UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

ANDRÉ LUIS MARTINOTTO

Resolução de Sistemas de Equações Lineares através de Métodos de Decomposição de Domínio

Dissertação apresentada como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. Tiarajú Asmuz Diverio
Orientador

Porto Alegre, feveiro de 2004

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Martinotto, André Luis

Resolução de Sistemas de Equações Lineares através de Métodos de Decomposição de Domínio / André Luis Martinotto. – Porto Alegre: PPGC da UFRGS, 2004.

96 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2004. Orientador: Tiarajú Asmuz Diverio.

1. Resolução paralela de sistemas de equações. 2. Métodos de decomposição de domínio . I. Diverio, Tiarajú Asmuz. II. Título.

"Não há ramo da Matemática, por mais abstrato que seja, que não possa um dia vir a ser aplicado aos fenômenos do mundo real."

— BOYER

AGRADECIMENTOS

Gostaria de expressar todos meus agradecimentos às pessoas que direta ou indiretamente contribuíram para o desenvolvimento deste trabalho. Em especial quero deixar registrados meus agradecimentos a:

Meu orientador Prof. Tiarajú Asmuz Diverio pela confiança depositada em mim ao ter me aceito no mestrado. Pelo seu esforço em concerder-me uma bolsa de pesquisa, pela orientação e pelo companherismo demonstrado durante esses dois anos de trabalho.

Aos meus amigos e colegas Cadinho, Delcino e Rogério que muito contribuíram para o desenvolvimento desse trabalho. A eles devo um enorme agradecimento pelo apoio, contribuições, sugestões e momentos de descontração.

Aos amigos e colegas dos grupos GMCPAD e GPPD que tornaram esse período muito enriquecedor e agrádavel.

Aos meus demais amigos simplesmente por estarem sempre a meu lado. A eles devo agradecer por todos os bons momentos compartilhados.

Aos meus padrinhos, Cláudio e Gema, pelo incentivo, apoio e por estarem ao meu lado sempre que precisei.

À Roberta pelo companherismo, carinho, estímulo, compreensão e por aturar os meus momentos de mau humor.

Aos meus pais, Adelar e Helena, pelo apoio irrestrito em todos os momentos. Sem o auxílio deles não seria possível chegar até aqui.

Aos professores do Instituto de Informática pelos ensinamentos transmitidos e aos funcionários pela constante disponibilidade e simpatia.

Ao CNPQ pelo auxílio financeiro e ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Informática da UFRGS por todos os recursos disponibilizados.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

API	*Application Program Interface*
ATM	*Asynchronous Transfer Mode*
Bi-CG	*Bi-Conjugate Gradient*
Bi-CGSTAB	*Bi-Conjugate Gradient Stabilized*
BLAS	*Basic Linear Algebra Subprograms*
CC	Condição de Contorno
CGNE	*Conjugate Gradient for Normal Equations with Minimisation of the Residual Norm*
CGNR	*Conjugate Gradient for Normal Equations with Minimisation of the Error Norm*
CGS	*Conjugate Gradient Squared*
CI	Condição Inicial
CR	*Conjugate Residual*
CSC	*Compressed Sparse Column*
CSR	*Compressed Sparse Row*
DECK	*Distributed Execution and Communication Kernel*
DSM	*Distributed Shared Memory*
EDF	Equações de Diferenças Finitas
EDP	Equações Diferenciais Parciais
FOM	*Full Orthogonalization Method*
GC	Gradiente Conjugado
GCR	*Generalized Conjugate Residual*
GPPD	Grupo de Processamento Paralelo e Distribuído
GMCPAD	Grupo de Matemática da Computação e Processamento de Alto Desempenho
GMRES	*Generalized Minimum Residual*
HP	*Hewlett-Packard*

IBM	*International Business Machines*
IC	*Incomplete Cholesky*
IEEE	*Institute of Electrical and Electronics Engineers*
ILU	*Incomplete LU*
IML++	*Iterative Methods Library*
LabTeC	Laboratório de Tecnologia em *Clusters*
LND	*Levelized Nested Dissection*
MDD	Métodos de Decomposição de Domínio
MPE	*MultiProcessing Environment*
MPI	*Message Passing Interface*
MSR	*Modified Sparse Row*
PAPI	*Performance Application Programming Interface*
PC	*Personal Computers*
PCI	*Peripheral Component Interconnect*
PETSc	*Portable, Extensible Toolkit for Scientific Computation*
PIM	*Parallel Iterative Methods*
POSIX	*Portable Operating System Interface*
PVM	*Parallel Virtual Machine*
QMR	*Quasi-Minimal Residual*
RAM	*Random Access Memory*
RBi-CGSTAB	*Restarted Bi-Conjugate Gradient Stabilized*
RCB	*Recursive Coordinate Bisection*
RSB	*Recursive Spectral Bisection*
RWCP	*Real World Computing Partnership*
SCI	*Scalable Coherent Interface*
SDP	Simétrica e Definida-Positiva
SOR	*Successive Overrelaxation*
SSOR	*Symmetric Successive Overrelaxation*
TFQMR	*Transpose-Free Quasi-Minimum Residual Method*
UFRGS	Universidade Federal do Rio Grande do Sul
VBR	*Variable Block Row*

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

A paralelização de métodos de resolução de sistemas de equações lineares e não lineares é uma atividade que tem concentrado várias pesquisas nos últimos anos. Isto porque, os sistemas de equações estão presentes em diversos problemas da computação científica, especialmente naqueles que empregam equações diferenciais parciais (EDPs) que modelam fenômenos físicos, e que precisam ser discretizadas para serem tratadas computacionalmente.

O processo de discretização resulta em sistemas de equações que necessitam ser resolvidos a cada passo de tempo. Em geral, esses sistemas têm como características a esparsidade e um grande número de incógnitas. Devido ao porte desses sistemas é necessária uma grande quantidade de memória e velocidade de processamento, sendo adequado o uso de computação de alto desempenho na obtenção da solução dos mesmos.

Dentro desse contexto, é feito neste trabalho um estudo sobre o uso de métodos de decomposição de domínio na resolução de sistemas de equações em paralelo. Esses métodos baseiam-se no particionamento do domínio computacional em subdomínios, de modo que a solução global do problema é obtida pela combinação apropriada das soluções de cada subdomínio. Uma vez que diferentes subdomínios podem ser tratados independentemente, tais métodos são atrativos para ambientes paralelos.

Mais especificamente, foram implementados e analisados neste trabalho, três diferentes métodos de decomposição de domínio. Dois desses com sobreposição entre os subdomínios, e um sem sobreposição. Dentre os métodos com sobreposição foram estudados os métodos aditivo de Schwarz e multiplicativo de Schwarz. Já dentre os métodos sem sobreposição optou-se pelo método do complemento de Schur.

Todas as implementações foram desenvolvidas para serem executadas em *clusters* de PCs multiprocessados e estão incorporadas ao modelo HIDRA, que é um modelo computacional paralelo multifísica desenvolvido no Grupo de Matemática da Computação e Processamento de Alto Desempenho (GMCPAD) para a simulação do escoamento e do transporte de substâncias em corpos de águas.

Palavras-chave: Resolução paralela de sistemas de equações, Métodos de decomposição de domínio .

Solution of Systems of Linear Equations through Domain Decomposition Methods

ABSTRACT

The parallelization of methods for the solution of linear and non-linear equations systems is an area where a lot of research has been developed in the last years. It happens because equation systems are present in several problems of scientific computing, specially in those where partial differential equations (PDEs) are used to model physical phenomena, and must be discretized to be treated computationally.

The discretization process results in equations systems that must be solved at each time step. Generally, these systems have as characteristics, sparsity and a great number of unknowns. Due to the size of these systems, it is necessary a great amount of memory and processing power, making the use of high performance computing adequate to obtain their solution.

Inside this context, in this work a study about the use of domain decomposition methods in the parallel solution of equations systems is done. These methods are based on the partitioning of the computational domain in subdomains, such that the global solution for the problem is obtained by the appropriate combination of the solutions of all the subdomains. As different subdomains can be treated independently, such methods are attractive for use in parallel environments.

More specifically, in these work three different methods of domain decomposition have been implemented and analyzed, two of them using domain overlapping and one of them without it. The methods with domain overlapping chosen were additive and multiplicative Schwarz methods. The method without overlapping chosen was Schur complement method.

All the implementations were developed to be executed in multiprocessor PC clusters and were incorporated to the HIDRA model, a multiphysics parallel computational model developed in GMCPAD (High Performance Processing and Computer Mathematics Group) to the simulation of flow and substance transport in water bodies.

Keywords: Parallel solution of equations systems, Domain decomposition methods.

1 INTRODUÇÃO

A simulação numérica é uma importante abordagem para o desenvolvimento científico e tecnológico, uma vez que esta pode ser utilizada em situações onde experiências reais são impossíveis ou economicamente inviáveis, como, por exemplo, a transferência de calor no núcleo de um reator nuclear ou na simulação de reservatórios de petróleo (MALISKA, 1995).

Em geral, problemas que empregam simulação são baseados em modelos matemáticos, que consideram os aspectos relevantes do fenômeno em questão. Esses problemas, freqüentemente, são expressos matematicamente através de Equações Diferenciais Parciais (EDPs).

As EDPs são definidas em infinitos pontos, e em geral, não possuem solução analítica conhecida, sendo necessário o uso de métodos numéricos de discretização, como diferenças finitas ou elementos finitos, para sua aproximação em um domínio discreto. No processo de discretização o domínio é dividido em um número finito de pontos, denominado de malha. Nesses pontos os termos das EDPs são aproximados, resultando em um sistema de equações lineares ou não lineares, que devem ser resolvidos a cada passo de tempo. A matriz dos coeficientes desses sistemas de equações é, geralmente, esparsa e de grande porte, onde o número de incógnitas pode chegar a ordem de milhares ou de milhões (CANAL, 2000).

Numericamente, a acurácia da solução depende, além do método de aproximação das EDPs utilizado, do número de pontos da malha. Quanto mais refinada for a malha, mais acurada será a solução. Em contrapartida, maior será a ordem do sistema de equações a ser resolvido (FORTUNA, 2000). Sendo assim, em simulações numéricas realísticas de grande escala espaço-temporal, e de alta acurácia numérica, o tempo computacional necessário para simular tais fenômenos pode ser muito elevado. Para remediar essa situação utiliza-se, cada vez mais freqüentemente, computação de alto desempenho e, em particular, a computação em *clusters* de PCs.

Existem, pelo menos, duas grandes abordagens para a resolução de sistemas de equações em paralelo. Em uma delas, chamada de decomposição de dados, gera-se um único sistema de equações para todo o domínio que é resolvido através de um método númerico paralelizado. Como exemplo dessa abordagem cita-se (DEMMEL; HEATH; VORST, 1993) e (PICININ, 2003).

A segunda abordagem consiste na utilização de métodos de decomposição de domínio (MDDs). Esses são baseados no particionamento do domínio computacional em subdomínios, de modo que a solução global do problema é obtida pela combinação apropriada das soluções obtidas em cada um dos subdomínios. Uma vez que diferentes subdomínios podem ser tratados independentemente em paralelo tais métodos são atrativos para ambientes computacionais paralelos de memória distribuída.

De acordo com a forma que é feito o particionamento do domínio os MDDs podem ser divididos em duas classes: métodos de Schwarz, onde os subdomínios apresentam uma região de sobreposição, e métodos de Schur, onde os subdomínios não apresentam região de sobreposição (SMITH; BJORSTAD; GROPP, 1996), (CHAN; MATHEW, 1994).

Neste trabalho foram desenvolvidas implementações utilizando a abordagem de decomposição de domínio, com e sem sobreposição, para resolução paralela dos sistemas de equações gerados pelo modelo paralelo de hidrodinâmica e transporte de substâncias desenvolvido no Grupo de Matemática da Computação e Processamento de Alto Desempenho (GMCPAD). As implementações propostas neste trabalho foram desenvolvidas de forma a explorar o paralelismo em *clusters* de PCs multiprocessados. Em *clusters* com essas características, além do paralelismo inter-nodos pode-se explorar o paralelismo intra-nodos, como será caracterizado na seção 2.1.3.

1.1 Motivação e Objetivos

O GMCPAD vem trabalhando no desenvolvimento de aplicações de alto desempenho desde 1998. Como resultado deste trabalho, tem-se o modelo HIDRA, que é um modelo computacional paralelo com balanceamento dinâmico de carga para a simulação do escoamento e do transporte de substâncias, tridimensional e bidimensional, em corpos de águas, tendo como estudo de caso o Lago Guaíba (RIZZI, 2002), (DORNELES, 2003).

O método utilizado na discretização do sistema de EDPs que modelam o fenômeno físico em questão, foi o de diferenças finitas que, combinado com métodos de aproximação apropriados para as variáveis envolvidas, gera sistemas de equações que são lineares, de grande porte e esparsos.

Particularmente no caso das EDPs discretas para a hidrodinâmica, os sistemas de equações gerados têm a característica de que as matrizes de coeficientes são simétricas e definidas-positivas (SDP). Nesse caso, pode-se resolver os sistemas de equações através do método do gradiente conjugado (GC), que é um dos mais efetivos métodos iterativos no subespaço de Krylov para obter a solução de sistemas com essas características (SHEWCHUK, 1994).

Como os sistemas de equações gerados pelo modelo HIDRA são de grande porte é conveniente o uso de processamento paralelo. Neste sentido, o presente trabalho procurou desenvolver implementações paralelas, baseadas na abordagem de decomposição de domínio para a resolução dos sistemas de equações resultantes da discretização das EDPs da hidrodinâmica no modelo HIDRA. Mais especificamente, foram desenvolvidos e analisados três métodos de decomposição de domínio, dois com região de sobreposição, que são os métodos multiplicativo de Schwarz e aditivo de Schwarz, e um sem região de sobreposição, que é o método do complemento de Schur.

1.2 Trabalhos Relacionados

Alguns trabalhos e dissertações relacionados com a paralelização de métodos de resolução de sistemas de equações lineares, utilizando troca de mensagens e *threads*, foram ou estão sendo desenvolvidos no GMCPAD. Os trabalhos já concluídos são:

- Ana Paula Canal, que desenvolveu a dissertação intitulada de "Paralelização de Métodos de Resolução de Sistemas Lineares Esparsos com o DECK em Clusters de PCs", onde foram implementadas versões paralelas do método do Gradiente Conjugado (GC) e o método de Thomas para matrizes banda. As implementações foram

desenvolvidas utilizando a biblioteca de troca de mensagens DECK e a biblioteca de *threads* Pthreads (CANAL, 2000);

- Delcino Picinin Júnior, que desenvolveu o trabalho individual sobre a paralelização do GC utilizando MPI e Pthreads (PICININ, 2001) e em sua dissertação de mestrado intitulada "Paralelização de Métodos Numéricos em Clusters Empregando as Bibliotecas MPICH, DECK e Pthreads", onde desenvolvou e analisou a paralelização do método do GC e do Método do Resíduo Mínimo Generalizado (GMRES) para matrizes esparsas irregulares (PICININ, 2003);

- André Luis Martinotto, desenvolveu o trabalho de conclusão de curso intitulado "Paralelização de Métodos Numéricos de Resolução de Sistemas Esparsos de Equações Utilizando MPI e Pthreads"(MARTINOTTO, 2001) e o trabalho individual sobre a paralelização de pré-condicionadores para o método do GC (MARTINOTTO, 2002).

A principal distinção entre este trabalho e os trabalhos já desenvolvidos no GMCPAD refere-se à abordagem utilizada na paralelização. Trabalhos anteriores utilizam a abordagem de decomposição de dados, enquanto este trabalho baseia-se na abordagem de decomposição de domínio. Essa abordagem recebeu nos últimos anos uma atenção especial por ser particularmente atrativa do ponto de vista da computação científica paralela. Isto porque os métodos de decomposição de domínio se baseiam no particionamento do domínio físico em subdomínios, de tal forma que os subdomínios podem ser tratados independentemente, restringindo a comunicação às fronteiras artificiais, criadas pelo particionamento do domínio.

A primeira abordagem no uso da decomposição de domínio como método de resolução foi aquela proposta por Hermann Amandus Schwarz em 1869, sendo empregado para resolver um problema elíptico definido em um domínio irregular, formado pela união de dois subdomínios regulares sobrepostos (disco e retângulo). O método desenvolvido por Schwarz foi baseado na solução do problema de modo alternado em cada subdomínio, sendo que os valores calculados em um subdomínio, em uma determinada iteração, são utilizados como condição de contorno do tipo Dirichlet para o outro subdomínio na iteração seguinte. Este método é conhecido como método alternado de Schwarz (FLEMISH, 2001)·

Em 1936 Sobolev propôs uma nova formulação matemática (abstrata) para o método original de Schwarz. Devido a essa nova formulação o método original de Schwarz passou a ser conhecido, na literatura técnica, como método multiplicativo de Schwarz. Nela o método multiplicativo de Schwarz, originalmente proposto para dois subdomínios, pode ser generalizado para vários subdomínios. Porém, o método multiplicativo de Schwarz é inerentemente seqüencial, já que utiliza como condições de contorno os valores calculados nos subdomínios vizinhos na mesma iteração. Para obter paralelismo nessa versão é necessário o uso de alguma técnica de coloração dos subdomínios. O método multiplicativo é abordado nos trabalhos (CAI; SAAD, 1993) e (CAI, 1994).

Posteriormente, Dryja e Widlund (DRYJA; WIDLUND, 1987) analisando o método multiplicativo de Schwarz, desenvolveram o método aditivo de Schwarz, que é uma abordagem com maior potencial de paralelismo. No método aditivo de Schwarz, os valores calculados nos subdomínios em uma determinada iteração são utilizados como condição de contorno nos subdomínios vizinhos na iteração seguinte. Alguns trabalhos que utilizam o método aditivo de Schwarz são os de (BJORSTAD; SKOGEN, 1992), (PAGLIERI et al., 1997), (SILVA et al., 1997) e (CHARÃO, 2001).

Na década de 1970 outros métodos de decomposição de domínio foram desenvolvidos. Esses métodos são baseados na decomposição do domínio em subdomínios disjuntos, isto é, sem sobreposição, e são chamados de métodos de Schur. Nesta abordagem, as condições de contorno dos subdomínios são obtidas através da resolução de um sistema de equações correspondente às células das fronteiras artificiais.

Métodos de Schur são utilizados em aplicações de simulação numérica e computacional paralelas onde, por exemplo, a malha gerada é não estruturada, ou não ocorre emparelhamento das submalhas entre os subdomínios, ou ainda, quando os diferentes subdomínios possuem diferentes modelos matemáticos. Como exemplo do uso de tais métodos pode-se citar os trabalhos de (BJORSTAD; SKOGEN, 1998), (RIXEN; FARHAT, 1998) e (CHARÃO, 2001).

Nos últimos anos, a abordagem de decomposição de domínio tem sido aplicada em uma grande variedade de problemas como, por exemplo, na solução de EDPs (SMITH, 1992), em mecânica de fluidos computacional (CAI; KEYES; VENKATAKRISHNAN, 1997), (SILVA et al., 1997), problemas não lineares (LUI, 1998), (CAI; KEYES; MARCINKOWSKI, 2002). Uma visão completa das áreas de utilização de métodos de decomposição de domínio, bem como os últimos avanços obtidos nestes métodos, podem ser obtidos no fórum *International Conferences on Domain Decomposition Methods* (*http://www.ddm.org/*), que realiza encontros anuais desde 1987 sobre métodos de decomposição de domínios.

1.3 Organização do Texto

Esse trabalho está organizado em seis capítulos. Nesse capítulo é realizada uma introdução e são apresentadas, ainda, as motivações para o desenvolvimento do mesmo, bem como, trabalhos relacionados.

O capítulo dois apresenta o ambiente computacional de desenvolvimento do trabalho. Nele são abordados aspectos relativos a arquitetura e a ferramentas de programação. Neste contexto, é feita uma breve explanação sobre *clusters* de PCs multiprocessados e sobre as ferramentas utilizadas para a exploração do paralelismo nesse tipo de arquitetura.

No capítulo três é feita uma introdução sobre o desenvolvimento de modelos computacionais. Nesse são apresentados conceitos relativos à solução numérica de EDPs, através da discretização por diferenças finitas. São discutidos, ainda, formatos de armazenamento, bem como métodos numéricos para a resolução dos sistemas de equações gerados pela discretização das EDPs.

No capítulo quatro é feito um estudo sobre a resolução de sistemas de equações em paralelo. Nesse são descritas duas das principais abordagens para se obter paralelismo, que são: a decomposição de dados e a decomposição de domínio, com ênfase nos métodos de decomposição de domínio, pois essa foi a abordagem adotada no desenvolvimento desse trabalho.

No capítulo cinco são descritos os testes realizados com as implementações desenvolvidas e, após, são apresentados e comparados os resultados obtidos.

E, por fim, no capítulo seis, são apresentadas as conclusões do trabalho, incluindo comentários sobre os resultados obtidos com as versões paralelas desenvolvidas. São discutidas ainda, as principais contribuições do trabalho e apresentadas propostas para trabalhos futuros.

2 AMBIENTE DE DESENVOLVIMENTO

As estratégias e implementações propostas nesse trabalho foram desenvolvidas para uma exploração eficiente do paralelismo em *clusters* de PCs multiprocessados. Nesse capítulo são abordados conceitos relativos à arquitetura utilizada, bem como as ferramentas usadas para a exploração do paralelismo nessa arquitetura.

2.1 Computadores Paralelos

Um computador paralelo é formado por um conjunto de processadores que trabalham em conjunto na solução de um determinado problema (FOSTER, 1995). Essa definição é abrangente incluindo, entre outros, supercomputadores paralelos com milhares ou centenas de processadores, computadores multiprocessados e *clusters*.

De acordo com a organização da memória os computadores paralelos podem ser divididos em dois grupos: máquinas com memória compartilhada, onde existe um único espaço de endereçamento e a comunicação entre os processadores é realizada através de um espaço de memória comum, e máquinas com memória distribuída, onde cada processador possui uma memória própria e a comunicação entre os processadores é feita através de uma rede de interconexão (TANENBAUM, 1999), (MOLDOVAN, 1993). A organização da memória e a forma como é feita a comunicação entre os processadores é de fundamental importância para o projeto e desenvolvimento de aplicações paralelas. Nas seções 2.1.1 e 2.1.2 são introduzidos conceitos sobre máquinas com memória compartilhada e máquinas com memória distribuída, respectivamente.

Na seção 2.1.3 são introduzidos conceitos sobre *clusters*, com uma maior ênfase a *clusters* de PCs multiprocessados, ambiente de desenvolvimento desse trabalho. *Clusters* multiprocessados podem ser considerados um sistema com memória híbrida, uma vez que possuem memória compartilhada entre os processadores de cada nodo e memória distribuída entre os nodos do *cluster*. Na seção 2.1.4 é apresentado o *cluster labtec*, um dos disponíveis Instituto de Informática da UFRGS, o qual foi utilizado neste trabalho.

2.1.1 Máquinas com Memória Compartilhada

Em máquinas com memória compartilhada, também chamadas de multiprocessadores, todos os processadores trabalham sobre uma memória comum. Desta forma, a comunicação entre os processadores pode ser feita através de operações de escrita e leitura na memória (TANENBAUM, 1999).

O uso de mémoria compartilhada provê algumas vantagens em relação a memória distribuída, entre as quais, uma transição mais natural de ambientes monoprocessados, menor custo de comunicação e a eliminação da necessidade de tarefas como particiona-

mento de dados e balanceamento de carga.

A forma mais comum de exploração de paralelismo em máquinas com esse tipo de memória é o uso de múltiplas *threads*. Uma *thread* pode ser definida como um fluxo de execução dentro de um processo e consiste em um contador de instruções, um conjunto de registradores e um espaço de pilha (SILBERSCHATZ; GALVIN, 2000).

A utilização de múltiplas *threads* em ambientes multiprocessados é uma maneira eficiente de explorar o paralelismo da arquitetura, uma vez que diferentes *threads* podem ser executadas em diferentes processadores simultaneamente (LEWIS; BERG, 1998). O uso de múltiplas *threads* apresenta vantagens em relação ao uso de múltiplos processos, isto porque as *threads* proporcionam uma maior flexibilidade de gerenciamento, escalonamento e sincronismo se comparadas a processos (PICININ, 2003).

Como já mencionado, múltiplas *threads*, em ambientes multiprocessados, são executadas concorrentemente e compartilham de um único espaço de endereçamento. Problemas podem surgir quando diferentes *threads* de um processo manipulam os mesmos dados (KLEIMAN; SHAH; SMAALDERS, 1996). Esses problemas recebem o nome de condição de corrida e os trechos de um programa que podem causar condições de corrida recebem o nome de seção crítica.

A maneira de eliminar as condições de corrida é bastante simples: basta garantir a exclusão mútua na execução das seções críticas de um programa (TOSCANI; OLIVEIRA; CARISSIMI, 2003). Isto é, quando uma *thread* está executando uma seção crítica as demais *threads* estão proibidas de executar esse trecho do programa (TANENBAUM, 1995). Os recursos mais comuns, disponibilizados pelas bibliotecas de *threads*, para a implementação de áreas de exclusão mútua são: *mutexes*, variáveis de condição e semáforos.

Várias bibliotecas desenvolvidas oferecem suporte para criação e manipulação de *threads*. Entre essas as mais utilizadas para desenvolvimento de aplicações paralelas são a biblioteca *Pthreads* (POSIX *threads*) (LEWIS; BERG, 1998) e a biblioteca OpenMP (OPENMP: SIMPLE, PORTABLE, SCALABLE SMP PROGRAMMING, 2003).

Para o desenvolvimento deste trabalho utilizou-se a biblioteca OpenMP, mais especificamente o nível *parallel region* da biblioteca OpenMP (descrito na seção 2.2.1). Isto porque, em testes realizados, o desempenho obtido com a biblioteca OpenMP, utilizando o nível *parallel region*, foi similar ao desempenho obtido com a biblioteca *Pthreads*. Para realização desses testes utilizou-se o algoritmo do GC na resolução dos sistemas de equações resultantes da discretização de equações diferenciais parciais da hidrodinâmica no modelo HIDRA. As matrizes de coeficientes dos sistemas de equações lineares foram armazenadas no formato CSR. Esse formato é descrito na seção 3.2.3.

Para escolha da biblioteca OpenMP, fez-se testes das implementações do GC desenvolvidas com *Pthreads* e com OpenMP, utilizando-se o Lago Guaíba discretizado com 4 diferentes tamanhos de células: $\triangle x = \triangle y = 200m$, $\triangle x = \triangle y = 100m$, $\triangle x = \triangle y = 50m$ e $\triangle x = \triangle y = 25m$ que resultam, respectivamente, em sistemas de equações com 11.506 equações, 46.024 equações, 184.096 equações e 736.384 equações. Os testes foram realizados utilizando um *Dual Pentium III* 1.1 GHz, com 1 GByte de RAM, com *cache* de 512 KBytes e disco rígido de 18Gbytes. Para a tomada de tempo foram feitas 20 execuções de cada implementação, para então efetuar uma média aritmética dos resultados obtidos.

Na Tabela 2.1 podem ser vistos os resultados obtidos com as implementações desenvolvidas. Observa-se que o desempenho obtido com a biblioteca OpenMP é muito similar ao obtido utilizando a biblioteca *Pthreads*. E baseado nestes resultados optou-

se pela biblioteca OpenMP devido, principalmente, à maior facilidade de utilização. O desenvolvimento de aplicações paralelas em OpenMP é consideravelmente mais simples que na biblioteca *Pthreads*. Além de uma maior facilidade de programação o código fonte em OpenMP é muito mais legível e compacto.

Tabela 2.1: Tempo de Execução - OpenMP x *Pthreads*

Nº de Equações	Seqüencial	*Pthreads*	OpenMP
11.506	0,030048	0,014837	0,013507
46.024	0,155209	0,099174	0,099445
184.096	0,689793	0,445179	0,456486
736.384	2,811246	1,806116	1,865284

Na Tabela 2.2 tem-se a eficiência obtida. Nesta observa-se que a eficiência teve uma queda muito significativa com o aumento da carga computacional. Essa queda é decorrente da competição das *threads* por recursos compartilhados, tais como barramento e memória principal. Enquanto a quantidade de dados for pequena, todas as operações são realizadas utilizando os dados já armazenados em *cache* eliminando a necessidade de acesso à memória principal. Segundo (TANAKA et al., 1998), e como pode ser observado no caso do Guaíba com células de tamanho $\triangle x = \triangle y = 200m$, se a quantidade de dados for suficientemente pequena, de forma a ser armazenada totalmente em *cache*, a eficiência excede a 1. Já com o aumento da quantidade de dados torna-se necessário o acesso a dados armazenados na memória principal, tornando a disputa pelo barramento e o acesso a memória principal um limitante no desempenho das máquinas multiprocessadas. Na seção 5.5 é feito um estudo mais completo sobre a contenção de memória em máquinas multiprocessadas.

Tabela 2.2: Eficiência - OpenMP x *Pthreads*

Nº de Equações	*Pthreads*	OpenMP
11.506	1,0126	1,11231
46.024	0,78251	0,780375
184.096	0,774735	0,755545
736.384	0,778255	0,753575

2.1.2 Máquinas com Memória Distribuída

Em máquinas com memória distribuída, também chamadas de multicomputadores, cada processador possui uma memória própria, que não pode ser acessada de forma direta por outro processador (TANENBAUM, 1999).

A forma mais utilizada para a comunicação em máquinas com memória distribuída é a troca de mensagens. Em aplicações baseadas na troca de mensagens a comunicação entre os processos é feita através do envio e recebimento de mensagens.

A principal limitação no uso do paradigma de troca de mensagens é o alto *overhead* na comunicação e sincronização dos processos. O alto *overhead* dificulta, ao programador, o desenvolvimento de aplicações de alto desempenho, podendo, até mesmo, tornar inviável o uso da troca de mensagens em aplicações com grande dependência entre as operações (PALHA, 2000).

Embora as aplicações que utilizam troca de mensagens sejam próprias para ambientes de memória distribuída, isso não impede que sejam executadas em computadores paralelos com memória compartilhada. Quando são usadas em ambientes de memória compartilhada, as aplicações que utilizam troca de mensagens não aproveitam a principal vantagem desse tipo de sistema, que é o uso de uma memória comum para a comunicação entre os processadores (BUYYA, 1999).

Apesar do alto custo de comunicação (*overhead*), o uso do modelo de troca de mensagens tem crescido muito nos últimos anos. Um dos maiores motivos pela difusão desse modelo é a disponibilidade de um grande número de bibliotecas que oferecem serviços de troca de mensagens. Entre essas pode-se citar: o PVM (*Parallel Virtual Machine*), desenvolvida pelo *Oak Ridge National Laboratory* (GEIST et al., 1994), o MPI (*Message Passing Interface*), padrão definido pelo *MPI Forum* (GROPP; LUSK, 2003), (SNIR et al., 1996), e o DECK (*Distributed Execution and Communication Kernel*), desenvolvido pelo Grupo de Processamento Paralelo e Distribuído (GPPD) do Instituto de Informática da UFRGS (BARRETO, 2000). No desenvolvimento deste trabalho optou-se pelo uso do MPI, descrito na seção 2.2.2, por ser um padrão amplamente utilizado pela comunidade científica no desenvolvimento de aplicações paralelas.

Para alguns autores, um outro problema do paradigma de troca de mensagens é que esse é mais complexo e difícil de programar, quando comparado ao uso de memória compartilhada. Uma alternativa para esse problema é o uso de ferramentas que permitem simular um ambiente de memória compartilhada em ambientes com memória distribuída. Essas ferramentas são chamadas de DSM (*Distributed Shared Memory*) e adicionam custos que podem diminuir o desempenho da aplicação (DREIER; MARKUS; THEO, 1998).

2.1.3 *Clusters* de Computadores

Um *cluster* é uma arquitetura baseada na união de um conjunto de máquinas independentes, interconectados por uma rede de interconexão dedicada e rápida, formando uma plataforma de alto desempenho para execução de aplicações paralelas (BUYYA, 1999). A utilização de *clusters* em aplicações que requerem uma alta capacidade de processamento só é uma realidade devido ao desenvolvimento e barateamento das tecnologias de redes locais e a evolução na capacidade de processamento dos computadores pessoais (DORNELES, 2003).

O uso desse tipo de arquitetura vem tendo um aumento significativo nos últimos anos devido, principalmente, ao seu baixo custo e a escalabilidade da arquitetura. No site *www.top500.org*, de atualização semestral, que lista as 500 máquinas com maior capacidade de processamento do mundo, pode-se observar um número cada vez maior de *clusters*.

As arquiteturas baseadas em *clusters* podem ser homogêneas ou heterogêneas. Um *cluster* é dito homogêneo quando todos os seus nodos possuem mesmas características e a mesma rede de comunicação interligando-os. Um *cluster* é dito heterogêneo quando seus nodos possuem diferentes características ou diferentes redes de comunicação entre grupos de nodos.

A característica de um *cluster* ser ou não homogêneo tem implicações importantes no balanceamento de carga entre os nodos. Em *clusters* homogêneos o balanceamento de carga é mais simples, isto porque, todos os nodos possuem mesmas características (mesma capacidade de processamento, memória, etc) podendo receber cargas iguais. Em *clusters* heterogêneos é necessário que seja feita uma análise mais profunda, e a distribuição

de carga entre os nodos deve levar em consideração a capacidade de processamento de cada nodo, devendo cada nodo receber uma carga de processamento compatível com sua capacidade (DORNELES, 2003).

Os nodos de um *cluster* podem ser monoprocessados ou multiprocessados. Como já mencionado anteriormente, *clusters* com nodos multiprocessados podem ser considerados um sistema de memória hibrída, por utilizarem ao mesmo tempo memória compartilhada e memória distribuída. Em *clusters* multiprocessados existe a possibilidade da exploração do paralelismo intra-nodal em conjunto com a exploração do paralelismo inter-nodal.

Segundo (BUYYA, 1999), em *clusters* multiprocessados, o paralelismo intra-nodal e inter-nodal pode ser explorado das seguintes formas:

- Somente com o uso de ferramentas de troca de mensagens: troca de mensagens são um padrão no desenvolvimento de aplicações para *clusters*. Quando usado em *clusters* multiprocessados, os processadores de um mesmo nodo comunicam-se entre si através do uso de troca de mensagens, não usufruindo das vantagens presentes em ambientes de memória compartilhada.

- Somente com o uso de memória compartilhada: mecanismos DSM, implementados em *software* ou *hardware*, são uma alternativa ao uso de troca de mensagens em *clusters*. Mecanismos DSM permitem simular ambientes de memória compartilhada em ambientes com memória distribuída. A comunicação inter-nodos é implícita ao programador. Esse tipo de mecanismos introduz um custo adicional para o gerenciamento de endereços e coerência de dados.

- Utilização de troca de mensagens em conjunto com memória compartilhada: para uma melhor exploração do paralelismo em *clusters* multiprocessados é adequado explorar as vantagens de ambas as arquiteturas. Nesse caso é conveniente, para a exploração do paralelismo inter-nodos (memória distribuída) o uso de troca de mensagens e para a exploração do paralelismo intra-nodos o uso de *multithreading*.

No desenvolvimento deste trabalho foram analisados, na paralelização dos métodos de decomposição de domínio propostos, duas alternativas: somente o uso de ferramentas de troca de mensagens e a utilização de troca de mensagens em conjunto com memória compartilhada. Mecanismos DSM não foram utilizados devido à indisponibilidade desses no *cluster* utilizado para o desenvolvimento e testes.

Um fator de grande influência na exploração do paralismo inter-nodal é a rede de interconexão dos nodos. Sendo importante, em *clusters* de alto desempenho, que a rede de interconexão seja eficiente. Na seção 2.1.3.1 é feita uma esquematização sobre as tecnologias de redes mais utilizadas, atualmente, em *clusters* para alto desempenho.

2.1.3.1 Rede de Interconexão

A utilização de *clusters* em computação de alto desempenho só é uma realidade graças ao barateamento e desenvolvimento das tecnologias de redes. Nos últimos anos várias tecnologias de rede foram desenvolvidas com o objetivo de diminuir a latência e aumentar a largura de banda. Entre essas as mais utilizadas, atualmente, em *clusters* de alto desempenho, estão:

- *Fast Ethernet*: o padrão (*Fast-Ethernet*), definido pela IEEE 802.3, permite operações *half-duplex* e *full-duplex* a uma taxa de 100 Mbps, sendo muito utilizada em

clusters devido a seu baixo custo. A principal desvantagem dessa tecnologia é que ela implementa as camadas de rede em *software* o que compromete a latência de forma significativa (RIGONI; DIVERIO; NAVAUX, 1999).

- *Myrinet*: é uma tecnologia de rede comutada de alta velocidade, desenvolvida pela empresa americana *Myricon* (*http://www.myri.com*). A *Myrinet* é considerada uma tecnologia de alto desempenho por ter canais robustos de comunicação com controle de fluxo, controle de erro, baixa latência (BODEN et al., 1995). Uma interconexão *Myrinet* é formada por um par de canais *full-duplex* que permitem uma taxa de transmissão de 2 Gbps em cada canal a uma latência inferior a 5 microsegundos (BARCELLOS; GASPARY, 2003). Em uma rede *Myrinet* o mapeamento e estabelecimento de rotas entre os nodos do *cluster* são realizados automaticamente pelo *hardware* da rede. Uma das maiores vantagens dessa tecnologia é a possibilidade de programação da interface de rede. Essas interfaces executam um programa de controle encarregado pelo envio, recebimento e gerencimento dos *buffers* de armazenamento, que pode ser modificado de forma a atender as exigências de uma determinada aplicação. Atualmente, a *Myrinet* é uma das tecnologias de rede mais utilizadas para interconexão em *clusters* de alto desempenho (BARRETO, 2002).

- *Scalable Coherent Interface* (SCI): é um padrão definido pela IEEE 1596 de 1992 que especifica *hardware* e protocolos para conectar até 64K nodos em uma rede de alto desempenho. O padrão SCI permite que a comunicação entre os nodos seja feita através do uso de troca de mensagens ou ainda, implicitamente, através do uso de memória compartilhada. Neste último caso, o *hardware* é capaz de mapear os segmentos físicos de memória de todos os nodos de modo a compor um único segmento de memória (lógico) compartilhada. O padrão SCI define taxas de transmissão de 1 Gbps com latência em torno de 3 microsegundos (BARRETO, 2002).

- *Gigabit Ethernet*: é uma extensão dos padrões *Ethernet* IEEE 802.3 de 10 Mbps (*Ethernet*) e 100 Mbps (*Fast-Ethernet*), tendo sido padronizada em 1998 pelo IEEE (IEEE 802.3z). O padrão Gigabit Ethernet apresenta uma largura de banda de 1 Gbps. Essa tecnologia tem-se mostrado uma boa alternativa, uma vez que fornece uma alta largura de banda a um custo relativamente baixo (BARCELLOS; GASPARY, 2003).

- *Infiniband*: é uma tecnologia de interconexão apoiada por fabricantes como Intel, IBM, Sun, HP e Microsoft, e especifica uma arquitetura de *hardware* para uma interconexão de alta largura de banda e baixa latência entre computadores. Além disso, a *Infiniband* define uma infra-estrutura para interligar os componentes internos do PC (substituindo por exemplo, a tecnologia PCI ou funcionando como um barramento complementar) (RIGHI, 2003). A proposta da arquitetura *Infiniband* é substituir o barramento local, compartilhado, por uma estrutura de interligação baseada em malhas de chaveadores (BARCELLOS; GASPARY, 2003).

Diversas outras tecnologias de rede foram desenvolvidas e testadas em *clusters* de alto desempenho, tais como ATM (*Asynchronous Transfer Mode*), *ServeNet*, *Fiber Channel*, entre outras. Em (BUYYA, 1999) e (BARCELLOS; GASPARY, 2003) são apresentadas essas e outras tecnologias de rede utilizadas em *clusters*.

2.1.4 *Cluster labtec* do Instituto de Informática da UFRGS

Para a avaliação das implementações desenvolvidas utilizou-se o *cluster labtec* do Instituto de Informática da UFRGS. Esse *cluster* faz parte do Laboratório de Tecnologia em *Clusters* (LabTeC), que é um laboratório de pesquisa desenvolvido pelo Instituto de Informática da UFRGS em convênio com a empresa Dell Computadores.

O *cluster labtec* é constituído por 21 nodos, onde 20 desses são dedicados exclusivamente para processamento e 1 nodo é servidor. A interconexão dos nodos de processamento é feita através de um *switch Fast Ethernet*, que é interconectado ao nodo servidor através de uma rede *Gigabit Ethernet*. Além do *switch* que interliga os nodos, o servidor está interconectado a um segundo *switch*, que tem a função de permitir o acesso às demais máquinas do LabTeC, bem como, a máquinas externas ao LabTeC. A topologia do *cluster labtec* é ilustrada na Figura 2.1, retirada de (PICININ, 2003).

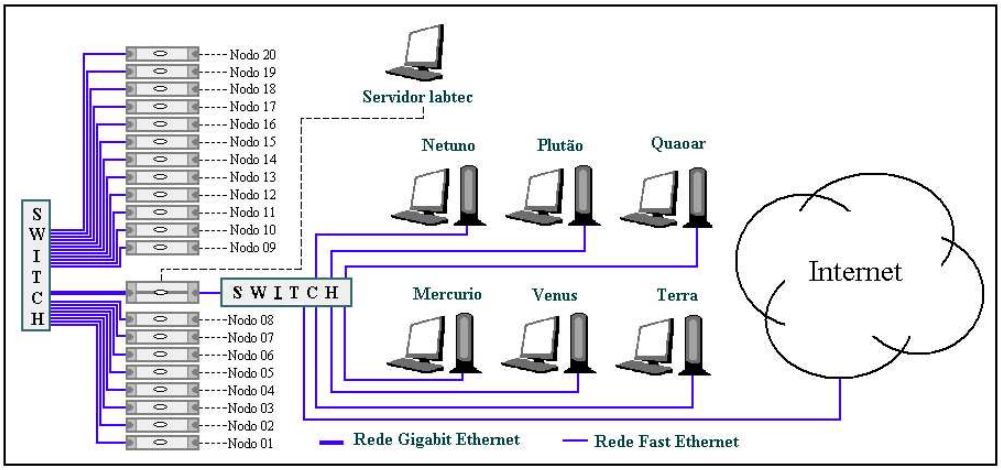

Figura 2.1: Topologia do *cluster labtec* do Instituto de Informática da UFRGS

No que se refere aos nodos desse *cluster*, cada nodo de processamento do *cluster labtec* é composto por uma placa *Dual Pentium III* 1.1 GHz, com 1 GB de memória RAM, 512 KB de *cache*, um disco rígido SCSI com 18 GB e um placa de rede *Gigabit Ethernet*; o nodo servidor é composto por uma placa *Dual Pentium IV Xeon* 1.8 GHz, com 1 GB de memória RAM, um disco rígido SCSI com 36 GB e uma placa de rede *Gigabit Ethernet* (DORNELES, 2003).

2.2 Bibliotecas Utilizadas

Na seção 2.1 foram descritos sistemas de memória de compartilhada e de memória distribuída bem como *cluster* multiprocessados, que podem ser considerados um sistema com memória híbrida. Como já mencionado, em clusters com essas características o paralelismo pode ser explorado em dois níveis: intra-nodal e inter-nodal. Para cada nível existem diferentes bibliotecas que permitem uma exploração eficiente do paralelismo. Nessa seção pretende-se fazer uma discussão sobre as bibliotecas adotadas nesse trabalho para a exploração de ambos os níveis de paralelismo.

2.2.1 OpenMP

OpenMP é uma API (*Aplication Program Interface*) para desenvolvimento de aplicações paralelas que fazem uso de memória compartilhada. Essa foi idealizada por um grupo de grandes fabricantes de *software* e *hardware* e fornece um conjunto de primitivas para criação, manipulação e gerenciamento de *threads* (OPENMP: SIMPLE, PORTABLE, SCALABLE SMP PROGRAMMING, 2003).

Em OpenMP o paralelismo pode ser obtido em dois diferentes níveis. Em um primeiro nível, através da paralelização de laços (*loop level*). Neste nível basta informar ao compilador o início e o fim do código (laços) a ser paralelizado e o número de *threads* que executarão o laço. Nesta forma de paralelização a geração do código paralelo é feita pelo próprio compilador.

O segundo nível (*parallel region*) é mais abrangente, e pode ser utilizado para a execução concorrente de um trecho genérico de código. Nesse nível, tem-se uma programação semelhante à biblioteca *Pthreads* (embora facilitada pelas diretivas do OpenMP) cabendo ao programador o gerenciamento de seções críticas, cuidados com condições de corrida, etc.

Um problema do OpenMP é que ele não garante fazer o melhor uso dos recursos de memória compartilhada. Dessa forma, não é surpreendente ter-se programas em *Pthreads* mais eficientes que o equivalente em OpenMP. Mas isso pode ser compensado pela maior facilidade de programação com o mesmo (OPENMP: SIMPLE, PORTABLE, SCALABLE SMP PROGRAMMING, 2003).

A biblioteca OpenMP possui implementações em diversos sistemas operacionais, entre os quais, Linux e Windows, sendo largamente portável. Essa biblioteca possui distribuições para as linguagens C, C++ e Fortran. Para o desenvolvimento deste trabalho utilizou-se o pré-compilador Omni de domínio público e desenvolvido pelo grupo de pesquisa *Tsukuba Research Center* do *Real World Computing Partnership* (RWCP) (OMNI OPENMP COMPILER PROJECT, 2003).

2.2.2 MPI (*Message Passing Interface*)

MPI é a especificação de um padrão para uma biblioteca de troca de mensagens desenvolvido pelo MPI Forum, um grupo composto por representantes das maiores empresas fabricantes de máquinas paralelas, de orgãos governamentais e universidades, principalmente da Europa e Estados Unidos (PACHECO, 1997).

O padrão MPI especifica a sintaxe e a semântica para 125 funções, divididas entre primitivas de gerência, primitivas de comunicação ponto a ponto e primitivas para comunicação coletiva. Serão descritas apenas as funções básicas e as funções coletivas mais utilizadas no desenvolvimento de aplicações paralelas envolvendo álgebra linear computacional. Para um estudo completo sobre o MPI veja (PACHECO, 1997) e (SNIR et al., 1996).

As primitivas de gerenciamento incluem primitivas para a inicialização e finalização do ambiente MPI, *MPI_Init* e *MPI_Finalize*, respectivamente. Além disso, incluem primitivas que possibilitam a obtenção do número total de processos de uma aplicação e do identificador do processo local, *MPI_Comm_size* e *MPI_Comm_rank* respectivamente.

A comunicação ponto a ponto é o mecanismo básico de troca de mensagens entre processos no MPI. Este tipo de comunicação envolve apenas dois processos, um processo emissor e um processo receptor. Neste tipo de comunicação o envio de mensagens é feito através da primitiva *MPI_Send* e o recebimento através da primitiva *MPI_Recv*.

As primitivas *MPI_Send* e *MPI_Recv* permitem realizar a comunicação síncrona entre os processos. O MPI, além de primitivas de comunicação sícrona, disponibiliza, ainda, primitivas que permitem a comunicação assíncrona entre processos, *MPI_Isend* e *MPI_Irecv*. O uso de primitivas assíncronas possibilita a utilização de sobreposição de comunicação e processamento.

As primitivas de comunicação coletiva permitem a troca de mensagens entre todos os processos de uma aplicação. Entre essas primitivas destacam-se o *MPI_Bcast*, na qual um processo envia uma mensagem a todos os demais processos, o *MPI_Gather*, onde um único processo coleta uma estrutura distribuída em todos os processos, o *MPI_Scatter*, que distribui uma estrutura de dados, armazenada em único processo, entre todos os processos, o *MPI_Reduce*, que realiza uma operação pré-definida sobre dados de todos os processos e envia o resultado para um único processo, e o *MPI_Barrier*, que sincroniza todos os processos. As primitivas de comunicação coletiva devem ser invocadas em todos os processos de um grupo com os mesmos argumentos (SNIR et al., 1996)

Existe uma série de bibliotecas de troca de mensagens que implementam o padrão MPI, algumas de código proprietário, como por exemplo o HP MPI, o SGI MPI e NEC MPI, e outras de domínio público, como por exemplo o MPICH (GROPP; LUSK, 2003), desenvolvido pela *Argonne National Laboratory* e pela *Mississipi State University*, e o MPI/LAM (LUMSDAINE et al., 2003), desenvolvido pelo *Ohio Supercomputer Center*. Neste trabalho utilizou-se a biblioteca MPICH.

Até o momento, as implementações existentes, que são baseadas no padrão 1.0, não fornecem suporte ao uso de múltiplas *threads*. Para superar essa limitação estão sendo desenvolvidas implementações baseadas no Padrão 2.0, que especifica o uso de primitivas MPI em múltiplas *threads* (PICININ, 2003).

2.3 Considerações Finais

Neste capítulo apresentou-se o ambiente de desenvolvimento do trabalho. Inicialmente foram apresentados conceitos relativos à arquitetura utilizada (*clusters de PCs multiprocessados*) e as características desse tipo de arquitetura. Para maiores informações sobre arquiteturas paralelas e *clusters* de computadores recomenda-se (DE ROSE; NAVAUX, 2003) e (BUYYA, 1999).

Em *clusters* formados por máquinas multiprocessadas o paralelismo pode ser explorado em dois níveis: intra-nodos e inter-nodos. Neste trabalho, a exploração do paralelismo intra-nodal é feita através do uso de múltiplas *threads* (*multithreading*) e a exploração do paralelismo inter-nodal através de troca de mensagens.

Para a exploração de cada nível de paralelismo existem diferentes bibliotecas, que proporcionam primitivas para facilitar o desenvolvimento de aplicações. Neste trabalho utilizou-se a biblioteca de *threads* OpenMP e e a biblioteca de troca de mensagens MPI. Maiores informações sobre o OpenMP podem ser encontradas na página do projeto (*http://www.openmp.org/*). E para maiores informações sobre o MPI recomenda-se (PACHECO, 1997) e (SNIR et al., 1996).

No próximo capítulo são descritas as características da aplicação desenvolvida. Nesse, é feita uma breve explanação sobre EDPs e sobre como a discretização dessas equações pode gerar sistemas de equações lineares. São discutidos ainda as características desses sistemas bem como os métodos numéricos que podem ser utilizados na resolução dos mesmos.

3 MODELO COMPUTACIONAL

Fenômenos físicos podem ser modelados através de equações diferenciais parciais (EDPs), que são definidas sobre domínios contínuos e para poderem ser tratadas computacionalmente necessitam ser discretizadas. A discretização das EDPs pode ser feita de modo a gerar sistemas de equações. Dependendo das aproximações empregadas para discretizar os termos da EDP e do porte do domínio esses sistemas podem ser de grande porte e esparsos. A solução desses sistemas pode ser obtida através de métodos diretos ou, mais freqüentemente, através de métodos iterativos.

O objetivo desse capítulo é apresentar como a discretização de equações diferenciais resulta em sistemas de equações. Nele é feita uma introdução às equações diferenciais parciais e à discretização dessas através do método de diferenças finitas.

São abordados, ainda, métodos que podem ser utilizados na resolução dos sistemas de equações oriundos da discretização das equações diferenciais parciais. E, por fim, são apresentadas algumas bibliotecas de métodos numéricos que podem ser utilizadas na resolução desses sistemas.

3.1 Equações Diferenciais Parciais

Para realização da simulação computacional de um determinado fenômeno físico é necessário representá-lo através de um modelo matemático, modelo esse que, em geral, considera apenas os aspectos essenciais do fenômeno em questão. A maioria desses fenômenos são expressos, matematicamente, por equações diferenciais parciais (EDPs) (EVANS, 1998). Equações diferenciais são equações que envolvem a função incógnita e as suas derivadas. Uma equação diferencial é dita ordinária se depende de apenas uma variável independente, e dita parcial se depende de duas ou mais variáveis independentes (PICININ, 2001).

Na especificação de um modelo matemático para um fenômeno físico, além da definição das EDPs, é necessário determinar as condições de contorno (CC) e as condições iniciais (CI) do problema. As CC especificam o que acontece nas fronteiras do domínio de definição e as CI informam o estado inicial do sistema. São as CC e CI que garantem a unicidade da solução (IÓRIO JÚNIOR; IÓRIO, 1988), (FORTUNA, 2000).

As EDPs não apresentam, em geral, uma solução analítica para problemas realistícos, sendo necessário o uso de algum método de discretização, como, por exemplo, diferenças finitas ou volumes finitos, para obter uma aproximação da solução. O modelo resultante da aproximação para o modelo matemático é, usualmente, chamado de modelo discreto.

3.2 Solução Numérica das EDPs

Como já mencionado, EDPs trabalham sobre domínios com um número infinito de pontos, e para serem tratadas computacionalmente é necessário que o domínio físico e as expressões envolvidas sejam discretizados. Nas seções 3.2.1 e 3.2.2 são discutidos, respectivamente, o processo de discretização do domínio e a discretização de EDPs através do método de diferenças finitas, técnica utilizada na discretização das equações da hidrodinâmica no modelo HIDRA.

3.2.1 Discretização do Domínio Computacional

Para obter a solução numérica de uma EDP em um determinado domínio físico é necessário, primeiramente, que o domínio físico seja discretizado, isto é, seja dividido em um número finito de pontos. A solução será obtida somente nesses pontos. Esse conjunto discreto de pontos recebe o nome de malha (HIRSCH, 1992), (FORTUNA, 2000).

Para a geração dos pontos de uma malha, o domínio físico é dividido em células (triângulos, quadriláteros, tetraedros ou paralelepípedos), como pode ser visto na Figura 3.1 (JUSTO, 1997).

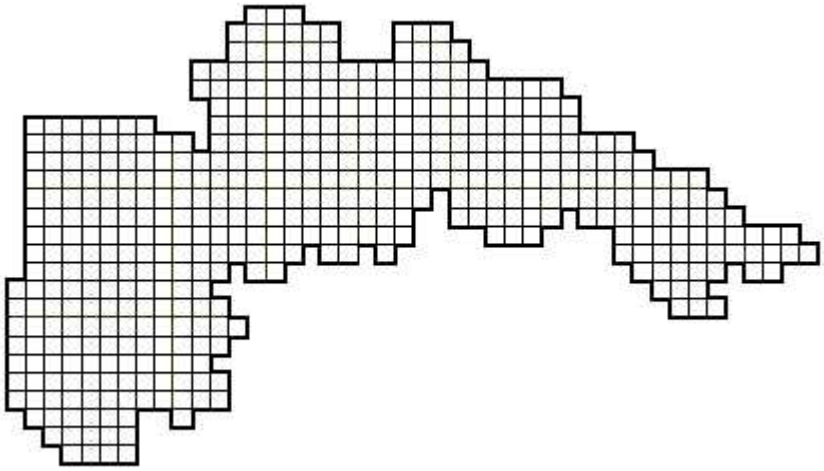

Figura 3.1: Lago Guaíba discretizado (aproximação grosseira)

De acordo com a sua "topologia"as malhas podem ser divididas em: malhas estruturadas e malhas não estruturadas. Malhas estruturadas são assim chamadas pois apresentam uma estrutura, ou regularidade, na distribuição dos pontos. Já malhas não estruturadas são assim chamadas devido à ausência de regularidade na disposição dos pontos (FORTUNA, 2000). As malhas não estruturadas são mais genéricas e apropriadas para domínios com geometrias complexas (MALISKA, 1995).

Em geral, malhas estruturadas são associadas a métodos de discretização do tipo diferenças finitas e malhas não estruturadas a métodos de discretização do tipo elementos finitos. No modelo HIDRA são utilizadas malhas estruturadas associadas a um método de diferenças finitas. A escolha por malhas estruturadas, no modelo HIDRA, deu-se devido à simplicidade quanto à definição das estruturas de dados (RIZZI, 2002) e diferenças finitas por ser um método suficientemente adequado para o problema modelado no HIDRA (DORNELES, 2003). Além disso, comparativamente ao método dos elementos finitos,

o instrumental matemático requerido pelo métodos das diferenças finitas é extremamente simples.

3.2.2 Método de Diferenças Finitas

O método de diferenças finitas é um dos mais antigos aplicados na obtenção da solução numérica de EDPs, sendo utilizado até hoje em uma gama de problemas. Em (FORTUNA, 2000) e (CUMINATO; MENEGUETTE, 1999), além de outros, encontra-se um material amplo e acessível sobre métodos de diferenças finitas e suas aplicações.

Para utilização deste método é gerada uma malha sobre todo o domínio computacional do problema, a qual contém os pontos onde são efetuadas as aproximações dos termos presentes na EDP. Para cada ponto da malha obtém-se uma equação algébrica que aproxima a EDP nesse ponto. Essa equação é denominada equação de diferenças finitas (EDF). É importante destacar que quanto mais refinada for a malha, isto é, quanto maior o número de pontos da malha, mais acurada será a aproximação da solução. Em contrapartida, maior será o número de equações algébricas a serem resolvidas (MALISKA, 1995).

A idéia do método de diferenças finitas é simples. Como exemplo, considera-se a derivada de uma função $f(x_i)$, suposta ser diferenciável, no ponto x_i, que é definida pela equação 3.1.

$$\frac{df}{dx}(x_i) = \lim_{\triangle x \to 0} \frac{f(x_i + \triangle x) - f(x_i)}{\triangle x} \tag{3.1}$$

Se a distância $\triangle x$ (finita) for suficientemente pequena, o resultado da equação 3.1 é uma aproximação para o valor de $f'(x_i)$. Esta aproximação pode ser melhorada através da redução do valor de $\triangle x$ (CARDOSO, 2001).

A forma mais comum para a definição de aproximações para as derivadas é a série de Taylor. Através dessa, uma aproximação do valor de $f'(x_i)$ pode ser obtida através da expansão da série de Taylor de $f(x_i + \triangle x)$ em torno do ponto x_i (equação 3.2), ou ainda, a expansão da série de Taylor de $f(x_i - \triangle x)$ em torno do ponto x_i (equação 3.3) (HIRSCH, 1992).

$$f(x_i + \triangle x) = f(x_i) + (\triangle x)\left.\frac{\partial f}{\partial x}\right|_i + \frac{(\triangle x)^2}{2!}\left.\frac{\partial^2 f}{\partial x^2}\right|_i + \frac{(\triangle x)^3}{3!}\left.\frac{\partial^3 f}{\partial x^3}\right|_i + ... \tag{3.2}$$

$$f(x_i - \triangle x) = f(x_i) - (\triangle x)\left.\frac{\partial f}{\partial x}\right|_i + \frac{(\triangle x)^2}{2!}\left.\frac{\partial^2 f}{\partial x^2}\right|_i - \frac{(\triangle x)^3}{3!}\left.\frac{\partial^3 f}{\partial x^3}\right|_i + ... \tag{3.3}$$

Uma aproximação para a derivada primeira da função $f(x_i)$ no ponto x_i, pode ser obtida isolando o termo $f'(x_i)$ nas equações 3.2 e 3.3. Desta forma, essas equações podem ser reescritas pelas equações 3.4 e 3.5, respectivamente. Observa-se que nas equações 3.4 e 3.5 o erro de truncamento é de ordem $O(\triangle x)$. Esse erro aparece devido à existência de um número infinito de termos na série de Taylor. Como é, computacionalmente, impraticável o emprego de um número infinito de termos, é necessário que a série seja truncada (ARAUJO, 2002). Nas equações 3.4 e 3.5 a série foi truncada a partir da derivada de segunda ordem.

$$\left.\frac{\partial f}{\partial x}\right|_i = \frac{f(x_i + \triangle x) - f(x_i)}{\triangle x} + O(\triangle x) \tag{3.4}$$

$$\left.\frac{\partial f}{\partial x}\right|_i = \frac{f(x_i) - f(x_i - \triangle x)}{\triangle x} + O(\triangle x) \tag{3.5}$$

A notação das equações 3.4 e 3.5 pode ser simplificada substituindo $f(x_i)$ por f_i e $f(x_i \pm k\triangle x)$ por $f_{i\pm k}$. Desta forma, as equações 3.4 e 3.5 podem ser reescritas como as equações 3.6 e 3.7, respectivamente.

$$\left.\frac{\partial f}{\partial x}\right|_i = \frac{f_{i+1} - f_i}{\triangle x} + O(\triangle x) \tag{3.6}$$

$$\left.\frac{\partial f}{\partial x}\right|_i = \frac{f_i - f_{i-1}}{\triangle x} + O(\triangle x) \tag{3.7}$$

A equação 3.4 é considerada uma aproximação por diferenças progressivas, isto porque, utiliza o ponto x_{i+1}, que é um ponto *à frente* de x_i, enquanto a equação 3.5 é considerada uma aproximação por diferenças regressivas, pois utiliza o ponto x_{i-1}, que é um ponto *atrás* de x_i (FORTUNA, 2000).

Para obter uma aproximação de $O(\triangle x)^2$ para a primeira derivada de $f(x_i)$, basta manipular convenientemente as expansões em série de Taylor das equações 3.2 e 3.3. Como deseja-se uma aproximação de $O(\triangle x)^2$, é necessário combinar essas equações de forma a eliminar a segunda derivada de $f(x_i)$. Eliminando-se essa derivada obtém-se a equação 3.8

$$\left.\frac{\partial f}{\partial x}\right|_i = \frac{f_{i+1} - f_{i-1}}{2\triangle x} + O(\triangle x)^2 \tag{3.8}$$

Nota-se que a aproximação da primeira derivada de $f(x_i)$, dada pela equação 3.8, é calculada utilizando os pontos x_{i+1} e x_{i-1}, um ponto *à frente* e um ponto *atrás* de x_i. Por essa razão essa aproximação é denominada de aproximação por diferenças centradas (FORTUNA, 2000).

Ainda utilizando as equações 3.2 e 3.3 é possível combiná-las de forma a obter a segunda derivada da função $f(x_i)$. Para isso basta combinar as equações de forma a eliminar a primeira derivada, obtendo a equação 3.9. Essa equação é a mais encontrada na literatura para derivadas de segunda ordem (MALISKA, 1995).

$$\left.\frac{\partial^2 f}{\partial x^2}\right|_i = \frac{f_{i+1} - 2f_i + f_{i-1}}{(\triangle x)^2} + O(\triangle x)^2 \tag{3.9}$$

É importante salientar que através da manipulação de expansões de uma função em séries de Taylor é possível representar derivadas de qualquer ordem. Logicamente, quanto maior for a ordem da derivada mais pontos são necessários em torno de x_i (MALISKA, 1995).

3.2.2.1 Estêncil Computacional

O estêncil computacional, ou molécula, de uma aproximação indica a posição dos pontos presentes em uma EDF. Isso permite visualizar a dependência entre os pontos da malha. Por exemplo, considere a equação da difusão bidimensional, dada pela equação 3.10, onde f é a temperatura, x e y são coordenadas espaciais, t o tempo e α_x e α_y são coeficientes de difusividade térmica do material (PICININ, 2001).

$$\frac{\partial f}{\partial t} = \alpha_x \frac{\partial^2 f}{\partial x^2} + \alpha_y \frac{\partial^2 f}{\partial y^2} \tag{3.10}$$

A solução numérica da equação envolve tanto a discretização espacial, como a discretização temporal. A equação 3.11 apresenta a equação da difusão discretizada no espaço, em malha uniforme, por diferenças centrais de segunda ordem. A discretização temporal será tratada na seção 3.2.2.2.

$$\frac{\partial f}{\partial t} \cong \alpha_x \frac{f_{i+1,j} - 2f_{i,j} + f_{i-1,j}}{(\triangle x)^2} + \alpha_y \frac{f_{i,j+1} - 2f_{i,j} + f_{i,j-1}}{(\triangle y)^2} \tag{3.11}$$

A Figura 3.2 apresenta a molécula computacional, ou estêncil, da equação da difusão bidimensional. Ela é uma representação gráfica da equação 3.11, pois estabelece a relação de dependência existente entre o ponto (i, j) e seus vizinhos. Note que o ponto (i, j) depende dos pontos $(i+1, j)$, $(i, j+1)$, $(i-1, j)$ e $(i, j-1)$ (AMES, 1977), (FORTUNA, 2000). Devido aos pontos que utiliza, essa aproximação é chamada de estêncil de cinco pontos (ou 5-pontos), e é a molécula computacional utilizada na discretização espacial das equações da hidrodinâmica no modelo HIDRA.

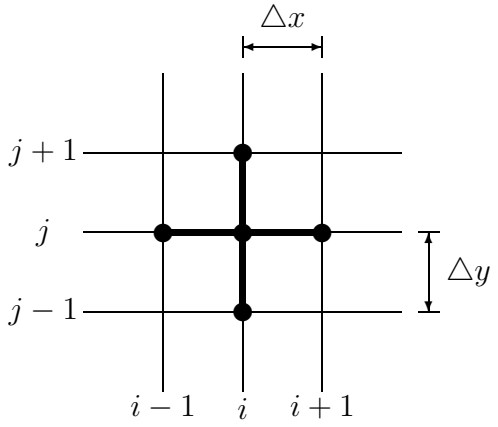

Figura 3.2: Estêncil de cinco pontos

3.2.2.2 Discretização Temporal

Fenômenos físicos podem ser divididos em dois grandes grupos: estacionários e transientes. Fenômenos estacionários são aqueles que não evoluem no tempo, isto é, fenômenos que estão em estado de equilíbrio. Desta forma, fenômenos estacionários necessitam de uma única solução. Já fenômenos transientes são aqueles que evoluem no tempo, isto é, fenômenos que necessitam do cálculo da solução em instantes sucessivos de tempo (FORTUNA, 2000)

Em fenômenos transientes como, por exemplo, o fenômeno da difusão bidimensional (equação 3.10), além da discretização espacial, o tempo também deve ser discretizado, isto é, dividido em passos, e para cada passo de tempo as EDFs devem ser resolvidas (DORNELES, 2003). Nessa seção são introduzidos conceitos relativos a discretização temporal de EDPs transientes. Para diferenciação dos termos das EPDs nas coordenadas

temporais e espaciais denota-se por $f_{i,j}^n$ o valor da aproximação da solução no ponto $x_{i,j}$ e no nível de tempo t^n.

A discretização temporal visa fornecer relações entre as soluções em instantes sucessivos. Uma questão importante surge na escolha do tipo de esquema temporal utilizado para a aproximação da derivada temporal. Aproximando a derivada temporal da equação 3.10 por diferenças progressivas de primeira ordem obtém-se a equação 3.12.

$$\left.\frac{\partial f}{\partial t}\right|_i^n = \frac{f_{i,j}^{n+1} - f_{i,j}^n}{\triangle t} + O(\triangle t) \tag{3.12}$$

Substituindo a equação 3.12 em 3.11 obtém-se a equação 3.13. Observa-se que o valor em um ponto $x_{i,j}$ no nível de tempo $n+1$ depende apenas dos valores na células vizinhos no passo de tempo anterior, isto é, no nível de tempo n. Essa formulação é conhecida como método explícito e dá origem a um conjunto de equações algébricas desacoplas. Assim, essas equações podem ser resolvidas uma a uma, não havendo necessidade de resolver um sistema de equações (MALISKA, 1995).

$$\frac{f_{i,j}^{n+1} - f_{i,j}^n}{\triangle t} \cong \alpha_x \frac{f_{i+1,j}^n - 2f_{i,j}^n + f_{i-1,j}^n}{(\triangle x)^2} + \alpha_y \frac{f_{i,j+1}^n - 2f_{i,j}^n + f_{i,j-1}^n}{(\triangle y)^2} \tag{3.13}$$

A derivada temporal da equação 3.10 pode ser aproximada, ainda, por diferenças regressivas. Neste caso obtém-se a equação 3.14.

$$\left.\frac{\partial f}{\partial t}\right|_i^n = \frac{f_{i,j}^n - f_{i,j}^{n-1}}{\triangle t} + O(\triangle t) \tag{3.14}$$

Substituindo a equação 3.14 em 3.11 obtém-se a equação 3.15. Nota-se que, neste caso, os valores em um ponto $x_{i,j}$ no nível de tempo n dependem dos valores das células vizinhas no mesmo passo de tempo pois somente um termo em $n-1$ é conhecido. Essa formulação é conhecida como método implícito, e dá origem a um conjunto de equações acopladas. Desta forma, essa formulação, exige a resolução de um sistema de equações a cada passo de tempo (MALISKA, 1995), (FORTUNA, 2000).

$$\frac{f_{i,j}^n - f_{i,j}^{n-1}}{\triangle t} \cong \alpha_x \frac{f_{i+1,j}^n - 2f_{i,j}^n + f_{i-1,j}^n}{(\triangle x)^2} + \alpha_y \frac{f_{i,j+1}^n - 2f_{i,j}^n + f_{i,j-1}^n}{(\triangle y)^2} \tag{3.15}$$

Métodos explícitos apresentam fortes restrições na escolha do tamanho do passo de tempo, isto porque, a estabilidade desse tipo de método é dependente de uma relação entre os tamanhos dos passos utilizados na discretização das variáveis temporais e espaciais da EDP (CUNHA, 2000). Já métodos implícitos, apesar de exigirem a resolução de um sistema de equações a cada passo de tempo são mais estáveis e, em geral, não possuem problemas quanto ao passo de tempo, possibilitando a utilização de um passo de tempo maior (FORTUNA, 2000), (RIZZI, 1999).

É importante destacar que pode ocorrer uma combinação entre métodos explícitos e implícitos, formando um esquema chamado semi-implícito. Neste tipo de esquema alguns termos da EDP são aproximados de modo implícito e outros termos de modo explícito. Maiores informações sobre métodos semi-implícitos podem ser obtidas em (RIZZI, 1999).

3.2.3 Esparsidade e Estrutura das Matrizes Geradas pela Discretização

Um fator de importância na estrutura das matrizes de coeficientes dos sistemas de equações gerados pela discretização de EDPs é a malha do domínio onde o fenômeno é resolvido. Isto porque a malha do domínio físico tem influência direta na disposição dos elementos no sistema de equações gerado.

Por exemplo, a discretização em diferenças finitas sobre malhas estruturadas de domínios retangulares gera matrizes estruturadas e regulares, tendo forma de bloco diagonal principal. Já a discretização em domínios não retangulares, como o Lago Guaíba utilizado para estudo de caso, resulta em matrizes estruturadas e irregulares (RIZZI, 2002). Mais especificamente no caso de domínios irregulares, empregando uma molécula computacional de 5-pontos, as matrizes de coeficientes dos sistemas de equações gerados são irregulares e esparsas, contendo no máximo 5 elementos por linha.

A esparsidade da matriz de coeficientes dos sistemas de equações gerados pela discretização possibilita o uso de técnicas especiais de armazenamento. Essas técnicas baseiam-se na idéia de armazenar somente os elementos não nulos da matriz, possibilitando uma significativa economia de memória.

Além disso, o uso de técnicas de armazenamento, freqüentemente, otimiza as operações de álgebra linear que envolvem a matriz de coeficientes do sistema. Como por exemplo, no caso da multiplicação de uma matriz esparsa por vetor, não é necessário varrer todos os elementos da matriz, mas somente os elementos não nulos (DEMMEL; HEATH; VORST, 1993). Em contrapartida, o uso de técnicas de armazenamento pode resultar em algoritmos e estruturas de dados mais complexos.

Existem diversas técnicas para armazenamento de matrizes esparsas, para um estudo mais completo veja (BARRETT et al., 1994), (JÚDICE; PATRÍCIO, 1996) e (SAAD, 1996). Para o desenvolvimento deste trabalho utilizou-se o formato CSR (*Compressed Sparse Row*). Esse formato é um dos mais efetivos, atualmente, para o armazenamento de matrizes irregulares, esparsas e de grande porte (JÚDICE; PATRÍCIO, 1996).

O formato CSR, um exemplo do qual pode ser visto na Figura 3.3, é baseado em três vetores: um vetor (*val*) do tipo do dado da matriz e dois vetores (*col*, *ptr*) de inteiros, onde:

- *val*: contém os valores não nulos da matriz da esquerda para a direita e de cima para baixo. O tamanho do vetor val é dado pelo número de elementos não nulos da matriz.
- *col*: contém a coluna da qual os elementos contidos em *val* foram obtidos. O tamanho de col é dado pelo número de elementos não nulos da matriz.
- *ptr*: contém ponteiros para o início de cada linha da matriz nos vetores *val* e *col*. O tamanho do vetor *ptr* é dado por $N + 1$, onde N é o número de linhas da matriz.

Se todos os elementos da diagonal da matriz de coeficientes forem não nulos, como por exemplo, no caso dos sistemas gerados pela discretização do modelo de hidrodinâmica, é possível alterar o formato CSR colocando a diagonal principal em um vetor separado. Essa alteração diminui o total de elementos armazenados no vetor de índices (*col*), bem como, pode otimizar operações que envolvam a diagonal principal da matriz de coeficientes. Como exemplo de operações onde o armazenamento da diagonal da matriz em separado possibilita uma implementação mais simples e eficiente pode-se citar técnicas de fatoração incompleta e técnicas de aproximação polinomial.

$$A = \begin{bmatrix} a_1 & 0 & 0 & a_2 & 0 & 0 & 0 & 0 \\ 0 & a_3 & a_4 & 0 & 0 & a_5 & 0 & 0 \\ 0 & a_6 & a_7 & a_8 & 0 & 0 & a_9 & 0 \\ a_{10} & 0 & a_{11} & a_{12} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & a_{13} & a_{14} & 0 & 0 \\ 0 & a_{15} & 0 & 0 & a_{16} & a_{17} & a_{18} & a_{19} \\ 0 & 0 & a_{20} & 0 & 0 & a_{21} & a_{22} & 0 \\ 0 & 0 & 0 & 0 & 0 & a_{23} & 0 & a_{24} \end{bmatrix}$$

$$val = \begin{bmatrix} a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 & a_8 & a_9 & a_{10} & a_{11} & a_{12} \\ a_{13} & a_{14} & a_{15} & a_{16} & a_{17} & a_{18} & a_{19} & a_{20} & a_{21} & a_{22} & a_{23} & a_{24} \end{bmatrix}$$

$$col = \begin{bmatrix} 0 & 3 & 1 & 2 & 5 & 1 & 2 & 3 & 6 & 0 & 2 & 3 & 4 & 5 & 1 & 4 & 5 & 6 & 7 & 2 & 5 & 6 & 5 & 7 \end{bmatrix}$$

$$ptr = \begin{bmatrix} 0 & 2 & 5 & 9 & 12 & 14 & 19 & 22 & 24 \end{bmatrix}$$

Figura 3.3: Formato CSR

3.3 Resolução de Sistemas de Equações Lineares

Como visto na seção 3.2.2.2, a discretização de EDPs pode ser feita de maneira que sejam gerados sistemas de equações lineares, que necessitam ser resolvidos a cada passo de tempo. Existem, basicamente, duas classes de métodos que podem ser aplicados na resolução desses sistemas de equações: os métodos diretos e os métodos iterativos (embora existam, ainda, métodos híbridos que procuram combinar ambas as classes). Nas seções 3.3.1 e 3.3.2 são discutidos métodos diretos e métodos iterativos, respectivamente.

3.3.1 Métodos Diretos

Métodos diretos caracterizam-se por fornecer a solução exata do sistema de equações, exceto por erros de arredondamento introduzidos pelo computador devido a aritmética de ponto flutuante, após um número finito e conhecido de passos (CUNHA, 2000), (RIZZI, 2002). São exemplos de métodos diretos o método de Eliminação Gaussiana e a Fatoração LU, dentre outros.

Métodos diretos possuem a vantagem de serem mais gerais e robustos que os métodos iterativos, podendo ser utilizados na resolução, sem restrições, de qualquer tipo de sistema de equações (KUMAR et al., 1993). Apesar dessas vantagens, métodos diretos são inadequados para a resolução de sistemas oriundos da discretização de EDPs, uma vez que não aproveitam a esparsidade da matriz de coeficientes do sistema. Métodos diretos baseiam-se em operações elementares sobre linhas e colunas da matriz de coeficientes, destruindo a esparsidade da mesma. Isso aumenta tanto o espaço necessário para o armazenamento da matriz de coeficientes, quanto o esforço computacional para o cálculo da solução numérica do sistema. Desta forma, o emprego de métodos diretos fica restrito a sistemas de pequena ordem ($N \leq 5000$) (MÜLLER; CHARÃO; SANTOS, 2003).

Além disso, uma outra desvantagem de métodos diretos é que esses apresentam grande dependência entre as operações, o que torna difícil a sua paralelização (PICININ, 2001).

3.3.2 Métodos Iterativos

Métodos iterativos são muito utilizados na resolução de sistemas de equações esparsos e de grande porte, devido a sua potencialidade quanto à otimização de armazenamento e sua eficiência computacional. Esses baseiam-se em sucessivas aproximações para obter

uma solução cada vez mais acurada do sistema de equações. Ao contrário dos métodos diretos, que resolvem o sistema de equações com um número pré-definido de passos, os métodos iterativos calculam a solução até que um determinado critério de parada seja alcançado. Esse critério pode ser um número máximo de iterações ou uma tolerência mínima de erro, definida pela norma do resíduo (RIZZI, 2002).

Os métodos iterativos podem ser divididos, ainda, em duas classes que são: os métodos estacionários e os métodos não estacionários. Os métodos estacionários, também chamados de métodos clássicos, caracterizam-se por não utilizarem informações de iterações anteriores nos seus algoritmos. Esses métodos são mais simples de entender e para implementar, mas, em geral, são menos eficientes que os métodos não-estacionários na resolução de sistemas de grande porte (SLAVIERO, 1997). Entre os métodos estacionários os mais conhecidos são: o método de Jacobi, o método de Gauss-Seidel e o método SOR.

Os métodos não estacionários caracterizam-se por utilizarem informações das iterações anteriores nos seus algoritmos. Esses métodos são mais sofisticados e adequados na resolução de sistemas de equações de grande porte (PICININ, 2003).

Os métodos não-estacionários incluem aqueles baseados no subespaço de Krylov, como o método do Gradiente Conjugado (GC) e o método de Resíduo Mínimo Generalizado (GMRES). Os métodos no subespaço de Krylov tornaram-se populares nas últimas décadas devido a sua robustez e eficiência computacional, principalmente em ambientes paralelos. Essa eficiência decorre do fato de que esses métodos são construídos sobre operações básicas em álgebra linear e, portanto, são altamente paralelizáveis.

A principal desvantagem dos métodos iterativos é que a convergência desses métodos é altamente dependente do condicionamento da matriz de coeficientes. No caso de sistemas de equações mal condicionados, um método iterativo pode tornar-se extremamente lento ou até mesmo falhar. Para sistemas de equações mal condicionados é recomendado o uso de pré-condicionadores, que são técnicas que transformam o sistema de equações em outro com mesma solução, porém, com propriedades mais favoráveis à convergência (MARTINOTTO, 2002).

Para a resolução dos sistemas de equações gerados pelo modelo de hidrodinâmica, já discreto, utilizou-se o método do gradiente conjugado (GC), uma vez que as matrizes de coeficientes desses sistemas são esparsas, de grande porte e simétricas e definidas-positivas (SDP). O GC é um dos métodos mais eficientes entre os métodos iterativos do subespaço de Krylov para a resolução de sistemas de equações com essas características. Nas seções 3.3.2.1 e 3.3.2.2 são abordados, respectivamente, conceitos relativos aos métodos no subespaço de Krylov e ao método do gradiente conjugado.

3.3.2.1 Métodos do Subespaço de Krylov

Na resolução de $Ax = b$, os métodos do subespaço de Krylov, encontram uma solução aproximada x^m para um subespaço m-dimensional, tal que:

$$x^m = x^0 + K^m \tag{3.16}$$

Por imposição da condição de Petrov-Galerkin na equação 3.16 obtém-se:

$$b - Ax^m \perp L^m \tag{3.17}$$

onde L^m é um subespaço de dimensão m. Um método do subespaço de Krylov é definido através da equação 3.18, onde $r^0 = b - Ax^0$ (SAAD, 1996).

$$K^m(A, r^0) = span\{r^0, Ar^0, A^2r^0, ..., A^{m-1}r^0\} \tag{3.18}$$

Dependendo da escolha de L^m os métodos no subespaço de Krylov podem ser divididos em três categorias:

- $L^m = K^m$: quando a matriz A é SDP essa escolha de L^m minimiza a A-norma do erro. São exemplo dessa categoria os métodos do GC, no caso simétrico, e o FOM no caso não simétrico.
- $L^m = AK^m$: essa escolha do L^m minimiza a norma residual $||b-Ax^0||_2$. Exemplos são os métodos GMRES e seus equivalentes, incluindo o GCR no caso de matrizes simétricas.
- $L^m = K^m(A^T, r_*^0)$: onde r_*^0 é um vetor não paralelo a r_0. Essa escolha é designada para A quando essa é não simétrica e fornece relações de recorrência para bases do subespaço de Krylov. Esses métodos podem exigir menos espaço para armazenamento que os métodos GMRES e FOM, mas esses são mais propensos a falhar. Exemplos desses métodos são: Bi-CG, QMR, CGS, Bi-CGSTAB e TFQMR (CHOW, 1997), (RIZZI, 2002).

A escolha por um determinado método do subespaço de Krylov é decorrente da natureza da matriz de coeficientes do sistema de equações. Como já mencionado, neste trabalho, os sistemas de equações são SDP e o método adotado para a resolução desses foi o método do GC, descrito a seguir. Para um estudo mais completo sobre ele consulte (SHEWCHUK, 1994).

3.3.2.2 Método do Gradiente Conjugado

Nessa seção é descrito o método do gradiente conjugado (GC), que é um método iterativo não estacionário no subespaço de Krylov. O GC é considerado um dos métodos mais eficientes para a resolução de sistemas de grande porte e esparsos, onde a matriz de coeficiente é simétrica e definida-positiva (SHEWCHUK, 1994) (SAAD, 1996). Uma matriz A é dita simétrica se $A = A^T$ e definida-positiva se $\forall x \neq 0, x^T Ax > 0$ (SHEWCHUK, 1994).

Diferentemente dos métodos iterativos clássicos, que se baseiam na intersecção de planos, o GC soluciona o sistema através da minimização da função quadrática (SHEWCHUK, 1994).

A forma da função quadrática é dada pela equação 3.19, onde A é a matriz, x e b os vetores e c é um escalar.

$$f(x) = \frac{1}{2}x^T Ax - b^T x + c \tag{3.19}$$

O gradiente da forma quadrática é definido por:

$$f'(x) = \begin{bmatrix} \frac{\partial}{\partial x_1} f(x) \\ \frac{\partial}{\partial x_2} f(x) \\ \vdots \\ \frac{\partial}{\partial x_n} f(x) \end{bmatrix} \tag{3.20}$$

Aplicando a equação 3.20 em 3.19, como pode ser visto em (CUNHA, 2000), obtém-se:

$$f'(x) = \frac{1}{2}A^T x + \frac{1}{2}Ax - b \tag{3.21}$$

Se A é simétrica, então a equação 3.21 pode ser escrita como na equação 3.22. Assim, a minização da função quadrática corresponde exatamente à solução do sistema de equações.

$$f'(x) = Ax - b = 0 \tag{3.22}$$

Como o objetivo é a encontrar o ponto mínimo da função e, como conhecido no cálculo diferencial, o gradiente é um vetor que aponta para a direção de crescimento máximo da função, inicia-se um seqüência de passos $x_1, x_2, ..., x_n$, a partir de um ponto x_0, na direção contrária ao gradiente. Esse processo ocorre até que a aproximação da solução atinja a acurácia desejada.

Para garantir que o GC não repita passos na mesma direção faz-se uso do resíduo da iteração anterior no cálculo da nova direção, isto porque, o resíduo é ortogonal ao vetor direção da iteração anterior. Desta forma, a definição dos vetores direção implica na construção de um subespaço formado pela combinação linear dos resíduos:

$$d^n = span\{r^0, r^1, r^2, ..., r^{i-1}\} \tag{3.23}$$

Assim, a cada iteração, o vetor de direção é construído, através da equação 3.24, a partir do subespaço dos resíduos, de forma a ser ortogonal a todos os vetores de resíduos e direções anteriores.

$$d_i = r_i + \beta d_{i-1} \tag{3.24}$$

O valor β da equação 3.24, é a razão entre a norma do resíduo atual e o do resíduo anterior. E é dado pela equação 3.25.

$$\beta = \frac{r_i^T r_i}{r_{i-1}^T r_{i-1}} \tag{3.25}$$

O resíduo, que é utilizado nas equações 3.24 e 3.25, pode ser calculado de duas formas. Na primeira delas, utilizando-se a aproximação atual do vetor x através da expressão:

$$r_i = b - Ax_i \tag{3.26}$$

Essa forma, apresenta como desvantagem a necessidade de uma operação de multiplicação matriz por vetor adicional, aumentando o custo computacional do algoritmo.

Na segunda forma, o resíduo é calculado a partir do resíduo da iteração anterior, através da equação 3.27. Esta forma elimina a necessidade da operação matriz por vetor, mas tem como desvantagem um acúmulo de erro a cada iteração do método.

$$r_i = r_{i-1} - \alpha_i A d_i \tag{3.27}$$

Uma solução para esse problema é calcular o valor do resíduo a cada iteração através da equação 3.27 e a cada k iterações (50 no caso do algoritmo da Figura 3.4) eliminar o erro acumulado através da equação 3.26 (MARTINOTTO, 2001). O valor α (tamanho do passo), necessário na equação 3.27, pode ser obtido através da expressão 3.28.

Defina i_{max} e ε
$i = 1$
$d = r$
$\delta_{novo} = r^T r$
$\delta_0 = \delta_{novo}$
Enquanto $i < i_{max}$ e $\delta_{novo} > \varepsilon^2 \delta_0$
  $q = Ad$
  $\alpha = \delta_{novo}/d^T q$
  $x = x + \alpha d$
  Se i é divisível por 50 e n é diferente de 0
    $r = b - Ax$
  Senão
    $r = r - \alpha q$
  $\delta_{velho} = \delta_{novo}$
  $\delta_{novo} = r^T r$
  $\beta = \delta_{novo}/\delta_{velho}$
  $d = r + \beta d$
  $i = i + 1$

Figura 3.4: Algoritmo do gradiente conjugado

$$\alpha = \frac{r_i^T r_i}{d_i^T A d_i} \tag{3.28}$$

No algoritmo do GC, a cada iteração, uma nova aproximação do vetor solução x é calculada através da equação 3.29.

$$x_i = x_{i-1} + \alpha_i A d_{i-1} \tag{3.29}$$

Conforme mencionado anteriormente, no método do GC gera-se uma sequência de vetores solução a partir de uma solução inicial até que um critério de parada seja atendido. Esse critério de parada pode ser uma acurácia desejada ou até que um número máximo de iterações seja atingido (PICININ, 2003).

Existem diferentes variantes do método do GC. Neste trabalho adotou-se o algoritmo do GC apresentado em (SHEWCHUK, 1994). Esse algoritmo pode ser visto na Figura 3.4.

3.4 Bibliotecas e Pacotes de Resolução de Sistemas de Equações

A paralelização de métodos de resolução de sistemas de equações é uma área com muitas pesquisas já realizadas, tendo como resultado algumas ferramentas e bibliotecas que facilitam a construção de aplicações paralelas. Nesta seção são descritas algumas das principais bibliotecas disponíveis para a resolução de sistemas. As bibliotecas apresentadas nessa seção são: Aztec, IML++, PETSc e PIM.

3.4.1 Aztec

Aztec é uma biblioteca paralela com métodos iterativos e pré-condicionadores para resolução de sistemas de equações de grande porte e esparsos. Essa biblioteca foi desenvolvida por Scott A. Hutchinson, John N. Shadid e Ray S. Tuminaro no *Sandia National Laboratories*, utilizando a linguagem de programação C e a biblioteca de troca de mensagens MPI (EIJKHOUT, 1998). A Aztec faz uso, ainda, de rotinas das bibliotecas: BLAS, Lapack, Linpack e Y12m (PICININ, 2003).

A bibliteca Aztec implementa vários métodos do subespaço de Krylov, entre os quais os métodos GC, CGS, BiCGSTAB e GMRES. Esses métodos podem ser usados em conjunto com pré-condicionadores, tais como polinomial e decomposição de domínio usando LU ou ILU nos subdomínios.

Na biblioteca Aztec a multiplicação de matriz por vetor, operação fundamental em métodos iterativos do subespaço de Krylov, pode ser implementada de duas diferentes formas. Na primeira delas, o usuário é responsável pela implementação dessa rotina. Já na segunda forma, o usuário pode utilizar rotinas disponibilizadas pela própria biblioteca. As rotinas de matriz por vetor implementadas na Aztec usam dois diferentes formatos de armazenamento, que são: uma versão distribuída do formato MSR (*Modified Sparse Row*) e uma versão, também distribuída, do formato VBR (*Variable Block Row*) (TUMINARO; SHADID; HEROUX, 2003).

Adicionalmente, a biblioteca Aztec possui como vantagem o fato de incorporar rotinas de particionamento implementadas no pacote Chaco (HENDRICKSON; LELAND, 1995) e a utilização de algoritmos para matrizes densas na resolução de sistemas esparsos onde os elementos estão distribuídos em blocos.

3.4.2 IML++/SpaseLib++

A IML++ (*Iterative Methods Library*) foi desenvolvida no *National Institute of Standards and Technology* por Jack Dongarra, Andrew Lumsdaine, Roldan Pozo e Karin A. Remington, e é uma coleção de métodos iterativos, implementados em C++, para a resolução de sistemas de equações simétricos e não simétricos. Os métodos implementados na biblioteca IML++ são: iteração de Richardson, iteração de Chebyshev, GC, CGS, Bi-CG, Bi-CGSTAB, GMRES e QMR (TUMINARO; SHADID; HEROUX, 2003).

Na IML++ as classes que implementam as operações de matriz por vetor e de pré-condicionamento devem ser fornecidas pelo usuário. Um exemplo de implementação, dessas classes, compatível com a IML++ é a biblioteca SparseLib++, desenvolvida por Roldan Pozo, Karin A. Remington e Andrew Lumsdaine (POZO; REMINGTON; LUMSDAINE, 1996).

A biblioteca SparseLib++ permite o uso de diversos formatos de armazenamentos para matrizes esparsas, entre os quais estão: coordenada, CSR (*Compressed Sparse Row*) e CSC (*Compressed Sparse Column*).

A SparseLib++ implementa, ainda, os pré-condicionadores: diagonal, ILU e IC. Além disso, a SparseLib++ fornece ferramentas para conversão das matrizes *Harwell-Boeing* para os formatos suportados por ela (EIJKHOUT, 1998).

3.4.3 PETSc

A PETSc (*The Portable, Extensible Toolkit for Scientific Computation*) foi desenvolvida no *Argonne National Laboratory* por Satish Balay, Kris Buschelman, William Gropp, Dinesh Kaushik, Matt Knepley Lois C. McInnes, Barry Smith e Hong Zhang e consiste

em uma conjunto de classes com estruturas de dados e rotinas para a solução numérica de EDPs em computadores paralelos. A PETSc pode ser usada em códigos desenvolvidos nas linguagens C, C++ e Fortran e utiliza o MPI como biblioteca de troca de mensagens (SMITH et al., 2003).

A biblioteca PETSc provê estruturas e rotinas para o armazenamento e manipulação de matrizes densas e esparsas em computadores seqüenciais ou paralelos. A PETSc suporta, para o armazenamento de matrizes esparsas, os formatos CSR e AIJ (também chamado de *Yale Sparse Matrix Format*) (SMITH et al., 2003). Formatos adicionais podem ser acrescentados pelo usuário.

Na biblioteca PETSc são implementados vários métodos iterativos do subespaço de Krylov, que são: Iteração de Richardson, Iteração de Chebychev, GC, Bi-CG, GMRES, Bi-CGSTAB, CGS, TFQMR, CR, LSM. Esses métodos podem ser utilizados em conjunto com vários pré-condicionadores, entre os quais estão: Jacobi, Jacobi em Bloco, Gauss-Seidel (apenas seqüencial), SOR, SSOR, Cholesky (apenas seqüencial), LU (apenas seqüencial), IC, ILU, Aditivo de Schwarz (EIJKHOUT, 1998). Além disso a biblioteca PETSc apresenta rotinas paralelas, baseadas no método de Newton, para a resolução de sistemas de equações não lineares (SMITH et al., 2003).

Para um maior desempenho, a biblioteca PETSc usa rotinas das bibliotecas BLAS, LAPACK, LINPACK, MINPACK, SPARSPAK e SPARSEKIT2 (SMITH et al., 2003).

3.4.4 PIM

A biblioteca PIM (*Parallel Iterative Methods*) foi desenvolvida na *University of Kent at Canterbury* por Rudnei Dias da Cunha e por Tim Hopkins e é uma coleção de rotinas em FORTRAN 77 para a resolução de sistemas de equações em máquinas paralelas usando métodos iterativos (CUNHA; HOPKINS, 2003).

Um grande número de métodos iterativos para matrizes simétricas e não simétricas estão disponíveis na biblioteca PIM, incluindo os métodos do GC, Bi-CG, CGS, Bi-CGSTAB, RBi-CGSTAB, GMRES, GCR, CGNR, CGNE, QMR, TFQMR e iteração de Chebyshev (CUNHA; HOPKINS, 2003).

É importante salientar que na biblioteca PIM cabe ao usuário implementar as operações de produto matriz por vetor, produto escalar e métodos de pré-condicionamento. Isso torna essa biblioteca altamente independente de estrutura de dados e protocolos de comunicação. Isso permite ao usuário o desenvolvimento de aplicações para diferentes arquiteturas, paralelas ou não (EIJKHOUT, 1998).

3.5 Considerações Finais

Neste capítulo proporcionou-se uma visão geral sobre a geração de sistemas de equações em aplicações que modelam alguns problemas físicos reais. Foram abordados conceitos introdutórios sobre EDPs e sobre a discretização dessas através do método de diferenças finitas. Para maiores informações sobre EDPs e sobre métodos de discretização sugere-se (MALISKA, 1995), (FORTUNA, 2000) e (CUNHA, 2000).

A discretização das EDPs pode ser feita de modo a gerar sistemas de equações lineares que precisam ser resolvidos a cada passo tempo. Esses sistemas podem ser resolvidos através de duas classes de métodos: métodos diretos e métodos iterativos. Neste trabalho empregou-se métodos iterativos, pois as matrizes de coeficientes são esparsas e de grande porte e o emprego de métodos diretos destruiria a esparsidades dessas, e, conseqüentemente, dificultaria a otimização no armazenamento. Mais especificamente, utilizou-

se, neste trabalho, o método do GC uma vez que a matriz de coeficientes é simétrica e definida-positiva.

Para um estudo mais aprofundado sobre métodos numéricos para a resolução de sistemas de equações recomenda-se (GOLUB; LOAN, 1996), (SAAD, 1996). Maiores informações sobre o método do GC podem ser encontradas em (SHEWCHUK, 1994) e (SLAVIERO, 1997), e para maiores informações sobre os formatos para armazenamento de matrizes esparsas consulte (BARRETT et al., 1994), (JÚDICE; PATRÍCIO, 1996) e (SAAD, 1996).

Uma vez que os sistemas de equações resultantes da discretização de EDPs, em aplicações realísticas, são de grande porte, é conveniente o uso de processamento paralelo. No próximo capítulo são discutidas técnicas para a resolução desses sistemas em paralelo.

4 ESTRATÉGIAS PARA A RESOLUÇÃO DE SISTEMAS DE EQUAÇÕES EM PARALELO

Como mencionado o processo de discretização do problema de estudo de caso resulta em sistemas de equações lineares que necessitam ser resolvidos a cada passo de tempo.

Numericamente a acurácia da solução das EDPs depende, além do método de aproximação das EDPs utilizado, do número de pontos utilizados na discretização. Quanto maior o número de pontos maior a acurácia obtida. Em contrapartida, maior será a ordem dos sistemas de equações a serem resolvidos (RIZZI, 2002). Sendo assim, em simulações numéricas realísticas de grande escala espaço-temporal, e que exigem uma alta acurácia numérica, o tempo computacional necessário para simular tais fenômenos pode ser muito elevado. Uma alternativa, cada vez mais freqüente para resolver esse problema, é o uso de computação paralela, em particular o uso de *clusters* de PCs (CHARÃO, 2001).

Neste capítulo são discutidos métodos de decomposição de domínio, abordagem utilizada, neste trabalho, para a resolução de sistemas de equações em parelelo. São discutidas, ainda, as técnicas utilizadas para o particionamento do domínio computacional no modelo HIDRA. O particionamento do domínio irá influenciar, consideravelmente, nos demais passos necessários para a resolução, em paralelo, dos sistemas de equações gerados pelo modelo de hidrodinâmica (PICININ, 2003).

4.1 Particionamento do Domínio

Em problemas de simulação, como, por exemplo, o modelo HIDRA, o paralelismo pode ser obtido através do particionamento do domínio computacional entre os processadores disponíveis. Com o particionamento, cada subdomínio pode ser tratado em paralelo com os demais, diminuindo o tempo total na solução do problema (DORNELES, 2003). É importante ressaltar que a fase de particionamento do domínio computacional é de grande importância, uma vez que irá afetar, consideravelmente, os demais passos necessários para a paralelização da aplicação (PICININ, 2003).

De modo geral, o particionamento do domínio, segundo (DORNELES, 2003), deve ser feito de forma a atender três requisitos básicos:

- A distribuição balanceada da carga computacional entre os processadores disponíveis;
- A minimização da comunicação entre os processadores;
- O tempo de execução do algoritmo de particionamento deve ser menor que o tempo ganho com o seu uso.

Em arquiteturas heterogêneas, isto é, em arquiteturas onde os nodos podem possuir diferentes capacidades de processamento, a distribuição da carga computacional deve levar em consideração a capacidade de cada nodo. Nodos com maior capacidade de processamento devem receber uma carga computacional proporcional.

4.1.1 Particionamento do Domínio visto como um Problema de Particionamento de Grafos

O problema do particionamento do domínio computacional pode ser modelado como um problema de particionamento de grafos, onde os vértices representam os pontos da malha e as arestas a relação de vizinhança entre esses. Neste caso, a comunicação total entre os subdomínios pode ser estimada pelo número de arestas que ligam os vértices de diferentes domínios (SANTOS; DORNELES, 2001).

Desta forma, considerando que um grafo representa o domínio computacional (malha), o problema consiste em dividir o grafo em k subgrafos, onde k representa o número de processadores disponíveis, de modo que cada subgrafo tenha um mesmo número de vértices (pontos da malha) e que o corte de arestas (comunicação) entre os subdomínios seja minimizado.

O problema de dividir um grafo em k subgrafos com as propriedades de maximizar a computação e minimizar a comunicação é chamado de Problema de k-particionamento, e é um problema NP-difícil (GAREY; JOHNSON, 1979), existindo apenas heurísticas que encontram uma aproximação e não a solução ótima (DORNELES, 2003).

Um grafo pode ser particionado considerando-se a localização espacial ou a conectividade de seus vértices. Baseado neste critério, os algoritmos de particionamento de grafos podem ser divididos em duas classes: algoritmos geométricos e algoritmos combinatórios.

Os algoritmos geométricos só podem ser utilizados em grafos que dispõem das informações sobre as coordenadas de cada vértice. Isso ocorre, freqüentemente, em grafos oriundos da discretização de um domínio físico. Durante a fase de particionamento os algoritmos geométricos ignoram informações relativas a conectividade dos vértices do grafo considerando apenas as coordenadas desses (SANTOS; DORNELES, 2001), (DORNELES, 2003).

Já os algoritmos combinatórios podem ser usados em grafos que não possuem coordenadas associadas aos vértices, isto é, em grafos onde não há identificação do vértice como um ponto físico no espaço. Desta forma, durante o particionamento, essa classe de algoritmos considera apenas a conectividade dos vértices. Os algorimos de particionamento implementados no pacote METIS, utilizado neste trabalho para o particionamento do domínio, fazem parte deste grupo.

Segundo (SCHLOEGEL; KARYPIS; KUMAR, 2000) os algoritmos combinatórios geram partições com um menor corte de arestas. Por outro lado, tendem a ser mais lentos que os algoritmos geométricos. Além disso, esses algoritmos são mais difíceis de serem paralelizados.

As heurísticas de particionamento podem ser divididas, ainda, de acordo com o momento em que são aplicados ao grafo. De acordo com esse critério as heurísticas de particionamento podem ser divididas em métodos globais e locais.

Métodos globais, algumas vezes também chamados de heurísticas de construção, recebem como entrada um grafo na forma original e geram um grafo particionado em k partes. Entre os métodos dessa classe cita-se: RCB (*Recursive Coordinate Bisection*) (DORNELES, 2003), RSB (*Recursive Spectral Bisection*) (SAAD, 1996) e LND (*Levelized Nested Dissection*) (SANTOS; DORNELES, 2001).

Já os métodos locais, também chamados de heurísticas de melhoramento, recebem como entrada um grafo particionado e tentam melhorar a qualidade da partição, isto é, diminuir o corte de arestas, através do rearranjo dos vértices. Entre os métodos dessa classe os mais conhecidos são o Kernighan-Lin e o Fiduccia-Mattheyses (SANTOS; DORNELES, 2001).

Existem, ainda, os métodos multinível, que entre outras características, procuram combinar métodos globais e métodos locais. Em um método multinível o grafo é inicialmente reduzido, através da condensação de vértices, a um grafo menor, sobre o qual é aplicado um método global. Após o particionamento o grafo reduzido é novamente expandido ao grafo original, sendo que na fase de expansão é utilizado um método local, como por exemplo, Fiduccia-Mattheyses (DORNELES, 2003). A Figura 4.1 (SCHLOEGEL; KARYPIS; KUMAR, 2000) mostra o esquema de funcionamento de um método multinível.

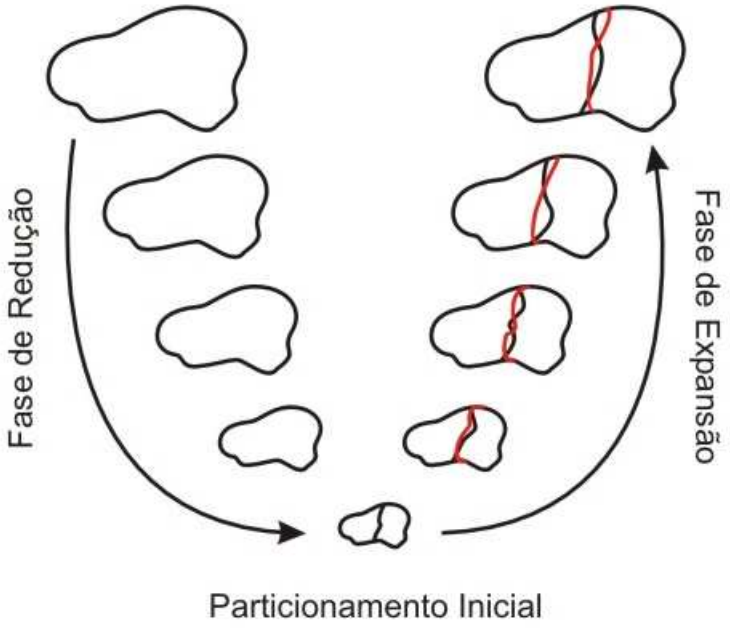

Figura 4.1: Método multinível

Existem várias bibliotecas que implementam algoritmos de particionamento de grafos, entre as quais as mais conhecidas são: o METIS (KARYPIS; KUMAR, 1998a), Chaco (HENDRICKSON; LELAND, 1995), Jostle (WALSHAW, 2000) e Scotch (PELLEGRINI, 1996). No desenvolvimento deste trabalho, no particionamento do domínio computacional, utilizou-se o Pacote METIS 4.0, descrito na seção 4.1.2. Optou-se pelo METIS, devido à facilidade de utilização do mesmo como uma biblioteca de rotinas de particionamento, o que não ocorre com os outros pacotes (DORNELES, 2003).

É importante destacar que, geralmente, quando utilizado algum algoritmo de particionamento de grafos esse resulta em subdomínios sem sobreposição. Para o uso de métodos de decomposição de domínio com sobreposição é necessário expandir os subdomínios de forma a garantir que todas as células existentes nas fronteiras artificiais, isto é criadas pelo particionamento, pertençam, também, ao interior dos subdomínios vizinhos (CAI, 2002).

4.1.2 METIS

O software METIS é um pacote de execução serial para o particionamento de grafos, malhas e reordenamento de matrizes esparsas desenvolvido na *University of Minnesota* por George Karypis e Vipin Kumar (KARYPIS; KUMAR, 1998a).

Os algoritmos implementados pelo METIS são baseados em esquemas multiníveis descritos em (KARYPIS; KUMAR, 1998b), (KARYPIS; KUMAR, 1998c) e (KARYPIS; KUMAR, 1998d). O pacote METIS disponibiliza dois métodos para o particionamento de um grafo: *pmetis* e *kmetis*. Esses diferenciam-se pelos algoritmos utilizados durante o particionamento.

No *pmetis* o particionamento do grafo reduzido é feito através do algoritmo *Region Growing*, descrito em (KARYPIS; KUMAR, 1998b), enquanto o *kmetis* é baseado em um algoritmo de bissecção recursiva multinível, descrito em (KARYPIS; KUMAR, 1998d). No *pmetis* o algoritmo local utilizado é uma variante do Fiduccia-Mattheyses, enquanto no *kmetis* pode-se escolher o método local a ser utilizado. O método *kmetis* disponibiliza algoritmos locais baseados na seleção aleatória de arestas e na seleção de arestas utilizando um método guloso (KARYPIS; KUMAR, 1998b). O método *pmetis* é recomendado, pela documentação disponível com o pacote METIS, para o particionamento do grafo em até oito partes, enquanto o *kmetis* é recomendado para casos onde é preciso particionar o grafo em mais de oito partes.

O METIS está disponível para *download* no endereço *http://www.cs.umn.edu/ karypis/mestis/*, não sendo necessária licença de uso. A Figura 4.2 mostra o particionamento do Lago Guaíba em 16 partes usando o pacote METIS. No particionamento utilizou-se o método *kmetis* com um método local baseado na seleção aleatória de arestas.

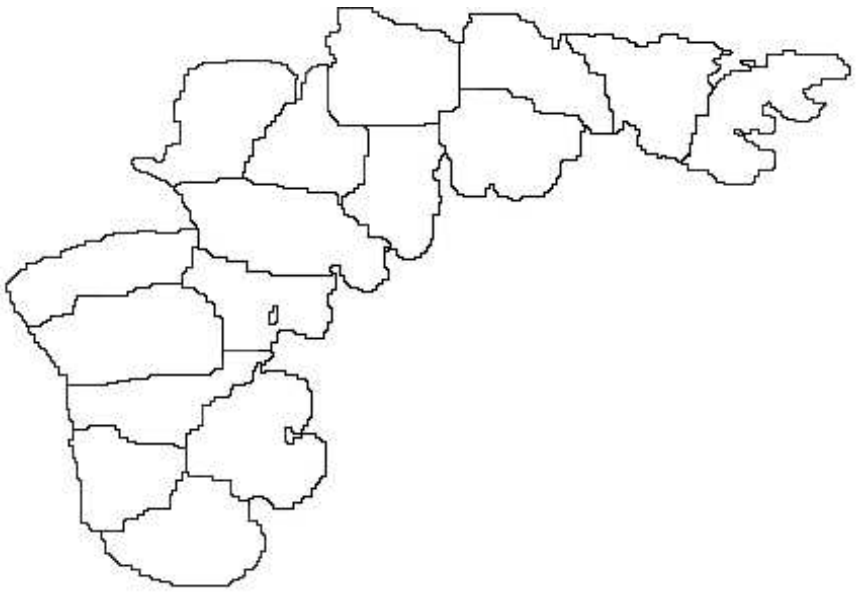

Figura 4.2: Particionamento do Lago Guaíba em 16 subdomínios usando METIS

4.2 Resolução de Sistemas de Equações em Paralelo

Após o particionamento do domínio a geração dos sistemas de equações é realizada em paralelo pelos processos, e a solução desses pode ser obtida através de duas aborda-

gens: decomposição de dados e decomposição de domínio.

Na primeira abordagem gera-se um único sistema de equações correspondente a todo o domínio computacional e esse sistema é resolvido através de um método numérico paralelizado, como por exemplo, o GC ou GMRES (CHARÃO, 2001).

A segunda abordagem consiste no emprego de métodos de decomposição de domínio. Esses métodos são um conjunto de técnicas e estratégias matemáticas e computacionais, e são baseados no particionamento do domínio computacional em subdomínios, de tal modo que a solução global do domínio é obtida pela combinação apropriada das soluções obtidas em cada um dos subdomínios (SMITH; BJORSTAD; GROPP, 1996).

Destaca-se que essas duas abordagens podem ser utilizadas de maneira complementar, onde um método de decomposição de domínio pode ser empregado para construir um pré-condicionador para acelerar a convergência de um método iterativo paralelizado via decomposição de dados (RIZZI, 2002).

As implementações desenvolvidas neste trabalho utilizam a abordagem de decomposição de domínio para a resolução em paralelo dos sistemas de equações lineares gerados pelo modelo paralelo de hidrodinâmica no HIDRA. Em (CANAL, 2000) e (PICININ, 2003) podem ser encontradas informações e resultados obtidos utilizando a abordagem de decomposição de dados.

4.3 Métodos de Decomposição de Domínio

Os métodos de decomposição de domínio são baseados no particionamento do domínio computacional em subdomínios e a solução global é obtida pela combinação apropriada das soluções locais. Desta forma, o problema original pode ser tratado como uma série de subproblemas de tamanho reduzido (FLEMISH, 2001). Uma vez que esses subdomínios podem ser tratados quase que independentemente, tais métodos são atrativos para ambientes paralelos de memória distribuída.

Segundo (SMITH; BJORSTAD; GROPP, 1996) alguns dos principais atrativos para o uso dos MDDs são:

- O uso de dados locais, necessitando de pouca comunicação, a qual, em geral, fica restrita às fronteiras dos subdomínios durante a sincronização da solução;
- A possibilidade de trabalhar com distintos modelos de EDPs em regiões com geometria complexa, onde as EDPs podem exibir comportamentos diferentes em diferentes partes do domínio;
- A construção de pré-condicionadores que são empregados para acelerar métodos iterativos.

De acordo com a existência ou não de regiões sobrepostas no domínio computacional os MDDs são classificados em: métodos de Schwarz, onde os subdomínios possuem uma região de sobreposição, e métodos de Schur, que não apresentam sobreposição dos domínios (CHAN; MATHEW, 1994). Na seção 4.4 são abordados os métodos de Schwarz e na seção 4.5 os métodos de Schur, mais especificamente o método do complemento de Schur que foi o utilizado no desenvolvimento desse trabalho.

4.4 Métodos de Schwarz

Em um método do tipo Schwarz, o domínio computacional Ω é particionado em k subdomínios Ω_i que se sobrepõem de forma que:

$$\Omega = \bigcup_{i=1,k} \Omega_i, \quad \Omega_i \cap \Omega_j \neq \emptyset \tag{4.1}$$

Nos métodos de Schwarz é a presença de regiões sobrepostas que garante a continuidade da solução entre os subdomínios (DEBREU; BLAYO, 1998). É importante destacar, ainda, que os valores das células da região de sobreposição devem ser calculados em ambos os subdomínios.

Na Figura 4.3 tem-se um exemplo do particionamento de um domínio Ω, formado por um disco e um retângulo, em subdomínios Ω_1 e Ω_2 sobrepostos, onde Ω_i corresponde às células internas de cada subdomínio; Γ_i corresponde às células pertencentes às fronteiras artificiais e $\partial\Omega_i$ corresponde às fronteiras reais.

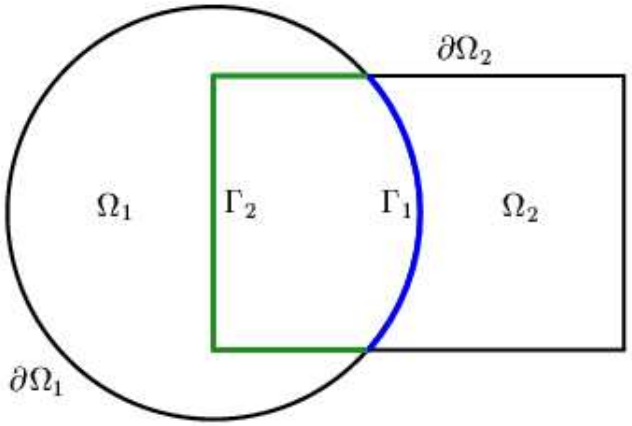

Figura 4.3: Domínio com particionamento com sobreposição

A idéia básica dos MDDs de Schwarz pode ser melhor compreendida através do primeiro MDD conhecido, o método alternado de Schwarz, desenvolvido por Hermann Amandus Schwarz em 1869, e que foi empregado para resolver um problema (EDP) elíptico definido em um domínio formado pela união de dois subdomínios regulares sobrepostos como o domínio da Figura 4.3. Mais especificamente, dado o problema:

$$\begin{cases} Lu = f, u \in \Omega \\ u = g, u \in \partial\Omega \end{cases} \tag{4.2}$$

definido em $\Omega = \Omega_1 \cup \Omega_2$, o método desenvolvido por Schwarz calcula a solução do problema no domínio Ω através da solução alternada nos subdomínios Ω_1 e Ω_2. As soluções nos subdomínios são obtidas da seguinte forma: as condições de contorno correspondentes a Γ_2 em Ω_2, a cada iteração n, são fornecidas pela solução dos valores nodais no domínio Ω_1 denotados por $u_1^n|\Gamma_2$, e vice-versa (CAI, 2002). Assim, dada uma solução inicial u_1^0 em Ω a solução do problema Ω pode ser resolvido iterativamente resolvendo os seguintes problemas:

$$\text{resolva em } \Omega_1\text{: } \begin{cases} L_1 u_1^n = f_1, u \in \Omega_1 \\ u_1^n = g, u \in \partial\Omega_1 \setminus \Gamma_1 \\ u_1^n = u_2^{n-1}|\Gamma_1, u \in \Gamma_1 \end{cases}$$

$$\text{e, então, resolva } \Omega_2\text{: } \begin{cases} L_2 u_2^n = f_2, u \in \Omega_2 \\ u_2^n = g, u \in \partial\Omega_2 \setminus \Gamma_2 \\ u_2^n = u_1^n|\Gamma_2, u \in \Gamma_2 \end{cases} \tag{4.3}$$

4.4.1 Método Multiplicativo de Schwarz

Estendendo-se o método alternado de Schwarz para muitos domínios obtém-se o método conhecido por multiplicativo de Schwarz que pode ser definido como:

$$\begin{cases} L_i u_i^n = f_i, u \in \Omega_i \\ u_i^n = g, u \in \partial\Omega_i \setminus \Gamma_i \\ u_i^n = \tilde{g}^n, u \in \Gamma_i \end{cases} \tag{4.4}$$

onde $\tilde{g}$ corresponde às condições de contorno em Γ_i. De acordo com o particionamento resultante, determinadas células podem pertencer a mais de uma Γ_i. Neste caso a solução encontrada no último subdomínio calculado é utilizada como condição de contorno nos subdomínios vizinhos(CAI, 2002).

Note-se que o método multiplicativo de Schwarz é inerentemente seqüencial, isto porque, os valores calculados em Γ_i na mesma iteração são utilizados como condição de contorno por outros subdomínios. Para obter paralelismo nessa versão é necessário o uso de alguma técnica de coloração dos subdomínios. Com essa técnica cada subdomínio recebe uma cor de forma que subdomínios vizinhos não possuam cores idênticas. Subdomínios com mesma cor podem ser resolvidos em paralelo, e conseqüentemente o número de passos seqüenciais pode ser minimizado. Pode-se identificar que, no método multiplicativo de Schwarz, o número de cores utilizadas na coloração definirá o número de passos seqüenciais do algoritmo, isto porque, apenas uma cor pode ser resolvida a cada passo de tempo. Desta forma, é importante que o algoritmo de coloração use o menor número de cores possíveis.

Além disso, a convergência do método multiplicativo de Schwarz é inversamente proporcional ao número de cores utilizadas, isto é, quanto menor o número de cores, mais rápida será a convergência. Assim, a minimização no número de cores não só diminuirá o número de passos seqüenciais como provocará, também, uma melhora na convergência do método (CAI; SAAD, 1993).

Pode ser visto que o problema de coloração dos subdomínios consiste em um problema de coloração de grafos, onde vértices do grafo representam os subdomínios e as arestas os subdomínios vizinhos, isto é, subdomínios que possuem uma região de sobreposição (CAI; SAAD, 1993). Na seção 4.4.1.1 é feita uma descrição sobre o problema de coloração de grafos e as soluções adotadas neste trabalho.

4.4.1.1 Coloração de Grafos

O problema de coloração de grafos consiste em atribuir cores aos vértices de um grafo de forma que nenhum par de vértices adjacentes receba a mesma cor. O número mínimo

de cores necessárias para a coloração de um determinado grafo é denominado de número cromático.

O problema de determinar o número cromático de um grafo é NP-difícil (GAREY; JOHNSON, 1979), sendo importante que se disponha de técnicas heurísticas eficientes para resolver o problema de forma satisfatória.

Existem muitas técnicas heurísticas de baixo custo computacional para a coloração de grafos. Na prática, segundo (CAI; SAAD, 1993), na paralelização do método multiplicativo de Schwarz, um algoritmo guloso com heurísticas oferece bons resultados.

Neste trabalho, foi analisado, um algoritmo guloso com três diferentes heurísticas. A diferença básica entre as heurísticas analisadas é a forma como é feita a seleção dos vértices durante a fase de coloração do grafo.

Na primeira heurística, descrita em (CAI; SAAD, 1993), a coloração é feita seguindo a ordem seqüencial de numeração dos vértices. A primeira cor que não pertence a nenhum dos vértices adjacentes será escolhida para colorir o vértice. Se os vértices adjacentes, já coloridos, usam todas as cores, o vértice será colorido com uma nova cor.

Essa primeira heurística pode ser melhorada considerando-se o seguinte fato: quanto maior for o grau do vértice (número de vértices adjacentes) maior será a restrição na atribuição de uma cor a este. Uma alternativa consiste em selecionar os vértices em ordem decrescente do grau dos vértices, assim, vértices com maior restrição são coloridos em primeiro lugar (WEST, 2001).

Heurísticas baseadas no grau dos vértices podem, ainda, ser aprimoradas, uma vez que ambigüidades podem surgir em grafos que apresentem muitos vértices de mesmo grau. Uma solução interessante para diminuir esse problema é proposta em (CONTE, 2003). Nessa solução define-se um vetor coluna d^0 com dimensão N, onde N é o número de vértices do grafo, preenchido com elementos 1. Indutivamente, para $j \geq 0$, define-se $d^{j+1} = Ad^j$, onde A é a matriz de adjacências do grafo. Observa-se que o vetor d^1 é exatamente o vetor de graus do grafo. A idéia é, então, utilizar d^k, para um k grande, ao invés de d^1 na seleção dos vértices para a coloração. Os melhores resultados com essa heurística foram obtidos com $d^{\sqrt{N}}$.

Na seção 5.1.1.1 são apresentados os resultados obtidos com essas diferentes heurísticas de coloração implementadas.

4.4.2 Método Aditivo de Schwarz

Outra variante dos MDDs com sobreposição é o método aditivo de Schwarz, que é uma abordagem com maior potencial de paralelismo em relação ao multiplicativo de Schwarz. A diferença entre esses é a forma como são feitas as atualizações das condições de contorno em Γ_i. Na versão multiplicativa os subdomínios utilizam as condições de contorno calculadas nos subdomínios vizinhos na mesma iteração. Já na versão aditiva os subdomínios utilizam as condições de contorno calculadas nos subdomínios na iteração anterior (SILVA et al., 1997). Desta forma, o método aditivo de Schwarz é definido como:

$$\begin{cases} L_i u_i^n = f_i, u \in \Omega_i \\ u_i^n = g, u \in \partial\Omega_i \setminus \Gamma_i \\ u_i^n = g^{n-1}, u \in \Gamma_i \end{cases} \tag{4.5}$$

Devido ao fato de utilizar os valores calculados em Γ_i na iteração anterior como condição de contorno, os subdomínios podem ser tratados independentemente, bastando atualizar as condições de contorno no final de cada iteração do método.

4.4.3 Convergência dos Métodos de Schwarz

No que diz respeito à velocidade de convergência, em muitos casos, o método aditivo de Schwarz requer até o dobro de iterações em relação ao método multiplicativo de Schwarz, como mencionado em (SMITH; BJORSTAD; GROPP, 1996),(CHARÃO, 2001), e como indicam experimentos numéricos realizados.

Além disso, é importante ressaltar que a taxa de convergência dos métodos de Schwarz é sensível ao número de células na região de sobreposição. Com o aumento da área de sobreposição, maior será a velocidade de convergência. Em compensação maior será o tamanho dos subdomínios Ω_i e, conseqüentemente, o custo computacional para o cálculo das soluções locais. A região de sobreposição adotada neste trabalho foi de 2 células. Tal escolha é baseada no processo de discretização adotado no modelo HIDRA (DORNELES et al., 2000). As experiências numéricas realizadas mostraram que, no caso da aplicação desenvolvida, uma área de sobreposição maior não afeta o número de iterações necessárias para a convergência do método.

Ressalta-se ainda que o número de iterações necessárias para convergência, em um método do tipo Schwarz, pode ser maior com o aumento no número de subdomínios (CHARÃO, 2001). Em particular, no caso do método multiplicativo de Schwarz a convergência é dependente, ainda, do número de cores utilizadas (CAI; SAAD, 1993). No caso do método aditivo de Schwarz pode-se mostrar, sobre a hipótese que $\Omega_i \cap \Omega_j \cap \Omega_k \neq \oslash, \forall i \neq j \neq k$, que o algoritmo converge. Para um número superior a três subdomínios a convergência desse não é garantida (DEBREU; BLAYO, 1998).

4.4.4 Considerações de Implementação

Na abordagem adotada no modelo HIDRA, modelo em que este trabalho está inserido, após o particionamento e a expansão dos subdomínios para criação da região de sobreposição, cada processador é responsável pela geração dos sistemas de equações locais. Essa abordagem, além de explorar o paralelismo na geração dos sistemas, elimina a necessidade de distribuição dos dados a cada ciclo da simulação (PICININ, 2003).

Para a geração dos sistemas de equações locais utilizou-se uma numeração local, sendo que as células pertencentes à região de sobreposição são numeradas depois das células internas. Os sistemas de equações locais são armazenados utilizando o formato CSR, descrito na seção 3.2.3.

Uma vez que em um método de Schwarz as células de fronteira de um subdomínio são utilizadas como condições de contorno nos subdomínios vizinhos gera-se uma dependência de dados entre esses subdomínios. Essa dependência cria uma necessidade de comunicação entre os subdomínios durante a resolução dos sistemas de equações. Como já mencionado, para a solução dos subsistemas locais a cada ciclo de Schwarz utilizou-se o método do GC, descrito na seção 3.3.2.2.

Devido à necessidade de comunicação entre os subdomínios algumas estruturas auxiliares são necessárias. De maneira geral, essas estruturas identificam informações a serem enviadas e recebidas durante a fase de solução dos sistemas. Essas estruturas são criadas durante a fase de particionamento e consistem, para cada processo, de uma lista com os subdomínios vizinhos, sendo que para cada vizinho são armazenados dois vetores. O primeiro deles, contendo as posições do vetor solução a serem enviadas no fim de cada iteração a esse vizinho. E o segundo contendo as posições do vetor de termos independentes que receberão os valores calculados no subdomínio vizinho.

A Figura 4.4 mostra a região de fronteira entre dois subdomínios P_0 e P_1. A região S_0

corresponde à região de sobreposição de P_0 em P_1 e a região S_1 é a região de sobreposição de P_1 em P_0. Na figura tem-se um exemplo das estruturas de dados necessárias ao subdomínio P_0 para a troca de informações com o subdomínio P_1.

Note-se que os métodos de Schwarz requerem a expansão dos subdomínios. O tamanho da sobreposição (expansão) no exemplo ilustrado na Figura 4.4 é de duas células (dois vetores-colunas no exemplo). Assim, por exemplo, o processo P_0 que originalmente tinha como células internas aquelas identificadas com sublinhado simples, recebe no particionamento e na distribuição inicial, as células identificadas como S_0. Deste modo o domínio P_0, já expandido, conterá as células internas a P_0 (sublinhado simples) mais aquelas identificadas na Figura 4.4 como S_0 (sublinhado duplo). Da mesma forma, o processo P_1, já expandido, conterá as células internas a P_1 mais as células identificadas na figura como S_1.

No caso específico do processo P_0 a estrutura de comunicação armazena um identificador do processo P_1 (subdomínio vizinho) e as células que devem ser enviadas para o processo P_1 (no caso 4, 16, 28, 40, 52 e 64). Essas são utilizadas como condições de contorno por P_1, isto é, são adicionadas nas posições do vetor de termos independentes correspondentes às células das fronteiras artificiais do processo P_1 (no caso 5, 17, 29, 41, 53 e 65).

O processo P_0 necessita ainda armazenar as posições do vetor de termos independentes correspondentes às células das suas fronteiras artificiais (no caso 8, 20, 32, 44, 56 e 68). Essas são utilizadas para identificar as posições onde devem ser adicionadas os valores das células (condições de contorno) enviadas pelo processo P_1 (no caso 9, 21, 33, 45, 57 e 69).

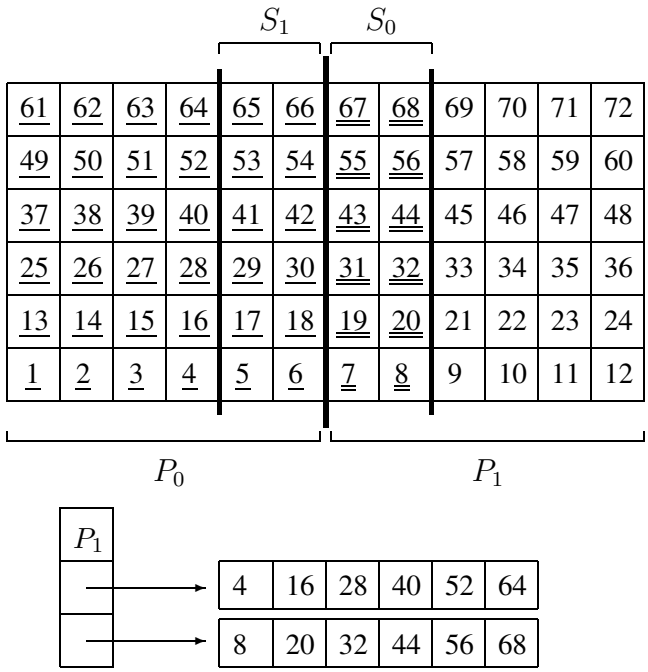

Figura 4.4: Estrutura de dados para a comunicação nos métodos de Schwarz

Para o cálculo das soluções locais, em clusters de PCs multiprocessados, estudou-se duas abordagens. Em uma primeira abordagem o particionamento de domínio é feito de acordo com o número de processadores disponíveis e a exploração do paralelismo intranodal é feito através do uso de múltiplos processos que se comunicam através da troca

de mensagens. Em uma segunda abordagem o particionamento do domínio é feito de acordo com o número de nodos disponíveis e com o uso de *threads* para a exploração do paralelismo intra-nodal, isto é, em cada subdomínio (nodo) o processamento é dividido entre *threads*.

A segunda abordagem apresenta algumas vantagens como: um número menor de subdomínios, o que pode resultar em uma melhor convergência do método; uma menor redundância de cálculo, pois nessa abordagem há uma quantidade menor de células associadas a diferentes subdomínios; e um menor volume de troca de mensagens, porque com a diminuição no número de subdomínios ocorre um redução no número de processos que necessitam se comunicar ao final de cada iteração do método. Como desvantagem, tem-se que essa abordagem apresenta uma implementação mais complexa.

É importante destacar que o sistema operacional Linux não diferencia as estruturas de dados de *threads* e processos, assim, ambos são tratados da mesma forma. Como conseqüência as *threads* no Linux apresentam um custo de escalonamento idêntico aos dos processos (BAR, 2000), (OLIVEIRA; CARISSIMI; TOSCANI, 2001).

4.5 Método do Complemento de Schur

No método do complemento de Schur o domínio é particionado em subdomínios disjuntos. Esse método é utilizado em aplicações de simulação numérica e computacional paralelas onde, por exemplo, a malha gerada é do tipo não estruturada, ou não ocorre o emparelhamento das submalhas entre os subdomínios, ou, ainda, quando se têm diferentes modelos matemáticos para os diferentes subdomínios.

O método do complemento de Schur apresenta diferentes variantes. A escolha de uma ou outra variante pode ser decorrente da forma de particionamento utilizada. Para o desenvolvimento desse trabalho utilizou-se a variante descrita em (SAAD, 1996). Nessa o domínio Ω é decomposto em k subdomínios disjuntos, de forma que:

$$\Omega = \bigcup_{i=1,k} \Omega_i, \quad \Omega_i \cap \Omega_j = \emptyset \tag{4.6}$$

Em cada subdomínio Ω_i, as células locais são ordenadas de forma que as células pertencentes à interface são listadas depois das células internas do subdomínio. Tal ordenação apresenta algumas vantagens, incluindo uma comunicação entre processos mais eficiente (SAAD; SOSONKINA, 1999).

Com essa ordenação, o vetor de incógnitas x_i é particionado em duas partes:

$$x_i = \begin{bmatrix} u_i \\ y_i \end{bmatrix} \tag{4.7}$$

onde o subvetor u_i representa as células internas do subdomínio e o subvetor y_i representa as células da interface do subdomínio Ω_i. Da mesma forma, o vetor dos termos independentes pode ser particionado em:

$$b_i = \begin{bmatrix} f_i \\ g_i \end{bmatrix} \tag{4.8}$$

De acordo com essa numeração a matriz de cada subdomínio A_i, também chamada de matriz local, apresenta a seguinte estrutura:

$$A_i = \begin{bmatrix} B_i & E_i \\ F_i & C_i \end{bmatrix} \tag{4.9}$$

onde B_i é uma matriz associada com as células internas do subdomínio Ω_i. As matrizes E_i e F_i representam as interações entre as células internas e as células de interface do subdomíno Ω_i. E a matriz C_i representa as células pertencentes à interface do subdomínio (SAAD, 1996).

Com isso, as equações locais podem ser escritas como:

$$\begin{array}{rcrcrcl} B_i u_i & + & E_i y_i & & & = & f_i \\ F_i u_i & + & C_i y_i & + & \sum_{j \in N_i} E_{i,j} y_j & = & g_i \end{array} \tag{4.10}$$

O termo $E_{i,j} y_i$ é a contribuição dos subdomínios vizinhos Ω_j para o sistema local do subdomínio Ω_i e N_i é conjunto de subdomínios vizinhos ao subdomínio Ω_i. A soma das contribuições é o resultado da multiplicação de elementos pertencentes ao estêncil externo (células dos subdomínios vizinhos que pertencentem a molécula computacional das células de fronteiras) pelas células de fronteiras calculadas nos subdomínios vizinhos (KUZNETSOV; LO; SAAD, 1997). Desta forma, para o cálculo dessas contribuições é necessária a troca de informações entre os subdomínios. Destaca-se que é a soma dessas contribuições que garante a continuidade da solução entre os subdomínios.

Isolando a variável u_i na primeira equação obtém-se:

$$u_i = B_i^{-1}(f_i - E_i y_i) \tag{4.11}$$

Através da substituição da variável u_i na segunda equação obtém-se:

$$S_i y_i + \sum_{j \in N_i} E_{i,j} y_j = g_i - F_i B_i^{-1} f_i \tag{4.12}$$

Na equação 4.12 S_i é a uma matriz chamada de complemento de Schur de A_i e é obtida através de:

$$S_i = C_i - F_i B_i^{-1} E_i \tag{4.13}$$

A equação 4.12 é um sistema correspondente às células de interface do subdomínio e a resolução desse é obtida através de um processo iterativo. Uma vez que a solução na interface y_i é conhecida o cálculo das células internas u_i pode ser realizado independentemente através da equação 4.11.

Com um pequeno número de subdomínios, as interfaces entre os subdomínios são pequenas e, conseqüentemente, os sistemas de interface são de pequeno porte. Entretanto, com o aumento no número de subdomínios ocorre um aumento nas interfaces entre os subdomínios. Esse aumento das interfaces dos subdomínios provoca um aumento nos sistemas de interface e, conseqüentemente, no custo para a resolução dos mesmos (NADEEM, 2001).

Observa-se que para a construção da matriz de complemento de Schur e, também, para o cálculo do lado direito da equação 4.12 é necessário o cálculo da inversa das matrizes correspondentes às células internas do subdomínio Ω_i. O custo desta operação é proibitivo

para sistemas de grande porte. Além disso, a necessidade do cálculo das inversas acaba destruindo a esparsidade da matriz de coeficientes (CHARÃO, 2001).

Para a solução deste problema, neste trabalho, estudou-se a possibilidade do uso de técnicas que permitam calcular uma aproximação da inversa da matriz de coeficientes e que mantenham o padrão de esparsidade dessa. Como exemplo dessas técnicas pode-se citar o cálculo de uma aproximação da inversa através do uso de fatoração incompleta ou através de métodos polinomiais (MARTINOTTO, 2002).

De maneira geral, fatorações incompletas fornecem aproximações das inversas de melhor qualidade que os métodos polinomiais. Porém, essas são de difícil paralelização devido a sua natureza seqüencial (BENZI; TUMA, 1999). Neste trabalho optou-se pelo uso de métodos polinomiais, descritos na seção 4.5.1, pois esses são atrativos para ambientes paralelos, uma vez que são baseados em operações de álgebra linear que podem ser implementadas eficientemente nesses ambientes (CUNHA, 1992).

Deste modo, a opção desse trabalho foi privilegiar o desempenho computacional às custas da qualidade numérica, quando do desenvolvimento das paralelizações para o método do complemento de Schur.

4.5.1 Métodos Polinomiais

Métodos polinomiais baseiam-se na série truncada de Neumann para obter uma aproximação M^{-1} para B^{-1}. Se uma matriz J possui raio espectral menor que 1 ($\rho(J) < 1$) então a inversa de $I - J$ pode ser expressa como um série de potências em J (VARGAS, 1999):

$$(I - J)^{-1} = \sum_{k=0}^{\infty} J^k = I + J + J^2 + J^3 + ... \tag{4.14}$$

Por outro lado, se B é decomposta como $B = L + D + U$ e escrevendo a matriz da iteração de Jacobi $J = -D^{-1}(L + U)$, pode-se mostrar que $\rho(J) < 1$ (VARGAS, 1999). Portanto, se B é escrita como $B = D(I - J)$, e $(I - J)^{-1}$ é expandida como na equação (4.14), a inversa de B pode ser escrita como:

$$\begin{aligned} B^{-1} &= (D(I - J))^{-1} \\ &= (I - J)^{-1}D^{-1} \\ &= \sum_{k=0}^{\infty} J^k D^{-1} \\ &= \sum_{k=0}^{\infty} (-1)^k (D^{-1}(L + U))^k D^{-1} \end{aligned} \tag{4.15}$$

Assim, a estratégia dos métodos polinomiais é truncar a série 4.15 após k termos e obter uma aproximação M^{-1} para B^{-1}.

Se a série é truncada com $k = 0$, obtém-se que:

$$M^{-1} = D^{-1} \tag{4.16}$$

Já se a série (4.15) é truncada com $k = 1$ a matriz M^{-1} é calculada como:

$$M^{-1} = D^{-1} - D^{-1}(L + U)D^{-1} \tag{4.17}$$

que possuirá o mesmo padrão de esparsidade da matriz B, isto é, não ocorre nenhum tipo de preenchimento (POLLARD, 1992).

Neste trabalho foram realizados experimentos utilizando aproximações polinomiais com $k = 0$ e $k = 1$. As aproximações polinomiais com $k = 0$ apresentam um custo computacional relativamente baixo se comparado com aproximações com $k = 1$, mas em contrapartida oferecem aproximações de pior qualidade numérica. Alguns resultados sobre o uso do método do complemento de Schur com aproximações polinomiais com $k = 0$ podem ser encontrados em (CHARÃO, 2001).

4.5.2 Considerações de Implementação

De forma análoga às implementações dos métodos de Schwarz desenvolvidas, no método do complemento de Schur os sistemas de equações são gerados após o particionamento do domínio.

A numeração das células é local, sendo que as células pertencentes às fronteiras artificiais são numeradas após as células internas. Com essa numeração, os sistemas locais gerados possuem a estrutura dada pela equação 4.10.

No desenvolvimento deste trabalho optou-se por armazenar as submatrizes B_i, E_i, F_i e C_i em estruturas CSR distintas. Da mesma forma, optou-se pelo armazenamento dos subvetores u_i, y_i, f_i e g_i separadamente. Optou-se por essa forma de armazenamento, porque ela facilita a implementação das operações de álgebra linear que compõem o método do complemento de Schur.

Na resolução dos sistemas de interface, optou-se por não calcular explicitamente o complemento de Schur S. Isto é feito utilizando um método iterativo. Neste trabalho utilizou-se o método do GC uma vez que os sistemas de interface são SDP.

Desta forma, o complemento de Schur é calculado implicitamente no passo $q = Ad$ do método GC, onde a matriz A corresponde ao complemento de Schur dado por $C_i - F_i B_i^{-1} E_i$. Assim $q = Ad$ pode ser reescrito como:

$$q = (C_i - F_i B_i^{-1} E_i)d \tag{4.18}$$

e, portanto, o complemento de Schur pode ser calculado implicitamente através de três operações de produto matriz por vetor e uma subtração de vetores.

Como mencionado o cálculo dos sistemas de interface é feito através de um processo iterativo, sendo necessário o cálculo das contribuições dos subdomínios vizinhos no sistema de equações local. No cálculo das contribuições é necessária a troca de informações entre os subdomínios. Desta forma, tornou-se necessária a criação de estruturas auxiliares para a comunicação entre os subdomínios. Essas estruturas, de maneira geral, descrevem as fronteiras dos subdomínios e são geradas durante o particionamento do domínio.

Basicamente cada subdomínio necessita de uma lista de subdomínios vizinhos, sendo que para cada subdomínio vizinho são armazenados dois vetores. O primeiro deles contendo as posições das células de interface, pois os valores dessas células são trocadas durante a resolução iterativa do sistema de interface. E o segundo contendo os valores do estêncil externo, que são trocados durante a fase de montagem dos sistemas de equações. Os valores do estêncil externo são multiplicados pelas células recebidas dos subdomínios vizinhos (células da interface) e o resultado é adicionado em um vetor que é responsável pelo armazenamento das contribuições dos subdomínios vizinhos ao sistema de equações local.

A Figura 4.5 mostra a região de fronteira entre dois subdomínios P_0 e P_1, bem como as estruturas de dados necessárias ao subdomínio P_0 para a troca de informações com o

subdomínio P_1 durante a resolução iterativa do sistema de interface. As células das regiões E_0 (sublinhado simples) e E_1 (sublinhado duplo) correspondem ao estêncil externo de P_0 (no caso as células 7, 19, 31, 43, 55 e 67) e ao estêncil externo de P_1 (células 6, 18, 30, 42, 54 e 66), respectivamente.

Especificamente no caso de P_0 é armazenado o identificador do processo P_1 e as posições do vetor solução que devem ser enviadas para o processo P_1 (no caso 6, 18, 30, 42, 54 e 66). Essas células serão multiplicadas pelos valores do estêncil externo de P_1 e os resultados adicionados ao vetor de contribuições dos subdomínios vizinhos ao sistema de equações local de P_1. Mais especificamente os resultados serão adicionados nas posições do vetor correspondentes às células de fronteira de P_1 com P_0 (no caso 7, 19, 31, 43, 55 e 67).

O processo P_0 necessita armazenar os valores correspondentes ao seu estêncil externo. Esses valores são multiplicados pelos valores do vetor solução enviados por P_1 e adicionados ao vetor de contribuições dos subdomínios ao sistema de equações local de P_0 nas posições correspondentes às células das fronteiras artificiais de P_0 com P_1 (no caso 6, 18, 30, 42, 54 e 66).

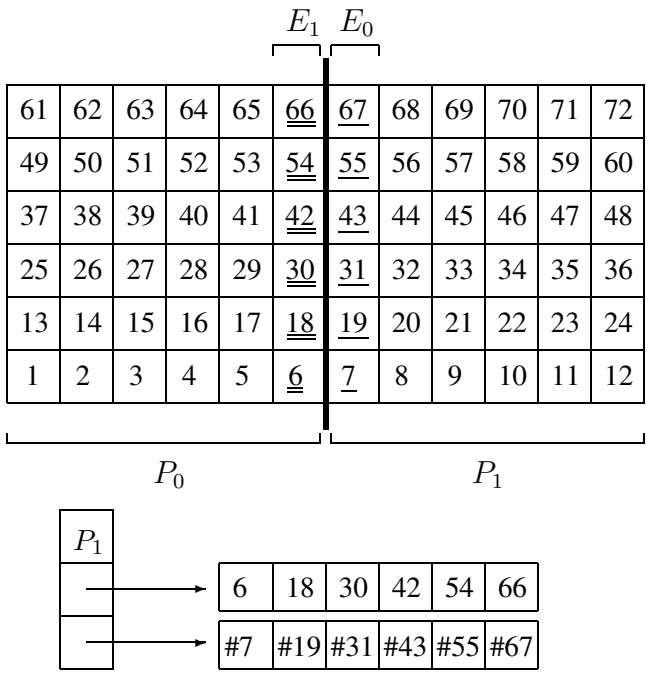

Figura 4.5: Estrutura de dados para a comunicação no método do complemento de Schur

Uma vez que a solução do sistema de interface é conhecida o sistema de equações correspondente às células internas de cada subdomínio pode ser resolvido de maneira totalmente independentemente. Para a resolução desses sistemas de equações utilizou-se o método do GC, uma vez que os sistemas correspondentes às células internas dos subdomínios também são SDP.

Para a exploração do paralelismo intra-nodal em *clusters* de PCs multiprocessados utilizou-se as abordagens já descritas na seção 4.4.4. Da mesma forma que em um método de Schwarz o uso de *threads* para a exploração do paralelismo intra-nodal resulta em uma diminuição no número de subdomínios que, por sua vez, provoca uma diminuição no tamanho das fronteiras artificiais e, conseqüentemente, no tamanho dos sistemas de interface. Além disso, a diminuição no número de subdomínios causa uma diminuição no volume de troca de mensagens.

4.6 Trabalhos Relacionados

Métodos de decomposição de domínio tem sido objeto de estudo de várias pesquisas nos últimos anos. Isso devido, principalmente, ao crescimento da importância dos computadores paralelos na computação científica. Essa seção objetiva fazer uma rápida revisão bibliográfica de trabalhos relacionados a métodos de decomposição de domínio, foco deste trabalho, bem como de algumas bibliotecas que disponibilizam implementações desses métodos.

Um estudo introdutório sobre métodos de decomposição de domínio pode ser encontrado em (MÜLLER; CHARÃO; SANTOS, 2003). Para um estudo mais completo sobre métodos de decomposição de domínio recomenda-se (CHAN; MATHEW, 1994) e (SMITH; BJORSTAD; GROPP, 1996). Nesses é feita uma discussão completa sobre métodos de decomposição com sobreposição de subdomínios (métodos de Schwarz) e métodos sem sobreposição de subdomínios (métodos de Schur).

Informações, bem como alguns resultados sobre métodos de Schwarz e métodos de Schur podem ser encontrados em trabalhos como: (SMITH, 1992), (CHARÃO, 2001), (GROOSS, 2001), (KUZNETSOV; LO; SAAD, 1997) e (NADEEM, 2001). Já trabalhos como (CAI; SAAD, 1993), (FLEMISH, 2001) e (CAI, 2002) tratam especificamente sobre métodos de decomposição de domínio com sobreposição (métodos de Schwarz). Por fim, em trabalhos como (SAAD; SOSONKINA, 1999), (ROUX, 1993) tem-se informações específicas sobre métodos de decomposição de domínio sem sobreposição (método do complemento de Schur).

Em relação a bibliotecas que disponibilizam implementações de métodos baseados na abordagem de decomposição de domínio pode-se citar as bibliotecas: Diffpack (DIFFPACK KERNEL AND TOOLBOXES DOCUMENTATION 4.0.00, 2004) e Ahpik (BENABDALLAH; CHARÃO; PLATEAU, 2001).

A biblioteca Diffpack consiste em um ambiente, desenvolvido em C++, para a resolução numérica de EDPs. Para a resolução de sistemas de equações em paralelo a biblioteca Diffpack disponibiliza as seguintes abordagens: a paralelização das operações de álgebra linear que compõem os métodos iterativos (matriz por vetor, produto escalar, etc.) e o uso de métodos de decomposição de domínio com sobreposição (métodos de Schwarz). Outra alternativa é o uso de métodos de decomposição de domínio como pré-condicionadores para métodos iterativos paralelizados.

As implementações desenvolvidas nesse trabalho diferem das existentes na biblioteca Diffpack por permitirem o uso de múltiplas *threads* na exploração do paralelismo intranodal. Na biblioteca Diffpack são disponibilizados apenas métodos de Schwaz com comunicação através de troca de mensagens. Como biblioteca de comunicação a Diffpack utiliza o padrão MPI (DIFFPACK KERNEL AND TOOLBOXES DOCUMENTATION 4.0.00, 2004).

Já a biblioteca Ahpik consiste em um ambiente, desenvolvido em C++, que provê abstrações para o desenvolvimento de *solvers* baseados na abordagem de decomposição de domínio para resolução de sistemas de equações em paralelo. Essas abstrações possibilitam ao desenvolvedor o uso de *multithreading* nas implementações de métodos de decomposição de domínio. Como biblioteca de comunicação a Ahpik utiliza a biblioteca Athapascan (BRIAT; GINZBURG; PASIN, 1998).

A principal diferença entre as implementações disponíveis na biblioteca Ahpik e as implementações desenvolvidas nesse trabalho é a forma de utilização das *threads*. A Ahpik baseia-se na forma de execução dos métodos de decomposição de domínio. Geralmente esses métodos consistem na resolução de sistemas locais e algumas operações

correspondentes a interface (*send*, *recv* e verificação de convergência). A idéia básica do Ahpik consiste em atribuir cada uma dessas tarefas a *threads* diferentes sendo possível dessa forma a sobreposição de comunicação com computação (CHARÃO; CHARPENTIER; STEIN, 2003). Já as implementações desenvolvidas nesse trabalho baseiam-se no uso de *multithreading* para a solução dos sistemas locais, isto é, os *solvers* locais são paralelizados através do uso de múltiplas *threads*. Uma vez que diferentes *threads* podem ser executadas em diferentes processadores essa abordagem é atrativa para *clusters* de PCs multiprocessados.

Embora as implementações desenvolvidas nesse trabalho não possibilitem a sobreposição de comunicação e computação essa funcionalidade ser facilmente incorporada através do uso de primitivas não bloqueantes disponíveis no padrão MPI.

4.7 Considerações Finais

Neste capítulo apresentou-se conceitos relativos à resolução de sistemas de equações em paralelo. Inicialmente introduziu-se técnicas, bem como algumas ferramentas que podem ser utilizadas para o particionamento do domínio computacional. Para um estudo mais completo sobre particionamento do domínio recomenda-se (SCHLOEGEL; KARYPIS; KUMAR, 2000), (SANTOS; DORNELES, 2001) e (DORNELES, 2003).

Em um segundo momento contextualizou-se as principais estratégias para a resolução de sistemas de equações em paralelo que são decomposição de dados e decomposição de domínio. A ênfase foi dada aos métodos de decomposição de domínio, pois essa foi abordagem adotada no desenvolvimento deste trabalho. Para um estudo introdutório sobre abordagens para a resolução de sistemas em paralelo recomenda-se (MÜLLER; CHARÃO; SANTOS, 2003). Maiores informações sobre decomposição de dados podem ser encontradas em (CANAL, 2000) e (PICININ, 2003) e para maiores informações sobre métodos de decomposição de domínio recomenda-se (CHAN; MATHEW, 1994) e (SMITH; BJORSTAD; GROPP, 1996).

No próximo capítulo são apresentados os resultados obtidos com paralelizações desenvolvidas nesse trabalho. É feito ainda um estudo sobre a contenção de memória e um estudo comparativo entre métodos de decomposição de domínio e decomposição de dados.

5 RESULTADOS OBTIDOS

Neste capítulo são apresentados os resultados obtidos com as paralelizações desenvolvidas neste trabalho. Essas foram implementadas em linguagem C, utilizando o compilador *gcc* 2.95.4 sobre o sistema operacional Debian Linux com *kernel* 2.4.21. Como biblioteca de troca de mensagens foi utilizado o MPICH 1.2.2, e como biblioteca de *threads* utilizou-se o pré-compilador Omni 1.4a.

Os testes foram realizados no *cluster labtec* apresentado na seção 2.1.4. Para a realização dos testes utilizou-se sistemas resultantes da discretização do Lago Guaíba com 3 diferentes tamanhos de células: $\triangle x = \triangle y = 200m$, que resulta em sistemas de equações com 11.506 equações, $\triangle x = \triangle y = 100m$, que resulta em sistemas de equações com 46.024 equações, e $\triangle x = \triangle y = 50m$, que resulta em sistemas de equações com 184.096 equações. Esses sistemas são chamados, respectivamente, de Guaíba200, Guaíba100 e Guaíba50. Para a solução dos sistemas locais (subdomínios) utilizou-se o método do GC com um erro de 10^{-7} e para a solução global, via MDDs (métodos de Schwarz e método do complemento de Schur), o erro é de 10^{-3}.

Na tomada de tempo foram feitas 20 execuções de cada implementação e o tempo considerado foi a média aritmética dessas. Os resultados não foram submetidos a análise estatística. Apenas os resultados anômalos foram descartados.

Já para uma verificação da qualidade númerica das versões paralelas desenvolvidas utilizou-se o procedimento descrito em (RIZZI, 2002). Nesse considera-se como domínio computacional uma malha de 30x30 pontos, que é particionada na direção Y em dois subdomínios, onde cada um deles possui 30x15 células. Considerando-se que a solução numérica monoprocessada é, numericamente, a correta (como pode ser visto em (RIZZI, 2002), um modo de avaliar a solução paralela é empregar a métrica do erro relativo para todas as células do domínio computacional. Nesse caso, se x_i^m e x_i^p são, respectivamente, as soluções monoprocessada e paralela, uma medida de qualidade pode ser obtida pela expressão $E_r = |x_i^m - x_i^p|/x_i^p$.

A apresentação dos resultados está organizada da seguinte forma: inicialmente são apresentados e comparados os resultados obtidos com os métodos de Schwarz. Em seguida são apresentados os resultados obtidos utilizando o método do complemento de Schur. É apresentado ainda, um estudo comparativo entre paralelizações do método do GC, utilizando as abordagens de decomposição de domínio e decomposição de dados. E por fim, são apresentados estudos referentes a contenção de memória, muito comum em arquiteturas com memória compartilhada.

5.1 Métodos de Schwarz

Nesta seção são apresentados os resultados obtidos com os métodos de Schwarz. Primeiramente serão apresentados e discutidos os resultados obtidos com o método multiplicativo de Schwarz. Após serão apresentados os resultados obtidos com o método aditivo de Schwarz. E, por fim, é feito um comparativo entre os resultados obtidos com esses métodos.

5.1.1 Método Multiplicativo de Schwarz

Para a análise do método multiplicativo de Schwarz foram considerados: os resultados obtidos com as diferentes heurísticas de coloração; o número de iterações e tempo de execução do método. Além disso, são apresentadas comparações das duas estratégias usadas para exploração do paralelismo em *clusters* multiprocessados, que são o uso de apenas múltiplos processos e o uso de múltiplos processos com múltiplas *threads*.

5.1.1.1 Heurísticas de Coloração

O método multiplicativo de Schwarz é naturalmente seqüencial e para obter paralelismo nesse método foi necessário o uso de técnicas de coloração para os subdomínios. Neste trabalho, analisou-se um algoritmo guloso com três diferentes heurísticas. Essas heurísticas foram descritas na seção 4.4.1.1.

Na primeira heurística a coloração é feita seguindo a ordem seqüencial de numeração dos vértices. Na segunda heurística, a coloração dos vértices é feita em ordem decrescente do grau dos vértices. E na terceira heurística, utiliza-se a multiplicação da matriz de adjacências por um vetor coluna. Essas heurísticas são chamadas aqui, respectivamente, de seqüencial, grau dos vértices e matriz de adjacência.

As Figuras 5.1, 5.2 e 5.3 mostram os resultados obtidos com essas diferentes heurísticas utilizando como domínio o Guaíba200, Guaíba100 e Guaíba50, respectivamente.

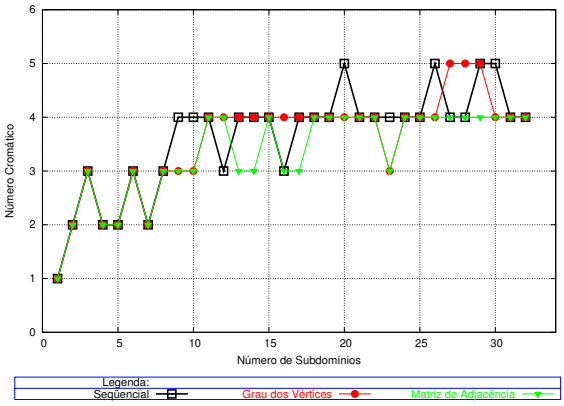

Figura 5.1: Guaíba200 - Heurísticas de coloração

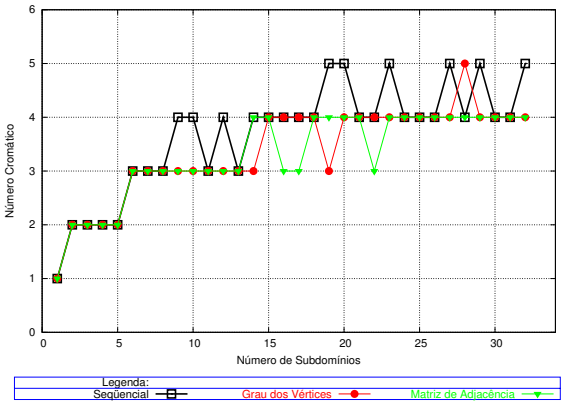

Figura 5.2: Guaíba100 - Heurísticas de coloração

Observa-se nessas figuras, que a heurística que usa a matriz de adjacência resultou em um número menor de cores que as demais heurísticas, principalmente com o aumento no número de subdomínios. Um menor número de cores provoca uma diminuição nos passos seqüenciais do algoritmo e, conseqüentemente, uma melhora no tempo de execução do mesmo. Na seção 5.1.1.3 pode-se observar que o método multiplicativo de Schwarz com essa heurística apresentou, em geral, melhores resultados que com as demais heurísticas.

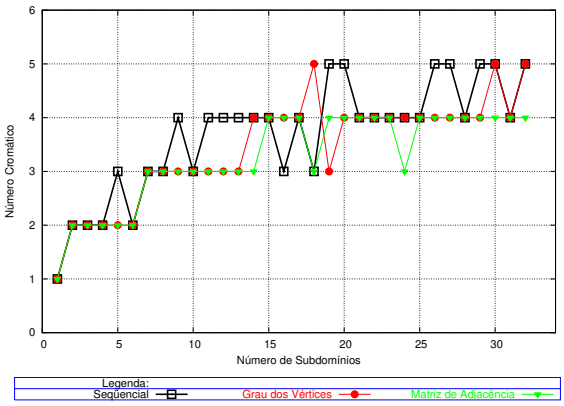

Figura 5.3: Guaíba50 - Heurísticas de coloração

Observa-se ainda nos gráficos 5.1, 5.2 e 5.3 que, em geral, a heurística que utiliza-se do grau dos vértices apresentou resultados um pouco melhores que a heurística que segue a numeração seqüencial dos vértices.

5.1.1.2 Número de Iterações

As Figuras 5.4, 5.5 e 5.6 ilustram o número de iterações necessárias para a convergência do método multiplicativo de Schwarz.

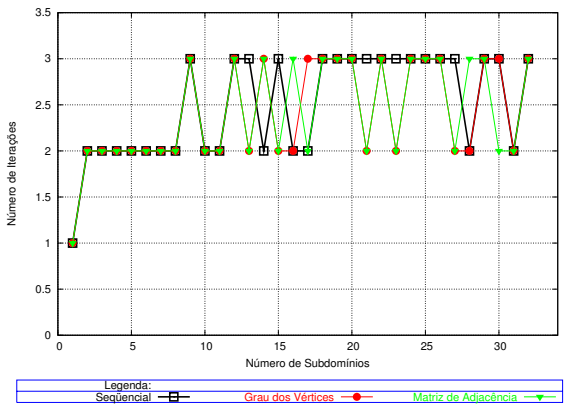

Figura 5.4: Guaíba200 - Número de iterações do método multiplicativo de Schwarz

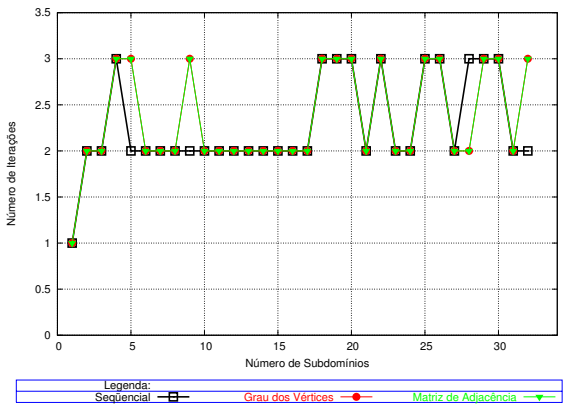

Figura 5.5: Guaíba100 - Número de iterações do método multiplicativo de Schwarz

Um resultado que pode ser observado nesses gráficos é que a heurística de coloração utilizada acaba afetando a convergência do método multiplicativo de Schwarz. Isto ocorre porque o uso de diferentes heurísticas altera a ordem da troca das condições de contorno entre os subdomínios, o que acaba influenciando na convergência do método.

Observa-se também, que para os domínios Guaiba100 e Guaiba50 onde ocorre um aumento no tamanho dos subdomínios e, conseqüentemente, um aumento das fronteiras artificiais, a influência da heurística de coloração torna-se bem menor.

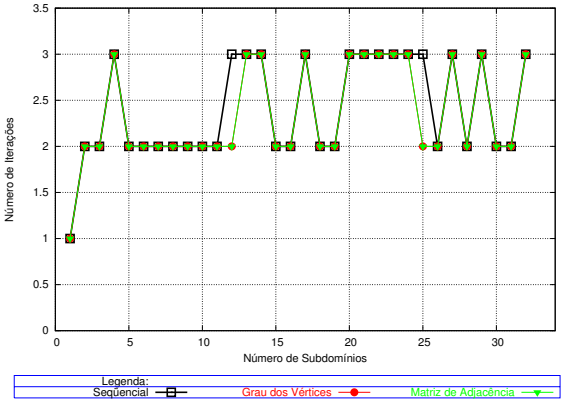

Figura 5.6: Guaíba50 - Número de iterações do método multiplicativo de Schwarz

5.1.1.3 Tempo de Execução

As Figuras 5.7, 5.8 e 5.9 apresentam o tempo de execução do método multiplicativo de Schwarz sem *threads*. Embora as máquinas do *cluster labtec* sejam duais, para eliminar possíveis problemas de contenção de memória utilizou-se nos testes apenas um processador de cada nodo.

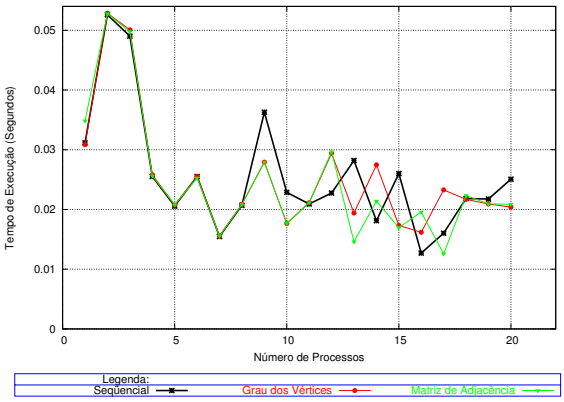

Figura 5.7: Guaíba200 - Tempo de execução do método multiplicativo de Schwarz

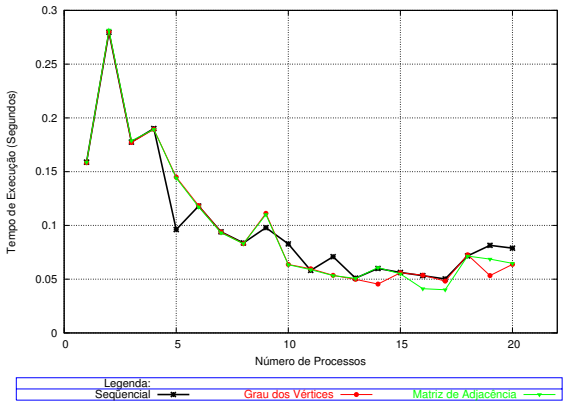

Figura 5.8: Guaíba100 - Tempo de execução do método multiplicativo de Schwarz

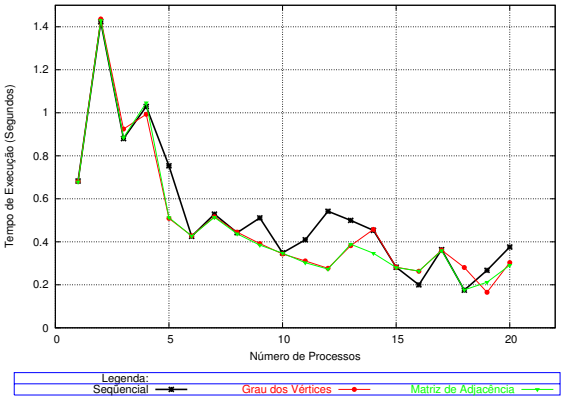

Figura 5.9: Guaíba50 - Tempo de execução do método multiplicativo de Schwarz

Pode-se observar nesses testes (utilizando o domínio do Lago Guaíba) que o método multiplicativo de Schwarz apresentou um baixo desempenho. Seu tempo de execução é altamente afetado pelo número de cores utilizada, já que o número de cores utilizadas define o número de passos seqüenciais do algoritmo.

Observa-se ainda, que o método multiplicativo com a heurística que usa a matriz de adjacência apresentou, em média, os melhores resultados. Isto porque essa heurística, em geral, resultou em um menor número de cores, o que diminui os passos seqüenciais do algoritmo.

Na Figura 5.10 tem-se uma visualização da execução em paralelo do método multiplicativo de Schwarz para o domínio Guaiba100 com 8 processos. Para a geração da figura utilizou-se a biblioteca MPE (*MultiProcessing Environment*) disponível com a distribuição MPICH. Essa biblioteca contém rotinas que permitem a geração de arquivos de LOG, que descrevem as etapas de cada execução de um programa MPI (CHAN; GROPP; LUSK, 2003). Para a visualização dos arquivos de LOG utilizou-se a ferramenta Jumpshot-4 (CHAN et al., 2003).

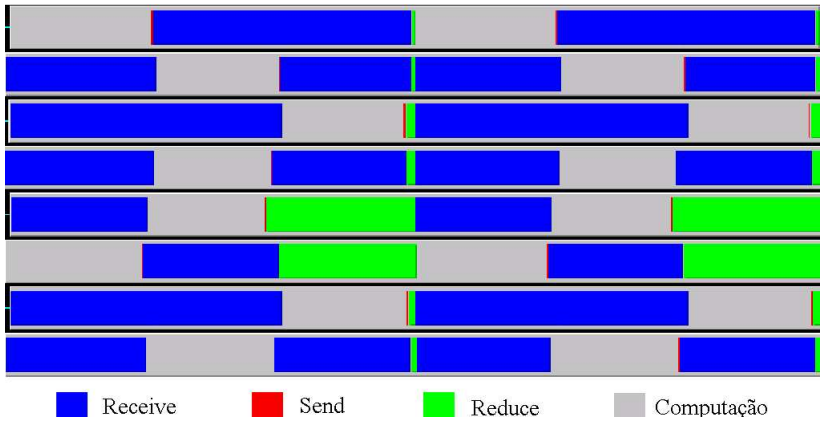

Figura 5.10: Execução do método multiplicativo de Schwarz

Na Figura 5.10 podem ser identificados os passos seqüenciais em cada ciclo do método multiplicativo de Schwarz. Em cada ciclo primeiramente são resolvidos os subdomínios com cor 0, após os subdomínios de cor 1 e por fim os subdomínios de cor 2. Desta forma a quantidade de passos seqüenciais do algoritmo corresponde ao número de cores utilizadas. Pode-se ver na Figura 5.10 que grande parte do tempo os processos ficam bloqueados em uma operação de *receive* (cor azul) a espera das condições de contorno dos subdomínios vizinhos, o que justifica o baixo desempenho dessa variante do método de Schwarz. A operação de *reduce* (cor verde) é utilizada em uma operação de produto escalar. Essa operação exige a sincronização de todos os processos e é usada ao final de cada ciclo do método multiplicativo para o cálculo da norma do resíduo e verificação da convergência do método.

5.1.1.4 *Múltiplos Processos Vs. Múltiplas Threads*

As Figuras 5.11, 5.12 e 5.13 apresentam um comparativo entre as duas estratégias usadas nesse trabalho para a exploração do paralelismo em *clusters* multiprocessados, as quais foram: o uso apenas de múltiplos processos e o uso de múltiplos processos com múltiplas *threads*. As duas estratégias fazem uso dos 2 processadores das máquinas duais e são afetadas pelo problema de contenção de memória.

Para essa comparação optou-se pela versão do método multiplicativo com a heurística que usa a matriz de adjacência, uma vez que ela, em geral, apresentou os melhores resultados. Na execução dos testes foram utilizados até 16 nodos do *cluster labtec*. Na execução da implementação que utiliza apenas múltiplos processos foram disparados 2 processos em cada nodo. Já na implementação utilizando múltiplos processos em conjunto com múltiplas *threads* foi disparado 1 processo com 2 *threads* em cada nodo.

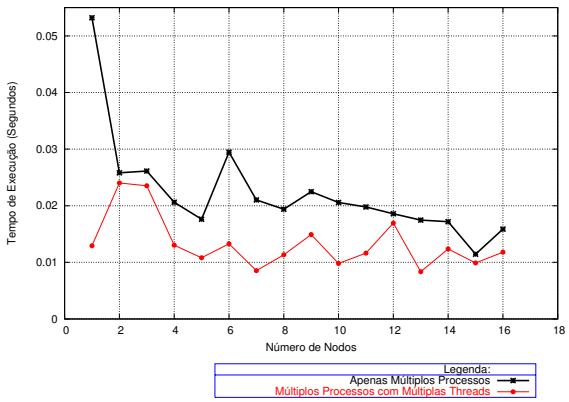

Figura 5.11: Guaíba200 - Processos Vs. *threads* no método multiplicativo de Schwarz

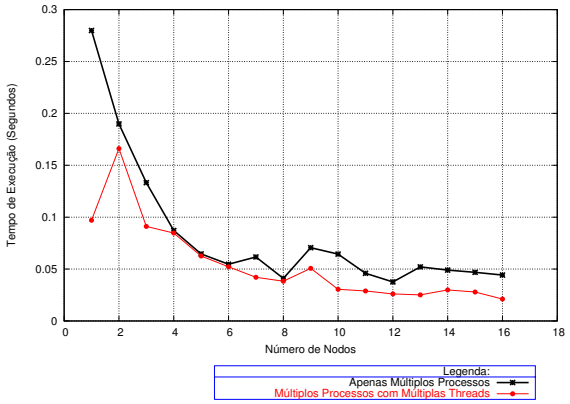

Figura 5.12: Guaíba100 - Processos Vs. *threads* no método multiplicativo de Schwarz

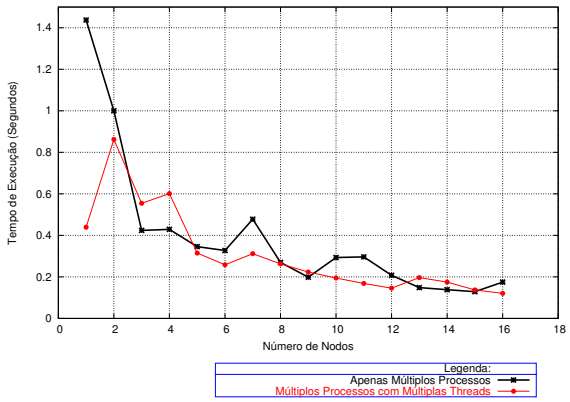

Figura 5.13: Guaíba50 - Processos Vs. *threads* no método multiplicativo de Schwarz

As Figuras 5.11 e 5.12 mostram que para o Guaiba200 e Guaiba100, respectivamente, a estratégia que usa apenas múltiplos processos apresenta um desempenho inferior à que usa múltiplos processos com múltiplas *threads*. Como mencionado na seção 4.4.4, nessa estratégia o particionamento de domínio é feito de acordo com o número de processadores

disponíveis e a exploração do paralelismo intra-nodal através de múltiplos processos. Já na abordagem que usa múltiplas *threads* o particionamento é feito de acordo com o número de nodos e a exploração do paralelismo intra-nodal é obtida através do uso de múltiplas *threads*.

Assim, na estratégia que usa apenas múltiplos processos, o número de subdomínios é o dobro daquele número da estratégia que usa múltiplas *threads*. O aumento no número de subdomínios afeta a convergência do método, provoca uma maior redundância de cálculo e um maior volume de comunicação. Além disso, o aumento no número de subdomínios, geralmente, ocasiona um aumento no número de cores utilizadas, o que acaba ocasionando uma queda no desempenho do método multiplicativo de Schwarz.

Observa-se ainda que com um único nodo o uso de múltiplas *threads* o desempenho computacional é muito superior ao uso de múltiplos processos. Isto ocorre porque com múltiplos processos o domínio computacional foi particionado em dois subdomínios e o método multiplicativo exige duas iterações para convergir, já com múltiplas *threads* o domínio computacional não é particionado e a solução deste é calculada uma única vez por duas *threads*.

Já a Figura 5.13 mostra que o desempenho da estratégia que usa apenas múltiplos processos em determinados casos apresenta um melhor desempenho. Isto ocorre porque nesses casos, como pode ser visto na Figura 5.6, o número de iterações para convergência com 3, 4, 13, 14 subdomínios é superior ao número de iterações do método com 6, 8, 26 e 28 subdomínios, respectivamente. Mas, em geral, a abordagem que utiliza-se de múltiplas *threads* apresenta um melhor desempenho.

Através dos resultados obtidos é possível concluir que a diferença entre o desempenho das estratégias é decorrente do modo como é feito o particionamento do domínio nessas. Esse afeta sensivelmente no número de cores utilizadas, bem como pode afetar no número de iterações necessárias para a convergência do método. É importante destacar que essas diferenças de desempenho não são decorrentes, simplesmente, das vantagens do uso de *threads* sobre o uso de processos. Não é objetivo deste trabalho fazer uma análise comparativa entre o desempenho de *threads* e processos, mas sim um comparativo entre diferentes abordagens na paralelização de métodos de decomposição de domínio em *clusters* de PCs multiprocessados.

5.1.2 Método Aditivo de Schwarz

Para a análise de desempenho do método aditivo de Schwarz considerou-se o número de iterações e o tempo de execução. Além disso, foi feita uma análise comparativa do uso de apenas múltiplos processos com o uso de processos com múltiplas *threads* para a exploração do paralelismo em *clusters* multiprocessados.

5.1.2.1 Número de Iterações

As Figuras 5.14, 5.15 e 5.16 mostram o número de iterações para a convergência do método aditivo de Schwarz. Observa-se que o número de iterações necessárias para a convergência no método aditivo de Schwarz sofre uma variação com a alteração no número de subdomínios, como justificado na seção 4.4.3.

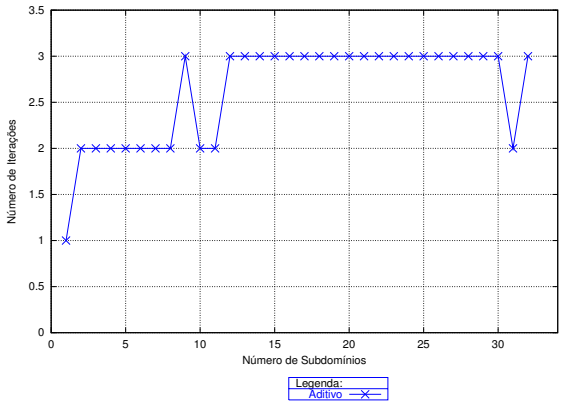

Figura 5.14: Guaíba200 - Número de iterações do método aditivo de Schwarz

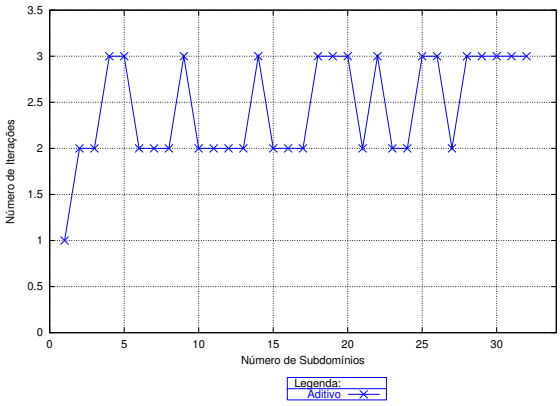

Figura 5.15: Guaíba100 - Número de iterações do método aditivo de Schwarz

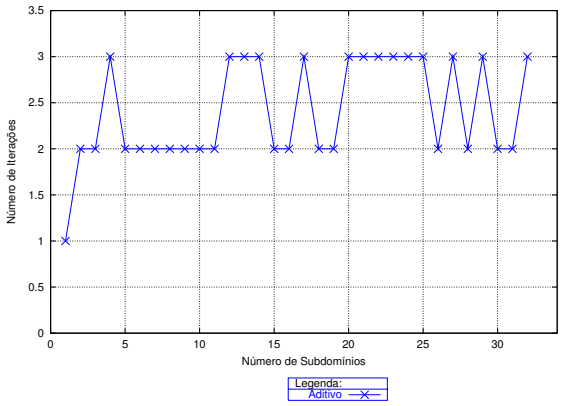

Figura 5.16: Guaíba50 - Número de iterações do método aditivo de Schwarz

5.1.2.2 Tempo de Execução

As Figuras 5.17, 5.18 e 5.19 ilustram o tempo de execução do método aditivo de Schwarz. Para esses testes foram utilizados até 20 nodos do cluster, sendo que para eliminar possíveis influências de contenção de memória utilizou-se apenas um processador de cada nodo. Observa-se que os resultados obtidos com as paralelizações desenvolvidas foram satisfatórios, apresentando uma boa escalabilidade. É possível observar no gráfico a presença de picos. Esses picos são causados pelo aumento do número de iterações.

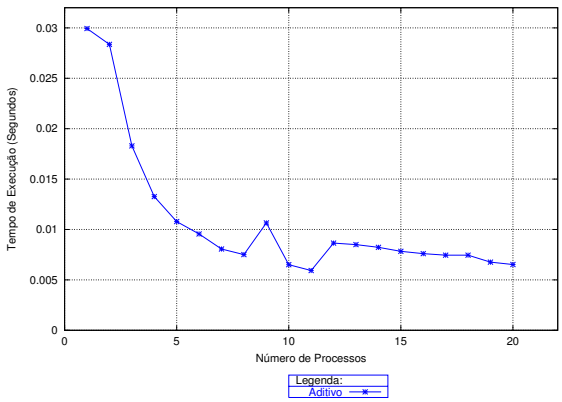

Figura 5.17: Guaíba200 - Tempo de execução do método aditivo de Schwarz

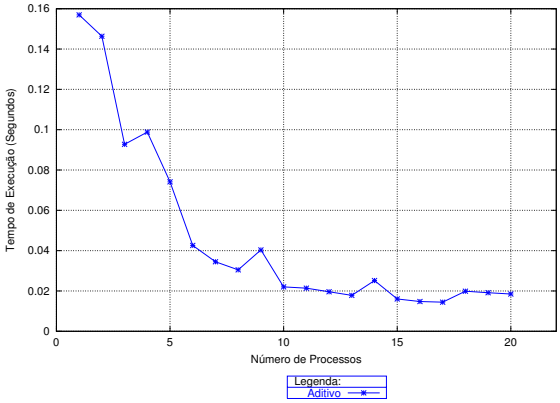

Figura 5.18: Guaíba100 - Tempo de execução do método aditivo de Schwarz

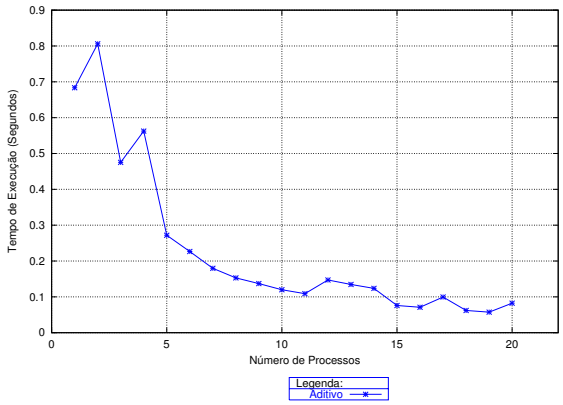

Figura 5.19: Guaíba50 - Tempo de execução do método aditivo de Schwarz

Nas Figuras 5.20, 5.21 e 5.22 tem-se os gráficos de *speedup* do método aditivo de Schwarz para os domínios Guaiba200, Guaiba100 e Guaiba50, respectivamente. Observa-se que com o aumento do domínio computacional (Guaiba100 e Guaiba50) e conseqüentemente, da granularidade do problema, ocorreu um aumento nos *speedups* obtidos. Observa-se ainda nos gráficos pontos de quedas significativas nos *speedups*. Esses pontos são decorrentes do aumento no número de iterações para a convergência do método.

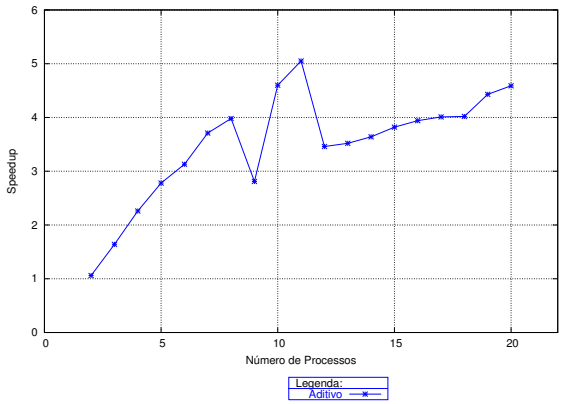

Figura 5.20: Guaíba200 - Speedup do método aditivo de Schwarz

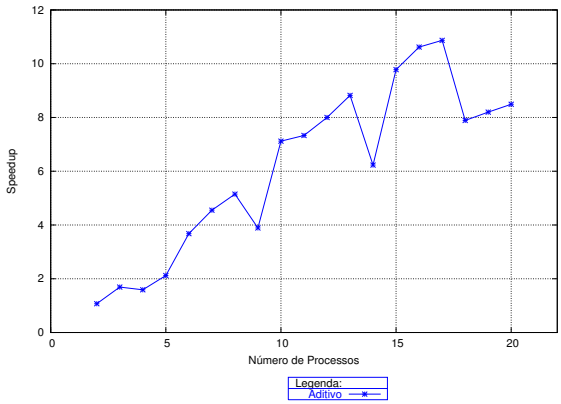

Figura 5.21: Guaíba100 - Speedup do método aditivo de Schwarz

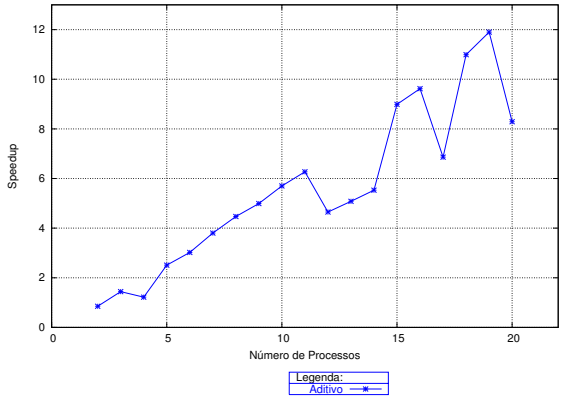

Figura 5.22: Guaíba50 - Speedup do método aditivo de Schwarz

A Figura 5.23 ilusta a execução do método aditivo de Schwarz para o domínio do Guaiba100 com 8 processos. É possível verificar nessa figura que o método aditivo de Schwarz apresenta um potencial de paralelismo muito maior que o método multiplicativo de Schwarz, como pode ser visto na seção 5.1.3. Nesse método as soluções dos subdomínios são obtidas simultaneamente, bastando trocar as condições de contorno no final de cada iteração. A operação de *reduce* (cor verde) corresponde a uma operação de produto escalar. Essa operação exige a sincronização de todos os processos da aplicação e é utilizada no final de cada ciclo do método aditivo de Schwarz para o cálculo da norma do resíduo na verificação da convergência do método.

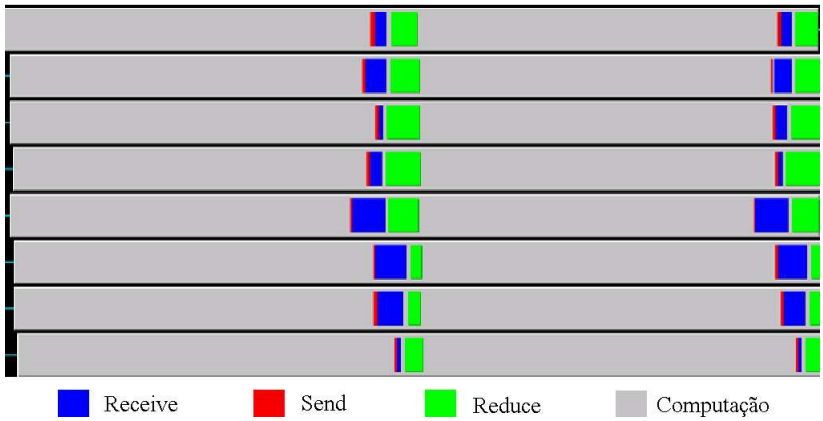

Figura 5.23: Execução do método aditivo de Schwarz

5.1.2.3 *Múltiplos Processos Vs. Múltiplas Threads*

As Figuras 5.24, 5.25 e 5.26 mostram um comparativo do uso do método aditivo de Schwarz em *clusters* multiprocessados. Na realização dos testes foram usados os mesmos procedimentos adotados para os testes do método multiplicativo de Schwarz nesse tipo de arquitetura.

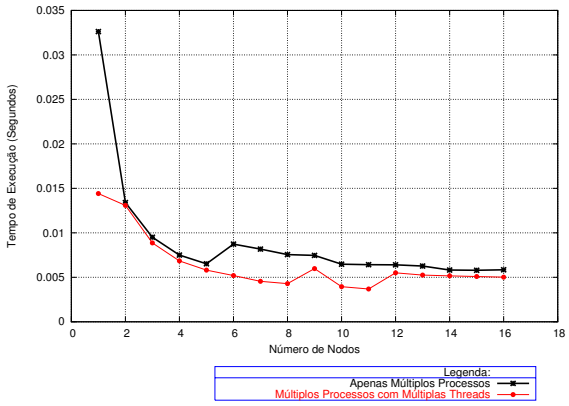

Figura 5.24: Guaíba200 - Processos Vs. *threads* no método aditivo de Schwarz

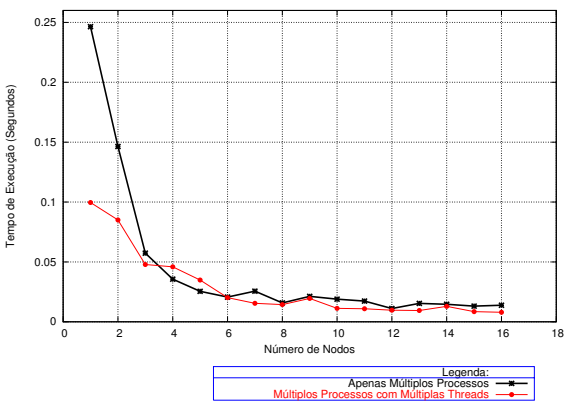

Figura 5.25: Guaíba100 - Processos Vs. *threads* no método aditivo de Schwarz

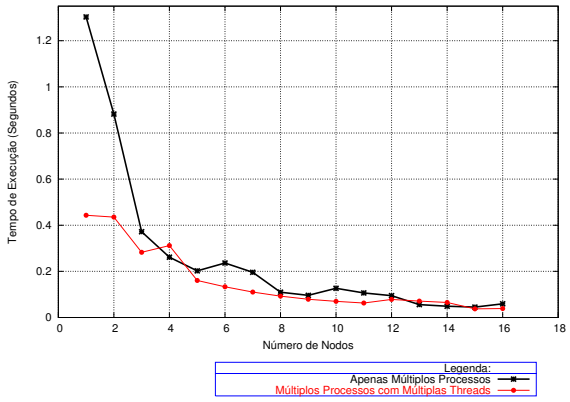

Figura 5.26: Guaíba50 - Processos Vs. *threads* no método aditivo de Schwarz

Essas figuras mostram que a principal vantagem, da mesma forma que no método multiplicativo de Schwarz, do uso de múltiplas *threads* para a exploração do paralelismo intra-nodal refere-se à redução do número de subdomínios gerados pelo particionamento. Como já mencionado, o uso da abordagem que utiliza múltiplas *threads* provoca uma diminuição no número de subdomínios e em muitos casos no número de iterações.

Na Figura 5.25 o desempenho da estratégia que usa apenas múltiplos processos apresenta um melhor desempenho para 4 e 5 nodos, isto porque, o número de iterações com 4 e 5 subdomínios é maior que com 8 e 10 subdomínios. O mesmo acontece para 4, 13 e 14 nodos na Figura 5.26. Mas em geral, a abordagem que utiliza-se de múltiplas *threads* apresenta um melhor desempenho em *clusters* multiprocessados.

Além disso, observou-se que em situações onde o número de iterações necessárias para a convergência é o mesmo, o uso de múltiplas *threads* apresentou tempo de execução menor que o uso de apenas múltiplos processos. Isto ocorre porque na abordagem que utiliza somente processos há uma quantidade maior de células associadas a diferentes subdomínios (região de sobreposição) o que acaba ocasionando uma maior redundância de cálculo. E com o uso de múltiplas *threads*, ocorre uma diminuição no volume de troca de mensagens entre os nodos do *cluster*, porque com a diminuição no número de subdomínios ocorre uma redução no número de processos que necessitam se comunicar ao final de cada iteração do método.

5.1.3 Método Multiplicativo de Schwarz Vs. Método Aditivo de Schwarz

Nessa seção é feito um comparativo entre os métodos multiplicativo de Schwarz e aditivo de Schwarz. Foram comparados o número de iterações e o tempo de execução das implementações que utilizam-se de múltiplas *threads* para a exploração do paralelismo intra-nodal. Para a comparação dos métodos utilizou-se a versão do método multiplicativo de Schwarz com a heurística que usa a matriz de adjacência pois essa, em geral, apresentou os melhores resultados.

5.1.3.1 Número de Iterações

As Figuras 5.27, 5.28 e 5.29 ilustram uma comparação entre os números de iterações necessárias para a convergência dos métodos multiplicativo de Schwarz e aditivo de Schwarz.

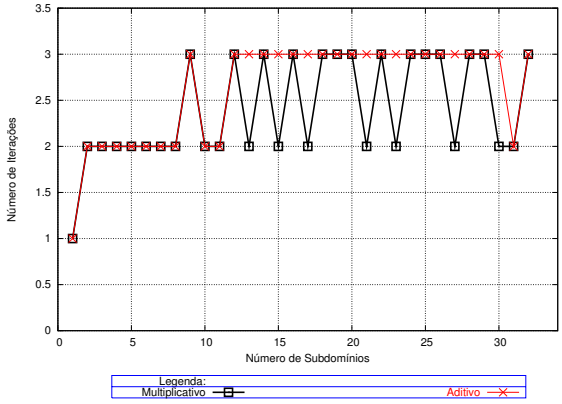

Figura 5.27: Guaíba200 - Número de iterações: método multiplicativo de Schwarz Vs. método aditivo de Schwarz

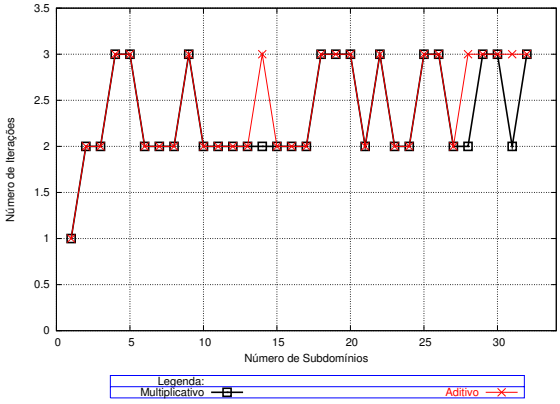

Figura 5.28: Guaíba100 - Número de iterações: método multiplicativo de Schwarz Vs. método aditivo de Schwarz

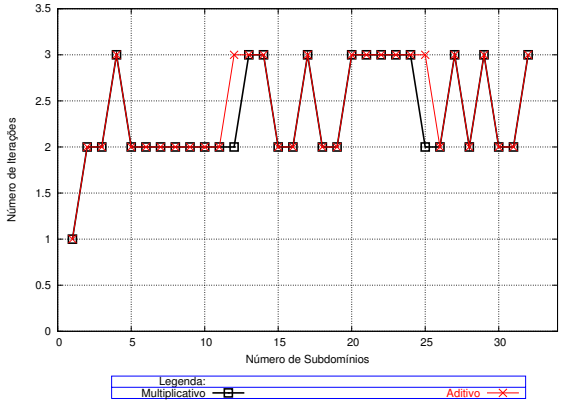

Figura 5.29: Guaíba50 - Número de iterações: método multiplicativo de Schwarz Vs. método aditivo de Schwarz

Observa-se que para o Guaiba200, o método multiplicativo necessita, em vários casos, de um número menor de iterações. Já para os domínios Guaiba100 e Guaiba50, onde ocorre um aumento significativo no número de células dos subdomínios (e, conseqüentemente, no número de células pertencentes às fronteiras artificiais) observa-se que o método multiplicativo, em grande parte dos casos, não ocasiona melhorias no número de iterações. É importante destacar que esses resultados são decorrentes, também, das características dos sistemas de equações gerados pelo modelo HIDRA.

5.1.3.2 Tempo de Execução

Nessa seção é feito um comparativo dos tempos de execução dos métodos multiplicativo de Schwarz e aditivo de Schwarz. Nos testes utilizou-se a abordagem que utiliza-se de múltiplas *threads* na exploração do paralelismo intra-nodal. Nas Figuras 5.30, 5.31 e 5.32 tem-se o tempo de execução dos métodos.

Observa-se que o método multiplicativo de Schwarz apresentou um desempenho muito inferior ao método aditivo de Schwarz. Esperava-se que a diferença no número de iterações entre os métodos acabasse compensando a seqüencialidade do método multiplicativo de Schwarz, mas na prática isso não ocorreu.

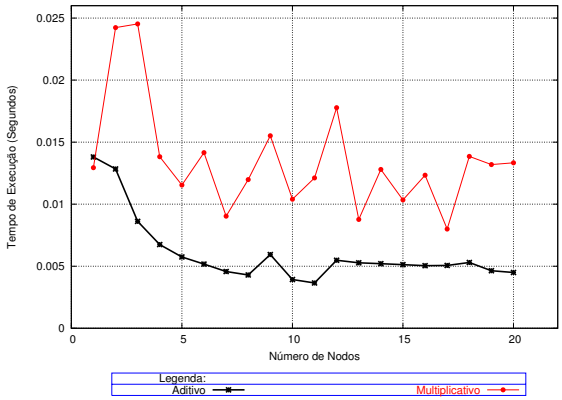

Figura 5.30: Guaíba200 - Tempo de execução: método multiplicativo de Schwarz Vs. método aditivo de Schwarz

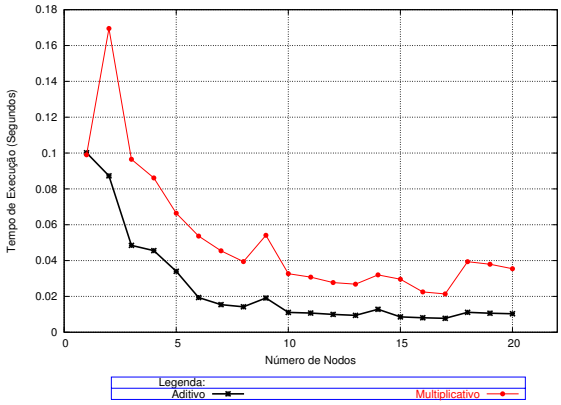

Figura 5.31: Guaíba100 - Tempo de execução: método multiplicativo de Schwarz Vs. método aditivo de Schwarz

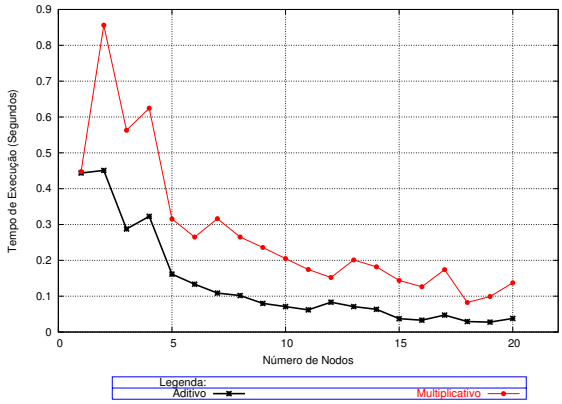

Figura 5.32: Guaíba50 - Tempo de execução: método multiplicativo de Schwarz Vs. método aditivo de Schwarz

Nas Figuras 5.33 e 5.34 é feito um comparativo do comportamento dos métodos multiplicativo de Schwarz e aditivo de Schwarz quando executados em paralelo para o domínio Guaiba100 com 7 processos. É fácil identificar que no método multiplicativo de Schwarz os processos (subdomínios) com cores diferentes ficam grande parte do tempo bloqueados (em uma operação de *receive*) à espera das condições de contorno dos subdomínios vizinhos. Já no método aditivo de Schwarz os processos (subdomínios) são executados simultaneamente. Isto justifica a grande diferença de desempenho dos métodos.

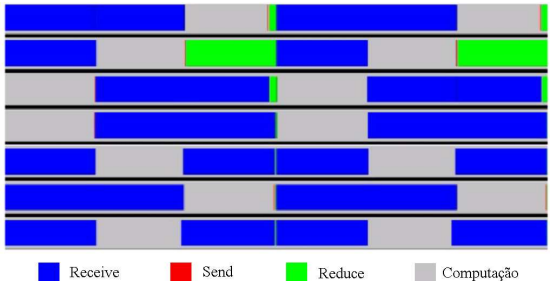

Figura 5.33: Execução do método multiplicativo de Schwarz

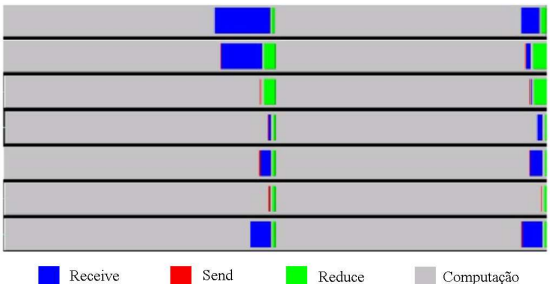

Figura 5.34: Execução do método aditivo de Schwarz

5.2 Método do Complemento de Schur

Nessa seção são apresentados os resultados obtidos no método do complemento de Schur. Inicialmente são apresentados os tempos de execução obtidos e após uma comparação das estratégias utilizadas para a exploração do paralelismo em *clusters* multiprocessados.

5.2.1 Tempo de Execução

Como já mencionado, para o cálculo das aproximações das inversas locais no método do complemento de Schur foram utilizadas aproximações polinomiais com $k = 0$ e $k = 1$. Nas Figuras 5.35, 5.36 e 5.37 tem-se o tempo de execução das implementações desenvolvidas. Nos testes utilizou-se apenas um processador de cada nodo para eliminar possíveis influências da contenção de memória.

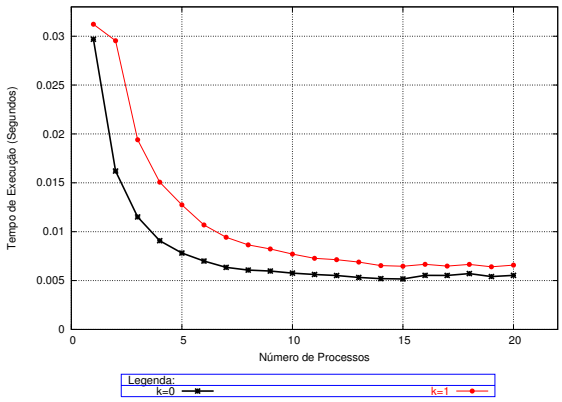

Figura 5.35: Guaíba200 - Tempo de execução do método do complemento de Schur

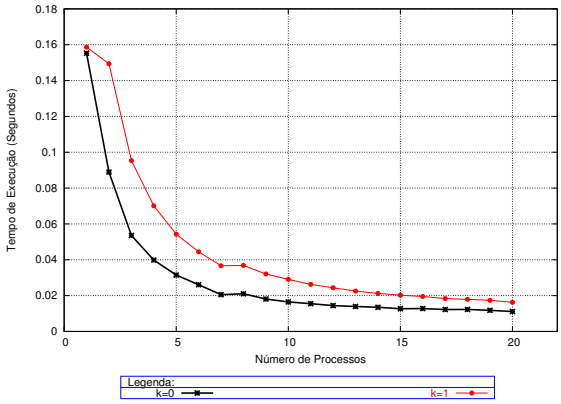

Figura 5.36: Guaíba100 - Tempo de execução do método do complemento de Schur

Observa-se nas figuras que o método do complemento de Schur com aproximações polinomiais com $k = 0$ apresentou um tempo de execução inferior ao método do complemento de Schur com $k = 1$. Isto ocorre devido ao baixo custo computacional para o cálculo das inversas locais. Observa-se que as implementações apresentaram um bom desempenho computacional, além de uma boa escalabilidade.

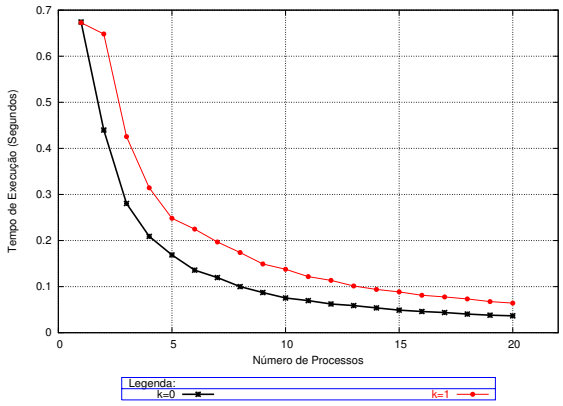

Figura 5.37: Guaíba50 - Tempo de execução do método do complemento de Schur

Na Figuras 5.38, 5.39 e 5.40 tem-se os *speedups* obtidos com o método do complemento de Schur utilizando aproximações polinomiais com $k = 0$ e $k = 1$. Observa-se que com o aumento do domínio computacional ocorre um aumento na eficiência das paralelizações desenvolvidas, devido a um aumento da carga computacional e, conseqüentemente, da granularidade do problema.

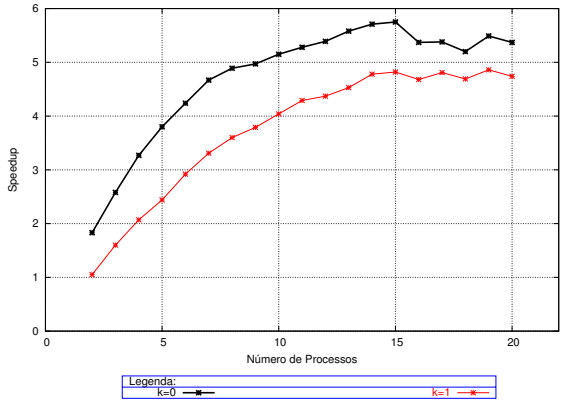

Figura 5.38: Guaíba200 - Speedup do método do complemento de Schur

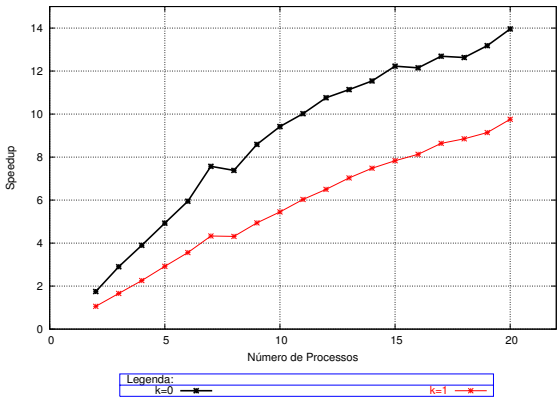

Figura 5.39: Guaíba100 - Speedup do método do complemento de Schur

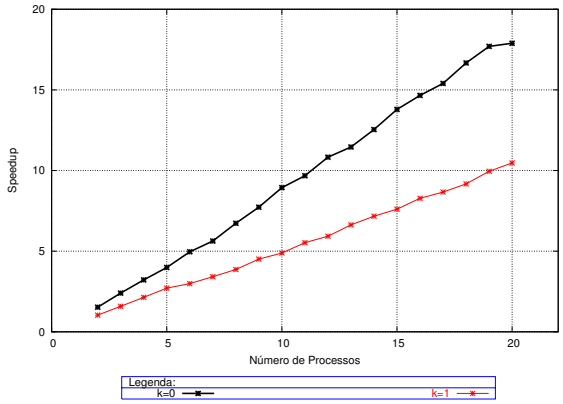

Figura 5.40: Guaíba50 - Speedup do método do complemento de Schur

Observa-se ainda que o método do complemento de Schur com aproximações polinomiais com $k = 0$ apresentou *speedups* elevados. É importante destacar que apesar do ótimo desempenho computacional o método do complemento de Schur com aproximações $k = 0$ resultou em soluções com baixa qualidade numérica, apresentando um erro relativo máximo superior a $9,9\%$. Já o método do complemento de Schur com aproximações $k = 1$ apresentou um erro relativo máximo da ordem de $3,37\%$.

Na Figura 5.41 tem-se o comportamento do método do complemento de Schur com aproximações polinomiais de $k = 1$ quando executado em paralelo para o domínio Guaiba100 com 8 processos. É possível verificar que o método do complemento de Schur apresenta um bom potencial de paralelismo.

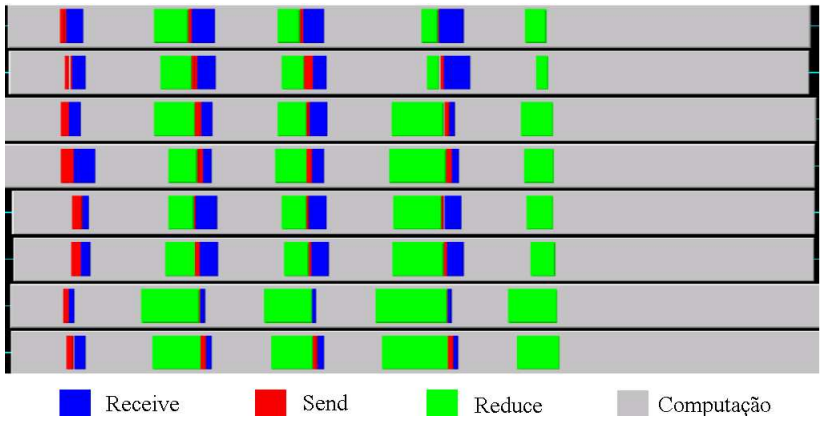

Figura 5.41: Execução do método do complemento de Schur

É possível identificar na Figura 5.41 as trocas de informações necessárias para o cálculo iterativo da solução do sistemas de interface. Observa-se, ainda, que a partir da obtenção das soluções dos sistemas de interfaces os sistema de equações correspondentes às células internas são resolvidos independentemente, isto é, sem a necessidade de nenhum tipo de comunicação.

5.2.2 Múltiplos Processos Vs. Múltiplas *Threads*

As Figuras 5.42, 5.43 e 5.44 ilustram um comparativo do uso do método do complemento de Schur em *clusters* multiprocessados. Nos testes foram adotados procedimentos análogos aos usados nos testes dos métodos de Schwarz nesse tipo de arquitetura. Para a execução desses testes utilizou-se a versão do método do complemento de Schur com uma aproximação polinomial $k = 1$, devido a sua maior qualidade numérica.

Observa-se que o tempo de execução da implementação com múltiplas *threads* apresenta um melhor desempenho em relação à implementação que utiliza apenas processos. Nessa o número de subdomínios gerados pelo particionamento é menor. Com a diminuição no número de subdomínios ocorre uma redução no tamanho das fronteiras artificiais, e conseqüentemente, dos sistemas de interface. A diminuição no tamanho dos sistemas de interface ocasiona uma redução no custo computacional para a resolução dos mesmos.

Observa-se ainda que para um único nodo existe uma diferença muito significativa entre as abordagens, isto porque, com múltiplos processos o domínio computacional é particionado em 2 subdomínios e são necessárias 3 iterações para o cálculo da solução dos sistemas de interface. Já com múltiplas *threads* o domínio computacional não é particionado e a solução deste é calculada uma única vez por duas *threads*.

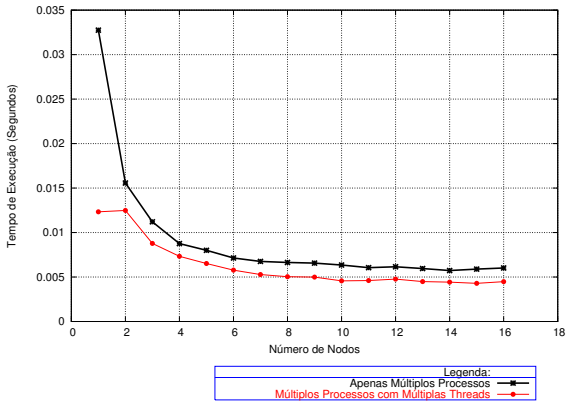

Figura 5.42: Guaíba200 - Processos Vs. *threads* no método do complemento de Schur

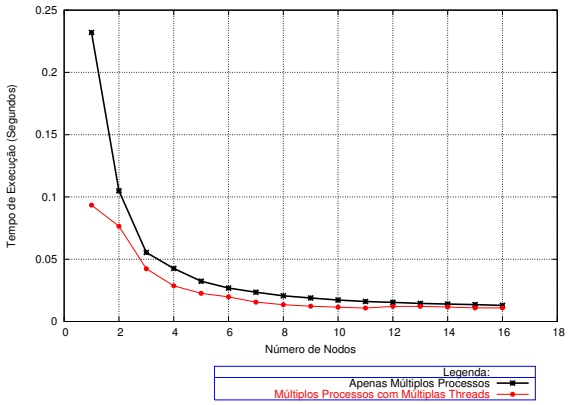

Figura 5.43: Guaíba 100 - Processos Vs. *threads* no método do complemento de Schur

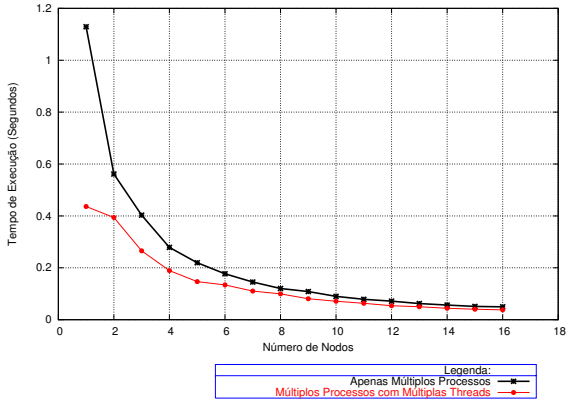

Figura 5.44: Guaíba50 - Processos Vs. *threads* no método do complemento de Schur

5.3 Método Aditivo de Schwarz Vs. Método do Complemento de Schur

Nas Figuras 5.45, 5.46 e 5.47 é feito um comparativo dos tempos de execução do método aditivo de Schwarz e do método do complemento de Schur com aproximação polinomial $k = 1$. Para essa comparação utilizou-se a implementação que faz uso de múltiplos processos e múltiplas *threads*. Para os testes foram usados até 20 nodos do *cluster*, sendo que em cada nodo foi disparado 1 processo com 2 *threads*.

Observa-se nos gráficos que, de maneira geral, os métodos aditivo de Schwarz e método do complemento de Schur apresentaram um comportamento similar. É importante destacar que o método do complemento de Schur resultou em soluções com menor qualidade numérica. Enquanto o método do complemento de Schur com aproximações $k = 1$ apresentou um erro relativo máximo da ordem de $3,37\%$ o método aditivo de Schwarz apresentou um erro relativo máximo da ordem de $0,6\%$. A menor qualidade da solução obtida com o método do complemento de Schur é conseqüência, principalmente, de erros introduzidos pelo uso de aproximações no cálculo das inversas locais.

Apesar da menor qualidade numérica o método do complemento de Schur mostrou-se uma alternativa para aplicações onde é inviável o uso de métodos de Schwarz ou métodos

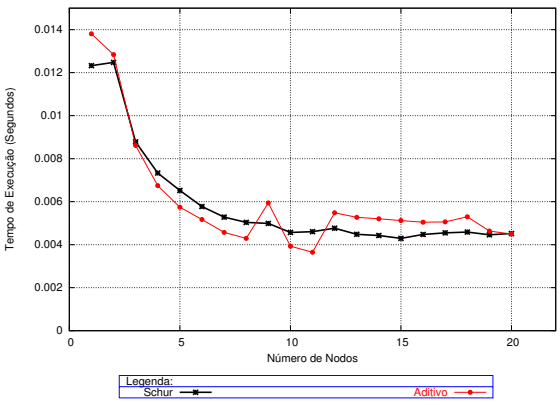

Figura 5.45: Guaíba200 - Método aditivo de Schwarz Vs. método do complemento de Schur

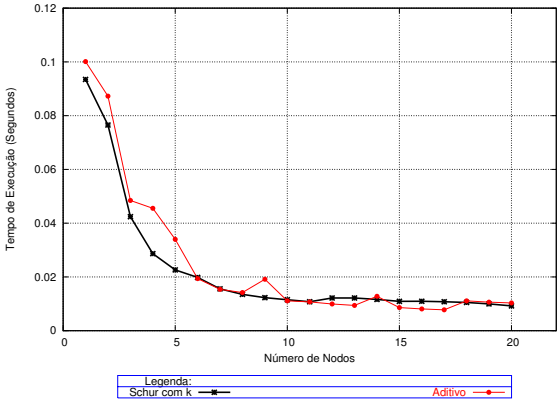

Figura 5.46: Guaíba100 - Método aditivo de Schwarz Vs. método do complemento de Schur

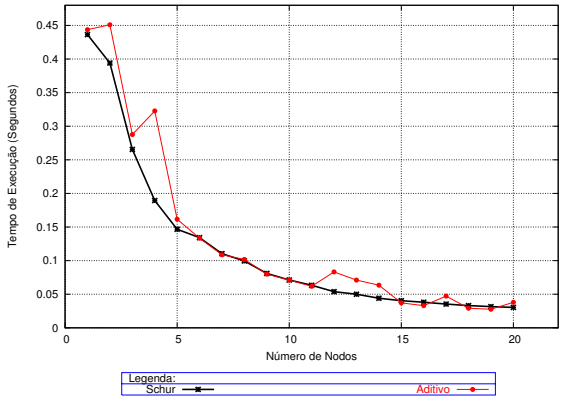

Figura 5.47: Guaíba50 - Método aditivo de Schwarz Vs. método do complemento de Schur

numéricos paralelizados, tais como, em aplicações onde não ocorre o emparelhamento das submalhas entre os subdomínios, ou, ainda, em aplicações onde os subdomínios possuem diferentes modelos matemáticos.

5.4 Métodos de Decomposição de Domínio Vs. Decomposição de Dados

Nessa seção é feito um estudo comparativo entre as abordagens de decomposição de domínio e decomposição de dados. Mais especificamente são comparados os tempos de execuções do método aditivo de Schwarz, desenvolvido neste trabalho, e da implementação do método do GC paralelo desenvolvida em (PICININ, 2003).

Ambas as implementações estão incorporadas ao modelo HIDRA, e utilizam-se de múltiplos processos e múltiplas *threads* para a exploração do paralelismo em *clusters* multiprocessados. Optou-se por utilizar o método aditivo de Schwarz nessa comparação devido ao desempenho computacional e a qualidade numéricas da soluções obtidas com esse método.

As Figuras 5.48, 5.49 e 5.50 mostram um comparativo dos tempos de execução dessas

implementações. É possível verificar nessas figuras que para poucos subdomínios o método do GC paralelo possui um melhor desempenho. À medida em que o número de subdomínios aumenta, e conseqüentemente a granularidade do problema diminui, o método aditivo de Schwarz passa a apresentar melhores desempenhos. Isso porque o método aditivo de Schwarz utiliza-se, em grande parte, de dados locais, necessitando de pouca comunicação. Já o método do GC exige um elevado número de operações de produto escalar a cada iteração e essa operação, quando executada em paralelo provoca sincronização de todos processos. Assim, devido à necessidade de um menor volume de comunicação pode-se concluir que o método aditivo de Schwarz apresenta uma maior escalabilidade.

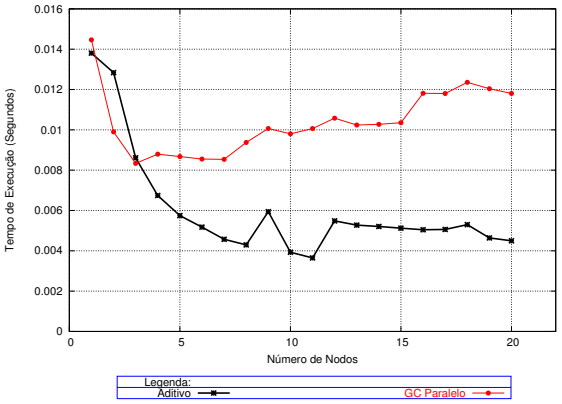

Figura 5.48: Guaíba200 - Método aditivo de Schwarz Vs. GC paralelo

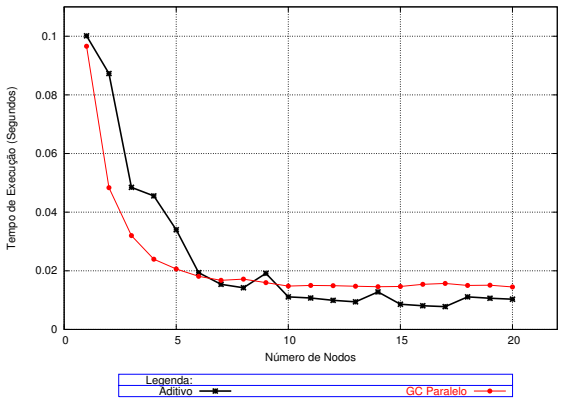

Figura 5.49: Guaíba100 - Método aditivo de Schwarz Vs. GC paralelo

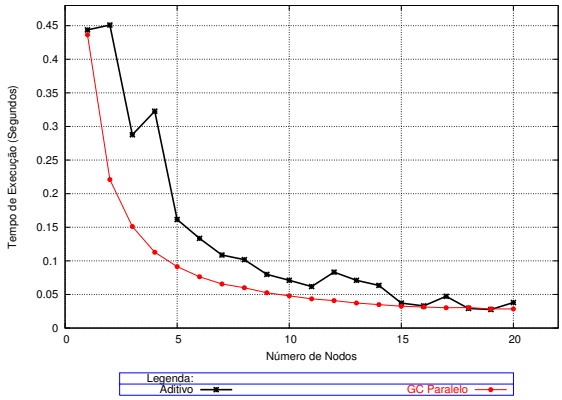

Figura 5.50: Guaíba50 - Método aditivo de Schwarz Vs. GC paralelo

As Figuras 5.51 e 5.52 mostram o comportamento do método aditivo de Schwarz e do GC paralelo, respectivamente, quando executados para o domínio Guaiba100 com 7 subdomínios. Através de uma análise dessas figuras é possível identificar que o GC paralelo apresenta um pior desempenho em relação ao método aditivo de Schwarz, com o aumento no número de processos (subdomínios), devido à necessidade de um maior volume de comunicações. As operações de *reduce* (cor verde) indicam, em ambos os métodos, uma operação de produto escalar e ocasionam uma sincronização de todos os processos da aplicação e, conseqüentemente, uma queda no desempenho da aplicação.

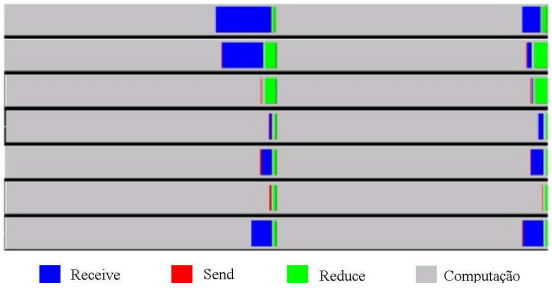

Figura 5.51: Execução do método aditivo de Schwarz

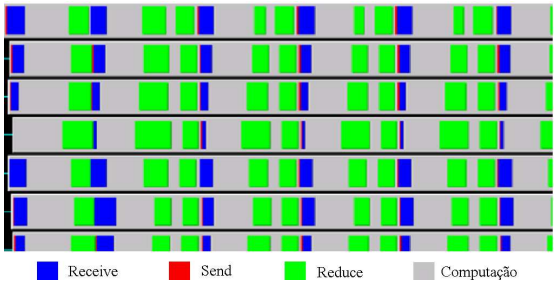

Figura 5.52: Execução do GC em paralelo

5.5 Contenção de Memória

Durante o desenvolvimento deste trabalho foram verificadas perdas significativas de desempenho devido à contenção no acesso à memória em máquinas multiprocessadas. Nessa seção são apresentados alguns resultados obtidos. Embora essa análise não fizesse parte dos objetivos iniciais deste trabalho, essas informações são importantes e poderão ser utilizadas no desenvolvimento de outros trabalhos que poderão discutir esses aspectos em maiores detalhes. Esses resultados mostram que, em determinadas situações, *clusters* formados por máquinas monoprocessados podem ter desempenho superior a *clusters* formado por máquinas multiprocessadas.

Na realização dos testes de contenção de memória utilizou-se as implementações do método aditivo de Schwarz e do método do complemento de Schur com aproximação polinomial $k = 1$. Primeiramente, foram executados processos comunicantes em nodos diferentes, isto é, utilizando apenas um processador de cada nodo. Após, foram executados, o mesmo número de processos comunicantes, mas utilizando os dois processadores do nodo.

Para uma análise mais detalhada sobre a contenção de memória em máquinas multiprocessadas utilizou-se a biblioteca PAPI (*Performance Application Programming Interface*). Essa é uma API desenvolvida pela *University of Tennessee* que possibilita o acesso aos contadores de hardware disponíveis na maioria dos processadores modernos. Esses contadores fornecem informações que podem ser utilizadas por desenvolvedores para uma análise detalhada de desempenho e otimização de programas. Para uma lista completa dos eventos que podem ser monitorados utilizando a biblioteca PAPI veja (PAPI USER'S GUIDE, 2003). Mais especificamente utilizou-se o evento PAPI_RES_STL. Esse evento indica o número de ciclos em que o processador fica ocioso a espera de um recurso.

As Figuras 5.53 e 5.54 mostram, respectivamente, as diferenças de tempo obtidas nos métodos aditivo Schwarz e complemento de Schur para o domínio Guaiba200. Observa-se que ambos os métodos apresentaram um comportamento muito similar. Verificou-se uma perda de desempenho até 4 processos, número após o qual praticamente não ocorrem perdas. Para 4 processos cada subdomínio possui aproximadamente 3100 células.

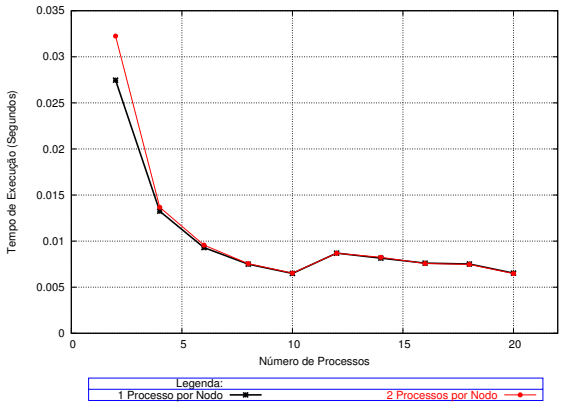

Figura 5.53: Guaíba200 - Contenção: método aditivo de Schwarz

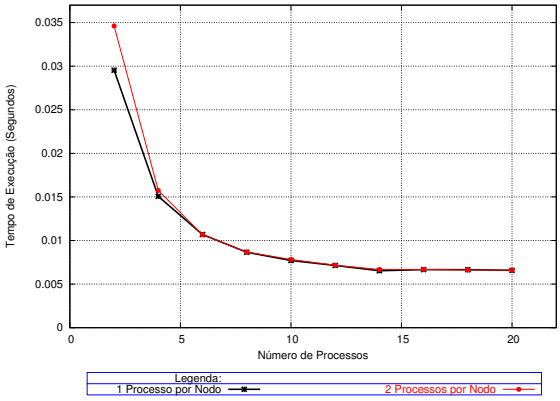

Figura 5.54: Guaíba200 - Contenção: método do complemento de Schur

Nas Figuras 5.55 e 5.56 tem-se o número de ciclos em que o processador fica ocioso para o domínio Guaiba200. Observa-se que os gráficos dessas figuras apresentam um comportamento muito similar aos gráficos das Figuras 5.53 e 5.54, respectivamente. Esse apresenta um aumento na diferença de ciclos ociosos para um pequeno número de subdomínios, após o qual essa diferença praticamente inexiste. Desta forma, conclui-se que o aumento no tempo de execução é decorrente no aumento de ciclos de espera, e conseqüentemente, do aumento no número total de ciclos de execução. Esse aumento no ciclo de espera é decorrente, provavelmente, da competição dos processadores por recursos compartilhados.

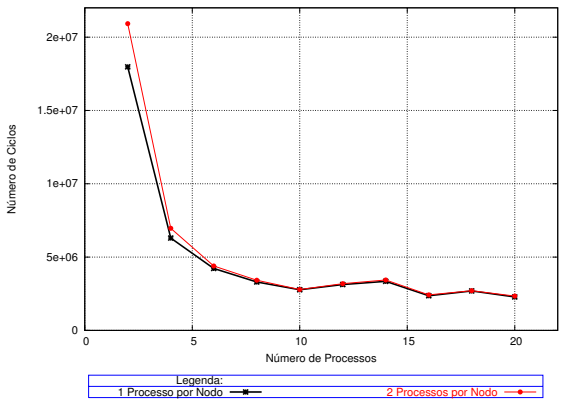

Figura 5.55: Guaíba200 - Ciclos ociosos: método aditivo de Schwarz

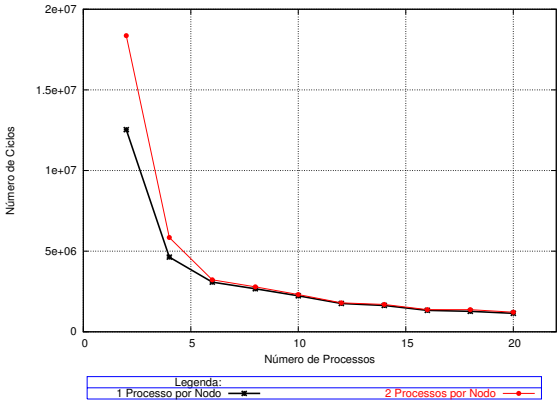

Figura 5.56: Guaíba200 - Ciclos ociosos: método do complemento de Schur

Já nas Figuras 5.57 e 5.58 tem-se as diferenças de tempo obtidas para o domínio Guaiba100. 0bserva-se que os métodos apresentam uma perda de desempenho até 12 processos, após o qual a perda é praticamente nula. Com 12 processos cada subdomínio possui aproximadamente 4200 células.

Através de uma comparação dos resultados obtidos com os domínios Guaiba200 e Guaiba100, pode-se observar que os pontos de término de contenção foram próximos em número de células, mas bem distantes se comparado o número de subdomínios (processos). Para o domínio Guaiba200 a contenção praticamente inexiste a partir de 4 subdomínios, onde cada subdomínio apresenta aproximadamente 3100 células. Já para o domínio Guaiba100 a contenção torna-se praticamente nula a partir de 12 subdomínios

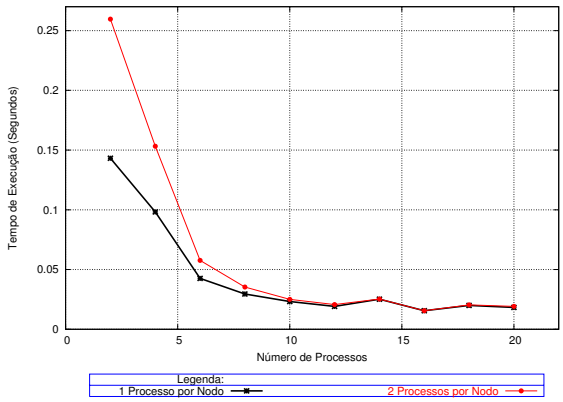

Figura 5.57: Guaíba100 - Contenção: método aditivo de Schwarz

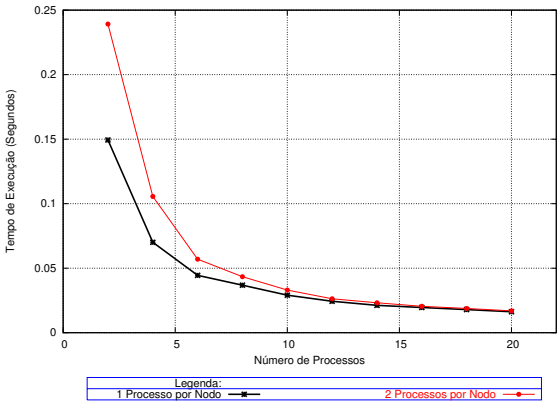

Figura 5.58: Guaíba100 - Contenção: método do complemento de Schur

(processos), onde cada subdomínio apresenta aproximadamente 4200 células. Devido a forma como é feito o particionamento não foi possível realizar testes com uma diferença menor no número de células.

Observa-se ainda que para o Guaiba200 ocorre contenção em particionamentos onde o número de subdomínios é inferior a 4. Nesse caso o número de células dos subdomínios é superior a 5900 células. No Guaiba100, o número mais próximo de 5900 células foi obtido com 8 subdomínios, onde o numero de células é próximo a 6200, e nesse caso, também observou-se a existência de contenção.

As Figuras 5.59 e 5.60 ilustram a diferença no número de ciclos ociosos para o domínio do Guaíba100. Nessas figuras observa-se, também, um comportamento muito semelhante aos gráficos de tempo de execução. Da mesma forma, verifica-se uma contenção até 12 processos, após o qual essa contenção deixa de existir.

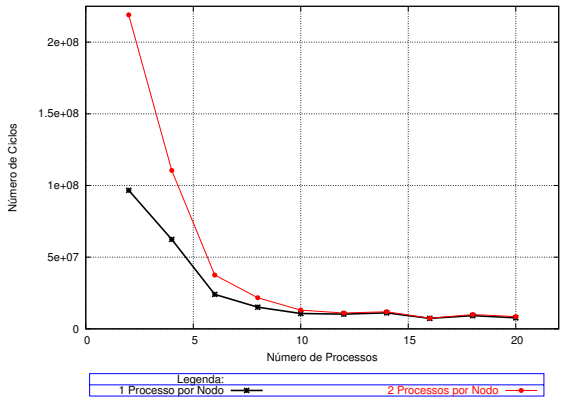

Figura 5.59: Guaíba100 - Ciclos ociosos: método aditivo de Schwarz

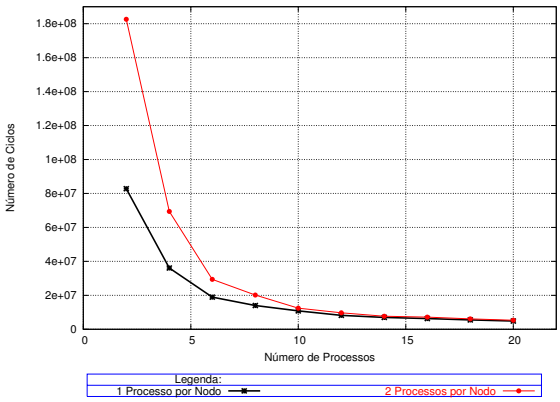

Figura 5.60: Guaíba100 - Ciclos ociosos: método do complemento de Schur

Por fim, tem-se nas Figuras 5.61 e 5.62 as diferenças de tempo obtidas para o Guaiba50. É possível verificar nessas figuras que a contenção diminui com o aumento no número de processos e, conseqüentemente, com a diminuição dos subdomínios, mas em nenhum momento essa deixa de existir. Com 20 processos cada subdomínio possui aproximadamente 9300 células. Acredita-se que esta contenção tenda a diminuir ainda mais com o aumento no número de processos (subdomínios) e, conseqüentemente, diminuição no número de células dos subdomínios, até tornar-se praticamente nula. Estes testes não foram feitos

devido à indisponibilidade de um número maior de nodos no *cluster labtec*.

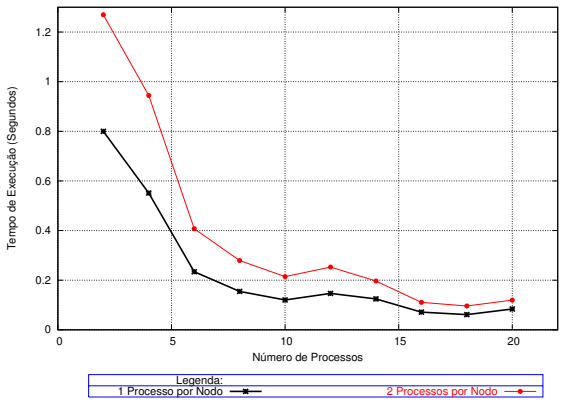

Figura 5.61: Guaíba50 - Contenção: método aditivo de Schwarz

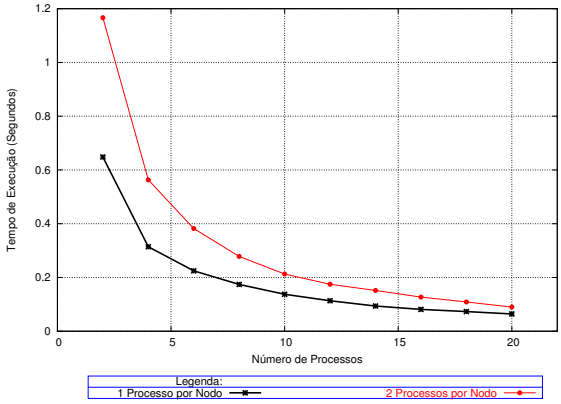

Figura 5.62: Guaíba50 - Contenção: método do complemento de Schur

Já as Figuras 5.63 e 5.64 ilustram a diferença de ciclos ociosos para o domínio Guaíba50. De forma análoga aos domínios Guaíba200 e Guaíba100 os gráficos de tempo de execução e ciclos ociosos apresentam um comportamento praticamente idêntico.

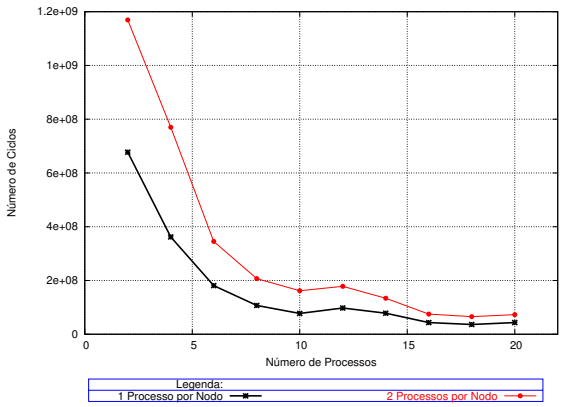

Figura 5.63: Guaíba50 - Ciclos ociosos: método aditivo de Schwarz

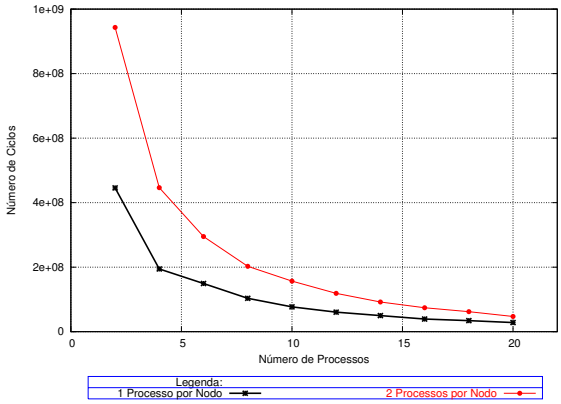

Figura 5.64: Guaíba50 - Ciclos ociosos: método do complemento de Schur

A partir da análise dos gráficos conclui-se que a diferença nos tempos de execução em máquinas monoprocessadas e multiprocessadas é conseqüência do aumento no número de ciclos em espera, aumento esse decorrente da competição entre os processadores por recursos de *hardware* compartilhados, tais como barramento e memória principal. Com a disputa entre os processadores a latência média de acesso à memória tende a ser maior. Desta forma, o processador fica ocioso por um número maior de ciclos.

Embora o uso de memórias *caches* reduza esse efeito, elas não o eliminam totalmente. Como pode ser observado nos gráficos apresentados nessa seção, com o aumento nos tamanhos dos subdomínios, e conseqüentemente na quantidade de dados, a perda de desempenho é significativa. Isto porque, enquanto a quantidade de dados for pequena, todas as operações são realizadas utilizando os dados armazenados em *cache*, eliminando a necessidade de acesso à memória principal. Com o aumento na quantidade de dados torna-se necessário o acesso dos dados na memória principal, tornando a disputa pelo barramento e o acesso à memória principal uma limitante de performance nas máquinas multiprocessadas.

5.6 Considerações Finais

Nesse capítulo foi realizada uma avaliação das paralelizações desenvolvidas e apresentados os principais resultados obtidos. Além disso, foi apresentado o nível de contenção de memória ocorrido em máquinas multiprocessadas.

A partir dos testes observou-se um baixo desempenho do método multiplicativo de Schwarz. Além disso, o método não apresentou uma redução significativa no número de iterações em relação ao método aditivo de Schwarz.

Já o método aditivo de Schwarz apresentou resultados melhores. Esse método mostrou-se ser uma boa alternativa para a resolução de sistemas de equações em paralelo, apresentando, inclusive, uma escalabilidade maior que métodos paralelizados via decomposição de dados.

Observou-se ainda, que o método do complemento de Schur apresentou um bom desempenho computacional mas uma baixa qualidade numérica. Apesar disso, esse mostrou-se uma alternativa para aplicações onde é inviável o uso de métodos de Schwarz ou métodos numéricos paralelizados, tais como, em aplicações onde não ocorre o emparelhamento das submalhas dos subdomínios ou em aplicações onde os subdomínios possuem diferentes modelos matemáticos.

Em relação ao uso de múltiplas *threads*, os testes mostraram que essa abordagem em *clusters* de PCs multiprocessados, em geral, tem um melhor desempenho se comparado com a abordagem que faz o uso apenas de múltiplos processos. Isso é devido, principalmente, a vantagens decorrentes de diferenças no particionamento do domínio computacional.

Por fim, os resultados obtidos nesse trabalho mostram que em determinadas situações o uso de *clusters* com máquinas monoprocessadas é mais vantajoso que o uso de *clusters* multiprocessados.

6 CONCLUSÕES

O principal objetivo deste trabalho foi o desenvolvimento e a avaliação de desempenho das soluções paralelas resultantes de três métodos de decomposição de domínio: dois MDDs com sobreposição e um MDD sem sobreposição. Dentre os métodos com sobreposição foram estudados os métodos aditivo de Schwarz e multiplicativo de Schwarz. Já, dentre os métodos sem sobreposição optou-se pelo método do complemento de Schur. As implementações desenvolvidas foram incorporadas ao modelo HIDRA, que é o modelo 2D e 3D desenvolvido no GMCPAD para a simulação da hidrodinâmica e para o transporte de massa em corpos de água.

Como mencionado o método multiplicativo é inerentemente seqüencial sendo necessário o uso de alguma técnica de coloração na sua paralelização. Nesse trabalho foram testadas três diferentes heurísticas, e verificou-se que em todas elas o método multiplicativo de Schwarz apresentou um baixo desempenho. A redução no número de iterações em relação ao método aditivo de Schwarz acabou não compensando a seqüencialidade do mesmo.

Já o método aditivo de Schwarz apresentou resultados satisfatórios. Esse método mostrou-se altamente paralelizável e com uma boa escalabilidade. Testes realizados mostraram inclusive que esse método apresenta um melhor desempenho que a abordagem de decomposição de dados inicialmente empregada no modelo HIDRA.

A paralelização do método do complemento de Schur apresentou um bom desempenho computacional. Em contrapartida, as soluções obtidas com esse método apresentaram uma baixa qualidade numérica se comparado as soluções obtidas utilizando um método de Schwarz. Isso é devido, principalmente, a erros numéricos introduzidos pelo uso de aproximações polinomiais no cálculo das inversas locais. Apesar da menor qualidade numérica o método do complemento de Schur apresentou-se uma alternativa viável para aplicações onde é inviável o uso de métodos de Schwarz ou métodos numéricos paralelizados.

No que se refere à implementação de métodos de decomposição de domínios em *clusters* de PCs multiprocessados verificou-se que a estratégia que faz uso de múltiplos processos com múltiplas *threads* apresentou um desempenho superior à implementação que faz uso apenas de múltiplos processos. Isto é devido, principalmente, a diferenças no particionamento do domínio durante a solução paralela.

Na abordagem que faz uso de múltiplas *threads* o particionamento do domínio é feito de acordo com o número de nodos disponíveis e a exploração do paralelismo intra-nodal é feita através do uso de múltiplas *threads*. Já na abordagem que faz uso de apenas múltiplos processos o particionamento é feito de acordo com o número de processadores. Desta forma, na primeira abordagem tem-se um número menor de subdomínios.

No caso dos métodos de Schwarz uma redução no número de subdomínios pode

resultar em uma melhor convergência do método. Além disso, em um método de Schwarz, um menor número de subdomínios ocasiona uma redução no número de células pertencentes às áreas de sobreposição e, conseqüentemente, provoca uma menor redundância de cálculo. Particularmente, no caso do método multiplicativo de Schwarz, a diminuição no número de subdomínios, geralmente, resulta em uma diminuição no número de cores utilizadas ocasionando um melhor desempenho do método.

Já em um método do complemento de Schur uma redução no número de subdomínios provoca uma diminuição das fronteiras artificiais e, conseqüentemente, no tamanho dos sistemas de interface. A diminuição no tamanho dos sistemas de interface ocasiona uma redução no custo computacional para a resolução dos mesmos.

Por fim, durante o desenvolvimento desse trabalho verificou-se uma grande perda de desempenho em *clusters* multiprocessados na exploração do paralelismo intra-nodal. Isto é devido à contenção no acesso à memória compartilhada dos nodos. Essa contenção aumenta a latência média de acesso à memória principal provocando um aumento no tempo de execução. Embora o uso de memórias *caches* reduza esse efeito elas não o eliminam totalmente, principalmente em aplicações de grande porte e com grande quantidade de dados.

A partir dos resultados obtidos conclui-se que, em determinadas situações, *clusters* formados por PCs multiprocessados podem apresentar um desempenho inferior a *clusters* formados por PCs monoprocessados.

6.1 Contribuições

A principal contribuição deste trabalho foi o desenvolvimento de *solvers* paralelos para a resolução dos sistemas de equações. Todas as implementações desenvolvidas foram incorporadas ao modelo HIDRA de modo a oferecer uma maior flexibilidade e eficiência ao mesmo. Algumas contribuições desse trabalho são:

- O desenvolvimento e implementação de diferentes *solvers* baseados na abordagem de decomposição de domínio para a resolução de sistemas de equações em paralelo;
- O estudo e a análise de diferentes heurísticas de coloração na paralelização do método multiplicativo de Schwarz;
- O estudo e a análise de diferentes aproximações para o cálculo das inversas no método do complemento de Schur;
- O estudo e uma análise comparativa do uso de apenas múltiplos processos ou o uso de múltiplos processos em conjunto com múltiplas *threads* na paralelização de métodos de decomposição e domínio em *clusters* de PCs multiprocessados;
- O estudo sobre influências da contenção de memória em *clusters* de PCs multiprocessados;
- A integração das implementações desenvolvidas no modelo HIDRA.

Os estudos realizados durante o desenvolvimento desse trabalho resultaram na publicação de doze (12) trabalhos, em autoria e co-autoria, que foram publicados em eventos internacionais, nacionais e regionais. As referências desses trabalhos são apresentados na bibliografia.

Mais especificamente foram publicados 2 artigos em eventos internacionais, os quais foram: (PICININ et al., 2002a) e (PICININ et al., 2003), publicados, respectivamente, nos anais dos eventos PDPTA2002 (*The 2002 International Conference on Parallel and Distributed Processing Techniques and Applications*), realizado em Las Vegas (USA), e ParCo2003 (*Parallel Computing 2003*) realizado em Dresden (Alemenha).

Além disso, foram publicados 2 artigos no evento WSCAD'2002 (Terceiro Workshop em Sistemas Computacionais de Alto Desempenho) realizado em Vitória, Espírito Santo, os quais foram: (PICININ et al., 2002b) e (GALANTE et al., 2002).

Também foram publicados 4 artigos em periódicos regionais os quais foram: Revista Scientia Vol. 13 (MARTINOTTO et al., 2002), Revista Perspectiva Vol. 28 (MARTINOTTO et al., 2003a) e Cadernos de Informática (UFRGS) Vol. 3 (MARTINOTTO et al., 2003b), (PICININ et al., 2003).

Foi submetido, ainda, um trabalho para o 1º Fórum de Pós-Graduação do RS. Esse será apresentado durante o ERAD 2004 (Escola Regional de Alto Desempenho 2004) a ser realizada em Pelotas, Rio Grande do Sul (MARTINOTTO et al., 2004).

Este trabalho ofereceu, também, algumas contribuições ao GMCPAD, servindo como motivação para o desenvolvimento de trabalhos de iniciação científica, entre os quais (GALANTE et al., 2004), (TRARBACH et al., 2004) e (ZEMBRZUSKI et al., 2004) submetidos e aceitos para publicação no Salão de Iniciação Científica do ERAD 2004.

E por fim, foram submetidos trabalhos para o evento VECPAR'2004 (Valência, Espanha) e para as revistas ACTA SCIENTIARUM e Revista Tecnológica. Além disso, esta sendo prerado um artigo a ser submetido no período de 2004 para o evento Euro-Par 2004 (Pisa, Itália).

6.2 Trabalhos Futuros

O desenvolvimento de aplicações para *clusters* e a paralelização de métodos numéricos são áreas de pesquisa relativamente recentes para o GMCPAD e requerem uma série de trabalhos. Especificamente no caso deste trabalho, alguns pontos não puderam ser profundamente avaliados durante o seu desenvolvimento. Dentro deste escopo, abaixo são listadas algumas sugestões para trabalhos futuros, bem como, atividades que possam vir a complementar os estudos realizados durante o desenvolvimento dessa dissertação:

- Um comparativo das implementações desenvolvidas em sistemas de equações gerados pela discretização de outros domínios computacionais, como por exemplo, os disponíveis em pacotes como: *Harwell-Boeing: Sparse Matrix Collection* e *SPARSKIT Collection*. Informações sobre esses e outros pacotes de matrizes podem ser encontradas no repositório *Matrix Market* (MATRIX MARKET, 2003);

- Um comparativo de desempenho das implementações desenvolvidas com implementações disponíveis em bibliotecas para a resolução de sistemas em paralelo, como por exemplo, PETSC (SMITH et al., 2003), Diffpack (DIFFPACK KERNEL AND TOOLBOXES DOCUMENTATION 4.0.00, 2004) e AHPIK (CHARÃO, 2001);

- Um estudo sobre o uso de fatoração incompleta no cálculo das aproximações das inversas locais no método do complemento de Schur. Essas resultam em aproximações de melhor qualidade numérica quando comparadas a aproximações polinomiais (BENZI; TUMA, 1999), mas podem apresentar maiores dificuldades para se obter soluções paralelas computacionalmente eficientes;

- Um estudo aprofundado sobre problemas de contenção de memória em máquinas multiprocessadas, bem como, possíveis maneiras de diminuir a perda de desempenho decorrente dessa contenção em *clusters* de PCs multiprocessados (TANAKA et al., 1998);

- O uso de métodos de decomposição de domínio como pré-condicionadores para métodos iterativos paralelizados via decomposição de dados (SILVA et al., 1997);

- O uso de métodos *multigrid* na solução paralela de EDPs da hidrodinâmica do modelo HIDRA (WESSELING, 1992).

REFERÊNCIAS

AMES, W. F. **Numerical Methods for Partial Differential Equations**. New York: Academic Press, 1977. 365p.

ARAUJO, D. R. **Considerações Numéricas Relativas à Solução de Escoamentos Incompressíveis Externos baseados no Método de Runge-Kutta**. 2002. Mestrado em Matemática — Instituto de Matemática, UFRGS, Porto Alegre - RS.

BAR, M. **Linux Internals**. New York: McGraw-Hill, 2000. 351p.

BARCELLOS, M. P.; GASPARY, L. P. Tecnologias de Rede para Processamento de Alto Desempenho. In: ESCOLA REGIONAL DE ALTO DESEMPENHO, ERAD, 3., 2003, Santa Maria. **Anais...** Santa Maria: UFSM, 2003.

BARRETO, M. **DECK**: Um Ambiente para Programação Paralela em Agregados de Multiprocessadores. 2000. Dissertação (Mestrado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre.

BARRETO, M. E. **Estudo sobre Computação baseada em Cluters e Grids**. 2002. Exame de Qualificação (Doutorado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre.

BARRETT, R.; BERRY, M.; CHAN, T. F.; DEMMEL, J.; DONATO, J. M.; DONGARRA, J.; EIJKHOUT, V.; POZO, R.; ROMINE, C.; VORST, H. V. der. **Templates for the Solution of Linear Systems**: Building Blocks for Iterative Methods. [S.l.]: Society for Industrial and Applied Mathematics, 1994.

BEN-ABDALLAH, A.; CHARÃO, A. S.; PLATEAU, B. Ahpik: A Parallel Multithreaded Framework Using Adaptivity and Domain Decomposition Methods for Solving PDE Problems. In: INTERNATIONAL CONFERENCE ON DOMAIN DECOMPOSITION METHODS, 13., 2001, Lyon. **Proceedings...** [S.l.: s.n.], 2001. p.295–301.

BENZI, M.; TUMA, M. A Comparative Study of Sparse Approximate Inverse Preconditioners. **Applied Numerical Mathematics: Transactions of IMACS**, [S.l.], v.30, n.2, p.305–340, 1999.

BJORSTAD, B.; SKOGEN, M. D. Domain Decomposition Algorithms of Schwarz Type, Designed for Massively Parallel Computers. In: INTERNATIONAL SYMPOSIUM ON DOMAIN DECOMPOSITION METHODS FOR PARTIAL DIFFERENTIAL EQUATIONS, 5., 1992, Philadelphia. **Proceedings...** [S.l.: s.n.], 1992. p.362–375.

BJORSTAD, B.; SKOGEN, M. D. Applications of Dual Schur Complement Preconditioning to Problems in Computational Fluid Dynamics and Computational Electro-Magnetics. In: INTERNATIONAL SYMPOSIUM ON DOMAIN DECOMPOSITION METHODS FOR PARTIAL DIFFERENTIAL EQUATIONS, 9., 1998, Bergen. **Proceedings...** [S.l.: s.n.], 1998. p.708–718.

BODEN, N. J.; COHEN, D.; FELDERMAN, R. E.; KULAWIK, A. E.; SEITS, C. L.; SEIZOVIC, J. N.; SU, W. K. Myrinet: a Gigabit-per-Second Local-Area Network. **IEEE Micro**, Los Alamitos, v.15, n.1, p.29–36, 1995.

BRIAT, J.; GINZBURG, I.; PASIN, M. **Athapascan-0**: User Manual Version 2.4.13. [S.l.: s.n.], 1998.

BUYYA, R. **High Performance Cluster Computing**: Architecture and Systems. [S.l.]: Prentice Hall, 1999. v.1.

CAI, X. Multiplicative Schwarz Methods for Parabolic Problems. **SIAM Journal on Scientific Computing**, Philadelphia, v.15, p.587–603, 1994.

CAI, X. Overlapping Domain Decomposition Methods. In: LANGTANGEN, H. P.; TVEITO, A. (Ed.). **Computational Partial Differential Equations Using Diffpack - Advanced Topics**. [S.l.: s.n.], 2002.

CAI, X.; KEYES, D. E.; MARCINKOWSKI, L. Nonlinear Additive Schwarz Preconditioners and Applications in Computational Fluid Dynamics. **International Journal of Numerical Methods in Fluid Mechanics**, [S.l.], v.40, p.1463–1470, 2002.

CAI, X.; KEYES, D. E.; VENKATAKRISHNAN, V. Newton-Krylov-Schwarz: An Implicit Solver for CFD. In: INTERNATIONAL CONFERENCE ON DOMAIN DECOMPOSITION METHODS, 8., 1997, New York. **Proceedings...** [S.l.: s.n.], 1997. p.387–400.

CAI, X.; SAAD, Y. **Overlapping Domain Decomposition Algorithms for General Sparse Matrices**. [S.l.]: University of Minnesota, 1993. (93-027).

CANAL, A. P. **Paralelização de Métodos de Resolução de Sistemas Lineares Esparsos com o DECK em Clusters de PCs**. 2000. Dissertação (Mestrado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre.

CARDOSO, C. **Análise de Fluxos Unidimensionais via Método de Runge-Kutta e Noções da Teoria de Shadowing**. 2001. Mestrado em Matemática — Instituto de Matemática, UFRGS, Porto Alegre - RS.

CHAN, A.; ASHTON, D.; LUSK, R.; GROPP, W. **Jumpshot-4's User's Guide**. Disponível em: <http://www-unix.mcs.anl.gov/perfvis/software/viewers/>. Acesso em: out. 2003.

CHAN, A.; GROPP, W.; LUSK, E. **User's Guide for MPE**: extensions for MPI programs. Disponível em: <http://www-unix.mcs.anl.gov/mpi/mpich/>. Acesso em: out. 2003.

CHAN, T. F.; MATHEW, T. Domain Decomposition Algorithms. **Acta Numerica**, Cambridge, v.3, p.61–143, 1994.

CHARÃO, A. S. **Multiprogrammation Parallèle Générique des Méthodes de Décomposition de Domaine**. 2001. Tese (Doutorado em Ciência da Computação) — Institut National Polytechnique de Grenoble.

CHARÃO, A. S.; CHARPENTIER, I.; STEIN, B. Generic Parallel Multithreaded Programming of Domain Decomposition Methods on PC Clusters. In: INTERNATIONAL CONFERENCE ON DOMAIN DECOMPOSITION METHODS, 14., 2003, Cocoyoc. **Proceedings...** [S.l.: s.n.], 2003. p.365–372.

CHOW, E. T. **Robust Preconditioning for Sparse Linear Systems**. 1997. Tese (Doutorado em Ciência da Computação) — Department of Computer Science, University of Minnesota, Minneapolis - MN.

CONTE, N. F. **Implicações Geométricas e Topológicas da Planaridade em Grafos**. 2003. Dissertação (Mestrado em Matemática) — Instituto de Informática, UFRGS, Porto Alegre.

CUMINATO, J. A.; MENEGUETTE, M. **Discretização de Equações Diferenciais Parciais**: Técnicas de Diferenças Finitas. São Paulo: Universidade de São Paulo, 1999.

CUNHA, M. C. C. **Métodos Numéricos**. Campinas: Ed. da Unicamp, 2000.

CUNHA, R. D. **A Study on Iterative Methods for the Solution of Systems of Linear Equations on Transputer Networks**. 1992. Tese (Doutorado em Ciência da Computação) — University of Kent at Canterbury.

CUNHA, R. D. da; HOPKINS, T. **PIM 2.0 The Parallel Iterative Methods Package for Systems of Linear Equations**. [S.l.: s.n.], 2003.

DE ROSE, C. A. F.; NAVAUX, P. O. A. **Arquiteturas Paralelas**. Porto Alegre: Sagra-Luzzatto, 2003. 152p.

DEBREU, L.; BLAYO, E. On the Schwarz Alternating Method for Solving Oceanic Models on Parallel Computers. **Journal of Computational Physics**, New York, v.141, p.93–111, 1998.

DEMMEL, J.; HEATH, M.; VORST, H. V. der. Parallel Numerical Linear Algebra. **Acta Numerica**, Cambridge, v.2, p.61–143, 1993.

DIFFPACK Kernel and Toolboxes Documentation 4.0.00. Disponível em: <http://www.nobjects.com/diffpack/refmanuals/current/>. Acesso em: fev. 2004.

DORNELES, R. V. **Particionamento de Domínio e Balanceamento de Carga no Modelo HIDRA**. 2003. Tese (Doutorado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre - RS.

DORNELES, R. V.; RIZZI, R. L.; ZEFERINO, C. A.; DIVERIO, T. A.; BAMPI, S.; SUZIN, A. A.; NAVAUX, P. O. A. A PC Cluster Implementation of a Mass Transport Two Dimensional Model. In: SBAC - SYMPOSIUM ON COMPUTER ARCHITECTURE AND HIGH PERFORMANCE COMPUTING, 12., 2000, São Carlos. **Proceedings...** São Carlos: UFSCAR, 2000. v.1, p.191–198.

DREIER, B.; MARKUS, Z.; THEO, U. Parallel and Distributed Programming with Pthreads and Rthreads. In: INTERNATIONAL CONFERENCE ON MASSIVELY PARALLEL COMPUTING SYSTEM, 1998. **Proceedings...** [S.l.: s.n.], 1998. v.1, p.34–40.

DRYJA, M.; WIDLUND, O. B. **An Additive Variant of the Alternating Method for the Case of Many Subregions**. New York: New York University, 1987. (339).

EIJKHOUT, V. Overview of Iterative Linear System Solver Packages. **NHSE Review**, [S.l.], v.3, n.1, 1998.

EVANS, L. C. **Partial Differential Equations**. 2nd.ed. Berkeley: American Mathematical Society, 1998. 662p. (Graduate Studies in Mathematics).

FLEMISH, B. **The Alternating Schwarz Method**: Mathematical Foudantion and Parallel Implementation. 2001. Dissertação (Mestrado em Matemática) — Departament of Mathematics, Iowa State University, Ames.

FORTUNA, A. O. **Técnicas Computacionais para Dinâmica dos Fluidos**. [S.l.]: Ed. da USP, 2000.

FOSTER, I. T. **Designing and Building Parallel Programs**. Reading, USA: Addison Wesley, 1995. 381p.

GALANTE, G.; BALBINOT, J. I.; MARTINOTTO, A. L.; RIZZI, R. L.; DIVERIO, T. A. Avaliação do Desempenho de Duas Versões do Algoritmo do Gradiente Conjugado Paralelizado em Cluster de PCs. In: WSCAD - WORKSHOP EM SISTEMAS COMPUTACIONAIS DE ALTO DESEMPENHO, 3., 2002, Vitória - ES. **Anais...** [S.l.: s.n.], 2002. p.162–163.

GALANTE, G.; MARTINOTTO, A. L.; RIZZI, R. L.; DIVERIO, T. A. Solução Paralela de Sistemas de Equações Lineares Através de Métodos de Decomposição de Domínio. In: ESCOLA REGIONAL DE ALTO DESEMPENHO, ERAD, 4., 2004, Pelotas. **Anais...** Pelotas: UFPel, 2004.

GAREY, M. R.; JOHNSON, D. S. **Computer and Intractability**: a Guide to the Theory of NP-completeness. San Francisco: Freeman, 1979.

GEIST, A.; BEGUELIN, A.; DONGARRA, J.; JIANG, W.; MANCHEK, R.; SUNDERAM, V. S. **PVM**: Parallel Virtual Machine. Cambridge: MIT Press, 1994. 299p.

GOLUB, G. H.; LOAN, C. F. V. **Matrix Computations**. Baltimore: The Johns Hopkins University Press, 1996.

GROOSS, J. **A Parallel Elliptic PDE Solver**. 2001. Dissertação (Mestrado em Engenharia) — Informatics and Mathematical Modelling, Technical University of Denmark, Lyngby.

GROPP, W. D.; LUSK, E. **MPICH - A Portable Implementation of MPI**. Disponível em: <http://www-unix.mcs.anl.gov/mpi/mpich/>. Acesso em: out. 2003.

HENDRICKSON, B.; LELAND, R. **The Chaco User´s Guide. Version 2.0**. Disponível em: <http://www.cs.sandia.gov/CRF/papers_chaco.html>. Acesso em: out. 2003.

HIRSCH, C. **Numerical Computation of Internal and External Flows**. Chinchester: John Wiley & Sons, 1992. v.1.

IÓRIO JÚNIOR, R.; IÓRIO, V. M. **Equações Diferenciais Parciais**: uma introdução. Rio de Janeiro: IMPA, 1988.

JÚDICE, J.; PATRÍCIO, J. M. **Sistemas de Equações Lineares**. Coimbra: Universidade de Coimbra, 1996.

JUSTO, D. A. R. **Geração de Malhas, Condições de Contorno e Discretização de Operadores para Dinâmica de Fluidos Computacional**. 1997. Mestrado em Matemática — Instituto de Matemática, UFRGS, Porto Alegre - RS.

KARYPIS, G.; KUMAR, V. **METIS**: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-reducing Orderings of Sparse Matrices. Disponível em: <http://www.cs.umn.edu/∼karypis>. Acesso em: out. 2003.

KARYPIS, G.; KUMAR, V. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. **SIAM Journal on Scientific Computing**, Philadelphia, v.20, n.1, p.359–392, 1998. Disponível em: <http://www.cs.umn.edu/∼karypis>. Acesso em: fev. 2004.

KARYPIS, G.; KUMAR, V. Multilevel Algorithms for Multi-Constraint Graph Partitioning. In: SUPERCOMPUTING, 1998. **Proceedings...** [S.l.: s.n.], 1998. Disponível em: <http://www.cs.umn.edu/∼karypis>. Acesso em: fev. 2004.

KARYPIS, G.; KUMAR, V. Multilevel K-way Partitioning Scheme for Irregular Graphs. **Journal of Parallel and Distributed Computing**, Orlando, v.48, n.1, p.96–129, 1998. Disponível em: <http://www.cs.umn.edu/∼karypis>. Acesso em: fev. 2004.

KLEIMAN, S.; SHAH, D.; SMAALDERS, B. **Programming With Threads**. [S.l.]: Prentice Hall, 1996. 534p.

KUMAR, V.; GRAMA, A.; GUPTA, A.; KARYPIS, G. **Introduction to Parallel Coputing**: Design and Analysis of Algorithms. [S.l.]: Benjamin Cummings Publishing Company, 1993.

KUZNETSOV, S.; LO, G.; SAAD, Y. **Parallel Solution of General Sparse Linear Systems**. [S.l.]: Minnesota Supercomputer Institute, University of Minnesota, 1997. (Technical Report UMSI 97/98).

LEWIS, B.; BERG, D. J. **Multithreaded Programming With Pthreads**. [S.l.]: Sun Microsystems Press, 1998. 382p.

LUI, S. H. On Schwarz Alternating Methods for Nonlinear Elliptic Problems. **Contemporary Mathematics**, [S.l.], v.218, p.447–452, 1998.

LUMSDAINE, A.; SQUYRES, J.; BARRETT, B.; SANKARAN, S.; CHABLANI, M.; SAHAY, V. **LAM / MPI Parallel Computing**. Disponível em: <http://www.lam-mpi.org/>. Acesso em: fev. 2004.

MALISKA, C. R. **Transferência de Calor e Mecânica dos Fluídos Computacional**. Rio de Janeiro: Ed. LTC, 1995.

MARTINOTTO, A. L. **Paralelização de Métodos Numéricos de Resolução de Sistemas Esparsos de Equações Utilizando MPI e Pthreads**. 2001. Trabalho de Conclusão (Curso de Ciência da Computação) — Universidade de Caxias do Sul, Caxias do Sul, RS.

MARTINOTTO, A. L. **Estudo de Pré-condicionadores para Métodos Iterativos do Subespaço de Krylov**. 2002. Trabalho Individual (Mestrado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre.

MARTINOTTO, A. L.; FRIZZO, E. J.; DORNELES, R. V.; DIVERIO, T. A. Paralelização do Método GMRES com MPI e Pthreads. **SCIENTIA**, São Leopoldo, v.13, n.1, p.99–112, 2002.

MARTINOTTO, A. L.; PICININ, D.; RIZZI, R. L.; DORNELES, R. V.; DIVERIO, T. A. Solução Paralela de Equações Diferenciais Parciais no Modelo HIDRA. **Revista Perspectiva**, Erechim, v.27, n.98, p.87–93, 2003.

MARTINOTTO, A. L.; PICININ, D.; RIZZI, R. L.; DORNELES, R. V.; DIVERIO, T. A. Métodos de Decomposição de Domínio na Solução de Equações Diferenciais Parciais. **Cadernos de Informática**, Porto Alegre, v.3, n.1, p.97–102, 2003.

MARTINOTTO, A. L.; PICININ, D.; RIZZI, R. L.; DORNELES, R. V.; DIVERIO, T. A. Paralelização de Métodos Númericos para a Solução de Sistemas de Equações Lineares. In: ESCOLA REGIONAL DE ALTO DESEMPENHO, ERAD, 4., 2004, Pelotas. **Anais...** Pelotas: UFPel, 2004.

MATRIX Market. Disponível em: <http://math.nist.gov/MatrixMarket/>. Acesso em: out. 2003.

MOLDOVAN, D. I. **Parallel Processing from Applications to Systems**. [S.l.]: Morgan Kaufmann Publishers, 1993.

MÜLLER, F. M.; CHARÃO, A. S.; SANTOS, H. G. Aplicações de Alto Desempenho. In: ESCOLA REGIONAL DE ALTO DESEMPENHO, ERAD, 3., 2003, Santa Maria. **Anais...** Santa Maria: UFSM, 2003.

NADEEM, S. A. **Parallel Domain Decomposition Preconditioning for the Adaptive Finite Element Solution of Elliptic Problems in Three Dimensions**. 2001. Tese (Doutorado em Ciência da Computação) — School of Computing, The University of Leeds, Leeds.

OLIVEIRA, R. S.; CARISSIMI, A. S.; TOSCANI, S. S. **Sistemas Operacionais**. 2.ed. Porto Alegre: Sagra-Luzzatto, 2001. 247p.

OMNI OpenMP Compiler Project. Disponível em: <http://phase.hpcc.jp/Omni/>. Acesso em: out. 2003.

OPENMP: Simple, Portable, Scalable SMP Programming. Disponível em: <http://www.openmp.org/>. Acesso em: out. 2003.

PACHECO, P. S. **Parallel Programming with MPI**. San Francisco: Morgan Kaufmann, 1997. 418p.

PAGLIERI, L.; AMBROSI, D.; FORMAGGIA, L.; QUARTERONI, A.; SCHEININE, A. L. Parallel Computation for Shallow Water Flow: a Domain Decomposition Approach. **Parallel Computing**, Amsterdan, v.23, p.1261–1277, 1997.

PALHA, M. A. K. **Avaliação de Desempenho e Perfilamento de Código em Programas Paralelos**. 2000. Trabalho de Conclusão (Curso de Ciência da Computação) — Universidade de Caxias do Sul, Caxias do Sul, RS.

PAPI User's Guide. Disponível em: <http://icl.cs.utk.edu/projects/papi/documents/>. Acesso em: out. 2003.

PELLEGRINI, F. **Scotch 3.1 User's Guide**. Disponível em: <http://www.labri.fr/Perso/∼pelegrin/scotch>. Acesso em: out. de 2003.

PICININ, D. **Paralelização do Algoritmo do Gradiente Conjugado através da Biblioteca MPI e de Threads**. 2001. Trabalho Individual (Mestrado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre.

PICININ, D. **Paralelização de Métodos Numéricos em Clusters Empregando as Bibliotecas MPI, DECK e Pthreads**. 2003. Dissertação (Mestrado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre.

PICININ, D.; MARTINOTTO, A. L.; HOLBIG, C. A.; RIZZI, R. L.; DORNELES, R. V.; DIVERIO, T. A.; NAVAUX, P. O. A. Parallelization of Krylov's Subspace Methods in Multiprocessor PC Clusters. In: PARCO - INTERNATIONAL CONFERENCE ON PARALLEL COMPUTING, 2003, Dresden. **Proceedings...** Dresden:Technical University of Dresden, 2003.

PICININ, D.; MARTINOTTO, A. L.; RIZZI, R. L.; DORNELES, R. V.; DIVERIO, T. A. Paralelizações de Métodos Numéricos em Clusters Empregando as Bibliotecas MPICH, DECK e Pthread. **Cadernos de Informática**, Porto Alegre, v.3, n.1, p.49–54, 2003.

PICININ, D.; MARTINOTTO, A. L.; RIZZI, R. L.; DORNELES, R. V.; DIVERIO, T. A.; NAVAUX, P. O. A. Parallelizing Conjugate Gradiente Method for Clusters Using MPI and Threads. In: INTERNATIONAL CONFERENCE ON PARALLEL AND DISTRIBUTED PROCESSING TECHNIQUES AND APPLICATIONS, PDPTA, 2002, Las Vegas. **Proceedings...** [Las Vegas]: CSREA, 2002.

PICININ, D.; MARTINOTTO, A. L.; RIZZI, R. L.; DORNELES, R. V.; DIVERIO, T. A.; NAVAUX, P. O. A. Ordenação de Mensagens e Pré-condicionamento na Solução Paralela do Gradiente Conjugado em Clusters de PCs Multiprocessados. In: WSCAD - WORKSHOP EM SISTEMAS COMPUTACIONAIS DE ALTO DESEMPENHO, 3., 2002, Vitória - ES. **Anais...** [S.l.: s.n.], 2002. p.95–102.

POLLARD, A. D. **Preconditioned Conjugate Gradient Methods for Serial and Parallel Computers**. 1992. Dissertação (Mestrado em Matemática) — Departament of Mathematics, University of Reading.

POZO, R.; REMINGTON, K. A.; LUMSDAINE, A. **SparseLib++** : Sparse Matrix Class Library Reference Guide. [S.l.: s.n.], 1996.

RIGHI, R. R. **libVIP - Desenvolvimento em Nível de Usuário de uma Biblioteca de Comunicação que Implementa o Protocolo de Interface Virtual**. 2003. Trabalho de Conclusão (Curso de Ciência da Computação) — Universidade Federal de Santa Maria, Santa Maria, RS.

RIGONI, E. H.; DIVERIO, T. A.; NAVAUX, P. O. A. **Introdução a Programação em Clusters de Alto Desempenho**. Porto Alegre: PPGC do Instituto de Informática, UFRGS, 1999. (RP-305).

RIXEN, D.; FARHAT, C. Dual Schur Complement Method for Semi-definite Problems. In: INTERNATIONAL CONFERENCE ON DOMAIN DECOMPOSITION METHODS, 10., 1998, Boulder. **Proceedings...** [S.l.: s.n.], 1998. p.341–348.

RIZZI, R. L. **Técnicas Matemáticas e Computacionais para Modelos Meteorológicos**. 1999. Exame de Qualificação (Doutorado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre.

RIZZI, R. L. **Modelo Computacional Paralelo para a Hidrodinâmica e para o Transporte de Massa Bidimensional e Tridimensional**. 2002. Tese (Doutorado em Ciência da Computação) — Instituto de Informática, UFRGS, Porto Alegre.

ROUX, F.-X. **Domain Decomposition Methods and Distributed Memory Machines**. Disponível em: <http://citeseer.nj.nec.com/159262.html>. Acesso em: fev. 2004.

SAAD, Y. **Iterative Methods for Sparse Linear Systems**. [S.l.]: PWS Publishing Company, 1996.

SAAD, Y.; SOSONKINA, M. Distributed Schur Complement Techniques for General Sparse Linear Systems. **SIAM Journal on Scientific Computing**, Philadelphia, v.21, n.4, p.1337–1356, 1999.

SANTOS, L. L. P.; DORNELES, R. V. **Particionamento de Grafos**. 2001. Trabalho de Conclusão (Curso de Ciência da Computação) — Universidade de Caxias do Sul, Caxias do Sul, RS.

SCHLOEGEL, K.; KARYPIS, G.; KUMAR, V. **Graph Partitioning for High Performance Scientific Simulations**. Disponível em: <http://www-users.cs.umn.edu/ karypis/publications/partitioning.html>. Acesso em: out. 2003.

SHEWCHUK, J. R. **An Introduction to the Conjugate Gradient Method without the Agonizing Pain**. Disponível em: <http://www.cs.cmu.edu/∼jrs/jrspapers.html>. Acesso em: out. 2003.

SILBERSCHATZ, A.; GALVIN, P. B. **Operating System Concepts**. São Paulo: Prentice Hall, 2000. 696p.

SILVA, R.; ALMEIDA, R.; GALEÃO, A.; COUTINHO, A. Iterative Local Solvers for Distributed Krylov-Schwarz Methods Applied to Convection-Diffusion Problems. **Computer Methods for Applied Mechanics and Engineering**, [S.l.], v.149, 1997.

SLAVIERO, V. M. P. **O Método do Gradiente Conjugado com Produto Interno Geral**. 1997. Dissertação (Mestrado em Ciência da Computação) — Instituto de Matemática, UFRGS, Porto Alegre.

SMITH, B.; BALAY, S.; KNEPLEY, M.; ZHANG, H.; BUSCHELMAN, K. **The Portable, Extensible Toolkit for Scientific Computation**. Disponível em: <http://www-unix.mcs.anl.gov/petsc>. Acesso em: out. 2003.

SMITH, B.; BJORSTAD, P.; GROPP, W. **Domain Decomposition**: Parallel Multilevel Methods for Elliptic Partial Differential Equations. Cambridge: Cambridge University Pres, 1996.

SMITH, B. F. **Domain Decomposition Methods for Partial Differential Equations**. Argonne: Mathematics and Computer Science Division, Argonne National Laboratory, 1992.

SNIR, M.; OTTO, S.; HUSS-LEDERMANN, S.; WALKER, D.; DONGARRA, J. **MPI**: The Complete Reference. [S.l.]: MIT Press, 1996.

TANAKA, Y.; MATSUDA, M.; ANDO, M.; KUBOTA, K.; SATO, M. COMPaS: A Pentium Pro Pc-based SMP Cluster and Its Experience. In: IPPS WORKSHOP ON PERSONAL COMPUTER BASED NETWORKS OF WORKSTATIONS, 1998. **Proceedings...** [S.l.: s.n.], 1998. p.486–497.

TANENBAUM, A. S. **Sistemas Operacionais Modernos**. 4.ed. Rio de Janeiro: Prentice Hall do Brasil, 1995.

TANENBAUM, A. S. **Structured Computer Organization**. 4th.ed. [S.l.]: Prentice Hall, 1999.

TOSCANI, S. S.; OLIVEIRA, R. S.; CARISSIMI, A. S. **Sistemas Operacionais e Programação Concorrente**. Porto Alegre: Sagra-Luzzatto, 2003. 247p.

TRARBACH, L. L.; ZEMBRZUSKI, M. C.; MARTINOTTO, A. L.; PICININ, D.; DIVERIO, T. A. Uso de Contadores de Hardware para Análise de Desempenho e Otimização. In: ESCOLA REGIONAL DE ALTO DESEMPENHO, ERAD, 4., 2004, Pelotas. **Anais...** Pelotas: UFPel, 2004.

TUMINARO, R. S.; SHADID, J. N.; HEROUX, M. **A Massively Parallel Iterative Solver Library for Solving Sparse Linear Systems**. Disponível em: <http://www.cs.sandia.gov/CRF/aztec1.html>. Acesso em: out. 2003.

VARGAS, R. S. **Matrix Iterative Analisys**. [S.l.]: Springer, 1999. (Series in Computational Mathematics).

WALSHAW, C. **The Jostle User Manual**: Version 2.2. Disponível em: <http://www.gre.ac.uk/jostle>. Acesso em: out. 2003.

WESSELING, P. **Introduction to Multigrid Methods**. Chichester: John Wiley & Sons, 1992.

WEST, D. B. **Introduction to Graph Theory**. Upper Saddle River: Prentice Hall, 2001.

ZEMBRZUSKI, M. C.; TRARBACH, L. L.; MARTINOTTO, A. L.; PICININ, D.; DIVERIO, T. A. Estudo Comparativo na Exploração de Paralelismo em Ambientes de Memória Compartilhada. In: ESCOLA REGIONAL DE ALTO DESEMPENHO, ERAD, 4., 2004, Pelotas. **Anais...** Pelotas: UFPel, 2004.