RÉVERTON LUÍS ANTUNES NEUNDORF

DESEMPENHO DE UM ALGORITMO *MULTIGRID* PARALELO APLICADO À EQUAÇÃO DE LAPLACE.

Curitiba

2013

RÉVERTON LUÍS ANTUNES NEUNDORF

DESEMPENHO DE UM ALGORITMO *MULTIGRID* PARALELO APLICADO À EQUAÇÃO DE LAPLACE.

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Ciências, pelo Programa de Pós-Graduação em Métodos Numéricos em Engenharia, Setor de Ciências Exatas e Tecnologia, Universidade Federal do Paraná.

Orientador:
Prof. Dr. Marcio Augusto Villela Pinto

Co-Orientador:
Dr. Leonardo Calvetti

Co-Orientador:
Prof. Dr. Luciano Kiyoshi Araki

Curitiba
2013

N494d Neundorf, Réverton Luís Antunes
Desempenho de um algoritmo multigrid paralelo aplicado à equação de Laplace [manuscrito] / Réverton Luís Antunes Neundorf. – Curitiba, 2013.
137f. : il. [algumas color.] ; 30 cm.

Dissertação (mestrado) - Universidade Federal do Paraná, Setor de Ciência Exatas e Tecnologia, Programa de Pós-graduação em Métodos Numéricos em Egenharia, 2013.

Orientador: Marcio Augusto Villela Pinto. -- Co-orientadores: Leonardo Cavetti; Luciano Kiyoshi Araki.

1.Métodos interativos (Matemática). 2.Método multigrid geométrico. I. Universidade Federal do Paraná. II. Pinto, Marcio Augusto Villela. III. Cavetti, Leonardo. IV. Araki, Luciano Kivoshi. V. Título.

CDD: 515.353

TERMO DE APROVAÇÃO

RÉVERTON LUIS ANTUNES NEUNDORF

DESEMPENHO DE UM ALGORITMO *MULTIGRID* PARALELO APLICADO À EQUAÇÃO DE LAPLACE.

Dissertação aprovada como requisito parcial para a obtenção do título do Mestre em Ciências, área de concentração: Mecânica Computacional, pelo Programa de Pós-Graduação em Métodos Numéricos em Engenharia, Setor de Ciências Exatas e Tecnologia, Universidade Federal do Paraná, pela seguinte banca examinadora:

Prof. Dr. Marcio Augusto Villela Pinto
Universidade Federal do Paraná (UFPR)

Prof. Dr. Rudimar Luiz Nos
Universidade Tecnológica Federal do Paraná (UTFPR)

Prof. Dr. Ricardo Carvalho de Almeida
Universidade Federal do Paraná (UFPR)

Curitiba, 28 de março de 2013.

AGRADECIMENTOS

Agradeço imensamente ao meu orientador, Prof. Dr. Marcio Augusto Villela Pinto, por ter aceitado me orientar neste trabalho, pelo apoio, didática, por todo o conhecimento recebido e, acima de tudo, pela infinita paciência que apresentou durante o desenvolvimento deste trabalho.

Agradeço ao meu co-orientador, Prof. Dr. Luciano Kiyoshi Araki, pela dedicação, conhecimento recebido e pelas valiosas (e precisas) observações sobre o texto desta dissertação, que, apesar do trabalho imposto, ajudaram a evitar um desastre.

Agradeço imensamente ao meu co-orientador, Dr. Leonardo Calvetti, por ter me mostrado o caminho da pós-graduação, pela paciência, pela confiança, por todo o conhecimento recebido, pelo trabalho que estamos realizando no Instituto Tecnológico SIMEPAR, pelo ambiente de trabalho positivo, enfim por tudo.

Aos professores, funcionários e colegas do CESEC, pela amizade e disposição em ajudar.

Aos meus pais, pela compreensão e apoio incondicional.

Agradeço o apoio dos Kamers, que considero minha família.

A todos aqueles que de alguma forma contribuíram para a realização deste trabalho.

Agradeço especialmente a, excelentíssima, Neyde Abdalla (“Dona Dina”), por seu esforço extraordinário em promover a educação e a cultura. Sem sua influência, tudo teria sido diferente (para pior).

RESUMO

Entre os métodos mais eficientes empregados na solução de sistemas de equações estão os métodos *multigrid*. Apesar de numericamente eficientes, a solução de sistemas de equações com um grande número de incógnitas pode resultar em elevado tempo de CPU, visto que normalmente apresentam tempo de processamento proporcional ao número destas. Uma possível solução para este problema é a paralelização destes métodos através do particionamento do domínio em subdomínios menores (menos incógnitas). Neste trabalho foi resolvido numericamente o problema de condução de calor bidimensional linear governado pela equação de Laplace com condições de contorno de Dirichlet. Utilizou-se o Método das Diferenças Finitas (MDF), com esquema de aproximação de segunda ordem (CDS) para discretização do modelo matemático. Os suavizadores (*solvers*) utilizados foram os métodos Gauss-Seidel *red-black* e Jacobi ponderado. Para a obtenção da solução, foi empregado o método *multigrid* geométrico, com esquema de correção CS, restrição por ponderação completa, prolongação utilizando interpolação bilinear e número máximo de níveis para os diversos casos estudados. A paralelização do *multigrid* foi realizada aplicando-se uma metodologia, proposta neste trabalho, a cada uma de suas componentes algorítmicas: *solver*, processo de restrição, processo de prolongação e cálculo do resíduo. Os resultados podem ser considerados positivos, pois verificou-se que, além do tempo de CPU ter sido reduzido significativamente, este diminuiu à medida que o número de processadores utilizados aumentou.

Palavras-chave: *multigrid*, métodos iterativos, diferenças finitas, paralelização, particionamento do domínio.

ABSTRACT

Multigrid methods are among the most effective techniques for solving systems of equations. Despite numerically efficient, the solution of large systems of equations can result in high CPU time, since, usually, the time needed for processing is proportional to the number of unknowns. A possible solution to this problem is the parallelization of these methods through the partitioning of the domain on subdomains (fewer unknowns). In the present work a two-dimensional linear heat transfer conduction problem, given by a Laplace-type equation with Dirichlet boundary conditions, was numerically solved. The numerical model was obtained by the use of the Finite Difference Method (FDM) with a second-order approximation scheme (CDS). The solvers associated with the Geometric Multigrid Method were the Gauss-Seidel red-black and weighted Jacobi. In order to achieve the numerical solution, the Geometric Multigrid Method was used, associated with correction scheme (CS), full-weighting scheme for restriction, bilinear interpolation for prolongation and the maximum number of levels for each one of the studied cases. The Multigrid parallelization was performed by applying a methodology, proposed in this work, at each one of its algorithmic components: solver, restriction process, prolongation process and residual calculating. The results are considered positive, since it has been found that the CPU time was significantly reduced, and in proportion to the increase of the number of processors.

Keywords*: multigrid, iterative methods, finite difference, parallelization, domain partitioning.*

LISTA DE ALGORITMOS

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CDS	*Central Difference Scheme*
CFD	*Computational Fluid Dynamic*
CMP	*Chip-level Multiprocessors*
CPU	*Central Processing Unit*
CS	*Correction Scheme*
DD	*Domain Decomposition*
DDS	*Downstream Difference Scheme*
FAS	*Full Approximation Scheme*
MG	*Multigrid*
MPE	*Minimum Perimeter Equi-partition*
SMP	*Symmetric Multi-Processor*
SPMD	*Single Program Multiple Data*
TME	*Textbook Multigrid Efficiency*
UDS	*Upstream Difference Scheme*

LISTA DE SÍMBOLOS

$a_{i,j}$	Coeficientes da matriz A
A	Matriz dos coeficientes
b	Vetor de termos independentes do sistema linear
C	Número que "ativa" a correção do número de nós por subdomínio
$cond(A)$	Número de condicionamento da matriz A
$\frac{d^n}{dx^n}$	Derivada de ordem *n* em relação à coordenada x
$\frac{\partial^2}{\partial x^2}$	Derivada parcial de segunda ordem em relação à coordenada x
$\frac{\partial^2}{\partial y^2}$	Derivada parcial de segunda ordem em relação à coordenada y
E	Eficiência da paralelização
e	Vetor contendo os valores do erro numérico
e_j	Componentes do vetor **e**
f	Vetor de termos independentes do sistema linear
f	Parcela do processamento que pode ser melhorada (com a paralelização)
f_t	Índice que determina o fim de cada subdomínio
h	Tamanho dos elementos da malha
h_x	Tamanho dos elementos da malha na direção x
h_y	Tamanho dos elementos da malha na direção y
i_t	Índice que determina o início de cada subdomínio
I_h^{2h}	Operador de Restrição
I_{2h}^h	Operador de Prolongação
k	Número de onda ou modo de Fourier
l	"Linha" em que um processador realiza soma
l_t	Número de vezes que t realizará soma
l_{1t}	Número máximo de vezes que $t-1$ é divisível por 2
l_{2t}	Número que indica o "vizinho" mais distante de t

lv	Nível do ciclo V
$\mathbf{L_2}$	Vetor de valores da norma euclidiana do resíduo
L	Número de níveis
$L_{\max}$	Número máximo de níveis
L_x	Comprimento do domínio bidimensional: direção x
L_y	Comprimento do domínio bidimensional: direção y
m	Aceleração (fator de melhoria) do processamento
n	Número de processadores
N	Número de incógnitas do problema
N_x	Número de incógnitas na direção x
N_y	Número de incógnitas na direção y
N_{sub}	Número de nós por subdomínio
r	Razão de engrossamento
$\mathbf{r}$	Vetor contendo os valores do resíduo
r'_t	Soma do resíduo do subdomínio t
$\mathbf{r}'$	Vetor contendo os termos r'_t
R	Matriz de iteração
R_J	Matriz de iteração (método Jacobi)
R_ω	Matriz de iteração (método Jacobi ponderado)
R_G	Matriz de iteração (método Gauss-Seidel)
R	Domínio real unidimensional
R^2	Domínio real bidimensional
S	Termo fonte
S_P	*Speed-up*
S_n	*Speed-up* obtido com n processadores
S_{Amdahl}	*Speed-up* previsto pela lei de Amdahl
t	Número que identifica o *thread*
t_c	*Thread* a partir do qual os subdomínios receberão um nó extra
T	Temperatura (variável dependente)
$\bar{T}$	Número de *threads*

T'	Tempo total de execução do programa
T_P	Tempo de execução empregado em processamento paralelo
T_P'	Fração do tempo de execução devido ao processamento paralelo
T_S	Tempo de execução empregado em processamento serial
T_S'	Fração do tempo de execução devido ao processamento serial
t_n	Tempo de CPU utilizando n processadores
$\mathbf{u}$	Vetor que representa a solução exata do sistema linear
$u(x)$	Função genérica
u^*	Termo ponderado do método de Jacobi ponderado
u_{xx}	Derivada segunda de $u(x)$ em relação a x
$\mathbf{v}$	Aproximação do vetor solução $\mathbf{u}$
$\mathbf{v}_k$	k -ésimo modo de Fourier
x, y	Variáveis do sistema cartesiano bidimensional
$w_{k,j}^h$	j -ésima componente do k -ésimo modo de

Fourier da malha h

Z	Conjunto dos números inteiros

Letras Gregas

ε	Tolerância
$\lambda(A)$	Autovalores de A
ν	Número de iterações internas
ν_1	Número de pré-suavizações
ν_2	Número de pós-suavizações
$\rho(A)$	Raio espectral de A
ω	Fator de amortecimento
Ω^h	Malha fina
Ω^{2h}	Malha com engrossamento padrão
Ω^H	Malha grossa genérica

Subscritos

i	Contador na direção coordenada x
j	Contador na direção coordenada y
E	Nó localizado à direita do nó P
N	Nó localizado ao norte do nó P
P	Nó genérico da malha
S	Nó localizado ao sul do nó P
W	Nó localizado à esquerda do P

Sobrescritos

$h, 2h, 3h, \cdots$	Indicadores do nível de malha
i	Iteração atual
lv	Nível do ciclo V

Símbolos

$\|\cdot\|_2$	Norma euclidiana ou norma-2
$\|\cdot\|_\infty$	Norma do máximo ou norma infinito
$\lfloor \ \rfloor$	Função *floor*

Sumário

1 Introdução

1.1 Generalidades em Dinâmica dos Fluidos Computacional

Modelos matemáticos que visam representar fenômenos físicos, em geral, não possuem soluções analíticas conhecidas. Buscam-se então soluções via métodos numéricos, transformando-se os modelos contínuos em modelos discretos.

Além dos métodos analíticos e numéricos, denominados métodos teóricos porque trabalham com modelos matemáticos (MALISKA, 2004), existe uma terceira estratégia que pode ser empregada na solução de problemas em engenharia: os métodos experimentais. Cada uma das estratégias possui vantagens e desvantagens, que dependem das características do problema estudado, mas devem ser consideradas complementares. A Tab. 1.1, extraída de Tannehill et al. (1997), traz uma comparação entre as três estratégias.

Os métodos numéricos consistem em resolver as equações diferenciais dos modelos matemáticos, que representam os fenômenos físicos, aproximando as derivadas destas por expressões algébricas que contenham a função incógnita. Este processo, de aproximação de derivadas por expressões algébricas, é denominado de discretização.

Existem várias formas de se obter as expressões algébricas que aproximam derivadas, das quais três métodos são citados abaixo:

- Elementos finitos (HUGHES, 2000; REDDYE e GARTLING, 1994);
- Diferenças finitas (FORTUNA, 2000; TANNEHILL et al., 1997);
- Volumes finitos (MALISKA, 2004; VERSTEEG e MALALASEKERA, 2007).

Neste trabalho será utilizado o método das diferenças finitas (MDF) (GOLUB e ORTEGA, 1992; TANNEHILL et al., 1997) que, em problemas bidimensionais para malhas estruturadas, consiste em particionar o domínio do problema $\{(x,y)\in R^2 : 0\le x\le L_x \text{ e } 0\le y\le L_y\}$ em subconjuntos através de um número de incógnitas (pontos ou nós), dados por $N = N_x N_y$, onde N_x e N_y são os números de

pontos nas direções coordenadas x e y, respectivamente (incluindo os contornos), e L_x e L_y determinam o comprimento do domínio de cálculo.

Considerando malhas estruturadas e uniformes tem-se os pontos:

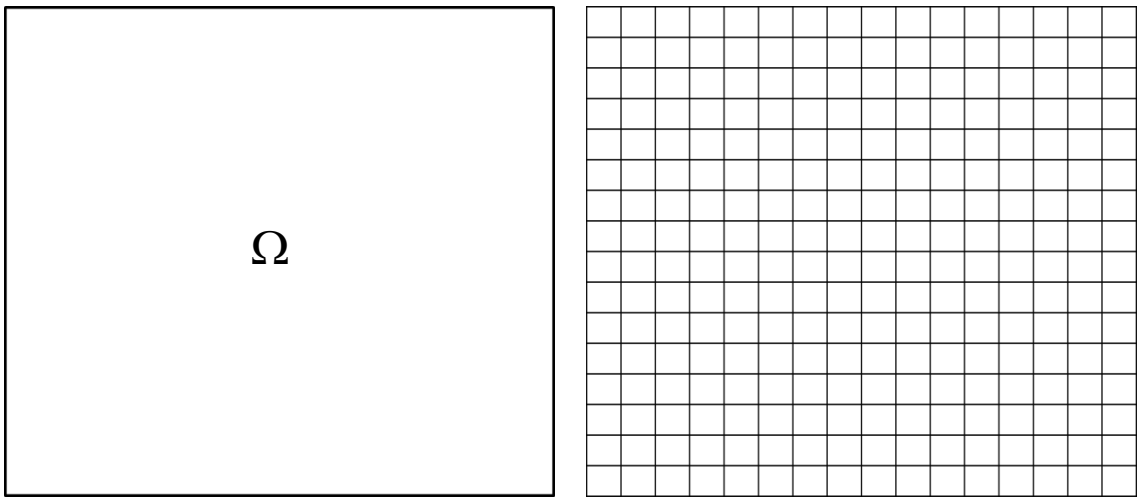
$$\left(x_i, y_j\right) = \left((i-1)h_x, (j-1)h_y\right), \text{ com } h_x = \frac{L_x}{N_x - 1} \text{ e } h_y = \frac{L_y}{N_y - 1}, \tag{1.1}$$

onde $i = 1, \cdots, N_x$, $j = 1, \cdots, N_y$, e h_x e h_y representam o tamanho de cada elemento (ou passo) nas direções coordenadas x e y, respectivamente. A este conjunto de pontos dados pela Eq. (1.1), dá-se o nome de malha e denota-se por Ω^h.

Tabela 1.1: Comparação entre os métodos de solução de problemas em engenharia (adaptada de Tannehill et al., 1997).

Método	Vantagens	Desvantagens
Experimental	- mais realístico	- são exigidos equipamentos (empregados na medição) - problemas de escala - dificuldades de medição - custo operacional
Analítico	- solução aplicável a qualquer ponto no domínio - representação matemática do fenômeno real	- restrita a geometrias e processos físicos simples - geralmente restrita a problemas lineares
Numérico	- não há restrição quanto à linearidade - geometrias e processos complicados - evolução temporal do processo	- erros numéricos - prescrições das condições de contorno apropriadas - custo computacional - erros de modelagem

Diferentemente do método analítico, que permite calcular os valores das variáveis dependentes em um número infinito de pontos (solução contínua), o método numérico fornece esses valores em um número discreto de pontos (pontos nodais ou nós) definido pela malha computacional (MESQUITA, 2000). A Fig. 1.1 (a) representa um domínio contínuo e a Fig. 1.1 (b) o mesmo domínio discreto.



a) Domínio contínuo b) Domínio discreto

Figura 1.1: Discretização do domínio.

Para localizar os nós do domínio discreto (malha obtida após o particionamento) é necessário algum tipo de ordenação. A Fig 1.2 mostra um exemplo de um tipo de ordenação conhecida como ordenação lexicográfica para $N_x = N_y = 9$.

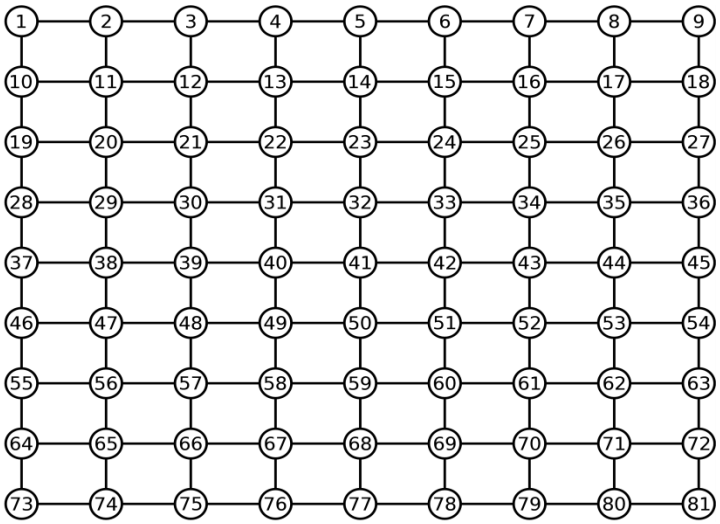


Figura 1.2: Ordenação lexicográfica.

Cada círculo na Fig 1.2 representa um nó na malha e é localizado por um índice (número) P.

O processo de discretização do domínio e da equação diferencial do modelo matemático gera um sistema de equações algébricas do tipo:

$$A\mathbf{u} = \mathbf{b}, \tag{1.2}$$

onde A é a matriz dos coeficientes, $\mathbf{b}$ é o vetor dos termos independentes e $\mathbf{u}$ o vetor de incógnitas. A estrutura da matriz A depende do método de discretização e pode, em alguns casos, ter elementos diferentes de zero apenas na diagonal principal (e algumas paralelas a esta).

Várias técnicas computacionais são estudadas com o objetivo de resolver o sistema representado pela Eq. (1.2) com o menor custo computacional (tempo de CPU) e solução mais próxima da solução exata. Métodos diretos, como a eliminação de Gauss, resolvem razoavelmente bem sistemas lineares de pequeno porte (da ordem de 10^2 elementos), mas na prática não são recomendados, visto que, a matriz A apresenta ordem elevada em problemas de interesse na engenharia, e o custo da inversão da mesma é alto (GOLUB e VAN LOAN, 1989). Para sistemas de grande porte, métodos iterativos são mais adequados (BURDEN e FAIRES, 2008), pois possuem custo computacional da ordem de $O\left(N^2/2\right)$, valor consideravelmente menor que o custo da ordem de $O\left(N^3\right)$ dos métodos diretos, onde N é o número de incógnitas.

Os métodos *multigrid* pertencem à família dos métodos iterativos utilizados para resolver com eficiência equações diferenciais parciais. Esses métodos têm como principal objetivo acelerar a convergência do esquema iterativo em que são aplicados (TANNEHILL et al., 1997).

Os métodos *multigrid* estão entre as técnicas mais eficientes utilizadas na solução de equações elípticas (BRIGGS et al., 2000; TROTTENBERG et al., 2001) porque o número de operações aritméticas que precisam ser realizadas para atingir o nível do erro de discretização é proporcional ao número de incógnitas do sistema de equações a ser resolvido (MCBRYAN et al., 1990). Esta relação entre o número de operações aritméticas e o número de incógnitas é uma das virtudes dos métodos *multigrid* e é conhecida como *textbook multigrid efficiency* (TME) (THOMAS et al., 2003).

O interesse em aplicar os métodos *multigrid* em problemas de larga escala, mantendo o custo computacional (tempo de processamento) aceitável, conduz à necessidade de paralelizar os métodos *multigrid*. Uma abordagem que satisfaz às condições exigidas é obtida através da decomposição do domínio em subdomínios menores (com menos incógnitas) e solução destes em paralelo.

A principal razão pela qual a decomposição do domínio pode apresentar desempenho insatisfatório é a distribuição de carga desbalanceada (CHAPMAN et al., 2008; TROTTENBERG et al. 2001): alguns processadores precisam realizar muito mais trabalho que outros (ociosos). Neste caso, os processadores que terminarem suas tarefas terão que esperar os processadores sobrecarregados terminarem seus processos para que possam iniciar novos procedimentos.

1.2 Motivação

A discretização de certos fenômenos, como o escoamento de um fluido em três dimensões, pode resultar em sistemas de equações com elevado número de incógnitas de forma que até mesmo os mais poderosos computadores seriais podem apresentar desempenho inadequado. Uma possível solução para estes casos, onde até mesmo o emprego do método *multigrid* é insuficiente, é o uso de computadores em paralelo (MCBRYAN et al., 1990).

O surgimento de processadores com vários núcleos representa um grande avanço sobre o problema associado ao tempo de comunicação entre processadores presente em sistemas com memória distribuída (detalhes na seção 2.6). O *multigrid* pode se beneficiar desse avanço e isto motiva um estudo a respeito da implementação de uma versão em paralelo do método *multigrid* capaz de fazer uso de processadores com vários núcleos.

1.3 Revisão bibliográfica

Vários autores (CHAN e SAAD, 1986; CHAN e SCHREIBER, 1985; HEMPEL e SCHÜLLER, 1988; HERBIN et al., 1988; THOLE, 1985; MCBRYAN, 1990) estudaram a paralelização dos métodos *multigrid*.

Contudo, segundo Trottenberg et al. (2001), o *multigrid* padrão não é completamente paralelizável, pois as malhas são processadas sequencialmente e o grau de paralelismo é diferente para cada malha considerada (menor para malhas mais grossas).

Visto que o custo de comunicação entre os processadores não é desprezível, várias versões do método *multigrid* paralelo foram criadas. Uma classe de algoritmos *multigrid* que processam vários (ou todos) níveis simultaneamente, ou seja, as malhas não são processadas sequencialmente (conhecida como versão aditiva), foi estudada por autores como Bramble et al. (1990), Gannon e Van Rosendale (1986), Greenbaum (1986) e Xu (1992).

Autores como Frederickson e McBryan (1987, 1988, 1990) desenvolveram o algoritmo Método *Multigrid* Paralelo Superconvergente (*Parallel Superconvergent Multigrid Method*) que trata de paralelismo massivo (emprego de uma grande quantidade de processadores em paralelo) e consiste em resolver várias malhas grossas simultaneamente com objetivo de acelerar a convergência do método *multigrid*.

Na tentativa de evitar o custo de comunicação crescente nas malhas mais grossas, Hempel e Schüller (1988) sugeriram uma técnica de aglomeração: nas malhas mais grossas pode-se aglomerar vários pontos de malha em um único processador para evitar comunicação em excesso.

A Decomposição do Domínio (DD) foi estudada por autores como Quarteroni e Valli (1999) e Smith et al. (1996). Esta técnica consiste em decompor o problema em vários outros problemas com domínios próprios, com ou sem a sobreposição dos mesmos.

Com a popularização dos sistemas com memória compartilhada surge o interesse em estudar o desempenho do método *multigrid* neste tipo de máquina.

Baker et al. (2010) observam que o método *multigrid* algébrico (AMG) apresenta bom desempenho em sistemas com memória distribuída, no entanto, enfrenta desafios que podem prejudicar sua eficiência quando executados em arquiteturas com memória compartilhada: sistema com vários processadores, processadores com vários núcleos, acesso não uniforme à memória, compartilhamento de memória *cache*, que é um tipo de memória presente dentro dos processadores com o propósito de diminuir a diferença com que os processadores executam instruções e a velocidade com que o dados são acessados na memória principal. Em seu trabalho resolvem um problema representado por Laplace 3D discretizado com diferenças finitas em um cubo com 1000^3 pontos de

malha. Foca no estudo da utilização das APIs (*Application Programming Interface*) OpenMP e MPI, além de uma aplicação híbrida destas. Apresentam melhorias no uso da API OpenMP em sistemas com acesso não uniforme à memória. O sistema testado é o Hera: um *cluster* (mais detalhes sobre *clusters* na seção 2.6) com 864 nós sendo estes equipados com 4 AMD Quadcore 8356 rodando a uma velocidade de 2.3 GHz, ou seja, um sistema com memória distribuída cujos nós são compostos por sistemas com memória compartilhada.

Courtecuisse e Allard (2009) estudam o desempenho do método Gauss-Seidel utilizando várias estratégias com objetivo de expor o paralelismo (processos que podem ser executados em paralelo). As estratégias testadas consistem em dividir a matriz do sistema de equações em linhas, colunas e blocos de forma que cada um destes possa ser processado em paralelo. Para os testes empregam sistemas com memória compartilhada com 2, 4, 8 e 16 núcleos, além disso, testam o desempenho das técnicas discutidas para o caso de um GPU (NVIDIA GeForce GTX 280) com 240 núcleos. Demonstram que a implementação do Gauss-Seidel utilizando as técnicas propostas é capaz de utilizar todos os núcleos disponíveis.

Wallin et al. (2006) resolvem um problema representado por Poisson 3D utilizando os métodos Gauss-Seidel *red-black*, *multigrid* e, proposta no trabalho, *parallel temporally blocked* Gauss-Seidel (TBGS). O sistema utilizado é o UltraSPARC IV com memória compartilhada e um total de 8 núcleos. Para os testes são empregadas malhas com 129^3, 257^3 e 513^3. O número de suavizações utilizados no método *multigrid* variam de 1 a 7 mas iguais na pré e pós-suavização. Demonstram que a versão do método Gauss-Seidel proposta no trabalho (TBGS) conduz a melhores resultados que a versão *red-black* convencional.

Em Kowarschik et al. (2000) é resolvido um problema representado por Poisson 2D utilizando o método Gauss-Seidel *red-black*. O trabalho apresenta possíveis modificações do método Gauss-Seidel *red-black* de forma que este faça uso intensivo da memória *cache*. Discute a técnica *loop fusion* na qual faz-se uma única varredura do domínio atualizando nós *black* e *red* alternativamente, ou seja, reduzindo pela metade (em relação ao *red-black* convencional) o número de vezes que as informações das malhas são escritas no *cache*. Discute a técnica *loop blocking* (ou *loop tiling*), na qual o domínio é dividido em blocos com tamanho apropriado à quantidade de memória *cache* disponível nos processadores; nesta técnica os blocos devem ser reutilizados o máximo possível para se beneficiar da velocidade superior da memória *cache*, no entanto é

prejudicada pela dependência local (nós vizinhos) dos dados. Introduz a técnica *Windshield Wiper*, que consiste em utilizar um pequeno (apropriado ao nível da memória *cache* que se deseja otimizar) bloco bidimensional que varre o domínio fazendo uso intensivo da informação armazenada na memória *cache*, além disso, beneficia-se dos diferentes níveis (L1, L2, L3, registradores, etc) que compõem esta. Demonstra que a otimização do uso da memória *cache* conduz a bons resultados.

McAdams et al. (2010) resolvem um problema modelado por Poisson 3D com condições de Dirichlet e Neumann. Utilizam o método *multigrid* como precondicionador da aplicação de um método de Krylov. O tipo de ciclo utilizado é o V e o suavizador escolhido é o Jacobi ponderado com fator de ponderação $\omega = 2/3$. Os testes são realizados em um servidor Intel Xeon X7350 e são obtidos bons resultados.

Em Thürey et al. (2009) são discutidas as tendências dos sistemas com vários núcleos e técnicas de otimização da memória *cache* tais como *loop fusion*, *loop blocking* entre outras. São discutidas a implementação do método *multigrid* em paralelo.

1.4 Objetivos

Neste trabalho serão implementados algoritmos capazes de utilizar todos os núcleos de um processador.

Os objetivos gerais deste trabalho são:

- paralelizar o método *multigrid* através do particionamento do domínio;
- acelerar o método *multigrid* através de sua paralelização (reduzir o tempo de CPU).

A decomposição de dados é uma forma bastante popular de paralelizar aplicações (CHAPMAN et al., 2008) e, no contexto do método *multigrid*, implica em particionar a malha que representa o domínio do problema. Uma metodologia bastante utilizada na decomposição da malha consiste em dividir o domínio em malhas (subdomínios) retangulares e entregar cada uma destas para um processador; nesta metodologia a diferença entre o número de nós por processador (desbalanceamento de carga) não pode se evitada, o que pode resultar em um desempenho insatisfatório da paralelização TROTTENBERG et al. 2001). Desta forma os objetivos específicos podem ser apontados como:

- desenvolver uma metodologia de particionamento do domínio de baixo custo computacional e com ótimo balanceamento de carga (diferença mínima no número de nós entre os subdomínios);
- estudar a aplicação da metodologia de particionamento do domínio na paralelização das componentes algorítmicas do método *multigrid*: métodos iterativos Jacobi ponderado e Gauss-Seidel *red-black*, processos de restrição e prolongação e cálculo do resíduo;
- implementar um método *multigrid* utilizando componentes algorítmicas paralelizadas, com a metodologia proposta, e verificar a redução no tempo de CPU obtida em relação à sua versão serial.

1.5 Organização da dissertação

Esta dissertação está organizada em sete capítulos e dois apêndices, descritos a seguir: No capítulo 1 são apresentadas algumas generalidades em Dinâmica dos Fluidos Computacional (*Computational Fluid Dynamic*, CFD), como são geradas as malhas e o método *multigrid*, a motivação, a revisão bibliográfica e os objetivos desta pesquisa. O capítulo 2 traz a fundamentação teórica dos métodos iterativos e apresenta uma introdução a respeito de sistemas paralelos, estratégias de paralelização e as métricas envolvidas na avaliação de metodologias de paralelização. O capítulo 3 apresenta o desenvolvimento do método *multigrid*. O capítulo 4 é destinado à descrição detalhada dos modelos matemático e numérico estudados e os detalhes da implementação. O capítulo 5 apresenta a metodologia de paralelização desenvolvida neste trabalho e sua aplicação no método *multigrid*. No capítulo 6, são apresentados os resultados dos testes com a paralelização do *multigrid*. O capítulo 7 é destinado à conclusão da dissertação e, finalmente, as referências bibliográficas. No apêndice A são apresentados resultados para um sistema paralelo mais acessível (microcomputador equipado com processador de vários núcleos) e o apêndice B contém a versão completa das tabelas apresentadas no capítulo 6.

2 Fundamentação

2.1 Método das diferenças finitas

Métodos numéricos para resolver as equações diferenciais, advindas dos modelos matemáticos que representam fenômenos físicos, aproximam as derivadas destas por expressões algébricas que contenham a função incógnita (PATANKAR, 1980; FERZIGER e PERIC, 2002). No caso do método das diferenças finitas (MDF), estas expressões algébricas podem ser obtidas por meio de uma expansão das derivadas em série de Taylor.

A expansão em série de Taylor de uma função $u(x)$, ao redor de um ponto x_i, é dada por

$$u(x) = u(x_i) + \frac{(x-x_i)^1}{1!}\left.\frac{du}{dx}\right|_{x=x_i} + \frac{(x-x_i)^2}{2!}\left.\frac{d^2u}{dx^2}\right|_{x=x_i} + \frac{(x-x_i)^3}{3!}\left.\frac{d^3u}{dx^3}\right|_{x=x_i} + \cdots . \qquad (2.1)$$

A escolha do valor de x, na Eq. (2.1), resulta em diferentes formas de se aproximar derivadas

a) Diferença progressiva (*Downstream Difference Scheme*, DDS): quando se aproxima a derivada considerando um ponto à direita, ou seja, $x = x_i + h$:

$$u(x_i + h) = u(x_i) + \frac{h^1}{1!}\left.\frac{du}{dx}\right|_{x=x_i} + \frac{h^2}{2!}\left.\frac{d^2u}{dx^2}\right|_{x=x_i} + \frac{h^3}{3!}\left.\frac{d^3u}{dx^3}\right|_{x=x_i} + \cdots . \qquad (2.2)$$

Para h pequeno, pode-se truncar a série, desprezando-se os termos com derivadas de ordem igual ou superior a dois. Desta forma a derivada primeira pode ser aproximada por

$$\left.\frac{du}{dx}\right|_{x=x_i} \approx \frac{u(x_i + h) - u(x_i)}{h} . \qquad (2.3)$$

A ordem de acurácia das aproximações está relacionada ao menor expoente de h nos termos desprezados na série de Taylor. Desta forma, a ordem de acurácia da aproximação dada pela Eq. (2.3) é 1.

b) Diferença regressiva (*Upstream Difference Scheme*, UDS): quando se aproxima a derivada considerando um ponto à esquerda, ou seja, $x = x_i - h$:

$$u(x_i - h) = u(x_i) - \frac{h^1}{1!}\frac{du}{dx}\bigg|_{x=x_i} + \frac{h^2}{2!}\frac{d^2u}{dx^2}\bigg|_{x=x_i} - \frac{h^3}{3!}\frac{d^3u}{dx^3}\bigg|_{x=x_i} + \cdots . \tag{2.4}$$

Para h pequeno, pode-se truncar a série, desprezando-se os termos com derivadas de ordem igual ou superior a dois. Desta forma a derivada primeira pode ser aproximada por

$$\frac{du}{dx}\bigg|_{x=x_i} \approx \frac{u(x_i) - u(x_i - h)}{h}, \tag{2.5}$$

com ordem de acurácia igual a 1.

c) Diferença central (*Central Difference Scheme*, CDS): quando se aproxima a derivada considerando um ponto à esquerda e outro à direita. Subtraindo a Eq. (2.4) da Eq. (2.2), isolando o termo com a derivada primeira e truncando a série, tem-se

$$\frac{du}{dx}\bigg|_{x=x_i} \approx \frac{u(x_i + h) - u(x_i - h)}{2h}, \tag{2.6}$$

com ordem de acurácia igual a 2.

Nota-se que a derivada primeira pode ser definida como

$$\frac{du}{dx}\bigg|_{x=x_i} = \lim_{h\to 0}\frac{u(x_i) - u(x_i - h)}{h} = \lim_{h\to 0}\frac{u(x_i + h) - u(x_i - h)}{2h} = \lim_{h\to 0}\frac{u(x_i + h) - u(x_i)}{h}.$$

Para o caso da derivada segunda, uma aproximação utilizando CDS pode ser obtida

1) Somando-se as Eqs. (2.2) e (2.4) tem-se:

$$u\left(x_i+h\right)+u\left(x_i-h\right)=2u\left(x_i\right)+2\frac{h^2}{2!}\frac{d^2u}{dx^2}\bigg|_{x=x_i}+2\frac{h^4}{4!}\frac{d^4u}{dx^4}\bigg|_{x=x_i}+\cdots.$$

2) Isolando o termo com a derivada segunda:

$$\frac{d^2u}{dx^2}\bigg|_{x=x_i}=\frac{u\left(x_i+h\right)-2u\left(x_i\right)+u\left(x_i-h\right)}{h^2}-2\frac{h^2}{4!}\frac{d^4u}{dx^4}\bigg|_{x=x_i}-\cdots.$$

3) Para h pequeno, pode-se truncar a série e desprezar os termos com derivadas de ordem igual ou superior a quatro. Tem-se então

$$\frac{d^2u}{dx^2}\bigg|_{x=x_i}\approx\frac{u\left(x_i+h\right)-2u\left(x_i\right)+u\left(x_i-h\right)}{h^2},\tag{2.7}$$

com ordem de acurácia igual a 2.

Para o caso de $u\left(x,y\right)$, a Eq. (2.7) pode ser estendida como:

$$\frac{\partial^2u}{\partial x^2}\bigg|_{(x,y)=(x_i,y_j)}\approx\frac{u\left(x_i+h_x,y_j\right)-2u\left(x_i,y_j\right)+u\left(x_i-h_x,y_j\right)}{h_x^2},\tag{2.8}$$

$$\frac{\partial^2u}{\partial y^2}\bigg|_{(x,y)=(x_i,y_j)}\approx\frac{u\left(x_i,y_j+h_y\right)-2u\left(x_i,y_j\right)+u\left(x_i,y_j-h_y\right)}{h_y^2},\tag{2.9}$$

onde $h_x=L_x/\left(N_x-1\right)$, $h_y=L_y/\left(N_y-1\right)$, com N_x e N_y denotando a quantidade de pontos nas direções x e y, respectivamente, $i=2,3,...,N_x-1$ e $j=2,3,...,N_y-1$.

A relação entre a coordenada lexicográfica P e $\left(i,j\right)$ é dada por

$$P=i+\left(j-1\right)N_x.\tag{2.10}$$

Na Fig. 2.1 (b) é apresentada a nomenclatura convencionada dos nós (i, j), $(i-1, j)$, $(i+1, j)$, $(i, j-1)$ e $(i, j+1)$, mostrados na Fig. 2.1 (a). Na Fig. 2.1 (c) é apresentada a posição dos nós na ordem lexicográfica calculado com a Eq. (2.10).

Considerando a Eq. (2.10) e a convenção apresentada na Fig. 2.1, define-se o seguinte esquema:

$$\begin{aligned} u(x_i, y_j) &= u(i, j) = u_P; \\ u(x_i + h_x, y_j) &= u(i+1, j) = u_E = u_{P+1}; \\ u(x_i - h_x, y_j) &= u(i-1, j) = u_W = u_{P-1}; \\ u(x_i, y_j + h_y) &= u(i, j+1) = u_S = u_{P+N_x}; \\ u(x_i, y_j - h_y) &= u(i, j-1) = u_N = u_{P-N_x}. \end{aligned} \tag{2.11}$$

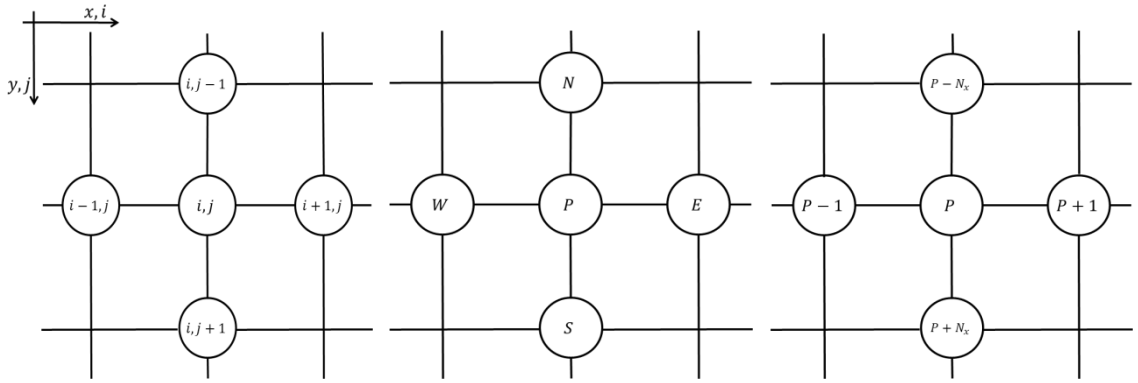


Figura 2.1: Nomenclatura dos vizinhos dos nós da malha bidimensional uniforme.

2.2 Método iterativo

Métodos iterativos são técnicas que aproximam a solução de um sistema de equações através de um processo de repetição conhecido como iteração.

Supondo um sistema de equações algébricas da forma

$$A\mathbf{u} = \mathbf{b}, \tag{2.12}$$

onde A é uma matriz $N \times N$, $\mathbf{u}$ e $\mathbf{b} \in R^N$, sendo $\mathbf{u}$ o vetor que contém as incógnitas e $\mathbf{b}$ o vetor dos termos independentes.

A matriz A pode ser dividida na forma:

$$A = M - N', \tag{2.13}$$

com M não-singular, ou seja, para a solução da Eq. (2.12), tem-se o seguinte método, que é chamado de método iterativo básico:

$$M\mathbf{u}^{(i+1)} = N'\mathbf{u}^{(i)} + \mathbf{b}, \tag{2.14}$$

onde o sobrescrito de $\mathbf{u}$ representa os níveis iterativos.

Isolando-se o vetor $\mathbf{u}^{(i+1)}$, na Eq. (2.14), obtém-se

$$\mathbf{u}^{(i+1)} = R\mathbf{u}^{(i)} + M^{-1}\mathbf{b}, \tag{2.15}$$

onde $R = M^{-1}N'$, $N' = M - A$, e a matriz R é chamada de matriz iteração do método.

Existe uma pequena, mas importante modificação (amortecimento) que pode ser feita na Eq. (2.15)

$$\mathbf{u}^{(i+1)} = \omega u^* + (1-\omega)\mathbf{u}^{(i)} = \mathbf{u}^{(i)} + \omega\left(u^* - \mathbf{u}^{(i)}\right), \tag{2.16}$$

com u^* definido como

$$u^* = R\mathbf{u}^{(i)} + M^{-1}\mathbf{b}. \tag{2.17}$$

O significado e importância de ω serão discutidos na seção 2.5.1.

2.2.1 Método de Jacobi e Jacobi ponderado

Um método iterativo básico bastante conhecido é o Método de Jacobi (BRIGGS et al., 2000). Neste caso a matriz $A = M - N'$ é decomposta na forma

$$A = D - L - U, \tag{2.18}$$

onde D é a diagonal, $-L$ a parte estritamente inferior e $-U$ a parte estritamente superior de A. Das Eqs. (2.13), (2.15) e (2.18) tem-se que $M = D$, $N' = L + U$ e

$$R = M^{-1}N' = D^{-1}\left(L + U\right) = R_J\,, \tag{2.19}$$

onde R_J é conhecida como matriz de iteração do método de Jacobi.

Da equação (2.15), pode-se notar que o Método de Jacobi é dado, na forma matricial, por

$$\mathbf{u}^{(i+1)} = R_J\mathbf{u}^{(i)} + D^{-1}\mathbf{b}\,. \tag{2.20}$$

Utilizando a versão amortecida dos métodos iterativos básicos, Eqs. (2.16) e (2.17), obtém-se o método conhecido como Método de Jacobi ponderado, na forma matricial

$$\mathbf{u}^{(i+1)} = R_\omega\mathbf{u}^{(i)} + \omega D^{-1}\mathbf{b}\,, \tag{2.21}$$

cuja matriz iterativa é dada por

$$R_\omega = \left(1-\omega\right)I + \omega R_J\,, \tag{2.22}$$

onde R_J é a matriz iterativa do método de Jacobi original.

2.2.2 Método de Gauss-Seidel

Como no caso do método de Jacobi, a matriz A é decomposta como apresentado na Eq. (2.18). No entanto, M é obtida de A fazendo-se $a_{ij} = 0$ se $i > j$. Desta forma

$$\left(D - L\right)\mathbf{u} = U\mathbf{u} + \mathbf{b}\,,$$

e portanto

$$\mathbf{u}^{(i)} = R_G\mathbf{u}^{(i)} + \left(D - L\right)^{-1}\mathbf{b}\,, \tag{2.23}$$

sendo $R_G = (D-L)^{-1}U$ a matriz iterativa do método Gauss-Seidel (BRIGGS et al., 2000). Diferentemente do método de Jacobi, que atualiza as componentes de $\mathbf{u}^{(i+1)}$ utilizando valores obtidos na iteração anterior $\mathbf{u}^{(i)}$, o método de Gauss-Seidel atualiza as componentes de $\mathbf{u}^{(i)}$ utilizando valores atualizados na própria iteração i.

Visto que as componentes de $\mathbf{u}^{(i)}$ são sobrescritas, ao invés de armazenadas em outro vetor, o método de Gauss-Seidel utiliza metade da quantidade de memória exigida pelo método de Jacobi.

2.3 Equação residual

Considere o sistema $A\mathbf{u} = \mathbf{b}$, com solução exata $\mathbf{u}$ e cuja solução aproximada $\mathbf{v}$ será obtida pelo método iterativo $M\mathbf{u}^{(i+1)} = N'\mathbf{u}^{(i)} + \mathbf{b}$.

O erro entre a solução aproximada $\mathbf{v}$ e a solução exata $\mathbf{u}$ é definido como

$$\mathbf{e} = \mathbf{u} - \mathbf{v}, \tag{2.24}$$

o erro na iteração i pode ser calculado de forma semelhante como

$$\mathbf{e}^{(i)} = \mathbf{u} - \mathbf{v}^{(i)}. \tag{2.25}$$

A magnitude do erro como definido pelas Eqs. (2.24) e (2.25) pode ser calculado pelas normas vetoriais comuns na literatura (BRIGGS et al., 2000), como a norma-2 (também conhecida como norma euclidiana)

$$\|\mathbf{e}\|_2 = \left(\sum_j^N e_j^2 \right)^{1/2}, \tag{2.26}$$

e também, a norma do máximo (também conhecida como norma-infinito)

$$\|\mathbf{e}\|_\infty = \max_{1 \le j \le N} |e_j|, \tag{2.27}$$

onde e_j é a j-ésima componente do vetor $\mathbf{e}$.

Da Eq. (2.24) é possível notar que o vetor erro $\mathbf{e}$ é tão inacessível quanto a própria solução exata $\mathbf{u}$, entretanto o vetor resíduo, que pode ser entendido como uma medida da proximidade entre $\mathbf{u}$ e $\mathbf{v}$, pode ser calculado com

$$\mathbf{r} = \mathbf{b} - A\mathbf{v}, \tag{2.28}$$

e o resíduo na iteração i pode ser calculado como

$$\mathbf{r}^{(i)} = \mathbf{b} - A\mathbf{v}^{(i)}. \tag{2.29}$$

O resíduo é uma medida de quanto a aproximação $\mathbf{v}$ falha em satisfazer o problema original $A\mathbf{u} = \mathbf{b}$. Para $\mathbf{e} \approx \mathbf{0}$ tem-se que $\mathbf{r} \approx \mathbf{0}$, porém $\mathbf{r} \approx \mathbf{0}$ não implica que $\mathbf{e} \approx \mathbf{0}$ (BRIGGS et al., 2000). Para matrizes bem condicionadas se $\mathbf{r} \approx \mathbf{0}$ então $\mathbf{e} \approx \mathbf{0}$ (maiores detalhes serão apresentados na seção 2.4).

A magnitude do resíduo pode ser calculado, como no caso do vetor erro, pelas normas vetoriais padrão, Eqs. (2.26) e (2.27).

A relação entre os vetores $\mathbf{u}$ e $\mathbf{v}$, de fundamental importância para o método *multigrid*, é dada pela equação residual

$$A\mathbf{e} = \mathbf{r}. \tag{2.30}$$

A equação residual permite que a solução aproximada $\mathbf{v}$ seja melhorada da seguinte forma

a) Resolver $A\mathbf{u} = \mathbf{b}$ (com um método iterativo) e obter $\mathbf{v}$;
b) Calcular o resíduo com $\mathbf{r} = \mathbf{b} - A\mathbf{v}$;
c) Resolver $A\mathbf{e} = \mathbf{r}$ (equação residual) e obter $\mathbf{e}$;
d) Usando a definição de erro, corrigir $\mathbf{u}$ com

$$\mathbf{u} = \mathbf{v} + \mathbf{e}. \tag{2.31}$$

Este procedimento é conhecido como “correção residual” ou “refinamento iterativo”.

2.4 Análise de convergência

Briggs et al. (2000) apresentam uma importante relação entre o erro e o resíduo

$$\frac{1}{cond\left(A\right)}\frac{\left\|\mathbf{r}^{(i)}\right\|}{\left\|\mathbf{b}\right\|}\leq\frac{\left\|\mathbf{e}^{(i)}\right\|}{\left\|\mathbf{v}^{(i)}\right\|}\leq cond\left(A\right)\frac{\left\|\mathbf{r}^{(i)}\right\|}{\left\|\mathbf{b}\right\|}, \tag{2.32}$$

onde $cond\left(A\right)=\left\|A\right\|\cdot\left\|A^{-1}\right\|$ é chamado número de condicionamento da matriz A. Por definição, $cond\left(A\right)\geq 1$ e, se $cond\left(A\right)\approx 1$, a matriz é considerada como bem condicionada (BURDEN e FAIRES, 2008).

Da Eq. (2.32), nota-se que $\mathbf{r}\approx\mathbf{0}$ implica em $\mathbf{e}\approx\mathbf{0}$ somente se $cond\left(A\right)$ for pequeno.

Pode-se demonstrar que o erro definido na Eq. (2.25) satisfaz

$$\mathbf{e}^{(i+1)}=R\mathbf{e}^{(i)}, \tag{2.33}$$

onde R é a matriz de iteração do método iterativo considerado.

Considerando o erro na iteração i, pela Eq. (2.33), tem-se

$$\mathbf{e}^{(i)}=R\mathbf{e}^{(i-1)}=R^{2}\mathbf{e}^{(i-2)}=R^{3}\mathbf{e}^{(i-3)}=\cdots=R^{i}\mathbf{e}^{(0)}, \tag{2.34}$$

onde R^{i} é a i-ésima potência de R.

Logo, das propriedades de normas (BURDEN e FAIRES, 2008), tem-se

$$\left\|\mathbf{e}^{(i)}\right\|\leq\left\|R^{i}\right\|\cdot\left\|\mathbf{e}^{(0)}\right\|=\left\|R\right\|^{i}\cdot\left\|\mathbf{e}^{(0)}\right\|, \tag{2.35}$$

para qualquer norma $\left\|\cdot\right\|$, tal que

$$\left\|R\right\|=\sup_{\mathbf{v}\neq\mathbf{0}}\frac{\left\|R\mathbf{v}\right\|}{\left\|\mathbf{v}\right\|}, \tag{2.36}$$

é a norma matricial induzida pela norma vetorial. Neste caso, $\|R\|$ é conhecido como "número de contração" do método iterativo $\mathbf{u}^{(i+1)} = R\mathbf{u}^{(i)} + M^{-1}\mathbf{b}$, dado pela Eq. (2.15).

Definição 2.1: O método iterativo $\mathbf{u}^{(i+1)} = R\mathbf{u}^{(i)} + M^{-1}\mathbf{b}$ é dito convergente se $\lim_{i\to\infty}\|R\|^i = 0$, com $R = M^{-1}N'$.

Definição 2.2: O raio espectral da matriz A é dado por $\rho(A) = \max|\lambda(A)|$, onde $\lambda(A)$ são os autovalores de A.

Teorema 2.1: O método iterativo $\mathbf{u}^{(i+1)} = R\mathbf{u}^{(i)} + M^{-1}\mathbf{b}$ é convergente se e somente se, $\rho(A) < 1$.

A prova do Teorema 2.1 pode ser encontrada em Burden e Faires (2008).

O raio espectral $\rho(A)$ é um importante indicador da razão de convergência do método iterativo, pois permite estimar o número de iterações necessárias para reduzir o erro por um fator de 10^{-d}, onde d é o número de casas decimais da precisão que se deseja atingir.

Suponha que na iteração i o erro deva ser reduzido por um fator de 10^{-d}, ou seja,

$$\frac{\left\|\mathbf{e}^{(i)}\right\|}{\left\|\mathbf{e}^{(0)}\right\|} \leq 10^{-d}. \tag{2.37}$$

Em Briggs et al. (2000) encontra-se que esta condição é aproximadamente satisfeita se

$$\left[\rho(A)\right]^{(i)} \leq 10^{-d}. \tag{2.38}$$

Isolando-se i na Eq. (2.38), nota-se que o número de iterações necessárias para reduzir o erro por um fator de 10^{-d} é

$$i \geq -\frac{d}{\log_{10}\left[\rho(A)\right]}. \tag{2.39}$$

A Eq. (2.39) implica em dois casos de interesse:

a) $\rho(A) \to 1 \Rightarrow \log_{10}\left[\rho(A)\right] \to 0 \Rightarrow i \to \infty$, ou seja, se $\rho(A) \to 1$ o método iterativo vai convergir muito lentamente $(i \to \infty)$.

b) $0 < \rho(A) \to 0 \Rightarrow \log_{10}\left[\rho(A)\right] \to -\infty \Rightarrow i \to 0$, ou seja, se $0 < \rho(A) \to 0$ o método iterativo vai convergir rapidamente $(i \to 0)$.

2.5 Propriedade de suavização dos métodos iterativos

Os métodos iterativos apresentam uma propriedade conhecida como suavização, que, como a equação residual, é fundamental no contexto do método *multigrid*.

Para entender esta propriedade é necessário fazer uma análise de erros de Fourier. Para tal é suficiente considerar o sistema $A\mathbf{u} = \mathbf{0}$, pois neste caso a solução exata é conhecida: $\mathbf{u} = \mathbf{0}$. Outra vantagem é que o erro resultante da estimativa inicial, vetor $\mathbf{v}$, utilizado para iniciar o esquema iterativo, é simplesmente $-\mathbf{v}$.

Para a análise de erros de Fourier serão consideradas estimativas iniciais 1D (a análise para o caso 2D é semelhante) conhecidas como modos de Fourier:

$$\mathbf{v}_k = sen\left(\frac{jk\pi}{N}\right),\ 0 \leq j \leq N,\ 1 \leq k \leq N-1, \tag{2.40}$$

onde $j = 1, \cdots, N$ denota a j-ésima componente do vetor $\mathbf{v}_k$ **(**k fixo).

A Fig. 2.2, extraída de Briggs et al. (2000), ilustra as estimativas iniciais dadas pelos vetores $\mathbf{v}_1$**,** $\mathbf{v}_3$ e $\mathbf{v}_6$. É possível notar que, para k pequeno, as ondas são longas e "suaves" e que para valores maiores de k, as ondas são curtas e "oscilátorias".

Formalmente o inteiro k é conhecido como número de ondas ou frequência e indica o número de "meios senos" do vetor $\mathbf{v}_k$**.**

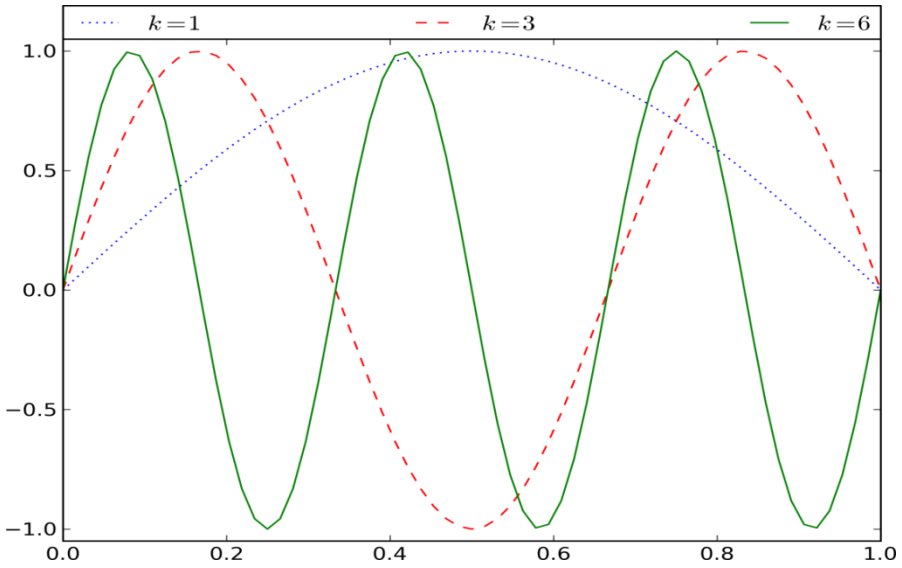


Figura 2.2: Modos de Fourier para $k=1,3,6$ (BRIGGS et al., 2000).

Uma definição mais rigorosa de “suave” e “oscilatório” é dada a seguir (BRIGGS et al., 2000).

Definição 2.3: Modos de Fourier de baixa frequência ou suaves estão localizados na faixa inferior do espectro, ou seja, $1 \le k < N/2$. Modos localizados na faixa superior do espectro, $N/2 \le k \le N-1$, são chamados de modos de alta frequência ou oscilatórios.

2.5.1 Análise de erros de Fourier para Poisson 1D

Nesta seção é apresentada uma análise de erros de Fourier. Por simplicidade, utilizou-se a Equação de Poisson 1D. A análise do caso 2D é análoga (TROTTENBERG et al., 2001).

Considere a Equação de Poisson unidimensional (1D), escrita como

$$\begin{cases} -u_{xx} = f, 0 < x < 1 \\ u(0) = u(1) = 0 \end{cases},$$

onde u_{xx} é a segunda derivada de u em relação à x (variável espacial).

A discretização da Equação de Poisson 1D pode ser realizada utilizando a Eq. (2.7) para aproximar a derivada segunda de u em relação à x, u_{xx}. Considerando o esquema das Eqs. (2.11) para o caso 1D a Equação de Poisson discretizada pode ser escrita como

$$-v_{i-1} + 2v_i - v_{i+1} = h^2 f_i,$$

onde $1 < i < N-1$, $v_0 = v_N = 0$ e $h = 1/N$. Como já observado, para a análise de erros de Fourier, é conveniente utilizar o caso homogêneo ($A\mathbf{u} = \mathbf{0}$). Assim

$$\begin{cases} -v_{i-1} + 2v_i - v_{i+1} = 0, 1 < i < N-1 \\ v_0 = v_N = 0 \end{cases}, \tag{2.41}$$

onde v_i é a aproximação de $u(x_i)$ (solução exata em x_i).

Aplicando o método de Jacobi ponderado, Eqs. (2.21) e (2.22), ao sistema da Eq. (2.41), cuja matriz A é dada por

$$A = \begin{bmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{bmatrix}_{(N-1)\times(N-1)},$$

e combinando as Eqs. (2.18) e (2.19), o que implica em $R_J = I - D^{-1}A$, tem-se que

$$R_\omega = (1-\omega)I + \omega R_J = I - \omega D^{-1}A = I - \frac{\omega}{2}A, \tag{2.42}$$

pois, neste caso, $D^{-1} = \frac{1}{2}I$.

Para avaliar a taxa de convergência do método de Jacobi ponderado é necessário calcular os autovalores de R_ω, visto que estes estão relacionados aos autovalores de A por

$$\lambda\left(R_\omega\right)=1-\frac{\omega}{2}\lambda\left(A\right). \tag{2.43}$$

Os autovalores de A do problema tratado nesta seção podem ser encontrados em Briggs et al. (2000) e são dados por

$$\lambda_k\left(A\right)=4sen^2\left(\frac{k\pi}{2N}\right), 1\le k\le N-1. \tag{2.44}$$

Os autovalores de R_ω podem ser calculados com as Eqs. (2.43) e (2.44)

$$\lambda_k\left(R_\omega\right)=1-2\omega sen^2\left(\frac{k\pi}{2N}\right), 1\le k\le N-1. \tag{2.45}$$

A Eq. (2.45) permite avaliar as taxas de convergência do método de Jacobi. O primeiro dado importante que a Eq. (2.45) apresenta, é que para $0<\omega\le 1$ o método iterativo é convergente, pois $\left|\lambda\left(R_\omega\right)\right|<1$ (exigência do Teorema 2.1).

Com a garantia de convergência do método (escolha de ω tal que $0<\omega\le 1$), pode-se estudar a taxa de convergência. Os modos de Fourier de baixa frequência, ou suaves, são de especial interesse por demonstrarem de forma contundente a ineficiência do método de Jacobi em reduzir este tipo erro.

Considerando $k=1$, modo suave segundo a Definição 2.3, a Eq. (2.45) é escrita como

$$\lambda_1=1-2\omega sen^2\left(\frac{\pi}{2N}\right)=1-2\omega sen^2\left(\frac{\pi h}{2}\right)\approx 1-\frac{\omega\pi^2h^2}{2}, \tag{2.46}$$

pois $0<x<1$ e $h=1/N$.

Como o valor de λ_1 está sempre próximo da unidade, o método iterativo vai convergir lentamente (que é o caso ‘a’ da Eq. (2.39)). Tentar refinar a malha aumentando o número de nós desta, ou seja, reduzir o valor de h, apenas piora a taxa de convergência do método.

A Eq. (2.46) também informa que nenhum valor de ω amortece de maneira satisfatória as componentes de erros suaves. A questão remanescente é qual valor de ω,

se existir, amortecerá de forma mais eficiente as componentes do erro oscilatório $\left(N/2 \le k \le N-1\right)$. Para obter tal valor pode-se, como a Fig. 2.3 sugere, impor a condição $\lambda_{N/2}\left(R_\omega\right) = -\lambda_N\left(R_\omega\right)$, o que resulta em

$$1-2\omega sen^2\left(\frac{\pi}{4}\right) = -1+2\omega sen^2\left(\frac{\pi}{2}\right) \Rightarrow 1-\omega = -1+2\omega \Rightarrow \omega = \frac{2}{3}. \tag{2.47}$$

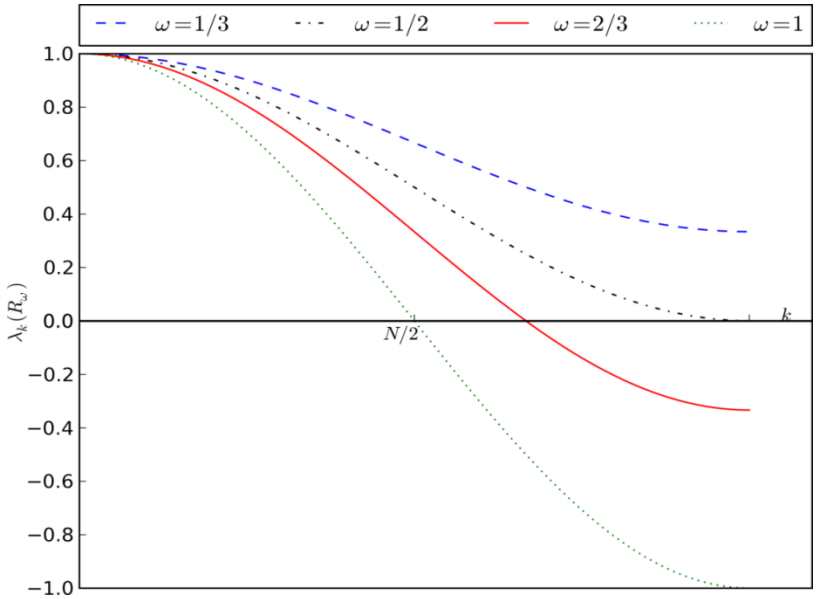


Figura 2.3: Autovalores da matriz de iteração R_ω.

Na Fig. 2.3 os autovalores, Eq. (2.45), são esboçados como se k fosse uma variável contínua em $0 \le k \le N$. De fato, k assume apenas valores inteiros no intervalo $1 \le k \le N-1$.

O valor $\omega = 2/3$ é conhecido como fator de ponderação ótimo (BRIGGS et al., 2000) pois, com auxílio da Fig. 2.3, pode-se deduzir que para todo $N/2 \le k \le N-1$ o fator de suavização será

$$\begin{cases} \lambda_k\left(R_\omega\right) \le 1-\frac{4}{3}sen^2\left(\frac{\pi}{4}\right) = \frac{1}{3}, N/2 \le k \\ \lambda_k\left(R_\omega\right) \ge 1-\frac{4}{3}sen^2\left(\frac{\pi}{2}\right) = -\frac{1}{3}, k \le N \end{cases}.$$

Logo,

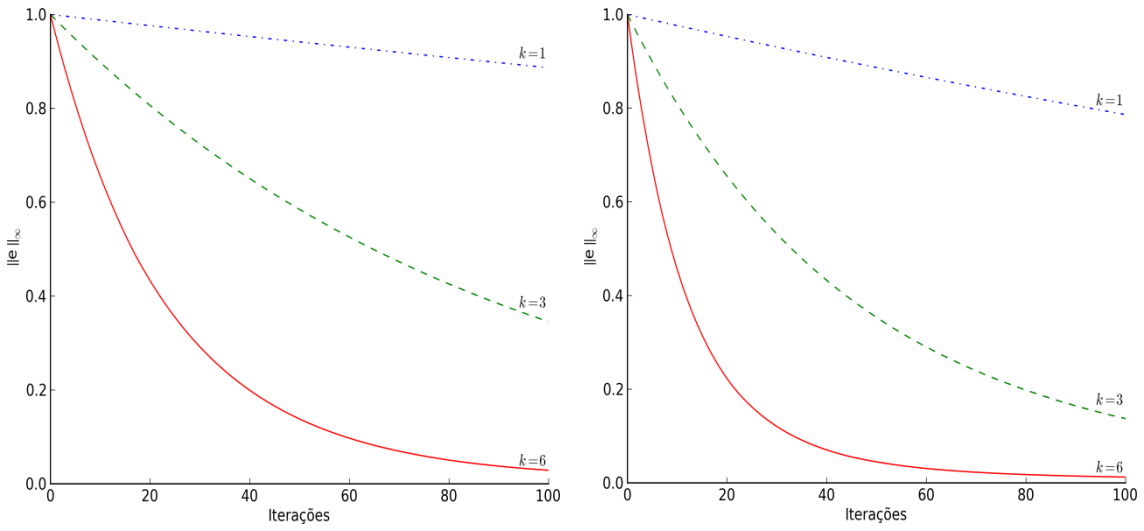
$$-\frac{1}{3} \leq \lambda_k \leq \frac{1}{3},$$

e portanto

$$|\lambda_k| < \frac{1}{3}, \text{ para todo } k \text{ no intervalo } N/2 < k < N. \tag{2.48}$$

Este fator de suavização, Eq. (2.48), implica que os erros associados aos autovalores no intervalo $N/2 < k < N$ serão reduzidos por um fator 3, no mínimo, a cada iteração do método de Jacobi ponderado. Além de convergir rapidamente (valor próximo de 0) o fator de suavização não depende do espaçamento da malha (como no caso dos modos de Fourier de baixa frequência) h, uma propriedade conveniente no contexto do método *multigrid*.

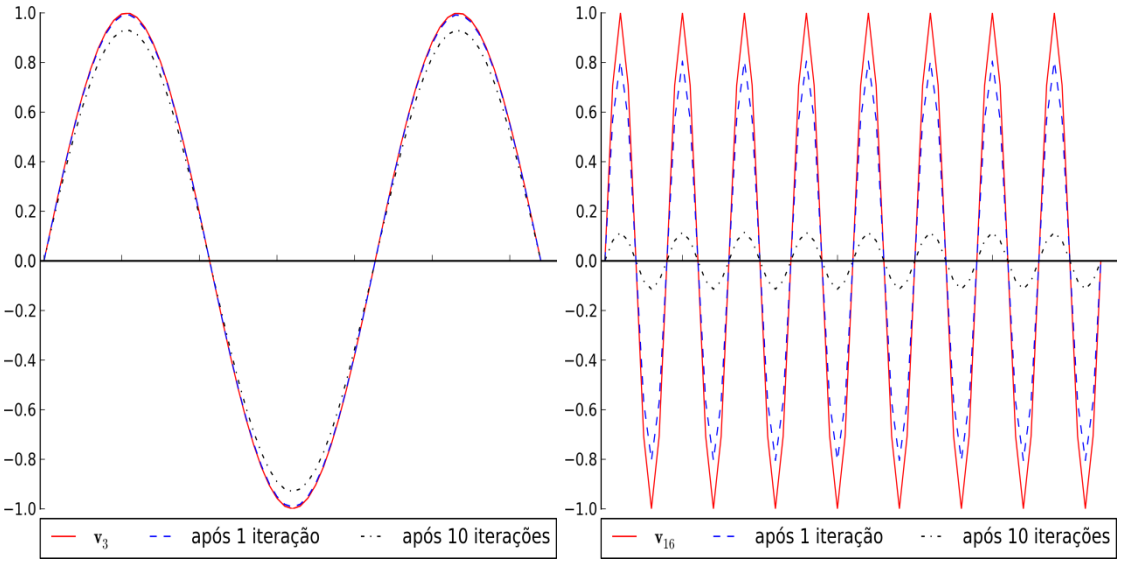
A Fig. 2.4, adaptada de Briggs et al. (2000), mostra a queda da norma infinito $\left(\|\cdot\|_\infty\right)$ do erro ao longo de 100 iterações do (a) método de Jacobi com $\omega = 2/3$ e (b) método de Gauss-Seidel. Esta figura mostra como o erro é reduzido muito mais rapidamente para valores crescentes de k.



a) Jacobi Ponderado b) Gauss-Seidel

Figura 2.4: Norma infinito do erro numérico *versus* iterações para os métodos de Jacobi ponderado com $\omega = 2/3$ e Gauss-Seidel .

A Fig. 2.5, adaptada de Briggs et al. (2000), mostra a queda da norma infinito $\left(\|\cdot\|_\infty\right)$ do erro após 1 e 10 iterações do método de Jacobi com $\omega = 2/3$. Em (a) foi utilizado como vetor estimativa inicial o vetor $\mathbf{v}_3$. Pode-se notar que o erro foi pouco reduzido após 10 iterações. Em (b) foi utilizado como estimativa inicial o vetor $\mathbf{v}_{16}$. Pode-se notar que a queda do erro é muito mais acentuada que no caso de $\mathbf{v}_3$.



a) Estimativa inicial $\mathbf{v}_3$ b) Estimativa inicial $\mathbf{v}_{16}$

Figura 2.5: Método de Jacobi ponderado com $\omega = 2/3$ aplicado nas estimativas iniciais $\mathbf{v}_3$ e $\mathbf{v}_{16}$ ao longo de 1 iteração e após 10 iterações (BRIGGS et al., 2000).

A Fig. 2.6, adaptada de Briggs et al. (2000), mostra a queda da norma infinito $\left(\|\cdot\|_\infty\right)$ do erro após 1 e 10 iterações do método de Jacobi com $\omega = 2/3$. Nesta figura foi utilizado como estimativa inicial o vetor $\left(\mathbf{v}_2 + \mathbf{v}_{16}\right)/2$, ou seja, um vetor com componentes de erro suave e componentes de erro mais oscilatório. Pode-se observar, neste caso, que as componentes do erro mais oscilatório (modos de alta frequência) foram rapidamente reduzidas e que o erro restante é, em sua maior parte, devido às componentes do erro suave (modos de baixa frequência).

Definição 2.4: A propriedade de eliminar os modos oscilatórios e deixar apenas os modos suaves é chamada de propriedade de suavização.

Uma consequência da propriedade de suavização, é que métodos iterativos que possuem esta propriedade (como o método de Gauss-Seidel), normalmente, apresentarão convergência rápida durante as primeiras iterações, indicando a presença de modos oscilatórios no erro. Após a eliminação dos modos oscilatórios, a convergência ficará muito mais lenta devido à predominância dos modos suaves como mostra a Fig. 2.7.

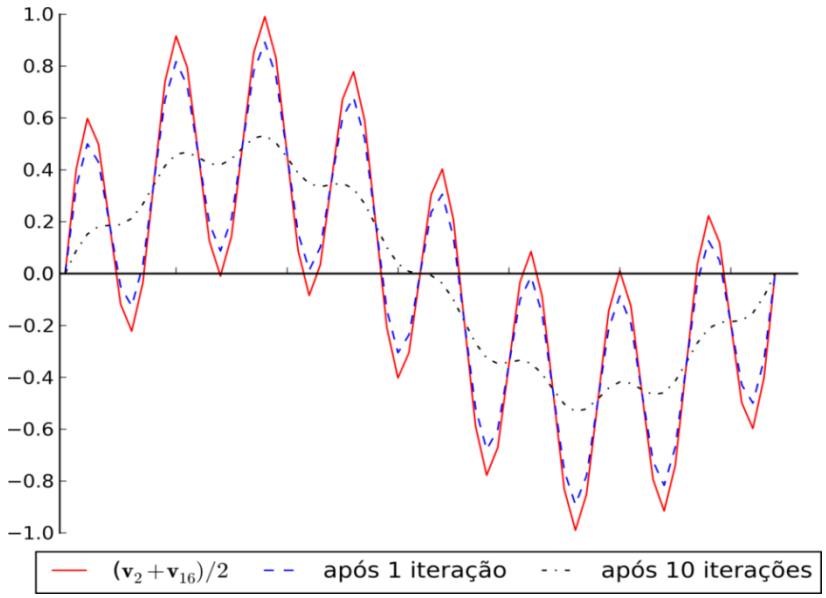
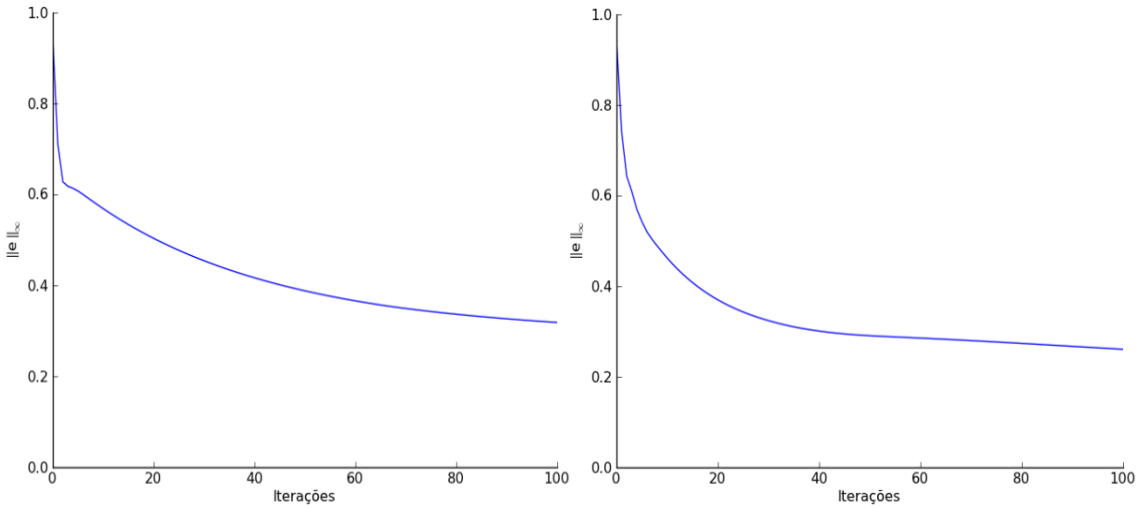


Figura 2.6: Método de Jacobi ponderado com $\omega = 2/3$ aplicado na estimativa inicial $(\mathbf{v}_2 + \mathbf{v}_{16})/2$ ao longo de 1 iteração e após 10 iterações (BRIGGS et al., 2000).



a) Jacobi Ponderado b) Gauss-Seidel

Figura 2.7: Métodos de Jacobi ponderado com $\omega = 2/3$ e Gauss-Seidel aplicados na estimativa inicial $(\mathbf{v}_1 + \mathbf{v}_6 + \mathbf{v}_{32})/3$ ao longo de 100 iterações.

Na Fig. 2.7, os métodos de (a) Jacobi ponderado com $\omega = 2/3$ e (b) Gauss-Seidel são aplicados na estimativa inicial $(\mathbf{v}_1 + \mathbf{v}_6 + \mathbf{v}_{32})/3$ ao longo de 100 iterações. Inicialmente a convergência é rápida e, após a redução das componentes do vetor $\mathbf{v}_{32}$, a convergência fica lenta devido à persistência das componentes dos vetores $\mathbf{v}_1$ e $\mathbf{v}_6$.

2.6 Introdução ao conceito de paralelização

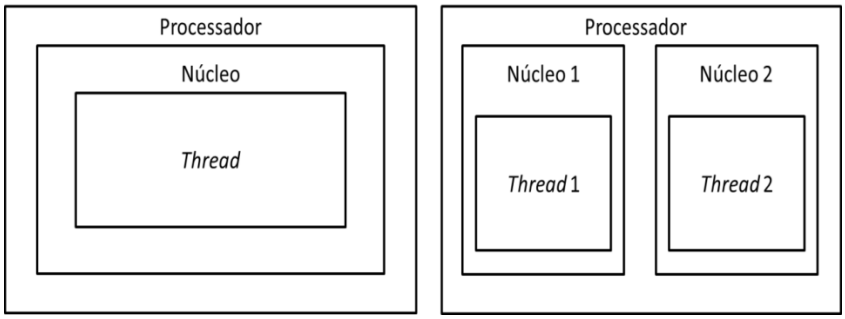
A paralelização pode ser entendida como o número de processos que podem ser realizados em paralelo, ou seja, simultaneamente. Em geral o objetivo da computação paralela é reduzir o tempo de processamento empregado na solução de problemas utilizando vários processadores.

No contexto do método *multigrid*, os principais processos que podem ser realizados em paralelo são (TROTTENBERG et al., 2001):

1) Os métodos iterativos (seções 5.6 e 5.7);
2) O cálculo do resíduo (seção 5.4);
3) O processo de restrição (seção 5.8);
4) O processo de prolongação (seção 5.9).

A terminologia utilizada nesta dissertação será:

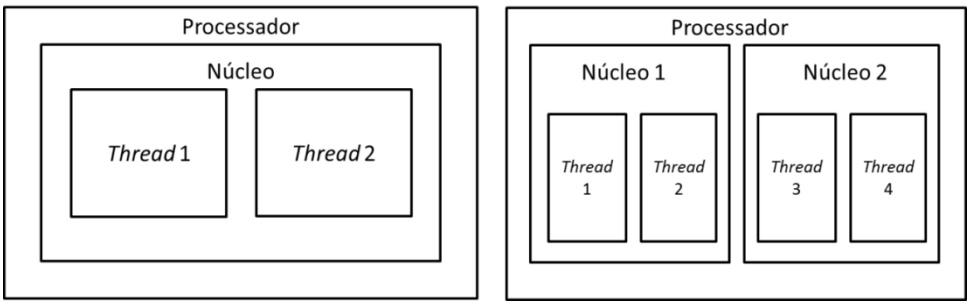
a) *Thread*: refere-se a uma unidade de processamento (cálculo, tomada de decisões, etc), que é também chamado de processador lógico ou *Kernel-Level Thread* (TANENBAUM, 2001; SILBERSCHATZ et al., 2005). É importante observar que cada *thread* é interpretado como um processador pelo sistema operacional. A Fig. 2.8 (a) ilustra a hierarquia principal de um processador;
b) Núcleo (*Core*): refere-se a uma fração do *chip* que executa um *thread*. Um *chip* pode conter mais de um núcleo, neste caso o processador é dito *multi-core* ou processador multinúcleo. Cada núcleo representa um processador físico. A Fig. 2.8 (a) apresenta o caso de um processador com apenas um núcleo e (b) ilustra o caso de um processador com dois núcleos;
c) Processador: refere-se ao *chip* que contém o(s) núcleo(s).



a) Processador com um núcleo b) Processador com dois núcleos

Figura 2.8: Hieraquia de um processador.

Observações: Algumas tecnologias, como a tecnologia *Hyper-Threading*, da companhia Intel, permitem que um núcleo execute dois *threads* simultaneamente. Neste caso, o sistema operacional interpreta um núcleo como dois processadores. Os detalhes sobre o desempenho desta tecnologia podem ser consultados em Tian et al. (2002). A Fig. 2.9 ilustra processadores utilizando uma tecnologia que permite executar dois *threads* por núcleo.



a) Processador com um núcleo b) Processador com dois núcleos

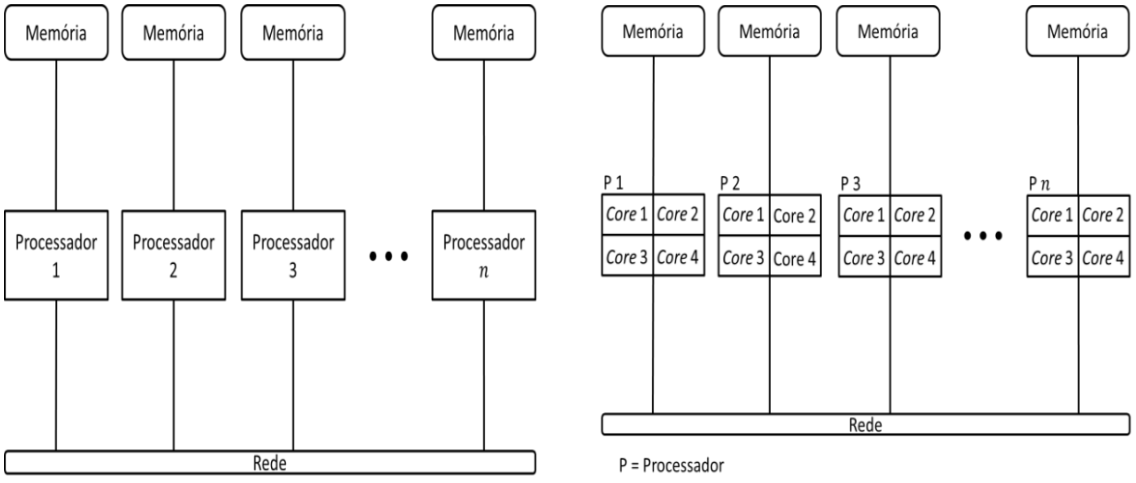
Figura 2.9: Ilustração de processadores executando mais de um *thread* por núcleo.

Em relação à memória, os sistemas (máquinas) podem ser divididos em dois grupos principais (HERMANNS, 2002):

1) Sistemas com memória distribuída: cada processador tem sua própria memória privada. Os processadores se comunicam via troca de mensagens realizada através de uma rede. O principal exemplo deste tipo de sistema é o *cluster*: vários computadores (chamados nós) conectados por uma rede de alta velocidade. Neste sistema, o processador de cada nó tem acesso apenas à memória disponível no nó. Como a comunicação entre os processadores é realizada através da rede, o custo desta, geralmente, não é negligenciável. A Fig.

2.10 (a) ilustra um *cluster*. É importante observar que o processador de cada nó pode ter mais de um núcleo, como demonstrado na Fig. 2.10 (b), e cada núcleo pode ter mais de um *thread* ;

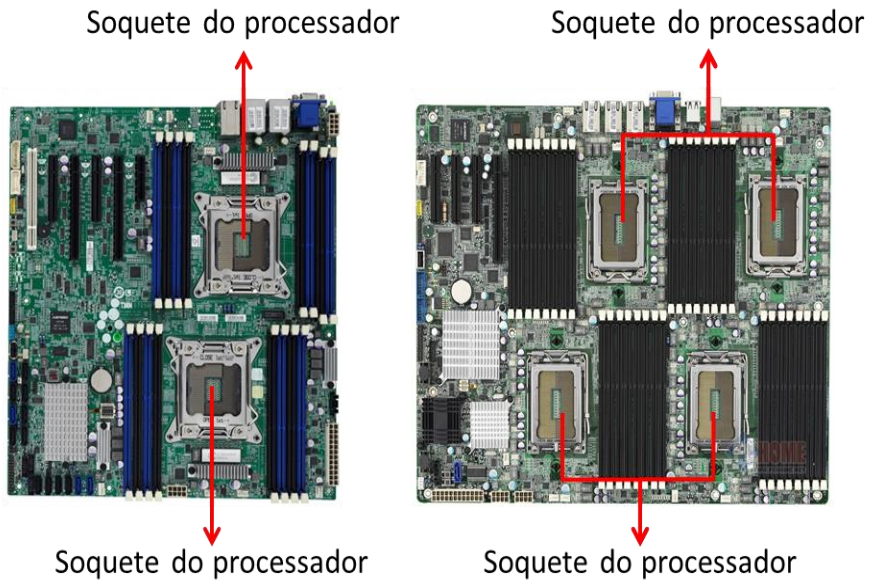
2) Sistemas com memória compartilhada: Neste caso os processadores compartilham a memória de todo o sistema através de um único barramento (linhas de comunicação que interligam os dispositivos de um computador). Como cada processador pode ler e escrever em qualquer endereço da memória, este tipo de sistema exige sincronização nos casos de leitura-escrita em um mesmo endereço na memória, um problema conhecido como *race condition* (CHAPMAN et al., 2008). Mais detalhes sobre este problema na seção 5.7. Exemplos importantes deste tipo de sistema são:
    a. Multiprocessadores Simétricos (*Symmetric Multi-Processor*, SMP): Servidores são bons exemplos deste tipo de sistema onde vários processadores (*chip*) são instalados em uma mesma "placa mãe". A Fig. 2.11 (a) mostra uma placa com dois processadores e (b) quatro processadores. Cada processador instalado pode ter apenas um núcleo (Fig. 2.12 (a)) ou mais de um núcleo (Fig. 2.12 (b)) e cada núcleo mais de um *thread*;
    b. Processadores multinúcleo (*chip-level multiprocessors*, CMP): são processadores (*chip*) com vários núcleos (processadores físicos). Cada núcleo tem acesso à mesma memória disponível ao processador.



a) Processadores com um núcleo b) Processadores com quatro núcleos

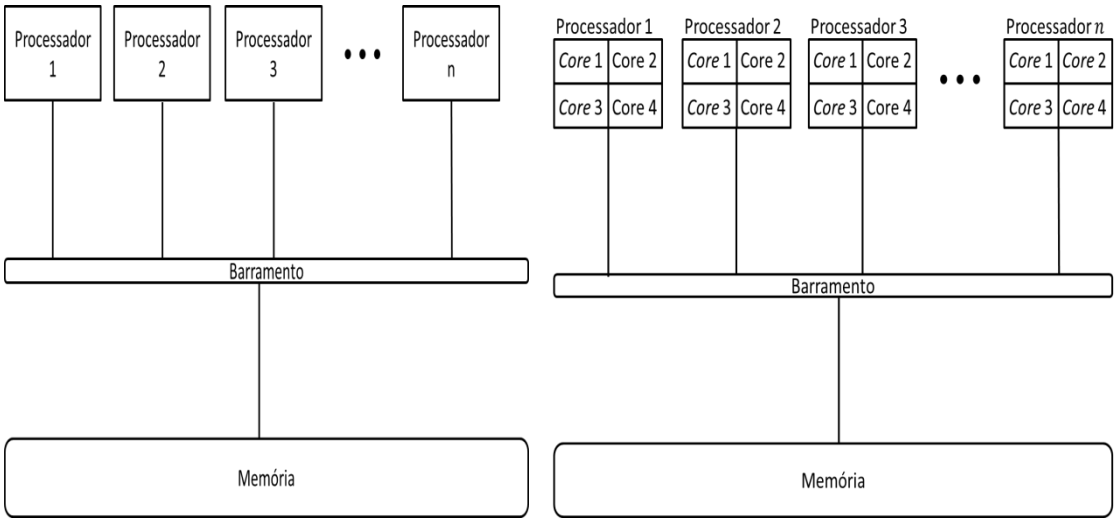
Figura 2.10: Representação de um *cluster*.

O *cluster* da Fig. 2.10 (b) representa um híbrido de um sistema com memória distribuída (processadores P1, P2, ..., Pn possuem memória privada) e memória compartilhada (os núcleos de P1, P2, ..., Pn compartilham a memória privada de cada processador).



a) Placa com dois processadores b) Placa com quatro processadores

Figura 2.11: Placas mãe com mais de um soquete (onde são instalados os processadores).



a) Processadores com um núcleo b) Processadores com quatro núcleos

Figura 2.12: Representação de sistemas com memória compartilhada

Os resultados que serão apresentados no capítulo 6 desta dissertação foram obtidos utilizando sistemas com memória compartilhada, mais precisamente o modelo da Fig. 2.12 (b).

O modelo de paralelização empregado nesta dissertação é conhecido como SPMD (*Single Program Multiple Data*) (CHAPMAN et al., 2008), no qual cada *thread* aplica o mesmo programa, simultaneamente, em uma parte específica (para cada *thread*) da malha, chamada de subdomínio.

Neste trabalho, a expressão *decomposição de domínio* refere-se ao particionamento do domínio computacional em subdomínios. Cada subdomínio representa uma parte do problema global e é atribuído a um *thread*. É crucial particionar o domínio de forma que os subdomínios representem uma distribuição balanceada de carga entre os processadores.

2.7 Lei de Amdahl e métricas de desempenho

O objetivo principal da paralelização de uma aplicação é reduzir o tempo de CPU, ou seja, melhorar o desempenho desta. Uma das principais formas de avaliar o ganho de desempenho obtido com a paralelização é através do *speed-up*.

Definição 2.5: O *speed-up* é uma medida empregada para avaliar o aumento de velocidade obtido durante a execução de um programa utilizando um algoritmo "A" em relação a sua execução utilizando um algoritmo "B" (GALANTE, 2006):

$$S_P = \frac{t_{CPU}\left(\text{Algoritmo A}\right)}{t_{CPU}\left(\text{Algoritmo B}\right)}. \tag{2.49a}$$

No contexto do paralelismo, o *speed-up* é empregado para avaliar o aumento de velocidade, ou "aceleração", obtido durante a execução de um programa utilizando um algoritmo "A", em apenas um processador, em relação a sua execução em n processadores (CHAPMAN et al., 2008; TROTTENBERG et al., 2001). O *speed-up*, S_n devido a n processadores é dado pela fórmula

$$S_n = \frac{t_1(\text{Algoritmo A})}{t_n(\text{Algoritmo A})}, \tag{2.49b}$$

onde t_1 é o tempo de CPU utilizando apenas 1 processador (tempo de execução serial do programa) e t_n é o tempo de CPU utilizando n processadores (tempo de execução em paralelo do programa).

Quanto maior o valor de S_n, mais rápido se encontra a versão em paralelo do programa. Porém, devido à execução sequencial de regiões do programa, o valor máximo que S_n pode atingir é limitado pelo tempo que o algoritmo "A" utiliza realizando o processamento serial. Este fenômeno é conhecido como "lei de Amdahl" (AMDAHL, 1967).

Seguem abaixo as premissas da lei de Amdahl:

1) O tempo de execução T', de um programa utilizando um algoritmo "A" é a soma do tempo T_S que o programa emprega realizando processamento serial, com o tempo T_P que o programa utiliza realizando processamento em paralelo, ou seja, $T' = T_S + T_P$;
2) Apenas o tempo de processamento realizado em paralelo, T_P, é afetado pela adição de processadores ao sistema $\left(T_P, \frac{T_P}{2}, \frac{T_P}{3}, \cdots\right)$. O tempo de processamento serial não pode ser reduzido pela adição de processadores;
3) Para n processadores, o tempo de CPU é dado por:

$$t_n = T_S + \frac{T_P}{n}; \tag{2.50}$$

4) Teoricamente, com a adição de "infinitos" processadores, o tempo de execução do programa tende para o tempo gasto no processamento serial do mesmo.

Na Fig. 2.13 estão ilustradas tais premissas.

A fórmula matemática que descreve a lei de Amdahl pode ser obtida substituindo a Eq. (2.50) na Eq. (2.49b), ou seja, o *speed-up* devido a n processadores pode ser calculado com:

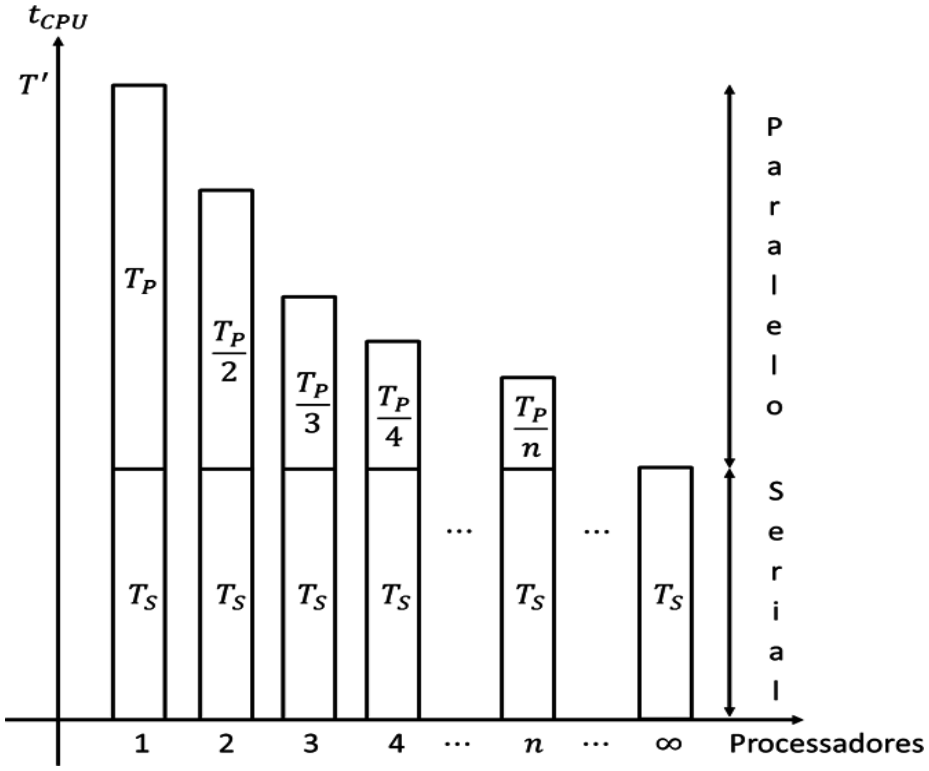


Figura 2.13: Premissas da lei de Amdahl.

$$S_n = \frac{t_1}{t_n} = \frac{T_S + T_P}{T_S + \frac{T_P}{n}} = \frac{1}{T_S^{'} + \frac{T_P^{'}}{n}} = \frac{1}{\left(1 - T_P^{'}\right) + \frac{T_P^{'}}{n}},$$

onde $T_S^{'} = \frac{T_S}{T'}$ é a fração do tempo T' de execução (do programa) devido ao processamento sequencial, e $T_P^{'} = \frac{T_P}{T'}$ é a fração do tempo empregado no processamento paralelo. Por simplicidade pode-se considerar $T' = 1$. Assim

$$S_n = \frac{1}{\left(1 - T_P\right) + \frac{T_P}{n}}. \tag{2.51}$$

Pode-se notar que se $n \to \infty$, então $S_n \to 1/T_S$, ou seja, o valor do *speed-up* $\left(S_n\right)$ é limitado pela parcela serial do processamento.

Na literatura (SUN e CHEN, 2009) é comum encontrar a lei de Amdahl em sua forma mais genérica

$$S_{\text{Amdahl}} = \frac{1}{(1-f)+\frac{f}{m}}, \tag{2.52}$$

onde f representa a parcela do processamento que pode ser melhorada por um fator m e o restante, $(1-f)$, a parcela que não pode ser melhorada. No contexto do paralelismo, f representa a parcela do trabalho que pode ser paralelizada e m, o número de processadores utilizados.

A Fig. 2.14 ilustra o *speed-up* teórico máximo previsto, pela Eq. (2.52), para programas que têm, hipoteticamente, frações paralelas f de 50%, 80%, 90% e 95%.

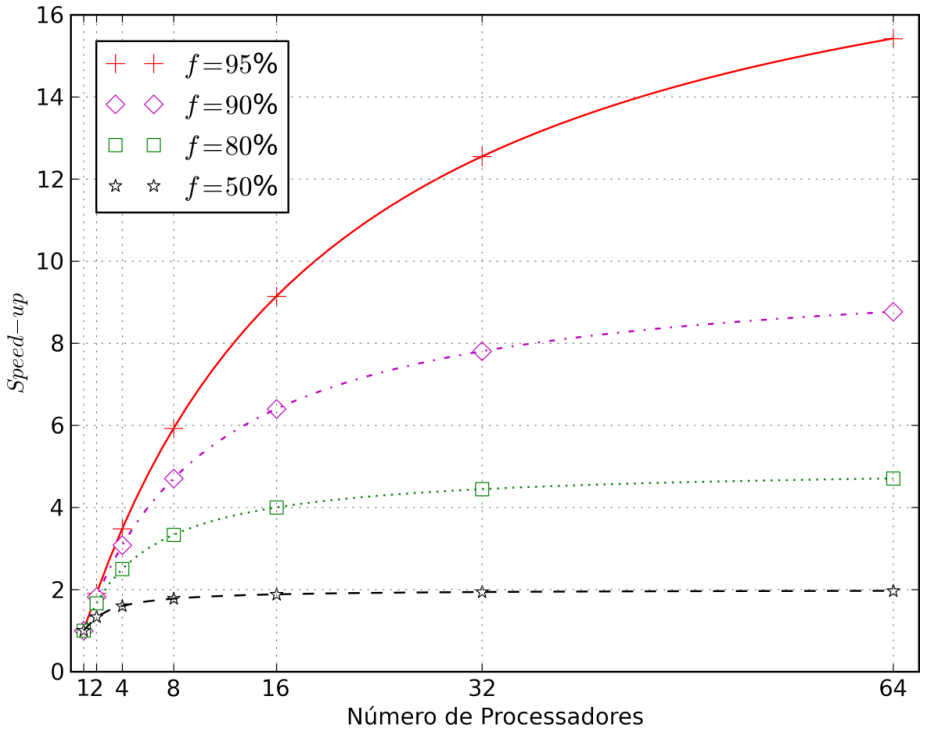


Figura 2.14: Limite superior de *speed-up* para programas com frações paralelas entre 50% a 95%.

O *speed-up* teórico calculado pela Eq. (2.52) representa o valor ideal desta métrica, pois considera que f é perfeitamente paralelizável, ou seja, ignora custos como a comunicação e sincronização dos processadores, tempo de acesso à memória, entre outros.

É importante observar que o *speed-up* previsto pela Eq. (2.52) é válido apenas para o caso em que a "carga de trabalho" é fixa (GUSTAFSON, 1988; SUN e CHEN, 2009).

Nesta dissertação, pode-se interpretar “carga de trabalho” como o número de incógnitas do sistema de equações (Eq. (2.12)).

A Eq. (2.52) é importante no contexto do paralelismo pois governa a escalabilidade do sistema (considerando uma carga de trabalho fixa). A escalabilidade de um sistema paralelo pode ser interpretada como uma medida de sua capacidade de aumentar o fator *speed-up* em proporção ao número de elementos de processamento. Desta forma, a escalabilidade reflete a capacidade de empregar eficientemente novos elementos de processamento introduzidos no sistema (GRAMA et al., 2003).

Existem outras leis de escalabilidade, como a lei de Gustafson (GUSTAFSON, 1988) e a lei de Sun e Ni (SUN e NI, 1993). Esta última é a generalização da lei de Amdahl e Gustafson. No contexto destas leis, a carga de trabalho não é fixa (a carga de trabalho aumenta com o número de processadores), portanto a escalabilidade é diferente da prevista pela lei de Amdahl.

Os problemas estudados nesta dissertação são governados pela lei de Amdahl, pois se tratam de problemas cuja carga de trabalho é fixa, ou seja, os mesmos sistemas de equações (Eq. (2.12)) são resolvidos com diferentes números de processadores.

Outra métrica utilizada na avaliação do desempenho da paralelização é conhecida como eficiência e é definida como

$$E = \frac{S_n}{n}, \tag{2.53}$$

onde S_n é o *speed-up* calculado com a Eq. (2.49b) e n o número de processadores.

O valor da eficiência, calculada pela Eq. (2.53), encontra-se entre 0 e 1, ou entre 0% e 100% quando a Eq. (2.53) é multiplicada por 100. Esta métrica pode ser interpretada como uma medida do tempo em que os processadores são efetivamente empregados no processamento paralelo (GRAMA et al., 2003).

3 O método *multigrid*

3.1 Relação entre os modos de Fourier e o engrossamento das malhas

No capítulo anterior estudou-se a propriedade de suavização dos métodos iterativos. Observou-se que a aplicação de um método iterativo (que possui a propriedade de suavização) reduz os modos oscilatórios do erro mantendo os modos suaves praticamente inalterados.

Na Fig. 3.1, adaptada de Briggs et al. (2000), observa-se o mesmo vetor de erro sobre uma malha fina Ω^h e sobre a malha imediatamente mais grossa Ω^{2h} com $k=4$.

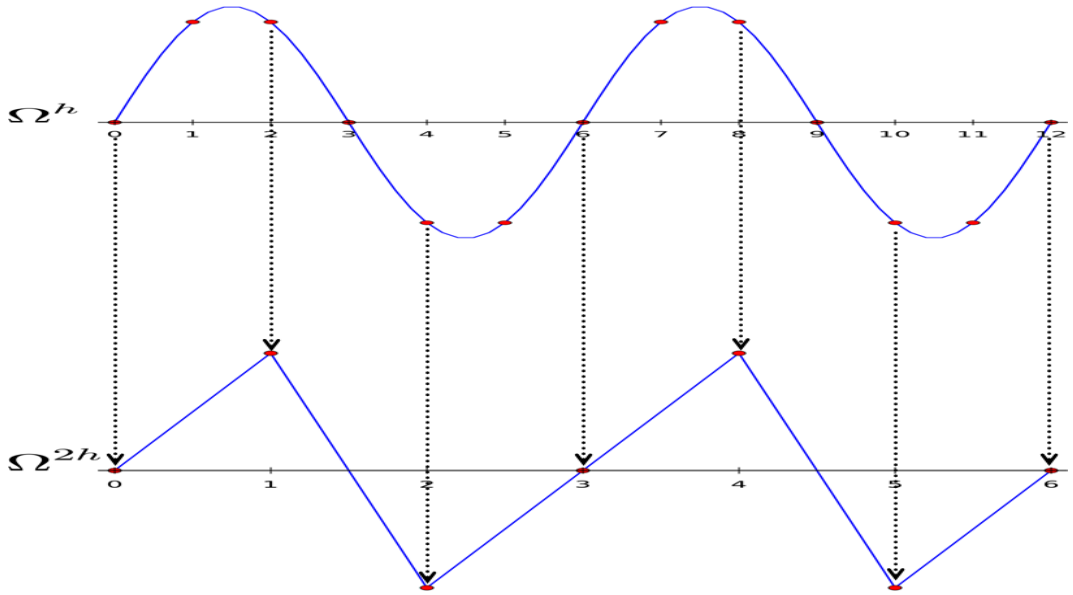


Figura 3.1: Número de ondas $k=4$ sobre uma malha fina Ω^h com $N=12$ e sobre a malha imediatamente mais grossa Ω^{2h} com $N=6$ (BRIGGS et al, 2000).

Na parte superior da Fig. 3.1 tem-se um número de ondas $k=4$ sobre uma malha fina Ω^h com $N=12$ elementos o que, segundo a definição 2.3, representa um modo de Fourier de baixa frequência (suave). Na parte inferior da Fig. 3.1 tem-se um número de ondas $k=4$ sobre a malha imediatamente mais grossa Ω^{2h} com $N=6$ elementos o que, segundo a definição 2.3, representa um modo de Fourier de alta frequência (oscilatório).

Na Fig. 3.1 pode-se notar que o número de ondas k do vetor permaneceu o mesmo após sua projeção da malha fina Ω^h para a malha grossa Ω^{2h}. A generalização deste

fenômeno pode ser obtida avaliando-se o vetor $\mathbf{v}_k$, $1 \le k \le N-1$, nos pontos pares de uma malha fina Ω^h qualquer.

Considere as componentes $w^h_{k,2j}$, do vetor $\mathbf{v}_k$, avaliadas nos pontos pares $2j$, da malha Ω^h. As componentes dos modos suaves, ou seja, $1 \le k < N/2$, são dadas por

$$w^h_{k,2j} = sen\left(\frac{2jk\pi}{N}\right) = sen\left(\frac{jk\pi}{N/2}\right) = w^{2h}_{k,j}, 1 \le k < N/2. \tag{3.1}$$

A Eq. (3.1) informa que o k-ésimo modo na malha fina Ω^h torna-se o k-ésimo modo na malha grossa Ω^{2h}, ou seja, o número de ondas k não se altera durante a mudança de malha. Uma importante consequência é que, para $1 \le k < N/2$, a malha grossa Ω^{2h} terá o mesmo número de modos que Ω^h e portanto, visto que o número de pontos N de Ω^{2h} é menor, os modos k tornam-se mais oscilatórios em Ω^{2h}. É neste ponto que o método *multigrid* explora a propriedade de suavização dos métodos iterativos: quando o processo de relaxação começa a ficar lento, indicando a predominância dos modos suaves, o método transfere o problema para uma malha mais grossa de forma que os modos suaves tornam-se oscilatórios e os métodos iterativos voltam a ficar eficientes (com a vantagem de operar sobre um sistema de equações com menos incógnitas).

É preciso destacar que a análise anterior não é válida para o caso em que $k \ge N/2$. Para o caso em que $k \ge N/2$ o modo em Ω^h torna-se nulo em Ω^{2h}. No caso em que $N/2 < k < N-1$, o k-ésimo modo oscilatório de Ω^h torna-se o $(N-k)$-ésimo modo em Ω^{2h}, um fenômeno conhecido como *aliasing* (BRIGGS et al., 2000).

3.2 O princípio do *multigrid*

O método *multigrid* consiste, basicamente, em explorar a propriedade de suavização dos métodos iterativos.

1) Aplicação do método iterativo em uma dada malha até que as componentes do erro tornem-se suaves. Neste passo a malha será chamada de malha "fina" e será

denotada por Ω^h, ou seja, uma malha com elementos de tamanho h, dados pela Eq. (1.1) com $h = h_x = h_y$.

2) Transferência do "problema" para uma malha mais grossa (com menos pontos que a original). Neste passo a malha será chamada de malha "grossa" e será denotada por Ω^H, ou seja, uma malha com elementos de tamanho H, com $H > h$.
3) Repetição dos passos 1) e 2) até que a malha mais grossa (ou conveniente) seja atingida.

O número de níveis $1 \le L \le L_{\max}$ que podem ser visitados durante o processo de engrossamento depende do problema considerado e da razão de engrossamento utilizada. A malha mais grossa possível é a malha com 3×3 nós. A quantidade de níveis até atingir essa malha, chamada número máximo de níveis, será aqui denotado por $L_{\max}$. Não é obrigatório engrossar as malhas até o nível máximo, mas Pinto e Marchi (2007) e Oliveira (2010) recomendam utilizar todos os níveis.

Para o caso bidimensional, considerando malhas uniformes (elementos da malha com mesmo tamanho), a razão de engrossamento é definida como $r = H/h$, onde h representa o tamanho dos elementos da malha fina Ω^h e H o tamanho dos elementos da malha grossa Ω^H.

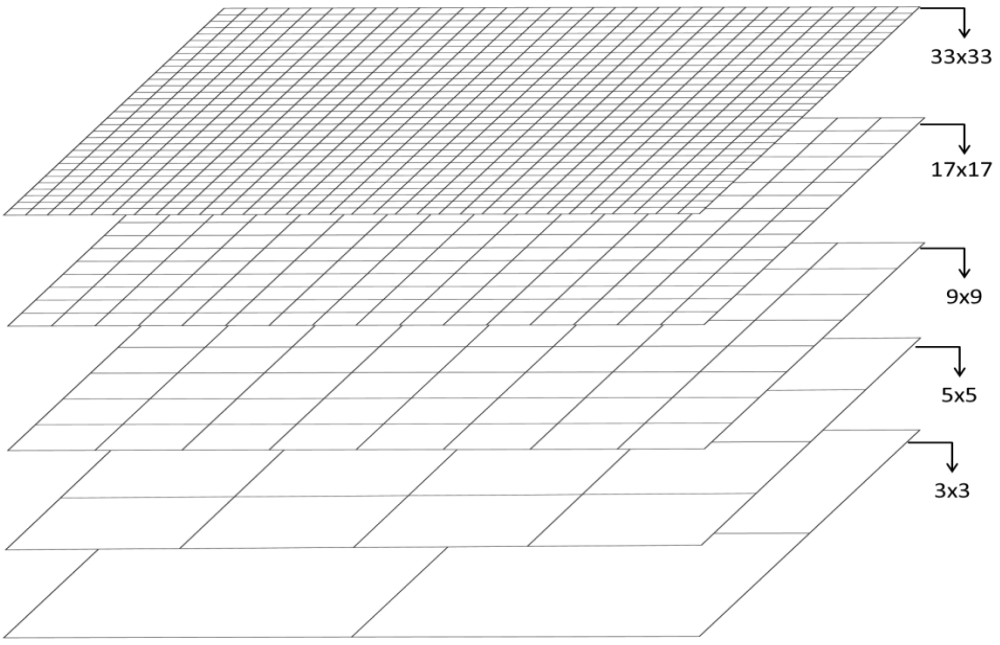
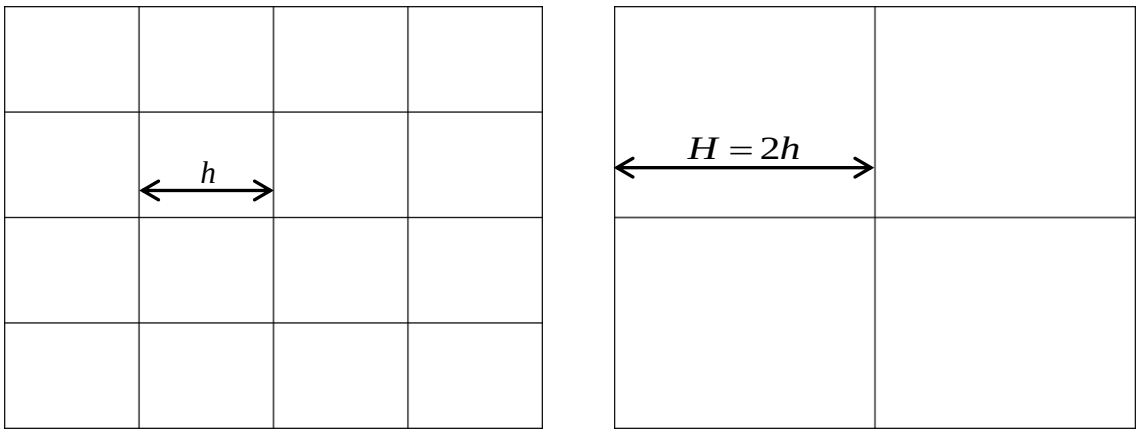


Figura 3.2: Processo de engrossamento de uma malha uniforme e razão de engrossamento $r = 2$.

A Fig. 3.2 mostra o processo de engrossamento de uma malha de 33×33 nós, ou seja, 32×32 elementos de tamanho h e razão de engrossamento $r=2$, ou seja, $H=2h$. Neste caso o processo de engrossamento inicia na malha de 33×33 nós (o primeiro nível) e termina no quinto nível, ou seja, na malha mais grossa possível (3×3 nós), totalizando $L=L_{\max}=5$ níveis.

Não existe restrição ao valor da razão de engrossamento utilizado, que pode ser $r=2$, $r=3$, $r=4$ entre outros valores, mas Brandt (1977) recomenda utilizar $r=2$ por ser o valor mais próximo da razão de engrossamento ótima e Briggs et al. (2000) observa que utilizar $r\neq2$ sem considerar as características do problema não traz vantagens.

A Fig. 3.3 mostra: a) uma malha "fina" Ω^h (uniforme) com 4×4 elementos de tamanho $h=h_x=h_y$ (5×5 nós) e b) a malha imediatamente mais grossa Ω^H (uniforme) com razão de engrossamento $r=2$, ou seja, uma malha Ω^{2h} com 2×2 elementos (3×3 nós) de tamanho $H=2h$.



a) Malha fina Ω^h b) Malha grossa $\Omega^H=\Omega^{2h}$

Figura 3.3: Exemplo de engrossamento com $r=2$.

Tendo estabelecido o papel da malha grossa, Ω^{2h}, torna-se necessário definir um procedimento para transferir o problema para esta malha (processo de restrição) e desta para a malha fina, Ω^h, (processo de prolongação). Estes processos de transferência são os temas das seções 3.3 e 3.4.

3.3 Processo de restrição

Os operadores que transferem informações da malha fina Ω^h para a malha grossa Ω^{2h} são chamados de operadores de restrição e são denotados por I_h^{2h}. Estes operadores "convertem" vetores da malha fina $\mathbf{v}^h$ em vetores da malha grossa $\mathbf{v}^{2h}$ de acordo com a regra $I_h^{2h}\mathbf{v}^h = \mathbf{v}^{2h}$.

Entre os operadores de restrição conhecidos, o mais simples é o operador de restrição por injeção (BRIGGS et al., 2000; TROTTENBERG et al., 2001), que para o caso bidimensional é dado por

$$v_{i,j}^{2h} = v_{2i,2j}^{h},\ 1 \le i \le \frac{N_x}{2} - 1,\ 1 \le j \le \frac{N_y}{2} - 1, \tag{3.2}$$

onde $N_x + 1$ e $N_y + 1$ são os números de nós da malha nas direções coordenadas x e y, respectivamente.

A Fig 3.4 mostra como o operador de injeção transfere os valores da malha fina Ω^h para a malha grossa Ω^{2h}. Neste caso, os valores dos círculos pretos da malha grossa Ω^{2h} irão receber os valores dos círculos pretos da malha fina Ω^h diretamente.

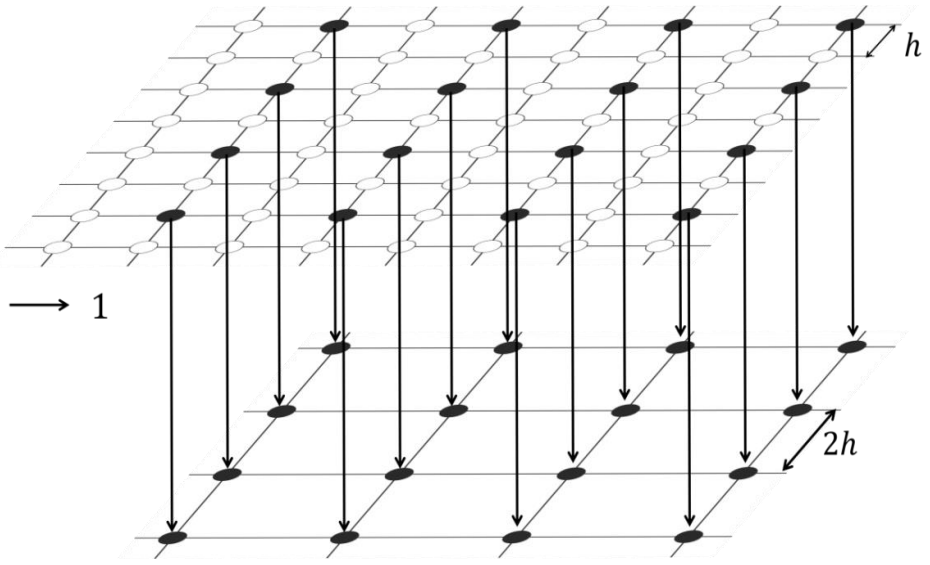


Figura 3.4: Restrição utilizando o operador de injeção.

Outro operador muito conhecido e utilizado é o operador de restrição por ponderação completa (BRIGGS et al., 2000; TROTTENBERG et al., 2001), que para o caso bidimensional é dado por

$$v_{i,j}^{2h} = \frac{1}{16}\left(v_{2i-1,2j-1}^{h} + v_{2i-1,2j+1}^{h} + v_{2i+1,2j-1}^{h} + v_{2i+1,2j+1}^{h}\right) + \frac{1}{8}\left(v_{2i,2j-1}^{h} + v_{2i,2j+1}^{h} + v_{2i+1,2j}^{h} + v_{2i-1,2j}^{h}\right) + \frac{1}{4}v_{2i,2j}^{h}, \tag{3.3}$$

com $1 \le i \le N_x/2-1$, $1 \le j \le N_y/2-1$, onde N_x+1 e N_y+1 são os números de nós da malha nas direções coordenadas x e y, respectivamente.

A Fig. 3.5 mostra como o operador de restrição por ponderação completa transfere os valores da malha fina Ω^h para a malha grossa Ω^{2h}. Neste caso, os valores dos círculos pretos da malha grossa Ω^{2h} irão receber os valores da malha fina Ω^h ponderados: $1/4$ dos círculos pretos, $1/8$ dos losangos vermelhos, $1/16$ dos quadrados azuis, como definido na Eq. (3.3).

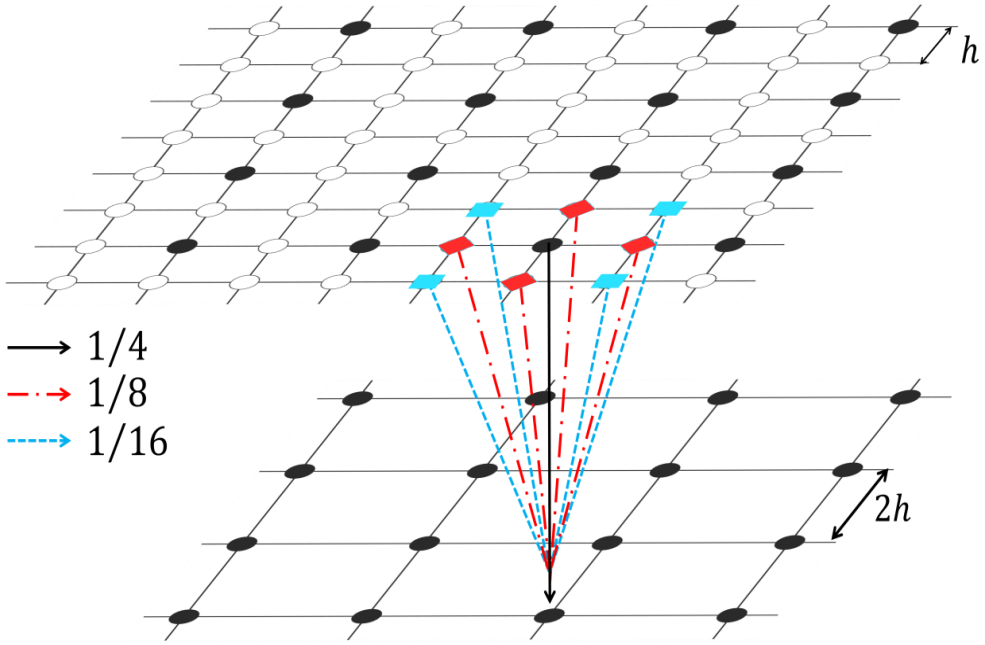


Figura 3.5: Restrição utilizando o operador de restrição por ponderação completa.

3.4 Processo de prolongação

Os operadores que transferem informações da malha grossa Ω^{2h} para a malha fina Ω^{h} são chamados de operadores de prolongação, também conhecidos como interpolação, e são denotados por I_{2h}^{h}. Estes operadores "convertem" vetores da malha grossa $\mathbf{v}^{2h}$ em vetores da malha fina $\mathbf{v}^{h}$ de acordo com a regra $I_{2h}^{h}\mathbf{v}^{2h} = \mathbf{v}^{h}$.

Entre os operadores de prolongação conhecidos, o mais simples, porém bastante eficiente, é o operador de interpolação bilinear (BRIGGS et al., 2000; TROTTENBERG et al., 2001), que para o caso bidimensional dado por

$$\begin{cases} v_{2i,2j}^{h} = v_{i,j}^{2h} \\ v_{2i+1,2j}^{h} = \dfrac{1}{2}\left(v_{i,j}^{2h} + v_{i+1,j}^{2h}\right) \\ v_{2i,2j+1}^{h} = \dfrac{1}{2}\left(v_{i,j}^{2h} + v_{i,j+1}^{2h}\right) \\ v_{2i+1,2j+1}^{h} = \dfrac{1}{4}\left(v_{i,j}^{2h} + v_{i+1,j}^{2h} + v_{i,j+1}^{2h} + v_{i+1,j+1}^{2h}\right) \end{cases}, \tag{3.4}$$

com $1 \le i \le N_x/2-1$, $1 \le j \le N_y/2-1$, onde N_x+1 e N_y+1 são os números de nós da malha nas direções coordenadas x e y, respectivamente.

A Fig. 3.6 mostra como o operador de prolongação (interpolação bilinear) transfere os valores da malha grossa Ω^{2h} para a malha fina Ω^{h}. Neste caso os valores dos círculos pretos $v_{2i,2j}^{h}$ da malha fina Ω^{h} irão receber os valores $v_{i,j}^{2h}$ da malha grossa Ω^{2h} diretamente (injeção). Os quadrados verdes $v_{2i+1,2j}^{h}$ da malha fina Ω^{h} receberão $1/2$ dos valores $v_{i,j}^{2h}$ e $v_{i+1,j}^{2h}$. Os losangos vermelhos $v_{2i,2j+1}^{h}$ da malha fina Ω^{h} receberão $1/2$ dos valores $v_{i,j}^{2h}$ e $v_{i,j+1}^{2h}$. Os triângulos azuis $v_{2i+1,2j+1}^{h}$ da malha fina Ω^{h} receberão $1/4$ dos valores $v_{i,j}^{2h}$, $v_{i+1,j}^{2h}$, $v_{i,j+1}^{2h}$ e $v_{i+1,j+1}^{2h}$.

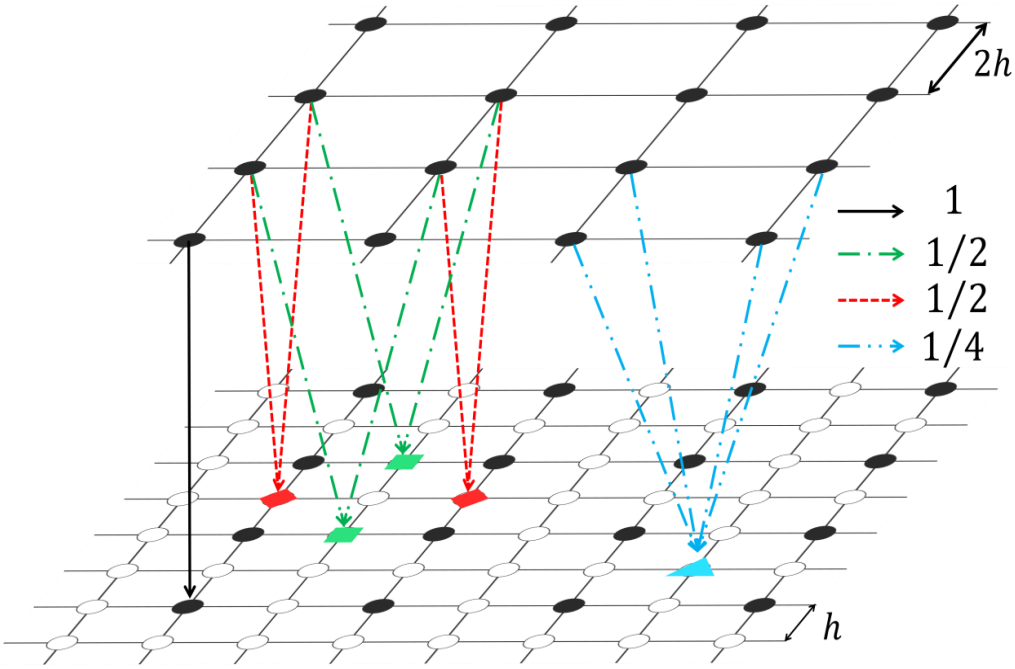


Figura 3.6: Prolongação utilizando interpolação bilinear.

3.5 Esquema de correção para duas malhas

O esquema de correção (*Correction Scheme*, CS) de duas malhas é semelhante ao procedimento chamado "correção residual" ou "refinamento iterativo", citado na seção 2.3, que consiste em:

a) Resolver $A\mathbf{u} = \mathbf{b}$ (com um método iterativo) e obter $\mathbf{v}$;
b) Calcular o resíduo com $\mathbf{r} = \mathbf{b} - A\mathbf{v}$;
c) Resolver $A\mathbf{e} = \mathbf{r}$ (equação residual) e obter $\mathbf{e}$;
d) Usando a definição de erro corrigir $\mathbf{u}$ com $\mathbf{u} = \mathbf{v} + \mathbf{e}$.

No método *multigrid*, o passo c) da "correção residual" é realizado na malha grossa, ou seja, o resíduo é transferido para a malha grossa, resolve-se a equação residual, transfere-se o valor do erro para a malha fina e corrige-se a solução anterior.

Utilizando os conceitos desenvolvidos até agora pode-se formalizar a discussão anterior no algoritmo a seguir

Algoritmo 3.1: Esquema de correção (CS) para duas malhas (adaptado de Briggs et al., 2000).

Esquema de correção para duas malhas

$\mathbf{CS}\left(\mathbf{u},\mathbf{v},\mathbf{b},h,\nu_1,\nu_2\right)$

Início

1. Iterar $A^h\mathbf{u}^h=\mathbf{b}^h$ ν_1 vezes em Ω^h com estimativa inicial $\mathbf{v}^h$**;**
2. Calcular $\mathbf{r}^h=\mathbf{b}^h-A^h\mathbf{v}^h$**;**
3. Restringir o resíduo da malha fina Ω^h para a malha grossa Ω^{2h}: $I_h^{2h}\mathbf{r}^h=\mathbf{r}^{2h}$;
    a. Iterar $A^{2h}\mathbf{e}^{2h}=\mathbf{b}^{2h}$ ν_1 vezes em Ω^{2h} com estimativa inicial $\mathbf{e}^{2h}=\mathbf{0}$;
    b. Interpolar $\mathbf{e}^{2h}$ da malha grossa Ω^{2h} para a malha fina Ω^h: $I_{2h}^h\mathbf{e}^{2h}=\mathbf{e}^h$;
4. Corrigir $\mathbf{v}^h\leftarrow\mathbf{v}^h+\mathbf{e}^h$;
5. Iterar $A^h\mathbf{u}^h=\mathbf{b}^h$ ν_2 vezes em Ω^h com estimativa inicial $\mathbf{v}^h$.

Fim de CS

Assim como a matriz A^h foi obtida a partir da discretização do problema, a matriz A^{2h} pode ser obtida pela rediscretização do problema na malha grossa Ω^{2h}.

3.6 Ciclo V

O algoritmo 3.1 apresenta um esquema de correção para duas malhas apenas, mas pode ser facilmente adaptado para o caso de várias malhas. De fato, para que o *multigrid* apresente um bom desempenho, diversos níveis de malhas devem ser utilizados (TANNEHILL et al., 1997). Pinto e Marchi (2007) e Oliveira (2010) recomendam usar todos os níveis.

A ordem na qual as malhas são visitadas é chamada de ciclo *multigrid*. Existem vários tipos de ciclos *multigrid* tais como o ciclo V, o ciclo W, o ciclo dente-de-serra, entre outros (BRIGGS et al., 2000; TROTTENBERG et al., 2001; WESSELING, 1992). Neste trabalho será utilizado exclusivamente o ciclo V, pois o ciclo W é cerca de 50% mais caro em relação ao número de operações envolvidas (HIRSCH, 1988). A Fig. 3.7 apresenta um exemplo de ciclo V.

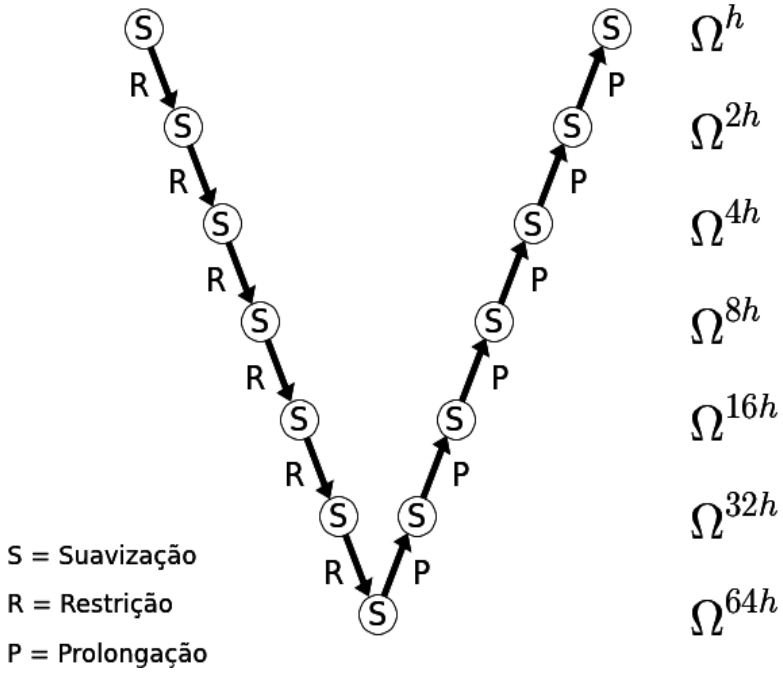


Figura 3.7: Diagrama do ciclo V.

Seguindo a orientação da Fig 3.7 pode-se adaptar o esquema de correção de duas malhas para o caso de várias malhas. Para facilitar este procedimento, foram feitas as seguintes modificações na notação do algoritmo 3.1: na equação residual o vetor $\mathbf{r}^{2h}$, foi substituído por $\mathbf{b}^{2h}$, e o vetor $\mathbf{e}^{2h}$, por $\mathbf{u}^{2h}$. Desta forma os vetores do lado esquerdo $\left(\mathbf{u}^h, \mathbf{u}^{2h}, \ldots, \mathbf{u}^{Lh}\right)$ sempre representam o vetor solução de um sistema de equações $A\mathbf{u} = \mathbf{b}$. De forma semelhante, os vetores do lado direito $\left(\mathbf{b}^h, \mathbf{b}^{2h}, \ldots, \mathbf{b}^{Lh}\right)$ sempre representam o vetor dos termos independentes. Nesta notação as equações residuais são interpretadas como um sistema de equações $A\mathbf{u} = \mathbf{b}$, onde o resíduo assume o papel do vetor de termos independentes. O algoritmo 3.2, que utiliza a notação discutida, é uma adaptação do esquema de correção para duas malhas para o caso de várias malhas.

Algoritmo 3.2: Ciclo V com esquema CS para várias malhas (adaptado de Briggs et al., 2000).

Esquema de correção para várias malhas (CS *multigrid*)

CSMG $(\mathbf{u},\mathbf{v},\mathbf{b},h,\nu_1,\nu_2)$

Início

1. Iterar $A^h\mathbf{u}^h=\mathbf{b}^h$ ν_1 vezes em Ω^h com estimativa inicial $\mathbf{v}^h$;
2. Calcular $\mathbf{r}^h=\mathbf{b}^h-A^h\mathbf{v}^h$;
3. Restringir o resíduo da malha Ω^h para a malha Ω^{2h}: $I_h^{2h}\mathbf{r}^h=\mathbf{b}^{2h}$;
    a. Iterar $A^{2h}\mathbf{u}^{2h}=\mathbf{b}^{2h}$ ν_1 vezes em Ω^{2h} com estimativa inicial $\mathbf{v}^{2h}=\mathbf{0}$;
    b. Calcular $\mathbf{r}^{2h}=\mathbf{b}^{2h}-A^{2h}\mathbf{v}^{2h}$;
    c. Restringir o resíduo da malha Ω^{2h} para a malha Ω^{4h}: $I_{2h}^{4h}\mathbf{r}^{2h}=\mathbf{b}^{4h}$;
        i. Iterar $A^{4h}\mathbf{u}^{4h}=\mathbf{b}^{4h}$ ν_1 vezes em Ω^{4h} com estimativa inicial $\mathbf{v}^{4h}=\mathbf{0}$;
        ii. Calcular $\mathbf{r}^{4h}=\mathbf{b}^{4h}-A^{4h}\mathbf{v}^{4h}$**;**
        iii. Restringir o resíduo da malha Ω^{4h} para a malha Ω^{8h}: $I_{4h}^{8h}\mathbf{r}^{4h}=\mathbf{b}^{8h}$;

        .
        .
        .

        Resolver $A^{Lh}\mathbf{u}^{Lh}=\mathbf{b}^{Lh}$**;**

        .
        .
        .

        iv. Corrigir $\mathbf{v}^{4h}\leftarrow\mathbf{v}^{4h}+I_{8h}^{4h}\mathbf{v}^{8h}$;
        v. Iterar $A^{4h}\mathbf{u}^{4h}=\mathbf{b}^{4h}$ ν_2 vezes em Ω^{4h} com estimativa inicial $\mathbf{v}^{4h}$**;**
    d. Corrigir $\mathbf{v}^{2h}\leftarrow\mathbf{v}^{2h}+I_{4h}^{2h}\mathbf{v}^{4h}$;
    e. Iterar $A^{2h}\mathbf{u}^{2h}=\mathbf{b}^{2h}$ ν_2 vezes em Ω^{2h} com estimativa inicial $\mathbf{v}^{2h}$**;**
4. Corrigir $\mathbf{v}^h\leftarrow\mathbf{v}^h+I_{2h}^h\mathbf{v}^{2h}$;
5. Iterar $A^h\mathbf{u}^h=\mathbf{b}^h$ ν_2 vezes em Ω^h com estimativa inicial $\mathbf{v}^h$.

Fim de CSMG

4 Modelos matemático e numérico

4.1 Modelo matemático

O modelo matemático considerado neste trabalho refere-se a um problema de condução de calor bidimensional linear governado pela equação de Poisson (INCROPERA et al., 2008)

$$\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}=S\left(x,y\right), \tag{4.1}$$

onde x e y são as direções coordenadas (variáveis independentes), T a temperatura (variável dependente) e $S\left(x,y\right)$ o termo fonte. O domínio de cálculo é dado por $0\le x\le 1, 0\le y\le 1$, conforme a Fig 4.1.

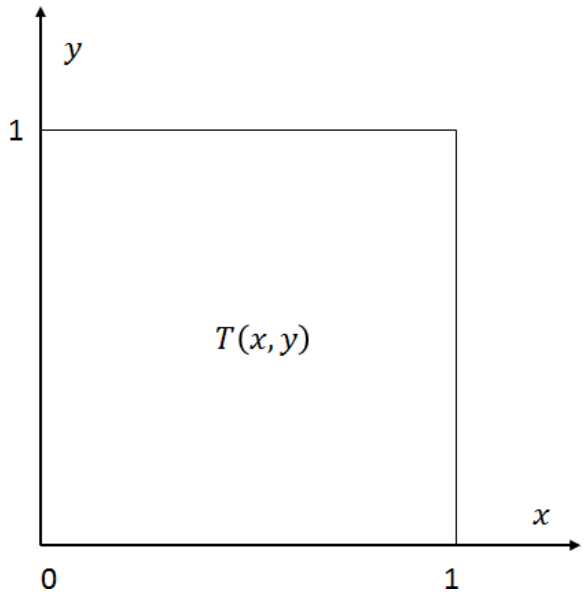


Figura 4.1: Domínio bidimensional para a equação de Laplace.

As condições de contorno de Dirichlet são dadas por

$$\begin{aligned} &T\left(0,y\right)=T\left(x,0\right)=T\left(1,y\right)=0\\ &T\left(x,1\right)=sen\left(\pi x\right) \end{aligned}. \tag{4.2}$$

Considerando-se o termo fonte $S(x,y)=0$, a equação tipo Poisson se reduz a uma equação de Laplace, que no caso apresenta como solução analítica, dada por (Oliveira, 2010)

$$T(x,y)=sen(\pi x)\frac{senh(\pi y)}{senh(\pi)}. \tag{4.3}$$

4.2 Modelo numérico

A discretização do domínio será realizada utilizando malhas uniformes nas duas direções, com $N=N_xN_y$ nós, onde N_x e N_y representam o número de nós nas direções coordenadas x e y, respectivamente.

Neste trabalho serão utilizadas apenas malhas isotrópicas tais que $N_x=N_y$, portanto o espaçamento entre os nós (ou dimensão dos elementos da malha), h_x e h_y definidos pela Eq. (1.1), deve ser igual em ambas as direções $(h_x=h_y=h)$.

Para cada um dos $(N_x-2)(N_y-2)$ nós no interior da malha, a Eq. (4.1) (com $S(x,y)=0$) é discretizada com o método das diferenças finitas (MDF), utilizando diferenças centrais (CDS), onde as derivadas são aproximadas pelas Eqs. (2.8) e (2.9). Considerando o esquema das Eqs. (2.11) para a variável T, tem-se

$$\frac{T_{i-1,j}-2T_{i,j}+T_{i+1,j}}{h^2}+\frac{T_{i,j-1}-2T_{i,j}+T_{i,j+1}}{h^2}=0. \tag{4.4}$$

A Eq. (4.4) pode ser rearranjada na forma

$$-\left(\frac{4}{h^2}\right)T_{i,j}+\left(\frac{1}{h^2}\right)T_{i-1,j}+\left(\frac{1}{h^2}\right)T_{i+1,j}+\left(\frac{1}{h^2}\right)T_{i,j-1}+\left(\frac{1}{h^2}\right)T_{i,j+1}=0, \tag{4.5}$$

ou, pelo esquema das Eqs. (2.11), na forma

$$a_P T_P + a_W T_W + a_E T_E + a_S T_S + a_N T_N = b_P . \tag{4.6}$$

Das Eqs. (4.5) e (4.6), conclui-se que

$$\begin{aligned} a_W &= a_E = a_S = a_N = \left(\frac{1}{h^2}\right) \\ a_P &= -\left(\frac{4}{h^2}\right) \\ b_P &= 0 \end{aligned} \quad . \tag{4.7}$$

Estes coeficientes são válidos para os nós internos da malha, ou seja, para $i = 2,3,\ldots,N_x - 1$ e $j = 2,3,\ldots,N_y - 1$. Para as condições de contorno impostas pela Eq. (4.2) considera-se

$$\begin{aligned} T_{1,j} &= T_{i,1} = T_{N_x,j} = 0 \\ T_{i,N_y} &= sen\left(\pi x_i\right) \end{aligned} \quad ,$$

onde $x_i = \left(i-1\right)h_x$, conforme a Eq. (1.1).

O sistema formado pela Eq. (4.6) pode ser representado por um sistema de equações na forma da Eq. (2.12), ou seja, pode ser resolvido pelos métodos iterativos discutidos no capítulo 2.

4.3 Dados de implementação

Nesta seção encontram-se os parâmetros empregados no método *multigrid* padrão utilizado neste trabalho.

O tipo de ciclo utilizado é o V, cuja descrição encontra-se na seção 3.6. O número de vezes que o ciclo V é repetido é chamado de número de iterações externas.

O processo de restrição é feito empregando-se ponderação completa, como discutido na seção 3.3. A paralelização deste processo é discutida na seção 5.8.

O processo de prolongação é realizado empregando-se o operador de interpolação bilinear discutido na seção 3.4. A paralelização deste processo é apresentada na seção 5.9.

A razão de engrossamento das malhas (r) é a razão de engrossamento padrão, ou seja, $r = 2$ (BRANDT, 1977; BRIGGS et al., 2000).

Os suavizadores (*solvers*) utilizados foram os métodos de Gauss-Seidel *red-black*, cuja paralelização pode ser encontrada na seção 5.7, e o método de Jacobi ponderado, paralelizado de acordo com a seção 5.6. As iterações realizadas pelo suavizador são chamadas de iterações internas. Neste trabalho, o número de iterações internas empregadas na pré-suavização, ν_1, é igual ao número de iterações internas utilizadas na pós-suavização, ν_2. Neste trabalho utilizou-se $\nu_1 = \nu_2 = 3$ (OLIVEIRA, 2010).

O critério de parada usado para interromper o número de iterações externas é a norma euclidiana do resíduo adimensionalizada pela norma euclidiana do resíduo na estimativa inicial (BRIGGS et al., 2000; TROTTENBERG et al. 2001), ou seja

$$\left\|\mathbf{r}\right\|_2 = \frac{\left\|\mathbf{r}^{(i)}\right\|_2}{\left\|\mathbf{r}^{(0)}\right\|_2}, \tag{4.8}$$

onde o termo $\mathbf{r}^{(i)}$ é o resíduo na iteração i e $\mathbf{r}^{(0)}$ o resíduo na estimativa inicial. Entre os trabalhos que usam essa norma pode-se citar: Briggs et al. (2000); Oliveira (2010); Oliveira et al. (2012); Trottenberg et al. (2001); Wesseling (1992) e Zhang (2002). O processo iterativo é interrompido quando a norma é menor ou igual à tolerância $\varepsilon > 0$, isto é

$$\frac{\left\|\mathbf{r}^{(i)}\right\|_2}{\left\|\mathbf{r}^{(0)}\right\|_2} \leq \varepsilon, \tag{4.9}$$

neste trabalho foi utilizado $\varepsilon = 10^{-10}$.

Em todas as simulações, o método *multigrid* partiu da malha mais fina, nível 1, e foi até a malha mais grossa, nível $L_{\max}$. A quantidade máxima de níveis $L_{\max}$ pode ser calculado com

$$L_{\max} = \log_2\left(N_x - 1\right). \tag{4.10}$$

O número de nós de cada malha pode calculado com $N = N_x N_y$ onde $1 \le \mathrm{lv} \le L_{\max}$ e

$$N_x = N_y = \left(N_x - 1\right) \Big/ 2^{(\mathrm{lv}-1)} + 1. \tag{4.11}$$

Para avaliar a paralelização utilizando as métricas discutidas na seção 2.7 é necessário medir o tempo de CPU. Entende-se por tempo de CPU o tempo gasto para realizar a geração de malhas, atribuir a estimativa inicial, calcular os coeficientes e resolver o sistema linear representado pela Eq. (4.6) até atingir a tolerância estabelecida. Este tempo é medido em segundos (s) com o uso da função OMP_GET_WTIME().

Os algoritmos foram implementados na linguagem Fortran 2003 utilizando o compilador GNU Fortran (GFortran), versão 4.6.3, com a opção OpenMP e precisão dupla. Os testes foram realizados em um servidor, disponibilizado pelo Instituto Tecnológico SIMEPAR, equipado com processadores Opteron 6276 de 2.3 GHz e totalizando 64 *threads* (processadores lógicos), das quais 55 estiveram disponíveis para testes; 32 GB de memória RAM e sistema operacional Linux 64 bits.

No Apêndice A são apresentados resultados utilizando um microcomputador equipado com processador Core I7 930 de 2.8 GHz, 6 GB de RAM e sistema operacional Windows 7 64 bits. Foi utilizado a tecnologia *Hyper-Threading* do processador Core I7 930, desta forma 8 *threads* estiveram disponíveis para testes nesse processador. A tecnologia *Turbo Boost*, que aumenta a frequência do processador de acordo com a demanda computacional (de 2.8 GHz para até 3.2 GHz), foi desligada para evitar erros nas medidas do tempo de CPU.

5 Paralelização do método *multigrid*

5.1 Particionamento da malha e trabalhos relacionados

Segundo Galante et al. (2004) existem duas formas de resolver um sistema físico de forma paralela: Decomposição de Dados e Decomposição de Domínios. Detalhes sobre a decomposição de domínios podem ser vistos em Chung (2003) e Trottenberg et al. (2001).

Neste trabalho, adota-se a estratégia da decomposição de dados através do particionamento do domínio em subdomínios, ou seja, considera-se que existe apenas um único sistema de equações para todos os subdomínios, e portanto, cada subdomínio pode ser resolvido com o mesmo algoritmo. O modelo de paralelização que aplica o mesmo programa em cada um dos subdomínios é conhecido como SPMD (*Single Program Multiple Data*) (CHAPMAN et al., 2008).

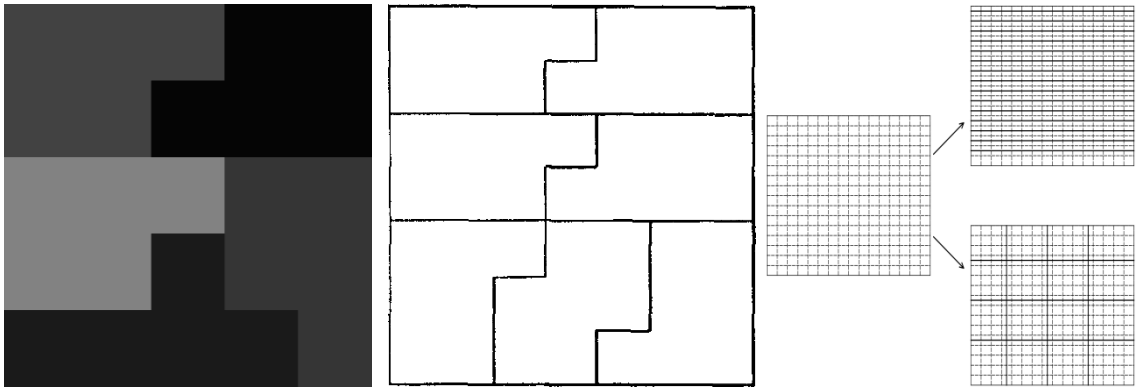
O particionamento do domínio consiste em dividir o domínio em subdomínios de forma que cada *thread* resolva uma parcela do sistema de equações que define o problema em um menor intervalo de tempo.

Não existe restrição na forma em que o domínio é dividido, mas algumas demandas precisam ser respeitadas (AL-NASRA e NGUYEN, 1991):

1) Distribuir a carga entre os processadores de forma balanceada. Cada processador deve receber carga proporcional à sua capacidade de processamento;
2) Minimizar o tempo de comunicação entre os processadores reduzindo as dimensões dos contornos dos subdomínios;
3) O tempo gasto no processo de partição do domínio deve ser pequeno em relação ao tempo gasto na solução do problema;
4) O algoritmo deve ser capaz de tratar geometrias irregulares e discretizações arbitrárias.

O problema do particionamento do domínio sujeito às restrições (1) e (2) pode ser classificado como um problema MPE (*Minimum Perimeter Equi-partition*): uma malha retangular $M \times N$ deve ser dividida em $\overline{T}$ (*threads* ou processadores) subdomínios com o mesmo número de nós e com o menor perímetro (contorno dos subdomínios) possível (CHRISTOU e MEYER, 1996; MARTIN, 1998). Muitas dos subdomínios resultantes

do MPE, que satisfazem (1) e (2), são figuras irregulares e, normalmente, representadas por uma união de retângulos (YAKEL e MEYER, 1992), como mostram as Figs 5.1 (a) e (b) extraídas de Christou e Meyer (1996) e Martin (1998) respectivamente, o que pode resultar em dificuldades na implementação do método *multigrid*.



a) Christou e Meyer (1996) b) Martin (1998) c) Trottenberg et al. (2001)

Figura 5.1: Particionamento da malha em subdomínios.

Pode-se particionar as malhas decompondo estas em subdomínios retangulares, como mostra a Fig. 5.1 (c) extraída de Trottenberg et al. (2001) onde uma malha com 16×16 elementos é particionada em subdomínios de 1×16 (1D) elementos e 4×4 (2D) elementos. Subdomínios retangulares implicam em bem menos dificuldades que os casos citados por Yakel e Meyer (1992), Christou e Meyer (1996), e Martin (1998). No entanto a diferença entre o número de nós por processador (desbalanceamento de carga) não pode ser evitada, o que pode resultar em um desempenho insatisfatório da paralelização (TROTTENBERG et al. 2001).

O algoritmo PERIX-GA de Christou e Meyer (1996) satisfaz o balanceamento de carga e minimiza o tempo de comunicação entre os processadores, no entanto torna-se inviável devido ao aumento expressivo no tempo necessário para encontrar as formas ótimas dos subdomínios (Martin, 1998).

O algoritmo MSP de Martin (1998) apresenta tempo de processamento significativamente menor que o de Christou e Meyer (1996), no entanto, o tempo de processamento cresce linearmente com N.

Revisões de outras metodologias podem ser encontradas em Donaldson (2000), no entanto, segundo este autor, as metodologias de Christou e Meyer (1996) e Martin (1998) são as que apresentam os melhores resultados.

Neste trabalho, são tratados com especial atenção o problema do balanceamento de carga e o problema do tempo gasto no processo de partição do domínio. Como foram utilizadas apenas malhas estruturadas e discretização pelo método das diferenças finitas, o item (4) é automaticamente satisfeito. O item (2) não é considerado porque a implementação e os experimentos foram realizados em sistemas com memória compartilhada.

A ausência da restrição (2) implica em encontrar subdomínios com balanceamento de carga ótimo, desta forma, representa um problema muito mais simples que os estudados por Yakel e Meyer (1992), Christou e Meyer (1996), e Martin (1998).

Neste trabalho propõe-se uma metodologia de particionamento do domínio com baixo custo computacional, que não depende do tamanho da malha utilizada e apresenta balanceamento de carga ótimo (no máximo 1 nó de diferença entre os processadores). Diferentemente dos casos apresentados na Fig. 5.1, os subdomínios resultantes da metodologia apresentada possuem forma não retangular e, em geral, com forma geométrica desconhecida, no entanto se ajustam perfeitamente aos algoritmos de paralelização que serão apresentados.

5.2 Ordenação lexicográfica

A implementação (em FORTRAN) dos conceitos desenvolvidos nesta, e outras seções, segue as recomendações de Briggs et al. (2000), ou seja, o vetor solução de cada nível de malha $\left(\mathbf{u}^h,\mathbf{u}^{2h},\ldots,\mathbf{u}^{Lh}\right)$ e o vetor de termos independentes $\left(\mathbf{b}^h,\mathbf{b}^{2h},\ldots,\mathbf{b}^{Lh}\right)$ são armazenados na memória em um único *array* 1D de forma contínua, como ilustrado na Fig 5.2.

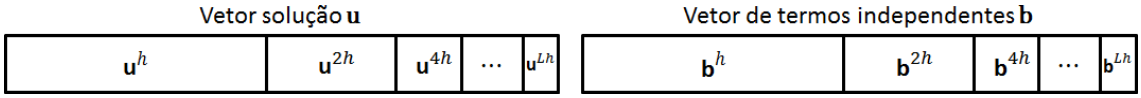


Figura 5.2: Armazenamento contínuo na memória.

A escolha pela ordenação lexicográfica dos nós do domínio deve-se à forma de armazenamento ilustrada na Fig 5.2 e ao método de particionamento da malha, principalmente pelo fato deste estar diretamente relacionado a esta ordenação (que será

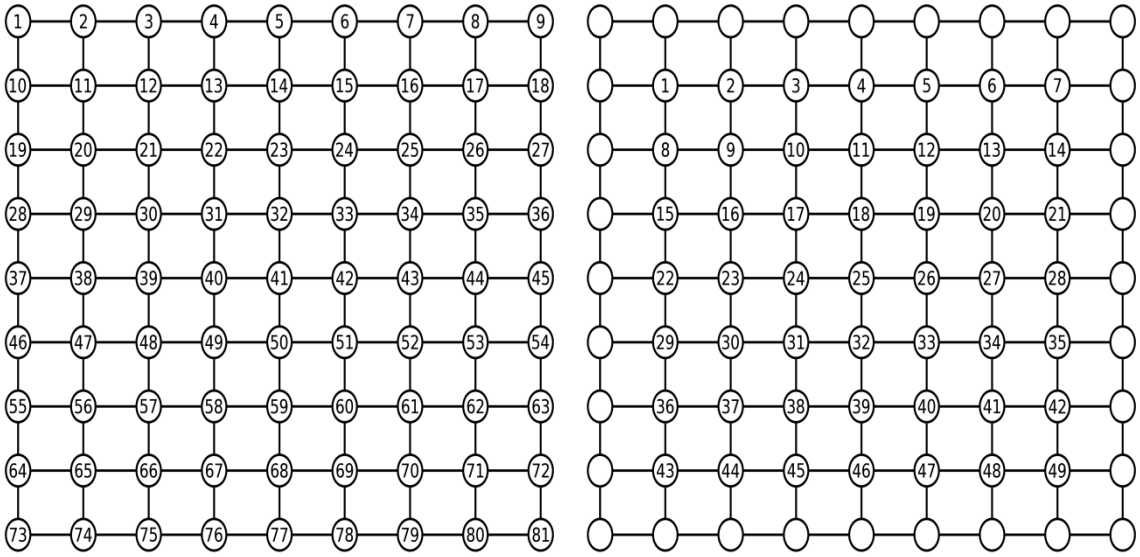
aqui chamado de ordenação lexicográfica real). A Fig 5.3 (a) demonstra este tipo de ordenação para o caso de um domínio com $N_x = N_y = 9$.

A relação entre a coordenada lexicográfica P_{real} e (i_{real}, j_{real}) é dada pela equação

$$P_{real} = i_{real} + (j_{real} - 1) N_x, \tag{5.1}$$

de forma que $P_{real} = 1, 2, \ldots, N_x N_y$. No caso da Fig 5.3 (a), tem-se que $P_{real} = 1, 2, \ldots, 81$.

Nesta dissertação são considerados modelos com condições de contorno de Dirichlet, ou seja, modelos com solução conhecida nos contornos. Assim a ordenação lexicográfica é aplicada apenas nos pontos internos, do domínio como ilustra a Fig. 5.3 (b). Esta nova ordenação será chamada de ordenação lexicográfica efetiva e suas coordenadas serão denotadas por $P_{ef} = 1, 2, \ldots, (N_x - 2)(N_y - 2)$.



a) Ordem lexicográfica real b) Ordem lexicográfica efetiva

Figura 5.3: Ordenação lexicográfica real e ordenação lexicográfica efetiva.

É necessário observar que a ordenação lexicográfica efetiva não é realmente necessária, pois a solução dos sistemas de equações é feita em relação à ordem lexicográfica real, contudo, o emprego da ordenação lexicográfica apresenta algumas vantagens:

1) A paralelização é bastante simplificada quando implementada em relação à ordem lexicográfica efetiva;

2) Esta nova ordenação preserva (ignora) os nós que pertencem aos contornos;
3) Não é necessário definir coeficientes para os nós dos contornos, ou seja, as Eqs. (4.7) são suficientes para resolver o problema estudado;
4) A quantidade de memória utilizada na implementação é substancialmente reduzida. Além disso, reduz o número de acessos à memória realizada pelos processadores.

A principal desvantagem que pode ser apontada está no fato de que esta nova ordenação aumenta o custo computacional SERIAL do algoritmo, pois exige que os índices da ondem lexicográfica efetiva sejam "convertidos" para índices da ordem lexicográfica real (mais detalhes na discussão do algoritmo 5.1).

As coordenadas da ordenação lexicográfica efetiva podem ser escritas como

$$P_{ef} = i_{ef} + \left(j_{ef} - 1 \right)\left(N_x - 2 \right), \tag{5.2}$$

com

$$\begin{aligned} j_{ef} &= j_{real} - 1 \\ i_{ef} &= i_{real} - 1 \end{aligned}. \tag{5.3}$$

Além disso,

$$j_{ef} = \left\lfloor \frac{P_{ef} - 1}{N_x - 2} \right\rfloor + 1, \tag{5.4}$$

onde o símbolo $\lfloor \ \rfloor$ refere-se à função menor inteiro, denotada por $floor$ e definida por (GRAHAM et al., 1989)

$$floor\colon\ R \to Z \text{ , com } x \mapsto floor(x) = \max\left\{ z \in Z \middle| z \le x \right\}. \tag{5.5}$$

Substituindo a Eq. (5.3) na Eq. (5.2) e resolvendo o sistema formado pelas Eqs. (5.1) e (5.2), chega-se a

$$P_{real} = P_{ef} + 2\left(j_{ef} + 1 \right) + N_x - 3 . \tag{5.6a}$$

Empregando a Eq. (5.4) na Eq. (5.6a), obtém-se

$$P_{real} = P_{ef} + 2\left\lfloor \frac{P_{ef} - 1}{N_x - 2} \right\rfloor + N_x + 1 . \tag{5.6b}$$

A Eq. (5.6b) é a equação que relaciona a ordenação lexicográfica real e efetiva. Para o caso da Fig. 5.3 (a), tem-se

$$\begin{aligned} P_{ef} &= 1 \Rightarrow P_{real} = 11; \\ P_{ef} &= 2 \Rightarrow P_{real} = 12; \\ &\vdots \\ P_{ef} &= 49 \Rightarrow P_{real} = 71. \end{aligned}$$

5.3 Subdomínios e balanceamento de carga

O modelo de paralelização SPMD consiste na aplicação do mesmo programa (mesmo conjunto de instruções sequenciais) em cada subdomínio. Logo, pode-se concluir que, para o caso de processadores com as mesmas especificações (desempenho), o balanceamento de carga pode ser atingido através de uma divisão do domínio em subdomínios de mesma dimensão.

De acordo com a conclusão acima, o número de nós de cada subdomínio pode ser calculado como

$$N_{sub} = \left\lfloor \frac{\left(N_x - 2 \right)\left(N_y - 2 \right)}{\bar{T}} \right\rfloor , \tag{5.7}$$

onde $\bar{T}$ é o número total de *threads* disponíveis na máquina.

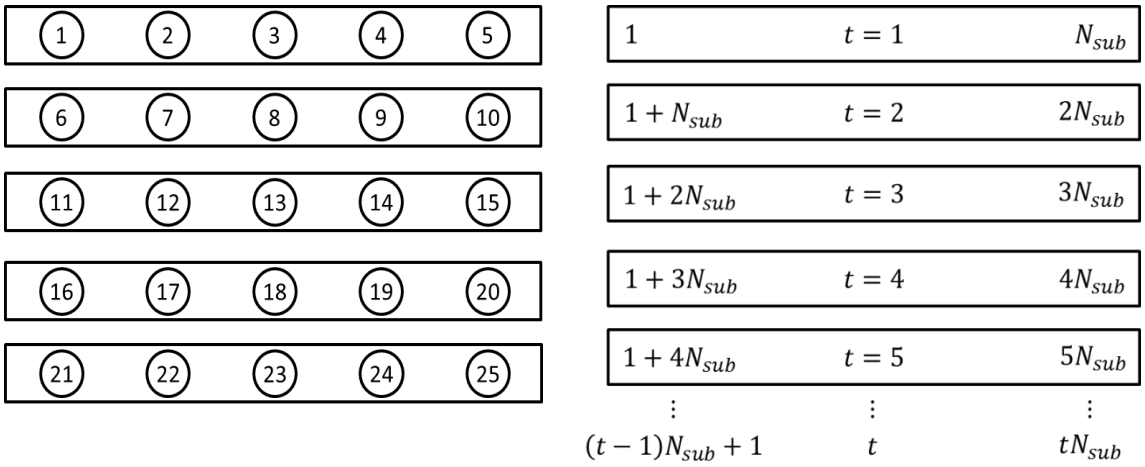
A Eq. (5.7) distribui o número de nós no interior do domínio entre os *threads* disponíveis. Uma consequência importante da Eq. (5.7) é que o número máximo de nós que podem "sobrar" na divisão é $\bar{T} - 1$, e estes podem ser redistribuídos entre os

threads de forma que a diferença entre o número de nós entre os subdomínios não seja maior que um, técnica que será apresentada a seguir.

A Fig 5.4 (a) mostra um domínio com 25 nós distribuídos entre 5 *threads* de forma que $N_{sub}=5$. Na Fig 5.4 (b), $t=1,2,\ldots,\bar{T}$ é o número que identifica o *thread* e, portanto, o subdomínio do qual este vai se encarregar. Ainda na Fig. 5.4 (b), pode-se notar como os índices do primeiro e do último nó de cada subdomínio estão relacionados com o *thread* t. O caso geral, onde a divisão na Eq. (5.7) não é exata, portanto existem nós que precisam ser redistribuídos, é apresentado na Fig. 5.5.

No exemplo da Fig. (5.5) (a) um domínio com 28 nós é distribuído entre 5 *threads* com a Eq. (5.7), de forma que 3 nós precisam ser redistribuídos. A escolha de quais *threads* receberão os nós que "sobram" é arbitrária, desde que cada *thread* receba apenas 1 nó extra. Nesta dissertação, optou-se por redistribuir os nós entre os *threads* $t \geq t_c$, onde

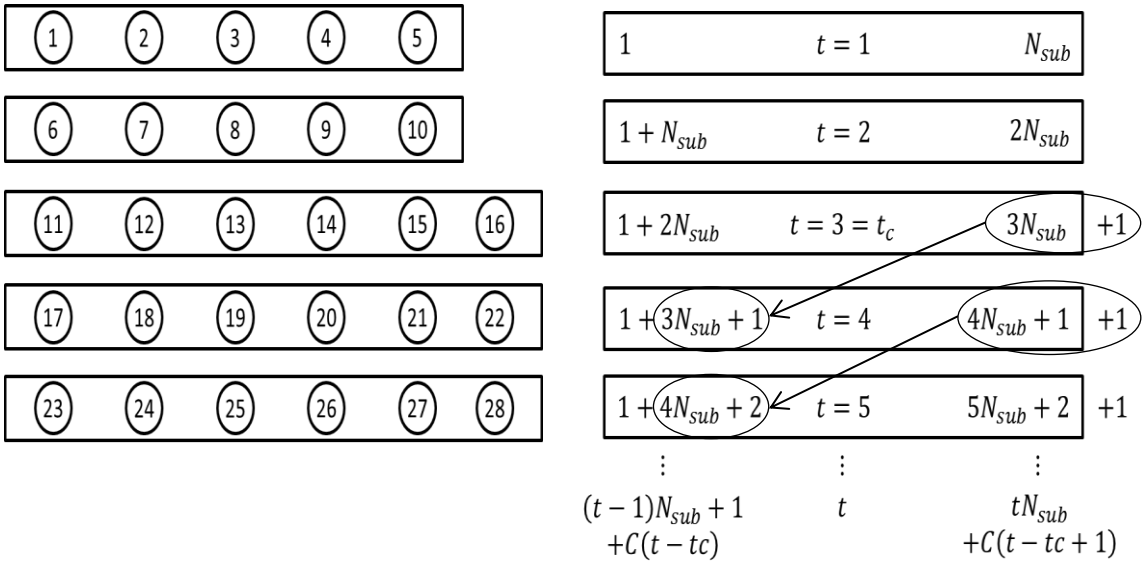
$$t_c = \bar{T} + 1 - \left[\left(N_x - 2\right)\left(N_y - 2\right) - \bar{T}N_{sub} \right]. \tag{5.8}$$



a) Domínio dividido em 5 *threads* b) Início e fim de cada subdomínio

Figura 5.4: Divisão exata de um subdomínio.

Cada *thread* a partir de t_c (este *thread* incluso) receberá 1 nó dos que sobraram na divisão do domínio com a Eq. (5.7). No exemplo da Fig. 5.5 $t_c = 3$, ou seja, todos os *threads* a partir do terceiro, $t=3$, 4 e 5, receberão 1 nó extra. Na Eq. (5.8), pode-se notar que $\bar{T} - t_c + 1$ é igual ao número de nós que sobram na divisão da Eq. (5.7).



a) Domínio dividido em 5 *threads* b) Início e fim de cada subdomínio

Figura 5.5: Divisão não exata de um subdomínio.

Na Fig. 5.5 (b) cada *thread* $t \geq t_c$ recebe $+1$ nó (colocado fora do subdomínio para ressaltar que a escolha é arbitrária) em seu subdomínio de forma que os índices do primeiro e último nó precisam ser corrigidos com o acréscimo dos termos $C\left(t-t_c\right)$ e $C\left(t-t_c+1\right)$, respectivamente. O valor C é definido por

$$C=\begin{cases}1, \text{se } t \geq t_c \\ 0, \text{se } t < t_c\end{cases}. \tag{5.9}$$

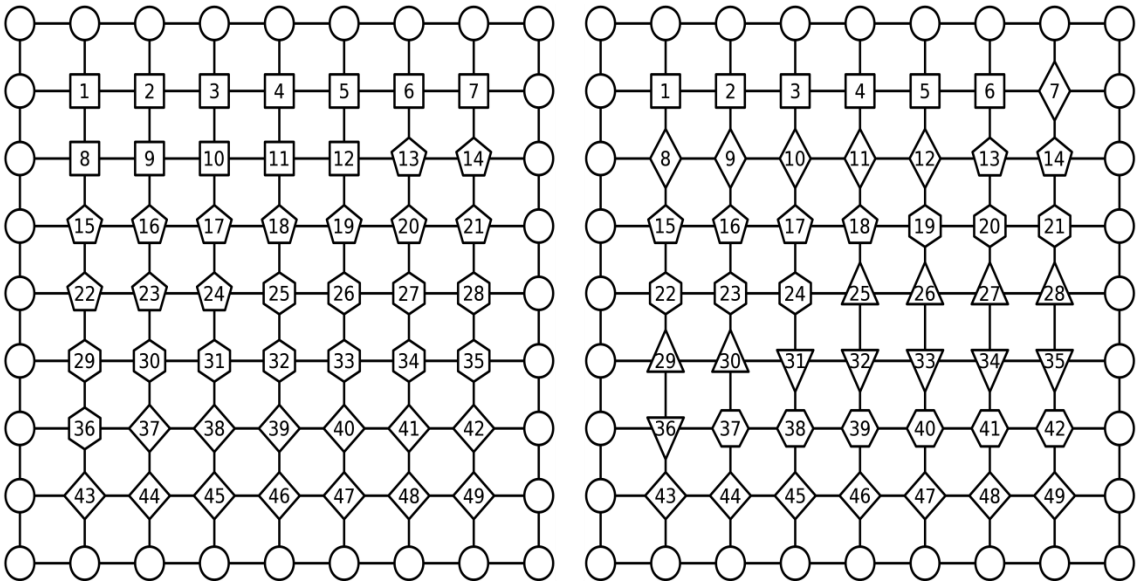
Da Fig. 5.5 (b) pode-se concluir que os índices que particionam o domínio em subdomínios, na ordenação lexicográfica efetiva, são dados por

$$\begin{aligned} i_t &= \left(t-1\right)N_{sub}+1+C\left(t-t_c\right) \\ f_t &= tN_{sub}+C\left(t-t_c+1\right) \end{aligned}, \tag{5.10}$$

onde o índice i_t representa o início e f_t representa o fim de cada subdomínio. O termo C é responsável por "ativar" a redistribuição dos nós que "sobraram" na Eq. (5.7).

A Eq. (5.10) resulta em um particionamento do domínio com ótimo balanceamento de carga, pois implica que um *thread* vai resolver, no máximo, uma equação a mais que

os outros. Além disso, implica também que o número de subdomínios sempre será igual ao número de *threads* disponíveis, evitando que processadores fiquem ociosos.



a) 4 *threads* b) 8 *threads*

Figura 5.6: Domínio com 49 nós particionado em (a) 4 subdomínios e (b) 8 subdomínios.

A Fig. 5.6 ilustra um domínio com 49 nós distribuídos, utilizando a Eq. (5.10), entre: (a) 4 *threads* e (b) 8 *threads*. As formas geométricas representam os subdomínios. Pode-se observar nas duas situações, (a) e (b), que a diferença no número de nós entre os subdomínios não passa de um.

Como cada domínio (malha utilizada no método *multigrid*) precisa ser particionado apenas uma vez, o custo computacional da Eq. (5.10) pode ser considerado pequeno em relação ao custo de resolver um sistema de equações com muitas incógnitas.

5.4 Resíduo em paralelo

O resíduo pode ser calculado pela Eq. (2.28)

$$\mathbf{r} = \mathbf{b} - A\mathbf{v}\,.$$

É importante notar que $\mathbf{v}$ é a solução aproximada de $\mathbf{T}$ na Eq. (4.1), portanto, A deve ter a forma pentadiagonal apresentada na Eq. (4.6). Considerando o esquema dos

índices apresentado na Fig. 2.1 e a Eq. (4.6), pode-se reescrever a Eq. (2.28) como

$$r_P = b_P + a_W v_W + a_E v_E + a_S v_S + a_N v_N - a_P v_P , \tag{5.11}$$

onde deve-se considerar $P = P_{real}$, $W = P_{real} - 1$, $E = P_{real} + 1$, $S = P_{real} + N_x$ e $N = P_{real} - N_x$.

Com a Eq. (5.11), o resíduo pode ser calculado em paralelo utilizando o algoritmo 5.1, que deve ser executado simultaneamente em todos os *threads*.

Fica subentendido que as variáveis contidas nos termos **Entrada** foram inicializadas e entregues aos algoritmos com os valores apropriados. O termo **Privado**, utilizado neste algoritmo e em outros desta dissertação, garante que cada *thread* vai possuir uma cópia exclusiva de variáveis declaradas como privadas. Variáveis privadas não podem ser modificadas por outros *threads*.

As variáveis de entrada são:

r : vetor que conterá o cálculo do resíduo;

v : vetor contendo a solução aproximada;

N_x : o número de nós da malha na direção coordenada x (incluindo os contornos);

a_P, a_W, a_E, a_S, a_N, b_P : são os coeficientes definidos nas Eqs. (4.7), cujos índices estão de acordo com a Fig. 2.1;

i_t, f_t : início e fim do subdomínio pertencente ao *thread* t, calculados com a Eq. (5.10).

Algoritmo 5.1: Resíduo em paralelo.

Resíduo em paralelo

Entrada $(\mathbf{r}, \mathbf{v}, a_P, a_W, a_E, a_S, a_N, b_P, N_x, i_t, f_t)$

Início do Paralelismo

Privado (P_{ef}, P_{real})

1. Para P_{ef} no intervalo $[i_t, f_t]$**;**
    a. Calcular P_{real} com a Eq. (5.6b);
    b. Calcular $r_{P_{real}}$ com a Eq. (5.11).

Fim do Paralelismo

No passo 1 é definido que os índices P_{ef} deverão estar no intervalo fechado $[i_t, f_t]$, ou seja, P_{ef} irá "varrer" todos os nós dentro de um subdomínio t. Como cada *thread* tem seu próprio subdomínio $[i_t, f_t]$, nota-se que a paralelização está sendo realizada em relação aos índices P_{ef} (ordem lexicográfica efetiva). Deve-se observar, Fig. 5.3 (b), que P_{ef} não "passa" pelos contornos, ou seja, P_{real}, W, E, S e N estão definidos apenas para o domínio.

Como observado, a paralelização é realizada em relação os índices P_{ef}, no entanto, a Eq. (5.11) precisa ser resolvida em relação aos índices P_{real}, o que é feito no passo 1.a. No passo 1.b, a Eq. (5.11) é resolvida utilizando a ordem lexicográfica real.

A conversão dos índices P_{ef}, da ordem lexicográfica efetiva, nos índices P_{real}, da ordem lexicográfica real, implica em um custo computacional que poderia ser evitado caso fosse utilizado apenas a ordem lexicográfica real: neste caso, P_{ef} seria igual a P_{real}, de forma que o passo 1.a não existiria; contudo, cuidados adicionais teriam que ser tomados ao "passar" pelos nós presentes nos contornos, além disso, novos coeficientes teriam que ser definidos para estes nós.

Como o passo 1.a (conversão de P_{ef} para P_{real}), é recorrente em vários algoritmos desta dissertação, pode-se evitar este passo calculando P_{real} apenas uma vez e armazenando os resultados em um vetor, contudo este procedimento implica em maior uso de memória e acesso à mesma pelos processadores.

Em concordância com os algoritmos apresentados nesta dissertação, as simulações foram realizadas utilizando a conversão de P_{ef} para P_{real} como descrito nos algoritmos.

Como pode-se observar, o algoritmo 5.1 representa o cálculo serial do resíduo e a paralelização consiste em executar o mesmo em cada *thread* independentemente.

5.5 Norma euclidiana em paralelo

A norma euclidiana, definida na Eq. (2.26), quando aplicada na Eq. (5.11) resulta em

$$\left\|\mathbf{r}\right\|_2 = \left(\sum_{P_{ef}=1}^{(N_x-2)(N_y-2)} r_{P_{real}(P_{ef})}^2 \right)^{1/2}, \tag{5.12}$$

que é calculada apenas no interior do domínio, ou seja, utilizando a ordenação lexicográfica efetiva P_{ef}. Como o vetor resíduo está definido para todo o domínio (que possui $N = N_x N_y$ nós), os índices do mesmo correspondem à ordem lexicográfica real. Na Eq. (5.12), o termo $P_{real}\left(P_{ef}\right)$ implica que o índice P_{ef} do somatório deve ser convertido em P_{real} com a Eq. (5.6b).

Para que possa ser calculada em paralelo, a Eq. (5.12) pode ser adaptada da seguinte forma

$$\left\|\mathbf{r}\right\|_2 = \left(\sum_{t=1}^{\bar{T}} r'_t \right)^{1/2}, \tag{5.13}$$

onde

$$r'_t = \sum_{P_{ef}=i_t}^{f_t} r_{P_{real}(P_{ef})}^2 . \tag{5.14}$$

O termo r'_t é a soma dos quadrados dos resíduos do subdomínio t e, portanto, pode ser calculado em paralelo. A Eq. (5.12) é equivalente às Eqs. (5.13) e (5.14), pois:

$$\sum_{t=1}^{\bar{T}} \sum_{P_{ef}=i_t}^{f_t} = \sum_{P_{ef}=1}^{f_1} + \sum_{P_{ef}=i_2}^{f_2} + \cdots + \sum_{P_{ef}=i_{\bar{T}}}^{(N_x-2)(N_y-2)} = \sum_{P_{ef}=1}^{(N_x-2)(N_y-2)} ,$$

onde $i_1 = 1$ e $f_{\bar{T}} = \left(N_x - 2\right)\left(N_y - 2\right)$, como pode ser demonstrado com a Eq. (5.10).

O algoritmo 5.2, que deve ser executado simultaneamente em todos os *threads*, apresenta como este processo pode ser realizado.

As variáveis de entrada são:

r : vetor contendo o cálculo do resíduo;

$\mathbf{r}'$: vetor que armazenará a soma dos quadrados dos resíduos do subdomínio t, e que tem como componentes os termos r_t', ou seja, $\left(r_1', r_2', r_3', \ldots, r_{\bar{T}}'\right)$;

N_x: o número de nós da malha na direção coordenada x (incluindo os contornos);

i_t, f_t: início e fim do subdomínio pertencente ao *thread* t, calculados com a Eq. (5.10);

t: o número que define o *thread*.

Algoritmo 5.2: Eq. (5.14) em paralelo.

Cálculo dos termos r_t' em paralelo

Entrada $\left(\mathbf{r}, \mathbf{r}', N_x, i_t, f_t, t\right)$

Início do Paralelismo

Privado $\left(P_{ef}, P_{real}\right)$

1. $r_t' = 0$;
2. Para P_{ef} no intervalo $\left[i_t, f_t\right]$**;**
    a. Calcular P_{real} com a Eq. (5.6b);
    b. Calcular o termo $r_t' = r_t' + r_{P_{real}}^2$.

Fim do Paralelismo

5.5.1 Soma dos termos r_t' em paralelo

Nesta seção sugere-se uma forma de somar em paralelo os termos r_t'. No entanto esta seção pode ser considerada opcional pois, para o número de *threads* testados (55), não apresenta redução do tempo de CPU significativa em relação à versão serial da soma. Além disso, o procedimento sugerido está muito distante da simplicidade almejada.

Para que a Eq. (5.13) possa ser calculada em paralelo, sugere-se o procedimento que soma r_t' ilustrado pela Fig. 5.7, onde l indica a linha em que os processadores representados pelos quadrados somarão seus valores com os valores dos processadores "vizinhos" (representados pelos círculos). O termo $v = 2^{(l-1)}$ pode ser interpretado como a "distância" entre um processador e seu "vizinho". Neste caso, o "vizinho" do

processador t será o processador $t+v$. Em cada linha l, os processadores representados pelos quadrados podem realizar a soma em paralelo. No final do processo, linha $l=5$ no exemplo da Fig. 5.7, o termo $r_t'=r_1'$ conterá a soma todos os termos r_t'. A Eq. (5.13) pode ser escrita (no final do processo) como

$$\|\mathbf{r}\|_2=(r_1')^{1/2}.$$

Na Fig 5.7, onde 11 processadores são utilizados, pode-se notar que na primeira linha apenas os processadores 1, 3, 5, 7 e 9 realizam a soma; na segunda linha apenas os processadores 1, 5, e 9; na terceira apenas o processador 1 realiza a soma e seu vizinho é o processador 5; na quarta linha apenas o processador 1 realiza a soma e seu vizinho é o processador 9; o processo de soma termina na quinta linha (o processador 1 não tem vizinho).

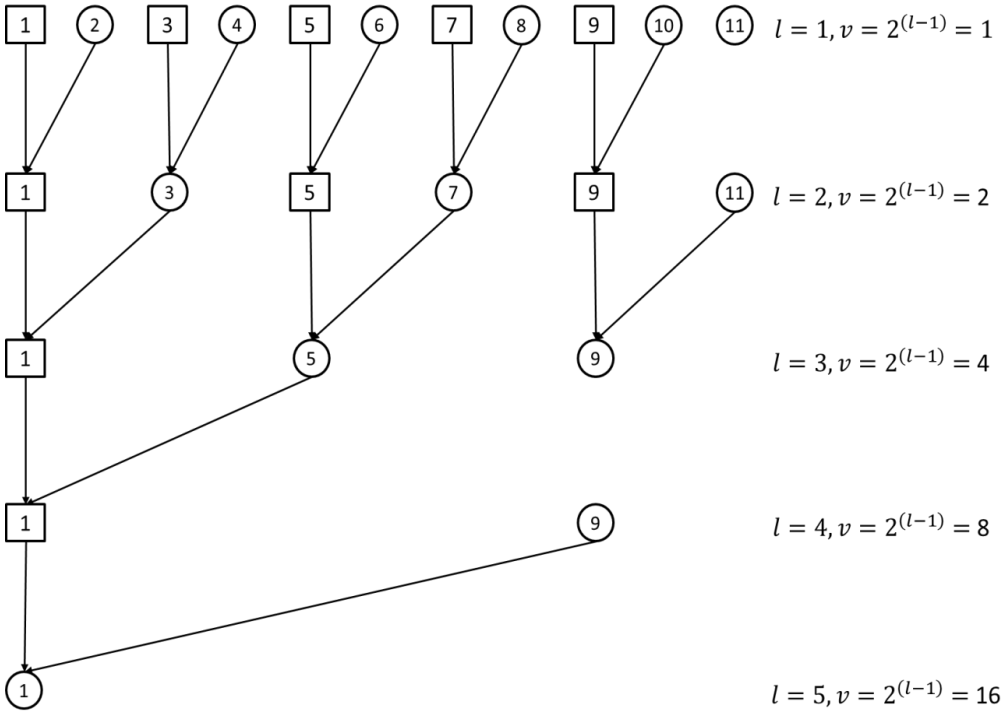


Figura 5.7: Soma dos resíduos parciais de 11 processadores em paralelo.

Calcular o número l_t de linhas nas quais um processador realiza a soma é parte importante do algoritmo utilizado para resolver a Eq. (5.13) em paralelo, este valor pode ser deduzido de uma condição que depende, basicamente de dois números, l_{1t} e l_{2t}.

Primeiramente, observa-se na Fig. 5.7 que os processadores ativos (quadrados) em uma linha l satisfazem (nem sempre, a razão será apresentada na discussão do número l_{2t}):

$$\frac{t-1}{2^l} = \text{inteiro}.$$

Esta relação, $(t-1)/2^l = \text{inteiro}$, decorre da expressão t=1+$(n-1)2^l$, onde t é o número que identifica o processador ativo (quadrado) e n, sua "posição" na linha l (com apenas processadores ativos). Considerando a linha $l=1$, no exemplo da Fig. 5.7, o processador $t=1$ ocupa a "posição" $n=1$, o processador $t=3$ ocupa a "posição" $n=2$ e, de forma semelhante, os processadores $t=5$, 7 e 9 ocupam as "posições" $n=3$, 4 e 5. Como n deve ser inteiro, conclui-se que $(t-1)/2^l = \text{inteiro}$.

Para $t=\text{par}$, $(t-1)/2^l \neq \text{inteiro}$, ou seja, processadores pares sempre serão círculos (não realizarão somas). O número l_{1t} é obtido considerando o maior valor $2^{l_{1t}}$ para o qual $(t-1)/2^l$ seja inteiro, ou seja, l_{1t} deve ser o número máximo de vezes que $(t-1)$ é divisível por 2.

O número l_{1t} pode ser interpretado como um indicador do número linhas em que um processador realiza soma, mas não pode ser considerado l_t pois, no exemplo da Fig. 5.7, não é válido para o caso dos processadores 1 (0 é infinitamente divisível por 2) e 9 (8 é 3 vezes divisível por 2). Ambos os casos podem ser resolvidos considerando que cada processador t tem como vizinho o processador $t+v$. O número l_{2t} pode ser obtido impondo que o vizinho $t+v$ mais distante seja o processador $\overline{T}$ (número de *threads*)

$$t + 2^{l_{2t}-1} \leq \overline{T},$$

onde, de acordo com a Fig. 5.7, foi considerado $v = 2^{(l-1)}$ e $l = l_{2t}$. Resolvendo para l_{2t}, tem-se

$$l_{2t} = \log_2\left(\overline{T} - t\right) + 1.$$

Como l_{2t} deve ser inteiro e $\left\lfloor \log_2\left(\overline{T} - t\right) + 1 \right\rfloor$ é o maior inteiro menor que $\log_2\left(\overline{T} - t\right) + 1$, pode-se utilizar (GRAHAM et al, 1989)

$$l_{2t} = \left\lfloor \log_2\left(\overline{T} - t\right) + 1 \right\rfloor = \left\lfloor \log_2\left(\overline{T} - t\right) \right\rfloor + 1.$$

Resumindo:

a) l_{1t} representa o número máximo de vezes que $t-1$ é divisível por 2 e é válido para a maioria dos valores de t. Para t par, $t-1$ será ímpar, logo $l_{1t} = 0$;

b) $l_{2t} = \left\lfloor \log_2\left(\overline{T} - t\right) \right\rfloor + 1$ resolve os casos extremos $t = 1$ e $t + v > \overline{T}$. Para $t = \overline{T}$ o *thread* não tem nenhum vizinho, logo $l_{2t} = 0$.

O valor l_t pode ser calculado de l_{1t} e l_{2t} através de

$$l_t = \min\left(l_{1t}, l_{2t}\right) . \tag{5.15a}$$

O valor l_t pode ser calculado utilizando apenas o número l_{2t} através do máximo divisor comum. Sendo $mdc(a,b)$ o máximo divisor comum entre a e b, então

$$mdc\left(t-1, 2^{l_{2t}}\right) = 2^{l_t} ,$$

e portanto

$$l_t = \log_2\left(mdc\left(t-1, 2^{l_{2t}}\right)\right). \tag{5.15b}$$

A Eq. (5.15b) é verdadeira porque:

1) se t é ímpar, então $t-1$ pode ser decomposto em $2^{l_{1t}} P$, onde P é um inteiro qualquer, logo

$$mdc\left(2^{l_{1t}}P, 2^{l_{2t}}\right) = mdc\left(2^{l_{1t}}, 2^{l_{2t}}\right) = \min\left(2^{l_{1t}}, 2^{l_{2t}}\right) \Rightarrow$$
$$l_t = \log_2\left(\min\left(2^{l_{1t}}, 2^{l_{2t}}\right)\right) = \min\left(l_{1t}, l_{2t}\right)$$

2) se t for par então

$$mdc\left(t-1, 2^{l_{2t}}\right) = 1 \Rightarrow l_t = \log_2(1) = 0 .$$

como esperado, processadores pares não realizam soma (ver Fig. 5.7).

Entre os vários métodos para se calcular o máximo divisor comum, citam-se dois: a fórmula explícita (POLEZZI, 1997), dada por:

$$mdc(a,b) = 2\sum_{k=1}^{a-1}\left\lfloor \frac{kb}{a} \right\rfloor + a + b - ab ,$$

e o algoritmo de Euclides, descrito a seguir

Algoritmo 5.3: Algoritmo de Euclides.

Algoritmo de Euclides para cálculo do *mdc(a,b)*

Entrada (a,b)

1. Definir $mdc = b$**;**
2. Calcular o *resto* da divisão a/b com

$$resto = a - \left\lfloor \frac{a}{b} \right\rfloor b ;$$

3. Se $resto \neq 0$ fazer**;**
    a. Atualizar a com $a = b$**;**
    b. Atualizar b com $b = resto$**;**
    c. Voltar ao passo 1.

Fim

Neste trabalho, o cálculo de l_t é feito utilizando a Eq. (5.15b). Para o cálculo do *mdc* optou-se pelo algoritmo de Euclides.

O algoritmo 5.4, utilizando o procedimento ilustrado na Fig. 5.7, paraleliza a Eq. (5.13)

$$\|\mathbf{r}\|_2 = \left(\sum_{t=1}^{\bar{T}} r_t' \right)^{1/2}.$$

O termo **Barreira,** utilizado no algoritmo e em outros desta dissertação, garante que os *threads* irão somar apenas os valores r_t' que estão na mesma linha l, como exige o processo definido na Fig. 5.7.

A variável de entrada $\mathbf{r}'$, é o vetor contendo os termos r_t'.

Algoritmo 5.4: Eq. (5.13) em paralelo e norma euclidiana.

Soma dos termos r_t' em paralelo e cálculo da norma euclidiana

Entrada $(\mathbf{r}')$

Início do Paralelismo

Privado (l)

1. Calcular l_t com a (5.15b)**;**
2. Para l (linha) no intervalo $[1, l_t]$ (se $l_t = 0$ vá para o passo 3)**;**
    a. Calcular o termo: $r_t' = r_t' + r_{t+2^{l-1}}'$ **;**
    b. **Barreira (espera que todos os *threads* atinjam este ponto);**
3. Em apenas um *thread*, calcular a norma euclidiana do resíduo $\|\mathbf{r}\|_2 = (r_1')^{1/2}$.

Fim do Paralelismo

No passo 2.b é realizado uma forma de sincronização dos *threads* para garantir a consistência no cálculo de r_t'. Após o término do passo 2, o resíduo de todo o domínio estará acumulado no termo r_1', então $(r_1')^{1/2}$ deve ser igual a norma euclidiana do resíduo.

5.6 Jacobi ponderado em paralelo

Considerando a convenção apresentada na Fig. 2.1, o método de Jacobi ponderado (seção 2.2.1) poder ser escrito, na forma da Eq. (4.6), como

$$u_P^{(i+1)} = u_P^{(i)} + \omega\left(u_P^{*} - u_P^{(i)}\right), \tag{5.16}$$

com

$$u_P^{*} = \left(a_W u_W^{(i)} + a_E u_E^{(i)} + a_S u_S^{(i)} + a_N u_N^{(i)} - b_P\right) \Big/ a_P \,, \tag{5.17}$$

onde o sobrescrito i representa a i-ésima iteração e $\omega = 2/3$ é o fator de ponderação ótimo (BRIGGS et al, 2000). Deve-se considerar $P = P_{real}$, $W = P_{real} - 1$, $E = P_{real} + 1$, $S = P_{real} + N_x$ e $N = P_{real} - N_x$.

O algoritmo 5.5 é a versão do método Jacobi ponderado em paralelo utilizado nesta dissertação. As variáveis de entrada são:

$\mathbf{u}$: vetor contendo a estimativa (com as condições de contorno já atualizadas);

$\mathbf{u_0}$: vetor contendo a estimativa anterior (atualizado com os valores do vetor $\mathbf{u}$);

ω: o fator de ponderação a ser utilizado no método Jacobi ponderado;

a_P, a_W, a_E, a_S, a_N, b_P: são os coeficientes definidos nas Eqs. (4.7), cujos índices estão de acordo com a Fig. 2.1;

i_t, f_t: início e fim do subdomínio pertencente ao *thread* t, calculados com a Eq. (5.10).

No passo 1 é definido que os índices P_{ef} deverão estar no intervalo fechado $[i_t, f_t]$, ou seja, P_{ef} irá "varrer" todos os nós dentro de um subdomínio t. Como cada *thread* tem seu próprio subdomínio, $[i_t, f_t]$, nota-se que a paralelização está sendo realizada em relação aos índices P_{ef} (ordem lexicográfica efetiva) e a Eq. (5.16) está sendo resolvida em relação aos índices P_{real} (ordem lexicográfica real).

Algoritmo 5.5: Jacobi ponderado em paralelo.

Jacobi ponderado em paralelo

Entrada $\left(\mathbf{u}, \mathbf{u_0}, \omega, a_P, a_W, a_E, a_S, a_N, b_P, i_t, f_t\right)$

Início do Paralelismo

Privado $\left(P_{ef}, P_{real}, u_P^*\right)$

1. Para P_{ef} no intervalo $\left[i_t, f_t\right]$**;**
   a. Calcular P_{real} com a Eq. (5.6b);
   b. Calcular o termo ponderado com a Eq. (5.17) e $P = P_{real}$;
   c. Calcular $u_{P_{real}}^{(i+1)}$ com a Eq. (5.16) ($u_{P_{real}}^{(i+1)} = \mathbf{u}$ e $u_{P_{real}}^{(i)} = \mathbf{u_0}$**).**

Fim do Paralelismo

Como pode-se observar, o algoritmo 5.5 representa o método de Jacobi ponderado serial e a paralelização consiste em executar o mesmo em cada *thread* independentemente.

5.7 Gauss-Seidel *red-black* em paralelo

Considerando a convenção apresentada na Fig. 2.1, o método de Gauss-Seidel (seção 2.2.2) poder ser escrito, na forma da Eq. (4.6), como

$$u_P = \left(a_W u_W + a_E u_E + a_S u_S + a_N u_N - b_P\right) / a_P \,, \tag{5.18}$$

onde deve-se considerar $P = P_{real}$, ou seja, $W = P_{real} - 1$, $E = P_{real} + 1$, $S = P_{real} + N_x$ e $N = P_{real} - N_x$.

A Fig. 5.8 mostra um exemplo de domínio com 25 nós distribuídos entre 5 *threads*. Os quadrados representam os nós cujos valores são atualizados, de acordo com a Eq. (5.18), pelos *threads* $t = 1, 2, \ldots, 5$. No exemplo da Fig. 5.8, considera-se que a velocidade com que os processadores atualizam os valores dos nós de seus subdomínios é diferente, sendo $t = 1$ o *thread* mais rápido e $t = 4$ o mais lento. Admite-se que os *threads* $t = 2$, 3 e 5 possuem a mesma velocidade. No exemplo considerado, os *threads*

$t = 2$ e $t = 3$ atualizam os valores dos nós 8 e 13, respectivamente. No caso da atualização do nó 8, o *thread* $t = 2$ faz a leitura dos valores contidos nos nós 3, 7, 9 e 13, mas o valor do nó 13 está, ao mesmo tempo, sendo atualizado pelo *thread* $t = 3$, ou seja, o endereço na memória que contém o valor do nó 13 é acessado por dois *threads* e em um dos acessos ($t = 3$) é realizada a operação de escrita. Nesta situação hipotética, o valor do nó 13 lido pelo *thread* $t = 2$ e o valor do nó 8 lido pelo *thread* $t = 3$, estão comprometidos por um problema conhecido como condição de corrida (*race condition*).

Definição 5.1: O problema conhecido como *race condition* ocorre nos casos em que dois (ou mais) processadores acessam o mesmo endereço na memória e pelo menos em um destes acessos escreve-se na memória.

No contexto de uma *race condition*, o valor lido na memória não é confiável (CHAPMAN et al., 2008).

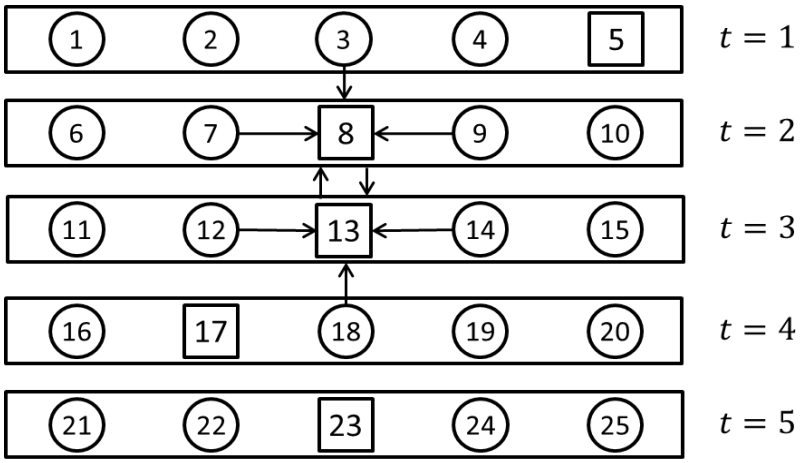


Figura 5.8: Gauss-Seidel lexicográfico com *race condition*.

Race condition é um problema de difícil detecção, pois depende da velocidade e ordem de execução dos *threads*, que são diferentes cada vez que o algoritmo é executado (mesmo para processadores com o mesmo desempenho), e seu impacto nos resultados vai desde nenhum efeito significativo até o comprometimento do algoritmo. Para detectar este tipo de problema, sugere-se executar os algoritmos várias vezes com os mesmos dados de entrada e observar variações nos resultados.

O método de Jacobi é imune ao problema da *race condition* porque utiliza duas estimativas: o vetor estimativa anterior, $\mathbf{u}_0$, no qual é realizado apenas a leitura e o vetor estimativa atual, $\mathbf{u}$, no qual é realizado apenas a escrita.

A sincronização dos acessos, de forma que a leitura ocorra antes (ou depois) da escrita, é uma forma de evitar o problema da *race condition*, mas representa custos adicionais no tempo de processamento. Tal problema pode ser evitado no contexto do método de Gauss-Seidel lexicográfico adaptando-se o domínio de acordo com a Fig. 5.9, onde os nós representados pelos quadrados (*red*) são atualizados, de acordo com a Eq. (5.18), antes (ou depois) dos nós representados pelos círculos (*black*).

A adaptação do domínio em nós *red-black*, apresentada na Fig. 5.9, representa uma superação do problema da *race condition* porque durante a atualização (operação de escrita) de um dos tipos de nós, *red* ou *black*, os outros, *black* ou *red*, são utilizados apenas para leitura, ou seja, mesmo que um endereço de memória seja acessado por dois *threads*, ambos os acessos serão apenas de leitura.

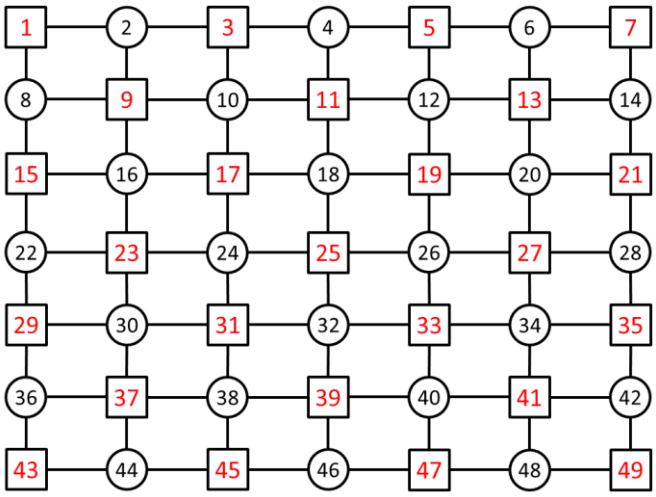


Figura 5.9: Ordenação da malha para Gauss-Seidel *red-black*.

No caso de N_x ser ímpar, a divisão dos subdomínios continua a ser dada pela Eq. (5.10), onde o início (i_red_t) e fim (f_red_t) dos nós *red* podem ser calculados com

$$\begin{aligned} i\_red_t &= i_t + MOD(i_t + 1, 2) \\ f\_red_t &= f_t - MOD(f_t + 1, 2) \end{aligned}, \tag{5.19}$$

enquanto que o início (i_black_t) e fim (f_black_t) dos nós *black* podem ser calculados com

$$\begin{aligned} i\_black_t &= i_t + MOD(i_t, 2) \\ f\_black_t &= f_t - MOD(f_t, 2) \end{aligned}, \tag{5.20}$$

onde i_t e f_t são dados pela Eq. (5.10) e

$$MOD(a,b) = a - \left\lfloor \frac{a}{b} \right\rfloor b . \tag{5.21}$$

É importante observar que os subdomínios continuam sendo delimitados pelos índices i_t e f_t. No entanto, cada subdomínio é composto de nós *red* e nós *black*.

O algoritmo 5.6 é a versão do método Gauss-Seidel *red-black* em paralelo utilizado nesta dissertação. As variáveis de entrada são:

u: vetor contendo a estimativa (com as condições de contorno já atualizadas);

a_P, a_W, a_E, a_S, a_N, b_P: são os coeficientes definidos nas Eqs. (4.7), cujos índices estão de acordo com a Fig. 2.1;

As variáveis de entrada (início e fim), admitem duas possibilidades:

a) i_red_t, f_red_t: Calculados com as Eqs. (5.19). Neste caso, os nós *red* do subdomínio t serão resolvidos utilizando o método Gauss-Seidel;

b) i_black_t, f_black_t: Calculados com as Eqs. (5.20). Neste caso, os nós *black* do subdomínio t serão resolvidos utilizando o método Gauss-Seidel.

Como o algoritmo 5.6 é executado por todos os *threads* simultaneamente, no caso a), todos os nós *red* do domínio serão atualizados; de forma semelhante, no caso b), todos os nós *black* do domínio serão atualizados.

No passo 1 é definido que os índices P_{ef} deverão estar no intervalo fechado $[\text{início}, \text{fim}, 2] = \text{início}, \text{início}+2, \text{início}+4, \text{início}+6, \ldots, \text{fim}$. Dependendo das variáveis de entrada (início e fim), P_{ef} irá "varrer" todos os nós *red* (ou *black*) dentro de um subdomínio t. Como cada *thread* tem seu próprio subdomínio $[i_t, f_t]$ (composto por nós *red* e *black*), nota-se que a paralelização está sendo realizada em relação aos índices

P_{ef} (ordem lexicográfica efetiva) e a Eq. (5.18) está sendo resolvida em relação aos índices P_{real} (ordem lexicográfica real).

Algoritmo 5.6: Gauss-Seidel *red-black* em paralelo.

Gauss-Seidel *red-black* em paralelo

Entrada $\left(\mathbf{u}, a_P, a_W, a_E, a_S, a_N, b_P, \text{início}, \text{fim}\right)$

Início do Paralelismo

Privado $\left(P_{ef}, P_{real}\right)$

1. Para P_{ef} no intervalo $\left[\text{início}, \text{fim}, 2\right]$**;**
    a. Calcular P_{real} com a Eq. (5.6b)**;**
    b. Calcular $u_{P_{real}}$ com a Eq. (5.18)**.**

Fim do Paralelismo

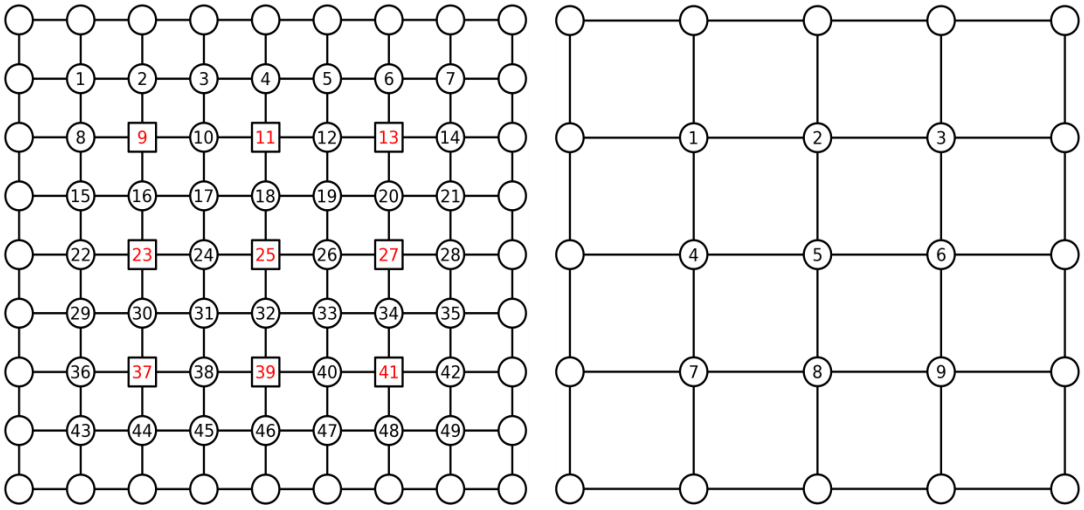
5.8 Restrição em paralelo

O processo de restrição, discutido na seção 3.3, pode ser paralelizado utilizando os conceitos desenvolvidos nas seções anteriores: particiona-se a malha grossa em subdomínios e, para cada um destes em paralelo, transfere-se a informação necessária da malha fina.

A restrição é imune ao problema da *race condition*, pois se trata de um processo de transferência de informação (leitura) da malha fina para malha grossa (onde se realiza a escrita).

Como a paralelização será realizada em relação aos nós da malha grossa é importante encontrar uma relação entre estes e os nós da malha fina. No exemplo da Fig. 5.10 (a), cuja razão de engrossamento é $r = 2$, os quadrados representam os nós da malha fina que correspondem exatamente aos nós da malha grossa de forma que, para este exemplo da Fig. 5.10, tem-se a relação

$$
\begin{aligned}
P_{ef}^{2h} = 1 &\Rightarrow P_{ef}^{h} = \ 9, \\
P_{ef}^{2h} = 2 &\Rightarrow P_{ef}^{h} = 11, \\
&\vdots \\
P_{ef}^{2h} = 9 &\Rightarrow P_{ef}^{h} = 41.
\end{aligned}
$$



a) Malha fina b) Malha grossa

Figura 5.10: Relação entre nós efetivos de uma malha fina e malha grossa para razão de engrossamento $r = 2$.

Substituindo as expressões, $j_{ef}^{h} = 2\, j_{ef}^{2h}$, $i_{ef}^{h} = 2 i_{ef}^{2h}$ e $N_x^h = 2N_x^{2h} - 1$ (válidas para $r = 2$ como indicado na Fig. 5.10), no sistema

$$
\begin{cases}
P_{ef}^{h} = i_{ef}^{h} + \left(j_{ef}^{h} - 1 \right)\left(N_x^h - 2 \right) \\
P_{ef}^{2h} = i_{ef}^{2h} + \left(j_{ef}^{2h} - 1 \right)\left(N_x^{2h} - 2 \right)
\end{cases},
$$

e resolvendo para P_{ef}^{h}, pode-se mostrar que a relação entre os nós efetivos da malha fina (P_{ef}^{h}) e da malha grossa (P_{ef}^{2h}) é dada por

$$
P_{ef}^{h} = 2P_{ef}^{2h} + 2\left(\left\lfloor \frac{P_{ef}^{2h} - 1}{N_x^{2h} - 2} \right\rfloor + 1 \right)\left(N_x^{2h} - 1 \right) - 1 . \tag{5.22}
$$

A Eq. (3.3), que corresponde ao operador de restrição por ponderação completa, pode ser reescrita na forma lexicográfica como

$$v^{2h}_{P^{2h}_{real}} = \Big(v^{h}_{P^{h}_{real}-N^{h}_{x}-1} + 2v^{h}_{P^{h}_{real}-N^{h}_{x}} + v^{h}_{P^{h}_{real}-N^{h}_{x}+1} + 2v^{h}_{P^{h}_{real}-1} + 4v^{h}_{P^{h}_{real}} + 2v^{h}_{P^{h}_{real}+1} + v^{h}_{P^{h}_{real}+N^{h}_{x}-1} + v^{h}_{P^{h}_{real}+N^{h}_{x}} + v^{h}_{P^{h}_{real}+N^{h}_{x}+1} \Big) \Big/ 16 \quad . \tag{5.23}$$

Com as Eqs. (5.22) e (5.23), o algoritmo 5.7, que deve ser executado em todos os *threads*, é capaz de realizar a restrição em paralelo.

As variáveis de entrada são:

$\mathbf{f}^{h}$: vetor contendo o resíduo calculado na malha fina;

$\mathbf{f}^{2h}$: vetor da malha grossa que receberá o valor do resíduo restrito da malha fina;

N^{h}_{x}, N^{2h}_{x}: o número de nós na direção coordenada x da malha fina e grossa, respectivamente;

i^{2h}_{t}, f^{2h}_{t}: início e fim do subdomínio na malha grossa pertencente ao *thread* t, calculados com a Eq. (5.10).

Algoritmo 5.7: Restrição em paralelo.

Restrição em paralelo

Entrada $\left(\mathbf{f}^{h}, \mathbf{f}^{2h}, N^{h}_{x}, N^{2h}_{x}, i^{2h}_{t}, f^{2h}_{t}\right)$

Início do Paralelismo

Privado $\left(P^{h}_{ef}, P^{h}_{real}, P^{2h}_{ef}, P^{2h}_{real}\right)$

1. Para P^{2h}_{ef} no intervalo $\left[i^{2h}_{t}, f^{2h}_{t}\right]$**;**
    a. Calcular P^{h}_{ef} com a Eq. (5.22);
    b. Calcular P^{h}_{real} com a Eq. (5.6b) e $N_x = N^{h}_{x}$;
    c. Calcular P^{2h}_{real} com a Eq. (5.6b) e $N_x = N^{2h}_{x}$;
    d. Calcular $\mathbf{f}^{2h}_{P^{2h}_{real}}$ com a Eq. (5.23).

Fim do Paralelismo

No passo 1 é definido que os índices P^{2h}_{ef} deverão estar no intervalo fechado $\left[i^{2h}_{t}, f^{2h}_{t}\right]$, ou seja, P^{2h}_{ef} irá "varrer" todos os nós dentro de um subdomínio t (na malha grossa). No passo 1.a, os índices P^{2h}_{ef} (malha grossa) são "convertidos" para P^{h}_{ef} (malha

fina). Como a Eq. (5.23) é resolvida em relação à ordem lexicográfica real, nos passos 1.b e 1.c, os índices P_{ef}^{2h} e P_{ef}^{h} são "convertidos" para P_{real}^{2h} e P_{real}^{h}, respectivamente. No passo 1.d, o operador de restrição por ponderação completa, Eq. (5.23), transfere a informação da malha fina para a grossa.

5.9 Prolongação em paralelo

O processo de prolongação, discutido na seção 3.4, pode ser paralelizado utilizando os conceitos desenvolvidos nas seções anteriores: particiona-se a malha fina em subdomínios e, para cada um destes em paralelo, transfere-se a informação necessária da malha grossa.

Assim como a restrição, a prolongação é imune ao problema da *race condition*, pois se trata de um processo de transferência de informação (leitura) da malha grossa para malha fina (onde se realiza a escrita).

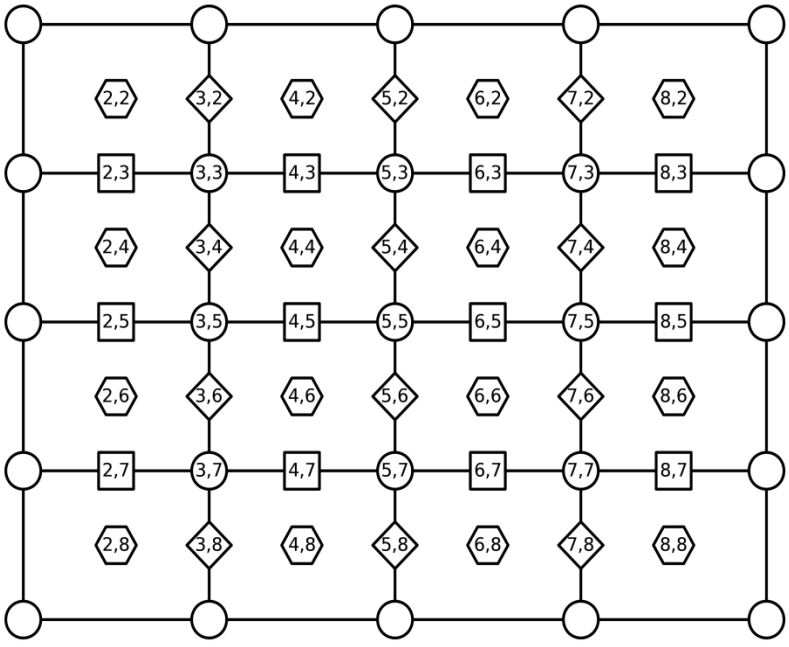


Figura 5.11: Malha fina sobreposta a uma malha grossa para razão de engrossamento $r = 2$.

A Fig. 5.11 mostra o interior de uma malha com $N = 9 \times 9$ nós, sobreposta à malha imediatamente mais grossa, com razão de engrossamento $r = 2$, com $N = 5 \times 5$ nós. Os nós da malha grossa são representados pelos círculos. Os nós da malha fina são representados por hexágonos, losangos, quadrados e círculos, ou seja, os círculos

pertencem às duas malhas e indicam que os nós destas se sobrepõem exatamente. Os números dentro dos nós da malha fina representam as coordenadas i_{real} e j_{real}, conforme a convenção apresentada na Fig. 2.1.

De acordo com a Fig. 5.11, as Eqs (3.4) podem ser reescritas na forma lexicográfica da seguinte forma

$$
\begin{cases}
\textbf{Círculos} \\
v^{h}_{P^{h}_{real}} = v^{2h}_{P^{2h}_{real}} \\
i^{h}_{real} = \text{ímpar e } j^{h}_{real} = \text{ímpar}
\end{cases}
$$

$$
\begin{cases}
\textbf{Losangos} \\
v^{h}_{P^{h}_{real}} = \dfrac{1}{2}\left(v^{2h}_{P^{2h}_{real}} + v^{2h}_{P^{2h}_{real}+N^{2h}_{x}} \right) \\
i^{h}_{real} = \text{ímpar e } j^{h}_{real} = \text{par}
\end{cases}
$$

$$
\begin{cases}
\textbf{Quadrados} \\
v^{h}_{P^{h}_{real}} = \dfrac{1}{2}\left(v^{2h}_{P^{2h}_{real}} + v^{2h}_{P^{2h}_{real}+1} \right) \\
i^{h}_{real} = \text{par e } j^{h}_{real} = \text{ímpar}
\end{cases}
$$

$$
\begin{cases}
\textbf{Hexágonos} \\
v^{h}_{P^{h}_{real}} = \dfrac{1}{4}\left(v^{2h}_{P^{2h}_{real}} + v^{2h}_{P^{2h}_{real}+1} + v^{2h}_{P^{2h}_{real}+N^{2h}_{x}} + v^{2h}_{P^{2h}_{real}+N^{2h}_{x}+1} \right) \\
i^{h}_{real} = \text{par e } j^{h}_{real} = \text{par}
\end{cases}
\qquad (5.24)
$$

onde

$$
j_{real} = \left\lfloor \frac{P^{h}_{real} - 1}{N^{h}_{x}} \right\rfloor + 1, \qquad (5.25)
$$

e i_{real} pode ser calculado com a Eq. (5.1).

Com as Eqs. (5.24) o algoritmo 5.8, que deve ser executado em todos os *threads*, é capaz de realizar o processo de prolongação em paralelo.

As variáveis de entrada são:

$\mathbf{u}^h$: vetor da malha fina que receberá a estimativa prolongada da malha grossa;

$\mathbf{u}^{2h}$: vetor contendo a estimativa na malha grossa;

N_x^h, N_x^{2h}: o número de nós na direção coordenada x da malha fina e grossa, respectivamente;

i_t^h, f_t^h: início e fim do subdomínio na malha fina pertencente ao *thread* t, calculados com a Eq. (5.10).

Algoritmo 5.8: Prolongação em paralelo.

Prolongação em paralelo

Entrada $\left(\mathbf{u}^h, \mathbf{u}^{2h}, N_x^h, N_x^{2h}, i_t^h, f_t^h\right)$

Início do Paralelismo

Privado $\left(P_{ef}^h, P_{real}^h, P_{ef}^{2h}, P_{real}^{2h}, i_{real}^h, j_{real}^h, i_{real}^{2h}, j_{real}^{2h}\right)$

1. Para P_{ef}^h no intervalo $\left[i_t^h, f_t^h\right]$**;**
    a. Calcular P_{real}^h com a Eq. (5.6b) e $N_x = N_x^h$;
    b. Calcular j_{real}^h com a Eq. (5.25);
    c. Calcular i_{real}^h com a Eq. (5.1);
    d. Determinar se i_{real}^h, j_{real}^h são pares ou ímpares;
    e. Para o caso apropriado (i_{real}^h, j_{real}^h par e/ou ímpar);
        i. Calcular i_{real}^{2h} e j_{real}^{2h} com

$$i_{real}^{2h} = i_{real}^h - \left\lfloor \frac{i_{real}^h}{2} \right\rfloor \text{ e } j_{real}^{2h} = j_{real}^h - \left\lfloor \frac{j_{real}^h}{2} \right\rfloor;$$

        ii. Calcular P_{real}^{2h} com a Eq. (5.1) e $N_x = N_x^{2h}$;
        iii. Calcular $\mathbf{u}_{P_{real}^h}^h$ com a Eq. (5.24).

Fim do Paralelismo

5.10 *Multigrid* em paralelo

Nas seções anteriores mostrou-se como processar em paralelo diferentes componentes do *multigrid*. Nesta seção será apresentado o método *multigrid* com tais componentes algorítmicas em paralelo.

Para os algoritmos 5.9 e 5.11, são consideradas apenas malhas com $N = N_x N_y$ nós e $N_x = N_y$, onde N_x e N_y são os números de nós nas direções coordenadas x e y, respectivamente. A razão de engrossamento adotada é $r = 2$. Nestes algoritmos, o método *multigrid* parte da malha mais fina, nível 1, e vai até a malha mais grossa, $N = 3 \times 3$ nós.

As variáveis de entrada, em ambos os algoritmos, são:

u: vetor contendo a estimativa inicial (com as condições de contorno já atualizadas);

f : vetor contendo os termos independentes;

N_x, N_y: o número de nós da malha (incluindo os contornos);

L_x, L_y: o comprimento do domínio nas direções coordenadas x e y, respectivamente. Como definido no capítulo 4, as simulações foram realizadas utilizando $L_x = L_y = 1$;

$\text{ciclos}_{\max}$: número máximo de ciclos V (iterações externas);

ν_1 : número de iterações na pré-suavização;

ν_2 : número de iterações na pós-suavização;

ε : tolerância admitida;

$\mathbf{L}_2$: vetor utilizado para armazenar os valores da norma euclidiana do resíduo, onde $L_2^{(i)}$ representa o valor da norma euclidiana na iteração (i) e $L_2^{(0)}$ a norma euclidiana na estimativa inicial. O critério de parada, Eq. (4.9), pode ser reescrito então como

$$\frac{L_2^{(i)}}{L_2^{(0)}} \leq \varepsilon ; \tag{5.26}$$

Procs: número de processadores que serão utilizados no processamento paralelo.

Algoritmo 5.9: *Multigrid* utilizando o *solver* Gauss-Seidel *red-black* em paralelo.

***Multigrid* + *solver* Gauss-Seidel *red-black* em paralelo**

Entrada $\left(\mathbf{u},\mathbf{f},L_x,L_y,N_x,N_y,\text{ciclos}_{\max},\nu_1,\nu_2,\varepsilon,\mathbf{L}_2,Procs\right)$

1. Obter o número de *threads* $\bar{T}$ **;**

   Se $0<Procs\le\bar{T}$ então $\bar{T}=Procs$ **;**

   Definir o número de *threads*, utilizadas pela máquina, como $\bar{T}$ **;**

2. Calcular o número de níveis $L_{\max}$ com a Eq. (4.10);

Início do Paralelismo

Privado $\left(t,C^{\text{lv}},i_t^{\text{lv}},f_t^{\text{lv}},l_{1t},l_{2t},l_t,l,\text{lv},\text{ciclo}\right)$

3. Obter o número t que define o *thread***;**
4. Para lv no intervalo $\left[1,L_{\max}\right]$**;**
   a. Em apenas um *thread***;**

      Calcular N_x^{lv} e N_y^{lv} com a Eq. (4.11)**;**

      Calcular N_{sub}^{lv} com a Eq. (5.7) utilizando N_x^{lv} e N_y^{lv} **;**

      Calcular t_c^{lv} com a Eq. (5.8) utilizando N_x^{lv}, N_y^{lv} e N_{sub}^{lv} **;**

      Calcular os coeficientes a_P^{lv}, a_W^{lv}, a_E^{lv}, a_S^{lv}, a_N^{lv} com as Eqs. (4.7) e utilizando N_x^{lv} e N_y^{lv} na Eq. (1.1)**;**

   b. Calcular C_t^{lv} com a Eq. (5.9)**;**

   c. Calcular os índices que definem os subdomínios para cada nível (i_t^{lv} e f_t^{lv}) com a Eq. (5.10) utilizando N_{sub}^{lv} e C_t^{lv} **;**

      Calcular os índices que definem os subdomínios *red* ($i\_red_t^{\text{lv}}$ e $f\_red_t^{\text{lv}}$) com a Eq. (5.19) utilizando i_t^{lv} e f_t^{lv} **;**

      Calcular os índices que definem os subdomínios *black* ($i\_black_t^{\text{lv}}$ e $f\_black_t^{\text{lv}}$) com a Eq. (5.20) utilizando i_t^{lv} e f_t^{lv} **;**

5. Obter as condições l_{1t} e l_{2t} (ou apenas l_{2t})**;**

   Calcular l_t utilizando a Eq. (5.15b)**;**

6. Para ciclo no intervalo $\left[1,\text{ciclos}_{\max}\right]$**;**
   a. Calcular o resíduo utilizando o algoritmo 5.1**;**

b. Calcular os termos r_t' (vetor $\mathbf{r}'$) com o algoritmo 5.2**;**

c. **Barreira (espera que todos os *threads* atinjam este ponto);**

d. Para l (linha) no intervalo $[1, l_t]$ (se $l_t = 0$ vá para o passo e)**;**

   i. Calcular o termo: $r_t' = r_t' + r'_{t+2^{l-1}}$ **;**

   ii. **Barreira (espera que todos os *threads* atinjam este ponto);**

e. Em apenas um *thread* atualizar $L_2^{(i)} = \sqrt{r_1'}$ (r_t' do processador 1)**;**

f. Verificar se a Eq. (5.26) é satisfeita (se sim então o algoritmo é encerrado)**;**

g. Para lv no intervalo $[1, L_{\max} - 1]$**;**

   i. Atualizar o coeficiente $b_P^{\text{lv}} = f_P^{\text{lv}}$ $(\mathbf{b} = \mathbf{f})$, ou seja, $b_P^{\text{lv}} = 0$ (Laplace) para lv=1 e $b_P^{\text{lv}} = \text{resíduo}$ para $\text{lv} > 1$**;**

   ii. **Barreira (espera que todos os *threads* atinjam este ponto);**

   iii. Suavizar ν_1 vezes (pré-suavização)**;**

      1. Utilizar o algoritmo 5.6 com $i\_red_t^{\text{lv}}$ e $f\_red_t^{\text{lv}}$**;**

      2. Barreira (espera que todos os *threads* atinjam este ponto);

      3. Utilizar o algoritmo 5.6 com $i\_black_t^{\text{lv}}$ e $f\_black_t^{\text{lv}}$**;**

      4. Barreira (espera que todos os *threads* atinjam este ponto);

   iv. Calcular o resíduo utilizando o algoritmo 5.1**;**

   v. **Barreira (espera que todos os *threads* atinjam este ponto);**

   vi. Fazer a restrição do vetor resíduo, $\mathbf{f}$, com o algoritmo 5.7**;**

   vii. **Barreira (espera que todos os *threads* atinjam este ponto);**

h. Para $\text{lv} = L_{\max}$ **;**

   i. Atualizar o coeficiente $b_P^{\text{lv}} = f_P^{\text{lv}}$ $(\mathbf{b} = \mathbf{f})$**;**

   ii. **Barreira (espera que todos os *threads* atinjam este ponto);**

   iii. Suavizar ν_1 vezes (pré-suavização)**;**

      1. Utilizar o algoritmo 5.6 com $i\_red_t^{\text{lv}}$ e $f\_red_t^{\text{lv}}$**;**

      OBS: Não tem nós *black* nesta malha ($N = 3 \times 3$).

      2. Barreira (espera que todos os *threads* atinjam este

ponto);

i. Para lv no intervalo $[L, 2]$**;**

i. Prolongar o vetor estimativa, $\mathbf{u}$, com o algoritmo 5.8**;**

ii. **Barreira (espera que todos os *threads* atinjam este ponto);**

iii. Suavizar ν_2 vezes (pós-suavização)**;**

1. Utilizar o algoritmo 5.6 com $i\_red_t^{lv}$ e $f\_red_t^{lv}$**;**

2. Barreira (espera que todos os *threads* atinjam este ponto);

3. Utilizar o algoritmo 5.6 com $i\_black_t^{lv}$ e $f\_black_t^{lv}$**;**

4. Barreira (espera que todos os *threads* atinjam este ponto).

Fim do Paralelismo

O algoritmo Atualiza Anterior, a seguir, demonstra como o algoritmo 5.11, *Multigrid* + *solver* Jacobi ponderado, atualiza o vetor estimativa anterior ($\mathbf{u}_0$).

As variáveis de entrada são:

$\mathbf{u}$: vetor estimativa;

$\mathbf{u}_0$: vetor com a estimativa de uma iteração anterior;

N_x: o número de nós da malha na direção coordenada x (incluindo os contornos);

i_t, f_t: início e fim do subdomínio de t, calculados com a Eq. (5.10).

Algoritmo 5.10: Atualiza Anterior.

Atualiza Anterior

Entrada $(\mathbf{u}, \mathbf{u}_0, N_x, i_t, f_t)$

Início do Paralelismo

Privado (P_{ef}, P_{real})

2. Para P_{ef} no intervalo $[i_t, f_t]$**;**

c. Calcular P_{real} com a Eq. (5.6b);

d. Fazer $u_{P_{real}}^{(i)} = u_{P_{real}}^{(i+1)}$, onde ($u_{P_{real}}^{(i+1)} = \mathbf{u}$ e $u_{P_{real}}^{(i)} = \mathbf{u}_0$)**.**

Fim do Paralelismo

O algoritmo 5.11 apresenta o método *multigrid* utilizando o *solver* Jacobi ponderado. As variáveis de entrada são as mesmas do algoritmo 5.9.

Algoritmo 5.11: *Multigrid* utilizando o *solver* Jacobi ponderado em paralelo.

***Multigrid* + *solver* Jacobi ponderado em paralelo**

Entrada $\left(\mathbf{u}, \mathbf{f}, L_x, L_y, N_x, N_y, \text{ciclos}_{\max}, \nu_1, \nu_2, \varepsilon, \mathbf{L}_2, Procs\right)$

1. Obter o número de *threads* $\bar{T}$ **;**

   Se $0 < Procs \leq \bar{T}$ então $\bar{T} = Procs$ **;**

   Definir o número de *threads*, utilizadas pela máquina, como $\bar{T}$ **;**

2. Calcular o número de níveis $L_{\max}$ com a Eq. (4.10);

   Inicializar e atualizar as condições de contorno do vetor estimativa anterior $\left(\mathbf{u}_0\right)$.

Início do Paralelismo

Privado $\left(t, C^{\text{lv}}, i_t^{\text{lv}}, f_t^{\text{lv}}, l_{1t}, l_{2t}, l_t, l, \text{lv}, \text{ciclo}\right)$

3. Obter o número t que define o *thread***;**
4. Para lv no intervalo $\left[1, L_{\max}\right]$**;**
   a. Em apenas um *thread***;**

      Calcular N_x^{lv} e N_y^{lv} com a Eq. (4.11)**;**

      Calcular N_{sub}^{lv} com a Eq. (5.7) utilizando N_x^{lv} e N_y^{lv} **;**

      Calcular t_c^{lv} com a Eq. (5.8) utilizando N_x^{lv}, N_y^{lv} e N_{sub}^{lv} **;**

      Calcular os coeficientes a_P^{lv}, a_W^{lv}, a_E^{lv}, a_S^{lv}, a_N^{lv} com as Eqs. (4.7) e utilizando N_x^{lv} e N_y^{lv} na Eq. (1.1)**;**

   b. Calcular C_t^{lv} com a Eq. (5.9)**;**

   c. Calcular os índices que definem os subdomínios para cada nível (i_t^{lv} e f_t^{lv}) com a Eq. (5.10) utilizando N_{sub}^{lv} e C_t^{lv}**;**

5. Obter as condições l_{1t} e l_{2t} (ou apenas l_{2t})**;**

   Calcular l_t utilizando a Eq. (5.15b)**;**

6. Para ciclo no intervalo $\left[1, \text{ciclos}_{\max}\right]$**;**

---

a. Calcular o resíduo utilizando o algoritmo 5.1**;**

b. Calcular os termos r_t' (vetor $\mathbf{r}'$) com o algoritmo 5.2**;**

c. **Barreira (espera que todos os *threads* atinjam este ponto);**

d. Para l (linha) no intervalo $[1, l_t]$ (se $l_t = 0$ vá para o passo e)**;**

   i. Calcular o termo: $r_t' = r_t' + r'_{t+2^{l-1}}$ **;**

   ii. **Barreira (espera que todos os *threads* atinjam este ponto);**

e. Em apenas um *thread* atualizar $L_2^{(i)} = \sqrt{r_1'}$ (r_t' do processador 1)**;**

f. Verificar se a Eq. (5.26) é satisfeita (se sim então o algoritmo é encerrado)**;**

g. Para lv no intervalo $[1, L_{\max} - 1]$**;**

   i. Atualizar o coeficiente $b_P^{\text{lv}} = f_P^{\text{lv}}$ $(\mathbf{b} = \mathbf{f})$, ou seja, $b_P^{\text{lv}} = 0$ (Laplace) para $\text{lv} = 1$ e $b_P^{\text{lv}} = \text{resíduo}$ para $\text{lv} > 1$**;**

   ii. **Barreira (espera que todos os *threads* atinjam este ponto);**

   iii. Suavizar ν_1 vezes (pré-suavização)**;**

      1. Atualizar o vetor estimativa anterior utilizando o algoritmo 5.10**;**
      2. **Barreira (espera que todos os *threads* atinjam este ponto);**
      3. Utilizar o algoritmo 5.5 (Jacobi ponderado)**;**
      4. **Barreira (espera que todos os *threads* atinjam este ponto);**

   iv. Calcular o resíduo utilizando o algoritmo 5.1**;**

   v. **Barreira (espera que todos os *threads* atinjam este ponto);**

   vi. Fazer a restrição do vetor resíduo, $\mathbf{f}$, com o algoritmo 5.7**;**

   vii. **Barreira (espera que todos os *threads* atinjam este ponto);**

h. Para $\text{lv} = L_{\max}$ **;**

   i. Atualizar o coeficiente $b_P^{\text{lv}} = f_P^{\text{lv}}$ $(\mathbf{b} = \mathbf{f})$**;**

   ii. **Barreira (espera que todos os *threads* atinjam este ponto);**

   iii. Para ν_1 vezes (pré-suavização)**;**

      1. Atualizar o vetor estimativa anterior utilizando o

algoritmo 5.10**;**

2. **Barreira (espera que todos os *threads* atinjam este ponto);**
3. Utilizar o algoritmo 5.5**;**
4. **Barreira (espera que todos os *threads* atinjam este ponto);**

i. Para lv no intervalo $[L, 2]$**;**

i. Prolongar o vetor estimativa, **u** , com o algoritmo 5.8**;**

ii. **Barreira (espera que todos os *threads* atinjam este ponto);**

iii. Suavizar ν_2 vezes (pós-suavização)**;**

1. Atualizar o vetor estimativa anterior utilizando o algoritmo 5.10**;**
2. **Barreira (espera que todos os *threads* atinjam este ponto);**
3. Utilizar o algoritmo 5.5**;**
4. **Barreira (espera que todos os *threads* atinjam este ponto).**

Fim do Paralelismo

6 Resultados

Neste capítulo são apresentados os resultados da paralelização do método *multigrid* utilizando a metodologia proposta no capítulo 5. Nesta seção o termo *thread* será considerado sinônimo de processador.

Os resultados apresentados referem-se ao método *multigrid* com todas as suas componentes paralelizadas, ou seja:

1) Os métodos iterativos paralelizados de acordo com as seções 5.6 e 5.7;
2) O cálculo do resíduo paralelizado de acordo com a seção 5.4;
3) O processo de restrição paralelizado de acordo com a seção 5.8;
4) O processo de prolongação paralelizado de acordo com a seção 5.9.

Os detalhes de implementação (parâmetros utilizados no método *multigrid*) podem ser consultados no capítulo 4.

Os parâmetros empregados para a avaliação do método *multigrid* são:

1) A norma-infinito do erro numérico ($\|\mathbf{e}\|_\infty$), Eq. (2.27), utilizada para validar a consistência da solução numérica obtida, comparando esta com a solução analítica conhecida;
2) A norma euclidiana do resíduo adimensionalizada pela norma do resíduo na estimativa inicial $\left(L_2^{(i)} / L_2^{(0)}\right)$, Eq. (5.26), utilizada como critério de parada do algoritmo. Representa uma medida do quanto o resíduo diminuiu em relação à estimativa inicial;
3) Número de ciclos V: indica quantas iterações externas o método *multigrid* realiza até que o critério de parada seja atingido.

Os parâmetros utilizados para avaliar a paralelização são:

a) O fator *speed-up* $\left(S_n\right)$, Eq. (2.49b), o qual indica o quanto a paralelização "acelera" o algoritmo em relação à sua versão serial;
b) Eficiência $\left(E\right)$, Eq. (2.53), que indica a efetividade com que os processadores são utilizados.

6.1 Paralelização do método *multigrid*

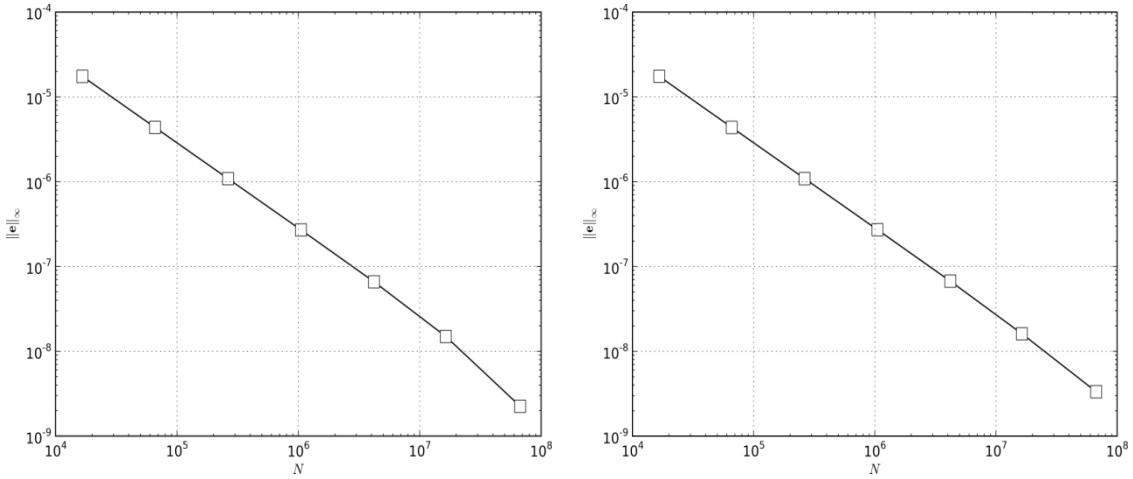
Os parâmetros $\|\mathbf{e}\|_\infty$, $L_2^{(i)}/L_2^{(0)}$ e Ciclos V são independentes do número de processadores utilizados, pois são calculados considerando a solução numérica após o critério de parada ser atingido. Esta independência do número de processadores é esperada visto que a solução numérica obtida paralelamente deve ser igual à solução numérica obtida pela versão serial do método *multigrid*.

Na Tab. 6.1 são apresentadas as malhas utilizadas $\left(N = N_x N_y \text{ e } N_x = N_y\right)$ nos testes; a norma-infinito do erro numérico $\left(\|\mathbf{e}\|_\infty\right)$; a norma euclidiana do resíduo adimensionalizada pela norma do resíduo na estimativa inicial $\left(L_2^{(i)}/L_2^{(0)}\right)$ e o número de ciclos V empregados pelo *multigrid* para atingir o critério de parada.

Tabela 6.1: Parâmetros de avaliação do método *multigrid*.

N	$\|\mathbf{e}\|_\infty$	$L_2^{(i)}/L_2^{(0)}$	Ciclos V
Gauss-Seidel *red-black*			
129×129	1,741E-05	3,789E-11	6
257×257	4,351E-06	1,964E-11	6
513×513	1,086E-06	1,536E-11	6
1025×1025	2,700E-07	1,461E-11	6
2049×2049	6,598E-08	1,449E-11	6
4097×4097	1,497E-08	1,447E-11	6
8193×8193	2,248E-09	1,446E-11	6
Jacobi ponderado			
129×129	1,741E-05	7,424E-11	11
257×257	4,352E-06	7,255E-11	11
513×513	1,087E-06	7,216E-11	11
1025×1025	2,711E-07	7,206E-11	11
2049×2049	6,710E-08	7,204E-11	11
4097×4097	1,609E-08	7,203E-11	11
8193×8193	3,337E-09	7,203E-11	11

Pode-se observar na Tab. 6.1 que a norma euclidiana do resíduo adimensionalizada pela norma do resíduo na estimativa inicial satisfaz o critério de parada $\left(\leq \varepsilon = 10^{-10}\right)$. Observa-se ainda que o número de ciclos V necessários para atingir o critério de parada não varia com o refinamento da malha.



a) Gauss-Seidel *red-black* b) Jacobi ponderado

Figura 6.1: Norma infinito do erro numérico para o método *multigrid* utilizando os métodos Gauss-Seidel *red-black* e Jacobi ponderado.

A Fig. 6.1 mostra como a norma infinito do erro numérico decai com o aumento do número de incógnitas $\left(N\right)$ do problema, ou seja, o erro de discretização diminui com o refinamento da malha. Os resultados são consistentes com os obtidos por Oliveira (2010).

A Tab. 6.2 resume os resultados obtidos com a paralelização do método *multigrid* utilizando os métodos de Jacobi ponderado e Gauss-Seidel *red-black* (no apêndice B encontram-se os resultados para 55 *threads*, ou processadores lógicos, Tabs. B.1 e B.2). Nela são apresentadas as malhas utilizadas $\left(N = N_x N_y \text{ e } N_x = N_y\right)$, o número de *threads* (1, 2, 4, 8, 16, 32) empregados em paralelo, o tempo de CPU $\left(t_{CPU}\right)$ em segundos e o *speed-up* $\left(S_n\right)$ obtido pelo processamento paralelo do método *multigrid*. As tendências apontadas a seguir são resultado do estudo da Tab. 6.2, Tabs. B.1 e B.2.

Na Tab. 6.2 observa-se a tendência geral dos valores de *speed-up* aumentarem com o número de processadores, ou seja, quanto maior o número de processadores maior o *speed-up* para cada malha. De forma geral, observa-se ainda que para um número fixo de processadores, o valor do *speed-up* aumenta com o refinamento da malha. Este

resultado sugere que o tempo que o método *multigrid* utiliza realizando processamento paralelo aumenta com o refinamento das malhas a uma taxa maior que o tempo perdido no processamento sequencial (que não pode ser melhorado). Considerando-se as duas tendências anteriores espera-se que o maior valor de *speed-up* seja atingido na malha mais refinada possível (limitada pela memória do computador) utilizando todos os processadores disponíveis, o que é uma propriedade desejável.

Tabela 6.2: Tempo de CPU e *Speed-up* obtidos com a paralelização do método multigrid.

CPUs	1		2		4		8		16		32	
N	$t_{CPU}(s)$	S_n	$t_{CPU}(s)$	S_n	$t_{CPU}(s)$	S_n	$t_{CPU}(s)$	S_n	$t_{CPU}(s)$	S_n	$t_{CPU}(s)$	S_n
Gauss-Seidel *red-black*												
129^2	0,105	1	0,066	1,6	0,037	2,8	0,031	3,4	0,029	3,6	0,036	2,9
257^2	0,421	1	0,246	1,7	0,132	3,2	0,094	4,5	0,084	5,0	0,087	4,8
513^2	0,969	1	0,734	1,3	0,563	1,7	0,403	2,4	0,273	3,5	0,212	4,6
1025^2	3,817	1	2,658	1,4	1,542	2,5	1,301	2,9	0,908	4,2	0,599	6,4
2049^2	15,39	1	9,190	1,7	6,484	2,4	5,156	3,0	3,146	4,9	2,021	7,6
4097^2	63,42	1	36,83	1,7	23,02	2,8	19,14	3,3	12,30	5,2	7,235	8,8
8193^2	260,6	1	144,6	1,8	89,80	2,9	68,21	3,8	42,94	6,1	26,27	9,9
Jacobi ponderado												
129^2	0,296	1	0,169	1,8	0,096	3,1	0,066	4,5	0,053	5,6	0,063	4,7
257^2	0,853	1	0,555	1,5	0,344	2,5	0,229	3,7	0,148	5,8	0,107	8,0
513^2	2,826	1	1,774	1,6	1,014	2,8	0,856	3,3	0,560	5,0	0,353	8,0
1025^2	12,57	1	7,523	1,7	4,699	2,7	2,916	4,3	1,923	6,5	1,111	11,3
2049^2	45,58	1	28,40	1,6	17,92	2,5	11,57	3,9	7,518	6,1	4,270	10,7
4097^2	181,5	1	114,1	1,6	64,82	2,8	40,78	4,5	26,20	6,9	15,87	11,4
8193^2	759,1	1	429,0	1,8	225,2	3,4	152,3	5,0	92,96	8,2	57,78	13,1

Na Tab. 6.3 pode-se ver que a eficiência do paralelismo apresenta a tendência de aumentar com o tamanho da malha e diminuir com o número de processadores. Considerando estas duas tendências, conclui-se que utilizar muitos processadores em malhas pouco refinadas pode resultar em baixa eficiência da paralelização. Uma das razões pela qual a eficiência diminui com a adição de processadores no sistema é o fato de que estes, apesar de contribuírem no processamento, introduzem custos computacionais extras, como o aumento do número de acessos à memória, custos de

sincronização dos *threads*, entre outros. Desta forma, os elementos de processamento não podem dedicar todo o seu tempo processando o algoritmo (GRAMA et al., 2003).

Tabela 6.3: Eficiência da paralelização do método *multigrid*.

CPUs	1	2	4	8	16	32
N	Eficiência E (%)					
Jacobi ponderado						
129×129	100,0	87,6	77,1	56,1	34,9	14,7
257×257	100,0	76,8	62,0	46,6	36,0	24,9
513×513	100,0	79,7	69,7	41,3	31,5	25,0
1025×1025	100,0	83,5	66,9	53,9	40,9	35,4
2049×2049	100,0	80,2	63,6	49,2	37,9	33,4
4097×4097	100,0	79,5	70,0	55,6	43,3	35,7
8193×8193	100,0	88,5	84,3	62,3	51,0	41,1
Gauss-Seidel *red-black*						
129×129	100,0	79,5	70,9	42,3	22,6	9,1
257×257	100,0	85,6	79,7	56,0	31,3	15,1
513×513	100,0	66,0	43,0	30,1	22,2	14,3
1025×1025	100,0	71,8	61,9	36,7	26,3	19,9
2049×2049	100,0	83,7	59,3	37,3	30,6	23,8
4097×4097	100,0	86,1	68,9	41,4	32,2	27,4
8193×8193	100,0	90,1	72,6	47,8	37,9	31,0

6.2 Gauss-Seidel *red-black versus* Jacobi ponderado

Nesta seção são realizadas comparações entre o desempenho do método *multigrid* utilizando os *solvers* Gauss-Seidel *red-black* e Jacobi ponderado. Pode-se apontar como objetivo dos testes:

- Qual estratégia apresenta melhor desempenho (menor tempo de CPU);
- Qual estratégia beneficia-se mais da metodologia de paralelização (maiores valores de *speed-up* e melhor eficiência no uso dos *threads*).

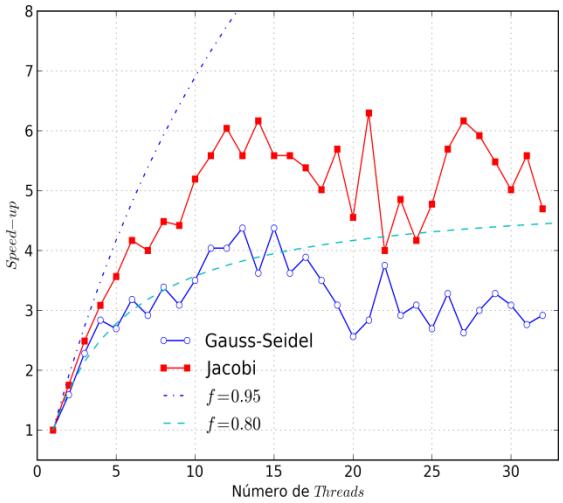
Em todas as figuras desta seção são apresentados valores de *speed-up* de referência, calculados com a Eq. (2.52), para programas que tenham, hipoteticamente, frações paralelas (f) de 80% e 95%.

6.2.1 Análise dos resultados para $N = 129 \times 129$ e 257×257

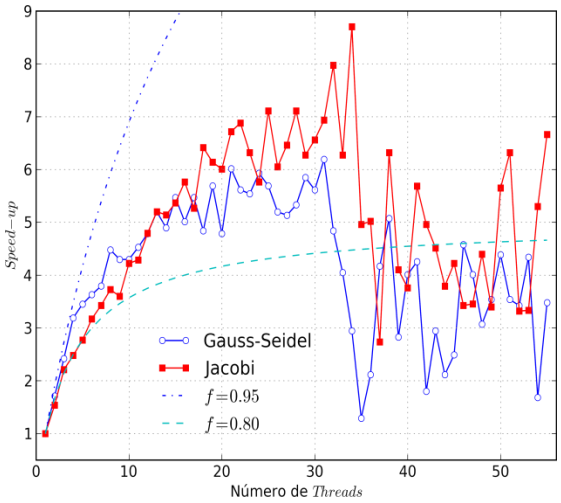
Na Fig. 6. 2 (a) observa-se que, até 15 processadores os valores de *speed-up*, para ambos os *solvers*, apresentaram tendência crescente, ou seja, empregar mais processadores utilizando este nível de refinamento $(N = 129 \times 129)$ não só implica em desperdício de recursos como ainda diminui o valor de *speed-up* obtido (o tempo de CPU passa a aumentar). Uma possível explicação para este resultado é que, para mais de 15 processadores, o tempo de sincronização dos *threads* passa a custar mais que a própria solução do problema. Pode-se observar que os valores de *speed-up*, Fig. 6.2 (a) e de eficiência, Fig. 6.2 (e), obtidos utilizando-se o *solver* Jacobi ponderado foram superiores, aos valores obtidos com o *solver* Gauss-Seidel *red-black* para todos os processadores utilizados. No entanto, o tempo de CPU do método *multi*grid utilizando o *solver* Jacobi ponderado foi maior, em relação ao *solver* Gauss-Seidel *red-black*, como pode ser observado na Fig. 6.2 (c).

Na Fig. 6. 2 (b) observa-se um comportamento semelhante ao da malha $N = 129 \times 129$: o sistema escalou (*speed-up* apresentou tendência crescente) até 31 processadores, para ambos os *solvers*. A partir de 31 processadores, custos computacionais extras, como o aumento do número de acessos à memória, custos de sincronização dos *threads*, entre outros, passam a afetar de forma negativa o tempo de CPU, como pode ser observado na Fig. 6.2 (d). Nesta malha, o uso do *solver* Gauss-Seidel *red-black* no método *multigrid* resultou em valores de *speed-up* maiores para até 11 *threads*. Em relação ao tempo de CPU, o uso do *solver* Gauss-Seidel *red-black* apresentou resultados melhores (tempo de CPU menor) para todos os 31 processadores (intervalo para o qual a adição de processadores resulta em aumento no *speed-up*).

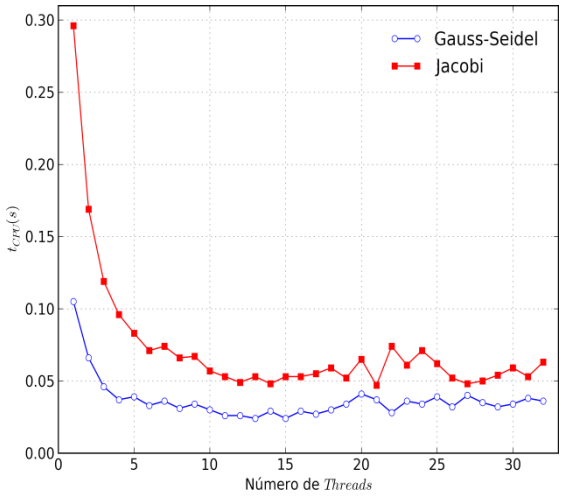
As oscilações (e queda no fator *speed-up*) nas Figs. 6.2 (a) e (b) não podem ser evitadas. Nas várias simulações realizadas para a faixa de processadores cuja adição resulta em diminuição de *speed-up* (e aumento no tempo de CPU), sempre foram observadas fortes oscilações. Não foi apresentado a média das simulações porque a curva resultante suaviza o comportamento oscilatório esperado.



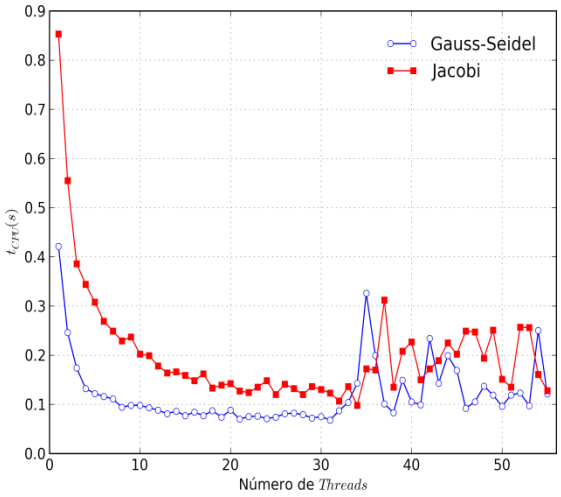
a) *Speed-up* para $N = 129 \times 129$



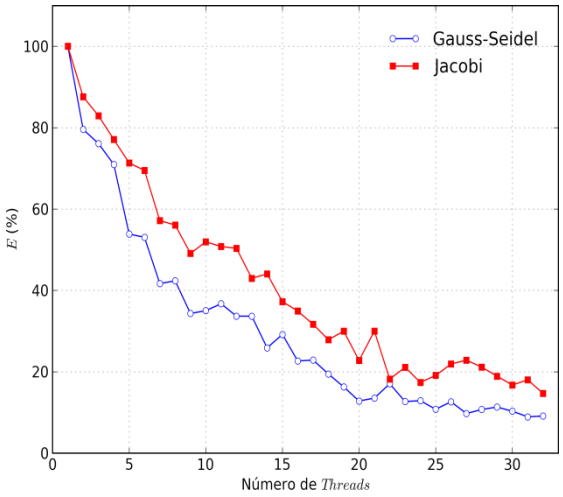
b) *Speed-up* para $N = 257 \times 257$



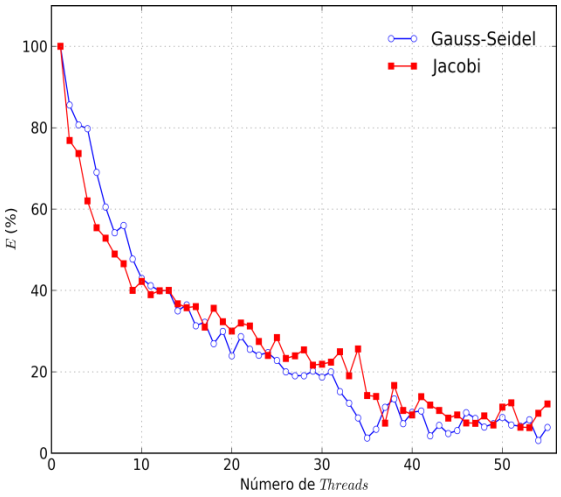
c) Tempo de CPU para $N = 129 \times 129$



d) Tempo de CPU para $N = 257 \times 257$



e) Eficiência para $N = 129 \times 129$



f) Eficiência para $N = 257 \times 257$

Figura 6.2: *Speed-up*, tempo de CPUs e eficiência em função do número de processadores.

6.2.2 Análise do fator *speed-up* para $N = 513 \times 513$, 1025×1025, 2049×2049, 4097×4097 e 8193×8193.

Pode-se notar que para todas as malhas apresentadas nesta seção os valores de *speed-up* são maiores para todos os processadores para o método *multigrid* utilizando o *solver* Jacobi ponderado, Figs. 6.3 (a), (b), (c), (d) e (e). Este resultado é esperado devido à diferença entre os graus de paralelismo dos *solvers* (TROTTENBERG et al. 2001): diferentemente do método Jacobi ponderado, em que todos os nós do domínio podem ser resolvidos paralelamente, no método Guass-Seidel *red-black* apenas metade dos nós da malha (nós *red* ou *black*) podem ser resolvidos paralelamente.

Das Figs. 6.3 (a), (b), (c), (d) e (e) pode-se afirmar que: para o problema estudado, metodologia empregada, hardware testado e refinamentos de malha utilizados, o método *multigrid* utilizando o *solver* Jacobi ponderado beneficia-se mais da paralelização implementada.

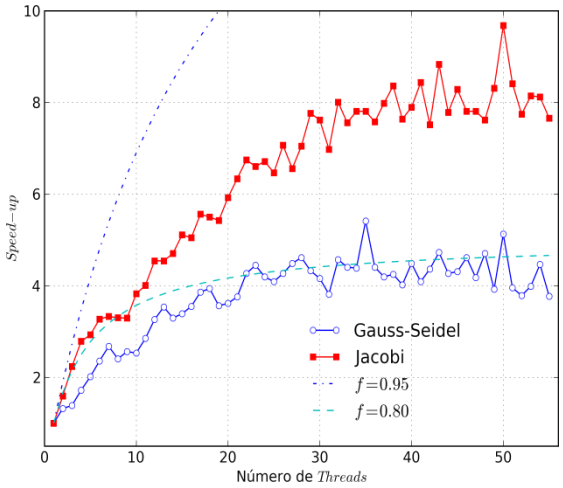
Quando foram utilizados 54 e 55 processadores na malha $N = 4097 \times 4097$, ocorreu um aumento acentuado de *speed-up*. Este exemplo ilustra a "sensibilidade" do fator *speed-up* em relação ao tempo de CPU, pois, para 53 processadores o fator *speed-up* é de 15,8 para um tempo de CPU de 11,51 segundos; para 55 processadores, o tempo de CPU é de 8,89 segundos, no entanto, o *speed-up* é de 20,4. Desses dados nota-se que a redução de 2,62 segundos no tempo de CPU gerou um aumento de 4,65 no *speed-up*.

Os valores máximos de *speed-up* obtidos utilizando o *solver* Gauss-Seidel *red-black* foram:

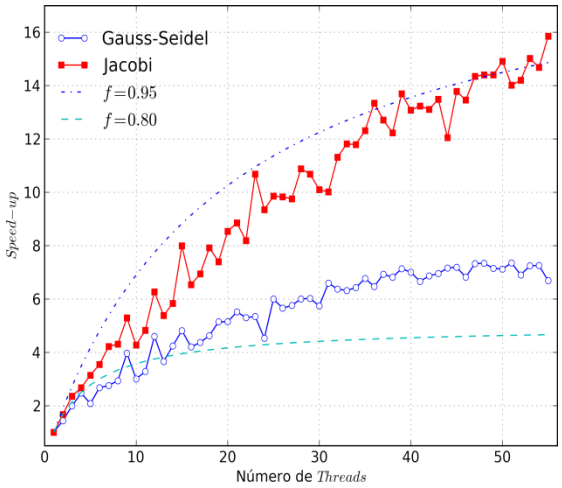
- $N = 513 \times 513$: 5,2.
- $N = 1025 \times 1025$: 7,4.
- $N = 2049 \times 2049$: 10,0.
- $N = 4097 \times 4097$: 11,3.
- $N = 8193 \times 8193$: 12,7.

Os valores máximos de *speed-up* obtidos utilizando o *solver* Jacobi ponderado foram:

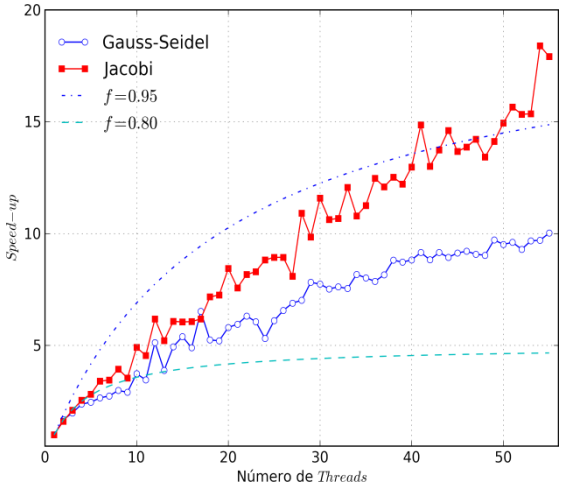
- $N = 513 \times 513$: 9,7.
- $N = 1025 \times 1025$: 15,9.
- $N = 2049 \times 2049$: 18,4.
- $N = 4097 \times 4097$: 20,4 (aumento acentuado durante o teste com 55 *threads*).
- $N = 8193 \times 8193$: 19,1.



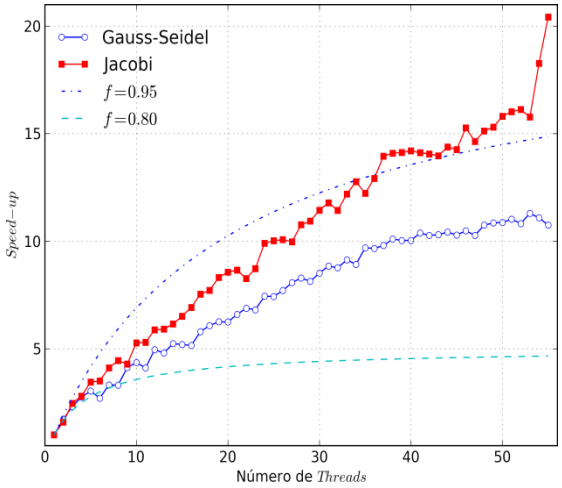
a) *Speed-up* para $N = 513 \times 513$



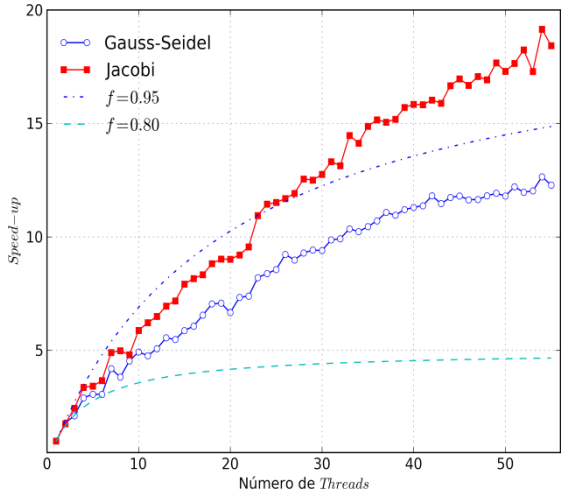
b) *Speed-up* para $N = 1025 \times 1025$



c) *Speed-up* para $N = 2049 \times 2049$



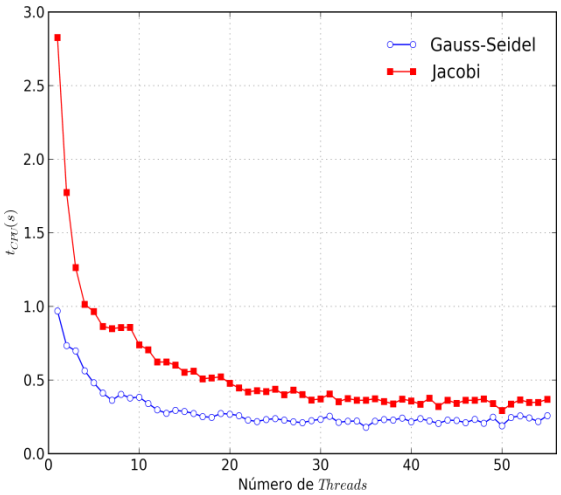
d) *Speed-up* para $N = 4097 \times 4097$



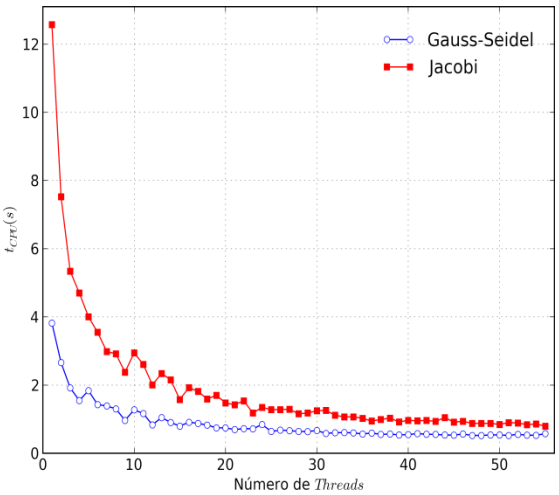
e) *Speed-up* para $N = 8193 \times 8193$

Figura 6.3: *Speed-up* em função do número de processadores.

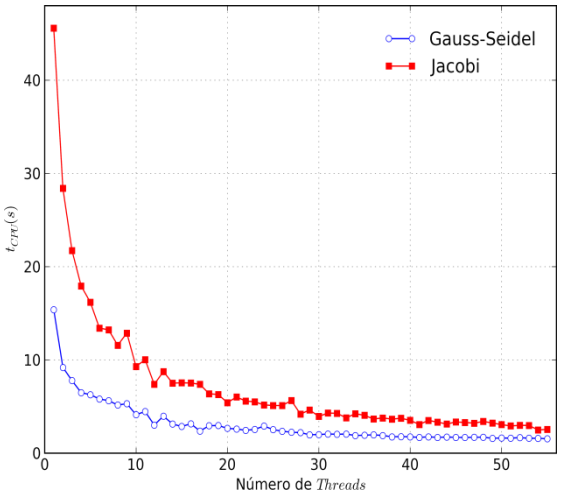
6.2.3 Análise do tempo de CPU para $N = 513\times513$, 1025×1025, 2049×2049, 4097×4097 e 8193×8193.



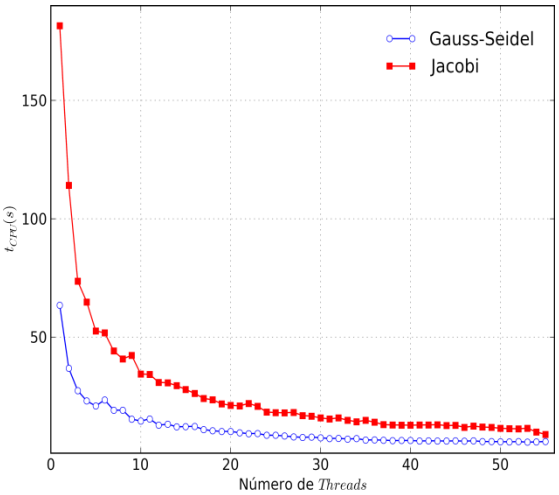
a) Tempo de CPU para $N = 513\times513$



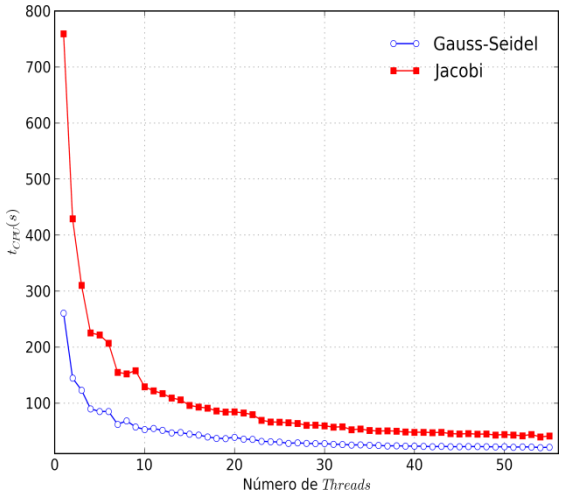
b) Tempo de CPU para $N = 1025\times1025$



c) Tempo de CPU para $N = 2049\times2049$



d) Tempo de CPU para $N = 4097\times4097$



e) Tempo de CPU para $N = 8193\times8193$

Figura 6.4: Tempo de CPU em função do número de processadores.

Pode-se notar que para todas as malhas o uso do *solver* Gauss-Seidel *red-black* apresentou melhores resultados em relação ao tempo de CPU para todos os processadores utilizados, Figs. 6.4 (a), (b), (c), (d) e (e).

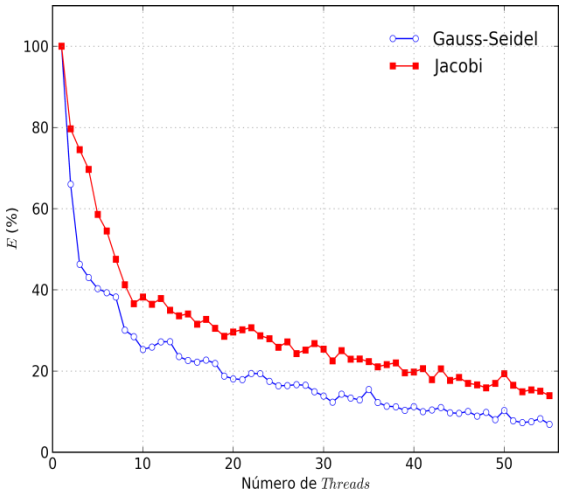
Das Figs. 6.4 (a), (b), (c), (d) e (e) pode-se afirmar que: para o problema estudado, metodologia empregada, hardware testado e refinamentos de malha utilizados, o método *multigrid* utilizando o *solver* Gauss-Seidel *red-black* apresenta tempo de CPU menor que o obtido com o *solver* Jacobi ponderado.

Estes resultados mostram a importância das propriedades de suavização do método Gauss-Seidel *red-black* visto que, apesar de possuir menor (metade) grau de paralelismo que o método Jacobi ponderado (TROTTENBERG et al. 2001), conduzem a um menor tempo e CPU. Nos experimentos simulados o fator de suavização do método Gauss-Seidel *red-black* não apenas compensou o *speed-up* superior do método Jacobi ponderado, como ainda garantiu tempos de CPU menores.

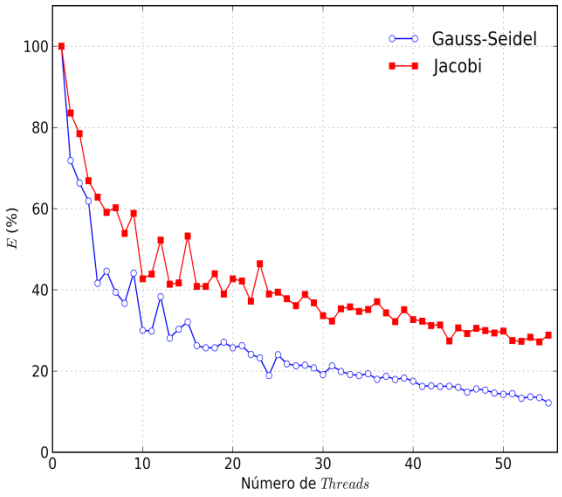
Dos resultados apresentados pode-se concluir que o método Jacobi ponderado é mais indicado para o estudo da paralelização pois se beneficia mais desta e apresenta resultados mais significativos em termos e *speed-up*, contudo em aplicações onde se pretende manter o menor tempo de processamento possível, ou viável em casos práticos, deve-se utilizar o método *multigrid* com o suavizador Gauss-Seidel *red-black*.

6.2.4 Análise da eficiência para $N = 513 \times 513$, 1025×1025, 2049×2049, 4097×4097 e 8193×8193.

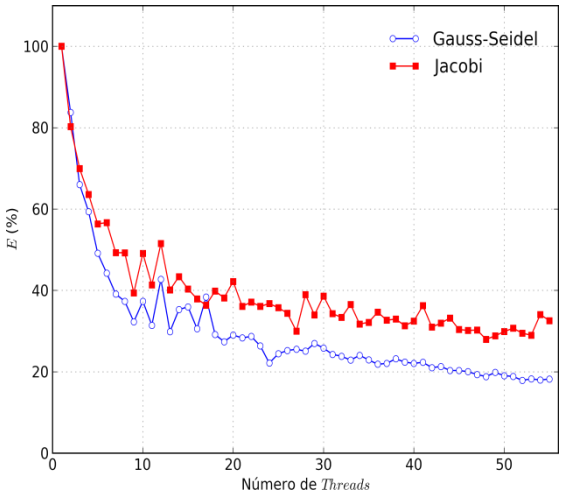
Pode-se notar que para todas as malhas apresentadas nesta seção os valores de eficiência são maiores para todos os processadores para o método *multigrid* utilizando o *solver* Jacobi ponderado, Figs. 6.5 (a), (b), (c), (d) e (e). Estes resultados são esperados pelas mesmas razões apontadas na avaliação do fator *speed-up*: diferença entre os graus de paralelismo dos *solvers* (TROTTENBERG et al. 2001). Como a eficiência pode ser interpretada como um indicador da efetividade com que os processadores são utilizados em paralelo era de se esperar que as sincronizações (uma para os nós *red* e outra para os nós *black*) reduzissem a eficiência do método *multigrid* utilizando o suavizador Gauss-Seidel *red-black*.



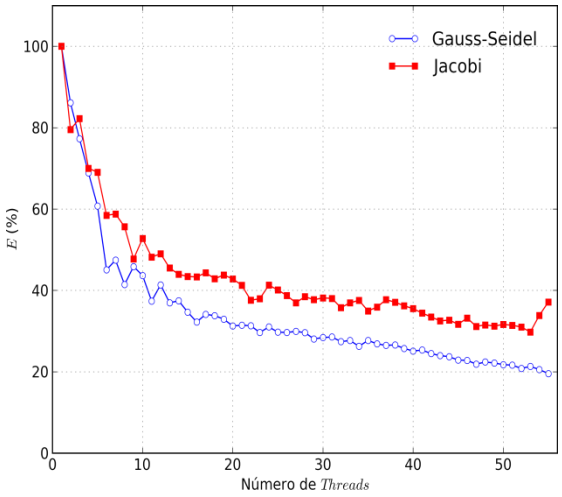
a) Eficiência para $N = 513 \times 513$



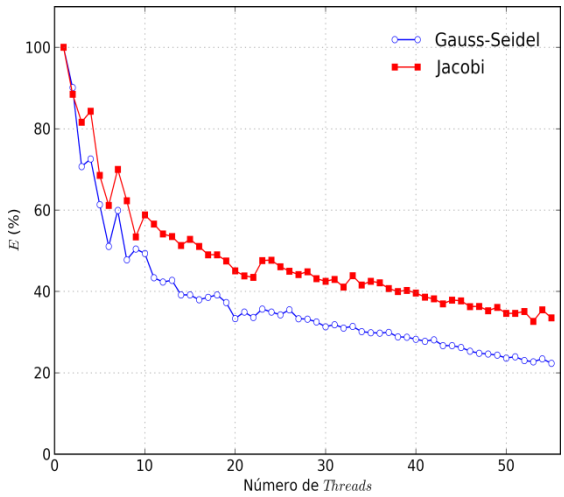
b) Eficiência para $N = 1025 \times 1025$



c) Eficiência para $N = 2049 \times 2049$



d) Eficiência para $N = 4097 \times 4097$



e) Eficiência para $N = 8193 \times 8193$

Figura 6.5: Eficiência em função do número de processadores.

7 Conclusão

A conclusão deste trabalho está divida em três seções: constatações gerais, contribuições e trabalhos futuros.

7.1 Constatações gerais

Neste trabalho foi resolvido numericamente o problema de condução de calor bidimensional linear governado pela equação de Laplace (INCROPERA et al., 2008), com condições de contorno de Dirichlet. Utilizou-se o MDF, com esquema de aproximação de segunda ordem (CDS) para discretização do modelo matemático. Os suavizadores (*solvers*) utilizados foram os métodos Gauss-Seidel *red-black* e Jacobi ponderado. Para a obtenção da solução, foi empregado o método *multigrid* geométrico, com esquema de correção CS, restrição por ponderação completa, prolongação utilizando interpolação bilinear e número máximo de níveis para os diversos casos estudados. A paralelização do *multigrid* foi realizada aplicando-se a metodologia proposta no capítulo 5, a cada uma de suas componentes algorítmicas: *solver*, processo de restrição, processo de prolongação e cálculo do resíduo.

Com base nos resultados obtidos neste trabalho, verificou-se que:

1) O emprego do *solver* Gauss-Seidel *red-black*, no método *multigrid*, apresenta tempo de CPU menor em relação ao uso do *solver* Jacobi ponderado. O método *multigrid* utilizando o *solver* Gauss-Seidel *red-black* foi 48% mais rápido na solução do problema proposto, considerando os melhores resultados (menor tempo de CPU) obtidos com ambos os *solvers*, na malha mais fina testada $\left(N = 8193 \times 8193\right)$.
2) O *speed-up* obtido com o *multigrid* utilizando Jacobi ponderado é, em geral, maior que o obtido com o uso do *solver* Gauss-Seidel *red-black*. O melhor resultado obtido com o uso do Jacobi ponderado resultou em *speed-up* 38% maior que o melhor resultado obtido com o uso do *solver* Gauss-Seidel.
3) A paralelização utilizando o *solver* Jacobi ponderado apresenta, em geral, maior eficiência em relação à paralelização obtida com o *solver* Gauss-Seidel *red-black*. Considerando os piores resultados de eficiência, pode-se apontar uma diferença de 15% entre ambos.

4) Em geral, o *speed-up* aumenta com a adição de processadores ao sistema. No entanto as malhas $N = 129 \times 129$ e $N = 257 \times 257$ apresentam limites de 15 e 31 processadores, respectivamente.
5) Para um número fixo de processadores, o valor do *speed-up* aumenta com o refinamento da malha.
6) Em geral, a eficiência do paralelismo aumenta com o refinamento da malha, mas diminui com a adição de processadores ao sistema.

7.2 Contribuições

Este trabalho contribui com a literatura no sentido que:

1) Neste trabalho propõe-se uma metodologia de particionamento do domínio que, diferentemente dos casos estudados por Yakel e Meyer (1992), Christou e Meyer (1996), e Martin (1998), não depende do tamanho da malha utilizada (tempo de CPU fixo) e apresenta balanceamento de carga ótimo (no máximo 1 nó de diferença entre os processadores).
2) Apresenta a paralelização do método *multigrid* utilizando o método de particionamento proposto (cujos subdomínios resultantes possuem forma não retangular e, em geral, forma geométrica desconhecida).
3) Demonstra que, para processadores com mais de um núcleo, a paralelização do método *multigrid*, utilizando a metodologia proposta, é vantajosa em relação à versão serial do mesmo.

7.3 Trabalhos futuros

Apresentam-se a seguir, algumas questões que servem de sugestões para novas pesquisas:

- O método de particionamento proposto apresentou robustez no trato de malhas estruturadas, propõe-se o estudo de sua viabilidade para o caso de malhas não estruturadas.

- A metodologia de paralelização, deste trabalho, foi desenvolvida considerando sistemas com memória compartilhada, propõe-se sua extensão ao caso de sistemas com memória distribuída.
- Neste trabalho, utilizaram-se processadores com vários núcleos, propõe-se paralelizar o método *multigrid* e realizar seu processamento em placas de vídeo (GPU).
- Propõe-se melhorar a metodologia de paralelização desenvolvida neste trabalho, focando no uso da memória *cache* dos processadores, como discutido em García et al. (2006) e Kowarschik et al. (2000).
- Propõe-se o estudo da paralelização do método *multigrid* para casos mais complexos como a Equação de Advecção-Difusão, Equações de Burgers e Navier-Stokes.

Referências bibliográficas

AL-NASRA, M.; NGUYEN, D., "An Algorithm for Domain Decomposition in Finite Element Analysis", Computer and Structures, [S.l.], v.39, p. 277–289, 1991.

AMDAHL, G. M., "Validity of the single processor approach to achieving large scale", AFIPS spring joint computer conference. IBM Sunnyvale, California: [s.n.], 1967, p. 483–485.

BAKER, A.H.; SCHULZ, M.; YANG, U.M., "On the Performance of an Algebraic Multigrid Solver on Multicore Clusters". In: J.M.L.M. Palma et al, editor, *VECPAR* 2010, Lecture Notes in Computer Science 6449, p. 102–115. Springer (2010) Berkeley, CA, June 2010.

BRAMBLE, J.H.; PASCIAK, J.E.; XU, J., "Parallel Multilevel Preconditioners", Math. Comp. v. 55, p. 1-22, 1990.

BRANDT, A., "Multi-Level Adaptive Solutions to Boundary-Value Problems, Mathematics of Computation", v. 31, p. 333-390, 1977.

BRIGGS, W., L.; HENSON, V., E.; MCCORMICK, S., "A Multigrid Tutorial", Second Edition, SIAM, Philadelphia, 2000.

BURDEN, R. L.; FAIRES, J. D., "Análise Numérica", Tradutor: Ricardo Lenzi Tombi. São Paulo: Pioneira Thomson Learning, 2008.

CHAN, T.F.; SAAD, Y., "Multigrid Algorithms on the Hypercube Multiprocessor", IEEE Trans. Comput, v. 35, p. 969-977, 1986.

CHAN, T.F.; SCHREIBER, R., "Parallel Networks for Multigrid Algorithms: Architecture and Complexity", SIAM J. Sci. Compu, v. 6, p. 698-711, 1985.

CHAPMAN, B.; JOST, G; VAN DER PAAS, A. R., "Using OpenMP - Portable Shared Memory Parallel Programming", The MIT PRESS. Massachusetts (EUA), 2008.

CHRISTOU, I. T.; MEYER, R. R., “Optimal Equi-partition of Rectangular Domains for Parallel Computation”, Journal of Global Optimization, vol. 8, p. 15-34, January, 1996.

CHUNG, T., “Computational Fluid Dynamics”, [S.l.]: Cambridge University Press, 2003.

COURTECUISSE, H.; ALLARD, J., “Parallel Dense Gauss-Seidel Algorithm on Many-Core Processors”, High Performance Computation Conference (HPCC), IEEE CS Press, June, 2009.

DATTA, K.; MURPHY, M.; VOLKOV, V.; WILLIAMS, S.; CARTER, J.; OLIKER, L.; PATTERSON, D.; SHALF, J.; YELICK, K., “Stencil Computation Optimization and Auto-Tuning on State-of-the-Art Multicore Architectures”, Proceedings of the 2008 ACM/IEEE conference on Supercomputing, November 15-21, Austin, Texas, 2008.

DONALDSON, W., “Grid-Graph Partitioning”, Dissertação (Mestrado), University of Wisconsin, Madison, EUA, 2000.

FERZIGER, J. H.; PERIC, M., “Computational Methods for Fluid Dynamics”, 3 ed., Berlin: Springer, 2002.

FORTUNA, A, O., “Técnicas Computacionais para Dinâmica dos Fluidos”, São Paulo: Edusp, 2000.

FREDERICKSON, P. O.; MCBRYAN, O. A, “Superconvergent Multigrid Methods”, Cornell Theory Center Preprint, May, 1987.

FREDERICKSON, P. O.; MCBRYAN, O. A, “Parallel Superconvergent Multigrid”, Multigrid Methods: Theory, Applications and Supercomputing, S. MCCORMICK, ed., MARCEL DEKKER, New York, 1988.

FREDERICKSON, P. O.; MCBRYAN, O. A, “Recent developments for Parallel Multigrid”, Proceedings of the Third European Conference on Multigrid Methods, October 1990, ed U. TROTTENBERG and W. HACKBUSCH.

GALANTE, G., “Métodos Multigrid Paralelos em Malhas Não-Estruturadas Aplicados à Simulação de Problemas de Dinâmica de Fluidos Computacional e Transferência de Calor”, Dissertação (Mestrado), Universidade Federal do Rio Grande do Sul, Porto Alegre, RS, 2006.

GALANTE, G.; MARTINOTTO, A. L.; JÚNIOR, D. P.; DORNELES, R. V.; RIZZI, R. L.; DIVERIO, T. A., “Comparação entre Métodos de Decomposição de Domínio e Decomposição de Dados na Solução de Sistemas de Equações”, In: V WORKSHOP EM SISTEMAS COMPUTACIONAIS DE ALTO DESEMPENHO, WSCAD, 2004, Foz do Iguaçu. Anais. **. .** SBC, p. 98–104, 2004.

GANNON, D.; VAN ROSENDALE, J., “On the Structure of Parallelism in a Highly Concurrent PDE Solver”, Parallel and Distributed Comput, v. 3, p. 106-135, 1986.

GARCÍA, C.; PRIETO, M.; TIRADO, F., “Multigrid Smoothers on Multicore Architectures”, Advances in Parallel Computing, v. 15, p. 279–286, 2008.

GOLUB, G. H.; ORTEGA, J. M., “Scientific Computing and Differential Equations: an Introduction to Numerical Methods”, Academic Press, Inc., 1992.

GOLUB, G. H.; VAN LOAN, C., “Matrix Computations”, 2 ed., Johns Hopkins Press, Baltimore, 1989.

GRAHAM, R. L.; KNUTH, D. E.; PATASHNIK, O., “Concrete Mathematics”, Addison-Wesley, 1989.

GRAMA, A.; GUPTA, A.; KARYPIS, G.; KUMAR, V., “Introduction to Parallel Computing, Second Edition”, 2. ed. [S.l.]: Addison Wesley, 2003.

GREENBAUM, A., “A Multigrid Method for Multiprocessors”, Appl. Math. Comput, v. 19, p. 75-88, 1986.

GUSTAFSON, J. L., “Reevaluating Amdahl's law”. Commun. ACM, v. 31, p. 532-533, Maio, 1988.

HEMPEL, R.; SCHÜLLER, A., “Experiments with Parallel Multigrid using the SUPRENUM Communications Library”, GMD-Studie, n. 141, 1988.

HERBIN, R.; GERBI, S.; SONNAD, V., “Parallel Implementation of a Multigrid Method on the Experimental ICAP Supercomputer”, Appl. Math. Comput., v. 27, p. 281-312, 1988.

HERMANNS, M., “Parallel Programming in Fortran 95 using OpenMP”, Universidad de Madrid. Madrid (Espanha), 75 p., 2002. Disponível em: http://www.openmp.org/presentations/miguel/F95_OpenMPv1_v2.pdf.

HIRSCH, C., “Numerical Computational of Internal and External Flows”, v.1. New York: John Wiley & Sons, 1988.

HUGHES, T. J. R., “The Finite Element Method – Linear Static and Dynamic Finite Element Analysis”, Mineola (USA): Dover Publications, Inc., 2000.

KOWARSCHIK, M.; RÜDE, U.; WEIß, C.; KARL, W., “Cache-Aware Multigrid Methods for Solving Poisson’s Equation in Two Dimensions”, Computing, v. 64, n. 04, p. 381–399, 2000.

MALISKA, C. R., “Transferência de Calor e Mecânica dos Fluidos Computacional”, Rio de Janeiro: LTC, 2 ed., 2004.

MARTIN, W., “Fast Equi-Partitioning of Rectangular Domains using Stripe Decomposition”, Discrete Applied Mathematics, v. 82, p. 193-207, 1998.

MCADAMS, A.; SIFAKIS, E.; TERAN, J., “A parallel multigrid Poisson solver for fluids simulation on large grids”, Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, Madrid, Spain, July 02-04, 2010.

MCBRYAN, O. A., “Sequencial and Parallel Efficiency of Multigrid Fast Solvers”, University of Colorado CS Dept. Tech Report, September 1990.

MCBRYAN, O., A.; FREDERICKSON, P., O.; LINDEN, J.; SHULLER, A.; SOLCHENBACH, K.; STUBEN, K.; THOLE, C. A.; TROTTENBERG, U., "Multigrid Methods on Parallel Computers – a Survey on Recent Developments", 1990.

MESQUITA, M. S., "Solução Numérica de Escoamentos bidimensionais Não-isotérmicos usando o Método Multigrid. Dissertação de mestrado em Engenharia Aeronáutica e Mecânica. Instituto Tecnológico de Aeronáutica (ITA) ", São José dos Campos, SP, 2000.

OLIVEIRA, F., "Efeitos de malhas anisotrópicas bidimensionais sobre o desempenho do método Multigrid geométrico", Tese (Doutorado em Engenharia Mecânica), Universidade Federal do Paraná, Curitiba, PR, 2010.

OLIVEIRA, F.; PINTO, M. A. V.; MARCHI, C. H.; ARAKI, L. K., "Optimized partial semicoarsening Multigrid algorithm for heat diffusion problems and anisotropic grids", Applied Mathematical Modelling, v.36, p. 4665-4676, 2012.

PATANKAR, S., "Numerical heat transfer and fluid flow", New York: Taylor and Francis, 1980.

PINTO, M. A. V.; MARCHI, C. H., "Optimum Parameters of a Geometric Multigrid for the Two-Dimensional Laplace's Equation", Proceedings of COBEM, 2007.

POLEZZI, M., "A Geometrical Method for Finding an Explicit Formula for the Greatest Common Divisor", v. 104, n. 5, p. 445-446, May, 1997.

QUARTERONI, A.; VALLI, A, "Domain Decomposition Methods for Partial Differential Equations", Oxford Science, Oxford, UK, 1999.

REDDYE, J. N.; GARTLING, D. K., "The Finit Element Method in Heat Transfer and Fluid Dynamics", Boca Raton (USA): CRC Press, 1994.

SILBERSCHATZ, A.; GALVIN, P.; GAGNE, G., “Operating System Concepts”, 7 ed, John Wiley and Sons, 2004.

SMITH, B.; BJORSTAD, P.; GROPP, W., “Domain Decomposition -Parallel Multilevel Methods for Elliptic Partial Differential Equations”, Cambridge University Press, Cambridge, 1996.

SUN, X. H.; CHEN, Y., “Reevaluating Amdahl's law in the multicore era”, Journal of Parallel and Distributed Computing, v. 70, p. 183-188, 2010.

SUN, X. H.; NI, L. M., “Scalable problems and memory-bounded speedup”, Journal of Parallel and Distributed Computing, v. 19, p. 27–37, 1993.

TANENBAUM, A. S., “Modern Operating Systems”, Prentice Hall PTR, Upper Saddle River, NJ, 2001.

TANNEHILL, J. C.; ANDERSON, D. A; PLETCHER, R. H., “Computational Fluid Mechanics and Heat Transfer”, 2 ed. Washington: Taylor & Francis, 1997.

TIAN, X.; BIK, A.; GIRKAR, M.; GREY, P.; SAITO, H.; SU, E., “Intel OpenMP C++/Fortran Compiler for Hyper-Threading Technology: Implementation and Performance”, Intel Technology Journal, v. 06, n. 01, p. 36-47, 2002.

THOLE, C. A., “Experiments with Multigrid Methods on the CalTech-hypercube”, GMD-Studie, n. 103, 1985.

THOMAS, J. L.; DISKIN, B.; BRANDT, A., “Textbook multigrid efficiency for fluid simulations”, Annual Review of Fluid Mechanics, 35, p. 317-340, 2003.

THÜREY, N.; GÖTZ, J.; STÜRMER, M.; DONATH, S.; FEICHTINGER, C.; IGLBERGER, K.; PRECLIC, T. ; NEUMANN, P., “Parallel Multigrid Methods Numerical Simulation in the Multi-Core Age”, Disponível em: http://www10.informatik.uni-

erlangen.de/Publications/Talks/2009/Ruede_Marseille_090213.pdf, Acessado 12/02/2012.

TROTTENBERG, U.; OOSTERLEE, C.; SCHÜLLER, A., "Multigrid", Academic Press, London, 2001.

VERSTEEG, H. K.; MALALASEKERA, W., "An Introduction to Computational Fluid Dynamic, The Finite Volume Method", England: Longman, 2 ed., 2007.

XU, J., "Iterative Methods by Space Decomposition and Subspace Correction", *SZAM* Rev. v. 34, p. 581-613, 1992.

WESSELING, P., "An introduction to Multigrid Methods", New York: John Wiley & Sons, 1992.

WALLIN, D., LOEF, H., HAGERSTEN, E., HOLMGREN, S., "Multigrid and Gauss-Seidel smoothers revisited: Parallelization on chip multiprocessors", In ICS 2006, Proceedings, p. 145–155, 2006.

YACKEL, J.; MEYER, R. R., "Optimal Tilings for Parallel Database Design", In PARDALOS, P. M. (org.), Advances in Optimization and Parallel Computing, Amsterdam: Elsevier / North Holland, p. 293-309, 1992.

ZHANG, J., "Multigrid Method and Fourth-Order Compact Scheme for 2D Poisson Equation with Unequal Mesh-Size Discretization", Journal of Computational Physics, v. 179, p. 170-179, 2002.

Apêndice A – Resultados para o processador Core I7

Neste apêndice são apresentados os resultados obtidos em um processador mais acessível. A motivação deste apêndice está no fato de que os resultados apresentados aqui, demonstram que a metodologia de paralelização desenvolvida é vantajosa, mesmo para processadores mais acessíveis, desde que disponham de mais de um núcleo (ou *thread*).

A Tab. A. 1, a seguir, resume os resultados obtidos com o *multigrid* utilizando o *solver* Gauss-Seidel *red-black*. Nela são apresentadas as malhas utilizadas (N), o número de *threads* $(CPUs)$ empregados em paralelo, o tempo de CPU $(t_{CPU}(s))$ em segundos, o *speed-up* (S_n) e a eficiência $(E(\%))$.

Tabela A. 1: Tempo de CPU, *speed-up* e Eficiência do método *multigrid* utilizando o *solver* Gauss-Seidel *red-black*.

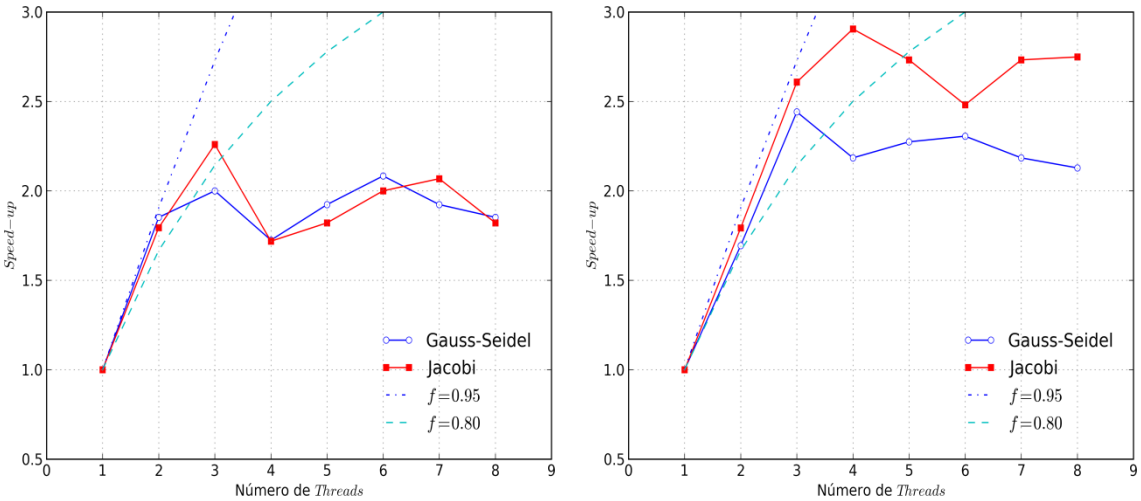
Multigrid em paralelo utilizando o *solver* Gauss-Seidel *red-black* (Parte 1)												
CPUs	1			2			3			4		
N	$t_{CPU}(s)$	S_n	$E(\%)$	$t_{CPU}(s)$	S_n	$E(\%)$	$t_{CPU}(s)$	S_n	$E(\%)$	$t_{CPU}(s)$	S_n	$E(\%)$
129^2	0,05	1	100	0,027	1,9	92,6	0,025	2,0	66,7	0,029	1,7	43,1
257^2	0,166	1	100	0,098	1,7	84,7	0,068	2,4	81,4	0,076	2,2	54,6
513^2	0,666	1	100	0,379	1,8	87,9	0,28	2,4	79,3	0,234	2,8	71,2
1025^2	2,696	1	100	1,509	1,8	89,3	1,153	2,3	77,9	0,942	2,9	71,5
2049^2	10,78	1	100	6,064	1,8	88,9	4,493	2,4	80,0	3,712	2,9	72,6
4097^2	43,14	1	100	24,31	1,8	88,7	17,70	2,4	81,2	14,58	3,0	74,0
8193^2	176,4	1	100	98,55	1,8	89,5	66,95	2,6	87,8	59,1	3,0	74,6
Multigrid em paralelo utilizando o *solver* Gauss-Seidel *red-black* (Parte 2)												
CPUs	5			6			7			8		
N	$t_{CPU}(s)$	S_n	$E(\%)$	$t_{CPU}(s)$	S_n	$E(\%)$	$t_{CPU}(s)$	S_n	$E(\%)$	$t_{CPU}(s)$	S_n	$E(\%)$
129^2	0,026	1,9	38,5	0,024	2,1	34,7	0,026	1,9	27,5	0,027	1,9	23,1
257^2	0,073	2,3	45,5	0,072	2,3	38,4	0,076	2,2	31,2	0,078	2,1	26,6
513^2	0,231	2,9	57,7	0,254	2,6	43,7	0,222	3,0	42,9	0,213	3,1	39,1
1025^2	0,917	2,9	58,8	0,85	3,2	52,9	0,761	3,5	50,6	0,807	3,3	41,8
2049^2	3,339	3,2	64,6	3,169	3,4	56,7	2,662	4,0	57,9	2,538	4,2	53,1
4097^2	13,75	3,1	62,7	11,94	3,6	60,2	10,15	4,3	60,7	9,543	4,5	56,5
8193^2	54,85	3,2	64,3	45,73	3,9	64,3	41,34	4,3	61,0	37,75	4,7	58,4

Semelhante à Tab. A. 1, a Tab. A. 2 a seguir resume os resultados obtidos com o *multigrid* utilizando o *solver* Jacobi ponderado.

Tabela A. 2: Tempo de CPU, *speed-up* e Eficiência do método *multigrid* utilizando o *solver* Jacobi ponderado.

Multigrid em paralelo utilizando o *solver Jacobi ponderado* (Parte 1)												
CPUs	1			2			3			4		
N	$t_{CPU}(s)$	S_n	$E(\%)$	$t_{CPU}(s)$	S_n	$E(\%)$	$t_{CPU}(s)$	S_n	$E(\%)$	$t_{CPU}(s)$	S_n	$E(\%)$
129^2	0,122	1	100	0,068	1,8	89,7	0,054	2,3	75,3	0,071	1,7	43,0
257^2	0,459	1	100	0,256	1,8	89,6	0,176	2,6	86,9	0,158	2,9	72,6
513^2	1,854	1	100	1,022	1,8	90,7	0,744	2,5	83,1	0,656	2,8	70,7
1025^2	7,48	1	100	4,027	1,9	92,9	2,865	2,6	87,0	2,359	3,2	79,3
2049^2	29,97	1	100	16,16	1,9	92,7	11,55	2,6	86,5	9,264	3,2	80,9
4097^2	119,8	1	100	64,45	1,9	92,9	45,74	2,6	87,3	36,84	3,3	81,3
8193^2	488,2	1	100	261,6	1,9	93,3	177,3	2,8	91,8	148,5	3,3	82,2
Multigrid em paralelo utilizando o *solver* Jacobi ponderado (Parte 2)												
CPUs	5			6			7			8		
N	$t_{CPU}(s)$	S_n	$E(\%)$	$t_{CPU}(s)$	S_n	$E(\%)$	$t_{CPU}(s)$	S_n	$E(\%)$	$t_{CPU}(s)$	S_n	$E(\%)$
129^2	0,067	1,8	36,4	0,061	2,0	33,3	0,059	2,1	29,5	0,067	1,8	22,8
257^2	0,168	2,7	54,6	0,185	2,5	41,4	0,168	2,7	39,0	0,167	2,7	34,4
513^2	0,563	3,3	65,9	0,599	3,1	51,6	0,552	3,4	48,0	0,57	3,3	40,7
1025^2	2,155	3,5	69,4	1,923	3,9	64,8	1,638	4,6	65,2	2,101	3,6	44,5
2049^2	8,28	3,6	72,4	7,367	4,1	67,8	6,241	4,8	68,6	5,929	5,1	63,2
4097^2	33,74	3,6	71,0	28,84	4,2	69,2	24,66	4,9	69,4	22,9	5,2	65,4
8193^2	134,2	3,6	72,8	113,6	4,3	71,6	101,2	4,8	68,9	92,25	5,3	66,2

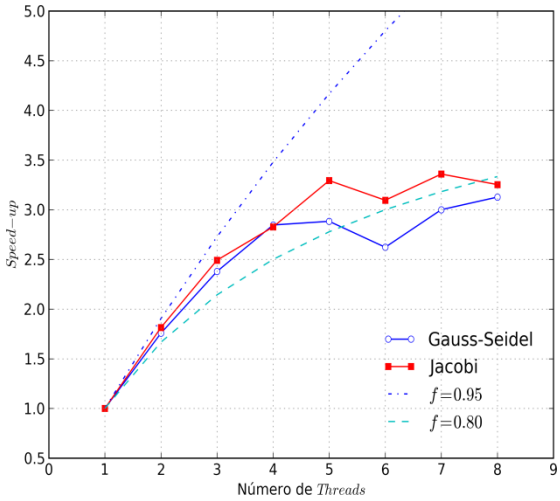
Em todas as figuras desta seção são apresentados valores de *speed-up* de referência, calculados com a Eq. (2.52), para programas que tenham, hipoteticamente, frações paralelas (f) de 80% e 95%.



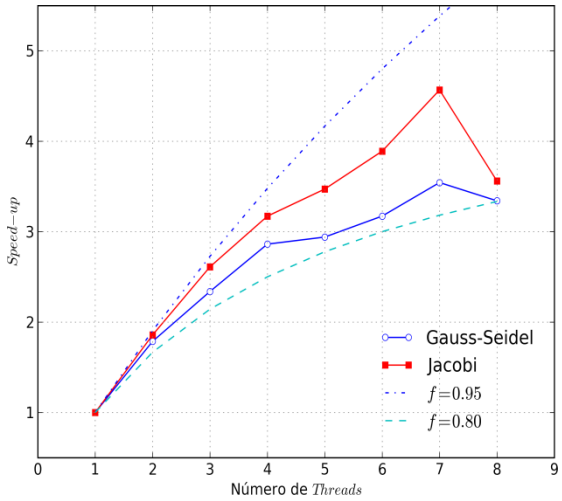
a) $N = 129 \times 129$ b) $N = 257 \times 257$

Figura A 1 : *Speed-up* oscilatório para Core I7

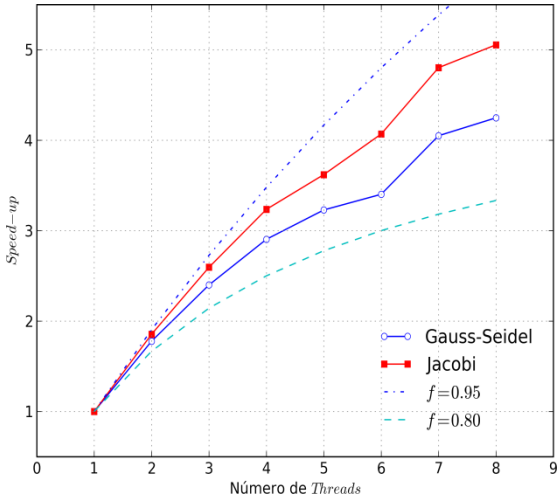
Na Fig. A 1 pode-se notar que para as malhas $N = 129 \times 129$ e 257×257 o fator *speed-up* apresenta comportamento oscilatório semelhante ao caso da Fig. 6.2 (a) e (b).



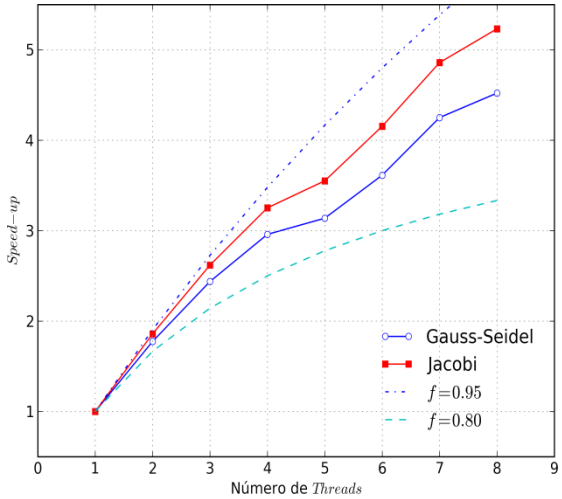
a) $N = 513 \times 513$



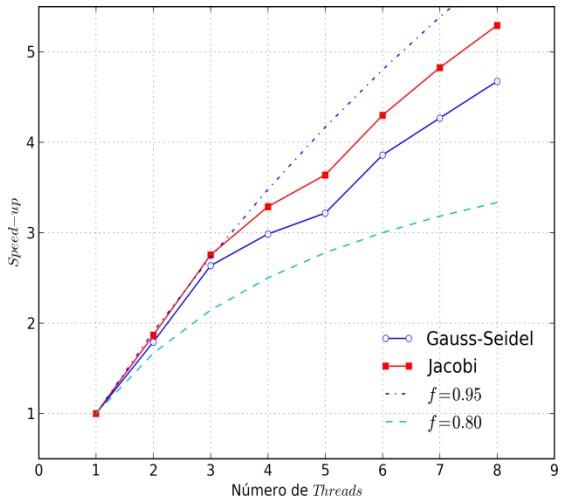
b) $N = 1025 \times 1025$



c) $N = 2049 \times 2049$



d) $N = 4097 \times 4097$



e) $N = 8193 \times 8193$

Figura A 2 : *Speed-up* para Core I7

Da Fig. A 2, nota-se que a paralelização é vantajosa para processadores mais acessíveis.

Apêndice B – Versão completa das tabelas dos resultados

Neste apêndice são apresentadas as versões completas (para todos os 55 *threads*) das tabelas apresentadas no capítulo 6.

A Tab. B. 1 apresenta todos os resultados obtidos com a paralelização do método *multigrid* utilizando o *solver* Gauss-Seidel *red-black*. Nela são apresentadas as malhas utilizadas (N), o número de *threads* $(CPUs)$ empregados em paralelo, o tempo de CPU $(t_{CPU}(s))$ em segundos, o *speed-up* (S_n) e a eficiência $(E(\%))$ obtido pelo processamento paralelo do método *multigrid*.

A Tab. B. 2 apresenta todos os resultados obtidos com a paralelização do método *multigrid* utilizando o *solver* Jacobi ponderado. Nela são apresentadas as malhas utilizadas (N), o número de *threads* $(CPUs)$ empregados em paralelo, o tempo de CPU $(t_{CPU}(s))$ em segundos, o *speed-up* (S_n) e a eficiência $(E(\%))$ obtido pelo processamento paralelo do método *multigrid*.

Nas Tabs. B. (1) e (2) não são apresentados resultados obtidos na malha $N = 129 \times 129$ porque os resultados são considerados não confiáveis. Na Fig. 6.2 (a) pode-se observar que a partir de 15 *threads* os resultados tornam-se altamente oscilatórios.

É necessário recomendar cautela ao considerar os resultados para a malha $N = 257 \times 257$ a partir de 30 *threads* pois são resultados semelhantes ao caso da Fig. 6.2 (a), ou seja, apresentam valores altamente oscilatórios para o fator *speed-up*.

Tabela B. 1: Resultados obtidos com a paralelização do método *multigrid* utilizando Gauss-Seidel *red-black*.

Gauss-Seidel *red-black*									
N	257x257			513x513			1025x1025		
$CPUs$	$t_{CPU}(s)$	S_n	$E(\%)$	$t_{CPU}(s)$	S_n	$E(\%)$	$t_{CPU}(s)$	S_n	$E(\%)$
1	0,421	1,0	100,0	0,969	1,0	100,0	3,817	1,0	100,0
2	0,246	1,7	85,6	0,734	1,3	66,0	2,658	1,4	71,8
3	0,174	2,4	80,7	0,698	1,4	46,3	1,918	2,0	66,3
4	0,132	3,2	79,7	0,563	1,7	43,0	1,542	2,5	61,9
5	0,122	3,5	69,0	0,481	2,0	40,3	1,835	2,1	41,6
6	0,116	3,6	60,5	0,411	2,4	39,3	1,427	2,7	44,6
7	0,111	3,8	54,2	0,362	2,7	38,2	1,384	2,8	39,4
8	0,094	4,5	56,0	0,403	2,4	30,1	1,301	2,9	36,7
9	0,098	4,3	47,7	0,378	2,6	28,5	0,962	4,0	44,1
10	0,098	4,3	43,0	0,383	2,5	25,3	1,271	3,0	30,0
11	0,093	4,5	41,2	0,34	2,9	25,9	1,163	3,3	29,8
12	0,088	4,8	39,9	0,297	3,3	27,2	0,83	4,6	38,3
13	0,081	5,2	40,0	0,274	3,5	27,2	1,045	3,7	28,1
14	0,086	4,9	35,0	0,294	3,3	23,5	0,9	4,2	30,3
15	0,077	5,5	36,5	0,286	3,4	22,6	0,793	4,8	32,1
16	0,084	5,0	31,3	0,273	3,5	22,2	0,908	4,2	26,3
17	0,077	5,5	32,2	0,251	3,9	22,7	0,873	4,4	25,7
18	0,087	4,8	26,9	0,246	3,9	21,9	0,825	4,6	25,7
19	0,074	5,7	29,9	0,272	3,6	18,8	0,742	5,1	27,1
20	0,088	4,8	23,9	0,268	3,6	18,1	0,742	5,1	25,7
21	0,07	6,0	28,6	0,258	3,8	17,9	0,692	5,5	26,3
22	0,075	5,6	25,5	0,227	4,3	19,4	0,72	5,3	24,1
23	0,076	5,5	24,1	0,218	4,4	19,3	0,714	5,3	23,2
24	0,071	5,9	24,7	0,231	4,2	17,5	0,843	4,5	18,9
25	0,074	5,7	22,8	0,237	4,1	16,4	0,636	6,0	24,0
26	0,081	5,2	20,0	0,227	4,3	16,4	0,675	5,7	21,7
27	0,082	5,1	19,0	0,216	4,5	16,6	0,663	5,8	21,3
28	0,079	5,3	19,0	0,21	4,6	16,5	0,636	6,0	21,4
29	0,072	5,8	20,2	0,224	4,3	14,9	0,634	6,0	20,8
30	0,075	5,6	18,7	0,233	4,2	13,9	0,665	5,7	19,1
31	0,068	6,2	20,0	0,254	3,8	12,3	0,579	6,6	21,3
32	0,087	4,8	15,1	0,212	4,6	14,3	0,599	6,4	19,9
33	0,104	4,0	12,3	0,22	4,4	13,3	0,604	6,3	19,2
34	0,143	2,9	8,7	0,221	4,4	12,9	0,594	6,4	18,9
35	0,326	1,3	3,7	0,179	5,4	15,5	0,564	6,8	19,3
36	0,199	2,1	5,9	0,22	4,4	12,2	0,59	6,5	18,0
37	0,101	4,2	11,3	0,231	4,2	11,3	0,551	6,9	18,7
38	0,083	5,1	13,3	0,228	4,3	11,2	0,56	6,8	17,9
39	0,149	2,8	7,2	0,241	4,0	10,3	0,535	7,1	18,3
40	0,105	4,0	10,0	0,216	4,5	11,2	0,545	7,0	17,5
41	0,099	4,3	10,4	0,237	4,1	10,0	0,573	6,7	16,2
42	0,234	1,8	4,3	0,222	4,4	10,4	0,556	6,9	16,3
43	0,143	2,9	6,8	0,205	4,7	11,0	0,549	7,0	16,2
44	0,199	2,1	4,8	0,227	4,3	9,7	0,533	7,2	16,3
45	0,169	2,5	5,5	0,225	4,3	9,6	0,531	7,2	16,0
46	0,092	4,6	9,9	0,21	4,6	10,0	0,56	6,8	14,8
47	0,105	4,0	8,5	0,232	4,2	8,9	0,521	7,3	15,6
48	0,137	3,1	6,4	0,206	4,7	9,8	0,52	7,3	15,3
49	0,119	3,5	7,2	0,247	3,9	8,0	0,534	7,1	14,6
50	0,096	4,4	8,8	0,189	5,1	10,3	0,536	7,1	14,2
51	0,119	3,5	6,9	0,245	4,0	7,8	0,519	7,4	14,4
52	0,123	3,4	6,6	0,256	3,8	7,3	0,553	6,9	13,3
53	0,097	4,3	8,2	0,243	4,0	7,5	0,527	7,2	13,7
54	0,25	1,7	3,1	0,217	4,5	8,3	0,526	7,3	13,4
55	0,121	3,5	6,3	0,257	3,8	6,9	0,571	6,7	12,2

Gauss-Seidel *red-black*									
N	2049x2049			4097x4097			8193x8193		
$CPUs$	$t_{CPU}(s)$	S_n	$E(\%)$	$t_{CPU}(s)$	S_n	$E(\%)$	$t_{CPU}(s)$	S_n	$E(\%)$
1	15,39	1,0	100,0	63,42	1,0	100,0	260,6	1,0	100,0
2	9,19	1,7	83,7	36,83	1,7	86,1	144,6	1,8	90,1
3	7,774	2,0	66,0	27,36	2,3	77,3	122,9	2,1	70,7
4	6,484	2,4	59,3	23,02	2,8	68,9	89,8	2,9	72,6
5	6,266	2,5	49,1	20,89	3,0	60,7	84,96	3,1	61,3
6	5,801	2,7	44,2	23,46	2,7	45,1	85,1	3,1	51,0
7	5,626	2,7	39,1	19,1	3,3	47,4	62,14	4,2	59,9
8	5,156	3,0	37,3	19,14	3,3	41,4	68,21	3,8	47,8
9	5,298	2,9	32,3	15,37	4,1	45,8	57,43	4,5	50,4
10	4,123	3,7	37,3	14,53	4,4	43,6	52,86	4,9	49,3
11	4,454	3,5	31,4	15,43	4,1	37,4	54,65	4,8	43,4
12	3,002	5,1	42,7	12,78	5,0	41,4	51,33	5,1	42,3
13	3,968	3,9	29,8	13,21	4,8	36,9	46,93	5,6	42,7
14	3,116	4,9	35,3	12,1	5,2	37,4	47,56	5,5	39,1
15	2,854	5,4	35,9	12,2	5,2	34,7	44,41	5,9	39,1
16	3,146	4,9	30,6	12,3	5,2	32,2	42,94	6,1	37,9
17	2,361	6,5	38,3	10,94	5,8	34,1	39,78	6,6	38,5
18	2,935	5,2	29,1	10,43	6,1	33,8	36,99	7,0	39,1
19	2,958	5,2	27,4	10,14	6,3	32,9	36,82	7,1	37,3
20	2,653	5,8	29,0	10,15	6,2	31,2	39,09	6,7	33,3
21	2,588	5,9	28,3	9,61	6,6	31,4	35,55	7,3	34,9
22	2,439	6,3	28,7	9,212	6,9	31,3	35,26	7,4	33,6
23	2,54	6,1	26,3	9,304	6,8	29,6	31,76	8,2	35,7
24	2,899	5,3	22,1	8,522	7,4	31,0	31,09	8,4	34,9
25	2,522	6,1	24,4	8,534	7,4	29,7	30,45	8,6	34,2
26	2,347	6,6	25,2	8,229	7,7	29,6	28,25	9,2	35,5
27	2,234	6,9	25,5	7,853	8,1	29,9	29	9,0	33,3
28	2,194	7,0	25,1	7,651	8,3	29,6	28,05	9,3	33,2
29	1,968	7,8	27,0	7,794	8,1	28,1	27,66	9,4	32,5
30	1,989	7,7	25,8	7,444	8,5	28,4	27,74	9,4	31,3
31	2,046	7,5	24,3	7,17	8,8	28,5	26,41	9,9	31,8
32	2,021	7,6	23,8	7,235	8,8	27,4	26,27	9,9	31,0
33	2,041	7,5	22,8	6,945	9,1	27,7	25,16	10,4	31,4
34	1,883	8,2	24,0	7,107	8,9	26,2	25,47	10,2	30,1
35	1,92	8,0	22,9	6,542	9,7	27,7	24,9	10,5	29,9
36	1,957	7,9	21,8	6,56	9,7	26,9	24,34	10,7	29,7
37	1,888	8,2	22,0	6,473	9,8	26,5	23,53	11,1	29,9
38	1,747	8,8	23,2	6,277	10,1	26,6	23,79	11,0	28,8
39	1,765	8,7	22,4	6,32	10,0	25,7	23,26	11,2	28,7
40	1,745	8,8	22,0	6,319	10,0	25,1	23,07	11,3	28,2
41	1,682	9,1	22,3	6,108	10,4	25,3	22,91	11,4	27,7
42	1,744	8,8	21,0	6,175	10,3	24,5	22,07	11,8	28,1
43	1,682	9,1	21,3	6,156	10,3	24,0	22,73	11,5	26,7
44	1,722	8,9	20,3	6,084	10,4	23,7	22,21	11,7	26,7
45	1,686	9,1	20,3	6,164	10,3	22,9	22,08	11,8	26,2
46	1,671	9,2	20,0	6,05	10,5	22,8	22,4	11,6	25,3
47	1,696	9,1	19,3	6,173	10,3	21,9	22,37	11,6	24,8
48	1,705	9,0	18,8	5,901	10,7	22,4	22,07	11,8	24,6
49	1,584	9,7	19,8	5,846	10,8	22,1	21,86	11,9	24,3
50	1,618	9,5	19,0	5,833	10,9	21,7	22,08	11,8	23,6
51	1,602	9,6	18,8	5,753	11,0	21,6	21,35	12,2	23,9
52	1,656	9,3	17,9	5,86	10,8	20,8	21,78	12,0	23,0
53	1,591	9,7	18,3	5,616	11,3	21,3	21,67	12,0	22,7
54	1,587	9,7	18,0	5,72	11,1	20,5	20,6	12,7	23,4
55	1,536	10,0	18,2	5,9	10,7	19,5	21,22	12,3	22,3

Tabela B. 2: Resultados obtidos com a paralelização do método *multigrid* utilizando Jacobi ponderado.

Jacobi ponderado									
N	257x257			513x513			1025x1025		
$CPUs$	$t_{CPU}(s)$	S_n	$E(\%)$	$t_{CPU}(s)$	S_n	$E(\%)$	$t_{CPU}(s)$	S_n	$E(\%)$
1	0,853	1,0	100,0	2,826	1,0	100,0	12,57	1,0	100,0
2	0,555	1,5	76,8	1,774	1,6	79,7	7,523	1,7	83,5
3	0,386	2,2	73,7	1,264	2,2	74,5	5,34	2,4	78,5
4	0,344	2,5	62,0	1,014	2,8	69,7	4,699	2,7	66,9
5	0,308	2,8	55,4	0,965	2,9	58,6	4,001	3,1	62,8
6	0,269	3,2	52,9	0,864	3,3	54,5	3,546	3,5	59,1
7	0,249	3,4	48,9	0,849	3,3	47,6	2,981	4,2	60,2
8	0,229	3,7	46,6	0,856	3,3	41,3	2,916	4,3	53,9
9	0,237	3,6	40,0	0,858	3,3	36,6	2,374	5,3	58,8
10	0,202	4,2	42,2	0,739	3,8	38,2	2,943	4,3	42,7
11	0,199	4,3	39,0	0,705	4,0	36,4	2,604	4,8	43,9
12	0,178	4,8	39,9	0,622	4,5	37,9	2,005	6,3	52,2
13	0,164	5,2	40,0	0,622	4,5	34,9	2,337	5,4	41,4
14	0,166	5,1	36,7	0,601	4,7	33,6	2,153	5,8	41,7
15	0,159	5,4	35,8	0,553	5,1	34,1	1,573	8,0	53,3
16	0,148	5,8	36,0	0,56	5,0	31,5	1,923	6,5	40,9
17	0,162	5,3	31,0	0,508	5,6	32,7	1,811	6,9	40,8
18	0,133	6,4	35,6	0,514	5,5	30,5	1,588	7,9	44,0
19	0,139	6,1	32,3	0,521	5,4	28,5	1,699	7,4	38,9
20	0,142	6,0	30,0	0,477	5,9	29,6	1,472	8,5	42,7
21	0,127	6,7	32,0	0,446	6,3	30,2	1,42	8,9	42,2
22	0,124	6,9	31,3	0,419	6,7	30,7	1,536	8,2	37,2
23	0,135	6,3	27,5	0,428	6,6	28,7	1,177	10,7	46,4
24	0,148	5,8	24,0	0,421	6,7	28,0	1,345	9,3	38,9
25	0,12	7,1	28,4	0,437	6,5	25,9	1,275	9,9	39,4
26	0,141	6,0	23,3	0,4	7,1	27,2	1,279	9,8	37,8
27	0,132	6,5	23,9	0,431	6,6	24,3	1,289	9,8	36,1
28	0,12	7,1	25,4	0,401	7,0	25,2	1,155	10,9	38,9
29	0,136	6,3	21,6	0,364	7,8	26,8	1,177	10,7	36,8
30	0,13	6,6	21,9	0,371	7,6	25,4	1,245	10,1	33,7
31	0,123	6,9	22,4	0,405	7,0	22,5	1,255	10,0	32,3
32	0,107	8,0	24,9	0,353	8,0	25,0	1,111	11,3	35,4
33	0,136	6,3	19,0	0,374	7,6	22,9	1,064	11,8	35,8
34	0,098	8,7	25,6	0,362	7,8	23,0	1,066	11,8	34,7
35	0,172	5,0	14,2	0,362	7,8	22,3	1,021	12,3	35,2
36	0,17	5,0	13,9	0,373	7,6	21,0	0,942	13,3	37,1
37	0,312	2,7	7,4	0,354	8,0	21,6	0,989	12,7	34,4
38	0,135	6,3	16,6	0,338	8,4	22,0	1,028	12,2	32,2
39	0,208	4,1	10,5	0,37	7,6	19,6	0,918	13,7	35,1
40	0,227	3,8	9,4	0,358	7,9	19,7	0,961	13,1	32,7
41	0,15	5,7	13,9	0,335	8,4	20,6	0,95	13,2	32,3
42	0,172	5,0	11,8	0,376	7,5	17,9	0,959	13,1	31,2
43	0,189	4,5	10,5	0,32	8,8	20,5	0,932	13,5	31,4
44	0,225	3,8	8,6	0,363	7,8	17,7	1,043	12,1	27,4
45	0,202	4,2	9,4	0,341	8,3	18,4	0,912	13,8	30,6
46	0,249	3,4	7,4	0,362	7,8	17,0	0,934	13,5	29,3
47	0,247	3,5	7,3	0,362	7,8	16,6	0,876	14,3	30,5
48	0,194	4,4	9,2	0,371	7,6	15,9	0,873	14,4	30,0
49	0,251	3,4	6,9	0,34	8,3	17,0	0,873	14,4	29,4
50	0,151	5,6	11,3	0,292	9,7	19,4	0,843	14,9	29,8
51	0,135	6,3	12,4	0,336	8,4	16,5	0,897	14,0	27,5
52	0,257	3,3	6,4	0,365	7,7	14,9	0,885	14,2	27,3
53	0,256	3,3	6,3	0,347	8,1	15,4	0,837	15,0	28,3
54	0,161	5,3	9,8	0,348	8,1	15,0	0,856	14,7	27,2
55	0,128	6,7	12,1	0,369	7,7	13,9	0,793	15,9	28,8

Jacobi ponderado									
N	2049x2049			4097x4097			8193x8193		
$CPUs$	$t_{CPU}(s)$	S_n	$E(\%)$	$t_{CPU}(s)$	S_n	$E(\%)$	$t_{CPU}(s)$	S_n	$E(\%)$
1	45,58	1,0	100,0	181,5	1,0	100,0	759,1	1,0	100,0
2	28,4	1,6	80,2	114,1	1,6	79,5	429	1,8	88,5
3	21,72	2,1	70,0	73,58	2,5	82,2	310,1	2,4	81,6
4	17,92	2,5	63,6	64,82	2,8	70,0	225,2	3,4	84,3
5	16,18	2,8	56,3	52,58	3,5	69,0	221,6	3,4	68,5
6	13,41	3,4	56,6	51,76	3,5	58,4	207	3,7	61,1
7	13,22	3,4	49,3	44,12	4,1	58,8	154,9	4,9	70,0
8	11,57	3,9	49,2	40,78	4,5	55,6	152,3	5,0	62,3
9	12,87	3,5	39,4	42,27	4,3	47,7	157,9	4,8	53,4
10	9,287	4,9	49,1	34,41	5,3	52,7	129,1	5,9	58,8
11	10,02	4,5	41,4	34,24	5,3	48,2	122	6,2	56,6
12	7,374	6,2	51,5	30,88	5,9	49,0	116,9	6,5	54,1
13	8,747	5,2	40,1	30,69	5,9	45,5	109,2	7,0	53,5
14	7,506	6,1	43,4	29,5	6,2	43,9	105,7	7,2	51,3
15	7,536	6,0	40,3	27,87	6,5	43,4	95,85	7,9	52,8
16	7,518	6,1	37,9	26,2	6,9	43,3	92,96	8,2	51,0
17	7,374	6,2	36,4	24,08	7,5	44,3	91,11	8,3	49,0
18	6,36	7,2	39,8	23,53	7,7	42,9	86,08	8,8	49,0
19	6,288	7,2	38,2	21,82	8,3	43,8	84,12	9,0	47,5
20	5,402	8,4	42,2	21,19	8,6	42,8	84,24	9,0	45,1
21	6,019	7,6	36,1	20,96	8,7	41,2	82,46	9,2	43,8
22	5,581	8,2	37,1	21,96	8,3	37,6	79,42	9,6	43,4
23	5,496	8,3	36,1	20,8	8,7	37,9	69,36	10,9	47,6
24	5,169	8,8	36,7	18,32	9,9	41,3	66,33	11,4	47,7
25	5,101	8,9	35,7	18,11	10,0	40,1	65,92	11,5	46,1
26	5,101	8,9	34,4	18,02	10,1	38,7	64,94	11,7	45,0
27	5,633	8,1	30,0	18,2	10,0	36,9	63,69	11,9	44,1
28	4,18	10,9	38,9	16,86	10,8	38,4	60,47	12,6	44,8
29	4,63	9,8	33,9	16,6	10,9	37,7	60,72	12,5	43,1
30	3,935	11,6	38,6	15,86	11,4	38,1	59,53	12,8	42,5
31	4,291	10,6	34,3	15,4	11,8	38,0	57,01	13,3	43,0
32	4,27	10,7	33,4	15,87	11,4	35,7	57,78	13,1	41,1
33	3,779	12,1	36,5	14,89	12,2	36,9	52,46	14,5	43,8
34	4,226	10,8	31,7	14,22	12,8	37,5	53,73	14,1	41,6
35	4,051	11,3	32,1	14,84	12,2	34,9	51,04	14,9	42,5
36	3,656	12,5	34,6	14,05	12,9	35,9	50,08	15,2	42,1
37	3,77	12,1	32,7	13,01	14,0	37,7	50,42	15,1	40,7
38	3,642	12,5	32,9	12,88	14,1	37,1	49,99	15,2	40,0
39	3,734	12,2	31,3	12,85	14,1	36,2	48,35	15,7	40,3
40	3,513	13,0	32,4	12,78	14,2	35,5	47,92	15,8	39,6
41	3,068	14,9	36,2	12,86	14,1	34,4	47,98	15,8	38,6
42	3,505	13,0	31,0	12,91	14,1	33,5	47,36	16,0	38,2
43	3,319	13,7	31,9	12,99	14,0	32,5	47,77	15,9	37,0
44	3,12	14,6	33,2	12,62	14,4	32,7	45,56	16,7	37,9
45	3,335	13,7	30,4	12,73	14,3	31,7	44,75	17,0	37,7
46	3,287	13,9	30,1	11,89	15,3	33,2	45,51	16,7	36,3
47	3,206	14,2	30,2	12,4	14,6	31,1	44,48	17,1	36,3
48	3,397	13,4	28,0	12	15,1	31,5	44,87	16,9	35,2
49	3,229	14,1	28,8	11,86	15,3	31,2	42,97	17,7	36,1
50	3,052	14,9	29,9	11,48	15,8	31,6	43,9	17,3	34,6
51	2,912	15,7	30,7	11,33	16,0	31,4	43,03	17,6	34,6
52	2,973	15,3	29,5	11,26	16,1	31,0	41,63	18,2	35,1
53	2,969	15,4	29,0	11,51	15,8	29,8	43,91	17,3	32,6
54	2,478	18,4	34,1	9,934	18,3	33,8	39,65	19,1	35,5
55	2,545	17,9	32,6	8,887	20,4	37,1	41,21	18,4	33,5