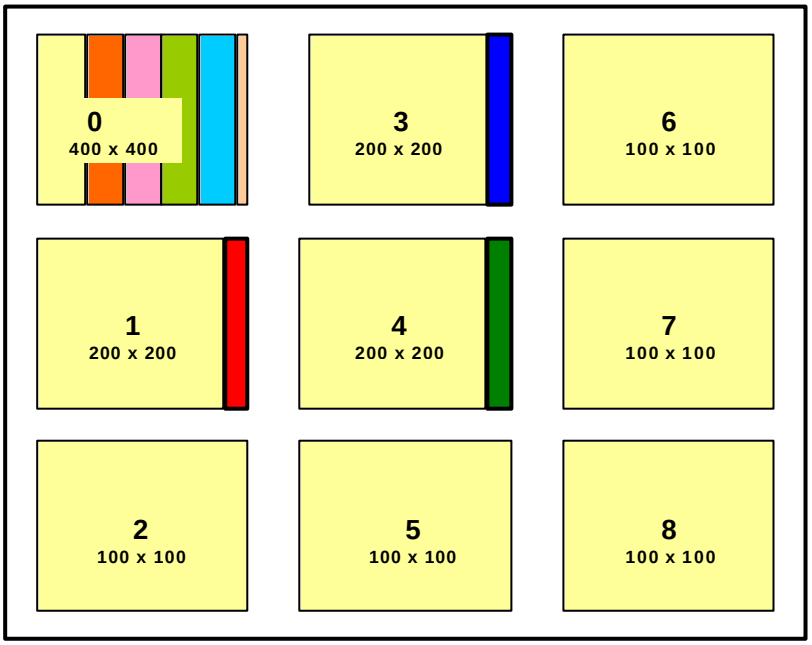


Figura 49 - Exemplo de refinamento

6.2.2 - Tráfego de Carga

O tráfego de carga para o exemplo considerado é ilustrado na figura 50 a seguir. Os processadores 0, 1 e 3 fornecem carga para os outros processadores.

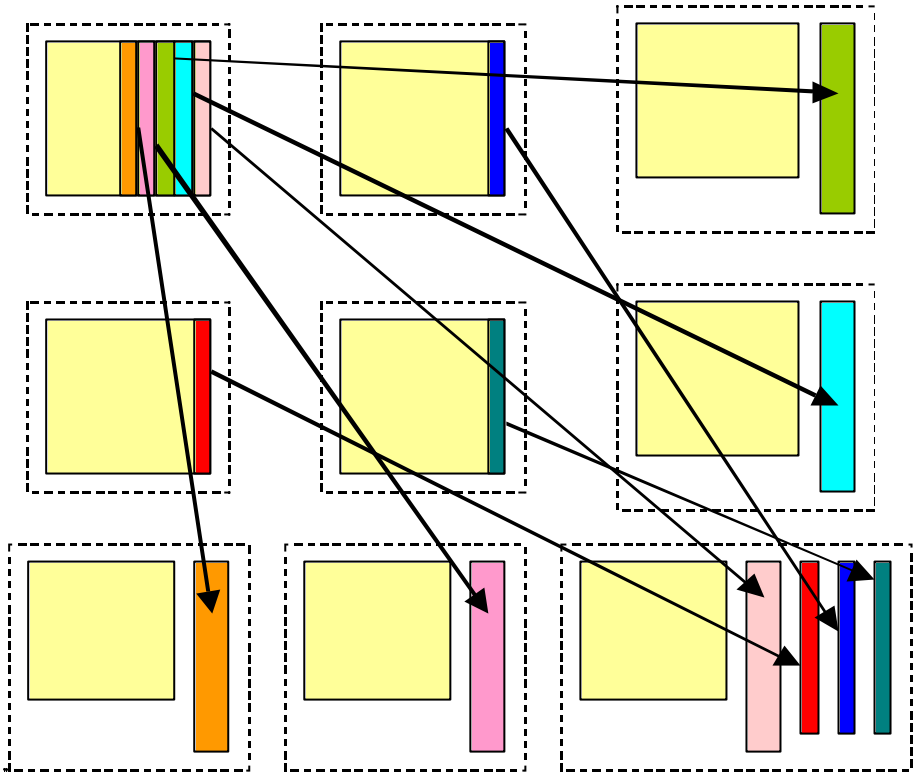


Figura 50 - Tráfego de carga

6.2.3 - Subdomínios Pertencentes a Cada Processador

Após a distribuição de carga, os processadores receptores estarão resolvendo, além do subdomínio próprio, um ou mais pedaços de subdomínios recebidos dos processadores fornecedores. A figura 51 a seguir apresenta os subdomínios que cada processador está resolvendo.

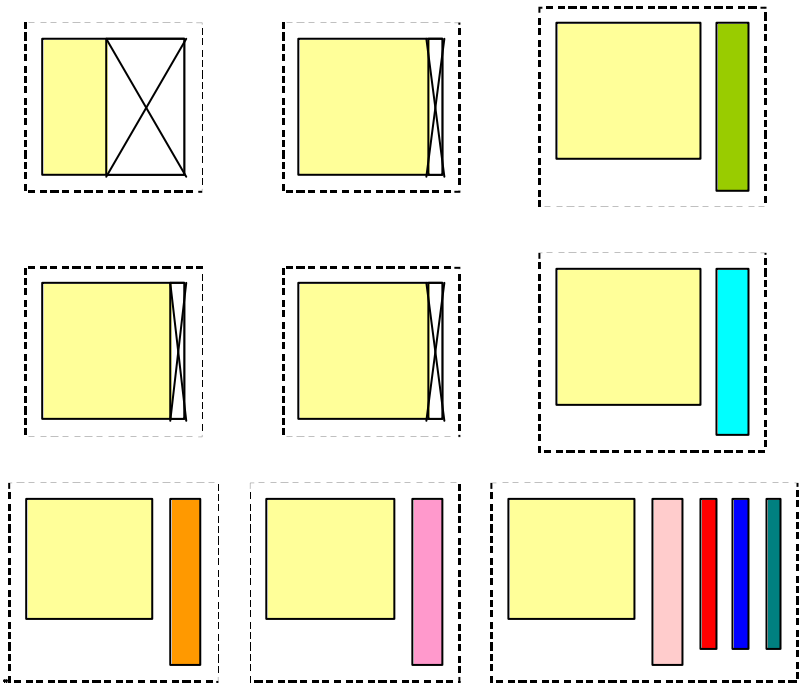


Figura 51 - Subdomínios pertencentes a cada processador

6.3 - TRÁFEGO DAS FRANJAS

A cada iteração, as franjas precisam ser comunicadas entre as regiões distribuídas nos processadores. As duas camadas de franjas são então empacotadas em um vetor e enviadas ao seu destino.

Três situações podem ocorrer:

a) Comunicação entre regiões com mesmo grau de refinamento: Nenhuma ação precisa ser tomada.

b) Comunicação entre regiões com graus de refinamento diferentes, com envio de franja de uma região menos refinada para uma mais refinada: Neste caso, a região mais refinada receberá menos pontos do que o necessário para compor sua franja. Assim, será necessário refinar a franja antes de integrá-la na região.

c) Comunicação entre regiões com grau de refinamento diferentes, com envio de franja de uma região mais refinada para uma menos refinada: Neste caso, só a quantidade de pontos necessária será comunicada.

Para dar idéia do grau de complexidade das comunicações quando pedaços de subdomínios ficam divididos em diversos processadores, a seguir é apresentado um exemplo de um balanceamento de carga usando apenas quatro processadores, onde um deles tem um grau de refinamento maior que todos, dois têm grau de refinamento médio, e um tem o menor grau de refinamento, o que obriga a que apenas um pedaço do subdomínio de maior refinamento seja movido do processador 1 para o processador 4, que contém o subdomínio de menor refinamento.

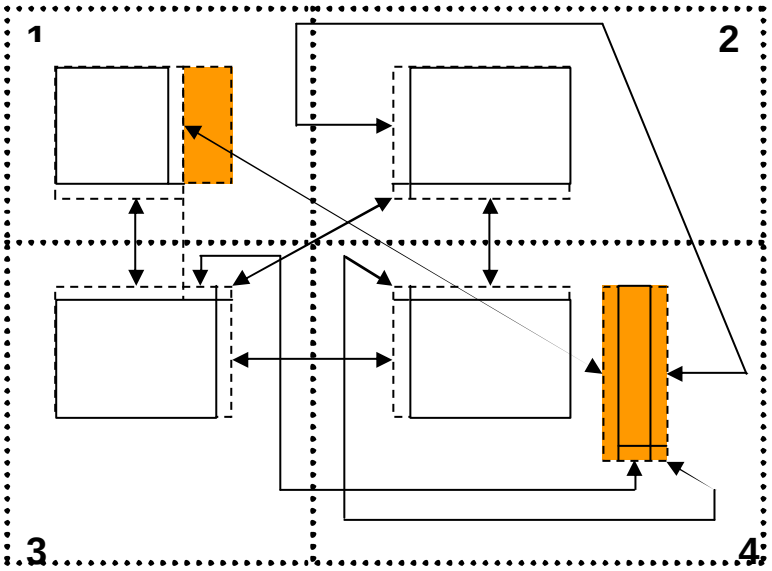


Figura 52 - Exemplo do tráfego de comunicação das franjas

6.4 - IMPLEMENTAÇÃO

O refinamento desigual entre regiões deve ser tratado na fronteira para a correta comunicação entre os pontos de ambas regiões, já que a região mais refinada tem mais pontos que a região menos refinada. Assim, a informação da parte da região menos refinada que fará parte da região mais refinada terá de ser interpolada e extrapolada nas posições que assumirá na região mais refinada.

Sendo dinâmico o grau de refinamento, a quantidade de pontos por região variará durante a simulação. Assim, a carga computacional também variará com o tempo, necessitando que um balanceamento de carga dinâmico seja implementado para aumentar a eficiência do refinamento adaptativo durante o processamento paralelo.

6.5 - O PROGRAMA

O programa inicializa os valores das diversos variáveis, vetores e matrizes. No início, todos os processadores têm a mesma carga, isto é, a mesma quantidade de pontos. O programa monta as estruturas de dados para o funcionamento paralelo, isto é, encontra os vizinhos de cada subdomínio e cria as estruturas de gerência dos diversos subdomínios que poderão ser mudados no transcorrer da computação.

Após isso, são iniciadas as iterações do método numérico. Em determinados instantes, fornecidos pelo usuário, a função *analisa* é chamada para verificar se é necessária uma mudança de refinamento em alguma região. Se for, todos os subdomínios são desrefinados, e são refeitos usando a função *retorna*. Um novo cálculo de balanceamento de carga é feito, e as colunas dos subdomínios em processadores sobrecarregados são enviadas aos processadores com tempo ocioso, usando a função *distribui*. Após, os subdomínios são refinados e o processamento continua. O fluxograma a seguir apresenta a dinâmica de funcionamento do programa.

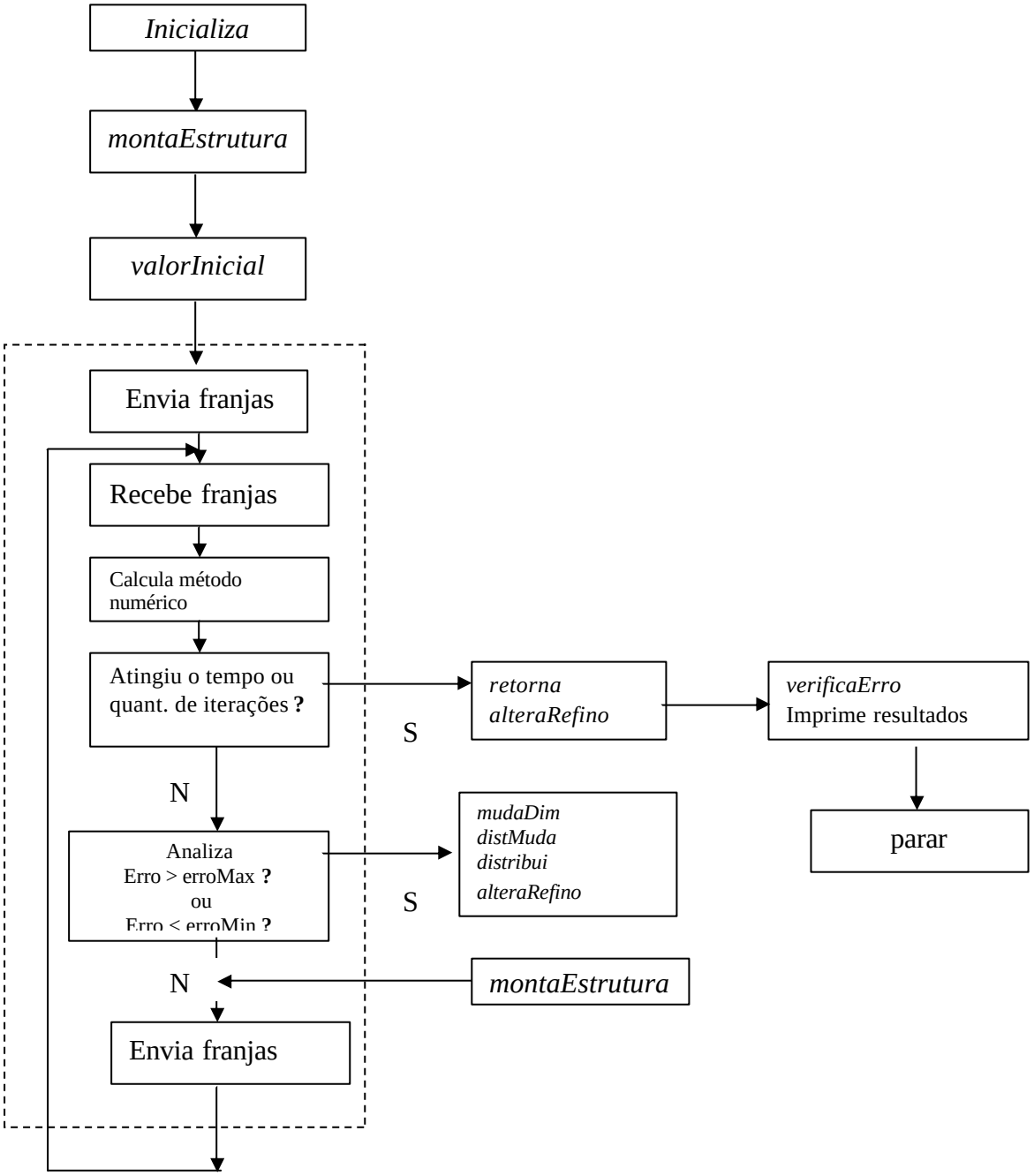


Figura 53 - Fluxograma de funcionamento do programa.
A linha pontilhada envolve os procedimentos do laço de resolução do método numérico

O cáculo do Hopscotch foi implementado segundo o pseudo-código da figura 54 a seguir

```
n = número de repetições
tFinal = tempo final
deltaT =  tFinal / nRep
t =  0
De 1 até n {
 t = t + deltaT
 De 1 até i {
  De 1 até j {
   Se i+j+n par
    cálculo explícito
```

```
   Se i+j+n ímpar
    cálculo implícito
  }
 }
 De 1 até i {
  De 1 até j {
   Se i+j+n ímpar
    cálculo explícito
   Se i+j+n par
    cálculo implícito
  }
 }
}
```

Figura 54 - Pseudo-código da implementação do método numérico

6.6 - DESEMPENHO DO PARTICIONADOR *PREPARA* E SUA COMPARAÇÃO COM OUTROS PROGRAMAS

Vários pacotes de particionamento foram desenvolvidos nos últimos 20 anos, na forma de softwares ou de bibliotecas para uso com códigos já existentes. Os mais citados na literatura pertinente ao assunto [6, 7, 13, 19, 20, 22, 24, 45, 47, 48, 50, 51, 56,] são os pacotes Chaco [42], de Hendrickson e Leland, MeTis [52], de Karypis e Kumar, Scotch [70], de Peligrinni e Roman, Party [75], de Preis e Diekmann e Jostle [96], de Walshaw, que servem de referência em comparações de desempenho envolvendo novos avanços no particionamento de diferentes malhas. É claro que seria interessante comparar os resultados obtidos aqui com aqueles obtidos por essas ferramentas. Naturalmente, a comparação é difícil por vários motivos. Todas essas ferramentas possuem um grande número de parâmetros a serem escolhidos pelo usuário em função da aplicação, e sua escolha demanda um grande envolvimento com o problema, normalmente por tentativa e erro. Mais ainda, elas consistem de implementações otimizadas, depuradas através dos anos, usando estratégias multinível direcionadas para grafos esparsos com grande número de nós. Por fim, todas essas ferramentas não consideram a migração dos dados durante o balanceamento e durante as iterações do método numérico. Para dificultar ainda mais a comparação, o resolutor desenvolvido aqui foi pensado para ser usado com um método específico, o método Hopscotch.

Além disso, o programa desenvolvido aqui resolve todo o problema, e não somente o particionamento do grafo. Todo o problema é construir a estrutura de dados para cada configuração de particionamento, realizar o refinamento e desrefinamento, migrar os dados para a nova configuração e efetivamente realizar a solução numérica da EDP, com a troca de mensagens entre os subdomínios e a análise do gradiente ou da convergência em cada subdomínio, iterando até que seja atingida a convergência ou o tempo desejado.

As referências [7, 56] indicam alguns sítios onde podem ser obtidos grafos para testes.

Por causa disso, apenas a parte que realiza o particionamento da carga foi comparada com os softwares Chaco, Metis e Scotch .

6.6.1 - Desempenho do particionador *Prepara*

O gráfico a seguir apresenta o desempenho do *Prepara*.

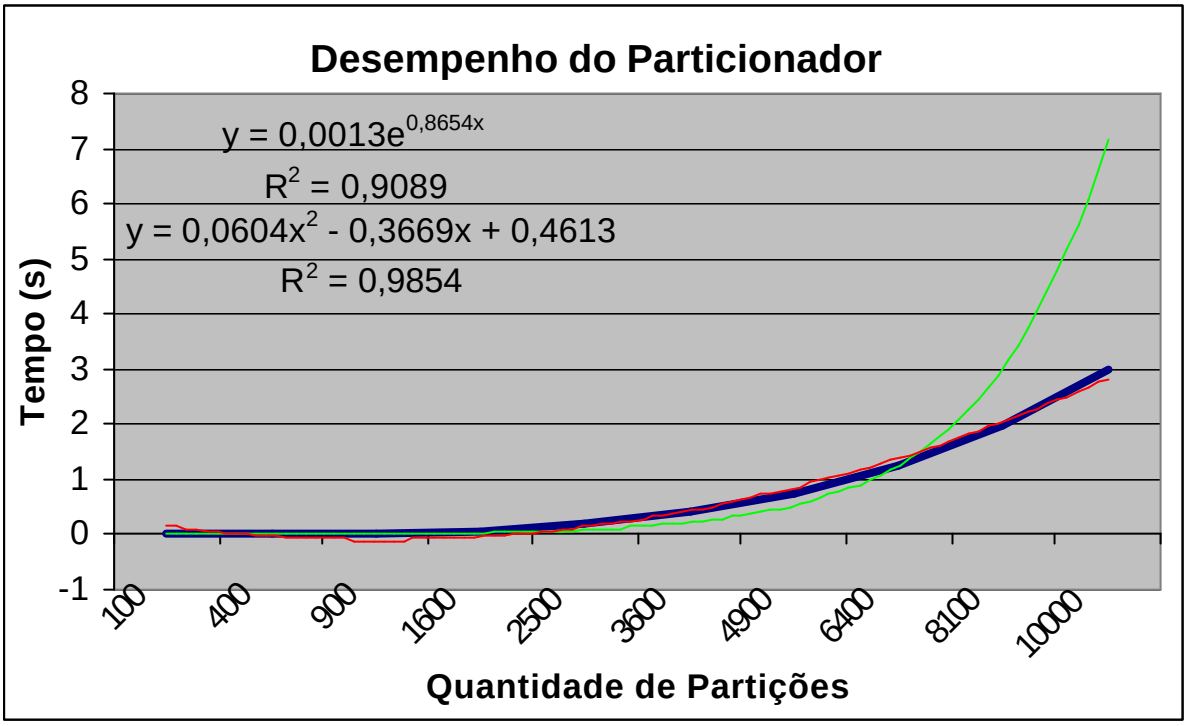


Figura 55 - Desempenho do Particionador com relação à quantidade de subdomínios

Para mostrar que a curva obtida para o desempenho do particionador (linha azul no gráfico) tem comportamento polinomial, duas curvas foram ajustadas a ela. Uma curva exponencial foi ajustada (linha verde), obtendo um ajuste muito pobre, como pode ser observado pelo valor de R^2, e uma curva polinomial de grau 2 (linha vermelha), que obteve ajuste bem melhor. Observando que os testes foram feitos em ambiente de utilização compartilhada, e provavelmente por isso o ajuste da curva polinomial não foi melhor, pode-se inferir que o desempenho é polinomial com a quantidade de partições, indicando que o comportamento está de acordo com o teórico, isto é, ele é aproximadamente $O(q^2)$, onde q é a quantidade de partições.

A figura 56 a seguir mostra o desempenho do *Prepara* em relação à quantidade de nós da malha de um subdomínio, considerando uma partição em 1024 (32^2) processadores.

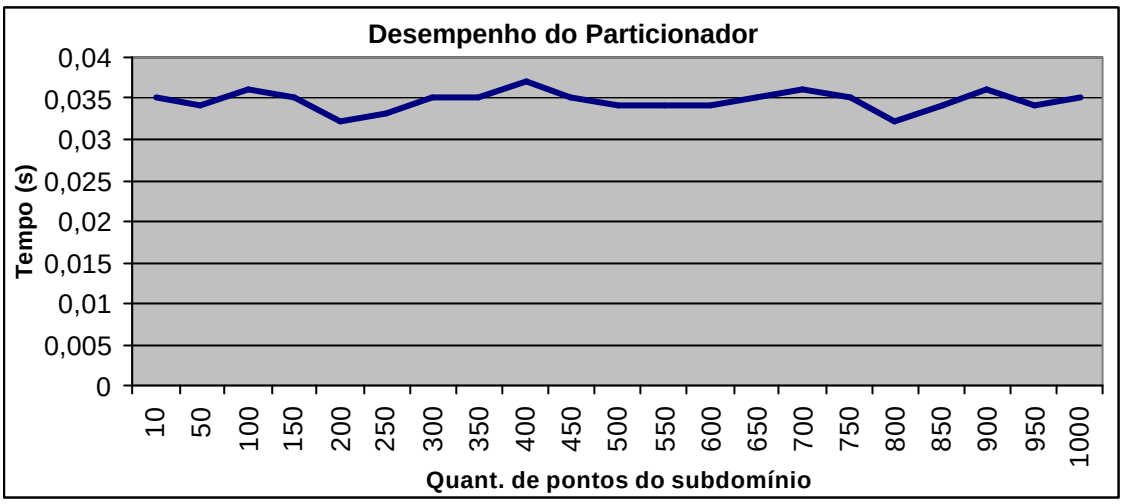


Figura 56 - Desempenho do particionador com relação à quantidade de nós de um subdomínio

Como pode ser observado na figura, o tempo não depende da quantidade de nós da malha, confirmando assim o resultado previsto, isto é, o particionador *Prepara* é *O(1)* considerando a quantidade de nós pertencentes a um subdomínio. Existiu uma pequena flutuação dos valores de tempo, que independeu da quantidade de nós e que pode ser atribuído a processos concorrentes, já que a máquina não estava com acesso exclusivo.

6.6.2 - Comparação com outros Particionadores

Foram usados os particionadores Metis, Chaco e Scotch como comparação. Os Apêndices A, B e C apresentam o resultado de testes realizados com esses particionadores. Esses pacotes não lidam com a migração dos dados entre os processadores.

O resolutor desenvolvido aqui lida com o particionamento do grafo juntamente com a migração dos dados e a solução numérica de uma EDP, mas considerando apenas a função de particionamento foi possível uma comparação, muito limitada pelos motivos já citados em 6.6, do desempenho desta aplicação com as aplicações estudadas.

Para comparar o *Prepara* com os outros particionadores, o ideal seria usar uma malha comum a todos. Mas não foi possível encontrar uma malha estruturada que tivesse um arquivo de entrada comum a Chaco, Metis e Scotch. Assim, foi usada a malha irregular *4elt* (ver figura 4), que tem 15.606 nós, com os pacotes Chaco, Metis e Scotch. O particionador da aplicação deste trabalho foi testado com uma malha estruturada de tamanho 125 x 125, resultando em 15.625 pontos, valor próximo ao da malha *4elt*, que tem 15.606 pontos.

Tabela 12 - Comparação entre os particionadores para ≈ 15.600 nós

	Tempo (s)			
Quantidade de subdomínios	`Prepara`	Chaco	Metis (kmetis)	Scotch
9	0,001024	0,068	0,030	0,144
16	0,001043	0,080	0,020	0,187
25	0,001285	0,080	0,030	0,221
36	0,001566	0,092	0,030	0,249
256	0,02635	0,204	0,060	0,476

Os resultados mostram que o *Prepara* é mais rápido nas situações testadas.

6.7 - DESCRIÇÃO DAS ROTINAS QUE IMPLEMENTAM O BALANCEAMENTO DE CARGA

A seguir estão descritas as rotinas usadas para realizar o balanceamento de carga. As principais variáveis, assim como os vetores e matrizes usados, também estão descritas.

a) Rotina *inicDim*

O vetor `dim` contém o grau de refinamento dos subdomínios.

b) Rotina *prepara*

É o particionador do domínio. Produz a "matriz de despacho". Essa matriz contém a quantidade de colunas que vão deixar um processador fornecedor para um processador receptor. O vetor `fica` contém a quantidade de colunas que vão ficar no processador fornecedor e a matriz `envia` contém a quantidade de colunas que sai de um determinado processador e vão para um processador ocioso. O vetor `fica` tem tamanho $\sqrt{q}$ (q é a quantidade de partições), e a matriz `envia` tem dimensão q x q.

No exemplo da seção 6.2, a matriz *envia* usada para o balanceamento foi a seguinte:

Tabela 13 - Matriz *envia* do exemplo 6.2

		Número da partição								
		0	1	2	3	4	5	6	7	8
Número da partição	0	0	0	66	0	0	66	66	66	45
	1	0	0	0	0	0	0	0	0	17
	2	0	0	0	0	0	0	0	0	0
	3	0	0	0	0	0	0	0	0	17
	4	0	0	0	0	0	0	0	0	12
	5	0	0	0	0	0	0	0	0	0
	6	0	0	0	0	0	0	0	0	0
	7	0	0	0	0	0	0	0	0	0
	8	0	0	0	0	0	0	0	0	0

Considerando-se que cada partição foi enviada a um processador, as linhas indicam de que processador a carga é retirada. As colunas indicam qual processador recebe a carga. Por exemplo, o processador 0 entrega 66 x 400 pontos para o processador 2, 66 x 400 pontos para o processador 5, 66 x 400 pontos para o processador 6, 66 x 400 pontos para o processador 7, e 45 x 400 pontos para o processador 8. As colunas indicam quem recebe a carga. Por exemplo, o processador 8 recebe 45 x 400 pontos do processador 0, 17 x 200 pontos do processador 1, 17 x 200 pontos do processador 3 e 12 x 200 pontos do processador 4. Pode-se notar que os processadores fornecedores têm zeros em suas colunas, e os processadores receptores têm zeros em suas linhas.

c) Rotina *meusSubdom*:

Identifica os subdomínios que estão no processador e os coloca no vetor `p`. A quantidade de subdomínios no processador é armazenada na variável *quantSub*.

d) Rotina *euLimites*:

Calcula os limites do que fica e, no caso de um processador fornecedor, de cada pedaço que vai para outro processador.

e) Rotina *meusVizinhos*:

Determina quem são os vizinhos dos subdomínios do processador. Tem que tratar 3 casos: 1 - se for receptor, tratar os sub recebidos; 2 - se for receptor, tratar o sub próprio; e 3 - se for fornecedor, tratar o que ficou. Chama a rotina *preIdentViz* para identificar todos os vizinhos dos subdomínios do processador. Depois chama a rotina *identViz* para atualizar os vizinhos Norte e Sul, pois podem ser mais de um de acordo com os limites dos subdomínios envolvidos e chama a rotina *identVizDiag* para atualizar os vizinhos Sudoeste, Oeste e Noroeste.

f) Rotina *preIdentViz*

Determina todos os vizinhos de todos os subdomínios pertencentes ao processador, um por direção.

g) Rotina *identViz*

Identifica todos os vizinhos Norte e Sul de todos os subdomínios do processador de acordo com seus limites. A matriz `s` contém a identificação desses subdomínios. Identifica as colunas que devem ir para cada vizinho. As matrizes tridimensionais `lim1` e `lim2` contêm os limites que devem ir para cada vizinho. A quantidade de vizinhos Norte e Sul está armazenada na matriz tridimensional `quantSubV`. Quando um vizinho Norte ou Sul é fornecedor a matriz tridimensional `ondeForam` identifica o processador para o qual eles foram enviados. Os índices das matrizes são `s[a][b]`, `lim1[a][b][c]`,

lim2[a][b][c], *ndeForam*[a][b][c], *quantSubV*[a][b], onde os índices a, b e c são:

a: indica seqüencialmente o subdomínio do processador

b: indica se é subdomínio próprio (b=0), se é subdomínio ao Norte (b=1) ou se é subdomínio ao Sul (b=2)

c: indica seqüencialmente o subdomínio do vizinho Norte ou Sul.

Essa rotina é necessária porque os subdomínios podem ter tamanho diferente no caso de algum deles ser fornecedor, ou podem ter mais de um vizinho Norte e mais de um vizinho Sul, no caso de algum deles ser fornecedor. A figura 57 ilustra essa situação.

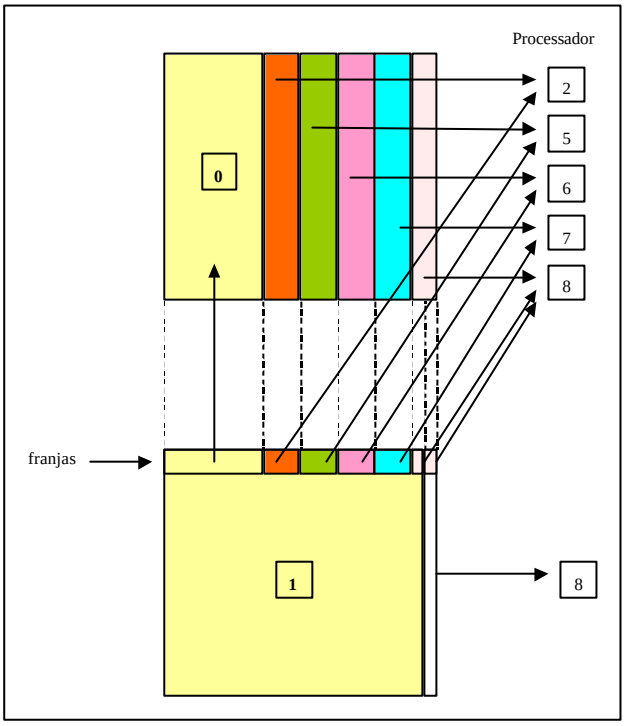


Figura 57 - Despacho das franjas

A figura 58 seguir apresenta os índices usados na identificação e controle do tráfego entre os subdomínios.

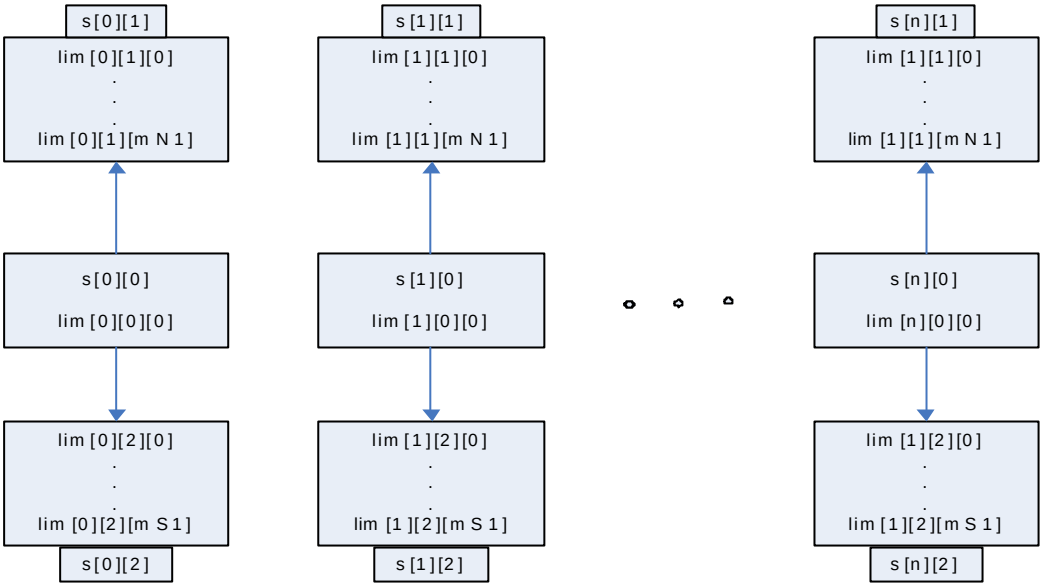


Figura 58 - Índices usados

h) Rotina *identVizDiag*

Atualiza os vizinhos Sudoeste, Oeste e Noroeste se o vizinho à esquerda for fornecedor.

Essa rotina é necessária se o vizinho a Oeste for fornecedor, porque aí as colunas mais à direita do vizinho Oeste estarão em outro processador. A figura 59 a seguir ilustra a situação. Nesta figura, um pedaço do subdomínio do processador 1 foi deslocado para o processador 3. Assim, o vizinho Oeste do processador 2 passou a estar no processador 4, e não mais no processador 1, como seria natural.

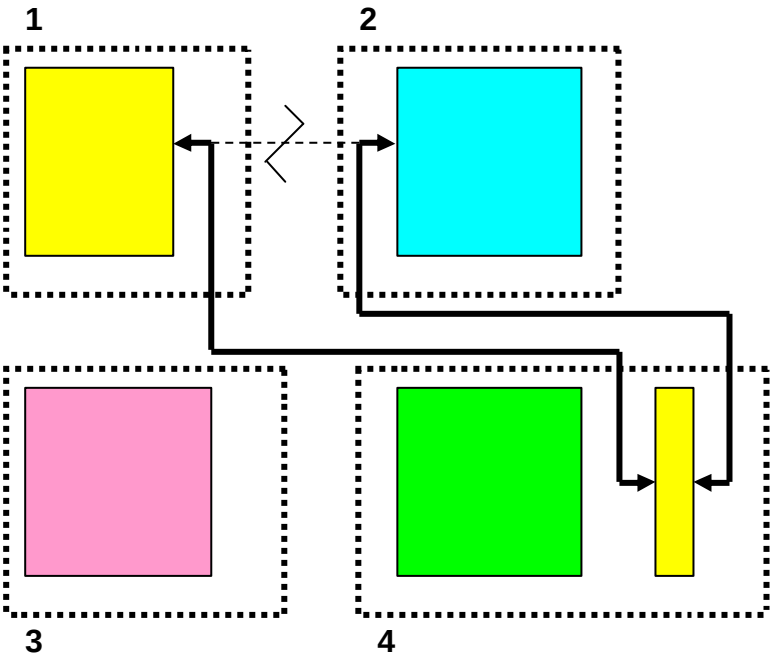


Figura 59 - Tráfego de dados

i) Rotina *distribui*

Executa duas funções principais:

a) Estabelece o valor inicial para o método numérico; e

b) Executa a migração dos dados entre os processadores.

É a rotina que efetivamente executa o balanceamento de carga. Envia e recebe partes do subdomínio dos processadores mais carregados para os menos carregados. É aonde os subdomínios são definidos fisicamente na máquina, da seguinte maneira: é alocada dinamicamente (`calloc`) uma área para o subdomínio próprio e áreas para os subdomínios recebidos, se for o caso. As áreas alocadas são contíguas e têm o tamanho dos subdomínios acrescidos das franjas, independente do tipo do subdomínio. A figura 60 a seguir ilustra a disposição na memória

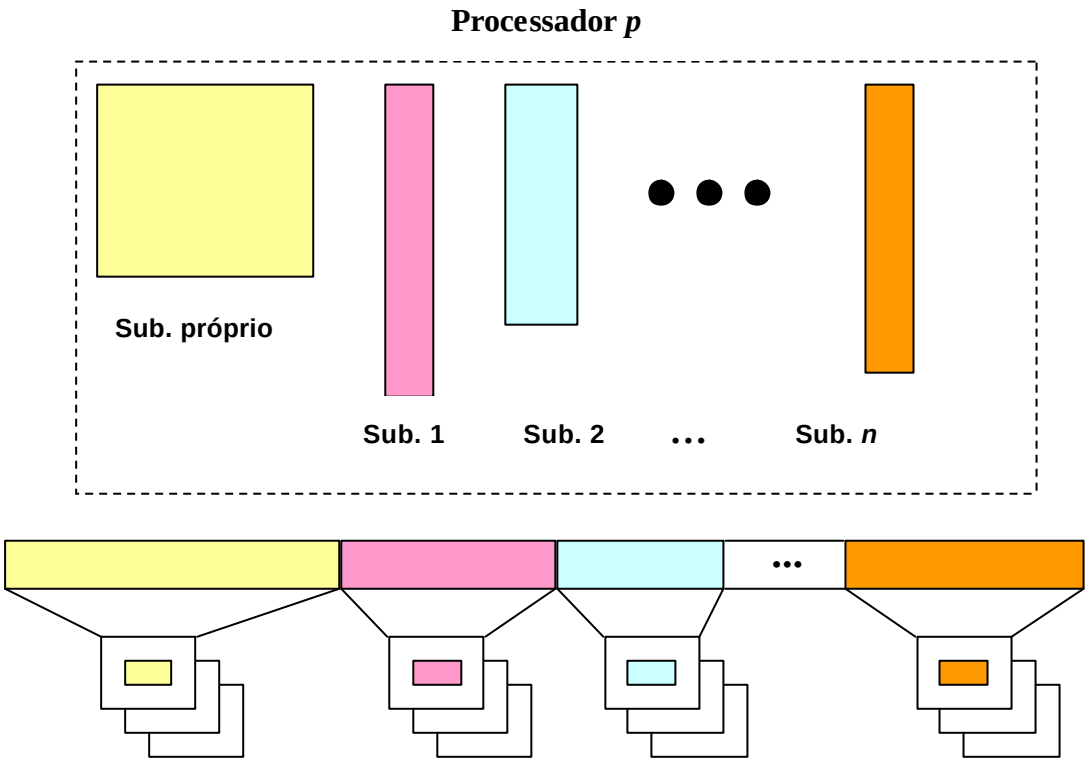


Figura 60 - Armazenamento no processador *p*

Ao remontar o subdomínio original, a rotina ajusta o espaço ocupado. Esse tamanho não varia muito, porque os processadores estarão sempre com a carga balanceada.

j) Rotina *alteraRefino*

Rotina que refina ou desrefina um subdomínio. Usa splines para o refinamento e o desrefinamento.

k) Rotina *retorna*

Rotina que recompõe os subdomínios, trazendo de volta aos processadores de origem os pedaços que foram distribuídos aos processadores ociosos.

l) Rotina *enviaIni*

É a rotina que executa as iterações do método numérico. As iterações são compostas por receber os dados provenientes das regiões em outros processadores, de executar um passo do método Hopscotch e por enviar os dados para os outros processadores.

A cada *x* iterações, onde *x* é definido pelo usuário, a rotina *analisa* verifica se é necessário alterar o refinamento em alguma região. Se for necessário, as rotinas de cálculo do balanceamento, construção das estruturas de dados e de migração dos dados são chamadas para realocar os subdomínios. Após isso as iterações prosseguem.

n) Rotina *analisa*

Dependendo da escolha feita pelo usuário, calcula, em cada ponto da malha, a diferença entre a iteração atual e a imediatamente anterior, ou seja, analisa a convergência, ou calcula a maior diferença do valor em cada ponto da malha em relação aos seus vizinhos Norte, Sul, Leste, Oeste, ou seja, calcula o gradiente.

Três casos podem ocorrer em ambas as situações:

1- Se a diferença for maior que um valor especificado pelo usuário, o subdomínio é refinado.

2 - Se a diferença for menor que um valor especificado pelo usuário, o subdomínio é desrefinado.

3 – Se a diferença cair entre os dois valores especificados pelo usuário para refinamento e desrefinamento, o subdomínio permanece como está.

o) Rotina *distAna*

Avisa a todos os processadores que é necessário uma mudança no refinamento, usando a função `MPI_Alltoall.`

p) Rotina *distMuda*

Distribui para todos os processadores um vetor que indica quais subdomínios devem ser refinados ou desrefinados, usando a função `MPI_Alltoall.`

q) Rotina *mudaDim*

Altera o grau de refinamento de um subdomínio. O grau é sempre alterado por um fator de 2, refinando ou desrefinando. Cada processo comunica seu valor, após o refinamento ou desrefinamento, usando a função `MPI_Alltoall`.

r) Rotina *verificaErro*

Calcula a norma infinita do erro entre a solução Hopscotch e a solução analítica da EDP. É chamada se existe solução analítica disponível. Cada processo calcula seu erro máximo e seu valor máximo, que são comunicados a todos os processos usando a função `MPI_Alltoall`. Cada processo então calcula a norma infinita do erro, usando o erro máximo e o valor máximo da variável dependente dentre todas as regiões.

s) Rotina *montaEstrutura*

Refaz a estrutura de dados sempre que houver necessidade de um novo balanceamento. Chama as rotinas *inicializa*, *prepara*, *meusSubdom*, *euLimites*, *meusVizinhos*, *defineVizinhos*, *defineLimites*.

6.8 - Casos Teste

Uma versão paralela com refinamento adaptativo e balanceamento dinâmico de carga foi implementada, e testada com as equações anteriormente usadas neste trabalho. O ambiente computacional usado nos testes está apresentado no Apêndice D.

O código de cores usados nas figuras para indicar o nível do refinamento é apresentado na figura 61 a seguir.

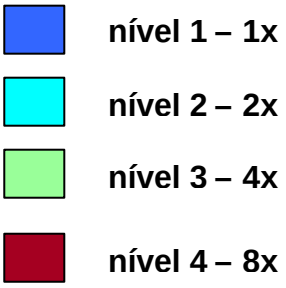


Figura 61 - Legenda para as figuras 62, 67 e 72

Nos três casos teste o refinamento foi limitado a três níveis, podendo atingir até 8 vezes o refinamento original. Por exemplo, um subdomínio 20 x 20 pode ser refinado até um máximo de 160 x 160.

As figuras 62, 66 e 70 foram geradas usando o aplicativo MatLab e utilizando parâmetros para facilitar a visualização da mudança do refinamento ao longo do processamento. Os valores gatilho de refinamento e desrefinamento foram obtidos empiricamente, assim como a freqüência de verificação do erro da solução.

Inicialmente a freqüência de verificação do erro da solução foi estabelecida em 1. Depois foi usado uma freqüência de verificação do erro da solução alta, da ordem de centenas de iterações. O tempo diminuiu enormemente, e o erro manteve-se estável, até diminuindo um pouco. Esse fato, aparentemente inesperado, provavelmente foi devido ao fato de que certos

subdomínios atingiram mais rapidamente o nível 4 de refinamento. Isto significa que subdomínios refinados 8x ficam se comunicando com os subdomínios em nível mais baixo de refinamento por mais tempo, o que tende a aumentar o erro. Assim, o ganho em precisão ocasionado por um refinamento que acompanhe mais rapidamente o aumento do erro é anulado pelo erro que aparece quando subdomínios com grau de refinamento muito diferentes comunicam-se. Deste modo, a freqüência de verificação do erro da solução foi determinada empiricamente para cada um dos três tipos de equações sendo testadas de modo a manter baixos o erro e o tempo de execução, para determinados valores de gatilho. Como exemplo, utilizou-se a equação parabólica, e construiu-se a tabela a seguir, que apresenta o comportamento do erro e o tempo de execução para diversos valores da freqüência de verificação para domínio 60 x 60 utilizando 9 processadores, isto é, cada subdomínio de tamanho 20 x 20. O valor da freqüência de verificação para os outros dois tipos de equações foi determinado de modo semelhante.

Tabela 14 - Erro e tempo de execução em função da freqüência de verificação

Freqüência de verificação	Erro	Tempo
1	0,08981	95,8
10	0,08765	30,6
50	0,08220	17,5
100	0,08332	16,7
500	0,08364	14,7
1000	0,09876	10,1
2000	0,11445	9,5
3000	0,14769	7,2

6.8.1 - Equação Parabólica

A equação parabólica foi resolvida agora usando refinamento adaptativo e balanceamento de carga.

A decisão de mudança no refinamento foi feita levando em conta a convergência da solução. A diferença entre a solução atual e a solução imediatamente anterior foi comparada com valores de gatilho informados pelo usuário. Existem dois valores de gatilho, um para refinamento e outro para desrefinamento. O critério de parada utilizado foi o tempo. Foi estabelecido um limite de três níveis de refinamento, isto é, um subdomínio pode ser refinado até 2^3 vezes.

A figura 62 a seguir é um exemplo da evolução do refinamento dinâmico para o caso em que um domínio 60 x 60 foi particionado em 25 subdomínios com refinamento inicial de 12 x 12, usando os seguintes valores de gatilho: 0,00015 para refinamento e 0,00010 para desrefinamento.

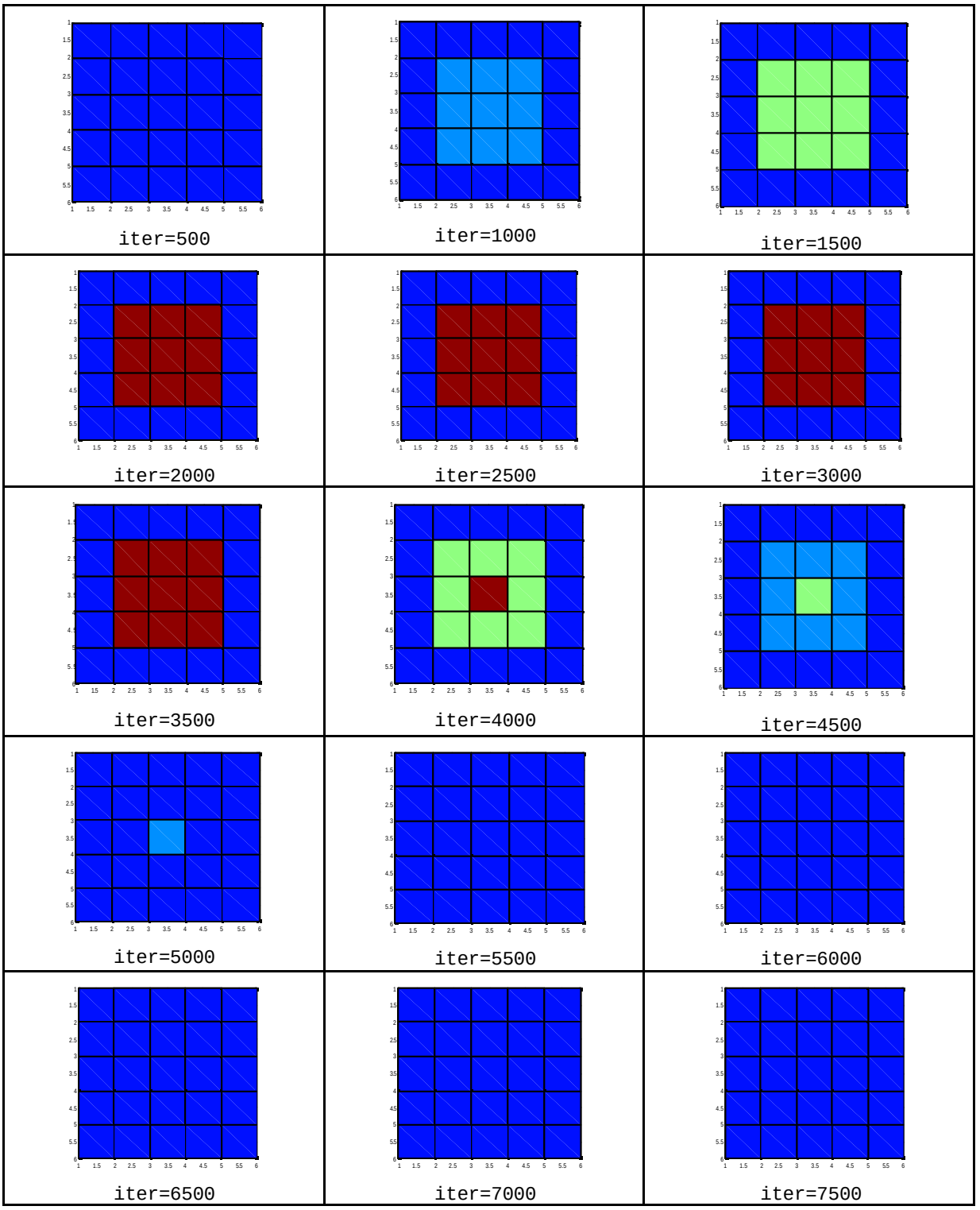


Figura 62 - Evolução do refinamento durante a resolução

Pode ser observado que o desrefinamento começou no instante 0,05s, na iteração (um meio-passo) 4.000, e que a partir do instante 0,06875s, o que correspondeu a 5.500 iterações, o refinamento voltou a ser uniforme.

As tabelas 14 e 15 a seguir apresentam as métricas do programa com resolução usando balanceamento de carga, quando comparado com a resolução sem balanceamento de carga. Para a construção da tabela, o domínio foi dividido em 4, 9, 16 e 25 subdomínios, com refinamento inicial de 60 x 60. Quando o refinamento alcança o nível 4 (8x) em uma região de um domínio 60 x 60, nesta região o refinamento alcança 480 x 480. Assim, a comparação foi feita com uma versão sem balanceamento de carga, onde o tempo de processamento será ditado pelo subdomínio mais refinado, isto é, como se todos tivessem refinamento 480 x 480. Para o cálculo do "speedup" e da eficiência, foi usado o valor de t_s = 6.888,3s (ver tabela 3 da seção 2.8.1)

Tabela 15 - "Speedup" e eficiência do balanceamento de carga

q	**Tamanho do subdomínio**	**Tempo com balanceamento (s)**	**"Speedup"**	**Eficiência**	**Erro** L_∞
4	30 x 30	1014,6	1,843	0,4604	0,08364
9	20 x 20	228,4	3,679	0,4091	
16	15 x 15	140,8	3,925	0,2453	
25	12 x 12	114,2	3,197	0,1281	

Tabela 16 - Ganho para o balanceamento de carga

q	**Tamanho do subdomínio**	**Tempo com balanceamento (s)**	**Tempo sem Balanceamento (s)**	**G**
4	30 x 30	1014,6	1868,8	1,842
9	20 x 20	228,4	840,2	3,679
16	15 x 15	140,8	552,7	3,925
25	12 x 12	114,2	365,2	3,198

A tabela mostra que houve redução acentuada do tempo de processamento com o emprego de balanceamento dinâmico da carga. É claro que o balanceamento dinâmico deixa o problema com menos pontos na média, mas sendo a paralelização do método Hopscotch de natureza síncrona, sem o balanceamento de carga o tempo seria ditado pelo processador mais carregado. O que o balanceamento dinâmico faz é tirar proveito dessa diminuição da carga média do problema. Deve-se notar que os tempos da versão com refinamento adaptativo dependem da dinâmica de refinamento e desrefinamento dos subdomínios, função dos valores-gatilho usados.

As figuras 63 e 64 a seguir apresentam os gráficos dos valores do “speedup” e da eficiência em função da quantidade de processadores utilizados no processamento. O domínio foi particionado na mesma quantidade dos processadores utilizados no processamento, isto é, cada subdomínio foi enviado para um processador.

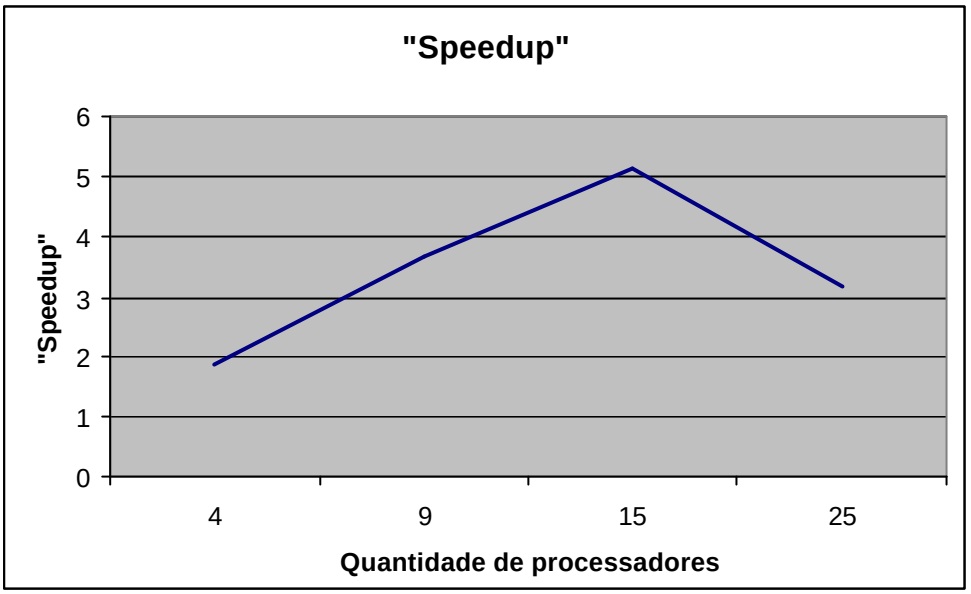


Figura 63 - “Speedup” do programa com balanceamento de carga

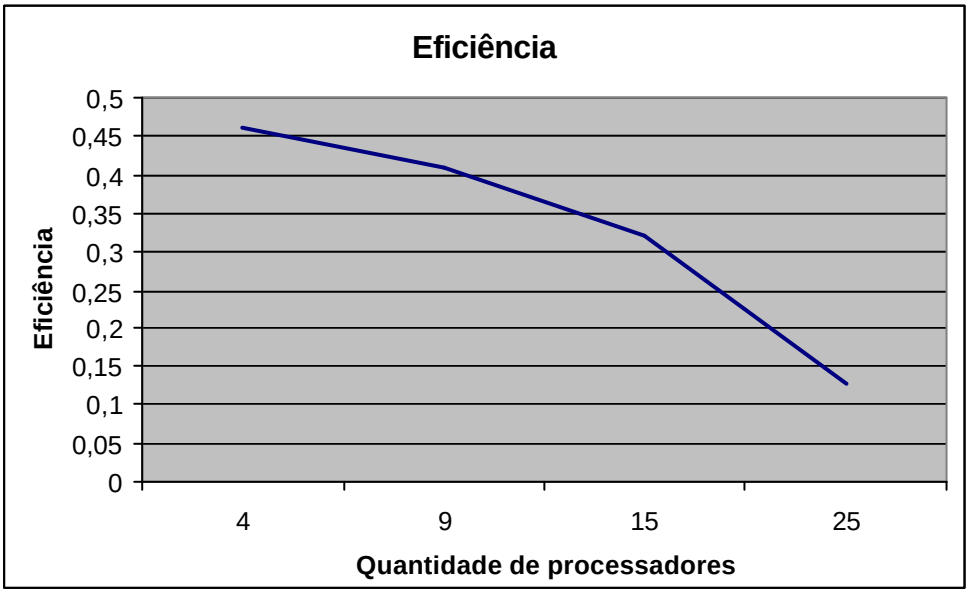


Figura 64 - Eficiência do programa com balanceamento de carga

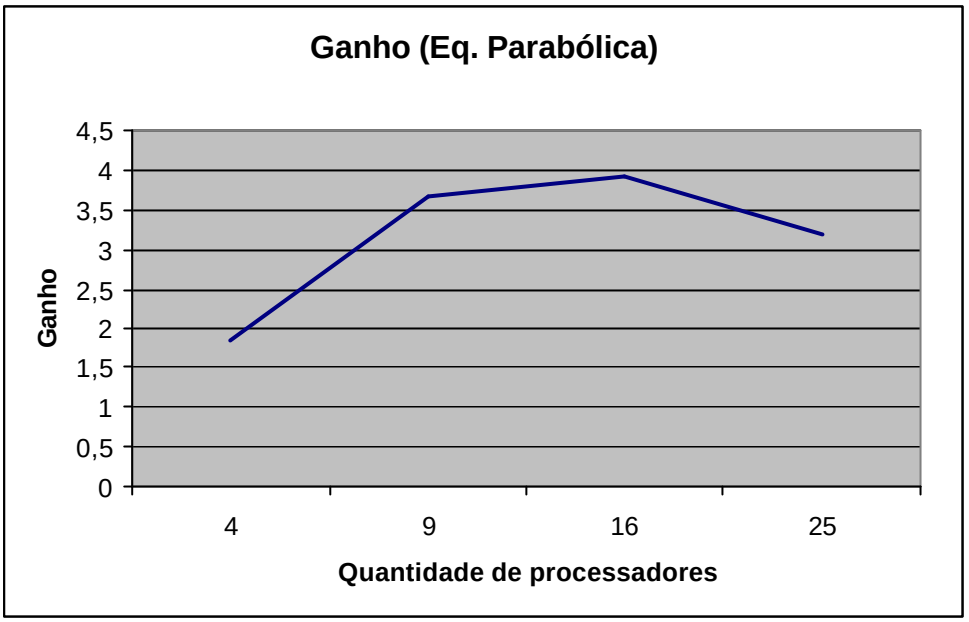


Figura 65 - Ganho do programa com balanceamento de carga quando comparado com a versão sem balanceamento de carga

O "speedup", a eficiência e o ganho apresentaram valores elevados, mostrando que o balanceamento de carga foi capaz de obter os resultados com muito mais rapidez. Pode-se ver que essas métricas sofrem diminuição de seu valor, ainda que continuem elevados, para as execuções usando maior quantidade de processadores. Uma das causas para esse comportamento parece ser devida ao aumento de comunicação quando a carga é balanceada, uma vez que o particionador *Prepara* não minimiza as comunicações. O ambiente computacional usado foi um "cluster" ligado por rede Ethernet (ver Apêndice D). Para quantidade de processadores maior que 14 (sítio1), a velocidade da comunicação cai de 1 Gigabit para 100 Megabit, o que também concorre para a queda no rendimento quando usando 16 e 25 processadores. Deve-se notar que esse ambiente tem velocidade de transmissão de dados muito menor que as comunicações de um supercomputador, ambiente para o qual o resolutor desenvolvido aqui foi projetado.

O gráfico a seguir mostra o índice médio de desbalanceamento (IMD) durante o balanceamento dinâmico que ocorre ao longo do processamento da EDP para os casos da tabela 8 da seção 3.8.1, onde o caso usando 4 processadores foi substituído por um caso usando 36 processadores para melhor entender o comportamento do IMD.

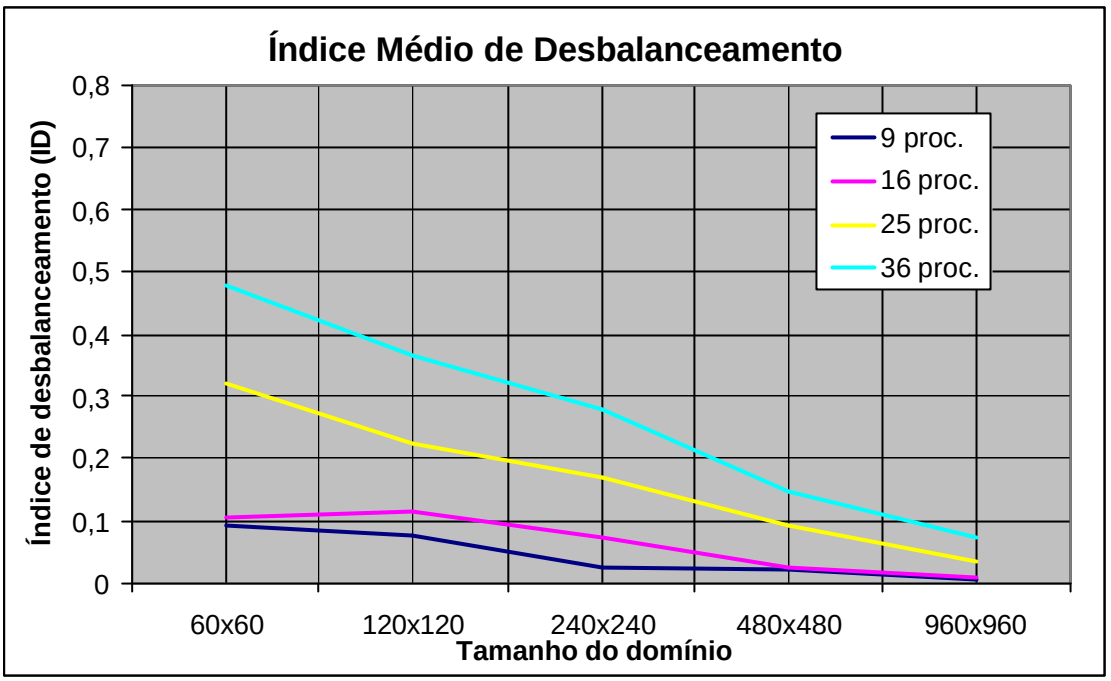


Figura 66 - Índice de desbalanceamento

Quanto maior o subdomínio for, mais flexibilidade o particionador tem para balancear a carga. À medida que a quantidade de processadores aumenta, a qualidade do balanceamento diminui porque os subdomínios ficam menores, o que diminui a flexibilidade de distribuir as colunas

6.8.2 - Equação Elíptica

A equação de Poisson foi resolvida agora usando refinamento adaptativo e balanceamento de carga.

A decisão de mudança no refinamento foi feita levando em conta a convergência da solução. A diferença entre a solução atual e a solução imediatamente anterior foi comparada com valores de gatilho. Existem dois valores de gatilho, um para refinamento e outro para desrefinamento. O critério de parada utilizado foi uma determinada quantidade de iterações. Foi estabelecido um limite de três níveis de refinamento, isto é, um subdomínio pode ser refinado até 2^3 vezes.

Para construir a figura 67 a seguir, ilustrativa da evolução do refinamento dos subdomínios, o domínio com refinamento inicial de 60 x 60 foi particionado em 9 subdomínios de 20 x 20, usando os seguintes valores de gatilho: 0,000000005 para refinamento e 0,000000002 para desrefinamento.

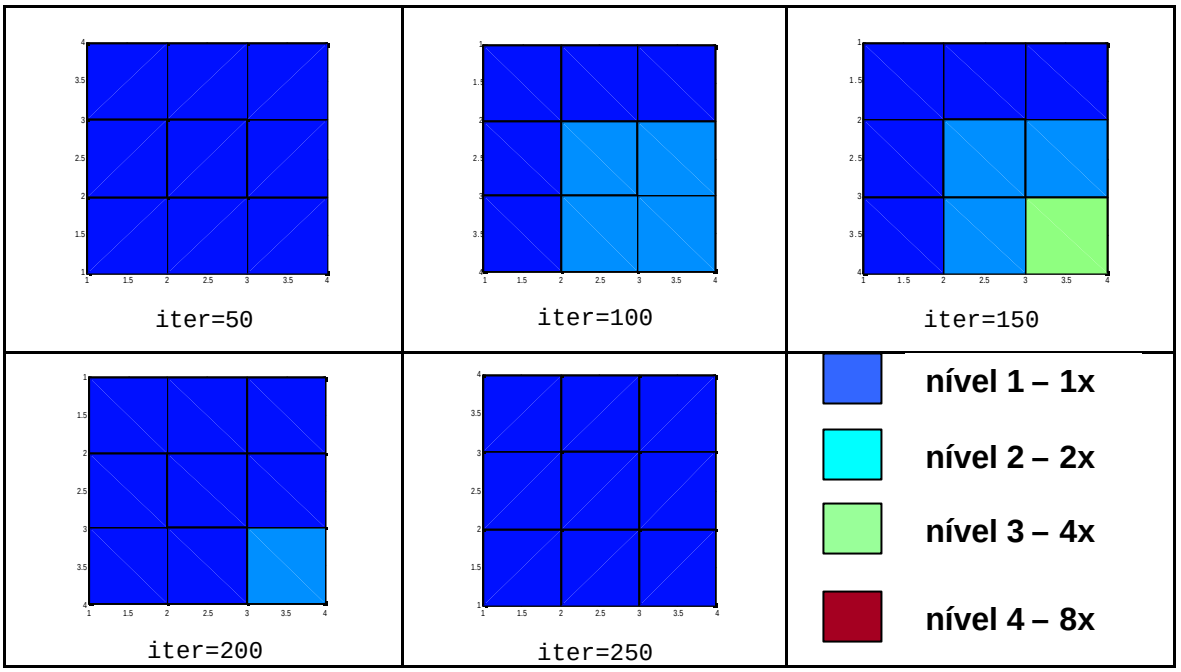


Figura 67 - Evolução do grau de refinamento durante a resolução da equação

Pode ser observado que o refinamento acompanhou a região de ocorrência de maiores gradientes (ver figuras 12 e 13 da seção 2.8.2) e que não atingiu o nível três, pois a solução convergiu antes.

As tabelas 16 e 17 a seguir apresentam as métricas do programa com resolução usando balanceamento de carga, quando comparado com a resolução sem balanceamento de carga. Para a construção da tabela, o domínio foi dividido em 9, 16, 25 e 36 subdomínios, com refinamento inicial de 20 x 20. Quando o refinamento alcança o nível 3 (4x) em uma região de um subdomínio 20 x 20, é como se na região refinada o domínio fosse 240 x 240. Assim, a comparação foi feita com a resolução usando domínios com refinamento uniforme de 240 x 240. Para o cálculo do "speedup" e da eficiência, foi usado o valor de t_s = 336,6s (ver tabela 5 da seção 2.8.2)

Tabela 17 - "Speedup" e eficiência do balanceamento de carga

q	**Tamanho do subdomínio**	**Tempo com balanceamento (s)**	**"Speedup"**	**Eficiência**	**Erro** L_∞
4	30 x 30	38,49	8,745	2,186	0,0724
9	20 x 20	16,21	20,76	2,307	
16	15 x 15	17,60	19,12	1,195	
25	12 x 12	18,37	18,32	0,7328	

Tabela 18 - Ganho para o balanceamento de carga

q	Tamanho do subdomínio	Tempo com balanceamento (s)	Tempo sem Balanceamento (s)	G
4	30 x 30	38,49	185,9	4,830
9	20 x 20	16,21	89,02	5,492
16	15 x 15	17,60	55,51	3,154
25	12 x 12	18,37	40,23	2,190

A tabela mostra que houve redução acentuada do tempo de processamento com o emprego de balanceamento dinâmico da carga. É claro que o balanceamento dinâmico deixa o problema com menos pontos na média, mas como a paralelização do método Hopscotch é de natureza síncrona, sem o balanceamento de carga o tempo seria ditado pelo processador mais carregado. O que o balanceamento dinâmico faz é tirar proveito dessa diminuição da carga média do problema. Deve-se notar que os tempos da versão com refinamento adaptativo dependem da dinâmica de refinamento e desrefinamento dos subdomínios, função dos valores-gatilho usados.

As figuras 68 e 69 a seguir apresentam os gráficos dos valores do "speedup" e da eficiência em função da quantidade de processadores utilizados no processamento. O domínio foi particionado na mesma quantidade dos processadores utilizados no processamento, isto é, cada subdomínio foi enviado para um processador.

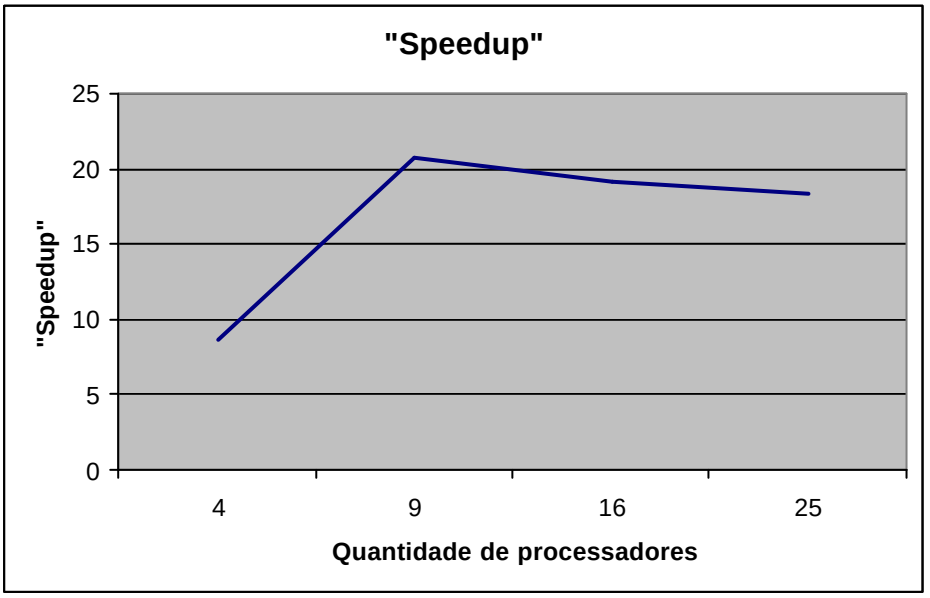


Figura 68 - "Speedup" do programa com balanceamento de carga

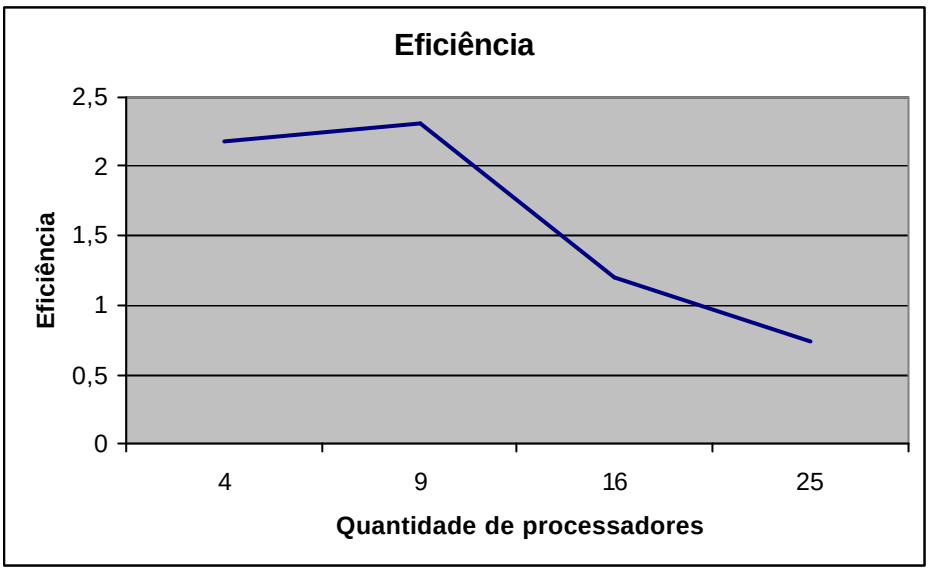


Figura 69 - Eficiência do programa com balanceamento de carga

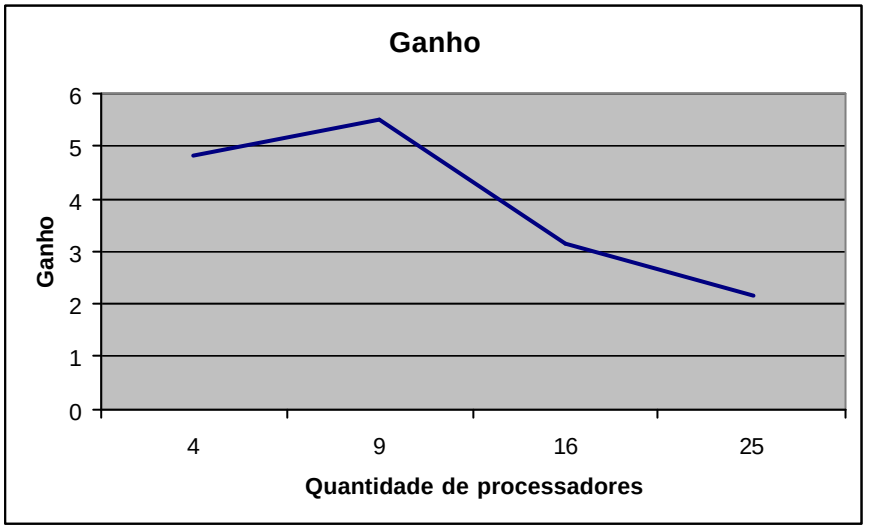


Figura 70 - Ganho do programa com balanceamento de carga quando comparado com a versão sem balanceamento de carga

O "speedup", a eficiência e o ganho apresentaram valores elevados, mostrando que o balanceamento de carga foi capaz de obter os resultados com muito mais rapidez. Pode-se ver que essas métricas sofrem diminuição de seu valor, ainda que continuem elevados, para as execuções usando maior quantidade de processadores. Uma das causas para esse comportamento parece ser o aumento de comunicação quando a carga é balanceada, uma vez que o particionador *Prepara* não minimiza as comunicações. O ambiente computacional usado foi um "cluster" ligado por rede Ethernet (ver Apêndice D). Para quantidade de processadores maior que 14 (sítio1), a velocidade da comunicação cai de 1 Gigabit para 100 Megabit, o que também concorre para a queda no rendimento quando usando 16 e 25 processadores. Deve-se notar que esse ambiente tem velocidade de transmissão de dados muito menor que as comunicações de um supercomputador, ambiente para o qual o resolutor

desenvolvido aqui foi projetado.

O gráfico a seguir mostra o índice médio de desbalanceamento (IMD) durante o balanceamento dinâmico que ocorre ao longo do processamento da EDP para os casos da tabela 9 da seção 3.8.2, onde o caso usando 4 processadores foi substituído por um caso usando 36 processadores para melhor entender o comportamento do IMD.

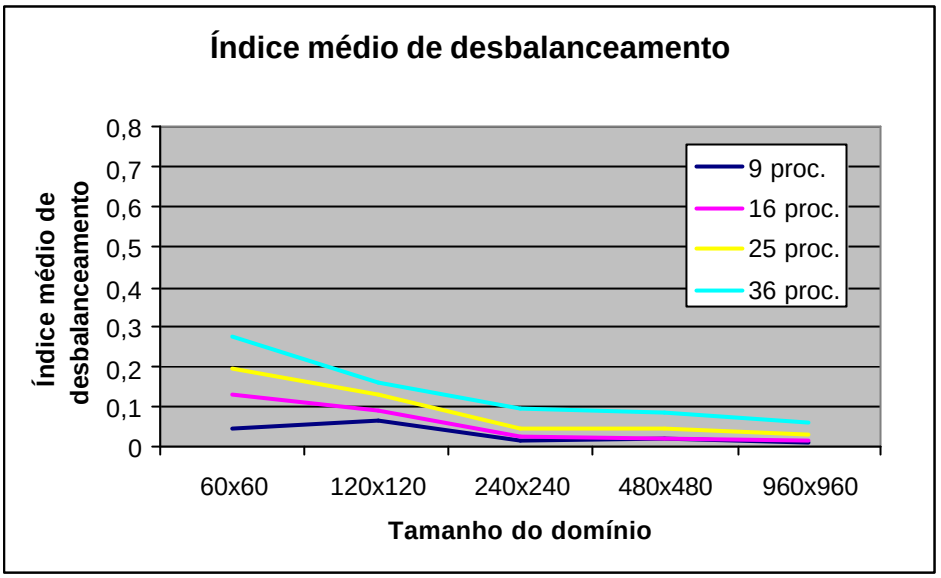


Figura 71 - Índice médio de desbalanceamento

Quanto maior o subdomínio for, mais flexibilidade o particionador tem para balancear a carga. À medida que a quantidade de processadores aumenta, a qualidade do balanceamento diminui porque os subdomínios ficam menores, o que diminui a flexibilidade de distribuir as colunas.

6.8.3 - Equação Hiperbólica

A equação da onda foi resolvida agora usando refinamento adaptativo e balanceamento dinâmico de carga.

A decisão de mudança no refinamento foi feita de acordo com o gradiente da solução. A diferença entre o valor de cada ponto e o valor em cada ponto vizinho foi comparada com valores de gatilho informados pelo usuário. Existem dois valores de gatilho, um para refinamento e outro para desrefinamento. O critério de parada utilizado foi uma determinada quantidade de iterações. Foi estabelecido um limite de três níveis de refinamento, isto é, um subdomínio pode ser refinado até 2^3 vezes.

Para construir a figura 72 a seguir, ilustrativa da evolução do refinamento dos subdomínios, para obter uma melhor visualização, o domínio, com refinamento inicial de 160 x 160, foi particionado em 64 subdomínios com refinamento inicial de 20 x 20. Foram usadas

20.000 iterações, com freqüência de verificação para refinamento de 1.000 iterações, usando o valor de gatilho de 0,03 para refinamento e desrefinamento.

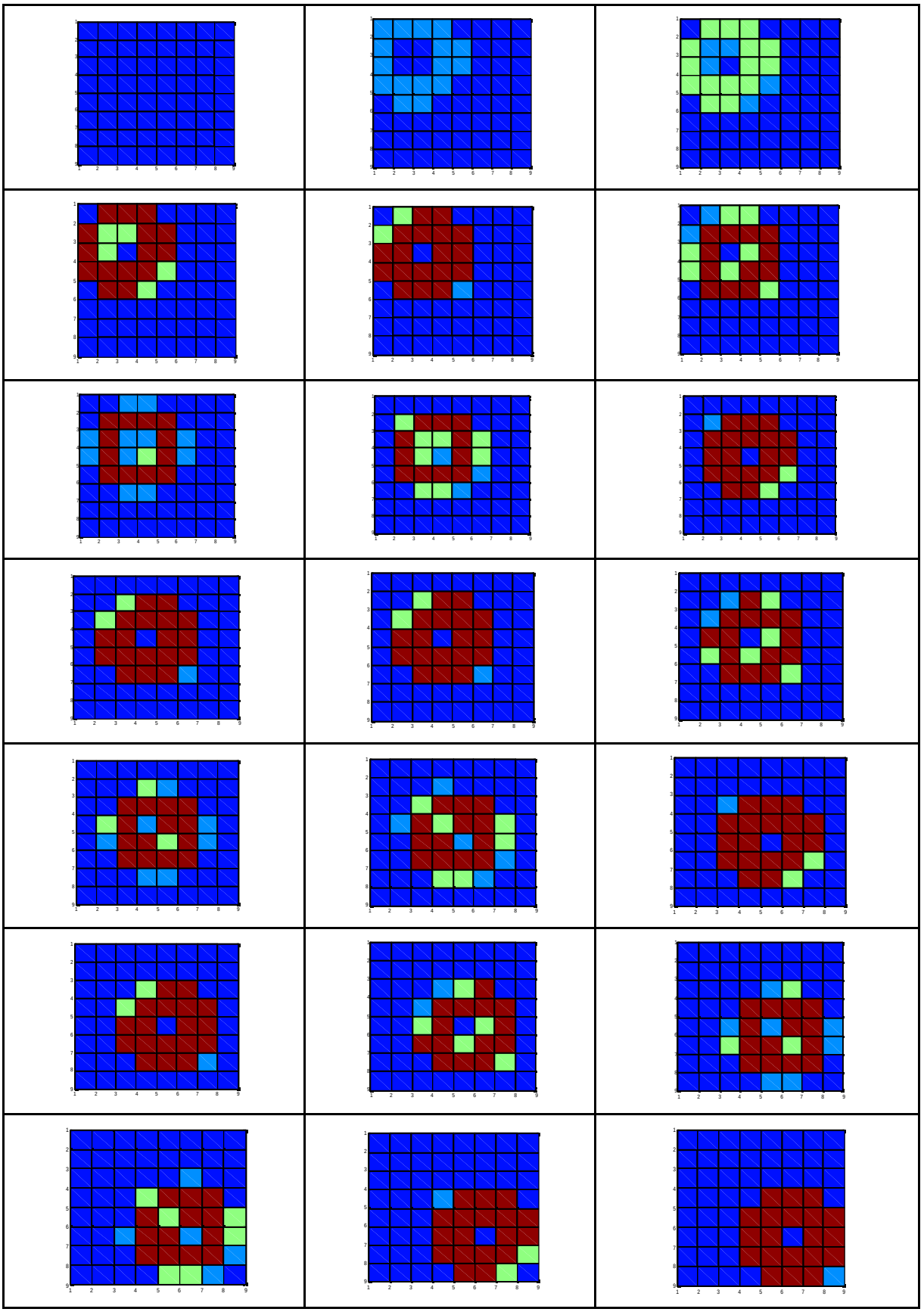


Figura 72 - Evolução do refinamento adaptativo

Pode ser observado, comparando com a figura 15 da seção 2.8.3, que o nível de refinamento acompanha o deslocamento da onda.

As tabelas 18 e 19 a seguir apresentam as métricas do programa com resolução usando balanceamento de carga, quando comparado com a resolução sem balanceamento de carga. Para a construção da tabela, o domínio foi dividido em 9, 16, 25 e 36 subdomínios, com refinamento inicial de 60 x 60. Quando o refinamento alcança o nível 4 (8x) em uma região de um domínio 60 x 60, é como se na região refinada o domínio fosse 480 x 480. Assim, a comparação foi feita com a resolução usando domínio com refinamento uniforme de 480 x 480. Para o cálculo do "speedup" e da eficiência, foi usado o valor de t_s = 27,95s (ver tabela 7 da seção 2.8.3)

Tabela 19 - "Speedup" e eficiência do balanceamento de carga

q	Tamanho do subdomínio	Tempo com balanceamento (s)	"Speedup"	Eficiência	Erro L_∞
4	30 x 30	2,248	12,43	3,107	0,3045
9	20 x 20	2,338	11,95	1,328	
16	15 x 15	3,331	8,391	0,5244	
25	12 x 12	3,721	7,511	0,3004	

Tabela 20 - Ganho para o balanceamento de carga

q	Tamanho do subdomínio	Tempo com balanceamento (s)	Tempo sem Balanceamento (s)	G
4	30 x 30	2,248	21,53	9,577
9	20 x 20	2,338	13,31	5,693
16	15 x 15	3,331	10,64	3,194
25	12 x 12	3,721	8,484	2,280

A tabela mostra que houve redução acentuada do tempo de processamento com o emprego de balanceamento dinâmico da carga. Como já mencionado, o balanceamento dinâmico deixa o problema com menos pontos na média, mas sendo a paralelização do método Hopscotch de natureza síncrona, sem o balanceamento de carga o tempo seria ditado pelo processador mais carregado. O que o balanceamento dinâmico faz é tirar proveito dessa diminuição da carga média do problema. Deve-se notar que os tempos da versão com refinamento adaptativo dependem da dinâmica de refinamento e desrefinamento dos subdomínios, função dos valores-gatilho usados.

As figuras 73 e 74 a seguir apresentam os gráficos dos valores do "speedup" e da eficiência em função da quantidade de processadores utilizados no processamento. O domínio

foi particionado na mesma quantidade dos processadores utilizados no processamento, isto é, cada subdomínio foi enviado para um processador.

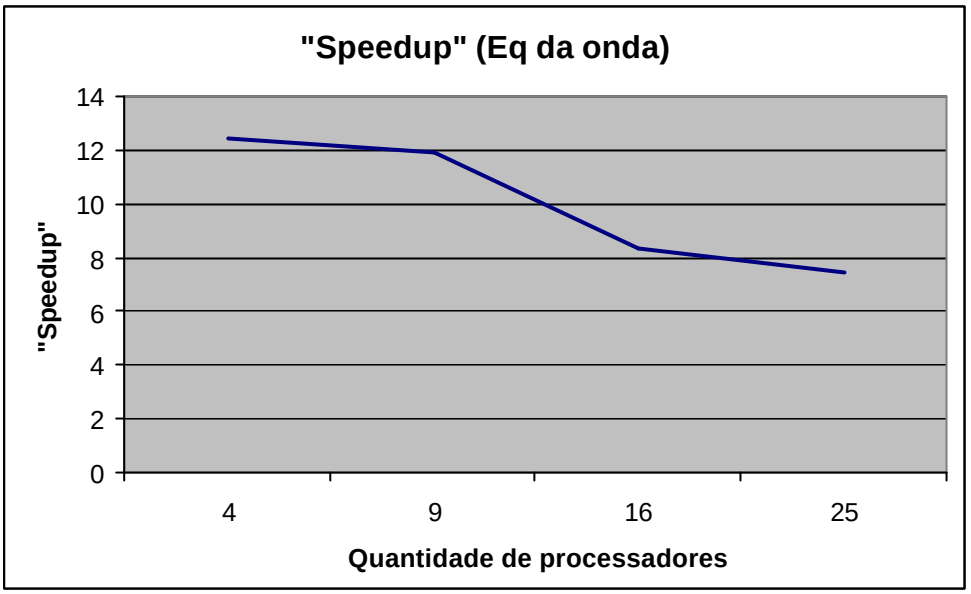


Figura 73 - "Speedup" do programa com balanceamento de carga

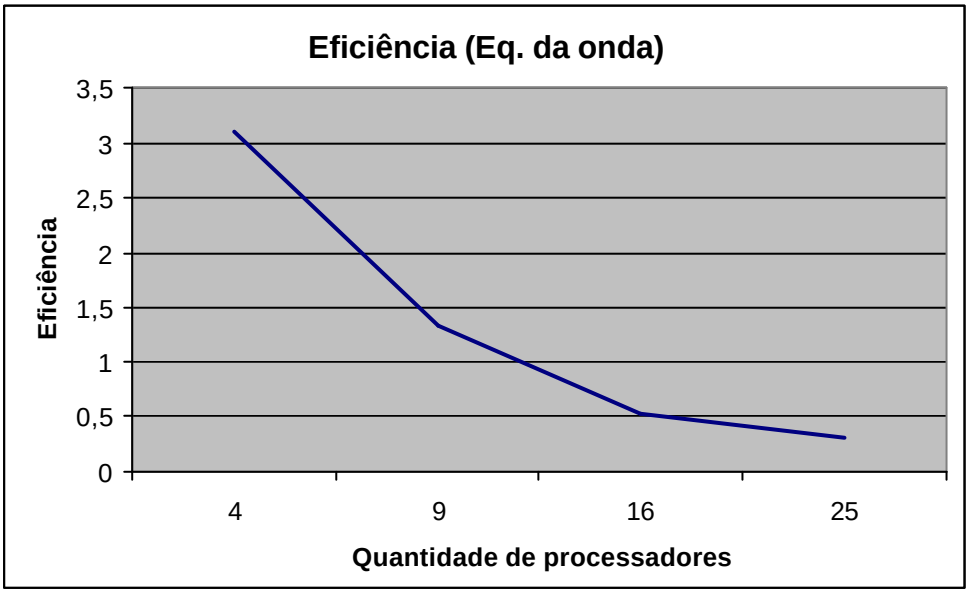


Figura 74 - Eficiência do programa com balanceamento de carga

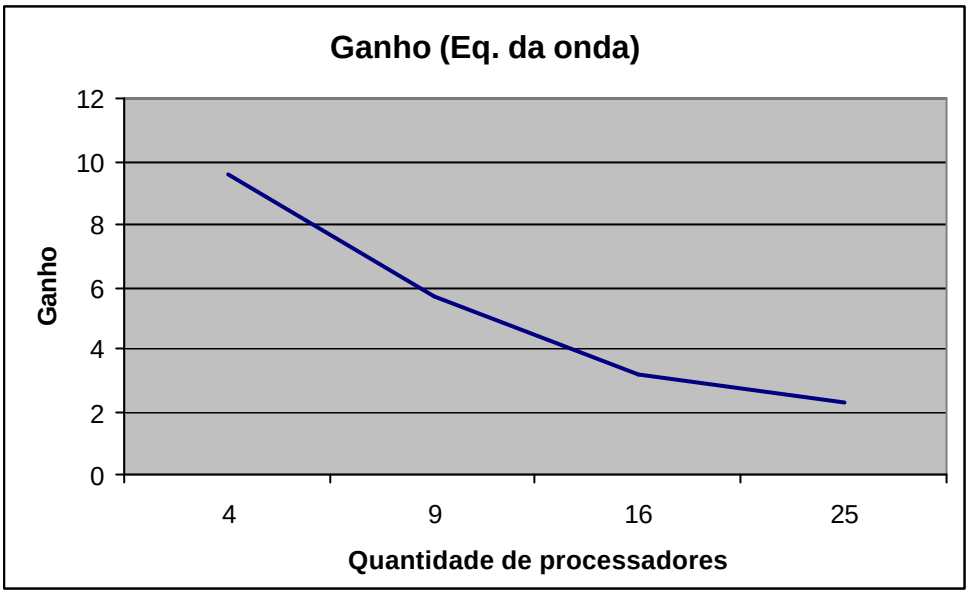


Figura 75 - Ganho do programa com balanceamento de carga quando comparado com a versão sem balanceamento de carga

O "speedup", a eficiência e o ganho apresentaram valores elevados, mostrando que o balanceamento de carga foi capaz de obter os resultados com muito mais rapidez. Pode-se ver que essas métricas sofrem, para equação da onda, diminuição mais acentuada de seu valor, ainda que continuem elevados, para as execuções usando maior quantidade de processadores. Uma das causas para esse comportamento parece ser devida ao aumento de comunicações quando a carga é balanceada, uma vez que o particionador *Prepara* não minimiza as comunicações. No caso da equação da onda, pode-se ver da figura 72 que os subdomínios permanecem refinados até o final das iterações, o que não aconteceu com as equações dos testes anteriores, o que certamente ajudou a ocasionar a perda de rendimento neste caso. O ambiente computacional usado foi um "cluster" ligado por rede Ethernet (ver Apêndice D). Para quantidade de processadores maior que 14 (sítio1), a velocidade da comunicação cai de 1 Gigabit para 100 Megabit, o que também concorre para a queda no rendimento quando usando 16 e 25 processadores. Deve-se notar que esse ambiente tem velocidade de transmissão de dados muito menor que as comunicações de um supercomputador, ambiente para o qual o resolutor desenvolvido aqui foi projetado.

O gráfico a seguir mostra o índice médio de desbalanceamento (IMD) durante o balanceamento dinâmico que ocorre ao longo do processamento da EDP para os tamanhos de domínio da tabela 10 da seção 3.8.3, onde o caso usando 4 processadores foi substituído por um caso usando 36 processadores para melhor entender o comportamento do IMD.

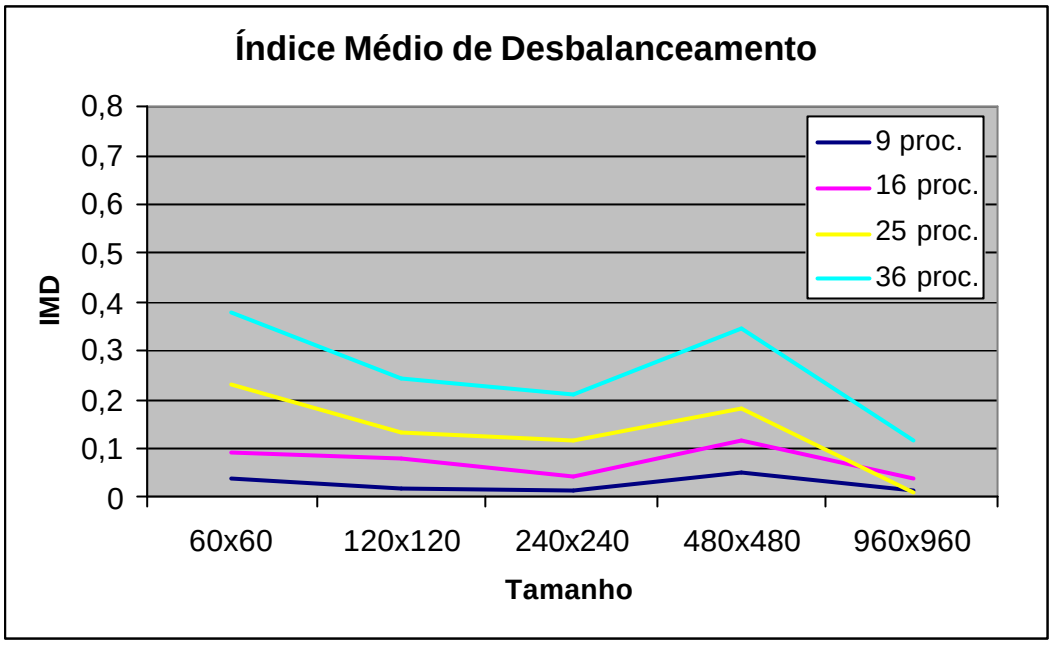


Figura 76 - Índice médio de desbalanceamento

Quanto maior o subdomínio for, mais flexibilidade o particionador tem para balancear a carga. À medida que a quantidade de processadores aumenta, a qualidade do balanceamento diminui porque os subdomínios ficam menores, o que diminui a flexibilidade de distribuir as colunas.

CAPÍTULO 7

CONCLUSÕES

Este trabalho implementou um resolutor de EDPs pelo método Hopscotch, com refinamento adaptativo do domínio e balanceamento dinâmico da carga em malhas estruturadas. O resolutor foi testado usando três EDPs representativas das classes parabólica, elíptica e hiperbólica. O resolutor mostrou a vantagem do refinamento adaptativo com balanceamento dinâmico de carga.

Existem vários parâmetros que o usuário deve introduzir, dependendo da EDP a ser resolvida. Assim, o tamanho da malha, o tempo final de execução, a quantidade de iterações, o tipo de cálculo para determinação de alteração no refinamento (convergência ou gradiente), os valores-gatilho para refinamento e desrefinamento, a quantidade de subdomínios em que o domínio será dividido, o tamanho (m x m) inicial dos subdomínios, a dimensão do domínio e a freqüência em que o refinamento atual deve ser examinado para uma eventual alteração devem ser fornecidos pelo usuário.

Comparações foram realizadas entre o particionador desenvolvido aqui e particionadores que são referência mundial nesse campo, o Chaco, Metis e o Scotch. Essa comparação tem um alcance muito limitado, pois esses particionadores foram construídos para particionar qualquer tipo de malha minimizando as comunicações entre os subdomínios criados, e o que foi desenvolvido aqui tira proveito da estruturação da malha que é utilizada pelo resolutor, particionando a malha de modo a tirar proveito do método Hopscotch, mas sem preocupação em minimizar as comunicações. Assim, nesta situação muito específica, o particionador desenvolvido aqui pôde ter melhor desempenho do que os particionadores citados.

Embora fosse usado o método Hopscotch, o resolutor pode ser adaptado com poucas modificações para diversos métodos explícitos, assim como para aplicações onde uma profundidade, ou altura, possa ser acoplada a cada nó da malha, como pode ser buscado em problemas como o de águas rasas ou meteorologia, por exemplo.

Os resultados mostraram que a versão paralela apresenta um “speedup” e uma eficiência que dependem do tamanho do problema. Quanto maior for o problema, melhores serão o “speedup” e a eficiência.

O particionador *Prepara* foi capaz, no caso particular de malhas estruturadas, de particionar os domínios em tempos melhores que os particionadores mais usados. Seu IMD foi melhor quando os subdomínios foram maiores.

A versão com balanceamento de carga foi bem mais rápida, apresentando tempos de processamento bem menores. Apesar do ambiente computacional usado nos testes não ser aquele para que o resolutor foi planejado, os resultados mostram a eficácia do balanceamento de carga em diminuir os tempos de processamento. Como já mencionado, o balanceamento dinâmico deixa o problema com menos pontos na média, mas sendo o problema de natureza síncrona, sem o balanceamento de carga o tempo seria ditado pelo processador mais carregado.

Apesar de este trabalho ter demonstrado a viabilidade do balanceamento dinâmico com o Hopscotch, muito trabalho deve ser ainda realizado para diminuir o erro quando existem refinamentos desiguais do domínio. A passagem de dados entre essas regiões é fonte de instabilidade e deve ser estudada com mais cuidado.

7.1 - Trabalhos Futuros

Os tempos de processamento podem ser reduzidos aplicando-se uma criteriosa análise do código, com a aplicação das técnicas usuais para obter redução do tempo de processamento, o que foi feito até certo ponto aqui por restrições de tempo.

Implementar a capacidade de cada região usar um ?t diferente, dependendo e seu grau de refinamento. Atualmente, todas as regiões usam o ?t da região mais refinada. Essa alteração melhorará o desempenho do resolutor.

Pesquisar maneiras de diminuir o erro na comunicação entre regiões com refinamentos diferentes. A diminuição desse erro, além de motivar uma maior precisão na resposta, permitirá que se possa usar mais do que três níveis de refinamento.

APÊNDICE A

Uma breve descrição do METIS

Uma cópia do METIS foi baixada e alguns testes foram realizados. O manual de utilização do Metis pode ser encontrado em [52].

A malha a ser particionada é colocada em um arquivo na forma de um vetor de nós. Uma malha de *n* elementos é guardada em um arquivo de texto que contém *n* + 1 linhas. A primeira linha contém a informação do tamanho e o tipo da malha. As restantes *n* linhas guardam os números dos nós que definem cada elemento da malha. Como exemplo, foi criada uma malha estruturada, com os nós numerados de 1 a 36 e os elementos numerados em negrito de 1 a 25.

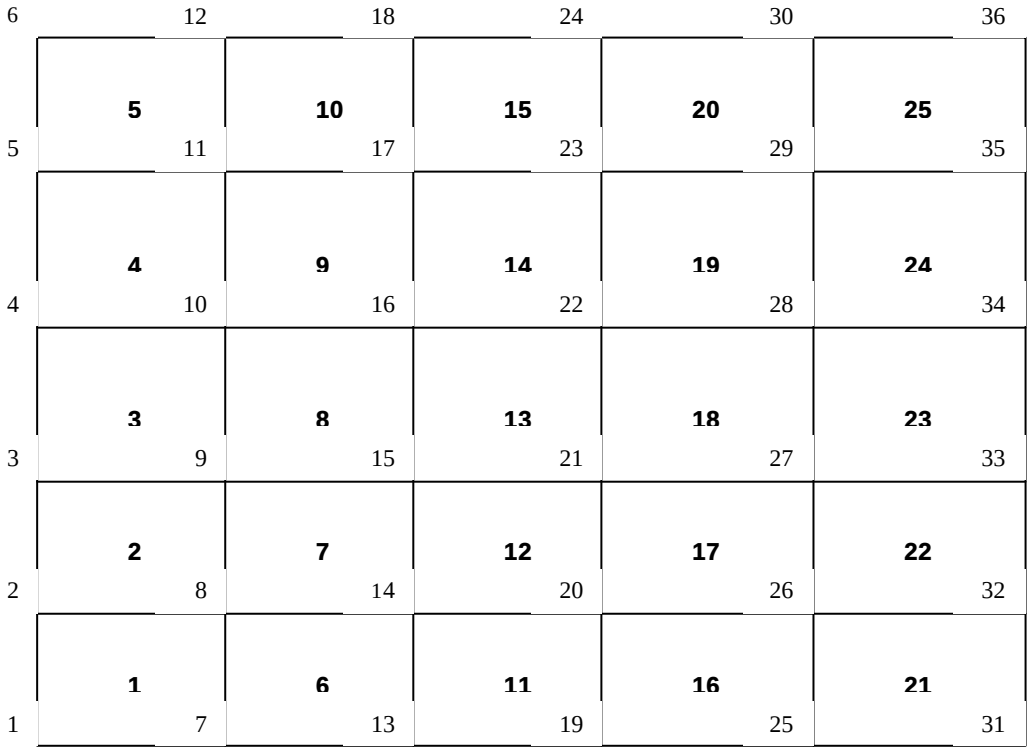


Figura A.1 – Malha estruturada exemplo

O arquivo de entrada para esta malha é:

```
25 4
1 2 8 7
2 3 9 8
3 4 10 9
4 5 11 10
5 6 12 11
7 8 14 13
8 9 15 14
9 10 16 15
10 11 17 16
11 12 18 17
13 14 20 19
14 15 21 20
15 16 22 21
```

```
16 17 23 22
17 18 24 23
19 20 26 25
20 21 27 26
21 22 28 27
22 23 29 28
23 24 30 29
25 26 32 31
26 27 33 32
27 28 34 33
28 29 35 34
29 30 36 35
```

Figura A.2 - Arquivo de entrada da malha do figura A.1

O comando para execução é:

```
/metis-4.0> pmetis Graphs/t01.mesh 2
```

O programa gera a seguinte informação em caso de sucesso.

```
**********************************************************************
  METIS 4.0.1 Copyright 1998, Regents of the University of Minnesota

Mesh Information ---------------------------------------------------
  Name: Graphs/t01.mesh, #Elements: 25, #Nodes: 36, Etype: QUAD

Partitioning Nodal Graph... ----------------------------------------
  2-way Edge-Cut: 6, Balance:  1.04

Timing Information -------------------------------------------------
  I/O:          0.000
  Partitioning: 0.000
**********************************************************************
```

Figura A.3 – Saída com informações da execução do Metis

O arquivo de saída contém o particionamento da malha original com *n* nós consiste de *n* linhas com um único número por linha. A *i*-ésima linha do arquivo contém o número da partição a que o *i*-ésimo vértice pertence. As partições começam a partir de zero até o número de partições menos um.

A malha particionada é mostrada abaixo.

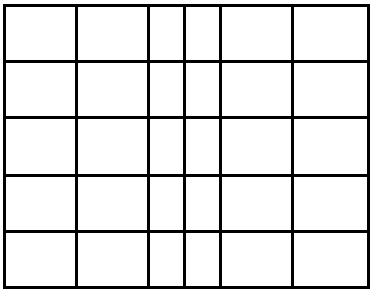


Figura A.4 - Malha particionada em duas partições

Agora, para fins de comparação, será usada a malha da figura 4 da seção 2.4, com 15.606 nós. Existem duas opções de particionamento, `pmetis` e `kmetis`. O comando de execução é:

```
/metis-4.0> pmetis Graphs/4elt.graph 9
```

ou

```
/metis-4.0> kmetis Graphs/4elt.graph 9
```

Esse comando executa `pmetis` ou `kmetis` para particionar a malha *4elt* que está no arquivo `4elt.graph` no diretório `Graphs`.

Segundo [52], o programa `pmetis` é baseado no particionamento multinível recursivo descrito em [47], enquanto `kmetis` á baseado no particionamento multinível *k*-way descrito em [50]. Ainda segundo [52], `kmetis` é preferível se a quantidade de subdomínios desejados for maior que 8. Nesses casos, `kmetis` é mais rápido do que `pmetis`.

Nas figuras A.5 e A.6 a seguir são apresentadas como exemplo, duas saídas resultantes do particionamento da malha *4elt* em 36 subdomínios, usando `pmetis` e `kmetis`.

```
*********************************************************************
  METIS 4.0.1 Copyright 1998, Regents of the University of Minnesota

Graph Information ---------------------------------------------------
  Name: Graphs/4elt.graph, #Vertices: 15606, #Edges: 45878, #Parts: 36

K-way Partitioning... -----------------------------------------------
  36-way Edge-Cut:    1939, Balance:  1.03

Timing Information --------------------------------------------------
  I/O:                0.020
  Partitioning:       0.010   (KMETIS time)
  Total:              0.030
*********************************************************************
```

Figura A.5 – Saída da execução do `kmetis`

```
*********************************************************************
  METIS 4.0.1 Copyright 1998, Regents of the University of Minnesota

Graph Information ---------------------------------------------------
  Name: Graphs/4elt.graph, #Vertices: 15606, #Edges: 45878, #Parts: 36

Recursive Partitioning... -------------------------------------------
  36-way Edge-Cut:    1910, Balance:  1.00

Timing Information --------------------------------------------------
  I/O:                0.010
  Partitioning:       0.050   (PMETIS time)
  Total:              0.060
*********************************************************************
```

Figura A.6 – Saída da execução do `p-metis`

Os tempos para particionamento em 9, 16, 25, 36 e 256 subdomínios estão na tabela A.1 a seguir.

Tabela A.1 – Tempos de execução

Quantidade de subdomínios	Tempo (s) `pmetis`	Tempo (s) `kmetis`
9	0,04	0,03
16	0,05	0,02
25	0,07	0,03
36	0,06	0,03
256	0,10	0,06

A tabela 12 da seção 6.6.2 apresenta uma comparação muito limitada, pelos motivos expostos na seção 6.6, entre os particionadores Chaco, Metis, Scotch e o particionador deste trabalho, o *Prepara*.

APÊNDICE B

Uma breve descrição do CHACO

Chaco é um pacote projetado para particionar grafos. Ele aplica recursivamente vários métodos para encontrar o mínimo corte em grafos valorados. Eles incluem os métodos inercial, espectral, Kernigham-Lin e multinível além de várias estratégias mais simples. Cada um desses métodos pode ser usado para particionar o grafo em dois, quatro ou oito peças em cada nível de recursão. O manual de utilização do programa pode ser encontrado em [42].

Uma versão foi baixada do sítio informado em 4.3.1 e instalada. Como exemplo, foi criada uma malha 20 x 20 para ser particionada. A seguir estão uma saída padrão do software, e o resultado da aplicação de um dos particionadores disponíveis no pacote.

O comando para execução é:

/Chaco-2.2/exec> chaco

Após esse comando o programa interativamente vai interrogando o usuário para a entrada dos parâmetros necessários. A seguir é mostrado um exemplo.

```
                    Chaco 2.0
          Sandia National Laboratories

Reading parameter modification file `User_Params'

Graph input file: grid20x20.graph
Assignment output file: grid20x20.out
Global partitioning method:
  (1) Multilevel-KL
  (2) Spectral
  (3) Inertial
  (4) Linear
   (5) Random
  (6) Scattered
  (7) Read-from-file
4

Local refinement method:
  (1) Kernighan-Lin
  (2) None
2

Total number of target hypercube dimensions: 2

Partitioning dimension:
  (1) Bisection
  (2) Quadrisection
2

           Input and Parameter Values
Graph file: `grid20x20.graph', # vertices = 400, # edges = 760
Global method: Linear
Local method: None
Partitioning target: 2-dimensional hypercube
Partitioning mode: Quadrisection
```

```
Random seed: 7654321
Assignment output file: `grid20x20.out' (normal format)
Active Parameters:
  CHECK_INPUT = True
  OUTPUT_METRICS = 2
  MAKE_VWGTS = False
  REFINE_MAP = False
  REFINE_PARTITION = 0
  INTERNAL_VERTICES = False
  DEBUG_PARAMS = 2

Starting to partition ...

         Partitioning Results
After level 1  (nsets = 4):
                             Total      Max/Set      Min/Set
                             -----      -------      -------
Set Size:                     400          100          100
Edge Cuts:                     60           40           20
Hypercube Hops:                80           60           20
Boundary Vertices:            120           40           20
Boundary Vertex Hops:         160           60           20
Adjacent Sets:                  6            2            1
Internal Vertices:            280           80           60
```

Figura B.1 – Saída com informações sobre a execução do Chaco

Assim como no MeTiS, o arquivo de saída que contém o particionamento da malha original com *n* nós consiste de *n* linhas com um único número por linha. A *i*-ésima linha do arquivo contém o número da partição a que o *i*-ésimo vértice pertence. As partições começam a partir de zero até o número de partições menos um.

Dependendo do método de particionamento escolhido, Chaco particiona levando em consideração a arquitetura da rede de ligação dos processadores sendo usada. Se a rede for baseada em um hipercubo, ele otimiza o particionamento para esse tipo de ligação dos processadores, quando então o usuário informa a dimensão do hipercubo. Senão, o usuário informa simplesmente a quantidade de partições que deseja.

Agora, para fins de comparação, será usada a malha *4elt* da figura 4 da seção 2.4, com 15.606 nós. Existem várias opções de particionamento. Foram testadas as opções para particionamento multinível, espectral e linear, e os tempos variaram entre 0,032s e 0,66s para um particionamento em 36 subdomínios. A seguir é apresentada como exemplo a saída de um particionamento da malha *4elt* em 36 subdomínios usando o método multinível.

```
            Input and Parameter Values

Graph file: `4elt.graph', # vertices = 15606, # edges = 45878
Global method: Multilevel-KL
Number of vertices to coarsen down to: 100
Eigen tolerance: 0.001
Local method: Kernighan-Lin
Partitioning target: 1-dimensional mesh of size 36
Partitioning mode: Bisection
Random seed: 7654321
```

```
Assignment output file: `a' (normal format)
Active Parameters:
  CHECK_INPUT = True
  LANCZOS_TYPE:  Selective orthogonalization
  EIGEN_TOLERANCE = 0.001
  SRESTOL = -1 ... autoset to square of eigen tolerance
  LANCZOS_MAXITNS = -1 ... autoset to twice # vertices
  LANCZOS_SO_PRECISION = 2 ... double precision
  LANCZOS_SO_INTERVAL = 10
  LANCZOS_CONVERGENCE_MODE = 0 ... residual tolerance
  BISECTION_SAFETY = 10
  LANCZOS_TIME = 0 ... no detailed timing
  WARNING_EVECS = 2
  MAPPING_TYPE = 1 ... min-cost assignment
  MAKE_CONNECTED = True
  PERTURB = False
  COARSEN_RATIO_MIN = 0.7
  COARSE_NLEVEL_KL = 2
  MATCH_TYPE = 1
  HEAVY_MATCH = False
  COARSE_KL_BOTTOM = True
  COARSEN_VWGTS = True
  COARSEN_EWGTS = True
  KL_ONLY_BNDY = True
  KL_RANDOM = True
  KL_METRIC = Hops
  KL_NTRIES_BAD = 1
  KL_BAD_MOVES = 20
  KL_UNDO_LIST = True
  KL_IMBALANCE = 0
  TERM_PROP = False
  OUTPUT_METRICS = 2
  MAKE_VWGTS = False
  REFINE_MAP = False
  REFINE_PARTITION = 0
  INTERNAL_VERTICES = False
  DEBUG_PARAMS = 2

Starting to partition ...

                      Partitioning Results

After full partitioning  (nsets = 36)

                            Total      Max/Set      Min/Set
                            -----      -------      -------
Set Size:                   15606          434          433
Edge Cuts:                   1979          217           73
Mesh Hops:                   8588         1358          107
Boundary Vertices:           2053          115           38
Boundary Vertex Hops:        9037          716           57
Adjacent Sets:                148           12            2
Internal Vertices:          13661          397          330

Total time: 0.108007 sec.
  input 0.016001
  reformatting 8.40257e-18
  checking input 0.004
  partitioning 0.084005
  evaluation 0.004001
  printing assignment file 4.49944e-18
```

```
KL time: 0.040004 sec.
  initialization 2.11094e-16
  nway refinement 0.040004
    bucket sorting 0.012

Coarsening 0.036001 sec.
  maxmatch 0.004
  makecgraph 0.024

Lanczos time: 0.004 sec.
```

Figura B.2 – Saída com informações sobre a execução do Chaco no particionamento do grafo *4elt*

Os tempos para particionamento em 9, 16, 25 e 36 e 256 subdomínios estão na tabela a seguir.

Tabela B.1 – Tempos de execução

Quantidade de subdomínios	Tempo (s)
9	0,068
16	0,080
25	0,080
36	0,092
256	0,204012

A tabela 12 da seção 6.6.2 apresenta uma comparação muito limitada, pelos motivos expostos na seção 6.6, entre os particionadores Chaco, Metis, Scotch e o particionador deste trabalho, o *Prepara*.

APÊNDICE C

Uma breve descrição do SCOTCH

Scotch é um software e bibliotecas [69, 70, 71] para particionamento de grafos, malhas e hipergrafos. Sua meta é o mapeamento estático de grafos, usando a técnica "Dual Recursive Bipartitioning". Ele usa a abordagem de dividir e conquistar para recursivamente alocar processos a processadores [69]. Foi desenvolvido no Laboratoire Bordelais de Recherche em Informatique, na França, que faz parte do Institut National de Recherche em Informatique et em Automatique (INRIA). O manual do programa pode ser encontrado em [70].

Uma versão foi baixada do sítio informado em 4.3.1 e instalada. Como exemplo, foi criada uma malha 32 x 32 para ser particionada em 8 subdomínios.

O comando para execução é:

```
/scotch_5.1> gmk_m2 32 32 | gmap – tgt/h8.tgt src/scotch/tmp/bro1.map –vt
```

A seguir está a informação de tempo de processamento.

```
T Mapping 0.565209
T I/O   0.0904648
T Total 0.655674
```

Figura C.1 – Tempos de processamento

Agora, para fins de comparação, será usada a malha da figura 4 da seção 2.4, com 15.606 nós , para um particionamento em 36 subdomínios.

O comando para execução do particionamento da malha *4elt* em 36 subdomínios é:

```
/scotch_5.1> gmap grf/4elt.grf tgt/k36.tgt src/scotch/tmp/bro1.map –vt
```

A seguir é apresentada a saída produzida pela execução do programa.

```
T Mapping 0.225463
T I/O   0.0231214
T Total 0.248585
```

Figura C.2 – Tempos de processamento no particionamento do grafo *4elt*

Os tempos para particionamento em 9, 16, 25, 36 e 256 subdomínios estão na tabela a seguir.

Tabela C.1 – Tempos de execução

Quantidade de subdomínios	Tempo (s)
9	0,144549
16	0,187325
25	0,221226
36	0,248585
256	0,476357

A tabela 12 da seção 6.6.2 apresenta uma comparação muito limitada, pelos motivos expostos na seção 6.6, entre os particionadores Chaco, Metis, Scotch e o particionador deste trabalho, o *Prepara*.

APÊNDICE D

Ambiente dos Testes

Os testes foram conduzidos em um total de 28 máquinas em um ambiente dividido em três sítios. As máquinas têm instalados Linux Fedora Core 2 e LAM/MPI 7.1.4. Todas as máquinas têm CPU Pentium IV, com 2,6 GHz e 520 Mb de memória RAM. As máquinas pertencentes a cada sítio têm interconexão por "switches" de 1 Gigabit. Os sítios são interligados entre si por rede Ethernet de 100 Mb. O sítio 1 contém 14 máquinas, o sítio 2 contém 8 máquinas e o sítio 3 contém 6 máquinas. A figura a seguir ilustra a topologia da instalação.

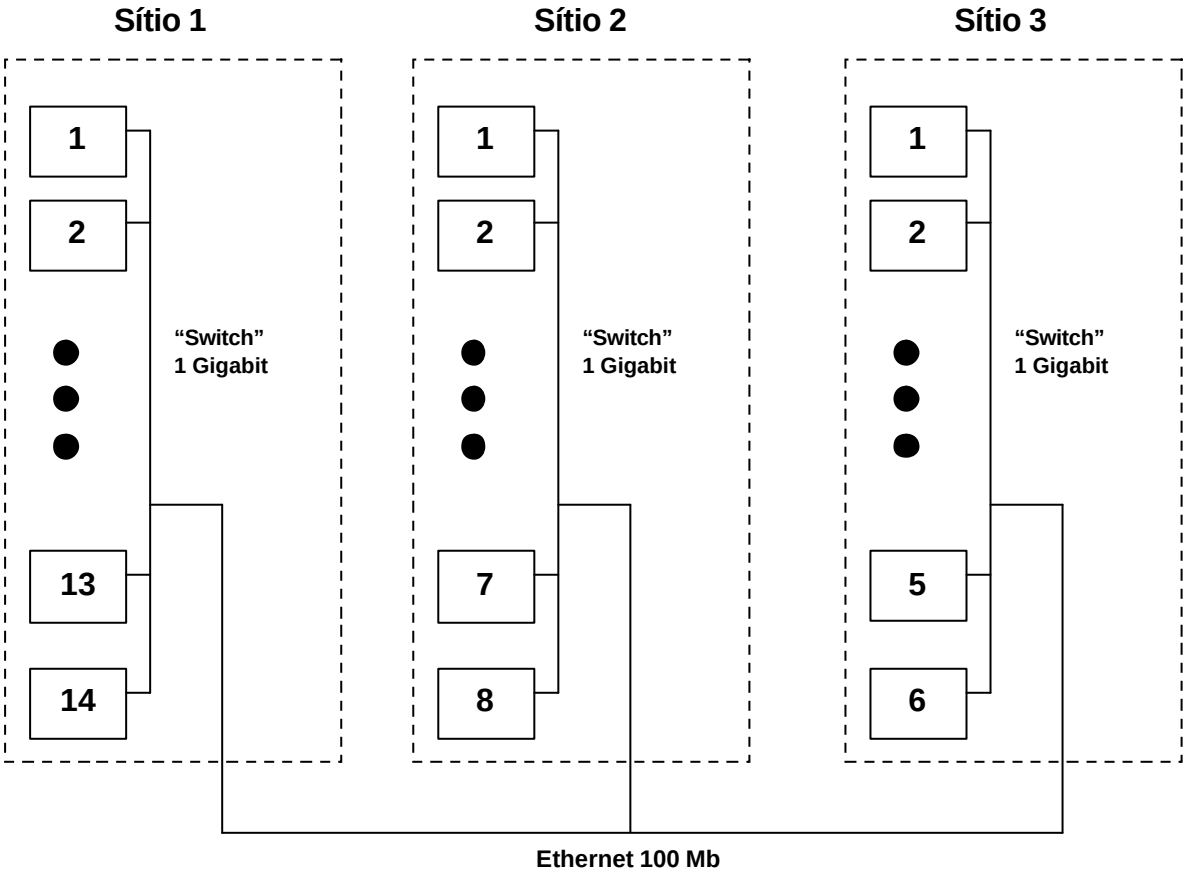


Figura D.1 – Topologia do "cluster " utilizado nos testes

BIBLIOGRAFIA

[1] Alden, J., *Computational Electrochemistry*, Tese de Doutorado, University of Oxford, Physical and Theoretical Chemestry Laboratory, Reino Unido, http://physchem.ox.ac.uk/~rgc/john/Thesis , 1998.

[2] Alpert, C.; Yao, S., *Spectral Partitioning: The More Eigenvectors, The Better*, 32nd ACM/IEEE Design Automatic Conference, pp. 195-200, 1995.

[3] Amdahl, G.M., *Validity of Single-processor approach to Achieving Large Scale Computing Capabilities*, AFIPS Conference Proceedings vol. 30, 1967.

[4] Bailey D., *CS267-Applications of Parallel Computers*, Berkeley Lab Computing Division, Computational Research Division (CRD), notas de aula, hospedado em http://crd.lbl.gov/~dhbailey/cs267/Lectures/Lect_13_2000.pdf, 2000.

[5] Basaruddin T., *Implementation of Parallel ADI and Hopscotch Methods on KSR1*, Proceedings of the 7th SIAM Conference on Parallel Processing for Scientific Computing, pp. 500-501, 1995.

[6] Battiti R.; Bertossi A., *Greedy, Prohibition, and Reactive Heuristics for Graph Partitioning*, IEEE Transactions on Computers, vol 48, 361-385, 1999.

[7] Battiti R., et alli, Università di Trento, Itália, lista de grafos teste, hospedado em http://lion.dit.unitn.it/reactive-search/graph-links.html, 2007.

[8] Beattie, P.D.; Wellington R.G., Girault H.H., *Cyclic voltammetry for assisted ion transfer at an ITIES*, Journal of Electroanalytical Chemestry, 396, 317-323, 1995.

[9] Berger, M.; Oliger, J., *Adaptive mesh refinement for hyperbolic partial differential equations*. J. Comp. Phys., 53, 484-512, 1984.

[10] Blom, J.G.; Boonkkamp, T.; J.H.M. ten; Verwer, J.G., *On the odd-even hopscotch scheme for the numerical integration of time-dependent partial differential equations*, Applied Numerical Mathematics, *3*(4), 361-362, 1987.

[11] Boman, E.; Devine, K., *Tutorial: Partitioning and Load Balancing with Zoltan and Isorropia*, Sandia National Laboratories, EUA, notas de aula, hospedado no sítio http://www.hpcsw.org/presentations/workshops/scalable_tools/boman.pdf, 2007.

[12] Bradie, B., *An Introduction to Numerical Analysis with Applications to the Physical, Natural and Social Sciences – Source Code Archive*, Christopher Newport University, Department of Physics, Computer Science & Engineering, EUA, hospedado em http://www.pcs.cnu.edu/~bbradie/cinterpolation.html, 2006.

[13] Brown, D.; Henshaw, W.; Quinlan, D., *Overture: An Object-oriented framework for solving partial differential equations*, Lecture Notes in Computer Science, vol. 1343, pp. 177-184, 2006.

[14] Bui, T.N.; Moon, B.R., *Genetic Algorithm and Graph Partitioning*, IEEE Transactions on Computers, Vol. 45, No. 7, July 1996.

[15] Cabral, F. L., *Métodos Hopmoc para Resolução de Equações de Convecção-Difusão e sua Implementação Paralela*, Dissertação de Mestrado, UFF, 2001.

[16] Carvalho, A.P., *Balanceamento de Carga de Aplicações Paralelas SPMD*, Tese de Doutorado, PUC-RJ, 2000.

[17] Christou, I. T.; Meyer, R.R., *Optimal Equi-partition of Rectangular Domains for Parallel Computation*, Journal of Global Optimization, vol. 8, no. 1, pp. 15-34, ftp://ftp.cs.wisc.edu/math-prog/tech-reports/95-19.ps.Z, 1996.

[18] Debreu, L.; Vouland, C.; Blayo, E., *AGRIF: Adaptive Grid Refinement In Fortran*, Computers & Geosciences, vol. 34, issue 1, pp. 8-13, 2008.

[19] Devine, K.; Hendrickson, B; Boman, E.; St. John, M.; Vaughan, C., *Zoltan: Parallel Partitioning, Load Balancing and Data-Manegement Services*, Sandia National Laboratories, manual de utilização, hospedado em http://www.cs.sandia.gov/Zoltan, 2007.

[20] Devine, K.D. et alli, New *Challenges in Dynamic Load Balancing*, Appl. Numer. Math, vol. 52, 2005.

[21] Devine, K.; Flaherty, J., *Parallel adaptive hp-refinement techniques for conservation laws*, Appl. Numer. Math. 20, 367-386, 1996.

[22] Diekmann; Lüling; Monien; Spräner, *Combining Helpful Sets and Parallel Simulated Annealing for the Graph-Partitioning Problem*, Parallel Algorithms and Applicators (PAA), Vol. 8, pp. 61-84, 1996.

[23] Diersch, H.-J.G., *FEFLOW White Papers Vol. 1*, Documentação do aplicativo FEFLOW, WASY, Institute for Water Resources Planning and Systems Research Ltd., Berlim, Alemanha, hospedado em http://geos.gsi.gov.il/vladi/FEFLOW/help/general/theory/whitepapers/error_norms/enornorm.html, 2002.

[24] Donaldson, W.W., *Grid-Graph Partitioning*, Tese de Doutorado, University of Wisconsin-Madison, EUA, 2000.

[25] Donaldson, W.W.; Meyer, R.R., *Generating Locally Optimal Partitions for the Regular Grid-Graph Problem*, Department of Computer Sciences, University of Wisconsin-Madison, EUA, http://www.cs.wisc.edu/~wwd/locopt.pdf, 2001.

[26] Donaldson W.W., Meyer R.R., *A Dynamic-Programming Heuristic for Regular Grid-Graph Partitioning*, Department of Computer Sciences, University of Wisconsin-Madison, EUA, http://pages.cs.wisc.edu/~wwd/rev4.pdf, 2000.

[27] El-Giar, O.; Hopkins, T., *The Parallel Solution of Linear Elliptic Equations using Hopscotch Algorithms*, Technical Report 62, Computing Laboratory, University of Kent, Canterbury, Kent, Reino Unido, 1989.

[28] Esen, A.; Kutluay, S., *A Numerical Solution of the Stefen problem with a Neumann-type boundary condition by enthalpy method*, Applied Mathematics and Computation 148, 321-329, 2004.

[29] Espenchitt D.G., *Segmentação de Dados em um Número Desconhecido de Grupos Utilizando Algoritmo De Colônia De Formigas*, Tese de Doutorado, COPPE-UFRJ, 2008**.**

[30] Finney, R. L., *Calculus*, Ed. Addison-Wesley Inc, EUA, 1994.

[31] Flaherty, J.; Loy, R.; Shephard, M.; Szymanski, B.; Teresco J.; Ziantz, L., *Adaptive local refinement with octree load-balancing for the parallel solution of three dimensional conservation laws*, J. Parallel Distrib. Comput. 47 (2), 139-152, 1998.

[32] Fortuna, A. O., *Técnicas Computacionais para Dinâmica dos Fluidos*, Ed. Edusp, São Paulo, 2000.

[33] Gaidamauskaitè, E.; Baronas, R., A *Comparison of Finite Differences Schemes for Computational Modelling of Biosensors*, Nonlinear Analysis: Modelling and Control, Vol. 12, No 3, 359-369, 2007.

[34] Golub, G.H.; Ortega, J.M., *Scientific Computing and Differential Equations, an Introduction to Numerical Methods*, Academic Press, Ed. 2, 1992.

[35] Gourlay, A.R., *Hopscotch: a fast second order partial differential equation solver*, J. Inst. Maths. Applics., 6:375-390, 1970.

[36] Gourlay, A.R.; McGuire, G.R., *General hopscotch algorithms for the numerical solution of partial differential equations*, J. Inst. Maths. Applics., 7:216-227, 1971.

[37] Gourlay, A.R.; Morris J.L., *Hopscoch Difference Methods for Nonlinear Hyperbolic Systems*, IBM Journal of Research and Development, Volume 16, Number 4, Page 349, 1972.

[38] Guedes, M. et alli, *Solução Numérica de Equações Diferenciais Parciais Parabólicas usando o Método Hopscotch com Refinamento Não-Uniforme*, Anais Eletrônicos do XXVIII CNMAC, São Paulo, 2005.

[39] Gustafson, J. L., *Reevaluating Amdahl's Law*, Communications of the ACM, vol. 31, pp. 532-533, 1988.

[40] Hartman-Baker, R., *A Crash Course in Supercomputing: MPI*, Oak Ridge National Laboratory, U.S. Department of Energy, EUA, notas de aula, hospedado no sítio http://www.csm.ornl.gov/~hqi/CrashCourse07/4-Project07-2up.pdf, 2006.

[41] Hendrickson, B.; Leland, R., *A multilevel algorithm for partitioning graphs*, Proceedings of the 1995 ACM/IEEE Conference on Supercomputing, Article no. 28, 1995.

[42] Hendrickson, B.; Leland, R., *The Chaco user's guide*, version 2.0, Tech. Rep. SAND94-2692, Sandia National Laboratories, Albuquerque, NM, EUA,1994.

[43] Hosking, R.J.; Joe, S.; Joyce, D.C; Turner, J.C., *First Steps in Numerical Analysis*, ed. Rainer Radok, 2ª edição, hospedado em http://kr.cs.ait.ac.th/~radok/math/mat7/step28.htm, 2003.

[44] Hu, Y.; Blake, R., *An optimal dynamic load balancing algorithm*, Tech. Report DL-P-95-011, Daresbury Laboratory, Warrington, Reino Unido, 1995.

[45] Hu, Y.F.; Blake, R.J., *Load **B**alancing for Unstructured Mesh Applications*, Progress in Computer Research, ISBN:1-59033-011-0, hospedado em www.dl.ac.uk/TCSC/Staff/Hu_Y_F/PROJECT/pdcp_siam/node10.html, 2001.

[46] International Center for Numerical Methods in Enginnering, *Structured Meshes*, hospedado em http://gid.cimne.upc.es/support_team/gidbeta/gid94.html, 2008.

[47] Karypis, G.; Kumar, V., *A fast and high quality multilevel scheme for partitioning irregular graphs*, Technical Report CORR 95{035), University of Minnesota, Dept. Computer Science, Minneapolis, MN, EUA, 1995.

[48] Karypis, G.; Kumar, V., *ParMETIS: Parallel graph partitioning and sparse matrix ordering library*, Tech. Rep. 97-060, Department of Computer Science, University of Minnesota, EUA, http://www.cs.umn.edu/~metis , 1997.

[49] Karypis, G., *Multilevel algorithms for multi-constraint hypergraph partitioning*, Technical Report TR 99-034, Department of Computer Science, University of Minnesota, EUA, 1999.

[50] Karypis, G.; Kumar, V., *Multilevel k-way partitioning scheme for irregular graphs*, Journal of Parallel and Distributed Computing, 48(1):96–129, 1998.

[51] Karypis, G.; Kumar, V., *A fast and highly quality multilevel scheme for partitioning irregular graphs*, SIAM Journal on Scientific Computing, vol. 20, no. 1, pp. 359-392, 1998.

[52] Karypis, G.; Kumar V., *METIS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-reducing Orderings of Sparse Matrices*, manual de aplicativo, University of Minnesota / Army HPC Research Center, EUA, 1998.

[53] Kernighan, B.W.; Lin, S., *An Efficient Heuristic Procedure for Partitioning Graphs*, The Bell system technical journal, Vol. 49, No. 1, pp. 291-307, 1970.

[54] Khalifa, A.K.; Farea, H.A., *A stable finite difference technique for nerve conduction equations*, J. Egypy. Math. Soc., Vol.10(1) pp 19-28, Cairo, Egito, 2002.

[55] Kurtinaitis, A.; Ivanauskas, F., *Finite Difference Solution Methods for a System of the Nonlinear Schrödinger Equations*, Nonlinear Analysis: Modelling and Control, Vol. 9, No. 3, 247–258, 2004.

[56] Korosec, P.; Silc, J.; Robic, B., *Solving the mesh-partitioning problem with an ant-colony algorithm*, Parallel Computing 30, 785–801, 2004.

[57] Luke, E. A., *Defining and Measuring Scalability*, Proceedings of Scalable Parallel Libraries Conference, pp. 183-186, hospedado em http://www.erc.msstate.edu/~lush/publications/scale.ps.gz, 1994.

[58] Martin, D.F.; Cartwright, K.L., *Solving Poisson's Equation using Adaptive Mesh Refinement*, University of California, Berkeley, Technical Report No. UCB/ERL M96/66, 1996.

[59] Message Passing Interface Forum, *3.6. Buffer allocation and usage*, forum de usuários MPI, hospedado em www.mpi-forum.org/docs/mpi-11-html/node42.html#node42, 1997.

[60] Mitchell, A.R.; Griffiths, D.F., *The Finite Difference Method in Partial Differential Equations*, Ed. John Wiley & Sons Ltd., Reino Unido, 1980.

[61] Mitchell, W.F., *Refinement tree based partitioning for adaptive grids*, in: Proc. Seventh SIAM Conf. on Parallel Processing for Scientific Computing, SIAM, pp. 587-592, 1995.

[62] Monien, B.; Schamberger, S., *Graph Partitioning with the Party Library: Helpful-Sets in Practice*, Proceedings of the 16th Symposium on Computer Architeture and High Performance Computing, pp. 198-205, 2004.

[63] Mulet, P.; Baeza A., *Highly Accurate Conservative finite difference schemes and adaptive* mesh *refinement techniques for hyperbolic systems of conservation laws*, Numerical Mathematics and Advanced Applications, ISBN: 978-3-540-34287-8, pp.

198-206, hospedado no sítio
http://benasque.edm.ub.es/2007pde/talks_contr/3110benasque07_baeza.pdf, 2007.

[65] National Center for Supercomputing Applications (NCSA), *Message-passing Standards: MPI and PVM*, University of Illinois, tutorial de curso "on-line", hospedado em www.ncsa.uiuc.edu/Useinfo/Resources/Hardware/CommonDoc/MessPass, 2008.

[66] Oliveira, S.R.F., *Análise da Convergência do Método HopMoc para uma Equação de Convecção-Difusão*, Tese de Mestrado, Universidade Federal Fluminense, 1995.

[67] Parashar, M.; Yotov, I., *An Environment for Parallel Multi-Block, Multi-Resolution reservoir Simulations*, Proceedings of the 11th International Conference on Parallel and Distributed Computing and Systems (PDCS 98), pp. 230-235, http://www.ece.rutgers.edu/~parashar/Papers/pdcs98.PDF, 1998.

[68] Patra, A.; Oden, J.T., *A Parallel Adaptive Strategy for hp finite element computations*, Tech. Rep. 94-01, TICAM, University of Texas; Austin, Texas, EUA, 1994.

[69] Pellegrini, F.; Roman, J., *Experimental Analysis of the Duak Recursive Bipartitioning Algorithm for Static Mapping*, Université Bordeaux I, Research Report 1038-96, França, 1996.

[70] Pelligrini, F., *SCOTCH 5.0 user's guide*, manual de aplicativo, Université Bordeaux I, França, 2007.

[71] Pellegrini F.; Roman, J., *SCOTCH: A Software package for static mapping by dual recursive bipartitioning of process and architecture graphs*, Proceedings of High-Performance Computing and Networking (HPCN'96), LNCS 1067, pp. 493-498, 1996.

[72] Pilkington, J.R.; Baden, S.B., *Partitioning with spacefilling curves*, CSE Technical Report CS94-349, Dept. Computer Science and Engineering, University of California, San Diego, CA, EUA, 1994.

[73] Pothen, A.; Simon, H.; Liou, K., *Partitioning sparse matrices with eigenvectors of graphs*, SIAM J. Matrix Anal. 11 (3), 430-452, 1990.

[74] Preis, R., *The PARTY Graphpartitioning- Library User Manual – Version 1.99*, manual de aplicativo, Universidade de Padeborn, Alemanha, 1998.

[75] Preis, R.; Diekmann, R., *The PARTY partitioning library, user guide version 1.1*, Tech. Rep. tr-rsfb-96-024, Dept. of Computer Science, University of Paderborn, Paderborn, Alemanha, 1996.

[76] Rebonatto, M.T., *Processamento Paralelo e Distribuído – Medidas de Desempenho e Obtenção de Resultados de Aplicações Paralelas*, notas de aula, hospedado em http://usuarios.upf.br/~rebonatto/procpar/procpar_formas.pdf, Universidade de Passo Fundo, 2002.

[77] Romero, L.F.; Zapata, E.L., Ramos J.I., *Parallel Computing of Semiconductor Laser* Equations, Proceedings of 8th SIAM Conf. on Parallel Processing for Scientific Computing, Minnesota, MN, 1997.

[78] Schloegel, K.; Karypis, G.; Kumar, V. *Multilevel difusion algorithms for repartitioning of adaptive meshes*, Journal of Parallel and Distributed Computing 47 (2), 109-124, 1997.

[79] Schloegel, K.; Karypis, G.; Kumar, V., *Parallel static and dynamic multiconstraint graph partitioning*, Concurrency and Computation - Practice and Experience 14 (3), 219-240, 2002.

[80] Schloegel, K; Karypis, G.; Kumar, V., *Graph Partitioning for Dynamic, Adaptive and Multi-phase Scientific Simulations*, Proccedings of the 2001 IEEE International Conference on Cluster Computing, pp. 271-273, http://www.cacr.calltech.edu/cluster2001/program/talks/kumar.pdf, 2001.

[81] Silva, L.A.P., *Implementação Paralela do algoritmo Gradiente Conjugado utilizando MPI*, UFRGS, RS, hospedado em http://www.inf.ufrgs.br/procpar/disc/cmp134/trabs/T2/021/gc-texto/index.html, 2002.

[82] Simon, H.D., *Partitioning of unstructured problems for parallel processing*, Computing Systems in Egineering (journal), vol 2, pp. 135-148, 1991.

[83] Silva Filho, F.C., *Modelagem de problemas de engenharia: solução de equações diferenciais parciais pelo método dos elementos finitos*, Rev. Tecnol, Fortaleza, vol. 26, no. 2, pp. 134-144, 2005.

[84] Smith, R.; Kidd, A., *Comparative Study of Two Numerical Techniques for the Solution of Viscous Flow in a Driven Cavity*, NASA SP-378, 132 pages, published by NASA, Washington, D.C., EUA, 1975.

[85] Sndergaard, P.; Hansen P.C.**,** *Numerical Computation with Shocks***,** Technical University of Denmark, hospedado no sítio http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.39.3457, e no sítio http://www2.imm.dtu.dk/~pch/Projekter/Shock/main/node17.html, 1999.

[86] Sommeijer, B.P.; Kok, J., *A Vector/Parallel Method for a Three-Dimensional Transport Model Coupled with Bio-Chemical Terms***,** CWI report NM-R9503, CWI

National Research Institute for Mathematics and Computer Science, Amsterdam, Holanda, 1995.

[87] Sommeijer, B.P.; Kok, J., *Splitting Methods for Three-Dimensional Bio - Chemical Transport***,** Applied Numerical Mathematics, vol. 21, no. 3, pp. 303-320, Amsterdam, Holanda, 1996.

[88] Steen, A.J.; Dongarra, J., *Overview of Recent Supercomputers*, Technical report UT-CS-96-325, University of Tenesse, EUA, ftp://cs.utk.edu/pub/TechReports/1996/ut-cs-96-325.ps.Z, 1996.

[89] Taylor, V.E.; Nour-Omid, B., *A study of the factorization fill-in for a parallel implementation of the finite element method*, Int. J. Numer. Math. Eng. 37, 3809-3823, 1994.

[90] Top 500 Supercomputers Sites, *The Main Architectural Classes*, hospedado em http://www.top500.org/2007_overview_recent_supercomputers/main_architectural_classes, 2007.

[91] Tsang, S.C.; Chow, K. W., *The Evolution of Periodic Waves of the Coupled Nonlinear Schrödinger Equations*, Mathematics and Computers Simulation 66, 551-564, 2004.

[92] Vetter, J.; Supinski, B., *Dynamic Software Testing of MPI Applications with Umpire*, ACM/IEEE Conference on Supercomputing, pp. 51-51, 2000.

[93] Verwer; Sommeijer, *Stability Analysis of an Odd-Even-line Hopscotch Method for Three–Dimensional Advection-Difusion Problems*, Society for Industrial and Applied Mathematics, vol 34, No 1, pp. 376-388, 1987.

[94] Wachsman, A., *Concepts of Parallel Computing: Clustering and Shared Memory*, Stanford University High Performance Computing Conference, Stanford University, CA, EUA, hospedado em http://researchcomp.stanford.edu/hpc/archives/HPCparallel.pdf, 2005.

[95] Walshaw C., *A Multilevel Algorithm for Force-Directed Graph-Drawing*, Journal of Graph Algorithms and Applications, vol. 7, no. 3, pp. 253–285, 2003.

[96] Walshaw, C., *The Parallel JOSTLE Library User's Guide, Version 3.0*, manual de aplicativo, Old Royal Naval College, University of Greenwich, London, Reino Unido, 2002.

[97] Walshaw, C.; Cross, M.; McManus, K., *Multiphase mesh partitioning*, App. Math. Modelling 25, 123-140, 2000.

[98] Walshaw, C.; Cross, M., *JOSTLE: Parallel Multilevel Graph-Partitioning Software – An Overview*, Mesh Partitioning Techniques and Domain Decomposition Techniques, pages 27-58, Civil-Comp Ltd., 2007.

[99] Walshaw, C. et alli, *Partitioning & Mapping of Unstructered Meshes to Parallel Machine Topologies*, Lecture Notes in Computer Sciences, Vol. 980, pp. 121-126, 1995.

[100] Warren, M.S.; Salmon, J.K., *A parallel hashed oct-tree n-body algorithm*, Proceedings of the 1993 ACM/IEEE conference on Supercomputing, pp. 12-21, Portland, OR, EUA, 1993.

[101] Weisstein, E.W., *Boundary Conditions*, MathWorld--A Wolfram Web Resource, hospedado em http://mathworld.wolfram.com/BoundaryConditions.html, 2008.