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

CARLOS AMARAL HÖLBIG

Ambiente de Alto Desempenho com Alta Exatidão para a Resolução de Problemas

Tese apresentada como requisito parcial
para a obtenção do grau de
Doutor em Ciência da Computação

Prof. Dr. Tiarajú Asmuz Diverio
Orientador

Prof. Dr. Dalcidio Moraes Claudio
Co-orientador

Porto Alegre, outubro de 2005

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Hölbig, Carlos Amaral

Ambiente de Alto Desempenho com Alta Exatidão para a Resolução de Problemas / Carlos Amaral Hölbig. – Porto Alegre: PPGC da UFRGS, 2005.

103 f.: il.

Tese (doutorado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2005. Orientador: Tiarajú Asmuz Diverio; Co-orientador: Dalcidio Moraes Claudio.

1. Alto Desempenho. 2. Alta Exatidão. 3. Biblioteca C–XSC. 4. Cluster de Computadores. 5. Sistemas Lineares. I. Diverio, Tiarajú Asmuz. II. Claudio, Dalcidio Moraes. III. Título.

AGRADECIMENTOS

É muito estranho ter que escrever esta parte da tese, depois de 4 anos de muito trabalho é a última parte da tese que estou escrevendo. Além disso, sempre corremos o risco de esquecer alguém. Bom, mas vou deixar de conversa e vou direto aos agradecimentos.

Primeiramente gostaria de fazer um agradecimento especial aos meus orientadores e amigos Tiarajú e Dalcidio por todos os ensinamentos transmitidos, pela ajuda, amizade e, principalmente, pela confiança depositada em meu trabalho, não somente durante este doutorado mas, principalmente, nestes 12 anos em que trabalhamos juntos.

Um agradecimento muito especial ao Paulo Sérgio e ao Bernardo pelo auxílio na implementação dos *solvers* e pela integração do C-XSC no *cluster* labtec, certamente sem o trabalho deles esta tese não estaria sendo defendida em 4 anos.

Gostaria de agradecer aos meus alunos e ex-alunos do curso de Ciência da Computação da UPF que, direta ou indiretamente, auxiliaram no desenvolvimento desta tese (Thaíze, Cassiano, Leila, Andriele, Alexandre, Gustavo e Maurício). Gostaria de agradecer, também, aos amigos e colegas de grupo de pesquisa ComPaDi da UPF, Marcelo e Brusso. Além disso, não posso deixar de agradecer a UPF e ao curso de Ciência da Computação que confiaram e investiram em mim e que me proporcionaram a ida para a Alemanha pelo período de quase um ano para a realização de meu doutorado *sandwich*.

Outro agradecimento que se faz necessário é aos professores Walter Krämer da Universidade de Wuppertal, Gerd Bohlender e Rudi Klatte da Universidade de Karlsruhe pela troca de experiências e pelo trabalho em conjunto realizado durante o meu doutorado *sandwich* e minhas missões de trabalho na Alemanha nestes últimos anos, realizados através de nossos projetos de cooperação internacional.

Aos todos os meus amigos que de uma maneira ou de outra me apoiaram e ajudaram durante o período de elaboração deste trabalho e, em especial, a: Márcia, Rabello, Cristiano, Mônica, Rafael, Karina, Mariana, Carina e Janaína.

E, por fim, mais do que agradecer gostaria de dedicar esta tese aos meus pais e minha irmã que, mesmo a distância, me deram todo o incentivo e carinho durante o período de realização deste trabalho.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

AE	Alta Exatidão
BLAS	***B**asic **L**inear **A**lgebra **S**ubroutines*
ComPaDi	Grupo de Processamento Paralelo e Distribuído
IEEE	***I**nstitute of **E**lectrical and **E**lectronic **E**ngineers*
II	Instituto de Informática
GC	Gradiente Conjugado
GMCPAD	Grupo de Matemática da Computação e Processamento de Alto Desempenho
MPI	***M**essage **P**assing **I**nterface*
MPMD	***M**ultiple **P**rogram **M**ultiple **D**ata*
NaN	***N**ot-**a**-**N**umber*
OpenMP	***Open M**ulti **P**rocessing*
PE	Produto Escalar
PEO	Produto Escalar Ótimo
PUCRS	Pontifícia Universidade Católica do Rio Grande do Sul
SELA	Sistema de Equação Linear Algébrica
SMP	***S**ymmetric **M**ulti-**P**rocessors*
SPMD	***S**ingle **P**rogram **M**ultiple **D**ata*
SSE	***S**treaming **SIMD E**xtensions*
UFRGS	Universidade Federal do Rio Grande do Sul
ULP	***U**nits in the **L**ast **P**lace*
UPF	Universidade de Passo Fundo
XSC	***Ex**tension **S**cientific **C**omputation*

LISTA DE SÍMBOLOS

Ox	Arredondamento para o número mais próximo de máquina
∇x	Arredondamento para baixo, por corte ou por truncamento
$\triangle\ x$	Arredondamento para cima ou por excesso

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Este trabalho visa a disponibilização de um ambiente de alto desempenho, do tipo *cluster* de computadores, com alta exatidão, obtida através da utilização da biblioteca C–XSC. A alta exatidão na solução de um problema é obtida através da realização de cálculos intermediários sem arredondamentos como se fossem em precisão infinita. Ao final do cálculo, o resultado deve ser representado na máquina. O resultado exato real e o resultado representado diferem apenas por um único arredondamento. Esses cálculos em alta exatidão devem estar disponíveis para algumas operações aritméticas básicas, em especial as que possibilitam a realização de somatório e de produto escalar. Com isso, deseja-se utilizar o alto desempenho através de um ambiente de *cluster* onde se tem vários nodos executando tarefas ou cálculos. A comunicação será realizada por troca de mensagens usando a biblioteca de comunicação MPI. Para se obter a alta exatidão neste tipo de ambiente, extensões ou adaptações nos programas paralelos tiveram que ser disponibilizadas para garantir que a qualidade do resultado final realizado em um *cluster*, onde vários nodos colaboram para o resultado final do cálculo, mantivesse a mesma qualidade do resultado que é obtido em uma única máquina (ou nodo) de um ambiente de alta exatidão. Para validar o ambiente proposto foram realizados testes básicos abordando o cálculo do produto escalar, a multiplicação entre matrizes, a implementação de *solvers* intervalares para matrizes densas e bandas e a implementação de alguns métodos numéricos para a resolução de sistemas de equações lineares com a característica da alta exatidão. Destes testes foram realizadas análises e comparações a respeito do desempenho e da exatidão obtidos com e sem o uso da biblioteca C–XSC, tanto em programas seqüenciais como em programas paralelos. Com a conseqüente implementação dessas rotinas e métodos será aberto um vasto campo de pesquisa no que se refere ao estudo de aplicações reais de grande porte que necessitem durante a sua resolução (ou em parte dela) da realização de operações aritméticas com uma exatidão melhor do que a obtida usualmente pelas ferramentas computacionais tradicionais.

Palavras-chave: Alto Desempenho, Alta Exatidão, Biblioteca C–XSC, Cluster de Computadores, Sistemas Lineares.

High Performance Environment with High Accuracy for Resolution of Problems

ABSTRACT

The goal of this work is to available a high performance environment, like cluster computers, with high accuracy obtained by use of C–XSC library. The high accuracy in the solution of a problem is obtained by realization of intermediate calculus without rounding how they have been done in infinite precision. At the end of calculus, the result must be represented in the machine. The real exact result and the approximate result are different only by one rounding. These calculus in high accuracy must be available for some basic arithmetic operations, mainly the operations that accomplish the summation and dot product. Because of this aspects, we wish to use the high performance through a cluster environment where we have several nodes executing tasks or calculus. The communication will be done by message passing using the MPI communication library. To obtain the high accuracy in this environment extensions or changes in the parallel programs had done to guarantee that the quality of final result done on cluster, where several nodes collaborate for the final result of the calculus, maintain the same result quality obtained in one unique machine (or node) of a high accuracy environment. To validate the environment developed in this work we done basic tests about the dot product, the matrix multiplications, the implementation of interval solvers for banded and dense matrices and the implementation of some numeric methods to solve linear systems with the high accuracy characteristic. With these tests we done analysis and comparisons about the performance and accuracy obtained with and without the use of C–XSC library in sequential and parallel programs. With the implementation of these routines and methods will be open a large research field about the study of real life applications that need during their resolution (or in part of their resolution) to calculate arithmetic operations with more accuracy than the accuracy obtained by the traditional computational tools.

Keywords: High Performance, High Accuracy, C–XSC Library, Cluster Computer and Linear Systems.

1 INTRODUÇÃO

Este trabalho visa a disponibilização de um ambiente de alto desempenho com alta exatidão. A alta exatidão na solução de um problema é obtida através da realização de cálculos intermediários sem arredondamentos como se fossem em precisão infinita. Ao final do cálculo, o resultado deve ser representado na máquina. O resultado exato real e o resultado representado diferem por um único arredondamento e, caso este seja representado através de intervalos, pode-se aplicar arredondamentos para cima e para baixo nesse valor real, transformando a resposta em um intervalo que contém o resultado real exato, onde os valores do intervalo representam os dois números de máquina localizados antes e depois do resultado exato que se deseja representar. Esses cálculos em alta exatidão devem estar disponíveis para algumas operações aritméticas básicas, em especial as que possibilitam a realização de somatório e de produto escalar. A biblioteca C–XSC levou a cabo tais conceitos sendo estes disponibilizados através da implementação de tipos de dados especiais e de operações aritméticas entre esses tipos.

Com isso, deseja-se utilizar o alto desempenho através de um ambiente de *cluster* onde se tem vários nodos executando tarefas ou cálculos. A comunicação será feita por troca de mensagens usando a biblioteca de comunicação MPI. Para se obter a alta exatidão neste tipo de ambiente tem-se que disponibilizar rotinas de comunicação capazes de suportar a troca de valores representados em precisão estendida. Logo, extensões ou adaptações nos programas paralelos tiveram que ser disponibilizadas para garantir que a qualidade do resultado final realizado em um *cluster*, onde vários nodos colaboram para o resultado final do cálculo, mantivesse a mesma qualidade do resultado obtido em uma única máquina (ou nodo) de um ambiente de alta exatidão.

Ainda mais, além de garantir a qualidade, é esperado que a partir de um certo tamanho de problema o ganho no tempo de resolução por se usar mais processadores para resolver o problema cooperativamente ultrapasse o tempo gasto de *overhead* de comunicação e do tratamento dessas novas rotinas de comunicação com dados com maiores quantidades de *bits*. Na tentativa de provar essas duas teses desenvolveu-se alguns testes que serão apresentados no transcorrer deste trabalho.

O trabalho apresentado nesta tese surgiu através da realização de pesquisas que abordam o uso do paradigma da Computação Verificada na resolução de problemas computacionais nas mais diversas áreas do conhecimento. Estas pesquisas iniciaram em 1996 com o estudo de métodos intervalares para a resolução de sistemas de equações lineares (HÖLBIG, 1996). A partir deste momento e através de outras pesquisas realizadas pelo grupo GMCPAD da UFRGS (em conjunto com outras universidades do Brasil e da Alemanha) pôde-se constatar a existência de aplicações de grande porte que poderiam fazer uso das técnicas que compõem a Computação Verificada e que poderiam ser executadas em ambientes de alto desempenho.

Através de um levantamento bibliográfico, pode-se citar que, principalmente nas décadas de 80 e 90, algumas pesquisas destacaram a importância e o uso da Computação Verificada em ambientes paralelos.

Por exemplo, em 1994, a revista ***Interval Computations*** publicou um número especialmente dedicado ao desenvolvimento de algoritmos paralelos utilizando a Computação Verificada. Nesta revista, Kreinovich e Bernat (KREINOVICH V.; BERNAT, 1994) realizaram um *overview* sobre a importância e aplicabilidade da aritmética intervalar paralela, abordando a paralelização de métodos intervalares, os requisitos necessários para sua implementação e as aplicações que poderiam ser desenvolvidas utilizando essas técnicas, de onde pode-se concluir a importância do estudo do paradigma da Computação Verificada e da possibilidade que se tem de utilizá-lo na solução de problemas reais que utilizam ambientes computacionais de alto desempenho. Além disso, pode-se destacar várias ferramentas computacionais que foram desenvolvidas incorporando as características da Computação Verificada. Dentre essas ferramentas pode-se destacar as bibliotecas IntLab, IntPakX, os compiladores da Sun Microsystems FORTE Fortran HPC e FORTE C++ HPC, as linguagens de programação para Computação Científica - linguagens XSC e o coprocessador VLSI para aritmética vetorial de exatidão. Maiores detalhes sobre esses tipos de ferramentas serão abordados no capítulo 2. Além dessas ferramentas, alguns grupos de pesquisa estão desenvolvendo trabalhos que podem ser relacionados ao tema desta tese. Por exemplo, no departamento de Engenharia Química da Universidade de Notre Dame (Estados Unidos) estão sendo realizados trabalhos relacionados à implementação paralela de análise intervalar em *cluster* de *worstations* (`http://www.nd.edu/~cgau/hpc.html`). Na *Ecole Normale Superieure* de Lyon (França) estão sendo realizados trabalhos que abordam a paralelização em *multi-clusters* e o desenvolvimento da biblioteca para aritmética intervalar baseada na aritmética de multi-precisão MPFI (*Multiple Precision Floating-point Interval library*) (`http://www.ens-lyon.fr/~nrevol/nr_software.html`). Na Universidade de Houston-Downtown (Estados Unidos) estão sendo realizadas implementações paralelas sobre otimização global não-linear utilizando o pacote GlobSol com MPI (`http://happy.dt.uh.edu/~sun/ParaGlobSol.html`).

Dentro desse mesmo levantamento é importante comentar as pesquisas que foram desenvolvidas na Universidade de Karlsruhe que abordaram aspectos da Computação Verificada e da resolução de problemas numéricos em ambientes paralelos. Este levantamento foi apresentado em agosto de 2005 pelo professor Gerd Bohlender durante sua visita a PUCRS. Desta apresentação pode-se destacar os trabalhos a seguir:

- Produto escalar ótimo realizado em processadores de 16 bits (Teulel/Bohlender – 1984);
- Algoritmos para o produto escalar exato em máquinas SIMD (Oberarjuev – 1984);
- Multiplicação exata de matrizes em máquinas SIMD (Wolf v. Gudenberg – 1991);
- Produto escalar ótimo e operações matriciais em *transputers* e implementação de um *solver* para sistemas lineares em *transputers* (Kaumam/Van de Coolet/Trier – 1991/1992);
- Primeiros resultados com solução verificada de sistemas lineares em *transputers* (Reith – 1991/1993);

- Multiplicação de matrizes em diferentes redes de *transputers* (Davidenkoff/Kersten/Bohlender – 1992/1994) (BOHLENDER; DAVIDENKOFF, 1992);
- Otimização global verificada em computadores paralelos (Wiethoff – 1997);
- Solução verificada de sistemas lineares usando a linguagem $C^{\#}_{++}$ (Kersten – 1998);

De acordo com os fatores e pesquisas apresentados neste capítulo, surgiu a motivação para o desenvolvimento deste trabalho, ou seja, disponibilizar à comunidade científica e acadêmica um ambiente computacional de alto desempenho que possibilite a resolução de problemas utilizando o paradigma da Computação Verificada ou algumas de suas características. Dentro deste contexto o ambiente computacional escolhido foi o de *clusters* de computadores e a biblioteca C–XSC foi a ferramenta que irá disponibilizar, neste tipo de ambiente, o paradigma da Computação Verificada.

Nesta pesquisa está sendo abordada, de maneira mais detalhada, a característica da alta exatidão. E qual é a importância de se implementar métodos numéricos com alta exatidão? É natural que, no decorrer dos tempos, vários métodos tenham sido desenvolvidos a fim de facilitar a solução dos mais diversos tipos de problemas numéricos. Com a utilização de computadores para abordar problemas matemáticos, novos algoritmos têm sido desenvolvidos, visando, principalmente, a obtenção de um melhor desempenho computacional. Entretanto, quando se trabalha com Computação Científica, utilizando números em ponto-flutuante, deve-se preocupar com o controle dos erros gerados pelas computações numéricas, que muitas vezes podem produzir resultados totalmente errados devido à inexatidão da representação numérica, já que o resultado gerado é apenas uma aproximação do resultado exato. Os erros podem ser gerados por arredondamentos, por cancelamento ou pela instabilidade numérica dos algoritmos ou problemas. Neste sentido, muitos têm sido os esforços de pesquisa para elaborar uma aritmética que supere as limitações impostas pelas características inerentes a aritmética computacional ordinária, assim como para elaborar uma fundamentação teórica para a área de Computação Científica. Tanto dos pontos de vista numéricos e teóricos, alguns dos problemas encontrados na tentativa de atingir tal objetivo não foram solucionados ainda, não tanto pela qualidade das pesquisas, mas por causa da abordagem utilizada. Até os dias de hoje tem-se buscado uma combinação de *software* (métodos de inclusão monotônica) com o *hardware* (aritmética de alta exatidão, matemática intervalar, arredondamentos direcionados, produto escalar ótimo, etc.) para que a tarefa de decidir se o resultado é ou não satisfatório seja transferido para o computador, ou seja, que se tenha a Computação Verificada ou a Validação Numérica (CSENDES, 1998).

Com a conseqüente implementação de métodos numéricos computacionais com alta exatidão, um vasto campo de pesquisa poderá ser aberto no que se refere ao estudo de aplicações reais de grande porte que necessitam durante a sua resolução (ou em parte dela) da realização de operações aritméticas com uma exatidão melhor do que a obtida usualmente pelas ferramentas computacionais tradicionais. Visando esse tipo de aplicações é que este trabalho irá disponibilizar um ambiente computacional de alto desempenho com alta exatidão e, através deste ambiente disponibilizado, possibilitar o desenvolvimento e/ou implementação de rotinas para a resolução dos mais diversos tipos de problemas. Neste trabalho procurou-se dar destaque a métodos para a resolução de sistemas de equações lineares que são utilizados na resolução de aplicações reais. Essas aplicações são relacionadas a hidrodinâmica (RIZZI et al., 2004,?) e a simulação de processos de aeração de grãos em silos de armazenagem (COPETTI; PADOIN; KHATCHATOURIAN, 2002a; COPETTI et al., 2003).

Este trabalho está estruturado da seguinte maneira: No capítulo 2 são apresentados os conceitos básicos do paradigma da Computação Verificada e da Alta Exatidão.

O capítulo 3 descreve o ambiente de alta exatidão e alto desempenho disponibilizado por este trabalho. O capítulo 4 descreve os dois *solvers* verificados que foram desenvolvidos como parte da pesquisa desta tese.

No capítulo 5 são abordados os métodos numéricos para a resolução de sistemas de equações lineares estudados e implementados com a característica da alta exatidão (métodos Gradiente Conjugado, Givens e Householder) relacionados a aplicações reais, as quais são apresentadas, de forma resumida, no capítulo.

No capítulo 6 são apresentados os testes realizados durante o desenvolvimento e validação do ambiente de alta exatidão com alto desempenho. Serão abordados os testes referente ao cálculo do produto escalar, multiplicação de matrizes e resolução de sistemas lineares (referente aos métodos citados no capítulo 5). Serão analisados os resultados tanto do ponto de vista do desempenho computacional como da exatidão dos resultados. Como informação adicional ao trabalho que está sendo apresentado alguns resultados de testes com outros métodos numéricos serão descritos e comentados.

Ao final, o capítulo 7 apresenta as conclusões obtidas com a pesquisa desenvolvida nesta tese, suas contribuições para a área de pesquisa, as possibilidades de futuros trabalhos originados através desta tese e uma descrição das principais publicações que foram originadas por este trabalho desde o ano de 2002.

2 ESTADO DA ARTE

Este capítulo apresenta uma visão geral sobre o estado da arte sobre a Computação Verificada. Durante o transcorrer do texto será apresentado uma descrição sobre a biblioteca C–XSC e suas características, sobre o desenvolvimento de *solvers* para a resolução de sistemas lineares e a contribuição desta tese para a área da Computação Verificada.

2.1 Computação Verificada

O cálculo numérico com a verificação de resultados é de fundamental importância para muitas aplicações como, por exemplo, para a simulação e modelagem matemática. Na análise clássica do erro em algoritmos numéricos, o erro em cada operação em ponto-flutuante é estimado. Na verdade, a possibilidade do resultado estar errado não é normalmente considerada. Do ponto de vista matemático, o problema da correção dos resultados é de grande importância para garantir a alta velocidade computacional atingida atualmente. Isto torna possível ao usuário distinguir entre inexatidão nos cálculos e as reais propriedades do modelo. No sentido de tornar possível o manuseio de milhões de adições e subtrações com o máximo de exatidão, é evidente que as capacidades da aritmética de ponto-flutuante tradicional têm que ser aumentadas. Dadas as possibilidades atuais, não há razão para que isto não possa ser feito em um *chip*, simulado por *software* ou pela combinação dos dois.

O grande objetivo da Computação Verificada é possibilitar que o próprio computador possa rapidamente estabelecer se o cálculo realizado é ou não correto e útil para o problema que se quer solucionar. Neste caso, o programa pode escolher um algoritmo alternativo ou repetir o processo usando uma maior precisão. Técnicas similares de computação verificada podem ser aplicadas para muitas áreas de problemas algébricos, tais como a solução de sistemas de equações lineares e não lineares, o cálculo de raízes, a obtenção de autovalores e de autovetores de matrizes, problemas de otimização, etc. Em particular, a validade e a alta exatidão da avaliação de expressões aritméticas e de funções no computador está incluída. Estas rotinas também funcionam para problemas com dados intervalares. É importante ressaltar que para que se desenvolvam programas computacionais que tenham o paradigma da Computação Verificada é obrigatório o uso de todas as suas características, ou seja, o uso da aritmética de alta exatidão, dos métodos intervalares de inclusão e da convergência garantida pelo Teorema de Ponto Fixo de Brouwer, além do uso de algoritmos apropriados. A figura 2.1 apresenta os requisitos necessários para se obter a Computação Verificada.

Com a Computação Verificada é possível desenvolver métodos numéricos para a realização de computações científicas com verificação automática do resultado. Uma abordagem detalhada sobre a Computação Verificada e suas técnicas poderá ser encontrada nos

livros textos básicos da área, como os de Moore (MOORE, 1966, 1979), Alefeld (ALEFELD; HERZBERGER, 1983), Neumaier (NEUMAIER, 1990), Claudio (CLAUDIO; MARINS, 1989), Kulisch (KULISCH; MIRANKER, 1981, 1983) e Miranker (MIRANKER; TOUPIN, 1986).

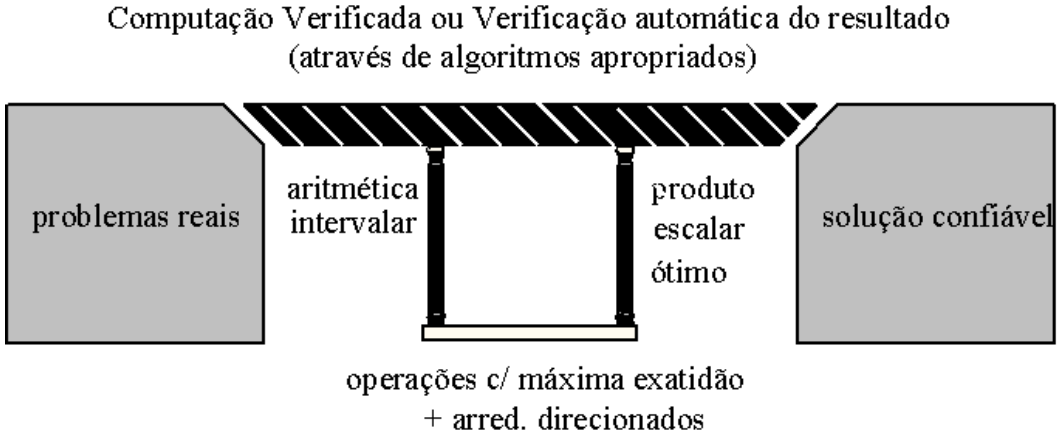

Figura 2.1: Requisitos para a Computação Verificada (FERNANDES, 1997)

Nas próximas seções desse item serão abordados os conceitos que possibilitam que se tenha a Computação Verificada no computador.

2.1.1 Aritmética Intervalar

A matemática intervalar tem sido utilizada em diversas áreas como, por exemplo, para resolver sistemas de equações lineares e não lineares, equações diferenciais ordinárias e parciais, equações integrais e problemas de otimização, como pode ser encontrado em (HAMMER et al., 1993) e (KRÄMER; KULISCH; LOHNER, 1996). Para cada uma dessas classes de problemas numéricos, o emprego da matemática intervalar tem sido acompanhada pelo desenvolvimento de novas técnicas, as quais vão além da mera substituição dos coeficientes reais por intervalos e do uso de operações intervalares.

A extensão ingênua, por assim dizer, de métodos pontuais em métodos intervalares tem-se mostrado ser ineficiente, pois não há convergência, como foi observado, por exemplo, na extensão ingênua do método de Newton para o cálculo de raízes reais (DIVERIO; FERNANDES, 1994).

Uma mera aplicação da aritmética intervalar levará a extremos confiáveis. O uso desta aritmética, aliada a um controle rígido dos algoritmos, é objeto do atual estado da arte nesta área.

A Aritmética Intervalar trata com dados na forma de intervalos numéricos e surgiu com o objetivo de automatizar a análise do erro computacional, trazendo uma nova abordagem que permite um controle de erros com limites confiáveis, além de provar a existência ou não existência da solução de diversas equações, ou seja, surgiu com o objetivo de automatizar a análise do erro computacional. Várias das características do padrão IEEE-754 são necessárias para a definição da aritmética intervalar. Entre elas pode-se destacar a forma de representação dos números de ponto-flutuante, os intervalos de representatividade, os símbolos especiais de mais e menos infinito ($+\infty$ e $-\infty$) e o símbolo *not-a-number* (NaN), o tratamento de *underflow* e *overflow* e de outras exceções e, por fim, as operações com máxima exatidão.

O uso da aritmética intervalar foi desenvolvido inicialmente por Moore (MOORE, 1966, 1979). Este ramo da matemática veio se desenvolvendo desde então. A matemática intervalar foi proposta em 1974 por Leslie Fox, combinando diferentes áreas como: aritmética intervalar, análise intervalar, topologia intervalar, álgebra intervalar entre outras. Intervalos podem ser aplicados para representar valores desconhecidos ou para representar valores contínuos que podem ser conhecidos ou não. No primeiro sentido servem para controlar o erro de arredondamento e para representar dados inexatos, aproximações e erros de truncamento de procedimentos (como consistência lógica de programas e critério de parada de processos iterativos). No segundo sentido servem, por exemplo, para testes de verificação computacional e para a existência e unicidade de soluções de sistemas não lineares. A condição necessária não é verificada manualmente mas automaticamente pelo computador.

O uso da Matemática Intervalar sofreu críticas em função de dois problemas principais, os quais se resumem em que, muitas vezes, podem resultar intervalos pessimistas, ou seja, demasiadamente grandes, que não possuem muita informação sobre o resultado, gastando-se muito tempo de máquina. Estas duas críticas foram facilmente refutadas, uma vez que, com a aritmética avançada, os resultados são produzidos com máxima exatidão. Quanto ao tempo de processamento, depende da forma como foi implementada. Existem várias formas, tanto via *software*, via *hardware* ou através da combinação dos dois. A seguir serão apresentados alguns conceitos básicos sobre aritmética intervalar.

Um intervalo real, ou simplesmente um intervalo, é um conjunto fechado e limitado de números reais. Sejam x_1 e x_2 dois números reais tal que x_1 é menor ou igual a x_2. Defini-se o intervalo finito X, o qual será denotado por $[x_1, x_2]$, pelo conjunto descrito em 2.1, onde x_1 é o limite inferior e x_2 é o limite superior do intervalo X. Um intervalo real cobre todos os números reais entre os dois limites. O conjunto de todos os intervalos reais é denotado por $I\Re$. Todas as operações aritméticas, operadores relacionais e funções elementares são definidas para argumentos intervalares.

$$X = [x_1, x_2] = \{x \mid x_1 \leq x \leq x_2\} \tag{2.1}$$

A partir da aritmética intervalar são desenvolvidos os conceitos que compõem a matemática intervalar e os métodos intervalares para resolução de problemas numéricos. A tabela 2.1 apresenta algumas das operações existentes sobre o conjunto de intervalos.

2.1.2 Computação de Alta Exatidão

O computador foi inventado para, entre outras coisas, fazer o trabalho complicado e/ou repetitivo para o homem. Na questão de cálculos em ponto-flutuante, a evidente discrepância entre o poder computacional e o controle dos erros computacionais sugere que se coloque o processo de estimativa de erro dentro do próprio computador. Para fazer isso o computador tem que ser feito aritmeticamente mais poderoso que o comum. Na aritmética de ponto-flutuante ordinária (definida pelo padrão IEEE), a maioria dos erros ocorre em acumulações, isso é, pela execução de uma seqüência de adições e subtrações. Na aritmética de ponto-fixo, contudo, a operação de acumulação é realizada sem erros. Assim, grande parte dos erros encontrados em ponto-flutuante poderiam ser evitados se a acumulação fosse realizada em ponto-fixo, em um acumulador especial (MIRANKER; TOUPIN, 1986).

Com a atual tecnologia pode-se realizar facilmente essa acumulação de ponto-fixo. Necessita-se, somente, providenciar um registrador de ponto-fixo na unidade de aritmética que cubra todo o domínio em ponto-flutuante. Se um registrador que possui esta

Tabela 2.1: Operações sobre o conjunto de intervalos

Descrição	**Operação**
Inverso aditivo de X	$-X = [-x_2, -x_1]$
Pseudo inverso multiplicativo de X	$\frac{1}{X}$ ou $X^{-1} = [\frac{1}{x_2}, \frac{1}{x_1}]$, $0 \notin X$
Adição de $X + Y$	$X + Y = [x_1 + y_1, x_2 + y_2]$
Subtração de X por Y	$X - Y = [x_1 - y_2, x_2 - y_1]$
Multiplicação de X por Y	$X \times Y = [min\{x_1 \times y_1, x_1 \times y_2, x_2 \times y_1, x_2 \times y_2\}, max\{x_1 \times y_1, x_1 \times y_2, x_2 \times y_1, x_2 \times y_2\}]$
Divisão de X por Y	$X \div Y = [min\{x_1/y_1, x_1/y_2, x_2/y_1, x_2/y_2\}, max\{x_1/y_1, x_1/y_2, x_2/y_1, x_2/y_2\}]$, $0 \notin Y$
Intersecção entre X e Y	$X \cap Y = \{[max\{x_1, y_1\}, min\{x_2, y_2\}]$, se $x_1 < y_2$ e $y_1 \leq x_2, \varnothing$ - caso contrário
União entre X e Y	$X \cup Y = \{[min\{x_1, y_1\}, max\{x_2, y_2\}]$, se $X \cap Y \neq \varnothing$, ERRO - caso contrário
Distância entre X e Y	$d(X, Y) = max\{\mid x_1 - y_1 \mid, \mid x_2 - y_2 \mid\}$
Valor Absoluto de X	$\mid X \mid = d(X, [0, 0]) = max\{\mid x_1 \mid, \mid x_2 \mid\}$
Diâmetro de X	$diam(X) = x_2 - x_1$
Ponto Médio de X	$med(X) = \frac{x_1 + x_2}{2}$

característica não está disponível, então esse pode ser simulado na memória principal por *software*. Isso infelizmente pode resultar em perda de velocidade, que, em muitos casos, é mais importante do que o ganho em confiabilidade do resultado.

Por outro lado, se esse registrador tiver dupla precisão, então os produtos escalares de vetores de qualquer dimensão finita sempre podem ser calculados de forma exata. A possibilidade de calcular os produtos escalares de vetores em ponto-flutuante de qualquer dimensão, com exatidão total, abre uma nova dimensão na análise numérica. Em particular, o produto escalar ótimo prova ser um instrumento essencial para alcançar maior exatidão computacional. No sentido de adaptar o computador para esta nova tarefa (o controle automático de erros) sua aritmética deve ser estendida ainda com um outro elemento.

Todas as operações com números de ponto-flutuante, que são a adição, a subtração, a divisão, a multiplicação e o produto escalar ótimo de vetores em ponto-flutuante devem ser supridos com os arredondamentos direcionados, isso é, arredondamentos para o número de máquina anterior (para baixo, por corte ou truncamento – ∇x), posterior (para cima ou por excesso – $\triangle\ x$) e para o mais próximo de máquina (simétrico – Ox).

A Aritmética de Alta Exatidão possibilita que os cálculos sejam efetuados com máxima exatidão, isto é, o resultado calculado difere do valor exato no máximo em um arredondamento. Para isto é necessário que o formato ou tipo de dado e as operações aritméticas suportadas pelo *hardware* ou pela linguagem de programação satisfaçam as condições de um semimorfismo. Todas as operações definidas deste modo são, então, de máxima exatidão.

O passo inicial para a obtenção da aritmética de alta exatidão foi a definição do padrão de aritmética binária IEEE-754 (IEEE, 1985), o qual surgiu com o objetivo de padronizar a aritmética binária de ponto-flutuante em todas as plataformas. Entretanto, esse padrão não especificou os arredondamentos para variáveis complexas, operações entre vetores e matrizes e, também, não incluiu o produto escalar ótimo, essencial para a garantia da alta

exatidão nos cálculos. Por causa disso, a GAMM/IMACS propôs outro padrão (IMACS; GAMM, 1990), que aborda esses tópicos.

Os requisitos da aritmética de alta exatidão são: arredondamentos direcionados e operações aritméticas com máxima exatidão (detalhes em (KULISCH; MIRANKER, 1981) e (MIRANKER; TOUPIN, 1986)).

As operações aritméticas com máxima exatidão são necessárias para se ter uma aritmética de alta exatidão. Elas são definidas de forma que só um arredondamento é aplicado nas operações aritméticas básicas, resultando que o valor calculado e o valor exato diferem por apenas um arredondamento. Se $\circ$ é uma operação aritmética no espaço dos números reais $\Re$, então a correspondente operação no computador $\boxdot$ no conjunto dos números de máquina F é definida por: $x \boxdot y := \Box(x \circ y)$ para todos $x, y \in F$ (onde $\Box$ é um arredondamento). Isto é, a operação no computador deve ser efetuada como se o resultado exato fosse calculado primeiramente e, então, aproximado com o arredondamento selecionado.

Resumindo, operações semimórficas para números reais e complexos, vetores e matrizes, assim como para intervalos reais e complexos, vetores e matrizes intervalares, podem ser realizadas em termos de 15 operações aritméticas fundamentais em aritmética de ponto-flutuante: $+, -, *, /, .$, cada uma com os arredondamentos para cima, para baixo e para o mais próximo de máquina.

O uso da Aritmética de Alta Exatidão, aliado à matemática intervalar, resulta em resultados confiáveis e com máxima exatidão, onde o resultado está contido em um intervalo cujos extremos diferem por apenas um arredondamento do valor real. Os cálculos intermediários são feitos em registradores especiais, de forma a simular a operação dos reais, sendo o arredondamento feito só no final, onde em cada extremo é aplicado o arredondamento direcionado para baixo e para cima. Na seção 3.2 é apresentado como a aritmética de alta exatidão é implementada e utilizada na biblioteca C–XSC.

2.1.3 Arredondamentos Direcionados

Os arredondamentos direcionados são funções de mapeamento utilizadas para representar os números reais em números de máquina (KULISCH; MIRANKER, 1981). Quando a função de mapeamento é aplicada a qualquer elemento do conjunto dos números de máquina ela produz o próprio número de máquina. Para se ter implementada uma aritmética de alta exatidão, deve-se ter os dois tipos de arredondamentos direcionados:

- Arredondamento para cima ($\triangle x$) é a função que aproxima o número real x para o maior número de máquina que o contém (conforme 2.3).
- Arredondamento para baixo (∇x) é a função que aproxima o número real x para o menor número de máquina que o contém (conforme 2.2).

$$\nabla x = min\{y \in F / y \geqslant x\} \forall x \in \Re \tag{2.2}$$

$$\triangle x = max\{y \in F / y \leqslant x\} \forall x \in \Re \tag{2.3}$$

Esses arredondamentos são necessários para se ter a garantia nos cálculos, sendo fundamentais para a implementação da aritmética intervalar de máquina. Outro tipo de arredondamento é o arredondamento simétrico (ou para o número mais próximo de máquina – Ox). Este arredonda o número real x para o número de máquina mais próximo. É definido em função dos arredondamentos anteriores e produz um menor erro de aproximação (conforme 2.4).

$$Ox = \begin{cases} \nabla x & \text{se } |x - \nabla x| < |x - \triangle x| \\ \nabla x & \text{se } |x - \nabla x| = |x - \triangle x| \text{ e } \nabla x \text{ é par} \\ \triangle x & \text{em caso contrário} \end{cases} \tag{2.4}$$

2.1.4 Produto Escalar Ótimo

A Aritmética de Alta Exatidão é assim chamada porque o resultado por meio dela calculado difere do valor exato no máximo em um arredondamento. Para isso é necessário que o formato ou tipo de dado e as operações aritméticas suportadas pelo *hardware* ou pela linguagem de programação satisfaçam as condições de um semimorfismo. Todas as operações definidas desse modo são, então, de máxima exatidão. Uma das formas de se obtê-la é por meio do uso de registradores especiais que armazenam o resultado parcial sem arredondamento (essa técnica é utilizada pela biblioteca C–XSC). O produto escalar entre dois vetores de dimensão n, definido da maneira tradicional, envolve $2n - 1$ arredondamentos (conforme 2.5), um após cada operação, o que pode, em certos casos, provocar o cancelamento de um grande número de dígitos significativos, gerando um resultado completamente incorreto.

$$v \times w = \square \sum_{i=1}^{n} \square v_i \times w_i \tag{2.5}$$

Com o objetivo de evitar este erro, o produto escalar ótimo é realizado através da definição de um registrador com precisão "infinita", que permitirá o acúmulo dos resultados parciais sem perda de exatidão. Ao final do cálculo é aplicado um arredondamento, escolhido pelo usuário, sobre o produto e este é transformado em um número em ponto-flutuante. Dessa forma, o resultado calculado difere do exato por apenas um arredondamento, obtendo-se, com isso, o produto escalar ótimo, conforme descrito em (2.6).

$$v \times w = \square \sum_{i=1}^{n} v_i \times w_i \tag{2.6}$$

2.2 *Solvers* Seqüenciais, Paralelos e Verificados

Esta seção tem por objetivo realizar um breve levantamento a respeito de *solvers* verificados ou bibliotecas científicas que estão vinculadas ao tema deste trabalho. Esse levantamento é importante para mostrar a existência desse grande conjunto de ferramentas as quais poderão, futuramente, servir de comparação para os *solvers* e bibliotecas que serão desenvolvidos nesse ambiente de alto desempenho e de alta exatidão. Para efeitos didáticos procurou-se dividir essas ferramentas em ferramentas verificadas (que possuam pelo menos algumas das características da Computação Verificada) e ferramentas tradicionais (ferramentas seqüenciais e paralelas que são utilizadas na área de Computação Científica).

As ferramentas verificadas podem ser bibliotecas desenvolvidas sob algum tipo de software matemático (MatLab, Maple, ...) ou linguagem de programação (C, Pascal, Fortran, ...) ou *solvers* para a resolução de algum problema ou aplicação em específico. Dentre essas ferramentas pode-se citar:

- Linguagens de programação para computação científica (linguagens ou bibliotecas XSC): essas linguagens são formadas pelo Pascal-XSC, pelo C-XSC e pelo

ACRITH-XSC (a biblioteca C–XSC é descrita em detalhes no capítulo 3). Com exceção do ACRITH-XSC, essas bibliotecas são *freeware*. Detalhes dessas ferramentas poderão ser encontrados em (KLATTE et al., 1991), (WALTER, 1991) e (KLATTE et al., 1993);

- IntLab (***Int**erval **Lab**oratory*): biblioteca desenvolvida pelo Prof. S. Rump da *Technical University Hamburg-Harburg* (Alemanha). É um *toolbox* para o Matlab que suporta a aritmética intervalar. Foi desenvolvida com o objetivo de ser rápida e de que seu código possa ser utilizado em uma grande variedade de computadores, desde PC's até computadores paralelos. Essa velocidade é obtida pelo uso de rotinas da BLAS (*Basic Linear Álgebra Subroutines* - a IntLab é a primeira biblioteca intervalar construída sobre a BLAS). Outro aspecto dessa ferramenta é que ela possibilita um ambiente de programação interativo de fácil utilização de rotinas intervalares pois ela é um *toolbox* do Matlab (RUMP, 1998, 1999);

- PROFIL/BIAS (***P**rogrammer's **R**untime **O**ptimized **F**ast **I**nterval **L**ibrary*): é uma biblioteca de classes em C++ que suporta operações com números reais e intervalares (`http://www.ti3.tu-harburg.de/english/index.html`).

- *Toolbox* b4m: é um *toolbox* para o Matlab baseado na BIAS (*Basic Interval Arithmetic Subroutines* - (KNÜPPEL, 1993)) que suporta a aritmética intervalar. O b4m providencia uma interface para as rotinas do PROFIL/BIAS, com isso possibilita que o usuário desenvolva suas rotinas no ambiente do Matlab e depois as converta facilmente para C++ para calcular computações intensivas. Essa ferramenta foi desenvolvida com dois objetivos: usar a biblioteca de aritmética intervalar BIAS (em ANSI C) de uma maneira interativa; e adicionar as operações aritméticas intervalares e os algoritmos de inclusão ao padrão de ponto-flutuante do ambiente do Matlab. Maiores informações podem ser encontradas em (ZEMKE, 1998).

- IntPakX: o pacote IntPakX disponibiliza o desenvolvimento de algoritmos com verificação numérica no Maple (GRIMMER; PETRAS; REVOL, 2003; KRÄMER, 2002);

- Compilador GNU FORTRAN: é um projeto que modificou o compilador GNU FORTRAN para suportar a aritmética intervalar (SCHULTE; SWARTZLANDER, 1996);

- Compilador FORTE FORTRAN/HPC: compilador desenvolvido pela Sun Microsystems que suporta o uso da aritmética intervalar (ver homepage do compilador no endereço `http://www.sun.com/software/sundev/previous/fortran/`);

- Compilador FORTE C++: compilador desenvolvido pela Sun Microsystems que suporta o uso da aritmética intervalar (ver homepage do compilador no endereço `http://www.sun.com/software/sundev/previous/cplusplus/index.xml`).

Sobre as ferramentas seqüenciais e paralelas revisadas para este trabalho pode-se destacar:

- BLAS (***B**asic **L**inear **A**lgebra **S**ubroutines*): é uma biblioteca matemática de subrotinas de operações básicas da Álgebra Linear. Inicialmente ela foi desenvolvida

em FORTRAN, para uso em programas desenvolvidos nesta linguagem, mas atualmente já existem diversas implementações e interfaces para diversas linguagens de programação, como C e, mais recentemente, JAVA (`http://www.netlib.org/blas`);

- PETSc (***P**ortable, **E**xtensible **T**oolkit for **S**cientific **C**omputatio*n): é um pacote paralelo de estruturas de dados e rotinas para a solução de aplicações científicas modeladas por equações diferenciais parciais (`http://www-unix.mcs.anl.gov/petsc/petsc-2/index.html`);

- IML++ (***I**terative **M**ethods **L**ibrary*): é biblioteca do C++ com métodos iterativos para a solução de sistemas de equações lineares simétricos e não-simétricos com matrizes densas e esparsas (`http://math.nist.gov/iml++/`);

- Aztec: é uma biblioteca paralela para a resolução de sistemas de equações lineares esparsos através do uso de métodos iterativos (`http://www.cs.sandia.gov/CRF/aztec1.html`);

- PSPASES (***P**arallel **SPA**rse **S**ymmetric dir**E**ct **S**olver*): é uma biblioteca de alto desempenho, escalável, paralela e baseada na biblioteca MPI, que visa a solução de sistemas de equações lineares envolvendo matrizes esparsas simétricas definidas e positivas, provendo várias *interfaces* para solucionar o sistema através de métodos diretos (`http://www-users.cs.umn.edu/~mjoshi/pspases/`);

- SuperLU: é uma biblioteca de propósito geral para a solução direta de grandes sistemas de equações lineares esparsos e não-simétricos em computadores de alto desempenho. É uma biblioteca escrita em C e suporta chamadas em C ou Fortran (`http://crd.lbl.gov/~xiaoye/SuperLU/`).

Informações sobre outras bibliotecas/ferramentas verificadas poderão ser obtidas no endereço `http://www.cs.utep.edu/interval-comp/intsoft.html`, portal sobre a Computação Verificada. A respeito de outras bibliotecas/*solvers* tradicionais, informações adicionais poderão ser encontradas em (DONGARRA et al., 2003) e (KARNIADAKIS; KIRBY II, 2003).

2.3 Instabilidade do Problema e do Algoritmo

Este trabalho, através da biblioteca C–XSC, foca a qualidade numérica dos resultados gerados pelo computador. Dessa forma, é importante citar o problema da instabilidade numérica. Esta instabilidade pode ser entendida como uma *sensibilidade a perturbações* que pode ocorrer tanto no problema como no algoritmo. Os erros causados por essa instabilidade podem ser causados pelos modelos ou entradas de dados, pelo truncamento ou pelos erros de arredondamentos. A instabilidade dos algoritmos podem causar, por exemplo, erros de truncamento quando da soma de grandezas de diferentes ordens ou, de acordo com a ordem de como as operações estão sendo realizadas, problemas com *overflow* e *underflow*. Já a instabilidade dos problemas ocorre, por exemplo, quando pequenas alterações nos dados de entrada de um problema provocam grandes alterações na sua resposta (problemas mal condicionados). Com o intuito de controlar ou minimizar esses tipos de erros é que o uso da alta exatidão, aliada com a matemática intervalar, pode nos propiciar uma melhor qualidade nos resultados obtidos pelo computador. Uma

descrição mais detalhada sobre instabilidade numérica pode ser obtida em (CLAUDIO; MARINS, 1989).

2.4 Conclusões do Capítulo

Neste capítulo procurou-se apresentar uma visão geral sobre a Computação Verificada e suas características. Um correto conhecimento dessas características é de fundamental importância para que os futuros usuários desse novo ambiente computacional possam tirar o máximo de proveito não só da alta exatidão mas também das demais características que, em conjunto, permitem que se tenha programas que verifiquem automaticamente os resultados sendo executados em *clusters* de computadores. Além do aspecto da alta exatidão não deve-se esquecer que vários requisitos devem ser abordados para que se tenha um programa de alto desempenho. Estes requisitos devem abordar os seguintes aspectos, conforme apresentado em (MAILLARD, 2005):

- otimização do *hardware* (processador/rede);
- adaptação do sistema operacional;
- uso de *middlewares* específicos;
- programação otimizada;
- algoritmos com eficiência comprovada.

Dentro da idéia central deste trabalho, apresentou-se um levantamento sobre algumas ferramentas computacionais que utilizam as técnicas da Computação Verificada e algumas bibliotecas científicas paralelas (não verificadas), em especial as que visam a resolução de problemas da álgebra linear, visto que para a validação do ambiente proposto por este trabalho optou-se por utilizar problemas desse tipo. Além disso, com os *solvers* verificados implementados para a solução de sistemas de equações lineares com matrizes densas e esparsas (descritos no capítulo 4) e com a idéia de sua futura paralelização, um estudo comparativo entre as bibliotecas citadas neste capítulo será de grande validade para a verificação da aplicabilidade, exatidão e desempenho desses *solvers* verificados paralelos a serem desenvolvidos.

Para finalizar, uma pergunta deve ser respondida: Qual o motivo da escolha da biblioteca C–XSC como a ferramenta que disponibilizará a alta exatidão nesta pesquisa? Essa escolha se deve, principalmente, a dois fatores: disponibilidade e usabilidade. Disponibilidade por ser uma biblioteca *freeware* e vir ao encontro da idéia dos grupos de pesquisa com qual esse trabalho está vinculado; e usabilidade por o C–XSC ser uma biblioteca da linguagem C/C++, linguagem que está disponível em praticamente todos os ambientes computacionais e que proporciona que sejam desenvolvidos programas em C/C++ que possam utilizar, em conjunto, rotinas do C–XSC com de outras bibliotecas científicas como, por exemplo, rotinas da biblioteca BLAS.

3 AMBIENTE DE ALTA EXATIDÃO E DE ALTO DESEMPENHO

Este capítulo apresenta uma descrição a respeito dos ambientes computacionais que foram utilizados durante a realização desta tese de doutorado. São abordados, também, aspectos sobre a biblioteca de alta exatidão C–XSC e sobre a biblioteca de troca de mensagens MPI utilizada na implementação dos programas paralelos. Ao final, é detalhado o processo de integração e utilização da biblioteca C–XSC neste tipo de ambiente computacional paralelo, apresentando os problemas encontrados para realizar essa integração e as soluções adotadas para resolvê-los.

3.1 *Clusters* de Computadores

Como já foi colocado, este trabalho visa a disponibilização de um ambiente de alto desempenho (*clusters* de computadores) com alta exatidão. Para isso foram utilizados durante a pesquisa os *clusters* labtec do Instituto de Informática da UFRGS, ALiCE da Universidade de Wuppertal e Colorado da Universidade de Passo Fundo com o objetivo de validar a idéia proposta neste trabalho. Estes ambientes serão caracterizados, resumidamente, nas próximas seções.

3.1.1 *Cluster* labtec

O *cluster* labtec faz parte do **LabTeC**[1] - Laboratório de Tecnologias de *Cluster* - do Instituto de Informática da UFRGS, o qual constitui-se em um laboratório voltado a tecnologia de alto desempenho, mais especificamente em *clusters*, onde são ministrados cursos de treinamentos, visando a formação de pessoal qualificado. O LabTeC tem por objetivo oferecer uma formação complementar aos currículos da graduação e da pós-graduação da UFRGS, oferecendo cursos avançados e desenvolvendo pesquisa relacionada ao processamento de alto desempenho com *clusters*, de forma a capacitar futuros profissionais para a atuação com excelência nesta área.

A configuração atual do *cluster* ***labtec***, seus nodos e servidor de acesso é estruturada da seguinte maneira:

- Configuração do nodos: processador Dual Pentium III 1.1 *Ghz*, 1 *Gb* de memória RAM, HD SCSI de 18 *Gb* e Placa de rede *Gigabit Ethernet*;
- Servidor de acesso ao *cluster* (front-end): processador Dual Pentium IV Xeon 1.8 *Ghz*, 1 *Gb* de memória RAM, HD SCSI de 36 *Gb* e Placa de rede *Gigabit Ethernet*.

[1] `http://www.inf.ufrgs.br/labtec/`

O labtec está equipado, atualmente, com um *cluster* da Dell de 20 nodos Pentium III 1 *GHz* biprocessados (totalizando 40 processadores), conectados por tecnologia *Fast Ethernet*. Do total de nodos, 8 estão permanentemente disponíveis para aplicações e treinamento em programação paralela e 12 são utilizados nos cursos de instalação e configuração.

3.1.2 *Cluster* ALiCE

O cluster ALiCE[2] (***A**lpha-**Li**nux-**C**luster-**E**ngine*) da Universidade de Wuppertal (Alemanha) é um cluster mono-processado formado por 128 Compaq DS10 *Workstations*. Cada um dos 128 nodos possuem a seguinte configuração:

- processadores Alpha 21264 EV6/7 de 616 *MHz* com 2 *Mb* de memória *cache*, conectados por uma rede Myrinet de 1.28 Gigabit, 256 *Mb* de memória RAM e HD de 10 *Gb*.

O *cluster* ALiCE possui espaço total em disco de 1.3 *Tb* e 32 *Gb* de memória RAM. É utilizado pelos grupos de pesquisa das áreas de física e matemática da Universidade de Wuppertal e para o ensino de alunos de graduação.

3.1.3 *Cluster* Colorado

O *cluster* da Universidade de Passo Fundo é chamado de *Cluster* Colorado[3]. Ele é composto por 6 máquinas do tipo PC, com sistema operacional Linux e placa de rede de 100 *Megabit Enthernet*. O *cluster* Colorado é estruturado da seguinte maneira:

- Servidor de acesso ao *cluster* (front-end): processador Intel Celeron 1.7 *GHz*, 128 *Mb* de memória RAM e HD de 40 *Gb*;
- Configuração de 1 nodo: processador Pentium III 800 *MHz*, 128 *Mb* de memória RAM e HD de 40 *Gb*;
- Configuração de 4 nodos: processador AMD Atlhon 1.16 *GHz*, 128 *Mb* de memória RAM e HD de 40 *Gb*.

O *cluster* Colorado é utilizado atualmente nas pesquisas desenvolvidas pelo grupo de pesquisa ComPaDi da UPF e para o ensino dos alunos do curso de Ciência da Computação da UPF.

3.2 Biblioteca de Alta Exatidão C–XSC

O C–XSC é uma biblioteca numérica para a Computação Científica baseada na linguagem C++. É uma ferramenta para desenvolvimento de algoritmos numéricos com a geração de resultados com alta exatidão e verificados automaticamente. Ela fornece um grande número de tipos de dados numéricos e operadores predefinidos. Estes tipos são implementados como classes da linguagem C++. Assim, o C–XSC permite a programação de alto nível de aplicações numéricas em C++. Ela está disponível para muitos ambientes computacionais que possuam um compilador C++. C–XSC obedece ao padrão ISO/IEC C++. O desenvolvimento da biblioteca C–XSC iniciou em 1990 no Instituto

[2] `http://www.theorie.physik.uni-wuppertal.de/Computerlabor/ALiCE.phtml`
[3] `http://inf.upf.br/compadi/br/cluster.php`

de Matemática Aplicada da Universidade de Karlsruhe (Alemanha). Desde então muitos pesquisadores vêm contribuído diretamente e indiretamente para o seu desenvolvimento. Atualmente o C–XSC é resultado da colaboração do Instituto de Matemática Aplicada da Universidade de Karlsruhe e do grupo de pesquisa em Engenharia de Software e Computação Científica da Universidade de Wuppertal. Atualmente a versão disponível do C–XSC é a C–XSC 2.0 (o C–XSC é uma biblioteca *freeware*). As características mais importantes do C–XSC são:

- Aritmética intervalar para números reais, complexos, intervalares e intervalares complexos com propriedades definidas matematicamente;
- Vetores e matrizes dinâmicos;
- *Subarrays* de vetores e matrizes;
- Tipos de dados de alta exatidão;
- Operadores aritméticos predefinidos com alta exatidão;
- Aritmética de múltipla precisão dinâmica e funções padrão;
- Controle de arredondamento para os dados de entrada e saída;
- Biblioteca de rotinas para a resolução de problemas numéricos;
- Resultados numéricos com rigor matemático.

Nos itens a seguir serão apresentados e exemplificados alguns dos mais importantes conceitos disponibilizados pela biblioteca C–XSC, em especial os referentes ao paradigma da Computação Verificada. Maiores detalhes sobre a biblioteca C–XSC são descritos em (HAMMER et al., 1995), (KLATTE et al., 1993), (KRÄMER, 2002) e (HOFSCHUSTER; KRÄMER, 2001).

3.2.1 Tipos de Dados

Além dos tipos numéricos tradicionais disponíveis no C/C++, o C–XSC fornece outros tipos de dados numéricos simples: *real*, *interval* (intervalo de reais), *complex* (número complexo) e *cinterval* (intervalo complexo), com seus operadores relacionais e aritméticos apropriados e funções matemáticas padrão. Todos os operadores aritméticos predefinidos entregam resultados com a exatidão de no mínimo 1 *ulp* (erro de arredondamento na última posição da mantissa). Assim, eles são de máxima exatidão no senso da computação científica. Funções de *typecast* estão disponíveis para todas as combinações úteis matematicamente. Constantes literais podem ser convertidas com máxima exatidão. Todas as funções matemáticas padrão para os tipos de dados numéricos simples podem ser chamadas através de seus nomes genéricos e entregam resultados com alta exatidão garantida para argumentos arbitrariamente admissíveis. As funções matemáticas elementares para o tipo de dados *interval* fornecem várias inclusões que são exatamente arredondadas. Funções elementares para o tipo de dados *cinterval* também estão disponíveis.

Para o tipo de dados escalares apresentados acima, estão disponíveis tipos dados que representam matrizes e vetores: *rvector* (vetor de reais), *ivector* (vetor de intervalos reais), *cvector* (vetor de complexos), *civector* (vetor de intervalos complexos), *rmatrix* (matriz de reais), *imatrix* (matriz de intervalos reais), *cmatrix* (matriz de complexos) e *cimatrix* (matriz de intervalos complexos).

Além desses tipos de dados, o C–XSC possui uma série de funções matemáticas padrão com suporte aos tipos de dados *real*, *complex*, *interval* e *cinterval* com seus nomes genéricos. Entre essas funções pode-se citar seno, co-seno, tangente, valor absoluto e raiz quadrada.

3.2.2 Manipulação de Matrizes e Vetores

Através dos tipos de dados especiais de matrizes e vetores do C-XSC, o usuário pode alocar ou desalocar em tempo de execução o espaço de armazenamento para um *array* dinâmico (vetor ou matriz). Assim, sem recompilação, o mesmo programa pode usar *arrays* de tamanho restrito somente através do armazenamento do computador. Além disso, a memória é usada eficientemente, desde que os *arrays* estejam armazenados somente em seus tamanhos requeridos. Quando acessam componentes do tipo *array*, é checada a variação do índice em tempo de execução para incrementar a segurança durante a programação, evitando acesso inválido à memória.

Além disso, o C-XSC fornece uma notação especial para manipular *subarrays* de vetores e matrizes. *Subarrays* são partes arbitrárias de *arrays* retangulares. Todos os operadores predefinidos podem também usar *subarrays* como operandos. Um *subarray* de uma matriz ou vetor é acessado usando o operador () ou o operador [ ]. O operador () especifica um *subarray* de um objeto do mesmo tipo que o objeto original. Por exemplo, se A é uma matriz real $n \times n$, então $A(i,i)$ está acima e a esquerda na submatriz $i \times i$. Note que os parênteses na declaração de um vetor ou matriz dinâmica não especificam um *subarray*, mas definem a variação do índice do objeto a ser alocado.

O operador [ ] gera um *subarray* de tipo "baixo". Por exemplo, se A é uma matriz $n \times n$, então $A[i]$ é a i linha de A do tipo *rvector* e $A[i][j]$ é o (i,j) elemento de A do tipo *rmatrix*. Ambos tipos de *subarray* acessados podem também estar combinados, por exemplo: $A[k](i,j)$ é um subvetor através do índice i para o índice j da linha k do vetor da matriz A. O uso de *subarrays* está ilustrado na figura 3.1 onde é apresentado um trecho de código em C–XSC que realiza a fatorização LU de uma matriz $A_{n\times n}$:

```
for (j=1; j<=n-1; j++) {
 for (k=j+1; k<=n; k++)
 {
  A[k][j] = A[k][j] / A[j][j]; /* Observacao */
  A[k](j+1,n) = A[k](j+1,n) - A[k][j] * A[j](j+1,n);
 }
}
```

Observação: Este teste não verifica a existência de elementos nulos na diagonal principal

Figura 3.1: Fatorização LU de uma matriz $A_{n\times n}$ em C–XSC

Este exemplo da fatorização demonstra duas características importantes do C–XSC. Primeiro, salva-se um laço no programa usando a notação de *subarray*, reduzindo assim a sua complexidade. Segundo, o fragmento de programa apresentado é independente do tipo numérico da matriz A (*rmatrix*, *imatrix*, *cmatrix* ou *cimatrix*), desde que todos os operadores aritméticos estejam adequadamente predefinidos no senso matemático.

Junto a essas características já apresentadas, o C–XSC possibilita que se trabalhe com a alocação ou redefinição dinâmica dos tamanhos de vetores e matrizes, ou seja, defini-se

um vetor ou uma matriz sem indicar explicitamente seus índices (enquanto não houver a alocação do tamanho do vetor ou da matriz os mesmos possuirão, respectivamente, tamanho 1 ou 1×1). Essa alocação será realizada em tempo de execução.

```
int n, m;
cout << "Entre com as dimensoes da matriz A n, m:";
cin >> n >> m;

matrix B, C, A(n, m); /* A[1][1] ... A[n][m] */
Resize(B, m, n); /* B[1][1] ... B[m][n] */ ...

C = A * B; /*C[1][1]... C[n][n]*/
```

Figura 3.2: Alocação e redefinição dinâmica matrizes em C–XSC

No exemplo apresentado na figura 3.2 usa-se a redefinição de tamanho para alocar um objeto do tamanho desejado. Alternativamente, pode ser determinado o índice correto quando é definido o vetor ou a matriz, conforme foi realizado neste exemplo com a matriz A. Um redimensionamento implícito de um vetor ou de uma matriz também é possível durante uma atribuição. Se o índice do objeto do lado direito de uma atribuição não corresponde a aquele do lado esquerdo, o objeto é mudado correspondentemente no lado esquerdo como apresentado no exemplo com a atribuição $C = A * B$. O espaço armazenado de um *array* dinâmico, que é local para um subprograma, é automaticamente liberado antes de o controle retornar para a rotina que o chamou. O tamanho de um vetor ou de uma matriz pode ser determinado em qualquer tempo chamando as funções *Lb()* e *Ub()* para o índice referente aos extremos inferior e superior da matriz/vetor, respectivamente.

3.2.3 Uso de Intervalos

O uso de intervalos no C–XSC é facilitado pela existência do tipo de dado intervalo (tipo *interval*). O dado intervalo é estendido, também, para intervalos complexos (*cinterval*), com seus respectivos dados para matrizes e vetores. A atribuição de dados intervalares pode ser realizada via solicitação ao usuário ou utilizando a função *interval()*. Para a manipulação de intervalos também existem funções predefinidas pelo C–XSC. Dentre essas funções pode-se destacar as funções *Inf()*, *Sup()*, *mid()* e *diam()* (extremo inferior de um intervalo, extremo superior de um intervalo, ponto médio de um intervalo e diâmetro de um intervalo, respectivamente). Todas as operações aritméticas básicas $(*, +, -, /, .)$ estão disponíveis para os tipos intervalares. Na figura 3.3 é apresentado um trecho de programa que apresenta algumas operações aritméticas sobre o tipo de dado intervalar.

3.2.4 Arredondamentos Direcionados

Usando o conceito corrente e operadores sobrecarregados $\ll$ e $\gg$ do C++, C–XSC possibilita o controle do tipo de arredondamento e formatação durante a entrada/saída para todos os tipos de dados, mesmo para os tipos de dados *dotprecision* e múltipla precisão. Parâmetros de entrada/saída tais como arredondamentos direcionados, campo de extensão, etc., também usam os operadores sobrecarregáveis de entrada/saída para mani-

```
...
interval a, b, c;

// Entrada de dados do tipo intervalo - via atribuicao
a = interval(-8.0,-2.0); b = interval(-3.0,4.0);

// Uso da funcao ponto medio (mid) de um intervalo
cout << "Ponto medio de a = " << mid(a) << endl;

// Uso da funcao diametro (diam) de um intervalo
cout << "Diametro de a = " << diam(a) << endl;

cout << "Extremo inferior de a = " << Inf(a) << endl;
cout << "Extremo superior de a = " << Sup(a) << endl;

c = a+b; /* adicao (c = a+b) */
c = a*b; /* multiplicacao (c = a*b) */
...
```

Figura 3.3: Uso de intervalos no C–XSC

pular dados de entrada/saída. Se um novo parâmetro de entrada/saída é setado para ser usado, o parâmetro antigo pode ser salvo em uma pilha interna. O valor do novo parâmetro pode então ser definido. Depois do uso das novas "setagens", as últimas podem ser restauradas da pilha.

Tabela 3.1: Funções de arredondamentos direcionados do C–XSC

Função	**Descrição**
addu()	adição com arredondamento para cima
addd()	adição com arredondamento para baixo
subu()	subtração com arredondamento para cima
subd()	subtração com arredondamento para baixo
mulu()	multiplicação com arredondamento para cima
muld()	multiplicação com arredondamento para baixo
divu()	divisão com arredondamento para cima
divd()	divisão com arredondamento para baixo

O C–XSC possui 4 modos de arredondamentos: para o número mais próximo de máquina (função *RndNext* – arredondamento *defaut*), para cima (função *RndUp*), para baixo (função *RndDown*) e sem arredondamento explícito (função *RndNone* - as entradas e saídas são processadas via as funções do C ANSI *scanf()* e *printf()*). Exemplos do uso desses arredondamentos são apresentados nas figuras 3.4 e 3.5.

Além dessas formas e baseado nesses tipos de arredondamentos, o C–XSC ainda disponibiliza funções (apresentadas na tabela 3.1) que permitem a realização de operações com arredondamentos direcionados (para baixo e para cima) diretamente, conforme descrito na figura 3.6.

```
...
real x = 0.1;

cout << "Arred. baixo: " << RndDown << x << endl;
cout << "Arred. proximo: " << RndNext << x << endl;
cout << "Arred. cima: " << RndUp << x << endl;
...
```

Figura 3.4: Arredondamentos Direcionados no C–XSC - exemplo 1

```
...
cout << RndUp;
cout << "a/b (com arred. p/cima)  = " << a/b << endl;

cout << RndDown;
cout << "a/b (com arred. p/baixo)  = " << a/b << endl;
...
```

Figura 3.5: Arredondamentos Direcionados no C–XSC - exemplo 2

3.2.5 A Alta Exatidão no C–XSC

Quando se calcula expressões aritméticas, a exatidão representa um papel decisivo em muitos algoritmos numéricos. Mesmo se todos os operadores aritméticos e funções padrão sejam de máxima exatidão, expressões compostas de vários operadores e funções não entregam necessariamente resultados com máxima exatidão. Por isso, têm sido desenvolvidos métodos para avaliar expressões numéricas com alta exatidão e matematicamente garantidos. Um tipo especial de tais expressões são chamadas de expressões exatas, que são definidas como a soma de expressões simples. Uma expressão simples é uma variável, uma constante ou um produto de dois objetos. As variáveis podem ser do tipo escalar, vetor ou matriz. Somente são permitidos operadores matematicamente relevantes para adição e multiplicação. O resultado de tal expressão é um escalar, um vetor ou uma matriz. Em análise numérica, expressões exatas são de importância decisiva. Por exemplo, métodos de controle de erros ou refinamento iterativo para problemas lineares ou não lineares são baseados em expressões com alta exatidão. O cálculo dessas expressões com máxima exatidão evita cancelamentos. Para obter um cálculo com 1 *ulp* de exatidão, o C–XSC fornece alguns tipos de dados de alta exatidão: *dotprecision*, *cdotprecision*, *idotprecision* e *cidotprecision*. Os resultados intermediários de uma expressão exata podem ser calculados e armazenados em uma variável *dotprecision* sem qualquer erro de arredondamento.

A figura 3.7 apresenta um exemplo do cálculo do produto escalar ótimo programado utilizando as funções da biblioteca C–XSC. Neste exemplo a função *accumulate()* calcula o somatório dos elementos dos vetores A e B e adiciona o resultado para o acumulador `accu` (variável de alta exatidão do C–XSC) sem erro de arredondamento. Ao final, o

```
...
// divisao de a por b com arred. para cima
cout << divu(a,b) << endl;

// divisao de a por b com arred. para baixo
cout << divd(a,b) << endl;
...
```

Figura 3.6: Arredondamentos Direcionados no C–XSC - exemplo 3

```
real result;
dotprecision accu;
rvector A(n), B(n);
...
accu = 0.0; for (int i=1; i<=n; i++)
  accumulate(accu, A[i],B[i]);
result = rnd(accu);
cout << "PEO = " << result << endl;
...
```

Figura 3.7: Produto Escalar Ótimo no C–XSC

acumulador é arredondado, através da função *rnd()*, para a variável `result`, conforme especificado pelo padrão IEEE-754. Com isso, o produto escalar realizado de forma ótima pelo C–XSC possui apenas um único arredondamento durante todo o processo.

3.2.6 A Estrutura de um Acumulador *dotprecision* real

Um acumulador *dotprecision* real ocupa um espaço de armazenamento (conforme figura 3.8) com um tamanho total de *bits* de acordo com (3.1), onde g denota o número de dígitos de guarda (*guard digits*) utilizados para o caso de ocorrência de *overflow* durante a acumulação e l denota o tamanho da mantissa (HAMMER et al., 1995).

g	$2 \times e_{max}$	l	l	$2 \times \lvert e_{min} \rvert$

Figura 3.8: Estrutura de um acumulador *dotprecision* real

$$L = g + 2e_{max} + 2|e_{min}| + 2l \text{ dígitos da base } b \tag{3.1}$$

O tipo de dado *real* do C–XSC corresponde a um número binário em ponto-flutuante no formato duplo (64 *bits* – 8 *bytes*) definidos pelo padrão IEEE-754. Assim, um acumulador *dotprecision* ocupa um espaço de armazenamento de no mínimo (usando $g = 31$) 4227 *bits* (529 *bytes*), conforme (3.2).

$$\begin{aligned} L &= 31 + 2 \times 1023 + 2 \times |-1022| + 2 \times 53 \\ &= 4227 \textit{ bits} \\ &= 529 \textit{ bytes} \\ &= 133 \text{ palavras de memória de } 32 \textit{ bits} \text{ cada} \end{aligned} \tag{3.2}$$

Na implementação do C–XSC, $L = 137$ palavras de memória de 32 *bits* cada, devido que cada acumulador *dotprecision* possui algumas *flags* internas de controle. Os outros tipos de dados *idotprecision*, *cdotprecision* e *cidotprecision*, correspondentes aos tipos escalares básicos *interval*, *complex* e *cinterval* consistem de dois ou quatro acumuladores *dotprecision* de reais, respectivamente.

3.3 Biblioteca de Troca de Mensagens MPI

Existem várias ferramentas de programação para *cluster* de computadores, dentre as quais pode-se destacar a Pthreads (POSIX *threads*), PVM (*Parallel Virtual Machine*), MPI (*Message Passing Interface*) e OpenMP (*Open Multi Processing*). Conforme (COSTA; STRINGHINI; CAVALHEIRO, 2002), PVM e MPI são ferramentas voltadas à exploração de arquiteturas com memória distribuída, já as *threads* exploram a concorrência em arquiteturas dotadas de diversos processadores compartilhando uma área de memória comum. Cada uma dessas ferramentas podem trazer uma ou outra vantagem sobre a outra e a escolha da biblioteca adequada a aplicação em desenvolvimento é uma questão complexa, determinada por aspectos como, por exemplo, o tipo de decomposição de dados que se irá adotar. Maiores detalhes sobre essas ferramentas poderão ser encontrados em (KARNIADAKIS; KIRBY II, 2003), (MPI_FORUM, 1994), (CONTESSA, ????), (COSTA; STRINGHINI; CAVALHEIRO, 2002) e (REBONATTO, 2004).

Para esta pesquisa optou-se pelo uso do MPI, devido as características da maioria dos *clusters* envolvidos nos testes. O MPI é uma das bibliotecas para exploração do paralelismo mais difundidas, tornando-se um padrão para a comunicação paralela com trocas de mensagens em *cluster* de computadores. Ele é largamente utilizado pela disponibilidade em um grande número das máquinas paralelas atuais. Foi definida pelo MPI FORUM com a participação de universidades, empresas e laboratórios de pesquisa. O MPI foi concluído no final do primeiro semestre de 1994 (versão 1.0) e atualizado na metade do ano seguinte (versão 1.1). A versão 2 foi aprovada em 1997. Em MPI, uma execução compreende um ou mais processos que se comunicam chamando rotinas da biblioteca para enviar e receber mensagens. Um programa MPI é formado por um conjunto fixo de processos, criados no momento da inicialização, sendo que é criado um processo por processador. Nesta pesquisa está se utilizando a versão do MPI chamada MPICH, desenvolvida pelo *Argonne National Laboratory da Mississipi State University*. Seu código serve de base para inúmeras outras implementações, disponibilizando versões para plataformas de comunicação e sistemas operacionais únicos, incluindo também versões próprias para famílias de computadores e máquinas específicas.

3.4 A Integração do C–XSC em *Clusters*

Neste item é descrito o processo que foi realizado para possibilitar o uso do C–XSC em *clusters* e a sua integração com a biblioteca de troca de mensagens MPI. Serão apresentados os problemas que surgiram durante a realização deste processo e as soluções

que foram adotadas para solucioná-los. Os resultados desse processo de integração foram publicados em (MORANDI JÚNIOR; HÖLBIG; DIVERIO, 2005; MORANDI JÚNIOR et al., 2004a,b).

3.4.1 Instalação e compilação do C–XSC

Toda a documentação necessária para instalação da biblioteca C–XSC encontra-se no arquivo presente na página `http://www.math.uni-wuppertal.de/org/WRST/index_en.html`. A versão do C–XSC utilizada no trabalho foi a versão 2.0 para PC (arquitetura x86) rodando Debian GNU/Linux 3.0 (*Kernel* 2.4.22) com compilador gcc 3.3.2. Além desse ambiente, o C–XSC pode ser instalado em ambientes computacionais com sistemas operacionais MS-DOS, SunOS e HP-UX, ou seja, o C–XSC é facilmente portável para essas plataformas. A compilação de programas do C–XSC e o seu uso com o MPI são realizados normalmente conforme as regras da linguagem C/C++ e da biblioteca MPI.

3.4.2 Soluções Adotadas para a Integração das Bibliotecas

Durante a primeira fase de desenvolvimento da pesquisa foram constatados alguns problemas durante a compilação da biblioteca C–XSC e no desenvolvimento dos programas iniciais. Também ocorreram problemas com a ambigüidade da função *abs* e a falta de algumas funções que estavam presentes no Pascal–XSC ((KLATTE et al., 1991; HÖHER; HÖLBIG; DIVERIO, 1997)) e que não existem no C–XSC. Além disso, durante a fase de integração do C–XSC com a biblioteca MPI, surgiram dois problemas:

- A compilação com MPI não funcionava, o compilador gerava um erro de redefinição de constantes da biblioteca do *MPICH*;

- O envio e recebimento dos tipos do C–XSC através das primitivas de comunicação do MPI (**MPI_Send** e do **MPI_Recv**, respectivamente) não funcionavam, uma vez que essa biblioteca não tem suporte para os tipos primitivos do C–XSC como *dotprecison* , *rmatrix*, *ivector*, etc...

Nas próximas seções será descrito como foram resolvidos cada um dos problemas que surgiram durante este processo de integração.

3.4.2.1 Erros de ambigüidade no overloading da função abs

A função *abs* quando usada para retornar o valor absoluto do índice de uma matriz, devido as suas várias redefinições feitas pelo C–XSC, faz com que o compilador *gcc* não consiga decidir sobre qual função utilizar e, com isso, retornava um erro de ambigüidade, conforme figura 3.9. Para resolver esse problema da ambigüidade, utilizou-se a sintaxe apresentada em (3.3).

$$x = \mathbf{cxsc} :: \mathbf{abs}(j); \tag{3.3}$$

Com essa estrutura deixa-se claro para o compilador qual é a função *abs()* que ele deve utilizar. Nesse caso, é mostrado ao compilador que ele deve utilizar a redefinição da função feita pelo C–XSC. Isso em C++ é o que se chama de *namespace*, onde se diz a qual classe esse método pertence, no caso, o método é a função *abs(int)* e a classe é *cxsc*.

```
band.cpp:  In function 'void lss_triangular(
cxsc::rmatrix&, cxsc::ivector&, cxsc::ivector&)':
band.cpp:373:  call of overloaded 'abs(int)' is
ambiguous
/usr/include/stdlib.h:699:  candidates are:  int
abs(int)
/usr/include/c++/3.2/cmath:95:  long double
std::abs(long double)
/usr/include/c++/3.2/cmath:91:  float std::abs(float)
/usr/include/c++/3.2/cmath:87:  double std::abs(double)
/usr/include/c++/3.2/cstdlib:142:
 long long int __gnu_cxx::abs(long long int)
/usr/include/c++/3.2/cstdlib:119:  long int
std::abs(long int)
/opt/cxsc/include/intvecto.hpp:38:  int cxsc::abs(int)
/opt/cxsc/include/real.inl:70:
 cxsc::real cxsc::abs(const cxsc::real&)
```

Figura 3.9: Erros de ambigüidade gerado pela função *abs()*

3.4.2.2 Adaptação das funções no C–XSC

Muitas vezes durante as transcrições dos algoritmos em Pascal-XSC para C–XSC presentes no *Toolbox II* (KRÄMER; KULISCH; LOHNER, 1996) verificava-se que algumas funções e expressões não existiam no C–XSC. A seguir relata-se alguns desses casos.

- **Expressões iniciadas com # (expressões exatas)**: Em Pascal-XSC, as expressões exatas são as operações delimitadas por **#** e executadas sem arredondamento, utilizando registradores especiais (*dotprecision*). Após efetivado o cálculo sem arredondamento, o resultado é passado para uma variável do tipo *real* ou do tipo *interval*. Por exemplo, na expressão (3.4) (onde b e x_0 são vetores e A é uma matriz), x_1 será o vetor mais próximo do resultado da expressão exata, ou seja, é o resultado arredondado da expressão exata. Os arredondamentos possíveis são:
  - Para valores do tipo *real*:
    - Arredondamento para o real mais próximo do resultado da expressão exata (**#***);
    - Arredondamento para o menor real mais próximo do resultado da expressão exata (**#**<);
    - Arredondamento para o maior real mais próximo do resultado da expressão exata (**#**>).
  - Para valores do tipo *interval*:
    - Arredondamento para o *intervalo* de menor diâmetro que contenha o resultado da expressão exata **##**.

$$x_1 := \# * (b - A * x_0) \tag{3.4}$$

Para usar expressões equivalentes em C–XSC tem que se entender exatamente o que acontece com as expressões exatas. Como não são utilizados arredondamentos, é

fácil supor que se deve usar o tipo *dotprecision* para calcular o valor sem arredondá-lo. Essa parte é facilmente aplicada, tomando como exemplo a expressão exata (3.4). Para calcular $(b - A * x_0)$ em C–XSC basta proceder como apresentado na figura 3.10.

```
for(i=Lb(x0);i <= Ub(x0); i++)
{
  Accu = b[i];
  accumulate(Accu,-A[Row(i)],x0[i]);
}
```

Figura 3.10: Cálculo de $(b - A * x_0)$ usando o C–XSC

A função $A[Row(i)]$ devolve a *i-ésima* linha da matriz A. Também pode-se usar, alternativamente, $Row(A, i)$. As funções *Lb()* e *Ub()* retornam o índice inferior e o índice superior, respectivamente, do vetor x_0, no caso. Caso as matrizes sejam o alvo desse procedimento, utiliza-se $Lb(A, ROW)$ para obtenção do índice inferior da linha (*ROW* é uma constante do C–XSC) e $Lb(A, COL)$ se o que se deseja é o índice inferior da coluna. Esse procedimento pode ser estendido analogamente caso deseja-se obter os índices superiores, bastando para isso utilizar a função *Ub()*. A função *accumulate(dot,x,y)* é equivalente a $dot = dot + x * y$, porém, sem fazer arredondamentos.

Agora tem-se o problema de como devolver a resposta arredondada para cada um dos tipos de arredondamentos citados anteriormente. Consultando a bibliografia do C–XSC, descobriu-se as correspondências entre a função *rnd* e os tipos de arredondamento citados. Considerando que var é a variável que irá armazenar o resultado arredondado e que $Accu$ possui o valor exato calculado, ou seja, é uma variável do tipo *dotprecision*, procede-se conforme apresentado nas figuras 3.11, 3.12, 3.13 e 3.14.

• **em Pascal-XSC**

```
x_1 := #*(b - A*x_0);
```

• **em C–XSC**

```
for(i=Lb(x0);i <= Ub(x0); i++)
{
  Accu = b[i];
  accumulate(Accu,-A[Row(i)],x0[i]);
}
x1 = rnd(Accu);
```

Figura 3.11: Arredondamento para o número mais próximo de máquina (Pascal-XSC × C–XSC)

• **em Pascal-XSC**

```
x_1 := #<(b - A*x_0);
```

• **em C–XSC**

```
for(i=Lb(x0);i <= Ub(x0); i++)
{
  Accu = b[i];
  accumulate(Accu,-A[Row(i)],x0[i]);
}
x1 = rnd(Accu,RND_DOWN);
```

Figura 3.12: Arredondamento para baixo (Pascal-XSC × C–XSC)

• **em Pascal-XSC**

```
x_1 := #>(b - A*x_0);
```

• **em C–XSC**

```
for(i=Lb(x0);i <= Ub(x0); i++)
{
  Accu = b[i];
  accumulate(Accu,-A[Row(i)],x0[i]);
}
x1 = rnd(Accu,RND_UP);
```

Figura 3.13: Arredondamento para cima (Pascal-XSC × C–XSC)

• **em Pascal-XSC**

```
x_1 := ##(b - A*x_0);
```

• **em C–XSC**

```
// a variavel Accu eh do tipo idotprecision
// o vetor x1 eh do tipo ivector
for(i=Lb(x0);i <= Ub(x0); i++)
{
  Accu = b[i];
  accumulate(Accu,-A[Row(i)],x0[i]);
}
rnd(Accu,x1);
```

Figura 3.14: Arredondamento para dados intervalares (Pascal-XSC × C–XSC)

- **Diferentes funções identidade e transposta:** Infelizmente o C–XSC não possui funções diferentes para calcular a identidade e a transposta de uma matriz. Em Pascal-XSC existem funções para todos os tipos de matrizes, mas para C–XSC só existe uma função definida, conforme apresentado em (3.5).

$$rmatrix \;\; Id(rmatrix\& \;\; A) \tag{3.5}$$

A função definida em (3.5) devolve uma matriz identidade de reais (*rmatrix*). Se for necessário realizar o cálculo da identidade de uma matriz intervalar (*imatrix*), por exemplo, isso deverá ser realizado conforme demonstrado em (3.6), sendo A uma *imatrix* definida como $A(1, m, 1, n)$.

$$\begin{array}{l} for(i = 1; i <= m; i++) \\ \quad for(j = 1; j <= n; j++) \\ \quad (j == i)?A[i][j] = interval(1) : A[i][j] = interval(0); \end{array} \tag{3.6}$$

Esse procedimento funciona também para *cmatrix* e *cimatrix*, bastando para isso substituir o *type casting* da expressão (3.6) por *complex(i)* e *cinterval(i)* respectivamente, substituindo o *i* por 1 ou 0 (zero).

Para o caso da matriz transposta há um problema parecido. O C–XSC também só possui a função *rmatrix transp(rmatrix A)*, o que fica novamente restrito a matrizes reais. No programa *clss.cpp* (programa integrante do *solver* LSS – seção 4.1), implementado em C–XSC e referente a resolução de sistemas de equações lineares para matrizes densas e com dados de entrada do tipo complexo, por exemplo, era necessário o cálculo de uma matriz transposta de matrizes complexas. Então, pesquisando sobre transpostas de matrizes complexas e utilizando as funções do Pascal-XSC para fazer comparações, criou-se a função apresentada na figura 3.15. Para matrizes complexas intervalares basta trocar o tipo *cmatrix* por *cimatrix*.

```
cmatrix Herm( cmatrix &A )
{
 int i;
 cmatrix herm(Lb(A,COL),Ub(A,COL),Lb(A,ROW),Ub(A,ROW));

 for(i=Lb(A,ROW); i<=Ub(A,ROW); i++)
  Col(herm,i) = Row(A,i);

 return SetIm(herm, -1*Im(herm) );
}
```

Figura 3.15: Cálculo da matriz transposta usando o C–XSC

- **Equivalência entre as funções *accumulate* e *sum*:** Durante as comparações entre as funções do Pascal–XSC e do C–XSC, observou-se uma outra função que possui

praticamente um mapeamento direto para C–XSC, a função que realiza o somatório do produto em alta exatidão. Entretanto, é necessário compreender o que realmente faz essa função. A função em questão é a $sum(x * y)$, onde x e y podem ser escalares, vetores ou matrizes. O que essa função faz é calcular o somatório do produto de $x * y$. Por exemplo, em Pascal-XSC essa função é definida como apresentada em (3.7).

$$x := \# * (for \;\; j = k \;\; to \;\; n \;\; sum(b[j] * A[i][j])); \tag{3.7}$$

Ou seja, o objetivo é calcular o valor exato do somatório do produto do vetor *b* pela matriz *A*. Equivalentemente, em C–XSC, essa função é definida como apresentada em (3.8).

$$\begin{aligned} &for(j = k; j <= n; j++) \\ &\quad accumulate(Accu, b[j], A[j][i]); \\ &x = rnd(Accu); \end{aligned} \tag{3.8}$$

Depois, obviamente, deve-se arredondar o resultado de *Accu* de acordo com o padrão IEEE–754 através do uso da função *rnd()*.

- **Expressões matemáticas com sinal de maior e menor:** Outras funções também bastante comuns nos códigos em Pascal-XSC são as de arredondamento específico das operações básicas de soma, divisão, subtração e multiplicação. Em Pascal-XSC, o cálculo de uma soma arredondada para cima seguida de uma subtração arredondada para baixo é feita conforme (3.9).

$$x := (y \;\; +> \;\; z) \;\; -< \;\; w; \tag{3.9}$$

Esse tipo de construção é importante pois, para implementar a matemática intervalar, os limites inferiores devem ser arredondados para baixo e os limites superiores para cima. Então fica claro que há a necessidade desse tipo de operação no C–XSC. Porém, devido a limitações da própria linguagem, não existem tais operadores, o que existe, na verdade, são funções que se equivalem a tal. Utilizando o exemplo descrito em 3.9, pode-se alterá-lo para C–XSC conforme descrito em (3.10).

$$x = subd(addu(y, z), w); \tag{3.10}$$

Em (3.10) *addu* significa "*add upwards*" (soma e arredonda para cima) e *subd* significa "*subtract downards*" (subtrai e arredonda para baixo). Equivalentemente, pode-se escrever multiplicação e divisão arredondando para cima (*mulu*, *divu*) ou para baixo (*muld*, *divd*).

Percebe-se que esse tipo de notação fica afastada da linguagem natural (matemática), dificultando ao programador o entendimento claro dessas operações. Trabalhos no sentido de melhoria dessa notação vem sendo desenvolvido dentro do grupo de pesquisa. Nas próximas seções serão abordados os problemas relacionados a segunda parte do processo de uso do C–XSC no *cluster* labtec, o qual consistiu em verificar a viabilidade da integração das bibliotecas C–XSC e MPICH e o desenvolvimento de programas paralelos utilizando os tipos de dados especiais do C–XSC.

3.4.3 Erro ocorrido durante a compilação

No momento da compilação da biblioteca C–XSC integrada a biblioteca MPI, utilizando o compilador *mpiCC*, foram observados alguns erros, conforme apresentado na figura 3.16.

```
 In file included from
/usr/local/mpich/include/mpi2c++/mpi++.h:115, from
/usr/local/mpich/include/mpi.h:677,
 from mpicxscpeo_1_var.cpp:4:
/usr/local/mpich/include/mpi2c++/constants.h:59:  parse
error before '-'
```

Figura 3.16: Erros na compilação do C–XSC com o MPI

Para esse problema, a solução encontrada foi bastante simples. Para a sua solução basta inverter as posições dos *includes*. O *include* do MPI (*#include <mpi.h>*) deve vir antes dos *includes* do C–XSC (*#include <imatrix.hpp>*, *#include <ivector.hpp>*, etc...). Este problema ocorre porque, provavelmente, o C–XSC redefine alguma constante utilizada pelo MPICH.

3.4.4 Envio e recebimentos dos tipos especiais do C–XSC através do MPI

Para que a continuação da pesquisa fosse possível era necessário uma completa integração do C–XSC com o MPI (MORANDI JÚNIOR et al., 2003a; HÖLBIG et al., 2004), pois, caso os algoritmos exigissem que um tipo específico seja enviado para outro processo, este deveria usar uma primitiva do MPI (como *MPI_Send*, por exemplo). Porém, nativamente, o MPI possui suporte apenas para os tipos do C/C++, como *MPI_DOUBLE*, *MPI_FLOAT*, *MPI_INT*,.... Então, o próximo passo foi encontrar uma maneira de dizer para o MPI como ele deveria enviar os tipos de dados do C–XSC (*real*, *interval*, *complex*, *rmatrix*, *rvector*, *intvector*, *intmatrix*, *civector*, *cimatrix*, *cinterval*). Pesquisando um pouco sobre os tipos do MPI, descobriu-se que este pode enviar também uma seqüência de bytes com o tipo *MPI_BYTE*, logo é possível, então, enviar esses dados conforme (3.11).

$$MPI\_Send(\&(\textbf{var}), sizeof(\textbf{var}), MPI\_BYTE, dest, tag, comm); \tag{3.11}$$

Em (3.11), **var** é qualquer uma das variáveis declaradas com um dos tipos do C–XSC. O primeiro parâmetro da primitiva é o endereço onde se encontra a variável que se pretende enviar, o segundo parâmetro corresponde ao tamanho da variável que se pretende enviar. A função *sizeof* devolve o tamanho da variável que se recebe como parâmetro. Os argumentos *dest*, *tag* e *comm*, correspondem, respectivamente, ao indentificador do destino da mensagem, o identificador da mensagem e o comunicador a ser utilizado.

Para os tipos *dotprecision*, *idotprecision*, *cdotprecision* e *cidotprecision* esse teste não funcionou. Durante a fase de depuração do código, observou-se que o receptor da mensagem havia recebido apenas 4 *bytes* do remetente. Como as variáveis *dotprecision* devem armazenar o resultado sem arredondamento, é de se esperar que estas tenham um tamanho considerável em memória. Consultando (HOFSCHUSTER; KRÄMER, 2001) descobriu-se que na verdade o *dotprecision* ocupa mais de 529 *bytes* em memória (ver seção 3.2.6).

Portanto, o que o *MPI_Send* estava fazendo era enviando um ponteiro de uma área de memória para o receptor. Com isso, pôde-se concluir que o tipo *dotprecision* é um ponteiro para a memória e que o MPI não consegue acessar o conteúdo do dado *dotprecision*. Caso fosse possível existir uma variável de tipo conhecida apontar diretamente para o conteúdo do *dotprecision*, provavelmente o MPI não teria dificuldades de enviar o conteúdo do *dotprecision*.

Outra questão que deveria ser respondida era determinar a quantidade de *bytes* que deveriam ser enviados, pois o livro base sobre o C–XSC (HAMMER et al., 1995) não especifica com precisão o tamanho do dado *dotprecision*. Utilizando uma ferramenta de depuração paralela (PADI - (STRINGHINI, 2002)) desenvolvida no GPPD (Grupo de Processamento Paralelo e Distribuído) da UFRGS, descobriu-se que, na verdade, o *dotprecision* era um ponteiro do tipo *unsigned long*. Portanto, o *type casting* (troca explícita de tipos) resolve o problema de acessar o conteúdo da variável, conforme (3.12).

$$\begin{aligned} &unsigned\ \ long\ \ *envia; \\ &dotprecision\ \ accu; \\ &envia\ \ =\ \ *(unsigned\ \ long\ \ **)(\&accu); \end{aligned} \tag{3.12}$$

Agora *envia* aponta para o mesmo conteúdo que *accu*. Na nova chamada da primitiva do MPI (*MPI_Send*) em (3.11) basta trocar *&(var)* por *envia*. Entretanto continuava a questão do tamanho da seqüência de bytes. Para resolver essa questão recorreu-se ao código fonte do C–XSC e descobriu-se que para alocar um *dotprecision* na memória, o C–XSC utiliza uma constante chamada **BUFFERSIZE** que possui 556 *bytes* logo, a nova chamada para o *MPI_Send* pode ser reescrita conforme (3.13).

$$MPI\_Send(\textbf{envia}, \textbf{BUFFERSIZE}, MPI\_BYTE, dest, tag, comm); \tag{3.13}$$

Como o recebimento é análogo ao envio, o mesmo caminho foi feito para o recebimento. Para os tipos normais do C–XSC procede-se conforme (3.14).

$$MPI\_Recv(\textbf{envia}, \textbf{BUFFERSIZE}, MPI\_BYTE, dest, tag, comm); \tag{3.14}$$

Para os tipos *dotprecision*, faz-se o mesmo *type casting* antes do recebimento, conforme (3.15).

$$\begin{aligned} &unsigned\ \ long\ \ *recebe; \\ &dotprecision\ \ accu2; \\ &recebe\ \ =\ \ *(unsigned\ \ long\ \ **)(\&accu2); \\ &MPI\_Recv(\textbf{recebe}, \textbf{BUFFERSIZE}, MPI\_BYTE, org, tag, comm, status); \end{aligned} \tag{3.15}$$

Durante a realização dos testes constatou-se que, caso haja a necessidade de enviar um pedaço (tam_ped) de um vetor ou matriz de uma só vez, é necessário utilizar *&(***var[0]***)* e *sizeof(***var[0]***)*tam_ped* para o correto funcionamento do *MPI_Send* e do *MPI_Recv* nesse caso particular. Além disso, caso esteja-se enviando/recebendo variáveis *dotprecisions* do tipo intervalo ao invés de *dotprecisions* do tipo real (conforme chamadas descritas em (3.14) e (3.15)), deve-se trocar a variável **BUFFERSIZE** por **2*BUFFERSIZE** visto que

esse tipo de dado, por conter intervalos, tem o dobro do tamanho de um dado *dotprecision* real. Os parâmetros *org* e *status* correspondem, respectivamente, ao identificador do remetente e ao estado da mensagem (maiores detalhes veja (KARNIADAKIS; KIRBY II, 2003; MPI_FORUM, 1994)).

3.4.4.1 Testes utilizados para envio do dotprecision

Para validar o envio dos tipos *rmatrix*, *imatrix*, *cmatrix*, *cimatrix*, *intmatrix*, *intvector*, *rvector*, *ivector*, *cvector*, *civector*, *real*, *interval*, *complex* e *cinterval*, utilizou-se um teste bastante simples, que consistia em enviar vetores, matrizes ou simplesmente um número (conforme cada caso) de um processo para outro. Assim, necessariamente, os dados do C–XSC trafegaram na rede através das diretivas do MPI. Em todos os casos, o destinatário recebeu os dados corretamente.

Nos testes para o tipo de dado *dotprecision* foi utilizado o modelo de comunicação mestre-escravo, onde o mestre inicializa e distribui os dados, e os escravos recebem as partes, fazem o cálculo parcial e devolvem o resultado para o mestre que, por sua vez, encarrega-se de juntar os resultados parciais. No caso, utilizou-se o cálculo de um produto escalar ótimo, onde ótimo significa que o resultado final difere do exato por apenas um arredondamento (alta exatidão). O processo 0 (mestre) inicializa os vetores de dimensão 180000, divide igualmente (60000) entre três processos (1 até 3). Cada processo, de 1 até 3, estará executando o código apresentado na figura 3.17 para calcular o produto escalar ótimo parcial.

```
accu = 0.0; for(i=Lb(a);i<=Ub(a);i++)
     accumulate(accu,a[i],b[i]);

envia = *(unsigned long **)(&accu);
MPI_Send(envia,BUFFERSIZE,MPI_BYTE,0,0,MPI_COMM_WORLD);
```

Figura 3.17: Cálculo parcial do produto escalar ótimo em C–XSC com MPI

```
totalAccu = 0; for(i=1;i<size;i++) {
 recv = *(unsigned long **)(&accu);

 MPI_Recv(recv,BUFFERSIZE,MPI_BYTE,i,0,
                        MPI_COMM_WORLD,&status);
 totalAccu += accu;
} scalarProduct = rnd( totalAccu );
```

Figura 3.18: Cálculo final do produto escalar ótimo em C–XSC com MPI

O processo 0 (mestre) recebe de cada processo escravo o cálculo do produto parcial (armazenado em *accu*), e executa o trecho de código apresentado na figura 3.18. Através desse processo, o mestre soma os produtos parciais (sem arredondamento) e arredonda no

final. Com isso é possível obter um resultado diferente do exato por apenas um arredondamento.

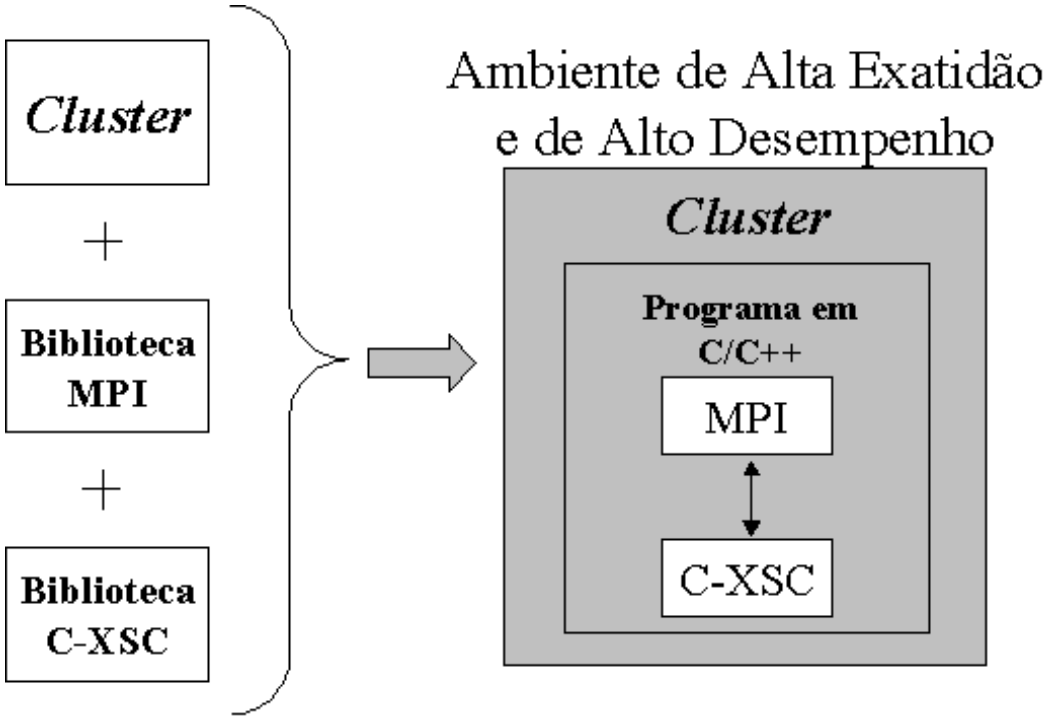

Figura 3.19: Ambiente de Alta Exatidão e de Alto Desempenho

3.5 Ambiente de Alto Desempenho e Alta Exatidão

Conforme apresentado nas seções anteriores deste capítulo, foi possível, através de uma série de atividades e testes, providenciar o uso da biblioteca C–XSC em ambientes do tipo de *clusters* de computadores e, consequentemente, realizar de maneira correta e efetiva a sua integração com a biblioteca de troca de mensagens MPI. Com isso, nessa tese de doutorado, conseguiu-se disponibilizar uma ferramenta computacional de alta exatidão (C–XSC) em um ambiente de *clusters* de computadores, conforme ilustrado pela figura 3.19. É importante ressaltar que, além da característica da alta exatidão que é o foco principal deste trabalho, o ambiente disponibilizado oferece a comunidade científica todas as características que possibilitam que se implemente os mais diversos métodos numéricos computacionais com o paradigma da Computação Verificada. Com isso, no desenvolvimento de programas paralelos, pode-se enviar corretamente entre o processadores os dados especiais da biblioteca C–XSC possibilitando, com isso, a elaboração desses programas paralelos com o paradigma da Computação Verificada. Todo esse processo de uso e integração do C–XSC em *clusters* foi validado através do uso dos *clusters* do II-UFRGS (*cluster* labtec), da Universidade de Passo Fundo (*cluster* Colorado) e do *cluster* da Universidade de Wuppertal (*cluster* ALiCE) onde, basicamente, os mesmos problemas de uso e integração foram encontrados e as mesmas soluções apresentadas nesse capítulo foram aplicadas de maneira satisfatória. O capítulo 6 apresenta em detalhes esses testes e os resultados que foram obtidos, permitindo assim validar todo o processo de uso e integração.

4 *SOLVERS* VERIFICADOS PARA SISTEMAS LINEARES

Após a disponibilização do ambiente de alta exatidão com a biblioteca C–XSC, foram implementados *solvers* verificados para a resolução de sistemas lineares com matrizes densas e bandas (*solver LSS* e *solver BAND*). Os *solvers* desenvolvidos implementaram métodos pontuais e intervalares para a resolução de sistemas lineares e as suas versões iniciais possuem apenas versões seqüenciais dos programas. Esses *solvers* originaram-se do Toolbox II (KRÄMER; KULISCH; LOHNER, 1994) e serviram, principalmente, para avaliar (com resultados satisfatórios) a boa exatidão fornecida pela biblioteca C–XSC no tratamento de problemas sujeitos a instabilidade numérica e a erros de arredondamento e cancelamento. Uma descrição completa destes *solvers* poderá ser encontrada em (HÖLBIG; KRÄMER, 2003) e (HÖLBIG; KRÄMER, 2006).

4.1 *Solver* para Sistemas Lineares Densos

O objetivo do *solver LSS* e de suas extensões[1] é de verificar a existência de uma solução de sistemas lineares com matrizes densas e computá-la para cada um dos seguintes problemas (os algoritmos para a solução desses problemas são descritos na seção 4.1.1):

- **Problema s –**] Sistema de equações lineares quadrado ($m \times n$, com $m = n$);
- **Problema o –**] Sistema de equações lineares sobre-determinado ($m \times n$, com $m > n$);
- **Problema u –**] Sistema de equações lineares sub-determinado ($m \times n$, com $m < n$);
- **Problema S –**] Inversa da matriz de coeficientes do sistema de equações lineares quadrado;
- **Problema O –**] Pseudo-inversa da matriz de coeficientes do sistema de equações sobre-determinado;
- **Problema U –**] Pseudo-inversa da matriz de coeficientes do sistema de equações sub-determinado.

Defini-se (4.1) como um sistema linear, onde A é a matriz de coeficientes do sistema, b o vetor de termos independentes e x o vetor que verifica a igualdade. Se (4.1) for um sistema linear quadrado e supondo-se conhecer uma aproximação $\tilde{x}$ da solução do sistema

[1] *solver LSS* – para dados de entrada do tipo real, *solver ILSS* – para dados de entrada do tipo intervalo, *solver CLSS* – para dados de entrada com números complexos e *solver CILSS* – para dados de entrada com números complexos intervalares

linear e uma aproximação R da matriz inversa de A, pode-se computar uma inclusão para $\tilde{x}$ através da minimização do erro, dado por (4.2), o que proporciona muito mais exatidão do que tentar aproximar a solução através de iterações diretamente em $\tilde{x}$.

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

$$Ay = b - A\tilde{x} \tag{4.2}$$

Multiplicando (4.2) por R, obtém-se (4.3) ou (4.4), ou seja, uma equação de ponto fixo para o erro y.

$$y = R(b - A\tilde{x}) + (I - RA)y \tag{4.3}$$

$$y = f(y) := R(b - A\tilde{x}) + (I - RA)y \tag{4.4}$$

Se R for uma aproximação suficientemente boa de A^{-1}, a iteração baseada em (4.4) deverá convergir, contanto que $I - RA$ tenha um raio espectral pequeno. Com base em (4.4), deriva-se a iteração (4.5) ou (4.6), na qual já se utiliza aritmética intervalar, com o intervalo $[y]_k$ para y, onde F é a extensão intervalar de f.

$$[y]_{k+1} = R\diamondsuit(b - A\tilde{x}) + \diamondsuit(I - RA)\,[y]_k \tag{4.5}$$

$$[y]_{k+1} = F\left([y]_k\right) \tag{4.6}$$

Aqui $\diamondsuit$ indica que as operações devem ser executadas exatamente, com o resultado sendo arredondado para um intervalo de inclusão somente após o término da computação de cada produto escalar. Como no cálculo do defeito $b - A\tilde{x}$ e da matriz iterada $I - RA$ pode ocorrer o cancelamento dos últimos dígitos de máquina, deve-se, para estes casos, utilizar o produto escalar ótimo. Com $z = R\diamondsuit(b - A\tilde{x})$ e $C = \diamondsuit(I - RA)$, pode-se reescrever (4.5) como (4.7).

$$[y]_{k+1} = z + C\,[y]_k\,. \tag{4.7}$$

A fim de garantir a existência de $\tilde{x}$ e, portanto, de x, utiliza-se o Teorema do Ponto Fixo de Brower, que se aplicará logo que houver uma iteração $k+1$ a qual possui a propriedade de inclusão apresentada em (4.8), onde $[y]_k^\circ$ é o interior de $[y]_k$.

$$[y]_{k+1} = F\left([y]_k\right) \subset [y]_k^\circ \tag{4.8}$$

Se o teste de inclusão (4.8) for satisfeito, então a função iterativa f mapeia $[y]_k$ dentro dela mesma e, pelo Teorema de Brower, segue que f possui um ponto fixo y^* em $[y]_k$ e, portanto, em $[y]_{k+1}$. O pré-requisito de que $[y]_k$ esteja mapeado dentro de seu próprio interior garante, ainda, a unicidade do ponto fixo, isto é, (4.2) tem uma única solução y^* e, assim, (4.8) tem uma única solução $x^* = \tilde{x} + y^*$.

Observação. O teste de inclusão (4.8) será satisfeito se e somente se o raio espectral de C (e até mesmo o de $|C|$, a matriz dos valores absolutos de C) for menor do que 1, o que assegura a convergência da iteração (também no caso intervalar). Além disso, esse fato implica também na não-singularidade de R e de A e, logo, na unicidade do ponto fixo. O raio espectral de uma matriz $C_{n\times n}$ é $\rho\left(C\right) = \max\limits_{1\leq i\leq n}|\lambda_i|$, onde λ_1, λ_2,. . .,λ_n são os autovalores de C. No entanto, não é possível dizer se será possível alcançar a condição

(4.8). Por exemplo, quando $n = 1$ com $a_{11} = 0.1$, $b = 0$ e $\tilde{x} = 1$, a iteração convergirá para a solução única $x^* = 0$, mas o fará decrescendo monotonicamente. Dessa forma, (4.8) nunca será satisfeita. Para forçar a inclusão (4.8) é introduzida a $\epsilon - inflation$, que provoca a inflação do intervalo a fim de alcançar o ponto fixo mais próximo. Para um intervalo real $[w]$, é definido ϵ-*inflation* como uma notação funcional $\mathrm{blow}([w], \epsilon)$ conforme 4.9.

$$\mathrm{blow}([w], \epsilon) := \begin{cases} (1+\epsilon)[w] - \epsilon[w] & \text{, se } \mathrm{diam}([w]) > 0, \\ [\mathrm{pred}(w), \mathrm{succ}(w)] & \text{, se } \mathrm{diam}([w]) = 0, \end{cases} \tag{4.9}$$

Em 4.9, $\mathrm{diam}([w])$ é o diâmetro de um intervalo ($\mathrm{diam}([w]) = \overline{w} - \underline{w}$) e, no caso do $\mathrm{diam}([w]) = 0$, $[w] = w$ é definido como sendo um número em ponto-flutuane e $\mathrm{pred}(w)$ e $\mathrm{succ}(w)$ são seus antecessor e sucessor em ponto-flutuante. Da mesma maneira, usa-se ϵ-inflation $\mathrm{blow}(\cdot)$ também para vetores e matrizes intervalares.

Até então nada foi dito sobre a computação da aproximação da solução $\tilde{x}$ e da aproximação da inversa R. Em princípio, nenhuma precaução precisa ser tomada quanto à natureza desses valores. No entanto, quanto mais próximos das soluções reais forem os valores sugeridos, melhor será a resposta gerada pelo algoritmo.

Inicialmente, não é adotado nenhum algoritmo especial para o cálculo da solução aproximada de $R = A^{-1}$, a partir da qual fica claro que uma aproximação para a solução é $\tilde{x} = Rb$. Para que a iteração intervalar convirja rapidamente, a aproximação $\tilde{x}$ é melhorada através da correção do resíduo em (4.10), usando apenas aritmética de ponto flutuante e produto escalar ótimo para o resíduo $b - A\tilde{x}_k$.

$$\tilde{x}_{k+1} = \tilde{x}_k + R\left(b - A\tilde{x}_k\right) \tag{4.10}$$

Melhorar a aproximação no princípio é interessante no sentido de evitar a execução de várias iterações intervalares mais tarde. A matriz R é obtida através do método de Gauss-Jordan com pivotamento por coluna. Para tornar o algoritmo mais eficiente no tratamento da maior parte das matrizes singulares, foram implementadas modificações no passo de eliminação, o qual pode produzir zeros exatos gerados pelos erros de arredondamento. Assim, quando, durante o cálculo, os pivôs são procurados, pode ocorrer que nenhum pivô seja encontrado em uma linha, se todos os seus elementos forem zeros de máquina. Por isso, os zeros exatos produzidos pela eliminação são substituídos por $a\epsilon$, ao invés de $a - a = 0$, forçando a execução do algoritmo até o final para qualquer matriz não-singular (e para quase toda singular também). Adicionalmente, se nenhum pivô for encontrado em uma linha então as entradas originais para essa linha eram todas zero. Para as matrizes muito mal-condicionadas, no entanto, a qualidade obtida para R não será suficiente, ou seja, seu raio espectral nunca será menor do que 1 o que implica na não-convergência do teste de inclusão. Quando isso ocorrer, o algoritmo recorre a um método, aqui chamado de *Dispositivo de Rump*, para obter uma aproximação para R com maior exatidão. Seja R uma aproximação da inversa da matriz A. Assim, até mesmo se A for muito mal-condicionada, a matriz RA é, normalmente, muito melhor condicionada do que A. A relação em (4.11) sugere que seja calculado outra aproximação S da inversa de RA, tal que o produto SR seja uma melhor aproximação A^{-1}.

$$A^{-1} = (RA)^{-1} R \tag{4.11}$$

Resumindo, pode-se calcular uma aproximação da inversa de $R_1 + R_2$ em precisão dupla (armazenada em duas matrizes reais em ponto-flutuante $R_1 and R_2$) através dos seguintes passos:

1. Cálculo da aproximação da inversa R de A utilizando o método de Gauss-Jordan.
2. Cálculo de RA.
3. Cálculo da aproximação da inversa S de RA através de Gauss-Jordan.
4. Cálculo de SR utilizando um acumulador longo e armazenando o resultado como a soma de duas matrizes em ponto-flutuante R_1 e R_2.

Agora têm-se uma aproximação da inversa $R := R_1+R_2$ em precisão dupla, conforme descrito acima. Com isso, a estratégia utilizada no método implementado no *solver* é:

- Calcular uma aproximação da inversa R em precisão simples e executar o algoritmo de inclusão. Se ele falhar então:
- Calcular a aproximação da inversa by mecanismo de Rump e execute o algoritmo de inclusão com precisão dupla para o cálculo da aproximação da inversa $R = R_1+R_2$.

Agora, seja X uma matriz. Se (4.12), então $X = R$ é exatamente a inversa de A.

$$AX = I \tag{4.12}$$

Sendo assim, se $X = \left[\begin{array}{cccc} x_1 & x_2 & \dots & x_n \end{array} \right]$ e $I = \left[\begin{array}{cccc} \epsilon_1 & \epsilon_2 & \dots & \epsilon_n \end{array} \right]$, pode-se reescrever (4.12) como o sistema (4.13) e resolvê-lo equação por equação, ou seja, obtendo-se uma coluna de X por vez, pelo método já apresentado anteriormente. Como o resultado procurado é obtido através da melhoria de $x_i = R\epsilon_i$ e a matriz de coeficientes é a mesma para cada uma das equações de (4.13), não é necessário recalcular uma nova aproximação para R e nem para a matriz de iteração $C = \diamondsuit (I - RA)$, o que poupa o computador de ônus redundante e, portanto, desnecessário. A comunicação entre as rotinas é estabelecida através de sinalizadores (*flags*) que indicam o que deve ou não ser calculado.

$$\left\{ \begin{array}{rcl} Ax_1 & = & \epsilon_1 \\ Ax_2 & = & \epsilon_2 \\ & \vdots & \\ Ax_n & = & \epsilon_n \end{array} \right. \tag{4.13}$$

Uma maneira muito difundida para a solução de sistemas lineares sobredeterminados é o chamado método dos mínimos quadrados, o qual consiste na solução das equações normais

$$A^H Ax = A^H b \tag{4.14}$$

onde A^H é a matriz hermitiana - transposta, no caso dos reais - de uma matriz A. Se A tem posto máximo (i.e. todas as colunas de A são linearmente independentes), a solução dessas equações é única e bem determinada e é aquela que minimiza a norma euclidiana do vetor residual $r = b - Ax$, uma vez que sistemas sobredeterminados são, na maior parte das vezes, impossíveis. No entanto, é de se esperar que a matriz $A^H A$ seja bastante mal-condicionada, isso sem levar em consideração que, no computador, ela só poderia ser obtida com erros de arredondamento ou na forma intervalar, o que torna essa abordagem pouco satisfatória. Ao invés, o sistema (4.14) é reescrito como um sistema quadrado maior $(n + m) \times (n + m)$, que pode ser resolvido com muito mais exatidão (mas também maior gasto computacional quando $m \gg n$). Introduzindo um m-vetor

$y = Ax - b$ obtém-se a igualdade $A^H y = 0$ a partir de (4.14). Essas duas equações são reescritas como

$$\begin{pmatrix} A & -I \\ 0 & A^H \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix} \tag{4.15}$$

dando origem a um sistema quadrado $(n+m) \times (n+m)$, onde I é a matriz identidade $m \times m$. O sistema (4.15) é muito melhor condicionado do que as equações normais que o originaram e, agora, pode ser resolvido lançando mão do algoritmo desenvolvido previamente para a resolução de sistemas quadrados. A parte x da inclusão resultante é a solução x das equações (4.14).

Os sistemas subdeterminados, por sua vez, são abordados de forma similar. Como esse tipo de sistema apresenta infinitas soluções, o vetor-solução y desejado é aquele que, entre todos os vetores x, possui norma euclidiana mínima. O vetor y procurado pode ser determinado como $y = A^H x$, para o qual x é a solução de $AA^H x = b$. Como no caso anterior, essas duas equações são escritas na forma de bloco, dando origem a (4.16) que novamente é um sistema quadrado $(n+m) \times (n+m)$, onde I é a matriz identidade $n \times n$.

$$\begin{pmatrix} A^H & -I \\ 0 & A \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ b \end{pmatrix} \tag{4.16}$$

O sistema (4.16) é resolvido com os métodos desenvolvidos anteriormente. A parte y da inclusão resultante é a solução desejada.

Finalmente, são abordados os dois problemas restantes, ou seja, o cálculo da pseudo-inversa A^+ de uma matriz A com posto máximo, com $m \neq n$. A pseudo-inversa a que se refere é a de *Moore-Penrose*. Sendo A uma matriz sobredeterminada, sua pseudo-inversa é dada por (4.17) ou, equivalentemente, pela solução $Y = A^+$ da equação matricial $A^H AY = A^H$.

$$A^+ = (A^H A)^{-1} A^H \tag{4.17}$$

A partir da introdução da matriz $m \times m$ $X = AY - I$ decorre que $A^H X = 0$ e, como feito anteriormente, escreve-se essas duas equações como em (4.18) com ambas matrizes identidade I de dimensão $m \times m$.

$$\begin{pmatrix} A & -I \\ 0 & A^H \end{pmatrix} \cdot \begin{pmatrix} Y \\ X \end{pmatrix} = \begin{pmatrix} I \\ 0 \end{pmatrix} \tag{4.18}$$

Resolvendo (4.18) coluna por coluna, da mesma forma como obtém-se a inversa de matrizes quadradas, calcula-se uma inclusão para a pseudo-inversa A^+, através da extração do bloco X da solução gerada. Como a pseudo-inversa obedece a propriedade $(A^+)^H = (A^H)^+$, segue de (4.17) que no caso subdeterminado tem-se (4.19) ou, ainda, que A^+ é a solução $Y = A^+$ de $YAA^H = A^H$.

$$A^+ = A^H (AA^H)^{-1} \tag{4.19}$$

Com a matriz $X = YA - I$ é fácil inferir que $XA^H = 0$. Tomando a hermitiana das duas últimas igualdades, pode-se escrever a equação matricial em bloco conforme (4.20), onde as matrizes identidades são $m \times m$.

$$\begin{pmatrix} A^H & -I \\ 0 & A \end{pmatrix} \cdot \begin{pmatrix} Y^H \\ X^H \end{pmatrix} = \begin{pmatrix} I \\ 0 \end{pmatrix} \tag{4.20}$$

Resolve-se a equação coluna por coluna, como no caso da pseudo-inversa de uma matriz sobredeterminada, e, ao final do processo, isola-se o bloco Y^H da inclusão calculada, que é a solução procurada.

4.1.1 Algoritmos

De acordo com a discussão teórica realizada na seção 4.1, pode-se apresentar os seguintes algoritmos utilizados na resolução dos problemas que são trabalhados no *solver* LSS:

Algoritmo 4.1: Solução do problema (s): {procedure}{Cálculo de uma inclusão para a solução do sisteme linear quadrado $Ax = b$}

Parte I (aproximação da inversa em precisão simples)

I.1 calcula uma aproximação da inversa R de A (por exemplo usando o algoritmo de Gauss-Jordan)

I.2 calcula uma aproximação $\tilde{x} := Rb$ de x
refina o valor de $\tilde{x}$ através da iteração:
repeat
$\tilde{x} := \tilde{x} + R(b - A\tilde{x})$
until $\tilde{x}$ sem exatidão ou contagem máxima das iterações excedidas

I.3 calcula a inclusão para o resíduo:
$Z := R\diamond(b - A\tilde{x})$
e para a iteração matricial:
$C := \diamond(I - RA)$

I.4 iteração intervalar
$Y := Z$
repeat
$Y_A := \text{blow}(Y, \epsilon)$ {ϵ - inflation }
$Y := Z + C \cdot Y_A$
until $Y \subset \text{int}(Y_A)$ ou contagem máxima das iterações excedidas

I.5 **if** $Y \subset \text{int}(Y_A)$ **then**
uma solução única x existe e $x \in \tilde{x} + Y$
else
if na Parte I **then**
Parte I falhou, **goto** Parte II com $R_1 := R$
else
algoritmo falhou, a matriz A é mal-condicionada ou singular

Parte II (aproximação da inversa em precisão dupla)

II.1 (calcula uma aproximação da inversa $R := R_1 + R_2$ de A:)
$S := R_1 \cdot A$
calcula uma aproximação da inversa S_1 para S (por exemplo como em I.1)
$S := S_1 \cdot R_1$
$R_2 := S_1 \cdot R_1 - S$
$R_1 := S$

II.2 **goto** passo I.2 da Parte I.

Algoritmo 4.2: Solução do problema (o): {procedure}{Calcula uma inclusão para a solução do sistema linear sobre-determinado $Ax = b$}

1. $A_{big} := \begin{pmatrix} A & -I \\ 0 & A^H \end{pmatrix} \in I\!R^{(n+m)\times(n+m)}$
   $B_{big} := \begin{pmatrix} b \\ 0 \end{pmatrix} \in I\!R^{n+m}$
   $Y_{big} := \begin{pmatrix} x \\ y \end{pmatrix} \in I\!R^{n+m}$
2. resolve $A_{big}Y_{big} = B_{big}$ usando o algoritmo 4.1
3. vetor x do vetor Y_{big} é a inclusão desejada.

Algoritmo 4.3: Solução do problema (u): {procedure}{Calcula uma inclusão para a solução de um sistema sub-determinado $Ax = b$}

1. $A_{big} := \begin{pmatrix} A^H & -I \\ 0 & A \end{pmatrix} \in I\!R^{(n+m)\times(n+m)}$
   $B_{big} := \begin{pmatrix} 0 \\ b \end{pmatrix} \in I\!R^{n+m}$
   $Y_{big} := \begin{pmatrix} x \\ y \end{pmatrix} \in I\!R^{n+m}$
2. resolve $A_{big}Y_{big} = B_{big}$ usando o algoritmo 4.1
3. vetor x do vetor Y_{big} é a inclusão desejada.

Algoritmo 4.4: Solução do problema (S): {procedure}{Calcula uma inclusão para a inversa da matriz quadrada A}

1. (resolve $AX = I$ por coluna)
   for $i := 1$ **to** n **do**
   begin
   $b_i := e_i$
   resolve $Ax_i = b_i$ usando o algoritmo 4.1
   end
2. $X = (x_1, \ldots, x_n)$ é a inclusão desejada.

Algoritmo 4.5: Solução do problema (O): {procedure}{Calcula uma inclusão para a pseudo-inversa da matriz $A_{m\times n}$, $m > n$}

1. $A_{big} := \begin{pmatrix} A & -I \\ 0 & A^H \end{pmatrix} \in I\!R^{(n+m)\times(n+m)}, I \in I\!R^{m\times m}$
   $B_{big} := \begin{pmatrix} I \\ 0 \end{pmatrix} \in I\!R^{n+m\times m}, I \in I\!R^{m\times m}, 0 \in I\!R^{n\times m}$
   $Y_{big} := \begin{pmatrix} X \\ Y \end{pmatrix} \in I\!R^{n+m\times m}, X \in I\!R^{m\times m}, Y \in I\!R^{n\times m}$
2. resolve $A_{big}Y_{big} = B_{big}$ usando o algoritmo 4.4

3. o bloco X da matriz Y_{big} é a inclusão desejada.

Algoritmo 4.6: Solução do problema (U): {procedure}{Calcula a inclusão da pseudo-inversa da matriz $A_{m\times n}$, $m < n$}

1. $A_{big} := \begin{pmatrix} A^H & -I \\ 0 & A \end{pmatrix} \in I\!R^{(n+m)\times(n+m)}$, $I \in I\!R^{m\times m}$
   $B_{big} := \begin{pmatrix} I \\ 0 \end{pmatrix} \in I\!R^{n+m\times n}$, $I \in I\!R^{n\times n}, 0 \in I\!R^{m\times n}$
   $Y_{big} := \begin{pmatrix} Y^H \\ X^H \end{pmatrix} \in I\!R^{n+m\times n}$, $X \in I\!R^{n\times n}, Y \in I\!R^{n\times m}$
2. resolve $A_{big}Y_{big} = B_{big}$ usando o algoritmo 4.4
3. o bloco X da matriz Y_{big} é a solução desejada.

No apêndice B é apresentado um exemplo de um pequeno programa que faz uso do *solver* LSS em sua versão intervalar (ILSS). Este programa possibilita que o usuário entre com os valores das matrizes e do vetor de um sistema linear na forma de intervalos e o resultado (verificado) apresentado ao final será na forma de um vetor intervalar.

4.2 *Solver* para Sistemas Lineares Esparsos

O *solver BAND* é um *solver* para a resolução de sistemas de equações lineares com matrizes esparsas, em especial para matrizes do tipo banda. Sistemas com matrizes bandas são um caso particular de sistemas esparsos, onde os elementos não nulos encontram-se em torno da diagonal principal. Um exemplo de um sistema linear esparso do tipo banda pode ser visto na equação 4.21.

$$Ax = \begin{pmatrix} a_{1,1} & & & & \\ \vdots & \ddots & & 0 & \\ a_{m,1} & & \ddots & & \\ & \ddots & & \ddots & \\ 0 & & a_{n,n-m+1} & \cdots & a_{n,n} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ \\ b_n \end{pmatrix} = b \qquad (4.21)$$

O algoritmo utilizado para resolver sistemas densos não é muito eficiente neste caso, uma vez que é necessário grande quantidade de memória para alocação dos sistemas e há um aumento no tempo de execução. Por se tratar de um caso particular de sistemas esparsos, as matrizes bandas podem ser alocadas em faixas (*bandwidths*) onde os elementos não nulos estão presentes. Neste *solver* empregou-se a Computação Verificada na elaboração de algoritmos numéricos auto-validáveis e com controle automático de erros. Entretanto, o uso de intervalos acarreta em um efeito conhecido como *wrapping effect* (NEUMAIER, 1993). O *wrapping effect* é um efeito que faz o diâmetro dos intervalos resultantes aumentar rapidamente com o uso de intervalar, ou seja, o intervalo calculado como resposta diverge bastante da resposta exata. Alguns simples exemplos podem ilustrar este efeito muito drasticamente, conforme exemplos 4.1, 4.2 e 4.3.

Exemplo 4.1:

Este exemplo é o clássico exemplo onde o *wrapping effect* foi descoberto primeiro. Esse exemplo parte de um problema com o valor inicial $u'' + u = 0, u(0) = u_0, u'(0) = u_1$, onde a matriz A é definida por (4.22) e uma simples recursão em $I\!R^2$ é definida por (4.23).

$$A_\phi = \begin{pmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{pmatrix} \tag{4.22}$$

$$x_{n+1} = A_\phi x_n, \;\; n \geq 0, \quad x_0 \in [x_0] := \epsilon \begin{pmatrix} [-1,1] \\ [-1,1] \end{pmatrix} \;\; \epsilon > 0. \tag{4.23}$$

Para cada vetor pontual $x_0 \in [x_0]$ têm-se obviamente a solução descrita em (4.24), isto é, o vetor x_n é constantemente rotacionado ao redor da origem por um ângulo ϕ em cada iteração. Com isso, o conjunto solução $\{x_n | x_0 \in [x_0]\}$ é justamente o quadrado original $[x_0]$ rotacionando constantemente ao redor da origem. Trivialmente este conjunto restante é limitado para todo $n \geq 0$, conforme (4.24).

$$x_n = A_\phi^n x_0 = \begin{pmatrix} \cos n\phi & \sin n\phi \\ -\sin n\phi & \cos n\phi \end{pmatrix} x_0, \; n \geq 0 \tag{4.24}$$

Ao invés disso, pode-se realizar a iteração (4.23) utilizando a aritmética intervalar (com computações exatas, sem erros de arredondamento), conforme apresentado em (4.25), onde todas as operações são agora intervalares.

$$[x_{n+1}] = A_\phi [x_n] \tag{4.25}$$

Com isso, obtêm-se as seguintes iterações intervalares para $[x_i]$:

$$\begin{aligned}
[x_1] &= \begin{pmatrix} [-\epsilon,\epsilon]\cos\phi + [-\epsilon,\epsilon]\sin\phi \\ [-\epsilon,\epsilon]\sin\phi + [-\epsilon,\epsilon]\cos\phi \end{pmatrix} = \epsilon(|\sin\phi| + |\cos\phi|) \begin{pmatrix} [-1,1] \\ [-1,1] \end{pmatrix} \\
[x_2] &= \epsilon(|\sin\phi| + |\cos\phi|)^2 \begin{pmatrix} [-1,1] \\ [-1,1] \end{pmatrix} \\
&\vdots \\
[x_n] &= \epsilon(|\sin\phi| + |\cos\phi|)^n \begin{pmatrix} [-1,1] \\ [-1,1] \end{pmatrix}
\end{aligned}$$

Os vetores intervalares $[x_n]$ são inclusão que limitam o conjunto solução $\{x_n | x_0 \in [x_0]\}$. Entretanto, mesmo que $|\sin\phi| + |\cos\phi| > 1$ para todo $\phi \neq k\pi/2$ vê-se que o diâmetro para esta inclusão diverge para ∞. Isto é efeito do *wrapping effect* e possui uma simples explanação geométrica: o vetor intervalar $[x_0]$ é rotacionado por um ângulo ϕ ao redor da origem e, então, é incluído pelo menor vetor intervalar possível $[x_1]$ cujo diâmetro é $|\sin\phi| + |\cos\phi|$ vezes maior que o de $[x_0]$. O mesmo processo é repetido em cada etapa das iterações. Nota-se que este efeito ocorre mesmo quando se usa uma inclusão ótima em cada etapa das iterações. A figura 4.1 ilustra este processo geométrico.

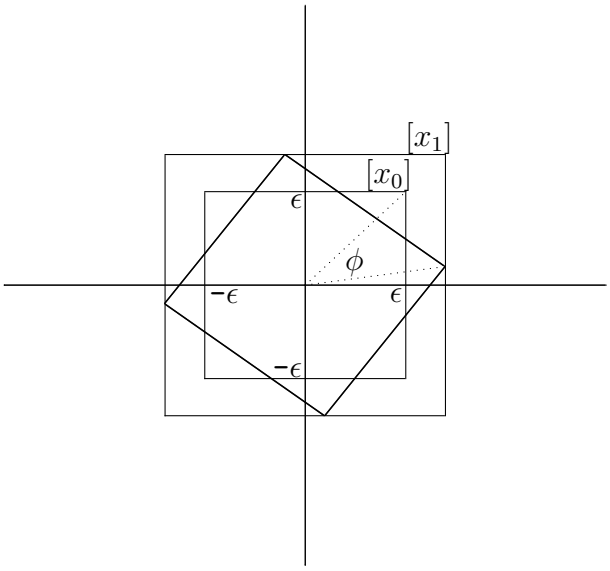

Figura 4.1: Efeito Geométrico do *Wrapping Effect*

Esta iteração pode ser reescrita como um sistema de equações lineraes $Ax = b$ sendo a matriz A uma matriz triangular superior e banda. Reescreve-se $A \in \mathbb{R}^{(2n+2)\times(2n+2)}$ e $x, b \in \mathbb{R}^{2n+2}$ em uma notação de blocos com 2×2-blocos em A, I sendo uma matriz identidade 2×2 e dois vetores x e b, conforme a seguinte notação:

$$\begin{aligned} A &= \begin{pmatrix} I & & & \\ A & -I & & \\ & \ddots & \ddots & \\ & & A & -I \end{pmatrix} \\ x &= (x_0, x_1, \ldots, x_n)^T \\ b &= (x_0, 0, \ldots, 0)^T \end{aligned}$$

Obviamente solucionando esse sistema em bloco em aritmética intervalar é equivalente a computação da iteração em (4.25). Com isso tem-se um primeiro exemplo que pode produzir uma grande sobre estimação nos intervalos de resposta.

Exemplo 4.2:

Outro simples exemplo como o anterior é o de uma matriz triangular inferior densa, sendo A uma matriz $n \times n$ conforme (4.26).

$$A = \begin{pmatrix} 1 & & & & & \\ 2 & 1 & & & & \\ 3 & 2 & 1 & & & \\ 4 & 3 & 2 & 1 & & \\ \vdots & \ddots & \ddots & \ddots & \ddots & \\ n & \cdots & 4 & 3 & 2 & 1 \end{pmatrix} \tag{4.26}$$

A inversa A^{-1} pode ser verificada para ser uma matriz triangular inferior banda, conforme (4.27).

$$A^{-1} = \begin{pmatrix} 1 & & & & \\ -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \end{pmatrix}. \tag{4.27}$$

Se for escolhido o vetor intervalar descrito em (4.28), então a inclusão ótima para a solução exata $\{x = A^{-1}b | b \in [b]\}$ é definida conforme (4.29), a qual é idêntica a $[-2\epsilon, 2\epsilon]$ para todos os elementos, menos o primeiro e o segundo.

$$[b] = \epsilon([0,1],[0,1],\ldots,[0,1])^T \tag{4.28}$$

$$[x]_{opt} = \epsilon([0,1],[-2,1],[-2,2],\ldots,[-2,2])^T \tag{4.29}$$

Com isso, resolvendo $Ax = [b]$ pela aritmética intervalar têm-se a inclusão apresentada em (4.30), onde os diâmetros estão, obviamente, divergindo. Este é um exemplo onde o aumento dos intervalos da inclusão ocorrem sem uma boa e óbvia interpretação geométrica.

$$x \in \epsilon \begin{pmatrix} [0,1] \\ [-2,1] \\ [-5,5] \\ [-17,17] \\ [-58,58] \\ [-198,198] \\ [-676,676] \\ [-2308,2308] \\ [-7880,7880] \\ [-26904,26904] \\ \vdots \end{pmatrix} \tag{4.30}$$

Nota-se que o sistema $By = [b]$ com a matriz inversa $B := A^{-1}$ como matriz de coeficientes também mostra um comportamento similar: o conjunto solução $\{y = B^{-1}b = Ab | b \in [b]\}$ não está limitado aqui mas a inclusão obtida através da aritmética intervalar também possui uma solução sobre-estimada. Este sistema também tem uma grande similaridade com o próximo exemplo.

Exemplo 4.3:

O último exemplo começa com um escalar, uma equação diferencial linear. Os bem conhecidos polinômios de Chebychev $T_n(x)$ podem ser definidos por uma relação recorrente de segunda ordem definida em (4.31).

$$T_0(x) = 1, \;\; T_1(x) = x, \;\; T_{n+1}(x) - 2xT_n(x) + T_{n-1}(x) = 0 \tag{4.31}$$

Se deseja-se computar uma inclusão de $T_n(x)$ para algum valor x_0 e um auto valor de n então pode-se usar essa equação diferencial e calcular o valor desejado pela aritmética intervalar. Na verdade, esta computação não é nada mais do que o cálculo intervalar do sistema (4.32).

$$\begin{pmatrix} 1 & & & & \\ 0 & 1 & & & \\ 1 & -2x_0 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2x_0 & 1 \end{pmatrix} \begin{pmatrix} T_0(x_0) \\ T_1(x_0) \\ T_2(x_0) \\ \vdots \\ T_n(x_0) \end{pmatrix} = \begin{pmatrix} 1 \\ x_0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{4.32}$$

Este é um sistema somente com dados pontuais (não intervalares) entretanto, usando a aritmética intervalar de ponto-flutuante, pode-se introduzir intervalos não-zeros para controlar os erros de arredondamento. Neste último exemplo também pode ocorre um rápido aumento no diâmetro da inclusão da solução. Se o argumento x_0 não é um número de ponto-flutuante, então tais intervalos poderão inciar a computação do início. Se for pego, por exemplo, $x_0 = 0.99$ então obtêm-se a seguinte saída, através de um pequeno programa em C–XSC (dupla precisão de acordo com padrão IEEE) para os valores de n a para as inclusões para $T_n(0.99)$:

```
 n                          T[n](0.99)

 2   [  9.601999999999999E-001,  9.602000000000007E-001 ]
 3   [   9.111959999999999E-001,   9.111960000000002E-001 ]
 4   [   8.439680799999998E-001,   8.439680800000004E-001 ]
 5   [    7.598607983999999E-001,    7.598607984000001E-001 ]
 6   [    6.605563008319800E-001,    6.605563008320200E-001 ]
 7   [     5.480406772473E-001,     5.480406772474E-001 ]
 8   [     4.245642401176E-001,     4.245642401179E-001 ]
 9   [     2.925965181856E-001,     2.925965181861E-001 ]
10   [      1.54776865889E-001,      1.54776865891E-001 ]
15   [       -5.246430527E-001,       -5.246430525E-001 ]
20   [        -9.52088247E-001,        -9.52088240E-001 ]
25   [          -9.222663E-001,          -9.222657E-001 ]
30   [             -4.496E-001,             -4.494E-001 ]
35   [                2.3E-001,                2.5E-001 ]
40   [                6.9E-001,                9.3E-001 ]
45   [               -8.0E+000,                1.0E+001 ]
50   [               -7.1E+002,                7.2E+002 ]
```

Obviamente os valores dos polinômios de Chebychev podem ser computados de uma maneira maus fácil, desde que $T_n(x) = \cos(n \arccos x)$. Este exemplo poderia demonstrar somente as dificuldades a respeito do cálculo de equações diferenciais mesmo sendo um simples exemplo como o apresentando anteriormente. De qualquer forma esse exemplo é relevante para muitas outras funções que podem ser computadas por equações diferenciais (por exemplo, funções de Bessel onde a recurssão $J_{n+1}(x) - \frac{2n}{x} J_n(x) + J_{n-1}(x) = 0$ é similarmente sensível ao *wrapping effect*).

Estes exemplos mostram claramente que é importante encontrar bons métodos para a inclusão da solução de sistemas de equações triangulares. Também é importante destacar o estreito relacionamento entra equações diferenciais e sistema triangulares bandas. Para tentar minimizar esse efeito, o programa foi desenvolvido baseado na resolução de equações diferenciais, o que mostrou bastante eficiência no tratamento deste efeito. Após essa experiência, um programa para a resolução de sistemas lineares triangulares com matrizes

bandas foi desenvolvido. O algoritmo implementado no *solver* foi baseado no estreito relacionamento existente entre as matrizes com estrutura banda e equações diferenciais. De acordo com esse relacionamento, as equações diferenciais podem ser reescritas equivalentemente como um sistema linear triangular com matrizes banda. Similarmente, pode-se reescrever um sistema triangular com matrizes banda como equações diferenciais. O cálculo da aproximação da inversa através do método de Gauss (utilizado para resolver sistemas densos) foi substituído pelo cálculo da aproximação através da Decomposição LU da matriz A. Para um melhor entendimento do *solver* BAND, são apresentados a seguir os algoritmos usados na sua implementação.

O algoritmo (4.7) classifica as colunas da matriz de entrada B de acordo com o tamanho das bordas de $P(B, z)$ onde o vetor de intervalos é um segundo parâmetro de entrada. O procedimento `SWAP` usado aqui troca o conteúdo dos seus dois parâmetros.

Algoritmo 4.7: SortColumns (B, z) {procedure}
{Classifica as colunas de B diminuindo o tamanho das bordas de P(B,z)}
entrada: *rmatrix* B, interval vector z
saída: *rmatrix* B com as colunas rearranjadas

1. {computa os comprimentos das bordas (quadrado desses comprimentos basta):}
   for $i := 1$ **to** m **do** $length_i := (B_{*,i} \cdot B_{*,i}) \cdot \mathrm{diam}(z_i)^2$

2. {classificando as colunas de B de acordo com o tamanho desses comprimentos}
   for $i := 1$ **to** $m - 1$ **do**
   for $j := i + 1$ **to** m **do**
   if $length_i < length_j$ **then** Swap $(B_{*,i}, B_{*,j})$
   Swap $(length_i, length_j)$

- **return** B

O algoritmo (4.8) computa uma aproximação da Decomposição-QR da matriz de entrada A com aritmética de ponto-flutuante ordinária. Aqui não será apresentada a teoria para este método, pois é um algoritmo padrão que pode ser encontrado em (STOER; BULIRSCH, 1980). A matriz A é sucessivamente multiplicada com as matrizes *Householder* para transformá-la na forma triangular R. A parte ortogonal Q é produto acumulado dessas matrizes *Householder*.

Algoritmo 4.8: QR (A) {function}
{computação em ponto-flutuante da Decomposição-QR de A }
entrada: *rmatrix* A
saída: aproximação ortogonal da matriz Q (function result)

(a) {Começa com a matriz identidade}
   $Q := I$

(b) {loop para todas as colunas de A (menos a última)}
   for $k := 1$ **to** $m - 1$ **do**

$b := A_{k..n,k}$ { b é um vetor de dimensão $(n-k+1)$ }
if $b \neq 0$ **then**
if $A_{k,k} < 0$ **then** $s := \|b\|_2$ **else** $s := -\|b\|_2$
$b_k := A_{k,k} - s$
$s := 1/(s \cdot b_k)$
{elimina as entradas abaixo da diagonal em A:}
for $i := k+1$ **to** n **do**
$r := s \cdot (b \cdot A_{k..n,i})$
for $j := k$ **to** n **do** $A_{j,i} := A_{j,i} + b_j \cdot r$
{acumula o produto das "Householder" Q:}
for $i := 1$ **to** n **do**
$r := s \cdot (Q_{i,k..n} \cdot b)$
for $j := k$ **to** n **do** $Q_{i,j} := Q_{i,j} + b_j \cdot r$

(c) {retorna o valor da função}
QR $:= Q$

O algoritmo (4.9) é uma função que computa a inclusão (*enclosure*) da inversa da aproximação da matriz ortogonal Q usando a estimativa da séries de Neumann. Um código de erro 1 é retornado se a norma $q = ||I - Q^T Q||_\infty \geq 1$, caso contrário, código 0 é retornado.

Algoritmo 4.9: Inv (Q, err_code) {function}
{Inclusão da inversa da aproximação da matriz ortogonal Q }
entrada: *rmatrix* Q (aproximação ortogonal)
saída: Inclusão de Q^{-1} como *imatrix* (resultado da função), *integer* err_code indicando sucesso na inversão (=0 ou =1 caso contrário).

(a) {norma do resíduo}
$q := \|I - Q^T Q\|_\infty$
if $q \geq 1$ **then** $err\_code := 1$ **else** $err\_code := 0$
if $err\_code = 1$ **then return**

(b) $\epsilon := \|Q^T\|_\infty \cdot q/(1-q)$
for $i := 1$ **to** m **do**
for $j := 1$ **to** m **do**
$Q1_{i,j} := Q_{j,i} + [-\epsilon, \epsilon]\|Q^T\|_\infty$

(c) {returna a inclusão da inversa como valor da função:}
Inv $:= Q1$

O algoritmo (4.10) computa a inclusão da solução de um vetor de equação diferencial de primeira ordem usando transformações de coordenadas com matrizes de bases ortogonais como discutido em (KRÄMER; KULISCH; LOHNER, 1994). No algoritmo, apenas um passo é computado, a entrada do algoritmo é um "parallel-epiped" P_0(y0,B0,z0) e a saída é um "parallel-epiped" P_1(y1,B1,z1). Para se obter uma inclusão componente por componente de P_1 o vetor de intevalos $y_1 + B_1 * z_1$ precisa ser computado depois de chamar este algoritmo. A matriz de entrada A (=A_i) e o vetor de entrada d (=d_i) foram escolhidos como vetores de intervalos aqui para possibilitar dados de intervalos e evitar dados de ponto-flutuante (inclusão em intervalos).

Algoritmo 4.10: MatVecIter $(A, d, y0, B0, z0, y1, B1, z1)$ {procedure}
{Computa um passo da iteração matriz-vetor $y_{i+1} = Ay_i + d_i$ }

entrada: *imatrix* $A = (A_i)$, *rmatrix* $B0 = (B_i)$ (do passo anterior), *ivector* $d = (d_i)$, *rvector* $y0 = (\tilde{y}_i)$ and *ivector* $z0 = ([z_i])$ (do passo anterior).
saída: *rvector* $y1 = (\tilde{y}_{i+1})$, *rmatrix* $B1 = (B_{i+1})$ e *ivector* $z1 = ([z_{i+1}])$.

(a) {computando aproximação}
$y1 := \mathrm{mid}(A) \cdot y0 + \mathrm{mid}(d)$

(b) {inclusão do erro}
$B1 := \mathrm{mid}(A) \cdot B0$
SortColumns $(B1, z0)$
QR $(B1)$
$BI1 :=$ Inv $(B1, err\_code)$
if $err\_code = 1$ **then** escreve na tela um warning: "inversion not successful" (inversão falhou)
$z1 := (BI1 \cdot A \cdot B0) \cdot z0 + BI1 \cdot \diamond(d + A \cdot y0 - y1)$

(c) **return** $y1, B1, z1$

Por último o algoritmo (4.11) faz a computação da solução de uma equação escalar diferencial de primeira ordem m. Isso é essencialmente idêntico ao algoritmo anterior 4.10 para vetores de equações diferenciais de primeira ordem, apenas a forma de entrada é diferente, i.e. , aqui precisa-se apenas de um `ivector` a, contendo os coeficientes $a_0, \ldots, a_m$ e um `interval` b para b_i.

Algoritmo 4.11: DiffIter $(A, b, y0, B0, z0, y1, B1, z1)$ {procedure}
{Computa um passo da iteração escalar com equações diferenciais $a_{i,m}x_{i+m} + \cdots + a_{i,0}x_i = b_i$ }
entrada: *ivector* A (vetor de coeficientes), *interval* b (lado direito), *rmatrix* $B0 = (B_i)$ (do passo anterior), *rvector* $y0 = (\tilde{y}_i)$ e *ivector* $z0 = ([z_i])$ (do passo anterior).
saída: *rvector* $y1 = (\tilde{y}_{i+1})$, *rmatrix* $B1 = (B_{i+1})$ e *ivector* $z1 = ([z_{i+1}])$.

(a) {computando aproximação de ponto-flutuante}
$y1_{1..m-1} := y0_{2..m}$
$y1_m := (\mathrm{mid}(b) - \sum_{i=1}^{m} \mathrm{mid}(a_{i-1}) \cdot y0_i)/\mathrm{mid}(a_m)$

(b) {inclusão do erro}
$B1 :=$ Coeff $(\mathrm{mid}(A)) \cdot B0$
SortColumns $(B1, z0)$
QR $(B1)$
$BI1 :=$ Inv $(B1)$
$d := \diamond(b - \sum_{i=1}^{m} a_{i-1} \cdot y0_i)/a_m - y1_m$
$z1 := (BI1 \cdot (\text{Coeff}\,(A) \cdot B0)) \cdot z0 + BI1_{*,m} \cdot d$

(c) return $y1, B1, z1$

$$A_i := \begin{pmatrix} 0 & 1 & \cdots & 0 & 0 \\ 0 & 0 & \ddots & 0 & \vdots \\ \vdots & \vdots & \ddots & \vdots & 0 \\ 0 & 0 & & 1 & 0 \\ 0 & 0 & \cdots & 0 & 1 \\ \dfrac{-a_{i,0}}{a_{i,m}} & \dfrac{-a_{i,1}}{a_{i,m}} & \cdots & \dfrac{-a_{i,m-2}}{a_{i,m}} & \dfrac{-a_{i,m-1}}{a_{i,m}} \end{pmatrix} \in I\!R^{m \times m} . \tag{4.33}$$

Neste último algoritmo usa-se uma pseudo-função `COEFF` que é apenas uma indicação de que seus argumentos não devem ser interpretados como um vetor, mas sim como uma matriz que contém os coeficientes do vetor de argumentos na forma descrita em (4.33).

No apêndice C é apresentado um exemplo de um pequeno programa que faz uso do *solver* BAND. Este programa possibilita que o usuário entre com os valores das matrizes e do vetor de um sistema linear na forma de reais e o resultado (verificado) apresentado ao final será na forma de um vetor intervalar.

4.3 Conclusões do Capítulo

O desenvolvimento dos *solvers* verificados é uma das mais importantes contribuições desta tese. Com a sua utilização e a sua futura adaptação para execução em paralelo será possível solucionar, de maneira eficiente e exata, sistemas de equações lineares densos e esparsos. Os resultados já obtidos até o momento (ver capítulo 6) demonstram a excelente qualidade numérica obtida através da utilização destes *solvers*. Outro fato interessante é que, com base nas versões seqüências atuais, um estudo mais apronfudado poderá ser realizado com o objetivo de usar todas as características da Computação Verificada no desenvolvimento e/ou implementação de métodos numéricos computacionais. Outros detalhes sobre essas possibilidades de continuidade de pesquisas serão abordados nas conclusões desta tese (capítulo 7).

5 RESOLUÇÃO DE SELAS EM APLICAÇÕES REAIS

A resolução de sistemas de equações lineares é um dos problemas numéricos mais comuns em aplicações científicas. Tais sistemas surgem, por exemplo, em conexão com a solução de equações diferenciais parciais, determinação de caminhos ótimos em redes (grafos) e interpolação de pontos, dentre outros (BORTOLI et al., 2003). Os problemas que podem ser resolvidos através desses sistemas lineares envolvem, por exemplo, a determinação de potenciais em certas redes elétricas, o cálculo do estresse numa armação de construção ou de uma estrutura de ponte, o cálculo do padrão de escoamento num sistema hidráulico com ramos interconectados ou o cálculo das estimativas das concentrações de reagentes sujeitos a simultâneas reações químicas. Além desses exemplos, pode-se citar aplicações nas áreas de hidrodinâmica (RIZZI et al., 2004,?), transporte de massa (mecânica de fluídos) (WANG; ZIAVRAS, 2003), sistemas elétricos (KOESTER; RANKA; FOX, 1994), fluxo de potências (BARBOZA; DIMURO; REISER, 2004) e a simulação de processos de aeração de grãos em silos de armazenagem (COPETTI; PADOIN; KHATCHATOURIAN, 2002a; COPETTI et al., 2003). Devido a esses fatores é que, para esta pesquisa, foram selecionados alguns métodos para a solução de sistemas lineares que são utilizados na solução de aplicações reais de grande porte. Essas aplicações estão sendo trabalhadas em pesquisas relacionadas aos grupos de pesquisa nos quais essa tese está vinculada. Foram selecionados dois métodos diretos (*Householder* e *Givens* – relacionados com a aplicação de aeração de grãos em silos de armazenagem) e um método iterativo (Gradiente Conjugado – relacionado a aplicação de hidrodinâmica).

5.1 Aplicação de Hidrodinâmica

O GMCPAD e o Grupo de Processamento Paralelo e Distribuído do Instituto de Informática da UFRGS vêm trabalhando no estudo e desenvolvimento de modelos hidrodinâmicos e de transporte de massa paralelos em *clusters* de máquinas multiprocessadas (projeto envolvendo o controle de poluentes do rio Guaíba). O desenvolvimento deste tipo de modelo envolve diversos aspectos como a implementação paralela de métodos numéricos, de métodos de decomposição de domínio e de balanceamento de carga. Nesta aplicação foi apresentado um modelo computacional paralelo multifísica para a simulação do transporte de substâncias e do escoamento hidrodinâmico, bidimensional (2D) e tridimensional (3D) em corpos de água. Sua motivação foi centrada no fato de que as margens e zonas costeiras de rios, lagos, estuários, mares e oceanos são locais de aglomerações de seres humanos, dada a sua importância para as atividades econômica, de transporte e de lazer, causando desequilíbrios a esses ecossistemas. Nesta aplicação um dos métodos utilizados é o método Gradiente Conjugado, o qual foi implementado por esse trabalho em sua versão seqüencial com e sem a característica da alta exatidão. Detalhes sobre essa

aplicação e seus aspectos matemáticos e numéricos podem ser encontrados nas teses de doutorado de Dorneles (DORNELES, 2003) e Rizzi (RIZZI, 2002).

5.1.1 Método do Gradiente-Conjugado

Os métodos iterativos de resolução de sistemas lineares favorecem a resolução de sistemas lineares de grande porte. Esses métodos calculam gradualmente a solução do sistema, até que um critério de parada seja satisfeito: número máximo de iterações ou limite de tolerância do erro. O método do Gradiente Conjugado (GC) pertence a essa classe e é um método não-estacionário por possuir hereditariedade em suas iterações. O Gradiente Conjugado é destinado à resolução de sistemas lineares cuja matriz é simétrica, positiva e definida (spd). É baseado na minimização da função quadrática e sua iteração composta por operações de álgebra linear. O gradiente (campo vetorial) aponta na direção crescente da função quadrática. Desse modo, a idéia básica do algoritmo é dar passos, em cada iteração, na direção oposta a do gradiente, de tal forma que a direção já pesquisada não seja repetida. O Gradiente Conjugado também pode ser pré-condicionado (GCP). Essa técnica consiste em gerar uma matriz aproximada à inversa da matriz do sistema linear para aumentar a convergência do método. Conforme a técnica utilizada para fazer essa aproximação, tem-se os diferentes pré-condicionadores. Detalhes numéricos e computacionais a respeito do método Gradiente Conjugado e seus pré-condicionadores poderão ser encontrados em (MAILLARD, 2005), (KARNIADAKIS; KIRBY II, 2003), (SAAD, 2003) e (BORTOLI et al., 2003).

5.2 Aplicação de Aeração de Grãos em Silos de Armazenagem

Entre os métodos diretos de solução de SELA, os métodos de reflexões e rotações apresentam a maior estabilidade numérica (em comparação com métodos com mesma ordem de operações aritméticas). Apesar disso, os métodos de *Householder* e *Givens* são utilizados principalmente para *QR* decomposição de matrizes, ortogonalização dos vetores, resolução de problemas de autovalores, etc. Os algoritmos seqüenciais para esses métodos são muito utilizados nas áreas da matemática aplicada e, como exemplo dessa aplicabilidade, pode-se citar que esses métodos são utilizados na simulação de processos de aeração de grãos em silos de armazenagem. Essa aplicação visa a simulação do escoamento do ar em armazéns de alta capacidade, considerando a não-uniformidade da massa de grãos. Essa simulação justifica-se pela existência, atualmente, de se construir silos com alturas cada vez maiores e a massa de grãos já não pode mais ser considerada homogênea. Por causa dessa não-homogeniedade há uma significante alteração nos parâmetros físicos envolvidos no processo de aeração, tais como velocidade do ar e queda da pressão estática. Detalhes sobre essa aplicação podem ser obtidos em (KHATCHATOURIAN; SAVICKI, 2001), (COPETTI; PADOIN; KHATCHATOURIAN, 2002a) e (COPETTI et al., 2003).

5.2.1 Método de *Householder*

Esta seção tem por objetivo realizar uma breve descrição do método de *Householder* ou Transformações de *Householder*. Considera-se um vetor-coluna unitário $w \in \Re^n$. A matriz $U = I - 2ww^T$ chama-se transformação de *Householder* (matriz de reflexão). É fácil de demonstrar que U é ortogonal, simétrica e que w é um autovetor desta matriz associada com um autovalor $\lambda = -1$. Denotando as colunas da matriz A por a_i, aplica-se a transformação de *Householder* para a matriz aumentada B, conforme (5.1).

$$UB = U[Ab] = (Ua_1 \;\; Ua_2 \;\; \ldots \;\; Ua_i \;\; \ldots \;\; Ua_n \;\; Ub) \tag{5.1}$$

Pode-se escolher um vetor w em um subespaço $V \in \Re^n$ com $dim(V) = 2$ formado pelos a_i e o vetor unitário $e_1 = (1, 0, 0, \ldots, 0, \ldots, 0)^T$ de modo para que o vetor-coluna Ua_1 tenha somente um componente não-nulo, isto é: $Ua_1 = |a_1|e_1 = (|a_1|, 0, 0, \ldots, 0, \ldots, 0)^T$. O vetor w que apresenta a direção de uma reta-bissetriz entre os valores a_1 e e_1 caso $a_1 1 \geqslant 0$ (ou ortogonal à esta direção caso $a_1 1 \leqslant 0$) foi calculado conforme (5.2).

$$w = (a_1 + sign(a_{11})|a_1|e_1)/|a_1 + sign(a_{11})|a_1|e_1| \tag{5.2}$$

$$B_1 = UB = U[Ab] = (Ua_1 Ua_2 \ldots Ua_i \ldots Ua_n Ub) = (|a_1|e_1 Ua_2 \ldots Ua_i \ldots Ua_n Ub) \tag{5.3}$$

```
1 - Divide a matriz em colunas
2 - Faça de 1 até a ordem da matriz
 2.1 - Calcula os vetores dos senos e co-senos
 2.2 - Altera a matriz usando os vetores dos senos e
co-senos
 2.3 - Desconsidera a primeira linha e a primeira
coluna da matriz (diminuindo a ordem do sistema)
3 - Calcula o vetor resultado
```

Figura 5.1: Algoritmo do Método de Householder

Então a matriz B, definida em (5.3), possui a primeira coluna com todos os elementos sub-diagonais nulos. Aplicando o mesmo procedimento para sub-matrizes $B_k = (b_{ij}), k \leqslant i \leqslant n, \;\; k \leqslant j \leqslant n+1, \;\; (k = 2, 3, \ldots, n-1)$, a matriz B seria transformada na forma triangular superior. Caso se realize a QR decomposição, a matriz ortogonal Q pode ser obtida multiplicando as matrizes ortogonais intermediárias, conforme (5.4).

$$Q^T = \begin{pmatrix} I_{n-1} & 0 \\ 0 & U_1 \end{pmatrix} \cdots \begin{pmatrix} I_2 & 0 \\ 0 & U_{n-2} \end{pmatrix} \times \begin{pmatrix} I_1 & 0 \\ 0 & U_{n-1} \end{pmatrix} \times U_n \tag{5.4}$$

Em (5.4) $U_n = U$ e I é a matriz identidade. O índice inferior corresponde a ordem da matriz. O algoritmo seqüencial básico para o método de *Householder* é apresentado na figura 5.1.

Para a paralelização desse algoritmo pode-se explorar diferentes formas para se alcançar um melhor desempenho. A Transformação de *Householder* pode ser perfeitamente utilizada como objeto de estudo por apresentar um grande volume de processamento com elevado tempo de computação, ou seja, a aplicação é de alto custo computacional, além de apresentar uma forte iteração durante o processamento. Na implementação paralela do algoritmo pode-se utilizar uma programação do tipo mestre-escravo, pretendendo escalonar o trabalhos entre os diversos nodos da máquina virtual. A seguir são descritas as tarefas desempenhadas pelo nodo mestre e pelos nodos escravos (COPETTI; PADOIN; KHATCHATOURIAN, 2002b). O algoritmo paralelo é descrito na figura 5.2.

- Nodo-mestre: o mestre divide as linhas da matriz B e as envia para os nodos-escravos iniciarem o processamento em paralelo. O programa entra num laço que começa com $i = 1$ e termina em (ordem inicial do sistema-1). Neste laço, o mestre

espera que cada trabalhador envie a sua parte da coluna i. Após receber de todos, calcula o vetor $w(w = (Ai - |Ai| * e)/(|Ai - |Ai| * e|))$ e envia o vetor w para todos os trabalhadores. O próximo passo é aguardar que todos os trabalhadores enviem o resultado parcial do vetor $w^t B$. Cada vetor que chega é somado com o anterior até que o resultado final de $w^t B$ seja encontrado. Esse vetor é então transmitido para todos os trabalhadores. Após o término do laço o mestre recebe do primeiro trabalhador o vetor x contendo a resposta.

- Nodos escravos: o trabalhador recebe as linhas da matriz B com as quais vai trabalhar e entra no laço que começa com $i = 1$ e termina em (ordem inicial do sistema-1). O escravo manda para o mestre a sua parte da coluna i e espera o vetor w do mestre. O próximo passo é calcular o resultado parcial do vetor $w^t B(w^t B = w^t * B)$ e enviar esse resultado para o mestre. O trabalhador recebe o resultado final de $w^t B$ e aplica a transformação em B, $B = B - 2 * w * w^t B$. Após cada iteração do laço a coluna i e a linha i são desconsideradas para a próxima iteração e a ordem do sistema diminui até que na última iteração a ordem do sistema seja igual a 2. Se durante o laço todas as linhas de um determinado trabalhador são desconsideradas este trabalhador é desativado deixando o cálculo para os trabalhadores restantes. Isto torna o algoritmo dinâmico pois, para um sistema de grande porte, é interessante um grande número de trabalhadores mas a medida que o sistema diminui, muitos trabalhadores degradam o desempenho devido a intensa troca de mensagens na rede. Pelo fato do algoritmo ir desativando os trabalhadores durante o processo enquanto a ordem do sistema diminui, esse problema é minimizado. Ao fim do laço, se o trabalhador não é aquele que recebeu as últimas linhas do sistema, ele espera o resultado do vetor x referente aos próximos trabalhadores. Cada trabalhador então calcula a sua parte do vetor.

Mestre	Escravos
1 - Divide a matriz em colunas 2 - Envia para os escravos	 3 - Recebe as colunas do mestre
faça 1 até a ordem	
5 - Calcula w e envia para os escravos 7 - Calcula $w^t * B$ e envia para os escravos	4 - Envia sua 1ª coluna para o mestre 6 - Recebe w, calcula $w^t * B$ parcial e envia para o mestre 8 - Faz a transformação $B = B - 2 * w * w^t * B$
Desconsidera primeira linha e primeira coluna (diminuindo a ordem do sistema) Desativa dinamicamente os processadores não necessários Fim do laço	
10 - Recebe o resultado do primeiro trabalhador	9 - Envia vetor x parcial para trabalhador anterior, se for o primeiro trabalhador envia para o mestre

Figura 5.2: Algoritmo Paralelo para o Método de *Householder*

O algoritmo automaticamente diminui a quantidade de processos na medida que o sistema vai diminuindo, já que a cada iteração a ordem da matriz B diminui. Ao final,

apenas os resultados são passados de processo para processo, num modelo em *pipeline*, sem que haja necessidade de que um único, conheça toda a matriz.

5.2.2 Método de *Givens*

Esta seção tem por objetivo realizar uma breve descrição do método de *Givens* ou Transformações de *Givens*. Chama-se transformação de *Givens* uma matriz de rotação conforme definido em (5.5).

$$P_{i,j} = \begin{pmatrix} I_1 & 0 & 0 & 0 & 0 \\ 0 & cos\varphi & 0 & sen\varphi & 0 \\ 0 & 0 & I_2 & 0 & 0 \\ 0 & -sen\varphi & 0 & cos\varphi & 0 \\ 0 & 0 & 0 & 0 & I_3 \end{pmatrix} \tag{5.5}$$

Em (5.5), I_1, I_2 e I_3 são matrizes identidades de ordem $(i-1)$, $(j-i-1)$ e $(n-j)$, respectivamente. As funções trigonométricas estão nas intersecções de linhas e colunas com os números (i,j). A matriz P é ortogonal e determina a rotação plana por ângulo φ em hiperplano (i,j). Denotado agora as linhas de matriz por $a_i = (i = 1, 2, \ldots, n)$, $c = cos\varphi$ e $s = sen\varphi$, pode-se escrever o produto de $P_{i,j}$ por A na forma apresentada em (5.6).

$$P_{i,j}A = (a_1 \quad \ldots \quad a_{i-1} \quad ca_i + sa_j \quad a_{i+1} \ldots a_{j-1} \quad -sa_i + ca_j \quad a_{j+1} \ldots a_n)^T \tag{5.6}$$

```
1 - Divide a matriz em linhas
2 - Faça de 1 até a ordem da matriz
 2.1 - Calcula w
 2.2 - Calcula wt × B
 2.3 - Faz a transformação B = B − 2 × w × wt × B
 2.4 - Desconsidera a primeira linha e a primeira
coluna da matriz (diminuindo a ordem do sistema)
3 - Apresenta o vetor resultado
```

Figura 5.3: Algoritmo Seqüencial do Método de *Givens*

Para anular o elemento a_{ji} na matriz A, os valores de $cos\varphi$ e $sen\varphi$ tem que serem calculados por $c = a_{i,i}/\sqrt{a_{i,i}^2 + a_{j,i}^2}$ e $s = a_{j,i}/\sqrt{a_{i,i}^2 + a_{j,i}^2}$. Aplicando as rotações conseqüentes em hiperplanos $(i, i+1), (i, i+2), ldots, (i, n)$, anulam-se todos os elementos sub-diagonais de coluna $i(i = 1, 2, \ldots, n-1)$. Finalmente a matriz inicial reduz á forma triangular superior. Então a matriz cuja transposta é igual ao produto de todas as matrizes $P_{i,j}$ aplicadas para escalonamento da matriz A é uma matriz ortogonal Q na QR decomposição da matriz A. O algoritmo seqüencial básico para o método de *Givens* é apresentado na figura 5.3. Já uma alternativa para o algoritmo paralelo é apresentada na figura 5.4 através de uma programação do tipo mestre-escravo, da mesma forma como no método de *Householder*.

5.3 Conclusões do Capítulo

Os métodos apresentados neste capítulo são utilizados na resolução de aplicações reais de grande porte. Versões seqüênciais desses métodos foram implementadas em C/C++

Mestre	Escravos
1 - Divide a matriz em colunas 2 - Envia para os escravos	3 - Recebe as colunas do mestre
faça 1 até a ordem	
	4 - O escravo ativo calcula os vetores dos senos e co-senos e envia aos escravos a direita 5 - Os escravos a direita do escravo ativo recebem os vetores dos senos e cossenos
	6 - Altera sua matriz usando os vetores de senos e cossenos
Desconsidera primeira linha e primeira coluna (diminuindo a ordem do sistema) Desativa dinamicamente os processadores não necessários Fim do laço	
8 - Recebe as matrizes calculadas pelos escravos	7 - Envia sua matriz para o mestre
Calcula vetor resultado	

Figura 5.4: Algoritmo Paralelo para o Método de *Givens*

e em C–XSC (usando o recurso da alta exatidão). Os resultados dessas implementações estão descritas no capítulo 6. Com base nessas versões iniciais será possível, no futuro, a implementação em paralelo desses métodos com a característica da alta exatidão e a realização de estudos visando a inclusão das demais características da Computação Verificada nessas versões intervalares. Como já foi colocado neste capítulo, esses métodos foram escolhidos para auxiliar na validação do ambiente que está sendo proposto e por essas aplicações serem objeto de estudos por parte dos grupos de pesquisa com os quais essa tese se relaciona.

6 ANÁLISE DOS TESTES

O uso efetivo do C–XSC em ambientes do tipo *cluster* de computadores era de fundamental importância para a continuidade desta pesquisa, pois esse fator possibilitaria as futuras implementações dos métodos para a resolução de sistemas lineares. Para se garantir o seu efetivo funcionamento, algumas atividades foram necessárias serem desenvolvidas em um primeiro momento durante esta tese de doutorado. Como foi apresentado na seção 3.4, os processos de instalação do C–XSC nos *clusters* e de sua integração com a biblioteca MPI necessitavam ser realizados. Após esses processos observou-se a necessidade de testar se o C–XSC estava funcionando de maneira correta neste ambiente computacional. Para tanto, foi necessária a realização de um conjunto de testes com o objetivo de avaliar o desempenho do C–XSC e validar o uso de seus tipos de dados especiais no ambiente disponibilizado. Esses testes foram divididos em quatro categorias:

- Testes dos *solvers* verificados: testes referentes a qualidade do resultado obtido com *solvers* verificados para matrizes densas e bandas - seção 6.1;
- Testes básicos: cálculo do produto escalar e da multiplicação de matrizes - seção 6.2;
- Testes dos métodos para sistemas lineares: testes referentes aos métodos implementados para a resolução de sistemas lineares relacionadas a aplicações reais - seção 6.3;
- Testes complementares: testes adicionais que abordam a implementações de alguns métodos tradicionais para a resolução de sistemas lineraes - seção 6.4.

Os testes realizados visaram a análise de desempenho e da exatidão do programas implementados, de acordo com o tipo de teste, em C/C++, em C/C++ usando MPI, C/C++ usando C–XSC e C/C++ usando C–XSC e MPI. Essas comparações foram necessárias para possibilitar a clara noção de quando e quanto pode ser aceitável a perda do desempenho em prol de um ganho de exatidão. Além disso, é de fundamental importância ao usuário dessas ferramentas que ele tenha a plena consciência dos pontos críticos de sua aplicação (em termos de exatidão) podendo, com isso, nesses pontos, utilizar as ferramentas que estão sendo desenvolvidas e disponibilizadas por esta pesquisa.

6.1 *Solvers* para sistemas lineares

No início desta pesquisa foram implementados *solvers* para a resolução de sistemas lineares com matrizes densas e bandas (*solver LSS – Linear System Solver* e *solver BAND*,

respectivamente). Os *solvers* desenvolvidos implementaram métodos pontuais e intervalares seqüenciais para a resolução de sistemas lineares, conforme apresentado no capítulo 4. Estes *solvers* serviram, principalmente, para avaliar (com resultados satisfatórios) a boa exatidão fornecida pela biblioteca C–XSC no tratamento de problemas sujeitos a instabilidade numérica e erros de arredondamento e cancelamento.

Para comprovar a qualidade dos resultados gerados pelos *solvers LSS* e *BAND*, foram selecionados alguns problemas clássicos que serão apresentados nas próximas seções[1].

6.1.1 Matrizes de Hilbert

Um exemplo muito conhecido de matrizes mal-condicionadas são as chamadas *matrizes de Hilbert*, cujas entradas verificam a relação apresentada em (6.1).

$$(H_n)_{i,j} = \frac{1}{i+j-1} \tag{6.1}$$

O *LSS* resolveu o sistema $(mmc_{10}H_{10})x = (mmc_{10}\epsilon_1)$, cujo vetor-solução é a primeira coluna da inversa de H_{10}, conforme (6.2):

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \end{pmatrix} = \begin{pmatrix} \text{1.00000000E+002}, & \text{1.00000000E+002} \\ \text{-4.95000000E+003}, & \text{-4.95000000E+003} \\ \text{7.92000000E+004}, & \text{7.92000000E+004} \\ \text{-6.00600000E+005}, & \text{-6.00600000E+005} \\ \text{2.52252000E+006}, & \text{2.52252000E+006} \\ \text{-6.30630000E+006}, & \text{-6.30630000E+006} \\ \text{9.60960000E+006}, & \text{9.60960000E+006} \\ \text{-8.75160000E+006}, & \text{-8.75160000E+006} \\ \text{4.37580000E+006}, & \text{4.37580000E+006} \\ \text{-9.23780000E+005}, & \text{-9.23780000E+005} \end{pmatrix} \tag{6.2}$$

Da mesma forma, o *LSS* resolveu o sistema $(mmc_{20}H_{20})x = (mmc_{20}\epsilon_1)$, cujo vetor-solução é a primeira coluna da inversa de H_{20}, conforme (6.3):

[1] Devido ao fato da biblioteca C–XSC simular a Computação Verificada via *software*, os resultados numéricos obtidos são os mesmos, independentemente do ambiente computacional utilizado.

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{15} \\ x_{16} \\ x_{17} \\ x_{18} \\ x_{19} \\ x_{20} \end{pmatrix} = \begin{pmatrix} 3.999999999999999E+002, & 4.000000000000001E+002 \\ -7.980000000000002E+004, & -7.979999999999998E+004 \\ 5.266799999999999E+006, & 5.266800000000001E+006 \\ -1.716099000000001E+008, & -1.716098999999999E+008 \\ 3.294910079999999E+009, & 3.294910080000001E+009 \\ -4.118637600000001E+010, & -4.118637599999999E+010 \\ 3.569485919999999E+011, & 3.569485920000001E+011 \\ -2.237302782000001E+012, & -2.237302781999999E+012 \\ 1.044074631599999E+013, & 1.044074631600001E+013 \\ -3.700664527560001E+013, & -3.700664527559999E+013 \\ 1.009272143879999E+014, & 1.009272143880001E+014 \\ -2.133234304110001E+014, & -2.133234304109999E+014 \\ 3.500692191359999E+014, & 3.500692191360001E+014 \\ -4.443186242880001E+014, & -4.443186242879999E+014 \\ 4.316238064511999E+014, & 4.316238064512001E+014 \\ -3.147256922040001E+014, & -3.147256922039999E+014 \\ 1.666194841079999E+014, & 1.666194841080001E+014 \\ -6.044040109800001E+013, & -6.044040109799999E+013 \\ 1.343120024399999E+013, & 1.343120024400001E+013 \\ -1.378465288200001E+012, & -1.378465288199999E+012 \end{pmatrix} \tag{6.3}$$

6.1.2 Inversão de Matrizes

A eficiência do *LSS* pode ser verificada também no que diz respeito à inversão de matrizes. O exemplo utilizado agora é aquele onde $A = \begin{pmatrix} 1 & 1 & 0 \\ 0 & \epsilon & 0 \\ 0 & 0 & \epsilon^2 \end{pmatrix}$, cuja $A^{-1} = \begin{pmatrix} 1 & -\frac{1}{\epsilon} & 0 \\ 0 & \frac{1}{\epsilon} & 0 \\ 0 & 0 & \frac{1}{\epsilon^2} \end{pmatrix}$. Utilizando $\epsilon = 1.084202172485504E-019$, foi obtido:

$$A = \begin{pmatrix} 1.0E+000 & 1.000000000000000E+000 & 0.000000000000000E+000 \\ 0.0E+000 & 1.084202172485504E-019 & 0.000000000000000E+000 \\ 0.0E+000 & 0.000000000000000E+000 & 1.175494350822288E-038 \end{pmatrix}$$

$$A^{-1} = \begin{pmatrix} 1.0E+000 & -1.000000000000000E+000 & 0.000000000000000E+000 \\ 0.0E+000 & 7.136238463529799E+044 & 0.000000000000000E+000 \\ 0.0E+000 & 0.000000000000000E+000 & 8.507059173023462E+037 \end{pmatrix}.$$

6.1.3 Sistema Linear Esparso de Grande Porte

Com o objetivo de testar a qualidade numérica do *solver BAND*, apresenta-se aqui um exemplo onde foi calculado a solução (intervalar) para um sistema linear esparso de grande porte (ordem $n = 200.000$). Utilizou-se a matriz simétrica *Toeplitz* com 5 bandas com os valores $1, 2, 4, 2, 1$ e definiu-se todos componentes de b iguais a 1. Então o *solver* produziu a seguinte saída para esse sistema (somente os dez primeiros e os dez últimos componentes da solução foram apresentados):

```
Dimensao   n = 200000
tamanho bandas l,k : 2 2
A = 1 2 4 2 1     troca elementos ? (s/n) n
b = =1            troca elementos ? (s/n) n

x =
   1: [   1.860146067479180E-001,   1.860146067479181E-001 ]
   2: [   9.037859550210300E-002,   9.037859550210302E-002 ]
   3: [   7.518438200412189E-002,   7.518438200412191E-002 ]
   4: [   1.160876404875081E-001,   1.160876404875082E-001 ]
   5: [   1.003153932563721E-001,   1.003153932563722E-001 ]
   6: [   9.427129202687645E-002,   9.427129202687647E-002 ]
   7: [   1.028361799416204E-001,   1.028361799416205E-001 ]
   8: [   1.005240450090008E-001,   1.005240450090009E-001 ]
   9: [   9.874921290539136E-002,   9.874921290539138E-002 ]
  10: [   1.004617422430963E-001,   1.004617422430964E-001 ]

199990: [ 1.001953939326196E-001,   1.001953939326197E-001 ]
199991: [ 1.004617422430963E-001,   1.004617422430964E-001 ]
199992: [ 9.874921290539136E-002,   9.874921290539138E-002 ]
199993: [ 1.005240450090008E-001,   1.005240450090009E-001 ]
199994: [ 1.028361799416204E-001,   1.028361799416205E-001 ]
199995: [ 9.427129202687645E-002,   9.427129202687647E-002 ]
199996: [ 1.003153932563721E-001,   1.003153932563722E-001 ]
199997: [ 1.160876404875081E-001,   1.160876404875082E-001 ]
199998: [ 7.518438200412189E-002,   7.518438200412191E-002 ]
199999: [ 9.037859550210300E-002,   9.037859550210302E-002 ]
200000: [ 1.860146067479180E-001,   1.860146067479181E-001 ]

max. rel. error = 1.845833860422451E-016  em i = 3
max. abs. error = 2.775557561562891E-017  em i = 1

min.abs.:
x[3] = [7.518438200412189E-002, 7.518438200412191E-002]
max.abs.:
x[1] = [1.860146067479180E-001, 1.860146067479181E-001]
```

6.2 Testes Básicos

Os testes básicos abordaram o cálculo do produto escalar e da multiplicação de matrizes de forma seqüencial e paralela. Estes testes serviram para validar a integração da biblioteca C–XSC com a biblioteca MPI e para verificar o comportamento nos *clusters* quando do envio dos dados especiais da biblioteca C–XSC, especialmente dos tipos de dados de alta exatidão (tipo `dotprecision`). Nas seções seguintes serão apresentados

os testes realizados e os resultados obtidos[2].

6.2.1 Cálculo do Produto Escalar

Para o cálculo do produto escalar foram implementados testes comparando a exatidão e desempenho entre programas desenvolvidos em C/C++ (produto escalar – PE) e C/C++ usando C–XSC (produto escalar ótimo – PEO). Para a obtenção dos resultados apresentados neste capítulo foram realizados 3 (três) testes, conforme a descrição a seguir:

- Teste 1: foram utilizados vetores (a e b) de ordem (n) igual a 1000, 5000 e 10000, sendo $a = [0.6,\ldots,0.6]$ e $b = [0.24,\ldots,0.24]$. Os resultados são apresentados nas tabelas 6.1 e 6.2. Os resultados referem-se a implementação seqüencial.

Tabela 6.1: Teste 1 – Resultados do produto escalar em C/C++

Vetor (n)	Resultado Obtido	Resultado Exato
1000	144.00000000000250111O429875553	144
5000	720.00000000002432898327242583O	720
10000	1440.00000000051613824325613680	1440

Tabela 6.2: Teste 1 – Resultados do produto escalar em C–XSC

Vetor (n)	Resultado Obtido	Resultado Exato
1000	144	144
5000	720	720
10000	1440	1440

- Teste 2: foram utilizados vetores (a e b) de ordem n igual a 10000, 20000 e 30000, sendo $a = [0.8,\ldots,0.8]$ e $b = [0.1,\ldots,0.1]$. Os resultados são apresentados nas tabelas 6.4 e 6.5. Os resultados também referem-se a implementação seqüencial. A tabela 6.3 e a figura 6.1 apresentam os tempos de execução obtidos com este teste para vetores com ordem n igual a 1000, 5000, 10000, 20000 e 30000.

Tabela 6.3: Teste 2 – Cálculo do produto escalar seqüencial – tempo em μsegundos

Tamanho do Vetor (n)	programa em C/C++	programa em C–XSC	Fator
1000	20	449	22.45
5000	97	2217	22.86
10000	166	4325	$26,05$
20000	309	8460	27.38
30000	464	12644	27.25

- Teste 3: foram utilizados vetores (a e b) de ordem (n) igual a 30000, 90000 e 180000, sendo
  $a = [10^{50}, 1.25, 10^{50}, 1.1, \ldots, 10^{50}, 1.25, 10^{50}, 1.1]$ e $b = [1, 1, -1, -1, \ldots, 1, 1, -1, -1]$.
  Os resultados apresentados nas tabelas 6.6, 6.7 e 6.8 e na figura 6.2 referem-se a implementação paralela do produto escalar (utilizando 4 nodos do labtec).

[2]Os testes para o cálculo do produto escalar e para a multiplicação de matrizes foram realizados no *cluster* labtec do II-UFRGS.

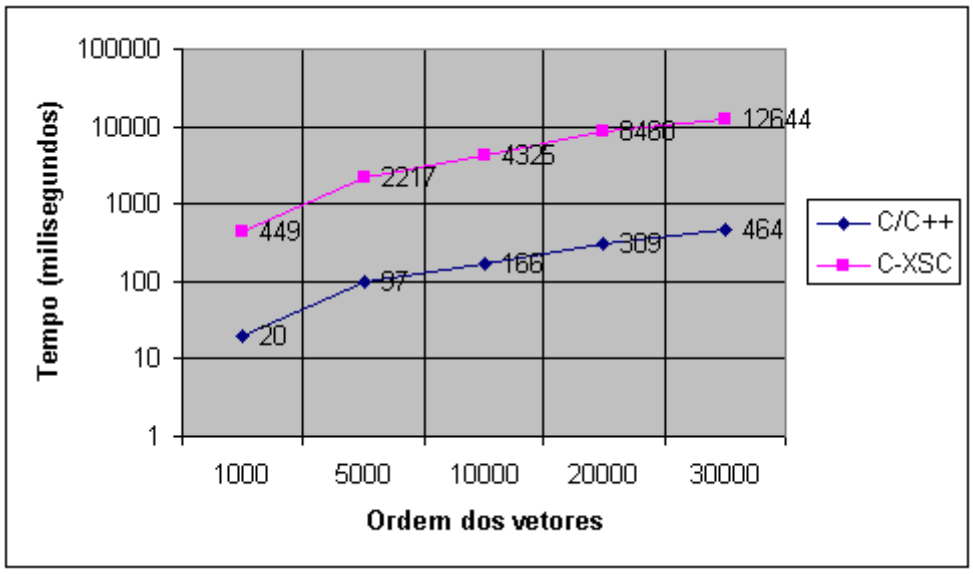

Figura 6.1: Cálculo do Produto Escalar Seqüencial no labtec

Tabela 6.4: Teste 2 – Resultados do produto escalar em C/C++

Vetor (n)	Resultado Obtido	Resultado Exato
10000	800.000000001140278982347808778285980224609	800
20000	1599.999999997046415955992415547370910644 53	1600
30000	2399.999999989772732078563421964645385742 19	2400

Tabela 6.5: Teste 2 – Resultados do produto escalar em C–XSC

Vetor (n)	Resultado Obtido	Resultado Exato
10000	800.000000000000113686837721616029739379 8828	800
20000	1600.000000000002273736754432320594787597 66	1600
30000	2400.000000000004547473508864641189575195 31	2400

Tabela 6.6: Teste 3 – Cálculo do Produto escalar em paralelo– tempos em segundos

Vetor (n)	programa em C/C++	programa em C–XSC	Fator
30000	0.03120470	0.04101492	1.31
90000	0.09672686	0.11384722	1.18
180000	0.18867004	0.22508256	1.19

Tabela 6.7: Teste 3 – Resultados do produto escalar em C/C++

Vetor (n)	Resultado Obtido	Resultado Exato
30000	$-3.300000000000000266453525910037569701671 6$	1125
90000	$-3.300000000000000266453525910037569701671 6$	3375
180000	$-3.300000000000000266453525910037569701671 6$	6750

Os valores dos elementos dos vetores nos testes 1 e 2 foram utilizados por serem valores que não são exatamente representáveis em binário, o que proporcionam a esta pesquisa uma avaliação de exatidão dos resultados gerados (para esse tipo específico de problema). Já para o teste 3 foram utilizados valores com grandezas muito diferentes, o que pode ocasionar, de acordo com a ordem na qual esses valores são operados, problemas na qua-

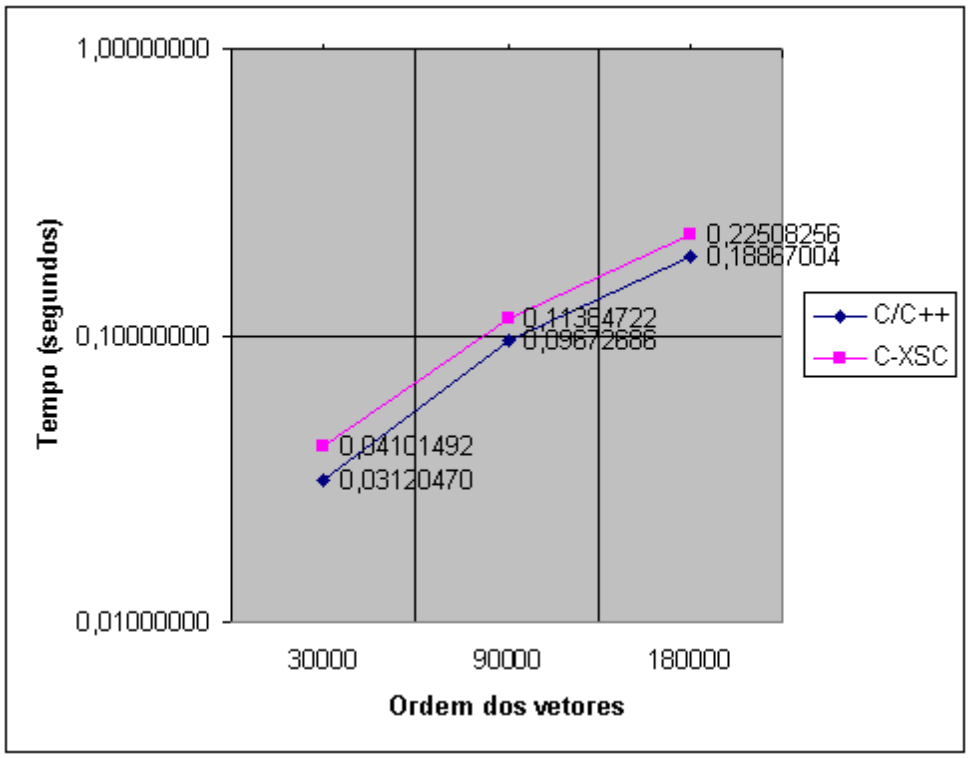

Figura 6.2: Cálculo do Produto Escalar em Paralelo no labtec

Tabela 6.8: Teste 3 – Resultados do produto escalar em C–XSC

Vetor (n)	Resultado Obtido	Resultado Exato
30000	1124.99999999999317878973670303821563720703	1125
90000	3374.99999999999818101059645414352416992l875	3375
180000	6749.99999999999636202119290828704833984375O	6750

lidade do resultado gerado por causa de erros de cancelamento. Nas versões paralelas dos testes implementados foi utilizado o modelo mestre-escravo, onde o mestre inicializa e distribui os vetores de forma igual a todos os escravos. Os escravos, por sua vez, recebem esses pedaços de vetores, calculam os produtos parciais e devolvem o resultado parcial para o mestre. O mestre junta esses pedaços e devolve o resultado. Vale lembrar que, no caso do C–XSC, o que ocorre é um produto escalar ótimo, ou seja, ocorre apenas um arredondamento em todo o cálculo realizado.

Além dos testes do cálculo do produto escalar realizados no *cluster* labtec, foram realizados no *cluster* Colorado dois testes do cálculo do produto escalar implementados em C–XSC, utilizando dados de entrada do tipo real (*real*) e do tipo intervalo (*interval*).

Tabela 6.9: Cálculo do Produto Escalar no *Cluster* Colorado (dados reais do C–XSC) – tempos em segundos

Vetor (n)	seqüencial	$np = 3$	$np = 4$
45000	0.02515	0.41527	0.40388
90000	0.04828	0.75191	0.73996
180000	0.09513	1.50062	1.41782
360000	0.18867	2.96064	2.88552

O objetivo desses testes, além de demonstrar o desempenho dos programas seqüenciais e paralelos implementados em C–XSC, foi o de validar nesse tipo de ambiente o envio/recebimento dos tipos de dados especiais do C–XSC (em especial os tipos de dados intervalo (*interval*) e intervalos de alta exatidão (*idotprecision*)), conforme descrito na seção 3.4.

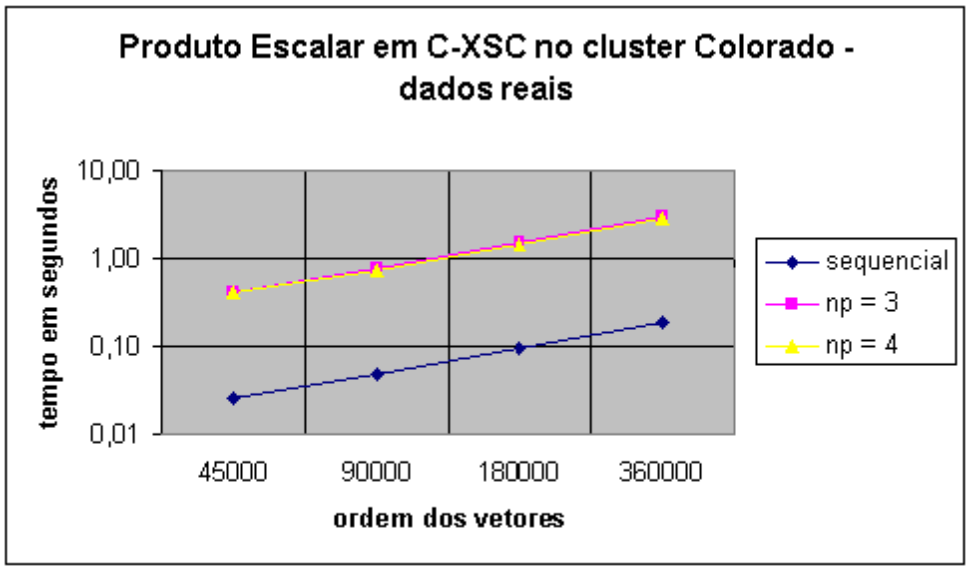

Figura 6.3: Cálculo do Produto Escalar no *Cluster* Colorado (dados reais do C–XSC)

Tabela 6.10: Cálculo do Produto Escalar no *Cluster* Colorado (dados intervalares do C–XSC) – tempos em segundos

Vetor (n)	seqüencial	$np = 3$	$np = 4$
45000	0.04777	0.47632	0.52525
90000	0.09412	0.95668	0.92494
180000	0.18699	1.93502	1.84891
360000	0.36927	3.85463	3.42321

Os resultados dos testes realizados no *cluster* Colorado são apresentados nas tabelas 6.9 e 6.10 e nas figuras 6.3 e 6.4. Nestes testes foram usados vetores de tamanho 45000, 90000, 180000 e 360000 e foram implementados programas seqüenciais e paralelos utilizando 3 e 4 processadores, utilizando a abordagem mestre-escravo. Pode-se constatar que a diferença de tempo apresentada entre os programas que utilizavam dados reais e os que utilizavam dados intervalares, tanto nas implementações seqüências como nas paralelas, deve-se ao fato de que o tipo de dado intervalo é composto de dois números reais, utilizados para representar o extremo inferior e superior do intervalo. Em compensação, o comportamento obtido por ambas as implementações é semelhante.

6.2.2 Multiplicação de Matrizes

Os testes abordando a multiplicação de matrizes foram realizados através do desenvolvimento de programas seqüenciais (C/C++ versus C/C++ usando C–XSC) e paralelos

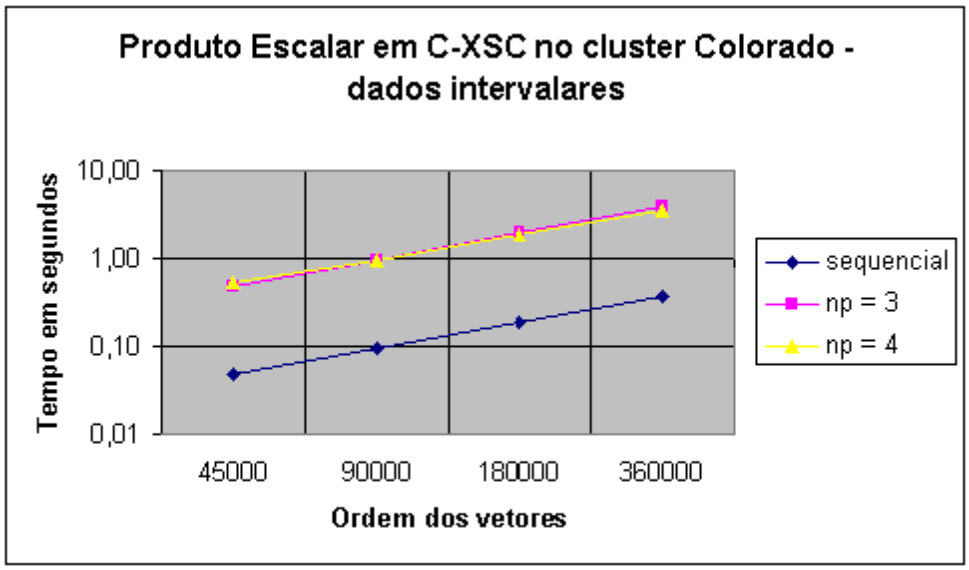

Figura 6.4: Cálculo do Produto Escalar no *Cluster* Colorado (dados intervalares do C–XSC)

(C/C++ usando MPI versus C/C++ usando C–XSC e MPI). Nesses testes foi implementada a multiplicação convencional de matrizes em versão seqüencial e paralela. A versão paralela seguiu um algoritmo de mestre e escravo, onde o mestre divide e distribui a matriz para os escravos (em linha). Por exemplo, se deseja-se multiplicar A por B, a matriz A é divida entre os escravos e a matriz B é enviada a todos os processos por *broadcast*. Foram utilizados 1 mestre e 7 escravos. Caso a divisão da matriz não seja exata, o mestre fica com a sobra. Após o cálculo de cada produto parcial, o mesmo é enviado para o mestre que monta a matriz resultante. As matrizes foram inicializadas conforme apresentadas em (6.4) considerando, como exemplo, matrizes de dimensão 4×4.

$$C = \begin{pmatrix} 10^{50} & 1.25 & 10^{50} & 1.1 \\ 10^{50} & 1.25 & 10^{50} & 1.1 \\ 10^{50} & 1.25 & 10^{50} & 1.1 \\ 10^{50} & 1.25 & 10^{50} & 1.1 \end{pmatrix} \times \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 \end{pmatrix} \tag{6.4}$$

Analisando a operação acima, percebe-se que, na matriz resultante (C), cada um dos elementos será $4 \times (10^{50} + 1.25 - 10^{50} - 1.1) = 4 \times 0,15 = 0,6$. Generalizando esse cálculo, como 10^{50} aparece duas vezes, tem-se que $1.25 - 1.1$ ocorre em $1/4$ de N nos cálculos, então o resultado esperado para uma multiplicação de uma matriz $N \times N$ é

$$\frac{N}{4} \times 0.15 \tag{6.5}$$

considerando N múltiplo de 4, ou seja, cada um dos elementos da matriz resultante será $\frac{N}{4} \times 0.15$. Portanto, segundo 6.5, o resultado exato esperado para 256 é uma matriz preenchida com 9.6, para 512 é 19.2, para 1024 é 38.4 e para 2048 é 76.8. Os resultados obtidos nos cálculos podem ser encontrados na tabela 6.11. Tanto a versão paralela quanto a seqüencial apresentaram a mesma exatidão, como era de se esperar. Para esses testes foram utilizadas matrizes quadradas de dimensões 512, 1024 e 2048. Foram realizadas 50 execuções da versão seqüencial e 10 execuções das versões paralelas, e o resultado, em

relação à média do tempo de execução, é apresentado nas tabelas 6.12 e 6.13 e nas figuras 6.5 e 6.6, respectivamente.

Tabela 6.11: Resultados da Multiplicação de Matrizes

Dimensão	Programa	Resultado Obtido
256	C/C++	$-1.10000000000000008881784197001252323389053$
	C–XSC	$9.59999999999994315658113919198513031005859$
512	C/C++	$-1.10000000000000008881784197001252323389053$
	C–XSC	$19.19999999999998863131622783839702606201172$
1024	C/C++	$-1.10000000000000008881784197001252323389053$
	C–XSC	$38.39999999999997726263245567679405212402344$

Tabela 6.12: Multiplicação de Matrizes em Seqüencial – tempos em segundos

Programa/Dimensão	512	1024	2048
C/C++	4.603131	34.13111	265.2844
C–XSC	115.5238	916.2339	7329.415
Fator	25.10	26.84	27.63

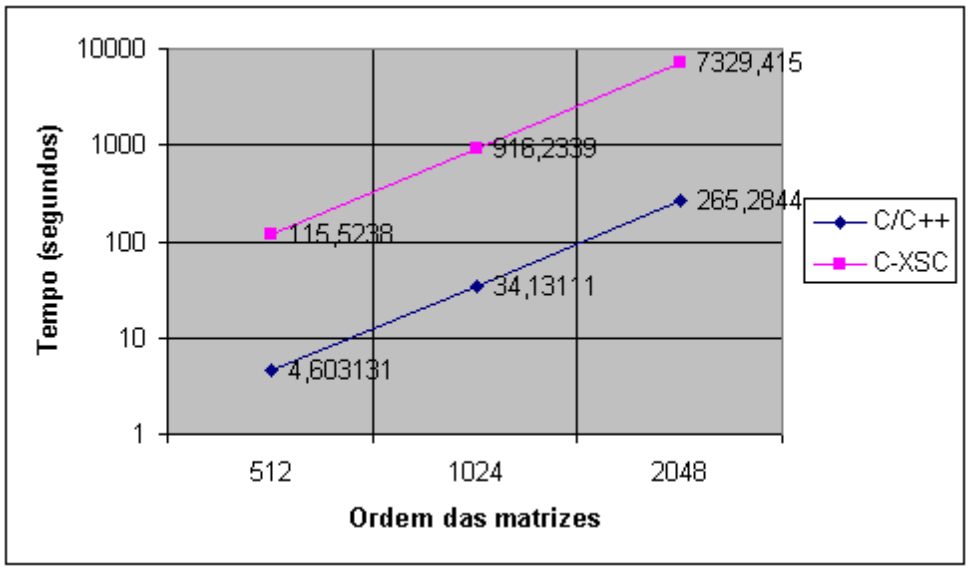

Figura 6.5: Multiplicação de Matrizes em Seqüencial no labtec

Tabela 6.13: Multiplicação de Matrizes em Paralelo – tempos em segundos

Programa/Dimensão	256	512	1024
C/C++	1.63666746	19.58703858	150.6339381
C–XSC	9.02638456	83.00522426	664.0793321
Fator	5.52	4.24	4.41

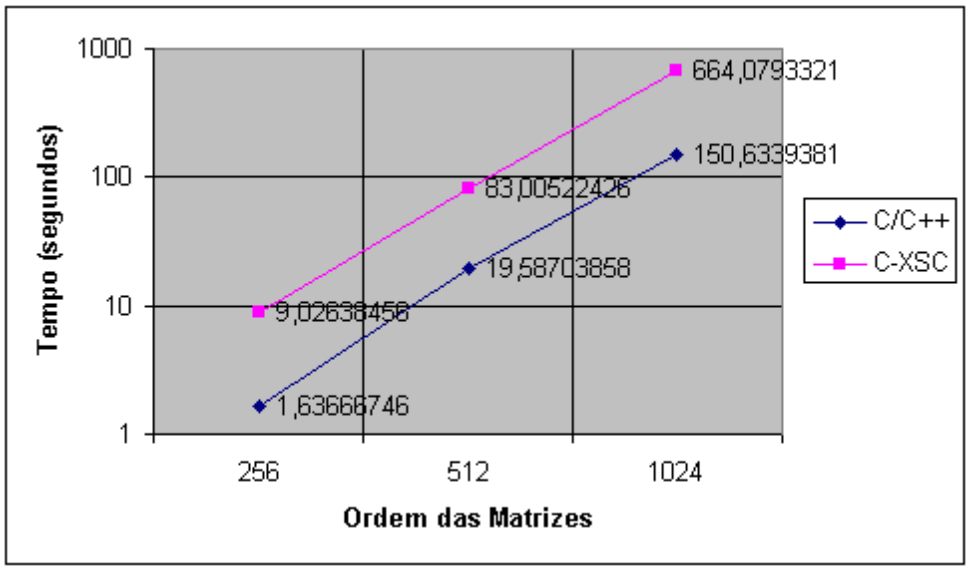

Figura 6.6: Multiplicação de Matrizes em Paralelo no labtec

6.3 Métodos para a Resolução de Sistemas Lineares

Os testes que abordaram métodos para a resolução de sistemas de equações lineares foram realizados no *cluster* Colorado da UPF. Foram implementados programas seqüenciais referentes aos métodos Gradiente Conjugado, Householder e Givens com e sem a característica da alta exatidão e os resultados obtidos, em termos de desempenho, seguem a mesma linha de comportamento dos resultados já obtidos através dos testes com o produto escalar e a multiplicação de matrizes. Os resultados referentes ao método do Gradiente Conjugado são apresentados na figura 6.7 e na tabela 6.14. A figura 6.8 e a tabela 6.15 apresentam os resultados obtidos com a implementação do método de Householder. Por fim, a figura 6.9 e a tabela 6.16 apresentam os resultados obtidos com a implementação do método de Givens.

Tabela 6.14: Método Gradiente Conjugado seqüencial – tempo em segundos

Tamanho da Matriz (n)	programa em C/C++	programa em C–XSC	Fator
512	0.04981	0.70404	14.13
1024	0.19552	2.79346	14.29
2048	0.77445	11.14190	14.39
4096	3.07566	45.31080	14.73

Tabela 6.15: Método Householder seqüencial – tempo em segundos

Tamanho da Matriz (n)	programa em C/C++	programa em C–XSC	Fator
512	4.45	20.47	4.60
1024	40.33	169.68	4.20
2048	345.34	1376.28	3.98

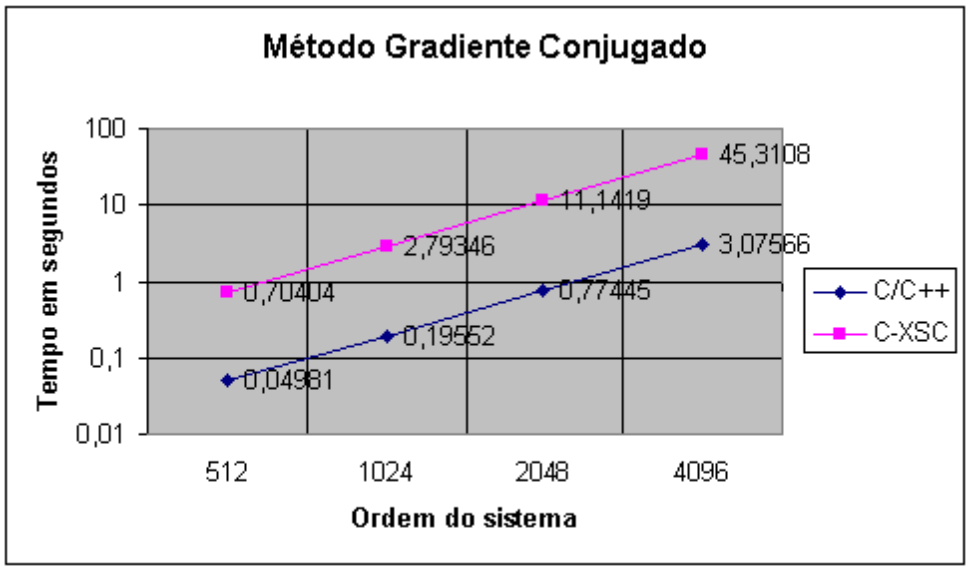

Figura 6.7: Método Gradiente Conjugado em Seqüencial no Colorado

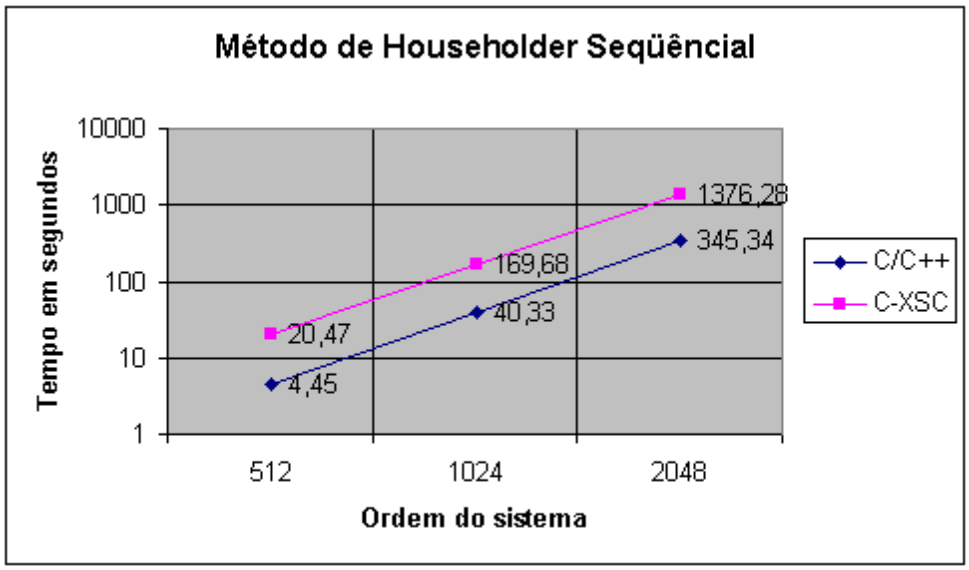

Figura 6.8: Método Householder em Seqüencial no Colorado

Tabela 6.16: Método Givens seqüencial – tempo em segundos

Tamanho da Matriz (n)	programa em C/C++	programa em C–XSC	Fator
512	4.55	17.27	3.89
1024	36.98	137.43	3.71
2048	310.52	1103.56	3.55

6.4 Outros Testes

Além dos testes apresentados das seções anteriores deste capítulo, algumas pesquisas realizadas pelo grupo ComPaDi da UPF estão em andamento e também abordam o uso

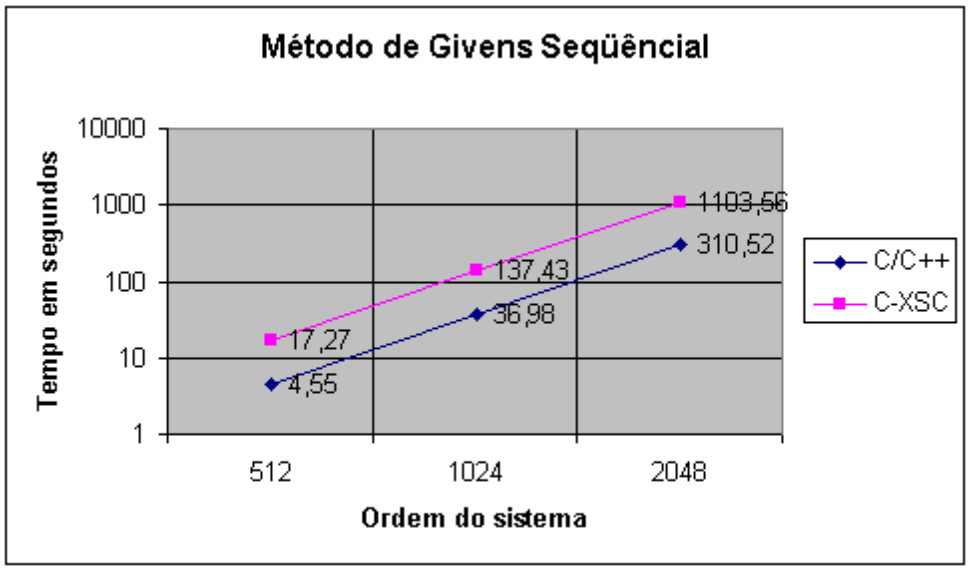

Figura 6.9: Método de Givens em seqüencial no Colorado

da biblioteca C–XSC na resolução de problemas da álgebra linear, em especial a resolução de sistemas lineares em *clusters* de computadores. Atualmente, essas pesquisas estão voltadas para a implementação de versões seqüências e com alta exatidão de um grupo de métodos numéricos (Eliminação de Gauss, Gauss-Seidel, Gauss-Jacobi e Decomposição LU). Seus objetivos são os de paralelizar esses métodos e, em uma etapa posterior, incluir a aritmética intervalar entre as características dos métodos implementados. Isto se deve ao fato de que existem algumas aplicações reais como, por exemplo, o controle de fluxo de potência em redes elétricas (BARBOZA; DIMURO; REISER, 2004) que fazem o uso desses métodos e que, atualmente, já estão sendo solucionadas através do uso da aritmética intervalar só que ambiente seqüencial e utilizando a biblioteca intervalar IntLab – ***Int****erval* ***Lab****oratory)* (RUMP, 1998, 1999). As figuras 6.10, 6.11, 6.12 e 6.13 apresentam alguns dos resultados já obtidos até o momento com a implementação dos métodos citados no parágrafo anterior[3].

É importante ressaltar que essas pesquisas já vem trabalhando, também, com o uso da biblioteca BLAS (***B****asic* ***L****inear* ***A****lgebra* ***S****ubroutines*) na resolução de problemas numéricos, da utilização das funções SSE (***S****treaming* ***SIMD E****xtensions*) e do uso das diretivas de otimização do compilador gcc/g++, o que pode ser útil em uma das linhas de pesquisa que serão seguidas com o término deste trabalho que é a da otimização da biblioteca C–XSC. Os resultados de um *benchmark* realizado para a biblioteca C–XSC a respeito do custo computacional das operações aritméticas básicas utilizando o dado *real* (com arredondamentos direcionados) em comparação ao tipo de dado *double* do C/C++, incluindo o uso das diretivas de otimização do compilador gcc/g++, são apresentados no apêndice A.

[3]Os testes apresentados nesta seção foram realizados no *cluster* Colorado da UPF.

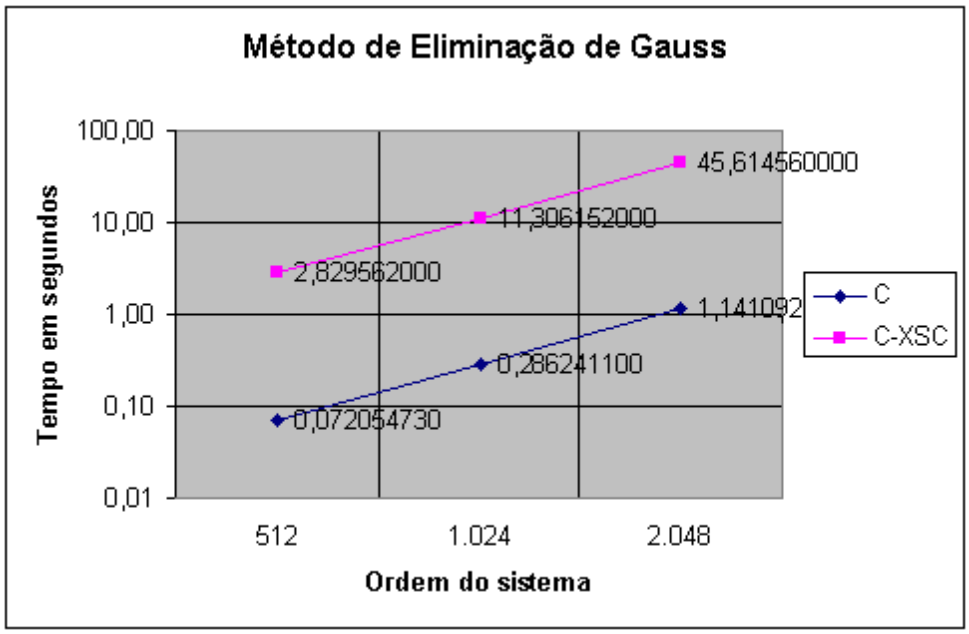

Figura 6.10: Método de Eliminação de Gauss – tempo em segundos

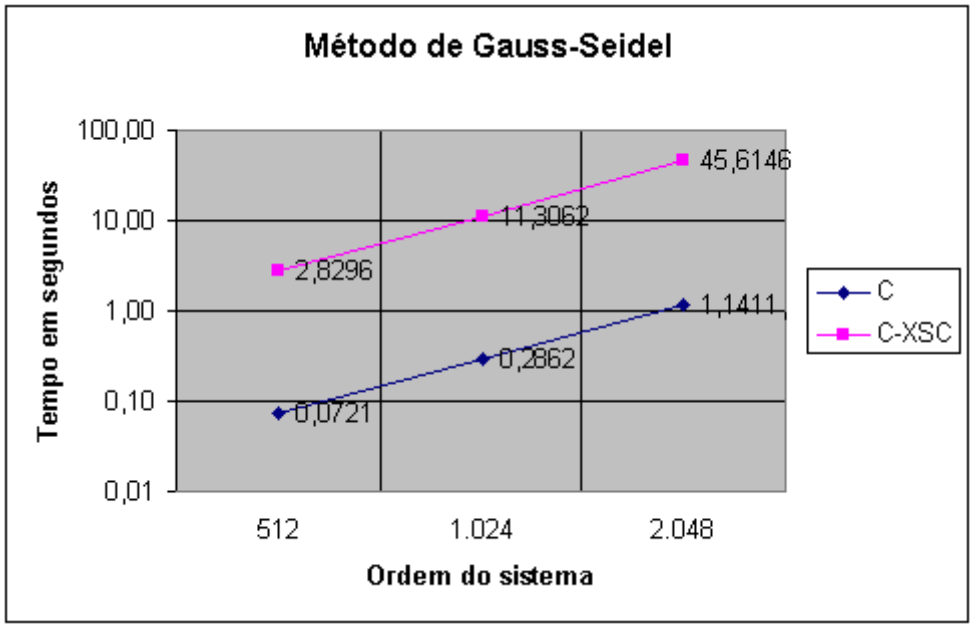

Figura 6.11: Método de Gauss-Seidel – tempo em segundos

6.5 Conclusões do Capítulo

Através dos testes realizados pôde-se observar que o uso da biblioteca C–XSC gera uma perda de desempenho em comparação as implementações tradicionais em C/C++, entretanto, não pode-se negar a qualidade numérica da mesma, em especial quando se trabalha com problemas instáveis ou sujeitos a erros de arredondamento ou de cancelamento. Outro fato observado é que, conforme os gráficos de tempo apresentados neste capítulo, pode-se observar que o comportamento dos tempos de execução dos programas implementados em C–XSC é semelhante aos tempos de execução obtidos com imple-

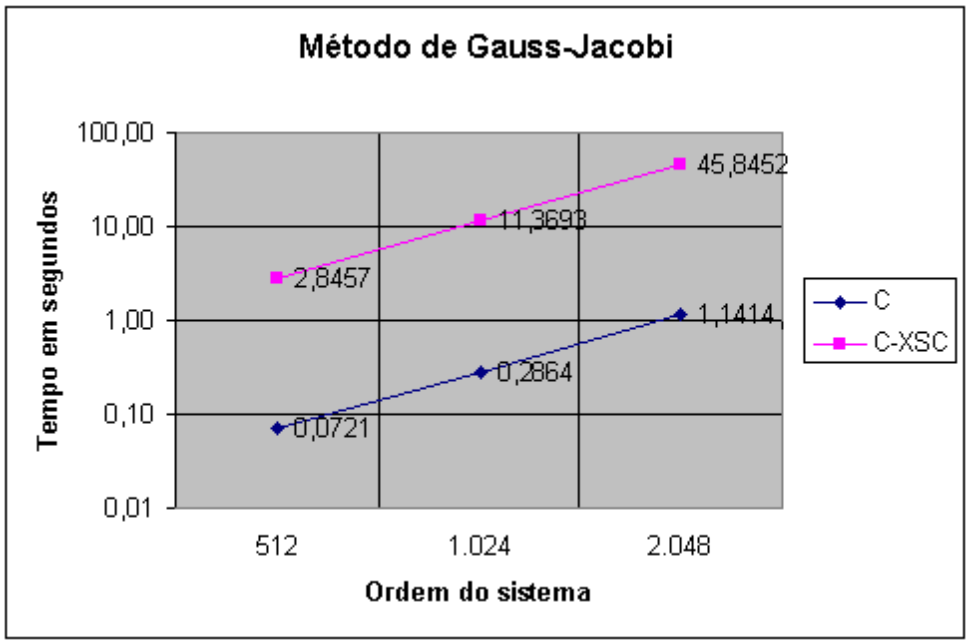

Figura 6.12: Método de Gauss-Jacobi – tempo em segundos

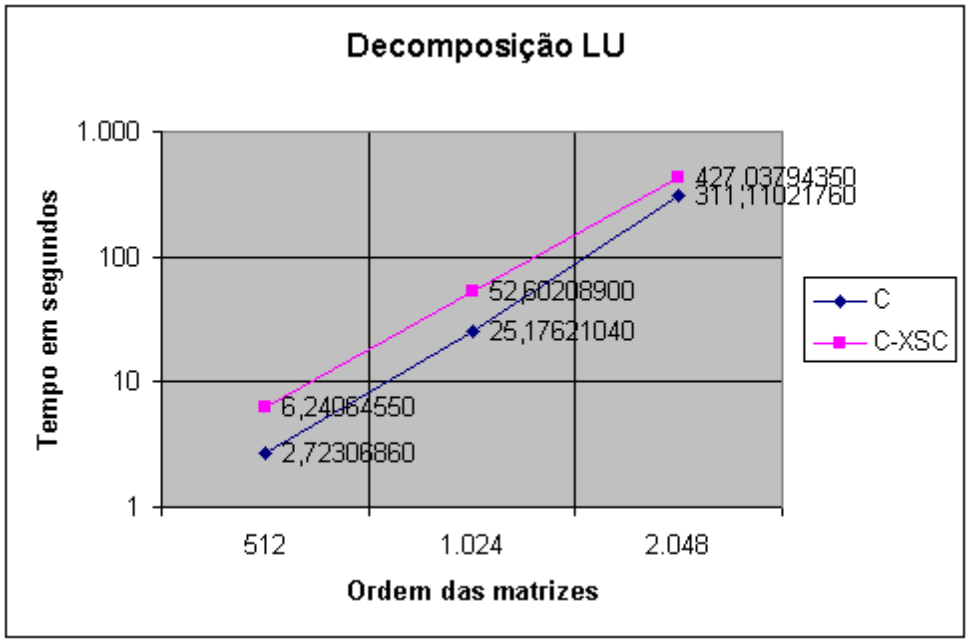

Figura 6.13: Decomposição LU – tempo em segundos

mentações tradicionais em C/C++ (testes em seqüencial e em paralelo). Isto nos leva a concluir que o fato de se trabalhar com a alta exatidão (dados *dotprecision* do C–XSC) não influi no modo de realização dos cálculos, apenas acarreta essa perda de desempenho devido ao tamanho das variáveis de alta exatidão que necessitam ser manipuladas pelo computador. Com isso, entende-se que um trabalho visando a otimização da biblioteca C–XSC e, em especial, das funções de somatório e de produto escalar poderá trazer uma melhora no desempenho, sem prejuízo a qualidade numérica do resultado. Outra alternativa nesta mesma linha de atuação, principalmente quando se trabalha com intervalos,

seria a melhoria do desempenho quando se necessita trocar o modo de arredondamento do compilador, conforme pode ser observado nos resultados apresentados no apêndice A. Alguns dos principais resultados obtidos neste trabalho foram publicados em (HÖLBIG et al., 2005, 2004; HÖLBIG; CLAUDIO; DIVERIO, 2004; HÖLBIG et al., 2004,a,b,c).

7 CONCLUSÕES

Este trabalho visou a disponibilização de um ambiente de alto desempenho e de alta exatidão. O alto desempenho foi obtido através do uso de *clusters* de computadores e a alta exatidão através do uso da biblioteca C–XSC onde a alta exatidão na solução de um problema é obtida através da realização de cálculos intermediários sem arredondamentos como se fossem em precisão infinita, sendo que o resultado exato real difere do resultado obtido por apenas um único arredondamento. Para o teste do ambiente foram utilizados os *clusters* da UFRGS, da UPF e da Universidade de Wuppertal. Para a sua validação, um conjunto de testes foi realizado com o intuito de avaliar a exatidão e o desempenho obtidos com o uso da biblioteca C–XSC, bem como a verificação da utilização dos dados especiais do C–XSC em um ambiente computacional paralelo, objetivando que os resultados obtidos mantivessem a mesma qualidade quando de sua resolução em um ambiente de alto exatidão seqüencial.

Para a realização dessa pesquisa, primeiramente, foi realizada a instalação e a configuração da biblioteca C–XSC nos *clusters*, a sua integração com a biblioteca de troca de mensagens MPI e a implementação em paralelo de programas que utilizavam os dados especiais do C–XSC, mais precisamente os dados de alta exatidão *dotpresicion* (do tipo real e intervalar) e os tipos de dados intervalo e matrizes e vetores reais e intervalares. Todo esse processo foi realizado de maneira satisfatória possibilitando, com isso, que as características da Computação Verificada estivessem disponibilizadas aos usuários desses ambientes. Esse processo de uso e integração do C–XSC nos *clusters* foi descrito no capítulo 3.

Após essa etapa a pesquisa voltou-se para o desenvolvimento de *solvers* verificados para a resolução de sistemas de equações lineares. Esses *solvers* foram voltados para o uso com matrizes densas (*solver* LSS) e para matrizes esparsas do tipo banda (*solver* BAND). Esses *solvers* realizam o cálculo da solução dos sistemas lineares com garantia, através do uso da matemática intervalar. Algumas extensões desses *solvers* foram realizadas, visto que, originalmente, os dados de entrada eram somente do tipo real e os dados de saída intervalos reais. Essas extensões possibilitaram que os dados de entrada pudessem ser, também, do tipo intervalo real, complexo e intervalo complexo para o *solver* LSS e intervalo real para o *solver* BAND. No capítulo 4 foi realizado um detalhamento a respeito dos *solvers* implementados.

A seguir foram pesquisadas algumas aplicações que estão sendo trabalhadas pelo grupo GMCPAD da UFRGS. Essas aplicações referem-se a hidrodinâmica e a aeração de grãos em silos de armazenagem. Através deste estudo pode-se selecionar três métodos para a resolução de sistemas de equações lineares e que são utilizados por essas aplicações (Gradiente Conjugado, Householder e Givens). Esses métodos foram descritos e implementados em C/C++ e em C–XSC (com alta exatidão). Os detalhes sobre as aplicações e

An Accurate and Efficient Selfverifying Solver for Systems with Banded Coefficient Matrix

Carlos Hölbig[a], Walter Krämer[b] and Tiarajú Diverio[c].

[a]Universidade de Passo Fundo and PPGC-UFRGS,
Passo Fundo - RS, Brazil
E-mail: holbig@upf.br

[b]University of Wuppertal,
Wuppertal, Germany
E-mail: kraemer@math.uni-wuppertal.de

[c]Instituto de Informática and PPGC at UFRGS,
Porto Alegre - RS, Brazil
E-mail: diverio@inf.ufrgs.br

In this paper we discuss a selfverifying solver for systems of linear equations $Ax = b$ with banded matrices A and the future adaptation of the algorithms to cluster computers. We present an implementation of an algorithm to compute efficiently componentwise good enclosures for the solution of a sparse linear system on typical cluster computers. Our implementation works with point as well as interval data (data afflicted with tolerances). The algorithm is implemented using C-XSC library (a C++ class library for extended scientific computing). Our goal is to provide software for validated numerics in high performance environments using C-XSC in connection with the MPICH library. Actually, our solver for linear system with banded matrices runs on two different clusters: ALiCE[1] at the University of Wuppertal and LabTeC[2] at UFRGS. Our preliminary tests with matrix-matrix multiplication show that the C-XSC library needs to be optimized in several ways to be efficient in a high performance environment (up to now the main goal of C-XSC was functionality and portability, not speed). This research is based on a joint research project between German and Brazilian universities (BUGH, UKA, UFRGS and PUCRS) [5].

1. Introduction

For linear systems where the coefficient matrix A has band structure the general algorithm for solving linear systems with dense matrices is not efficient. Since an approximate inverse R is used there this would result in a large overhead of storage and computation time, especially if the bandwidth of A is small compared with its dimension. To reduce this overhead, we will replace the approximate inverse by an approximate LU-decomposition of A which needs memory of the same order of magnitude only as A itself. Then we will have to solve systems with triangular banded matrices (containing point data) in interval arithmetic. This seems to be a trivial task and several methods have been developed using such systems and simple forward and backward substitution in interval arithmetic, see e.g. [8], [12]. However, at this point there appears suddenly a very unpleasant effect which makes the computed intervals blow up very rapidly in many cases. This effect is known in literature as *wrapping effect* (see e.g. [13]) and was recognized first in connection with the verified solution of ordinary initial value problems. However it is a common problem within interval arithmetic and may show

[1]Cluster with 128 Compaq DS10 Workstations, 616 MHz Alpha 21264 processors, 2 MB cache, Myrinet multistage crossbar connectivity, 1 TB disc space and 32 GB memory.
[2]Cluster with 20 Dual Pentium III 1.1 GHz (40 nodes), 1 GB memory RAM, HD SCSI 18 GB and Gigabit Ethernet. Cluster server (front-end) with Dual Pentium IV Xeon 1.8 GHz, 1 GB memory RAM, HD SCSI 36 GB and Gigabit Ethernet. LaBTeC Cluster Homapage: http://www.inf.ufrgs.br/LabTeC

up whenever computations in $\mathbb{IR}^n$, $n > 1$, are performed, e.g. if an interval vector is multiplied repeatedly by matrices or more general if a function $f : \mathbb{IR}^n \to \mathbb{IR}^n$ is applied repeatedly to an argument x, $x_n = f(f(\cdots f(f(x))\cdots))$. It should be noted that the wrapping effect is not related to roundoff errors or any other sources of errors in an algorithm (like truncation errors, discretization errors and the like) but it is introduced solely by interval arithmetic itself (though any additional errors may contribute to an increase of the wrapping effect).

2. The Algorithms

The algorithms implemented in our work were described in [9] and can be applied to any system of linear equations which can be stored in the floating point system on the computer. They will, in general, succeed in finding and enclosing a solution or, if they do not succeed, will tell the user so. In the latter case, the user will know that the problem is very ill conditioned or that the matrix A is singular.

In the implementation in C-XSC, there is the chance that if the input data contains large numbers or if the inverse of A or the solution itself contain large numbers, an overflow may occur, in which case the algorithms may crash. In practical applications, this has never been observed, however. This could also be avoided by including the floating point exception handling which C-XSC offers for IEEE floating point arithmetic [2].

For this work we implemented interval algorithms for solution of linear systems of equations with dense and sparse matrices. There are numerous methods and algorithms computing approximations to the solution x in floating-point arithmetic. However, usually it is not clear how good these approximations are, or if there exists a unique solution at all. In general, it is not possible to answer these questions with mathematical rigour if only floating-point approximations are used. These problems become especially difficult if the matrix A is ill conditioned.

We implemented some algorithms which answer the questions about existence and accuracy automatically once their execution is completed successfully. Even very ill conditioned problems can be solved with these algorithms. Most of the algorithms implemented here can be found in [14] and [15].

3. Band Matrices

Matrices with band structure and difference equations are closely related. There the difference equations could be rewritten equivalently as a linear system with band matrix which was triangular even. Similarly we can go in the other direction and write a triangular banded matrix as a difference equation.

The system

$$Ax = \begin{pmatrix} a_{1,1} & & & & \\ \vdots & \ddots & & 0 & \\ & & & & \\ a_{m,1} & & \ddots & & \\ & \ddots & & \ddots & \\ 0 & & a_{n,n-m+1} & \cdots & a_{n,n} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ \\ b_n \end{pmatrix} = b \tag{1}$$

is equivalent with the difference equation

$$a_{i,i-m+1}x_{i-m+1} + \cdots + a_{i,i}x_i = b_i \ , \quad i = m, \ldots, n, \tag{2}$$

of order $m-1$ with the starting values

$$x_i = (b_i - a_{i,i-1}x_{i-1} - \cdots - a_{i,1}x_1)/a_{i,i} \ , \quad i = 1, \ldots, m-1 \ . \tag{3}$$

In [9] we saw that the solution of triangular systems by interval forward substitution can result in severe overestimations. Therefore we use the relationship to difference equations and solve triangular banded systems with the method for difference equations which was presented in [9].

For general banded systems we will then apply a LU-decomposition without pivoting to the coefficient matrix A and derive an interval iteration similar to

$$[y]_{k+1} = R\diamondsuit(b - A\tilde{x}) + \diamondsuit(I - RA)[y]_k \tag{4}$$

Here, however, we will not use a full approximate inverse R but rather the iteration will be performed by solving two systems with banded triangular matrices (L and U).

A similar approach to banded and sparse linear systems can be found, e.g. , in [12]. There, however, the triangular systems were solved by interval forward and backward substitution which often results in gross overestimations as we have seen already.

For a different approach to the verified solution of linear systems with large banded or arbitrary sparse coefficient matrix see Rump, [15].

The mathematical background for the verified solution of large linear systems with band matrices is exactly the same as it was already in [6] for systems with dense matrices.

For dense systems the interval iteration (4) was derived by use of an approximate inverse R of the coefficient matrix A. This is however what we want to avoid for large banded matrices A. Therefore we chose a different approximate inverse, namely

$$R := (LU)^{-1} \approx A^{-1} \tag{5}$$

where

$$LU \approx A \tag{6}$$

is an approximate LU-decomposition of A without pivoting. Since we do not use pivoting both L and U are banded matrices again, and of course they are lower and upper triangular, resp.

The analogue of the iteration (4) now reads in our case

$$y_{k+1} = (LU)^{-1}(b - A\tilde{x}) + (I - (LU)^{-1}A)y_k \tag{7}$$

or, multiplying with LU and taking intervals:

$$LU[y]_{k+1} = \diamondsuit(b - A\tilde{x}) + \diamondsuit(LU - A)[y]_k. \tag{8}$$

Therefore we have to solve two linear systems with triangular, banded coefficient matrices, L and U, in order to compute $[y]_{k+1}$, i.e. to perform one step of the iteration (8). In each iteration we first compute an enclosure for the solution of

$$L[z]_{k+1} = \diamondsuit(b - A\tilde{x}) + \diamondsuit(LU - A)[y]_k$$

and then $[y]_{k+1}$ from

$$U[y]_{k+1} = [z]_{k+1}.$$

In both systems we do not use just plain interval forward or backward substitution, however, as discussed above, we treat the systems as difference equation and apply the corresponding method. Here again, as in [6], the inclusion test

$$[y]_{k+1} = F([y]_k) \subset [y]_k^\circ \tag{9}$$

has to be checked in the same way and if it is satisfied then the same assertions hold as in the dense case.

Remark:

If we compute the LU-decomposition with Crout's algorithm, then we can get the matrix $\diamondsuit(LU-A)$ virtually for free, since the scalar products which are needed here have to be computed in Crout's algorithm anyway.

4. Tests and Results

A very well known set of ill conditioned test matrices for linear system solvers are the $n \times n$ Hilbert matrices H_n with entries $(H_n)_{i,j} := \frac{1}{i+j-1}$. As a test problem, we report the results of our program for the linear systems $H_n x = e_1$, where e_1 is the first canonical unit vector. Thus the solution x is the first column of the inverse H_n^{-1} of the Hilbert matrix H_n. We give results for the cases $n = 10$ and

$n = 20$. Since the elements of these matrices are rational numbers which can not be stored exactly in floating point, we do not solve the given problems directly but rather we multiply the system by the least common multiple lcm_n of all denominators in H_n. Then the matrices will have integer entries which makes the problem exactly storable in IEEE floating point arithmetic. For $n = 10$, we have $lcm_{10} = 232792560$ and for $n = 20$, we have $lcm_{20} = 5342931457063200$.

For the system $(lcm_{10}H_{10})x = (lcm_{10}e_1)$, the program computes the result showed in (10), which is the exact solution of this ill conditioned system.

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \end{pmatrix} = \begin{pmatrix} [\ 1.000000000000000E+002,\ 1.000000000000000E+002] \\ [-4.950000000000000E+003, -4.950000000000000E+003] \\ [\ 7.920000000000000E+004,\ 7.920000000000000E+004] \\ [-6.006000000000000E+005, -6.006000000000000E+005] \\ [\ 2.522520000000000E+006,\ 2.522520000000000E+006] \\ [-6.306300000000000E+006, -6.306300000000000E+006] \\ [\ 9.609600000000000E+006,\ 9.609600000000000E+006] \\ [-8.751600000000000E+006, -8.751600000000000E+006] \\ [\ 4.375800000000000E+006,\ 4.375800000000000E+006] \\ [-9.237800000000000E+005, -9.237800000000000E+005] \end{pmatrix} \tag{10}$$

For the system $(lcm_{20}H_{20})x = (lcm_{20}e_1)$, the program computes the enclosures (here an obvious short notation for intervals is used) showed in (11), which is an extremely accurate enclosure for the exact solution (the exact solution components are the integers within the computed intervals).

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{15} \\ x_{16} \\ x_{17} \\ x_{18} \\ x_{19} \\ x_{20} \end{pmatrix} = \begin{pmatrix} [\ 3.999999999999999E+002,\ 4.000000000000001E+002] \\ [-7.980000000000002E+004, -7.979999999999998E+004] \\ [\ 5.266799999999999E+006,\ 5.266800000000001E+006] \\ [-1.716099000000001E+008, -1.716098999999999E+008] \\ [\ 3.294910079999999E+009,\ 3.294910080000001E+009] \\ [-4.118637600000001E+010, -4.118637599999999E+010] \\ [\ 3.569485919999999E+011,\ 3.569485920000001E+011] \\ [-2.237302782000001E+012, -2.237302781999999E+012] \\ [\ 1.044074631599999E+013,\ 1.044074631600001E+013] \\ [-3.700664527560001E+013, -3.700664527559999E+013] \\ [\ 1.009272143879999E+014,\ 1.009272143880001E+014] \\ [-2.133234304110001E+014, -2.133234304109999E+014] \\ [\ 3.500692191359999E+014,\ 3.500692191360001E+014] \\ [-4.443186242880001E+014, -4.443186242879999E+014] \\ [\ 4.316238064511999E+014,\ 4.316238064512001E+014] \\ [-3.147256922040001E+014, -3.147256922039999E+014] \\ [\ 1.666194841079999E+014,\ 1.666194841080001E+014] \\ [-6.044040109800001E+013, -6.044040109799999E+013] \\ [\ 1.343120024399999E+013,\ 1.343120024400001E+013] \\ [-1.378465288200001E+012, -1.378465288199999E+012] \end{pmatrix} \tag{11}$$

As other example, we compute an enclosure for a very large system. We take the symmetric Toeplitz matrix with five bands having the values $1, 2, 4, 2, 1$ and on the right hand side we set all components of b equal to 1. Then the program produces the following output for a system of size $n = 200000$ (only the first ten and last ten solution components are printed):

```
Dimension  n = 200000
Bandwidths l,k : 2 2
A = 1 2 4 2 1
change elements ? (y/n) n
b = =1
change elements ? (y/n) n
```

```
x =
  1: [  1.860146067479180E-001,  1.860146067479181E-001 ]
  2: [  9.037859550210300E-002,  9.037859550210302E-002 ]
  3: [  7.518438200412189E-002,  7.518438200412191E-002 ]
  4: [  1.160876404875081E-001,  1.160876404875082E-001 ]
  5: [  1.003153932563721E-001,  1.003153932563722E-001 ]
  6: [  9.427129202687645E-002,  9.427129202687647E-002 ]
  7: [  1.028361799416204E-001,  1.028361799416205E-001 ]
  8: [  1.005240450090008E-001,  1.005240450090009E-001 ]
  9: [  9.874921290539136E-002,  9.874921290539138E-002 ]
 10: [  1.004617422430963E-001,  1.004617422430964E-001 ]

199990: [  1.001953939326196E-001,  1.001953939326197E-001 ]
199991: [  1.004617422430963E-001,  1.004617422430964E-001 ]
199992: [  9.874921290539136E-002,  9.874921290539138E-002 ]
199993: [  1.005240450090008E-001,  1.005240450090009E-001 ]
199994: [  1.028361799416204E-001,  1.028361799416205E-001 ]
199995: [  9.427129202687645E-002,  9.427129202687647E-002 ]
199996: [  1.003153932563721E-001,  1.003153932563722E-001 ]
199997: [  1.160876404875081E-001,  1.160876404875082E-001 ]
199998: [  7.518438200412189E-002,  7.518438200412191E-002 ]
199999: [  9.037859550210300E-002,  9.037859550210302E-002 ]
200000: [  1.860146067479180E-001,  1.860146067479181E-001 ]

max. rel. error =  1.845833860422451E-016  at i = 3
max. abs. error =  2.775557561562891E-017  at i = 1
min. abs. x[3] = [ 7.518438200412189E-002,  7.518438200412191E-002 ]
max. abs. x[1] = [ 1.860146067479180E-001,  1.860146067479181E-001 ]
```

5. Integration between C-XSC and MPI Libraries

As part of our research, we did the integration between C-XSC and MPI libraries on cluster computers. This step is necessary and essential for the adaptation of our solvers to high performance environments. This integration was developed using, with first tests, algorithms for matrix multiplication in parallel environments of cluster computers. Initially, we did some comparations about the time related to the computational gain using parallelization, the parallel program performance depending on the matrix order, and the parallel program performance using a larger number of nodes. We also studied some other information like the memory requirement in each method to verify the performance relation with the execution time and memory.

We want to join the high accuracy given by C-XSC with the computational gain provided by parallelization. This parallelization was developed with the tasks division among various nodes on the cluster. These nodes execute the same kind of tasks and the communication between the nodes and between the nodes and the server uses message passing protocol.

Measures and tests were made to compare the routines execution time in C language, C using MPI library, C using C-XSC library and C using C-XSC and MPI libraries. The developed tests show that simple and small changes in the traditional algorithms can provide an important gain in the performance [11]. We observed the way that the processor pipeline is used, and we notice that it is decisive for the results. Based in these tests, we could also observe that the use of just 16 nodes is enough for this multiplication.

In the results obtained with these tests, the execution time of the algorithms using C-XSC library are much larger than the execution time of the algorithms that do not use this library. Even in this tests, it is possible to conclude that the use of high accuracy operations make the program slower. It shows that the C-XSC library need to be optimized to have an efficient use on clusters, and make it possible to obtain high accuracy and high performance in this kind of environment.

6. Conclusions

In our work we provide the development of selfverifying solvers for linear systems of equations with sparse matrices and the integration between C-XSC and MPI libraries on cluster computers. Our preliminary tests with matrix multiplication show that the C-XSC library needs to be optimized to be efficient in a High Performance Environment (up to now the main goal of C-XSC was functionality and portability, not speed). Actually we are working in to finish this integration and in the development of parallel software tools for validated numerics in high performance environments using the C++ class library C-XSC in connection with the MPICH library.

Acknowledgement: We would like to thank the students Bernardo Frederes Krämer Alcalde and Paulo Sérgio Morandi Júnior from the Mathematic of Computation and High Performance Computing Group at UFRGS for their help in implementing and testing the C-XSC routines. This work was supported in part by DLR international bureau (BMBF, Germany) and FAPERGS (Brazil).

REFERENCES

[1] Albrecht, R., Alefeld, G., Stetter, H. J. (Eds.): *Validation Numerics – Theory and Applications*. Computing Supplementum **9**, Springer-Verlag (1993).

[2] American National Standards Institute / Institute of Electrical and Electronics Engineers: *A Standard for Binary Floating-Point Arithmetic*. ANSI/IEEE Std. 754-1985, New York, 1985.

[3] Hammer, R., Hocks, M., Kulisch, U., Ratz, D.: *C-XSC Toolbox for Verified Computing I: basic numerical problems*. Springer-Verlag, Berlin/Heidelberg/New York, 1995.

[4] Hofschuster, W., Krämer, W., Wedner, S., Wiethoff, A.: *C-XSC 2.0: A C++ Class Library for Extended Scientific Computing*. Universität Wuppertal, Preprint BUGHW - WRSWT 2001/1 (2001).

[5] Hölbig, C.A., Diverio, T.A., Claudio, D.M., Krämer, W., Bohlender, G.: Automatic Result Verification in the Environment of High Performance Computing In: IMACS/GAMM INTERNATIONAL SYMPOSIUM ON SCIENTIFIC COMPUTING, COMPUTER ARITHMETIC AND VALIDATED NUMERICS, 2002, Paris. Extended abstracts, pg. 54-55 (2002).

[6] Hölbig, C., Krämer, W.: *Selfverifying Solvers for Dense Systems of Linear Equations Realized in C-XSC*. Universität Wuppertal, Preprint BUGHW - WRSWT 2003/1, 2003.

[7] Klatte, R., Kulisch, U., Lawo, C., Rauch, M.,Wiethoff, A.: *C–XSC, A C++ Class Library for Extended Scientific Computing*. Springer-Verlag, Berlin/Heidelberg/New York, 1993.

[8] Klein, W.: *Verified Solution of Linear Systems with Band-Shaped Matrices*.DIAMOND Workpaper, Doc. No. 03/3-3/3, January 1987.

[9] Krämer, W., Kulisch, U., Lohner, R.: *Numerical Toolbox for Verified Computing II - Advanced Numerical Problems*. University of Karlsruhe (1994), see *http://www.uni-karlsruhe.de/~Rudolf.Lohner/papers/tb2.ps.gz*.

[10] Kulisch, U., Miranker, W. L. (Eds.): *A New Approach to Scientific Computation*. Proceedings of Symposium held at IBM Research Center, Yorktown Heights, N.Y., 1982. Academic Press, New York, 1983.

[11] Marquezan, C.C., Contessa, D.F., Alves, R.S., Navaux, P.O.A., Diverio, T.A.: An evolution of simple and efficient optimization techniques for matrix multiplication. In: The 2002 INTERNATIONAL CONFERENCE ON PARALLEL AND DISTRIBUTED PROCESSING TECHNIQUES AND APPLICATIONS, PDPTA 2002. Proceedings ..., Las Vegas, 24-27, June, 2002. Las Vegas: CSREA Press (2002). Vol. IV. p. 2029-2034.

[12] Miranker, W. L., Toupin, R. A.: *Accurate Scientific Computations*.Symposium Bad Neuenahr, Germany, 1985.Lecture Notes in Computer Science, No. **235**,Springer-Verlag, Berlin, 1986.

[13] Neumaier, A.: *The Wrapping Effect, Ellipsoid Arithmetic, Stability and Confidence Regions*. In [1], pp 175–190, 1993.

[14] Rump, S. M.: *Solving Algebraic Problems with High Accuracy*. Habilitationsschrift. In [10], pp 51–120, 1983.

[15] Rump, S. M.: *Validated Solution of Large Linear Systems*. In [1], pp 191–212 (1993).

sobre os métodos foram descritas no capítulo 5.

Na seqüência, foi realizado um conjunto de testes visando validar o ambiente disponibilizado por esta tese. Estes testes visavam avaliar a exatidão e o desempenho dos programas implementados com e sem o uso dos tipos de dados especiais da biblioteca C–XSC. Estes testes foram divididos em quatro categorias:

- Testes dos *solvers* verificados: foram realizados testes referentes a qualidade numérica do resultado obtido com *solvers* verificados para matrizes densas e bandas. Para esses testes foram utilizadas matrizes mal-condicionadas e valores com grandezas muito pequenas (no cálculo da matriz inversa) visando aferir a qualidade numérica dos resultados. Nos resultados obtidos pode-se constatar que os *solvers* conseguiam determinar a existência ou não existência da solução e, caso ela existisse, o resultado correto estava incluído no intervalo resposta final;

- Testes básicos: os testes do cálculo do produto escalar e da multiplicação de matrizes foram de extrema importância para a validação desse trabalho. Com eles pode-se verificar a viabilidade do uso das características da Computação Verificada em ambientes paralelos através do seu envio/recebimento entre os processadores dos *clusters* utilizando para isso as funções da biblioteca de comunicação MPI. Foram realizados cálculos com valores do tipo *double*, *real* e *interval*, o que propiciou a realização da análise do desempenho do C–XSC frente a um programa em C/C++. Obviamente, pelo modo como o C–XSC armazena e opera seus dados especiais (principalmente os de alta exatidão), o desempenho obtido em C–XSC foi inferior ao em C/C++. Entretanto, desta avaliação pode-se concluir que um trabalho de otimização da biblioteca C–XSC poderá minimizar esse problema e que o comportamento (em termos de desempenho) dos programas em C–XSC é semelhante aos programas tradicionais e que a manipulação dos tipos de dados especiais do C–XSC está ocorrendo corretamente nos *clusters*. É importante ressaltar a qualidade numérica dos resultados obtidos com o C–XSC quando os dados de entrada estão sujeitos a instabilidade numérica. Este fato foi observado quando se realizou o cálculo do produto escalar em vetores com valores com grandezas muito diferentes o que, sem o uso da alta exatidão, provocaria erros de cancelamento;

- Testes dos métodos para sistemas lineares: com o objetivo de comparar os programas implementados com e sem a característica da alta exatidão, foram implementados os métodos Gradiente Conjugado, Householder e Givens. Analisando os resultados obtidos pode-se verificar a semelhança do comportamento do desempenho das implementações, conforme fora observado nos testes seqüenciais e paralelos sobre o produto escalar e a multiplicação de matrizes;

- Testes complementares: por fim, foram realizados testes adicionais que abordaram implementações de alguns outros métodos tradicionais para a resolução de sistemas lineares, como Eliminação de Gauss, Gauss-Seidel, Gauss-Jacobi e Decomposição LU. Essas implementações são importantes para trabalhos futuros que poderão ser realizados em relação a implementação verificada paralela de métodos que são utilizados na solução de outras aplicações reais de grande porte como, por exemplo, aplicações relacionadas ao cálculo do fluxo de potência em redes elétricas (BARBOZA; DIMURO; REISER, 2004).

Todos os resultados obtidos foram detalhados no capítulo 6. Com base nessas avaliações pode-se, neste momento, destacar as principais contribuições deste trabalho:

- Disponibilização em um ambiente de alto desempenho (do tipo *cluster* de computadores) da característica da alta exatidão (na verdade com o correto funcionamento da biblioteca C–XSC tem-se nesse ambiente todas as características necessárias para que se desenvolva programas paralelos verificados – com alta exatidão, arredondamentos direcionados e aritmética intervalar);

- Correto funcionamento da biblioteca C–XSC em *clusters* de computadores e sua integração com a biblioteca de troca de mensagens MPI;

- Correta manipulação dos dados especiais do C–XSC (*real, interval, complex, cinterval, rvector, rmatrix, ivector, imatrix, cvector, cmatrix, civector, cimatrix, dotprecision, idotprecision e cidotprecision*) entre os processadores dos *clusters* através do uso de funções da biblioteca MPI;

- Implementação em C–XSC de *solvers* verificados para a resolução de sistemas de equações lineares com matrizes densas e bandas (para diferentes tipos de dados de entrada);

- Implementação de vários métodos para a resolução de sistemas de equações lineares com a característica da alta exatidão;

Todos esses resultados e contribuições vêm sendo avaliados e publicados no meio científico da área desde 2002. Na seção 7.2 é feito um relato das publicações originadas por esta tese e da orientação de trabalhos de alunos de graduação vinculados ao seu tema.

Destes resultados pode-se destacar que a biblioteca C–XSC deve ser utilizada, em um ambiente de alto desempenho, como uma ferramenta de apoio ao cálculo de problemas com alta exatidão e não como a ferramenta computacional principal, ou seja, deve-se resolver um determinado problema utilizando ferramentas computacionais tradicionais e, caso tenha-se problemas com a exatidão do resultado, verifica-se no problema que está sendo trabalhado quais são as áreas ou rotinas mais cruciais a exatidão e, nessas áreas, a biblioteca C–XSC (com suas características) deverá ser utilizada procurando obter um resultado mais exato e confiável. No restanto do problema sugere-se o uso de ferramantas tradicionais com o objetivo de não ser ter uma grade perda de desempenho na solução do problema. Além disso, através desta tese, observa-se viável o uso da biblioteca de alta exatidão C–XSC na resolução de problemas reais de grande porte em ambientes paralelos desde que essa utilização seja realizada em conjunto com um estudo do método computacional que está sendo implementado, verificando áreas mais críticas em relação a exatidão do resultado a ser gerado, e a otimização da biblioteca C–XSC visando a obtenção de um melhor desempenho computacional quando da sua utilização.

7.1 Trabalhos Futuros

Após a apresentação dos principais resultados e das contribuições desta tese é necessário identificar os principais pontos de pesquisa que poderão ser trabalhados com base nos resultados que foram obtidos até o momento. Entre esses possíveis pontos de pesquisa destaca-se:

- Otimização da biblioteca C–XSC através da implementação de novos algoritmos de somatório e/ou de produto escalar, de alterações na forma de implementação das atuais rotinas do C–XSC, da inclusão nas rotinas do C–XSC de rotinas da BLAS, de

instruções em *Assembly* ou das funções SSE (*Streaming SIMD Extensions*) e do uso das diretivas de otimização do compilador gcc (alguns resultados usando o C–XSC com o uso dessas diretivas podem ser obtidos no apêndice A);

- Paralelização dos *solvers* verificados para a resolução de sistemas lineares através de um estudo a ser realizado sobre alternativas de paralelização das diversas rotinas que compõem esses *solvers*;

- Adaptação de alguns métodos paralelos tradicionais para o suporte às técnicas da Computação Verificada através do estudo da viabilidade ou possibilidade de se fazer tal adaptação em busca de um equilíbrio entre o ganho de exatidão e a perda de desempenho;

- Desenvolvimento de novos métodos verificados paralelos para a resolução de problemas numéricos, não exclusivo a sistemas de equações lineares abordando, em um primeiro momento, alguns dos problemas tratados em (HAMMER et al., 1995) e em (KRÄMER; KULISCH; LOHNER, 1996);

- Identificação de novas aplicações reais de grande porte e estudo da viabilidade de sua solução através da Computação Verificada em *clusters* de computadores;

Além disso, é importante ressaltar a importância da continuidade das pesquisas que estão sendo realizadas no momento entre as universidades brasileiras e alemães que visam o uso do C–XSC em ambientes paralelos e a implementação de métodos numéricos verificados nesse tipo de ambiente computacional.

7.2 Publicações Obtidas

Durante o desenvolvimento da pesquisa relacionada a esta tese alguns artigos foram elaborados e publicados em congressos, livros e revistas científicas. Essas publicações relatam, de forma cronológica, toda a evolução do trabalho desenvolvido a partir de 2001, quando do seu início, até o presente momento.

Em 2002 foi publicado um resumo no congresso internacional SCAN´2002 (X IMACS–GAMM *International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics*), realizado em Paris, França (HÖLBIG et al., 2002). Esse resumo apresentou a idéia central do trabalho que seria desenvolvido na tese vinculado aos projetos de cooperação internacional existentes entre os grupos de pesquisa do Brasil e Alemanha.

Em 2003, já baseado nos primeiros resultados da pesquisa referentes a implementação dos *solvers* intervalares, foram publicados resumos nos congressos internacionais ParCo 2003 (*Internacional Conference on Parallel Computing 2003*), realizado em Dresden, Alemanha (HÖLBIG; DIVERIO; KRÄMER, 2003) e PPAM 2003 (V *International Conference on Parallel Processing and Applied Mathematics*), realizado em Czestochowa, Polônia (HÖLBIG et al., 2003). Dessas publicações originaram-se dois artigos completos publicados em 2004, um em uma edição da série *Lecture Notes of Computer Science* (HÖLBIG et al., 2004) e outro como capítulo de livro publicado pela editora Elsevier (HÖLBIG; KRÄMER; DIVERIO, 2004). Além desses artigos, foi publicado um relatório de pesquisa, em conjunto com pesquisadores alemães, onde são detalhados os algoritmos dos *solvers* para a resolução de sistemas lineares densos que foram implementados nesta pesquisa (HÖLBIG; KRÄMER, 2003). Outro relatório sobre o *solver* para matrizes bandas está em fase final de desenvolvimento (HÖLBIG; KRÄMER, 2006).

Em 2004, já com a efetiva integração da biblioteca C–XSC nos *clusters* e com os resultados obtidos com os testes básicos (produto escalar e multiplicação de matrizes), foram publicados resumos e artigos completos em vários congressos internacionais: 75th GAMM (75th *Gesellschaft für Angewandte Mathematik und Mechanik e.V.*), realizado em Dresden, Alemanha (HÖLBIG et al., 2004a), PARA´2004 (VII *Workshop on State-of-the-Art in Scientific Computing*), realizado em Copenhagem, Dinamarca (HÖLBIG et al., 2004), MCM´2004 (*Modern Computational Methods in Applied Mathematics* 2004), realizado em Bedlewo, Polônia (HÖLBIG et al., 2004b) e ICCAM´2004 (XI *International Congress on Computational and Applied Mathematics*), realizado em Leuven, Bélgica (HÖLBIG et al., 2004c).

Em 2005 estão sendo publicados alguns dos resultados finais desta tese. Um resumo foi publicado no congresso internacional ParCo 2005 (*International Conference on Parallel Computing* 2005), realizado em Málaga, Espanha (HÖLBIG et al., 2005) e um artigo completo foi publicado como relatório técnico na Universidade Técnica de Copenhagen na Dinamarca (HÖLBIG et al., 2005). Neste congresso da Espanha foram apresentados os resultados a respeito da implementação dos métodos para resolução de sistemas lineares com alta exatidão em *clusters* de computadores e a disponibilização do ambiente. Um artigo completo sobre esse tema foi submetido e aceito pelos organizadores do *Parallel Computing* para posterior publicação em livro da editora Elsevier.

Além dessas publicações, a tese propiciou a orientação de vários alunos de graduação da UPF e a co-orientação de alunos da UFRGS. Esses alunos desenvolveram ou desenvolvem atividades de pesquisa em assuntos relacionados ao tema da tese e na continuidade do trabalho que foi descrita na seção anterior. Com essas orientações vários artigos foram desenvolvidos e apresentados por estes alunos em diversos eventos científicos como, por exemplo, as ERADs (Escola Regional de Alto Desempenho da SBC Regional Sul), os *Workshops* do GPPD da UFRGS, as Mostras de Iniciação Científica da UPF, os Simpósios de Informática do Planalto Médio, os Salões de Iniciação Científica da UFRGS, o WSCAD (ALCALDE et al., 2003a) e o Congresso Nacional de Matemática Aplicada e Computacional (ALCALDE et al., 2003b; MORANDI JÚNIOR et al., 2003b). Além disso, foram publicados dois artigos na Revista de Iniciação Científica da SBC referentes a trabalhos desenvolvidos por esses alunos (MORANDI JÚNIOR et al., 2004b) e (ALCALDE et al., 2004).

REFERÊNCIAS

ALCALDE, B. F. K.; MORANDI JÚNIOR, P. S.; HÖLBIG, C. A.; DIVERIO, T. A. Resolução de Sistemas Lineares com Alta Exatidão em Ambiente de Alto Desempenho. In: WORKSHOP DE SISTEMAS E COMPUTAÇÃO DE ALTO DESEMPENHO, 2003, São Paulo, Brasil. **Anais...** São Paulo: USP, 2003. p.168–171.

ALCALDE, B. F. K.; MORANDI JÚNIOR, P. S.; HÖLBIG, C. A.; DIVERIO, T. A. Solução de Sistemas Lineares Densos com Computação Verificada. In: CONGRESSO NACIONAL DE MATEMÁTICA APLICADA E COMPUTACIONAL, 26., 2003, São José do Rio Preto, Brasil. **Anais...** São José do Rio Preto: UNESP, 2003.

ALCALDE, B. F. K.; MORANDI JÚNIOR, P. S.; HÖLBIG, C. A.; DIVERIO, T. A. Resolução de Sistemas Lineares com Alta Exatidão. **Revista Eletrônica de Iniciação Científica (online)**, Porto Alegre, v.1, 2004. Disponível em: <http://www.sbc.org.br/reic>. Acesso em: 30 set. 2005.

ALEFELD, G.; HERZBERGER, J. **An Introduction to Interval Computations**. New York: Academic Press, 1983.

BARBOZA, L. V.; DIMURO, G. P.; REISER, R. H. S. Towards Interval Analysis of the Load Uncertainty in Power Electric Systems. In: INTERNATIONAL CONFERENCE ON PROBABILITY METHODS APPLIED TO POWER SYSTEMS, 8., 2004. **Proceedings...** Washington: IEEE, 2004. p.1–6.

BOHLENDER, G.; DAVIDENKOFF, A. Accurate Vector and Matrix Arithmetic for Parallel Computers. In: WORKSHOP ON PARALLEL AND DISTRIBUTED PROCESSING, 3., 1992. **Proceedings...** Sofia: Elsevier Science Publishers, 1992.

BORTOLI, A.; CARDOSO, C.; FACHIN, M.; CUNHA, R. **Introdução ao Cálculo Numérico.** 2. ed. Porto Alegre, Brasil: Universidade Federal do Rio Grande do Sul, 2003.

CLAUDIO, D.; MARINS, J. **Cálculo Numérico Computacional**. São Paulo: Atlas, 1989.

CONTESSA, D. F. **Uso Eficiente da Biblioteca de Rotinas Básicas de Álgebra Linear no Ambiente de Programação Paralela por Troca de Mensagens em Cluster**. 2003. Trabalho de Conclusão (Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

COPETTI, A.; PADOIN, E. L.; KHATCHATOURIAN, O. Transformações de Householder e Givens para a Resolução de Sistemas Lineares em Clusters de PC. In: ENCONTRO

REGIONAL DE MATEMáTICA APLICADA E COMPUTACIONAL, 7., 2002, Porto Alegre, Brasil. **Anais...** Porto Alegre: SBMAC, 2002. p.156–158.

COPETTI, A.; PADOIN, E. L.; KHATCHATOURIAN, O. Paralelização do Método de Householder para a Resolução de Sistemas Lineares. In: SEMANA DE INFORMÁTICA DA REGIÃO CENTRO DO RS, 2002, Santa Maria, Brasil. **Anais...** Santa Maria: UNIFRA, 2002.

COPETTI, A.; PADOIN, E. L.; POSSANI, M.; BINELO, M.; KHATCHATOURIAN, O. Resolução Paralela de Sistemas de Equações Lineares Algébricos. In: ESCOLA REGIONAL DE ALTO DESEMPENHO, 3., 2003, Santa Maria, Brasil. **Anais...** Santa Maria: SBC, 2003. p.209–212.

COSTA, C. M.; STRINGHINI, D.; CAVALHEIRO, G. G. H. Programação Concorrente: Threads, MPI e PVM. In: ESCOLA REGIONAL DE ALTO DESEMPENHO, 2., 2002, São Leopoldo, Brasil. **Anais...** São Leopoldo: SBC, 2002. p.31–65.

CSENDES, T. **Developments in Reliable Computing**. Dordrecht: Kluwer Academic Publishers, 1998.

DIVERIO, T.; FERNANDES, U. **Aplicações de Intervalos**. Porto Alegre, Brasil: PGCC - Universidade Federal do Rio Grande do Sul, 1994. (RP–235).

DONGARRA, J. et al. **Sourcebook of Parallel Computing**. San Francisco: Morgan Kaufmann Publishers, 2003.

DORNELES, R. V. **Particionamento de Domínio e Balanceamento de Carga no modelo HIDRA**. 2003. Tese (Doutorado em Ciência da Computação) — Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

FERNANDES, Ú. A. L. **Núcleo de Alta Exatidão para a Biblioteca libavi.a**. 1997. Dissertação (Mestrado em Ciência da Computação) — Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

GRIMMER, M.; PETRAS, K.; REVOL, N. **Multiple Precision Interval Packages**: Comparing Different Approaches. Wuppertal, Germany: BUGH, Universität Wuppertal, 2003. (Preprint BUGHW-WRSWT 2003/2).

HAMMER, R. et al. **Numerical Toolbox for Verified Computing I**: Basic Numerical Problems. Berlin: Springer–Verlag, 1993.

HAMMER, R. et al. **C–XSC Toolbox for Verified Computing I**: basic numerical problems. New York: Springer-Verlag, 1995.

HOFSCHUSTER, W.; KRÄMER, W. **C–XSC 2.0**: A C++ Class Library for Extended Scientific Computing. Wuppertal, Germany: BUGH, Universität Wuppertal, 2001. (Preprint BUGHW-WRSWT 2001/1).

HÖHER, C. L.; HÖLBIG, C. A.; DIVERIO, T. A. **Programando em Pascal-XSC**. Porto Alegre: Sagra-Luzzato, 1997.

HÖLBIG, C. A. **Métodos Intervalares para a Resolução de Sistemas de Equações Lineares**. 1996. Dissertação (Mestrado em Ciência da Computação) — Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

HÖLBIG, C. A.; CLAUDIO, D. M.; DIVERIO, T. A. Resolução de Sistemas Lineares com Alta Exatidão no Ambiente de Agregados. In: ESCOLA REGIONAL DE ALTO DESEMPENHO, 4., 2004, Pelotas, Brasil. **Anais...** Pelotas: SBC, 2004. p.153–154.

HÖLBIG, C. A.; DIVERIO, T. A.; CLAUDIO, D. M.; KRÄMER, W.; BOHLENDER, G. Automatic Result Verification in the Environment of High Performance Computing. In: IMACS/GAMM INTERNATIONAL SYMPOSIUM ON SCIENTIFIC COMPUTING, COMPUTER ARITHMETIC AND VALIDATED NUMERICS, 10., 2002, Paris, France. **Proceedings...** Paris: University of Paris V, 2002. p.54–55.

HÖLBIG, C. A.; DIVERIO, T. A.; KRÄMER, W. An Accurate and Efficient Selfverifying Solver for Systems with Banded Coefficient Matrix. In: INTERNATIONAL CONFERENCE ON PARALLEL COMPUTING, PARCO, 2003, Dresden, Germany. **Proceedings...** Dresden: University of Dresden, 2003. p.58.

HÖLBIG, C. A.; KOLBERG, M. L.; DIVERIO, T. A.; CLAUDIO, D. M. Solving Linear Systems with High Accuracy on Cluster Computers. **Proceedings In Applied Mathematics And Mechanics (online)**, Berlin, Germany, v.4, 2004. Disponível em: <http://www.wiley-vch.de/publish/en/journals/bySubjectMA00/2130/>. Acesso em: 30 set. 2005.

HÖLBIG, C. A.; KOLBERG, M. L.; MORANDI JÚNIOR, P. S.; ALCALDE, B. F. K.; DIVERIO, T. A.; CLAUDIO, D. M. Solving Linear Systems with High Accuracy on Cluster Computers. In: GESELLSCHAFT FüR ANGEWANDTE MATHEMATIK UND MECHANIK E.V., 75., 2004, Dresden, Germany. **Proceedings...** Dresden: Universität Dresden, 2004. p.259.

HÖLBIG, C. A.; KOLBERG, M. L.; MORANDI JÚNIOR, P. S.; ALCALDE, B. F. K.; DIVERIO, T. A.; CLAUDIO, D. M. Linear Systems: Solvers with High Accuracy on Cluster Computers. In: MODERN COMPUTATIONAL METHODS IN APPLIED MATHEMATICS, 2004, Bedlewo, Poland. **Proceedings...** Bedlewo: University of Bedlewo, 2004. p.20.

HÖLBIG, C. A.; KOLBERG, M. L.; MORANDI JÚNIOR, P. S.; ALCALDE, B. F. K.; DIVERIO, T. A.; CLAUDIO, D. M. Solvers with High Accuracy to Linear Systems on Clusters. In: INTERNATIONAL CONGRESS ON COMPUTATIONAL AND APPLIED MATHEMATICS, 11., 2004, Leuven, Belgium. **Proceedings...** Leuven: Katholieke Universiteit Leuven, 2004. p.70.

HÖLBIG, C.; KRÄMER, W.; DIVERIO, T. An Accurate and Efficient Selfverifying Solver for Systems with Banded Coefficient Matrix. In: JOUBERT, G. R. et al. **Parallel Computing**: Software Technology, Algorithms, Architectures, and Applications. London: Elsevier Science Publishers, 2004. p.283–290. Artigos selecionados do ParCo 2003.

HÖLBIG, C. A.; MORANDI JÚNIOR, P. S.; ALCALDE, B. F. K.; DIVERIO, T. A. Selfverifying Solvers for Linear Systems of Equations in C–XSC. In: INTERNATIONAL CONFERENCE ON PARALLEL PROCESSING AND APPLIED MATHEMATICS, 5., 2003, Czestochowa, Poland. **Proceedings...** Czestochowa: University of Czestochowa, 2003. p.161.

HÖLBIG, C. A.; MORANDI JÚNIOR, P. S.; ALCALDE, B. F. K.; DIVERIO, T. A. Self-verifying Solvers for Linear Systems of Equations in C–XSC. In: WYRZYKKOWSKI, R. et al. **Parallel Processing and Applied Mathematics**. Berlin: Springer-Verlag, 2004. p.292–297. (Lecture Notes in Computer Science, 3019). Artigos revisados do PPAM 2003.

HÖLBIG, C. A.; MORANDI JÚNIOR, P. S.; ALCALDE, B. F. K.; DIVERIO, T. A.; CLAUDIO, D. M. Solving Linear Systems on Cluster Computers with High Accuracy. In: WORKSHOP ON STATE-OF-THE-ART IN SCIENTIFIC COMPUTING, 7., 2004, Lyngby, Denmark. **Para´04: complementary proceedings...** Lyngby, Denmark: Technical University of Denmark, 2004. p.28–29.

HÖLBIG, C. A.; MORANDI JÚNIOR, P. S.; ALCALDE, B. F. K.; DIVERIO, T. A.; CLAUDIO, D. M. **Solving Linear Systems on Cluster Computers with High Accuracy**. Lyngby, Denmark: Department of Informatics and Mathematical Modelling of the Technical University of Denmark, 2005. p.89–96. (IMM-Technical report-2005-09).

HÖLBIG, C. A.; MORANDI JÚNIOR, P. S.; CLAUDIO, D. M.; DIVERIO, T. A. Solving Real Life Applications With High Accuracy. In: INTERNATIONAL CONFERENCE ON PARALLEL COMPUTING, PARCO, 2005, Málaga, Spain. **Proceedings...** Málaga: Universidad de Málaga, 2005. p.98.

HÖLBIG, C.; KRÄMER, W. **Selfverifing Solvers for Dense Systems of Linear Equations Realized in C–XSC**. Wuppertal, Germany: BUGH, Universität Wuppertal, 2003. (Preprint BUGHW-WRSWT 2003/1).

HÖLBIG, C.; KRÄMER, W. **A Selfverifying Solver for Linear Systems with Banded Coefficient Matrix Realized in C–XSC**. Ainda não publicado.

IEEE. **IEEE 754**: Standart for Binary Floating-Point Arithmetic. New York, 1985.

IMACS; GAMM. Resolution on Computer Arithmetic. In: SCAN, 1990. **Proceedings...** [S.l.: s.n.], 1990. p.477–479.

KARNIADAKIS, G.; KIRBY II, R. **Parallel Scientific Computing in C++ and MPI**: A Seamless Approach to Parallel Algorithms and Their Implementation. 2nd ed. Cambridge: Cambridge University Press, 2003.

KHATCHATOURIAN, O.; SAVICKI, L. Optimization of the air distribution in Grain Storehouse with Aeration in No-Uniform Conditions of the Mass of Grains. In: BRAZILIAN CONGRESS OF MECHANICAL ENGINEERING, 16., 2001, Uberlândia, Brasil. **Anais:** engenharia para o novo milênio. Uberlândia: Universidade de Uberlândia, 2001. p.73–82.

KLATTE, R. et al. **PASCAL-XSC - Language Reference with Examples**. New York: Springer-Verlag, 1991.

KLATTE, R. et al. **C–XSC - A C++ Class Library for Extended Scientific Computing**. New York: Springer-Verlag, 1993.

KNÜPPEL, O. **BIAS - Basic Interval Arithmetic Subroutines**. Hamburg, Germany: Technische Informatik III, Technische Universität Hamburg-Harburg, 1993. (Bericht 93.3).

KOESTER, D. P.; RANKA, S.; FOX, G. A parallel Gauss-Seidel algorithm for sparse power system matrices. In: SUPERCOMPUTING, 1994. **Proceedings...** [S.l.: s.n.], 1994. p.184–193.

KRÄMER, W. **Advanced Software Tools for Validated Computing**. Wuppertal, Germany: BUGH, Universität Wuppertal, 2002. (Preprint BUGHW-WRSWT 2002/1).

KRÄMER, W.; KULISCH, U.; LOHNER, R. **Numerical Toolbox for Verified Computing II - Advanced Numerical Problems**. Karlsruhe, Germany: Universität Karlsruhe, 1994. (Draft Version).

KRÄMER, W.; KULISCH, U.; LOHNER, R. **Numerical Toolbox for Verified Computing II**: Advanced Numerical Problems. Berlin: Springer–Verlag, 1996.

KREINOVICH V.; BERNAT, A. Parallel Algorithms for Interval Computations: An Introduction. **Interval Computations**, Moscow, v.3, p.6–62, 1994.

KULISCH, U.; MIRANKER, W. **Computer Arithmetc in Theory and Practice**. New York: Academic Press, 1981.

KULISCH, U.; MIRANKER, W. **A New Approach to Scientific Computation**. New York: Academic Press, 1983.

MAILLARD, N. Algoritmos Matriciais em Processamento de Alto Desempenho. In: ESCOLA REGIONAL DE ALTO DESEMPENHO, 5., 2005, Canoas, Brasil. **Anais...** Canoas: SBC, 2005. p.33–56.

MIRANKER, W.; TOUPIN, R. Accurate Scientific Computations. In: SYMPOSIUM ON ACCURATE SCIENTIFIC COMPUTATIONS, 1985, Bad Neuenahr, RFA. **Proceedings...** Berlin: Springer-Verlag, 1986. (Lecture Notes in Computer Science, 235).

MOORE, R. **Interval Analysis**. Englewood Cliffs: Prentice Hall PTR, 1966.

MOORE, R. **Methods and Applications of Interval Analysis**. Philadelphia: Society for Industrial and Applied Mathematics, 1979.

MORANDI JÚNIOR, P. S.; ALCALDE, B. F. K.; HÖLBIG, C. A.; DIVERIO, T. A. O Uso da Biblioteca de Alta Exatidão C–XSC em Agregados de Computadores. In: SIMPÓSIO DE INFORMÁTICA DO PLANALTO MÉDIO, 4., 2003, Passo Fundo, Brasil. **Anais...** Passo Fundo: Universidade de Passo Fundo, 2003.

MORANDI JÚNIOR, P. S.; ALCALDE, B. F. K.; HÖLBIG, C. A.; DIVERIO, T. A. Sistemas Lineares Esparsos - Solver com Alta Exatidão. In: CONGRESSO NACIONAL DE MATEMÁTICA APLICADA E COMPUTACIONAL, 26., 2003, São José do Rio Preto, Brasil. **Anais...** São José do Rio Preto: UNESP, 2003.

MORANDI JÚNIOR, P. S.; ALCALDE, B. F. K.; HÖLBIG, C. A.; DIVERIO, T. A. Integrando a Biblioteca MPICH e a Biblioteca C–XSC no Cluster labtec. In: WORKSHOP DE PROCESSAMENTO PARALELO E DISTRIBUÍDO, 2., 2004, Porto Alegre, Brasil. **Anais...** Porto Alegre: Universidade Federal do Rio Grande do Sul, 2004. p.87–88.

MORANDI JÚNIOR, P. S.; ALCALDE, B. F. K.; HÖLBIG, C. A.; DIVERIO, T. A. A Integração da Biblioteca de Alta Extatidão C–XSC em Agregados de Computadores. **Revista Eletrônica de Iniciação Científica (online)**, Porto Alegre, v.2, 2004. Disponível em: <http://www.sbc.org.br/reic>. Acesso em: 30 set. 2005.

MORANDI JÚNIOR, P. S.; HÖLBIG, C. A.; DIVERIO, T. A. Integração entre as Bibliotecas C–XSC e MPICH. In: ESCOLA REGIONAL DE ALTO DESEMPENHO, 5., 2005, Canoas, Brasil. **Anais...** Canoas: SBC, 2005. p.165–168.

MPI_FORUM. **The MPI Message Passing Interface Standard**. Knoxville: University of Tennesse, 1994.

NEUMAIER, A. **Interval Methods for Systems of Equations**. Cambridge: Cambridge University Press, 1990.

NEUMAIER, A. The Wrapping Effect, Ellipsoid Arithmetic, Stability and Confidence Regions. **Computing Supplementum**, New York, v.9, p.175–190, 1993.

REBONATTO, M. T. Introdução á Programação Paralela com MPI em Agregados de Computadores (clusters). In: CONGRESSO BRASILEIRO DE COMPUTAÇÃO, 4., 2004, Itajaí, Brasil. **Anais...** Itajaí: Univali, 2004. p.938–955.

RIZZI, R. L. **Modelo Computacional Paralelo para a Hidrodinâmica e para o Transporte de Substâncias Bidimensional e Tridimensional**. 2002. Tese (Doutorado em Ciência da Computação) — Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

RIZZI, R. L.; DORNELES, R. V.; PICCININ JÚNIOR, D.; MARTINOTTO, A. L.; HÖLBIG, C. A.; NAVAUX, P. O. A.; DIVERIO, T. A. Parallelization of Krylov´s Subspace Methods in Multiprocessor PC Cluster. In: JOUBERT, G. R. et al. **Parallel Computing**: Software Technology, Algorithms, Architectures, and Applications. London: Elsevier Science Publishers, 2004. p.543–550. Artigos selecionados do ParCo 2003.

RIZZI, R. L.; DORNELES, R. V.; PICCININ JÚNIOR, D.; MARTINOTTO, A. L.; HÖLBIG, C. A.; NAVAUX, P. O. A.; DIVERIO, T. A. Parallel Computational Model with Local Refinement and Dynamic Load Balancing for 3D Hydrodynamics and Substances Transportation. In: INTERNATIONAL CONGRESS ON COMPUTATIONAL AND APPLIED MATHEMATICS, 11., 2004, Leuven, Belgium. **Proceedings...** Leuven: Katholieke Universiteit Leuven, 2004. p.69.

RUMP, S. M. **INTLAB - INTerval LABoratory**. Dordrecht: Kluwer Academic Publishers, 1998. p.77–104.

RUMP, S. M. Fast and Parallel Interval Arithmetic. **Bit**, New York, v.39, p.534–554, 1999.

SAAD, Y. **Iterative Methods for Sparse Linear Systems**. 2nd ed. Philadelphia: Society for Industrial and Applied Mathematics, 2003.

SCHULTE, M. J.; SWARTZLANDER, E. E. Software and Hardware Techniques for Accurate, Self-Validating Arithmetic. In: APPLICATIONS OF INTERVAL COMPUTATIONS, 1996. **Proceedings...** [S.l.: s.n.], 1996. p.381–404.

STOER, J.; BULIRSCH, R. **Introduction to Numerical Analysis**. New York: Springer-Verlag, 1980.

STRINGHINI, D. **Depuração de Programas Paralelos - Projeto de uma interface intuitiva**. 2002. Tese (Doutorado em Ciência da Computação) — Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

WALTER, W. V. Some Numerical Aspects of ACRITH–XSC and Fortran 90. In: SHARE EUROPE ANNIVERSARY MEETING, 7., 1991. **Proceedings...** [S.l.: s.n.], 1991. v.2, p.651–655.

WANG, X.; ZIAVRAS, S. Parallel Direct Solution of Linear Equations on FPGA-Based Machines. In: WORKSHOP ON PARALLEL AND DISTRIBUTED REAL-TIME SYSTEMS, 2003. **Proceedings...** [S.l.: s.n.], 2003.

ZEMKE, J. **b4m**: a free interval arithmetic toolbox for Matlab based on BIAS. Hamburg, Germany: Technische Informatik III, Technische Universität Hamburg-Harburg, 1998. (Documentation version 1.00).

APÊNDICE A BENCHMARK DA BIBLIOTECA C–XSC

Os testes apresentados a seguir referem-se a medição do desempenho das operações básicas (adição, subtração, multiplicação e divisão) realizadas utilizando o tipo de dado *real* C–XSC em relação ao tipo de dado *double* da linguagem C/C++. Estes testes foram realizados em um processador Pentium III 1GHz com 386 MB de memória RAM utilizando sistema operacional Linux com compiladores g++/gcc versão 3.2.2. Em cada um dos testes foram realizadas $5.000.000$ de repetições para cada operação e os tempos apresentados estão em milisegundos.

A.1 Compilador g++: sem diretriz de otimização

Coluna Tipo de Dado Arredondamento	(1) *double* Ox	(2) *cxsc::real* Ox	(3) *cxsc::real* ∇x	(4) *cxsc::real* $\triangle\ x$	Fator $(2)/(1)$	Fator $(3)/(1)$	Fator $(4)/(1)$
Adição	40	210	3300	3430	5.2	82.5	85.8
Subtração	40	190	3700	4000	4.8	92.5	100.0
Multiplicação	30	210	3410	3590	7.0	113.7	119.7
Divisão	190	310	9100	9330	1.6	47.9	49.1

A.2 Compilador g++: com diretriz de otimização $-O1$

Coluna Tipo de Dado Arredondamento	(1) *double* Ox	(2) *cxsc::real* Ox	(3) *cxsc::real* ∇x	(4) *cxsc::real* $\triangle\ x$	Fator $(2)/(1)$	Fator $(3)/(1)$	Fator $(4)/(1)$
Adição	10	20	2770	2930	2.0	277.0	293.0
Subtração	10	10	3150	3290	1.0	315.0	329.0
Multiplicação	10	10	2830	2950	1.0	283.0	295.0
Divisão	10	10	8630	8760	1.0	863.0	876.0

A.3 Compilador g++: com diretriz de otimização $-O2$

Coluna Tipo de Dado Arredondamento	(1) *double* Ox	(2) *cxsc::real* Ox	(3) *cxsc::real* ∇x	(4) *cxsc::real* $\triangle\ x$	Fator $(2)/(1)$	Fator $(3)/(1)$	Fator $(4)/(1)$
Adição	10	20	2730	2940	2.0	273.0	294.0
Subtração	10	10	3150	3350	1.0	315.0	335.0
Multiplicação	10	10	2820	2960	1.0	282.0	296.0
Divisão	10	10	8590	8820	1.0	859.0	882.0

A.4 Compilador g++: com diretriz de otimização $-O3$

Coluna Tipo de Dado Arredondamento	(1) *double* Ox	(2) *cxsc::real* Ox	(3) *cxsc::real* ∇x	(4) *cxsc::real* $\triangle\ x$	Fator $(2)/(1)$	Fator $(3)/(1)$	Fator $(4)/(1)$
Adição	10	10	2760	2870	1.0	276.0	287.0
Subtração	10	20	3090	3260	2.0	309.0	326.0
Multiplicação	10	10	2900	3000	1.0	290.0	300.0
Divisão	10	10	8550	8750	1.0	855.0	875.0

APÊNDICE B PROGRAMA PARA USO DO *SOLVER* ILSS

```
//------------------------------------------------------------------
// Example: Use of the Interval Linear System Solver - ILSS
// Input: interval vector and matrices || Output: interval vector
//------------------------------------------------------------------
#include <lss_aprx.hpp> // Matrix inversion
#include <ilss.hpp>     // Interval Linear System Solver

using namespace cxsc;
using namespace std;

int main ( )
{
  int  Err, m, n;
  cout << SetPrecision(23,15) << Scientific;    // Output format
  do {
    cout << "Enter␣the␣dimension␣of␣the␣system␣m␣x␣n:␣" << endl;
    cout << "m:␣"; cin  >> m;
    cout << "n:␣"; cin  >> n;
  } while (n <= 0 || m <= 0);

  imatrix A(m,n); ivector b(m), x(n); // Dynamic allocation
  cout << "Enter␣matrix␣A:" << endl;
  cin  >> A;  cout << endl;
  cout << "Enter␣vector␣b:" << endl;
  cin  >> b;  cout << endl;

  LSS(A, b, x, Err); // Function to solve linear system

  if (Err == 0) cout << "Solution␣found␣in:" << endl << x << endl;
   else if (Err == 1)
     cout << "No␣enclosure␣obtained,␣bad␣condition" << endl;
    else if (Err == 2)
     cout << "No␣enclosure␣obtained,␣matrix␣A␣singular" << endl;
  return 0;
}
```

D.3 *Selfverifying Solvers for Linear Systems of Equations in C-XSC*

Artigo publicado em 2004 na série *Lecture Notes in Computer Science* pela editora Springer-Verlag, número 3019, páginas 292 a 297 (HÖLBIG et al., 2004).

APÊNDICE C PROGRAMA PARA USO DO *SOLVER* BAND

```
//------------------------------------------------------------
// Example: Use of the BAND Solver - ILSS
// Input: real vector and matrices || Output: interval vector
//------------------------------------------------------------
#include <band.hpp>

using namespace cxsc;
using namespace std;

int main()
{
  real    nrm, e_rel;
  int     i, i_rel, i_abs, i_min, i_max, total, k, l, n;

  do {
     cout << " Dimension n = "; cin >> n;
     if (n > 0) {
        cout << " Bandwidths l = 2 and k = 0 "
             << endl; l = 2; k = 0;

        rmatrix A(1,n,-l,k);
        ivector b(1,n),x(1,n);
        rvector xr(1,n);

        read_Ab( A, b); // read matrix A and vector b
            lss( A, b, x ); // function to solve
                            // banded linear system

        cout << SaveOpt;    // to save the prevous
                            // configurations of output
        cout << Scientific << SetPrecision(20,15);
        // to set new configurations of output

        cout << "x = " << endl;
        if ( n <= 20 ) writeln_i(x);
        else{
```

```
          writeln_i(x(1,10)); cout << endl;
          writeln_i(x(n-10,n)); cout << endl;
        }

        i_min = 1; i_max = 1; e_rel = 0.0; i_abs = 1; i_rel = 0;
        for(i = 1; i <= n; i++) {
           if ( Sup(abs(x[i])) <  Sup(abs(x[i_min])) )
             i_min = i;
           if ( Inf(abs(x[i])) >  Inf(abs(x[i_max])) )
             i_max = i;
           if ( !(in(0.0,x[i])) ) {
               nrm = diam(x[i]) / Inf(abs(x[i]));
               if (nrm > e_rel) {
                  e_rel = nrm;
                  i_rel = i;
               }
           }
           if ( diam(x[i]) > diam(x[i_abs]) )
                i_abs = i;
        } // for
        cout << endl;
        cout << "max. rel. error = " << e_rel
             << " at i = " << i_rel << endl;
        cout << "max. abs. error = " << diam(x[i_abs])
             << " at i = " << i_abs << endl;
        cout << "min. abs. x[" <<i_min<<"] = "
             << x[i_min] << endl;
        cout << "max. abs. x[" <<i_max<<"] = "
             << x[i_max] << endl;
        cout << endl;
        cout << RestoreOpt; // to restore the output
                            // configuration prevous saved
     }
    }while( (n > 0) );

   return 0;
}
```

APÊNDICE D PUBLICAÇÕES RELACIONADAS A TESE

Nos anexos a seguir são apresentadas três das principais publicações originadas por esta tese.

D.1 *Solving Real Life Applications with High Accucary*

Artigo completo originado do congresso *International Conference on Parallel Computing*, realizado em Málaga (Espanha) em setembro de 2005 (HÖLBIG et al., 2005) e que será publicado em livro da Editora Elsevier em 2006.

D.2 *An Accurate and Efficient Selfverifying Solver for Systems with Banded Coefficient Matrix*

Artigo publicado em 2004 no livro "*Parallel Computing: Software Technology, Algorithms, Architectures, and Applications - Proceedings of the 10th ParCo Conference in Dresden, 2003*", organizado por Joubert, G.R., Nagel, W.E., Peters, F.J. e Walter, W.V., páginas 283 a 290 e publicado pela editora Elsevier B.V. (Londres). Referência em (HÖLBIG; KRÄMER; DIVERIO, 2004).

Solving Real Life Applications With High Accuracy

Carlos Amaral Hölbig[a], Paulo Sérgio Morandi Júnior[b], Dalcidio Moraes Claudio[c], Tiarajú Asmuz Diverio[b]

[a]Universidade de Passo Fundo – ICEG – Passo Fundo – Brazil

[b]Universidade Federal do Rio Grande do Sul – II and PPGC – Porto Alegre – Brazil

[c]Pontifícia Universidade Católica do Rio Grande do Sul – Porto Alegre – Brazil

1. Introduction

In this paper we discuss the implementation of methods to solve linear systems, with high accuracy, used in real life applications. It is important because many different numerical algorithms of these applications contain the solution of linear system (1) as a subproblem. Because of these aspects, this work aims the development of solvers to linear systems (for dense and sparse matrices) with high accuracy on cluster computers using C–XSC library. The methods implemented are used in real life applications like hydrodynamic, agriculture and power electric systems.

$$Ax = b \tag{1}$$

In our research some programs were developed in C–XSC with the high accuracy characteristic, where the results are obtained with a good quality. The C-XSC library is a (free) C++ class library for scientific computing for the development of numerical algorithms delivering highly accurate and automatically verified results by use of the interval arithmetic (see details about this library in [1] and [2]). It provides a large number of predefined numerical data types and operators. These types are implemented as C++ classes. Thus, C-XSC allows high-level programming of numerical applications in C++. Actually, our software run on clusters at UFRGS (Brazil), UPF (Brazil) and Wuppertal (Germany).

During the development of this work was necessary to available a high performance (cluster computers) and high accuracy (by use of C–XSC library) environment. Because of this, our work was divided in 4 (four) steps:

- Integration between the libraries C–XSC and MPI on clusters (section 2);
- Development of solvers for linear systems (section 3);
- Implementation of basic tests in the environment (section 4);
- Solve real life applications with high accuracy (section 5).

Because of these aspects, this work aims the development of solvers with high accuracy for linear systems of equations and the adaptation of the algorithms implemented to cluster computers using C–XSC library. This library is available for download in *www.math.uni-wuppertal.de/wrswt/index_en.html*. Our solvers work with dense and sparse (in special banded matrices) linear equation systems. Nowadays, the solver for dense matrices works with all four basic numerical C–XSC data types: *real, interval, complex*, and *complex interval* and the solver for sparse matrices works with *real* and *interval* data types. All our programs are freeware [3].

2. Integration between C–XSC and MPI Libraries

As part of our research, we done the integration between C–XSC and MPI libraries on cluster computers. This step was necessary and essential for the adaptation of our solvers to high performance environments. This integration was developed using, initially, algorithms for matrix multiplication in parallel environments of cluster computers. We done some comparations about the time related to the computational gain using parallelization, the parallel program performance depending on the matrix order and the parallel program performance using a larger number of nodes. We also studied some other information like the memory requirement in each method to verify the performance relation with the execution time and memory. The initial tests of integration has developed on labtec cluster at II-UFRGS (cluster with 20 Dual Pentium III 1.1 GHz (40 nodes), 1 GB memory RAM, HD SCSI 18 GB and Gigabit Ethernet; cluster server (front-end) with Dual Pentium IV Xeon 1.8 GHz, 1 GB memory RAM, HD SCSI 36 GB and Gigabit Ethernet). We want to join the high accuracy given by C-XSC with the computational gain provided by parallelization [4].

This parallelization was developed with the tasks division among various nodes on cluster. These nodes execute the same kind of tasks and the communication between the nodes and between the nodes and the server uses message passing protocol. About the C–XSC programs executed on cluster, some changes were made in the programs for their correct use in this environment, mainly about how to manipulate *dotprecisions* variables (high accuracy variables of C–XSC) [5].

3. Solvers for Linear Systems

The algorithms implemented in our solvers were described in [6] and can be applied to any system of linear equations which can be stored in the floating point system on the computer. They will, in general, succeed in finding and enclosing a solution or, if they do not succeed, will tell the user so. In the latter case, the user will know that the problem is very ill conditioned or that the matrix A is singular. In the implementation in C–XSC, there is a chance that if the input data contains large numbers or if the inverse of A or the solution itself contain large numbers, an overflow may occur, in which case the algorithms may crash. In practical applications, this has never been observed, however. This could also be avoided by including the floating point exception handling which C–XSC offers for IEEE floating point arithmetic [7].

For this work we implemented interval algorithms for solution of linear systems of equations with dense and sparse matrices [5,8,9]. There are numerous methods and algorithms computing approximations to the solution x in floating-point arithmetic. However, usually it is not clear how good these approximations are, or if there exists a unique solution at all. In general, it is not possible to answer these questions with mathematical rigour if only floating-point approximations are used. These problems become especially difficult if the matrix A is ill conditioned. We present some algorithms which answer the questions about existence and accuracy automatically once their execution is completed successfully. Even very ill conditioned problems can be solved with these algorithms. Most of the algorithms presented here can be found in [10].

3.1. Solver for Dense Linear Systems

The C-XSC programs implemented in our solver for dense linear systems were written for the case of *real* input data (i.e. A is of type *rmatrix* and b is of type *rvector*) and for the case of the data types *interval, complex*, and *complex interval*. The changes made for the use of these other types are mainly changes of the data type of certain variables and functions in the program. Our C–XSC program *verifies the existence* of a solution and *computes an enclosure* for each of the following

types of problems:

(s) compute an enclosure for the solution of system (1) for a *square* $n \times n$ matrix A.

(o) compute an enclosure for the solution of system (1) in the *over-determined* case, i.e. for an $m \times n$ matrix A where $m > n$.

(u) compute an enclosure for the solution of system (1) in the *under-determined* case, i.e. for an $m \times n$ matrix A where $m < n$.

(S) compute an enclosure of the *inverse* A^{-1} of A.

(O) compute an enclosure of the *pseudo inverse* A^{+} of A in the *over-determined* case, i.e. for an $m \times n$ matrix A where $m > n$.

(U) compute an enclosure of the *pseudo inverse* A^{+} of A in the *under-determined* case, i.e. for an $m \times n$ matrix A where $m < n$.

This solver has two modules: the module `lss_aprx` contains the function `MINV` which computes an approximate inverse of the input matrix A of type *rmatrix* using the Gauss-Jordan algorithm (see i.e. [11]), when A is a square matrix. In the over- or under-determined case we use the Moore-Penrose pseudo inverse A^{+} of A (if A has full rank). The second module `lss` contains the functions which solve the dense linear system. This system may be square and non square ($m \times n$). In the over-determined case ($m > n$) a vector $x \in \mathbb{R}^n$ is sought whose residuum $b - Ax$ has minimal Euclidian norm whereas in the under-determined case ($n < m$) a solution $x \in \mathbb{R}^n$ is sought which has minimal norm.

3.2. Solver for Sparse Linear Systems

For the solution of a sparse linear system we present an implementation of an algorithm to compute efficiently componentwise good enclosures. Our implementation works with point as well as *interval* data (data afflicted with tolerances). We assume linear systems whose coefficient matrix has a banded structure. In this case the well known general algorithm (using the Krawczyk operator) to solve systems with dense matrices is not efficient. Since the approximate inverse R of a banded matrix A is in general a full matrix, a lot of additional storage would be required, especially if the bandwidth of A is small compared with its dimension. So a special algorithm is used to reduce the amount of storage and runtime. This method is based on the fact that matrices with banded structure are closely related to difference equations. For the banded system, we apply a LU-decomposition without pivoting (to avoid fill in) to the coefficient matrix A and derive an interval iteration similar to the well known interval iteration used in case of dense matrices. Here, however, we do not use a full approximate inverse R, but rather the interval iteration will be performed by solving two systems with banded triangular matrices L and U. The banded triangular systems are solved with the special method for difference equations described in [6]. In case of point matrices the method is designed to give almost sharp enclosures for all components (large or small in modulus) of the solution vector. A different approach to compute an enclosure for the solution vector of a large linear systems with banded or arbitrary sparse coefficient matrix (which gives enclosures with respect to the infinity norm $||\ \ ||_\infty$ only) is described in [10].

In addition to the implementation of the solution method in C–XSC, the program includes a small demonstration part (a driver) which can be used to solve some simple systems. First the program reads the number of lower and upper bands and then one value for each of the bands, i.e. initially a

Toeplitz matrix is generated. In the next step, however, any number of elements of the matrix can be changed, such that arbitrary banded matrices can be entered. To change the element $a_{i,j}$, only i, j and the new value for this element must be entered. Changing of elements is finished by entering zeros for i and j. Next the right hand side must be entered. There are several choices of predefined solutions, such that the right hand side b will be determined from this given solution. Alternatively b can be set to a constant value in all components or all components can be entered successively. In any case, the values of the components of b may be changed again similarly as for the matrix. When no changes are done anymore, the solution algorithm starts. The banded solver is called and the solution and error statistics are printed. In this way it is quite easy to explore the our C–XSC solver.

4. Basic Tests and Results

Measures and tests were made to compare the routines execution time in C/C++ language, C/C++ using MPI library, C/C++ using C–XSC library and C/C++ using C–XSC and MPI libraries. Our first tests were made to compare the routines execution time, the accuracy of results and to validate the environment developed.

4.1. Scalar product

We done tests comparing the accuracy and execution time between programs in C/C++ language (scalar product) and C/C++ using C–XSC library (optimal scalar product - with high accuracy). In the tables 1, 2 and 3 we showed the results of test with scalar product between two vectors a and b. The values of this vectors are: $a = [10^{50}, 1.25, 10^{50}, 1.1, \ldots, 10^{50}, 1.25, 10^{50}, 1.1]$ and $b = [1, 1, -1, -1, \ldots, 1, 1, -1, -1]$ with size of this vectors (n) equal to $30000, 90000$ and 180000.

Table 1
Parallel Scalar Product $a \times b$ (using 4 nodes of cluster labtec – time in seconds)

Order (n)	Program in C/C++	Program in C–XSC
30000	0.031205	0.041015
90000	0.096727	0.113847
180000	0.188670	0.225083

Table 2
Results of Scalar Product $a \times b$ in C/C++

Order (n)	Results in C/C++	Exact Results
30000	$-3.300000000000000266453525910037569701671 6$	1125
90000	$-3.300000000000000266453525910037569701671 6$	3375
180000	$-3.300000000000000266453525910037569701671 6$	6750

Table 3
Results of Scalar Product $a \times b$ in C–XSC

Order (n)	Results in C–XSC	Exact Results
30000	1124.99999999999931787897367030382156372070 3	1125
90000	3374.99999999999818101059645414352416992187 5	3375
180000	6749.99999999999636202119290828704833984375 0	6750

4.2. Matrix multiplications

We done tests with sequential and parallel programs in C/C++ language, C/C++ using MPI library, C/C++ using C–XSC library and C/C++ using C–XSC and MPI libraries. The results about the performance are showed in the table 4.

Table 4
Parallel Matrix Multiplication (using 8 nodes of cluster labtec – time in seconds)

Program/Order	512×512	1024×1024	2048×2048
C/C++ with MPI	4.603	34.131	265.284
C–XSC with MPI	115.523	916.233	7329.415

4.3. Solvers to linear systems with dense and banded matrices

To validate our solvers on clusters we will describe some tests about the quality of the results of ours solvers. How a first example, a very well known set of ill conditioned test matrices for linear system solvers are the $n \times n$ Hilbert matrices H_n with entries $(H_n)_{i,j} := \frac{1}{i+j-1}$. As a test problem, we report the results of our program for the linear systems $H_n x = e_1$, where e_1 is the first canonical unit vector. Thus the solution x is the first column of the inverse H_n^{-1} of the Hilbert matrix H_n. Since the elements of these matrices are rational numbers which can not be stored exactly in floating point, we do not solve the given problems directly but rather we multiply the system by the least common multiple lcm_n of all denominators in H_n. Then the matrices will have integer entries which makes the problem exactly storable in IEEE floating point arithmetic. For the system $(lcm_{10}H_{10})x = (lcm_{10}e_1)$, the program computes the enclosures (here an obvious short notation for intervals is used) showed in (2), which is an extremely accurate enclosure for the exact solution (the exact solution components are the integers within the computed intervals).

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \end{pmatrix} = \begin{pmatrix} 1.000000000000000E+002, & 1.000000000000000E+002 \\ -\ 4.950000000000000E+003, & -\ 4.950000000000000E+003 \\ 7.920000000000000E+004, & 7.920000000000000E+004 \\ -\ 6.006000000000000E+005, & -\ 6.006000000000000E+005 \\ 2.522520000000000E+006, & 2.522520000000000E+006 \\ -\ 6.306300000000000E+006, & -\ 6.306300000000000E+006 \\ 9.609600000000000E+006, & 9.609600000000000E+006 \\ -\ 8.751600000000000E+006, & -\ 8.751600000000000E+006 \\ 4.375800000000000E+006, & 4.375800000000000E+006 \\ -\ 9.237800000000000E+005, & -\ 9.237800000000000E+005 \end{pmatrix} \tag{2}$$

As an other example, we compute an enclosure for a very large system. We take a symmetric Toeplitz matrix with five bands having the values $1, 2, 4, 2, 1$ and on the right hand side we set all components of b equal to 1. Then the program produces the following output for a system of size $n = 200000$ (only the first ten and last ten solution components are printed):

```
Dimension  n = 200000
Bandwidths l,k : 2 2
A = 1 2 4 2 1
change elements ? (y/n) n
```

```
b = =1
change elements ? (y/n) n

x =
  1: [  1.860146067479180E-001,  1.860146067479181E-001 ]
  2: [  9.037859550210300E-002,  9.037859550210302E-002 ]
  3: [  7.518438200412189E-002,  7.518438200412191E-002 ]
  4: [  1.160876404875081E-001,  1.160876404875082E-001 ]
  5: [  1.003153932563721E-001,  1.003153932563722E-001 ]
  6: [  9.427129202687645E-002,  9.427129202687647E-002 ]
  7: [  1.028361799416204E-001,  1.028361799416205E-001 ]
  8: [  1.005240450090008E-001,  1.005240450090009E-001 ]
  9: [  9.874921290539136E-002,  9.874921290539138E-002 ]
 10: [  1.004617422430963E-001,  1.004617422430964E-001 ]

199990: [  1.001953939326196E-001,  1.001953939326197E-001 ]
199991: [  1.004617422430963E-001,  1.004617422430964E-001 ]
199992: [  9.874921290539136E-002,  9.874921290539138E-002 ]
199993: [  1.005240450090008E-001,  1.005240450090009E-001 ]
199994: [  1.028361799416204E-001,  1.028361799416205E-001 ]
199995: [  9.427129202687645E-002,  9.427129202687647E-002 ]
199996: [  1.003153932563721E-001,  1.003153932563722E-001 ]
199997: [  1.160876404875081E-001,  1.160876404875082E-001 ]
199998: [  7.518438200412189E-002,  7.518438200412191E-002 ]
199999: [  9.037859550210300E-002,  9.037859550210302E-002 ]
200000: [  1.860146067479180E-001,  1.860146067479181E-001 ]

max. rel. error =  1.845833860422451E-016  at i = 3
max. abs. error =  2.775557561562891E-017  at i = 1
min. abs. x[3] = [7.518438200412189E-002, 7.518438200412191E-002]
max. abs. x[1] = [1.860146067479180E-001, 1.860146067479181E-001]
```

Our last example is about the matrix inversion (test about accuracy).

In this example $A = \begin{pmatrix} 1 & 1 & 0 \\ 0 & \epsilon & 0 \\ 0 & 0 & \epsilon^2 \end{pmatrix}$ and $A^{-1} = \begin{pmatrix} 1 & -\frac{1}{\epsilon} & 0 \\ 0 & \frac{1}{\epsilon} & 0 \\ 0 & 0 & \frac{1}{\epsilon^2} \end{pmatrix}$.

With $\epsilon = 1.084202172485504E - 019$, our program in C–XSC obtained:

$$A = \begin{pmatrix} 1.0E+000, & 1.000000000000000E+000, & 0.000000000000000E+000 \\ 0.0E+000, & 1.084202172485504E-019, & 0.000000000000000E+000 \\ 0.0E+000, & 0.000000000000000E+000, & 1.175494350822288E-038 \end{pmatrix}$$

$$A^{-1} = \begin{pmatrix} 1.0E+000, & 1.000000000000000E+000, & 0.000000000000000E+000 \\ 0.0E+000, & 7.136238463529799E+044, & 0.000000000000000E+000 \\ 0.0E+000, & 0.000000000000000E+000, & 8.507059173023462E+037 \end{pmatrix}.$$

5. Solving Real Life Applications With High Accuracy

In the original solvers we included three new methods to solve linear systems: Conjugate Gradient, Householder and Givens methods. All these methods have versions with and without the high accuracy characteristic. We use this methods in real life applications like hydrodynamic (parallel computational model with local refinement and dynamic load balancing for the simulation of substances transportation and hydrodynamic – [12]), agriculture (optimization of the air distribution in grain storehouse with aeration of the mass of grains – [13]) and power electric systems [14]. This methods and applications are research topics of ours research groups in Brazil [9].

In our research we are implementing other methods to solve linear systems. We are implementing, initially, the sequential versions of Gauss-Seidel, Gauss-Jacobi, Gauss Elimination and LU Decomposition methods [15–17]. Nowadays, we are implementing the parallel versions of these methods with and without high accuracy (these programs are been implemented only with the high accuracy characteristic, we are not using interval arithmetic).

6. Conclusions

In our research some programs were developed in C–XSC with the validated numeric paradigm, where the results are obtained with a good quality. The main contributions of our work are:

- integration between the libraries C–XSC and MPI;
- effective use of library C-XSC on cluster computers;
- resolution of linear systems with high accuracy.

In our work we provide the development of selfverifying solvers for linear systems of equations with dense and sparse matrices and the integration between C–XSC and MPI libraries on cluster computers. This integration was not trivial because was necessary to send correctly the special high accuracy variables (*dotprecision*) of C–XSC to the cluster processors using the library MPI without lost of the high accuracy characteristic. Our software run on clusters at UFRGS, UPF and Wuppertal and the integration between C-XSC and MPI was done correctly. Our tests with matrix multiplication, scalar product and methods to solve linear system of equations show that the C-XSC library needs to be optimized to be efficient in a High Performance Environment. This optimization is other research topic of Brazilian groups.

Nowadays we are working in the implementation of parallel versions of methods to solve linear systems (without and with high accuracy). These methods are used in real life applications like hydrodynamic (parallel computational model with local refinement and dynamic load balancing for the simulation of substances transportation and hydrodynamic), agriculture (optimization of the air distribution in grain storehouse with aeration of the mass of grains) and power electric systems.

Acknowledgement

This work is supported in part by CAPES and FAPERGS (Brazil) and DAAD (Germany). It is part of a international cooperation project between Brazilian universities (Universidade Federal do Rio Grande do Sul, Pontifícia Universidade Católica do Rio Grande do Sul and Universidade de Passo Fundo) and German universities (Universität Wuppertal and Universität Karlsruhe).

References

[1] Hammer, R., Hocks, M., Kulisch, U., Ratz, D.: *C-XSC Toolbox for Verified Computing I: basic numerical problems*. Springer-Verlag, Berlin/Heidelberg/New York, 1995.

[2] Hofschuster, W., Krämer, W., Wedner, S., Wiethoff, A.: *C-XSC 2.0: A C++ Class Library for Extended Scientific Computing*. Universität Wuppertal, Preprint BUGHW - WRSWT 2001/1 (2001).

[3] Hölbig, C.A., Krämer, W.: *Selfverifing Solvers for Dense Systems of Linear Equations Realized in C-XSC*. Universität Wuppertal, Preprint BUGHW - WRSWT 2003/1, Wuppertal, 2003.

[4] Hölbig, C.A., Diverio, T.A., Claudio, D.M., Krämer, W., Bohlender, G.: Automatic Result Verification in the Environment of High Performance Computing In: IMACS/GAMM International Symposium on Scientific Computing, Computer Arithmetic and Validated Numerics, 2002, Paris. Extended abstracts, pg. 54-55 (2002).

[5] Hölbig, C.A., Morandi Júnior, P.S., Alcalde, B.F.K., Diverio, T.A., Claudio, D.M.: Solving Linear Systems on Cluster Computers with High Accuracy. In: VII WORKSHOP ON STATE-OF-THE-ART IN SCIENTIFIC COMPUTING, 2004, Copenhagen. Book of Abstracts, pg. 28–29. Copenhagen, 2004.

[6] Krämer, W., Kulisch, U., Lohner, R.: *Numerical Toolbox for Verified Computing II - Advanced Numerical Problems*. University of Karlsruhe (1994), see *http://www.uni-karlsruhe.de/~Rudolf.Lohner/papers/tb2.ps.gz*.

[7] American National Standards Institute / Institute of Electrical and Electronics Engineers: *A Standard for Binary Floating-Point Arithmetic*. ANSI/IEEE Std. 754-1985, New York, 1985.

[8] Hölbig, C.A., Krämer, W., Diverio, T.A.: *An Accurate and Efficient Selfverifying Solver for Systems with Banded Coefficient Matrix*. In [19], pp. 283–290, 2004.

[9] Hölbig, C.A., Kolberg, M.L., Morandi Júnior, P.S., Alcalde, B.F.K., Diverio, T.A., Claudio, D.M.: Solvers with High Accuracy to Linear Systems on Clusters. In: XI INTERNATIONAL CONGRESS ON COMPUTATIONAL AND APPLIED MATHEMATICS, 2004, Leuven. Abstracts of Talks, pg. 70. Leuven, 2004.

[10] Rump, S. M.: *Validated Solution of Large Linear Systems*. In [18], pp 191–212, 1993.

[11] Stoer, J., Bulirsch, R.: *Introduction to Numerical Analysis*. Springer-Verlag, New York, 1980.

[12] Rizzi, R.L., Dorneles, R.V., Piccinin Júnior, D., Martinotto, A.L., Hölbig, C.A., Navaux, P.O.A., Diverio, T.A.: *Parallelization of Krylovs Subspace Methods in Multiprocessor PC Cluster*. In [19], pp. 543–550, 2004.

[13] Khatchatourian, O., Savicki, L.D.: Optimization of the air distribution in Grain Storehouse with Aeration in No-Uniform Conditions of the Mass of Grains. In: 16th Brazilian Congress of Mechanical Engineering, 2001, Uberlândia. Proceedings, pp. 73–82, Uberlândia, 2001.

[14] Barboza, L.V., Dimuro, G.P., Reiser, R.H.S.: Towards Interval Analysis of the Load Uncertainty in Power Electric Systems. In: 8th International Conference on Probability Methods Applied to Power Systems, 2004, Washington. Proceedings, pp 1–6, Washington, 2004.

[15] Dongarra, J., Foster, I., Fox, G., Gropp, W., Kennedy, K., Torczon, L., White, A.: *Sourcebook of Parallel Computing*. Morgan Kaufmann Publishers, San Francisco. 2003.

[16] Karniadakis, G.E., Kirby II, R.M.: *Parallel Scientific Computing in C++ and MPI: A Seamless Approach to Parallel Algorithms and Their Implementation*. Cambridge University Press, Cambridge. 2003.

[17] Yousef Saad: *Iterative Methods for Sparse Linear Systems*. SIAM, Philadelphia. 2003.

[18] Albrecht, R., Alefeld, G., Stetter, H. J. (Eds.): *Validation Numerics – Theory and Applications*. Computing Supplementum **9**, Springer-Verlag (1993).

[19] Joubert, G.R., Nagel, W.E., Peters, F.J., Walter, W.V. (Org.): Parallel Computing: Software Technology, Algorithms, Architectures, and Applications. Elsevier Science Publishers, Londres, 2004.

Selfverifying Solvers for Linear Systems of Equations in C-XSC

Carlos Amaral Hölbig[1], Paulo Sérgio Morandi Júnior[2], Bernardo Frederes Krämer Alcalde[2], and Tiarajú Asmuz Diverio[2]

[1] Universidade de Passo Fundo and PPGC-UFRGS
Campus 1 - BR 285, Passo Fundo (RS), Brazil, CEP 99001-970
holbig@upf.br
[2] Instituto de Informática and PPGC-UFRGS
Campus do Vale, Av. Bento Gonçalves, 9500,
Porto Alegre (RS), Brazil, CEP 91501-970
{sergio, bfkalcalde, diverio}@inf.ufrgs.br

Abstract. In this paper we discuss the implementation of selfverifying solvers for systems of linear equations $Ax = b$ with dense and banded matrices A and the future adaptation these solvers to high performance environments. The algorithms were implemented using C-XSC (a C++ class library for extended scientific computing). We discuss, too, the integration between C-XSC and MPI libraries on cluster computers. The main topics of our research are the development of software tools for Validated Numerics in High Performance Environments using C-XSC and MPI, the optimization of C-XSC and its use on cluster computers and the application these software tools to real life problems [5].

1 Introduction

One of the most frequent tasks in numerical analysis is the solution of linear systems of equations

$$Ax = b \tag{1}$$

with an $m \times n$ matrix A and a right hand side $b \in \mathbb{R}^n$. Many different numerical algorithms contain this task as a subproblem. Because of these aspects, this work aims the development of selfverifying solvers for linear systems of equations and the adaptation of the algorithms implemented to cluster computers using C-XSC library (see details about this library in [3] and [4]). Our solvers work with dense and sparse (in special banded matrices) linear systems of equations. Nowadays, the solver for dense matrices works with all four basic numerical C-XSC data types: *real, interval, complex*, and *complex interval* and the solver for sparse matrices works with *real* and *interval* data types. All our programs are freeware (C++ templates and the C++ exception handling are not used in the actual implementations, these characteristics will be used in future versions of our solvers).

2 The Algorithms

The algorithms implemented in our work were described in [6] and can be applied to any system of linear equations which can be stored in the floating point system on the computer. They will, in general, succeed in finding and enclosing a solution or, if they do not succeed, will tell the user so. In the latter case, the user will know that the problem is very ill conditioned or that the matrix A is singular. In the implementation in C-XSC, there is the chance that if the input data contains large numbers or if the inverse of A or the solution itself contain large numbers, an overflow may occur, in which case the algorithms may crash. In practical applications, this has never been observed, however. This could also be avoided by including the floating point exception handling which C-XSC offers for IEEE floating point arithmetic [2].

For this work we implemented interval algorithms for solution of linear systems of equations with dense and sparse matrices. There are numerous methods and algorithms computing approximations to the solution x in floating-point arithmetic. However, usually it is not clear how good these approximations are, or if there exists a unique solution at all. In general, it is not possible to answer these questions with mathematical rigour if only floating-point approximations are used. These problems become especially difficult if the matrix A is ill conditioned. We present some algorithms which answer the questions about existence and accuracy automatically once their execution is completed successfully. Even very ill conditioned problems can be solved with these algorithms. Most of the algorithms presented here can be found in [7].

3 Solvers for Dense and Sparse Linear Systems

The C-XSC programs implemented in solver for dense linear systems were written for the case of *real* input data (i.e. A is of type *rmatrix* and b is of type *rvector*) and for the case of the data types *interval, complex*, and *complex interval*. The changes made for the use of these other types are mainly changes of the data type of certain variables and functions in the program. This solver has two modules: the module `lss_aprx` contains the function `MINV` which computes an approximate inverse of the input matrix A of type *rmatrix* using the Gauss-Jordan algorithm (see i.e. [8]), when A is a square matrix. In the over- or under-determined case we use the Moore-Penrose pseudo inverse A^+ of A (if A has full rank). The second module `lss` contains the functions which solve the dense linear system. This system may be square and non square ($m \times n$). In the over-determined case ($m > n$) a vector $x \in \mathbb{R}^n$ is sought whose residuum $b - Ax$ has minimal Euclidian norm whereas in the under-determined case ($n < m$) a solution $x \in \mathbb{R}^n$ is sought which has minimal norm. Example solved with this solver is showed in Sect. 4.

For the solution of a sparse linear system we present an implementation of an algorithm to compute efficiently componentwise good enclosures. Our implementation works with point as well as *interval* data (data afflicted with tolerances).

We assume linear systems whose coefficient matrix has a banded structure. In this case the well known general algorithm (using the Krawczyk operator) to solve systems with dense matrices is not efficient. Since the approximate inverse R of a banded matrix A is in general a full matrix a lot of additional storage would be required, especially if the bandwidth of A is small compared with its dimension. So a special algorithm is used to reduce the amount of storage and runtime. This method is based on the fact that matrices with banded structure are closely related to difference equations. For the banded system, we apply a LU-decomposition without pivoting (to avoid fill in) to the coefficient matrix A and derive an interval iteration similar to the well known interval iteration used in case of dense matrices. Here, however, we do not use a full approximate inverse R, but rather the interval iteration will be performed by solving two systems with banded triangular matrices L and U. The banded triangular systems are solved with the special method for difference equations described in [6]. In case of point matrices the method is designed to give almost sharp enclosures for all components (large or small in modulus) of the solution vector. A different approach to compute an enclosure for the solution vector of a large linear systems with banded or arbitrary sparse coefficient matrix (which gives enclosures with respect to the infinity norm $||\ ||_\infty$ only) is described in [7].

In addition to the implementation of the solution method in C-XSC, the program includes a small demonstration part (a driver) which can be used to solve some simple systems. First the program reads the number of lower and upper bands and then one value for each of the bands, i. e. initially a Toeplitz matrix is generated. In the next step, however, any number of elements of the matrix can be changed, such that arbitrary banded matrices can be entered. To change the element $a_{i,j}$, only i, j and the new value for this element must be entered. Changing of elements is finished by entering zeros for i and j. Next the right hand side must be entered. There are several choices of predefined solutions, such that the right hand side b will be determined from this given solution. Alternatively b can be set to a constant value in all components or all components can be entered successively. In any case, the values of the components of b may be changed again similarly as for the matrix. When no changes are done anymore, the solution algorithm starts. The banded solver is called and the solution and error statistics are printed. In this way it is quite easy to explore the our C-XSC solver. Example solved with this solver is showed in next section.

4 Tests and Results

A very well known set of ill conditioned test matrices for linear system solvers are the $n \times n$ Hilbert matrices H_n with entries $(H_n)_{i,j} := \frac{1}{i+j-1}$. As a test problem, we report the results of our program for the linear systems $H_n x = e_1$, where e_1 is the first canonical unit vector. Thus the solution x is the first column of the inverse H_n^{-1} of the Hilbert matrix H_n. Since the elements of these matrices are rational numbers which can not be stored exactly in floating point, we do not solve the given problems directly but rather we multiply the system by the

least common multiple lcm_n of all denominators in H_n. Then the matrices will have integer entries which makes the problem exactly storable in IEEE floating point arithmetic. For $n = 20$, we have $lcm_{20} = 5342931457063200$. For the system $(lcm_{20}H_{20})x = (lcm_{20}e_1)$, the program computes the enclosures (here an obvious short notation for intervals is used) showed in (2), which is an extremely accurate enclosure for the exact solution (the exact solution components are the integers within the computed intervals).

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{15} \\ x_{16} \\ x_{17} \\ x_{18} \\ x_{19} \\ x_{20} \end{pmatrix} = \begin{pmatrix} [\ 3.999999999999999E+002,\ 4.000000000000001E+002] \\ [\text{-}7.980000000000002E+004,\ \text{-}7.979999999999998E+004] \\ [\ 5.266799999999999E+006,\ 5.266800000000001E+006] \\ [\text{-}1.716099000000001E+008,\ \text{-}1.716098999999999E+008] \\ [\ 3.294910079999999E+009,\ 3.294910080000001E+009] \\ [\text{-}4.118637600000001E+010,\ \text{-}4.118637599999999E+010] \\ [\ 3.569485919999999E+011,\ 3.569485920000001E+011] \\ [\text{-}2.237302782000001E+012,\ \text{-}2.237302781999999E+012] \\ [\ 1.044074631599999E+013,\ 1.044074631600001E+013] \\ [\text{-}3.700664527560001E+013,\ \text{-}3.700664527559999E+013] \\ [\ 1.009272143879999E+014,\ 1.009272143880001E+014] \\ [\text{-}2.133234304110001E+014,\ \text{-}2.133234304109999E+014] \\ [\ 3.500692191359999E+014,\ 3.500692191360001E+014] \\ [\text{-}4.443186242880001E+014,\ \text{-}4.443186242879999E+014] \\ [\ 4.316238064511999E+014,\ 4.316238064512001E+014] \\ [\text{-}3.147256922040001E+014,\ \text{-}3.147256922039999E+014] \\ [\ 1.666194841079999E+014,\ 1.666194841080001E+014] \\ [\text{-}6.044040109800001E+013,\ \text{-}6.044040109799999E+013] \\ [\ 1.343120024399999E+013,\ 1.343120024400001E+013] \\ [\text{-}1.378465288200001E+012,\ \text{-}1.378465288199999E+012] \end{pmatrix} \tag{2}$$

As other example, we compute an enclosure for a very large system. We take the symmetric Toeplitz matrix with five bands having the values $1, 2, 4, 2, 1$ and on the right hand side we set all components of b equal to 1. Then the program produces the following output for a system of size $n = 200000$ (only the first ten and last ten solution components are printed):

```
Dimension  n = 200000
Bandwidths l,k : 2 2
A = 1 2 4 2 1
change elements ? (y/n) n
b = =1
change elements ? (y/n) n
x =
  1: [  1.860146067479180E-001,  1.860146067479181E-001 ]
  2: [  9.037859550210300E-002,  9.037859550210302E-002 ]
  3: [  7.518438200412189E-002,  7.518438200412191E-002 ]
```

```
  4: [  1.160876404875081E-001,  1.160876404875082E-001 ]
  5: [  1.003153932563721E-001,  1.003153932563722E-001 ]
  6: [  9.427129202687645E-002,  9.427129202687647E-002 ]
  7: [  1.028361799416204E-001,  1.028361799416205E-001 ]
  8: [  1.005240450090008E-001,  1.005240450090009E-001 ]
  9: [  9.874921290539136E-002,  9.874921290539138E-002 ]
 10: [  1.004617422430963E-001,  1.004617422430964E-001 ]

199990: [  1.001953939326196E-001,  1.001953939326197E-001 ]
199991: [  1.004617422430963E-001,  1.004617422430964E-001 ]
199992: [  9.874921290539136E-002,  9.874921290539138E-002 ]
199993: [  1.005240450090008E-001,  1.005240450090009E-001 ]
199994: [  1.028361799416204E-001,  1.028361799416205E-001 ]
199995: [  9.427129202687645E-002,  9.427129202687647E-002 ]
199996: [  1.003153932563721E-001,  1.003153932563722E-001 ]
199997: [  1.160876404875081E-001,  1.160876404875082E-001 ]
199998: [  7.518438200412189E-002,  7.518438200412191E-002 ]
199999: [  9.037859550210300E-002,  9.037859550210302E-002 ]
200000: [  1.860146067479180E-001,  1.860146067479181E-001 ]

max. rel. error =  1.845833860422451E-016  at i = 3
max. abs. error =  2.775557561562891E-017  at i = 1
min. abs. x[3] = [ 7.518438200412189E-002,  7.518438200412191E-002 ]
max. abs. x[1] = [ 1.860146067479180E-001,  1.860146067479181E-001 ]
```

5 Integration between C-XSC and MPI Libraries

As part of our research, we did the integration between C-XSC and MPI libraries on cluster computers. This step is necessary and essential for the future adaptation of our solvers to high performance environments. This integration was developed using, initially, algorithms for matrix multiplication in parallel environments of cluster computers. We did some comparations about the time related to the computational gain using parallelization, the parallel program performance depending on the matrix order, and the parallel program performance using a larger number of nodes. We also studied some other information like the memory requirement in each method to verify the performance relation with the execution time and memory. This integration has been developed on LabTeC Cluster at II-UFRGS (cluster with 20 Dual Pentium III 1.1 GHz (40 nodes), 1 GB memory RAM, HD SCSI 18 GB and Gigabit Ethernet; cluster server (front-end) with Dual Pentium IV Xeon 1.8 GHz, 1 GB memory RAM, HD SCSI 36 GB and Gigabit Ethernet). We want to join the high accuracy given by C-XSC with the computational gain provided by parallelization. This parallelization was developed with the tasks division among various nodes on the cluster. These nodes execute the same kind of tasks and the communication between the nodes and between the nodes and the server uses message passing

protocol. Measures and tests were made to compare the routines execution time in C language, C using MPI library, C using C-XSC library and C using C-XSC and MPI libraries. In the results obtained until now, the execution time of the algorithms using C-XSC library are much larger than the execution time of the algorithms that do not use this library. Even in this initial tests, it is possible to conclude that the use of high accuracy operations make the program slower. It shows that the C-XSC library need to be optimized to have an efficient use on clusters, and make it possible to obtain high accuracy and high performance in this kind of environment.

6 Conclusions and Future Work

In our work we provide the development of selfverifying solvers for linear systems of equations with dense and sparse matrices and the integration between C-XSC and MPI libraries on cluster computers. Actually, our software run on LabTeC Cluster at UFRGS and the integration between C-XSC and MPI was done correctly. Our tests with matrix multiplication show that the C-XSC library needs to be optimized to be efficient in a High Performance Environment (up to now the main goal of C-XSC was functionality and portability, not speed).

Acknowledgement: This work is supported by FAPERGS and LabTeC/Dell/II-UFRGS Project (Brazil).

References

1. Albrecht, R., Alefeld, G., Stetter, H. J. (Eds.): *Validation Numerics – Theory and Applications*. Computing Supplementum **9**, Springer-Verlag (1993).
2. American National Standards Institute / Institute of Electrical and Electronics Engineers: *A Standard for Binary Floating-Point Arithmetic*. ANSI/IEEE Std. 754-1985, New York, 1985.
3. Hammer, R., Hocks, M., Kulisch, U., Ratz, D.: *C-XSC Toolbox for Verified Computing I: basic numerical problems*. Springer-Verlag, Berlin/Heidelberg/New York, 1995.
4. Hofschuster, W., Krämer, W., Wedner, S., Wiethoff, A.: *C-XSC 2.0: A C++ Class Library for Extended Scientific Computing*. Universität Wuppertal, Preprint BUGHW - WRSWT 2001/1 (2001).
5. Hölbig, C.A., Diverio, T.A., Claudio, D.M., Krämer, W., Bohlender, G.: Automatic Result Verification in the Environment of High Performance Computing In: IMACS/GAMM INTERNATIONAL SYMPOSIUM ON SCIENTIFIC COMPUTING, COMPUTER ARITHMETIC AND VALIDATED NUMERICS, 2002, Paris. Extended abstracts, pg. 54-55 (2002).
6. Krämer, W., Kulisch, U., Lohner, R.: *Numerical Toolbox for Verified Computing II - Advanced Numerical Problems*. University of Karlsruhe (1994), see *http://www.uni-karlsruhe.de/~Rudolf.Lohner/papers/tb2.ps.gz*.
7. Rump, S. M.: *Validated Solution of Large Linear Systems*. In [1], pp 191–212 (1993).
8. Stoer, J., Bulirsch, R.: *Introduction to Numerical Analysis*. Springer-Verlag, New York, 1980.