3

Resolução Numérica de Aplicações com Alta Exatidão em Ambientes de Alto Desempenho

Professores:

Carlos Amaral. Hölbig[1]
(holbig@upf.br)
Marcelo Trindade Rebonatto[2]
(rebonatto@upf.br)
Marcos José Brusso[3]
(brusso@upf.br)

Resumo:

Este curso apresenta uma visão geral sobre a resolução numérica de aplicações com alta exatidão em ambientes de alto desempenho, destacando suas principais características, requisitos e principais áreas de aplicação. São abordados os aspectos matemáticos básicos necessários para a resolução destas aplicações em ambientes computacionais paralelos e apresentadas algumas ferramentas e técnicas computacionais voltadas à melhora do desempenho computacional dos programas seqüenciais e paralelos com alta exatidão. Ao final do texto são descritas algumas aplicações reais com alta exatidão que estão sendo trabalhadas em ambientes de alto desempenho.

1 Graduado pela Universidade Católica de Pelotas. Cursou mestrado e doutorado em Ciência da Computação pela UFRGS (2005) com doutorado sandwich na Universidade de Wuppertal (Alemanha) em 2002/2003. É professor titular da Universidade de Passo Fundo (UPF) ministrando as disciplinas de Aritmética Computacional e Computação Científica no curso de Bachalerado em Ciência da Computação. Atualmente é coordenador da Divisão de Pesquisa da VRPPG-UPF. Atua principalmente nas áreas de computação verificada, ambientes de alto desempenho e computação científica. É líder do grupo de pesquisa Computação Paralela e Sistemas Distribuídos (ComPaDi) da UPF

2 Bacharel em Ciência da Computação pela Universidade de Passo Fundo (1993) e Mestre em Ciência da Computação pela Universidade Federal do Rio Grande do Sul (2000) na linha de pesquisa em Processamento Paralelo e Distribuído. Professor universitário desde 1996, atualmente colabora em dois projetos de pesquisa e coordena um, tendo já atuado em nove projetos. Ministra disciplinas de sistemas distribuídos, processamento paralelo e programação.

3 Graduado em Ciência da Computação pela Universidade de Passo Fundo (1994) e mestrado em Ciência da Computação pela Universidade Federal do Rio Grande do Sul (2000). Atualmente é professor Adjunto III da Universidade de Passo Fundo. Tem experiência na área de Ciência da Computação, com ênfase em Arquitetura de Sistemas de Computação e em Programação de Computadores, atuando principalmente nos seguintes temas: sistemas distribuídos, processamento paralelo, arquitetura de computadores e mineração de dados na web

3.1 Introdução

No decorrer dos tempos vários métodos matemáticos foram 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 foram 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 em formatos binários, 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 dos problemas. Neste sentido, muitos esforços de pesquisa têm sido dirigidos na elaboração de uma aritmética que supere as limitações impostas pelas características inerentes à 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, através de métodos de inclusão monotônica, com o hardware através da 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].

Além do aspecto da alta exatidão não se deve esquecer que vários requisitos devem ser abordados para que se tenha um programa de alto desempenho. Segundo Maillard estes requisitos devem abordar os seguintes aspectos (2005):

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

Este curso relata a utilização em um ambiente de alto desempenho (do tipo clusters de computadores) de uma biblioteca que disponibiliza a característica da alta exatidão (biblioteca C-XSC). Através desta utilização e da conseqüente implementação de métodos numéricos computacionais com alta exatidão em ambientes paralelos, 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.

O restante deste texto esta organizado como segue. A Seção 3.2 apresenta os conceitos básicos necessários para se ter a Validação Numérica no computador. A Seção 3.3 relata algumas ferramentas computacionais relacionadas ao tema deste curso e, em especial, descreve as principais características da biblioteca de alta exatidão C-XSC. A Seção 3.4 descreve como a biblioteca C-XSC pode ser utilizada em *clusters* de

computadores. A Seção 3.5 apresenta algumas alternativas de otimização para o C-XSC visando a melhora de seu desempenho computacional em ambientes paralelos. Por fim, a Seção 3.6 mostra exemplos de algumas aplicações que estão utilizando a Validação Numérica na resolução de seus problemas e um breve histórico do uso de ferramentas de alta exatidão em ambientes computacionais paralelos.

3.2. Conceitos Básicos de Validação Numérica

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 Validação Numérica 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 Validação Numérica 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 Validação Numérica 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 3.1 apresenta os requisitos necessários para se obter a Validação Numérica. Com a Validação Numérica 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 (1966, 1979), Alefeld (1983), Neumaier (1990), Claudio (1989), Kulisch (1981, 1983) e Miranker (1986). Nas próximas subseções serão abordados os conceitos básicos para se ter a Validação Numérica no computador.

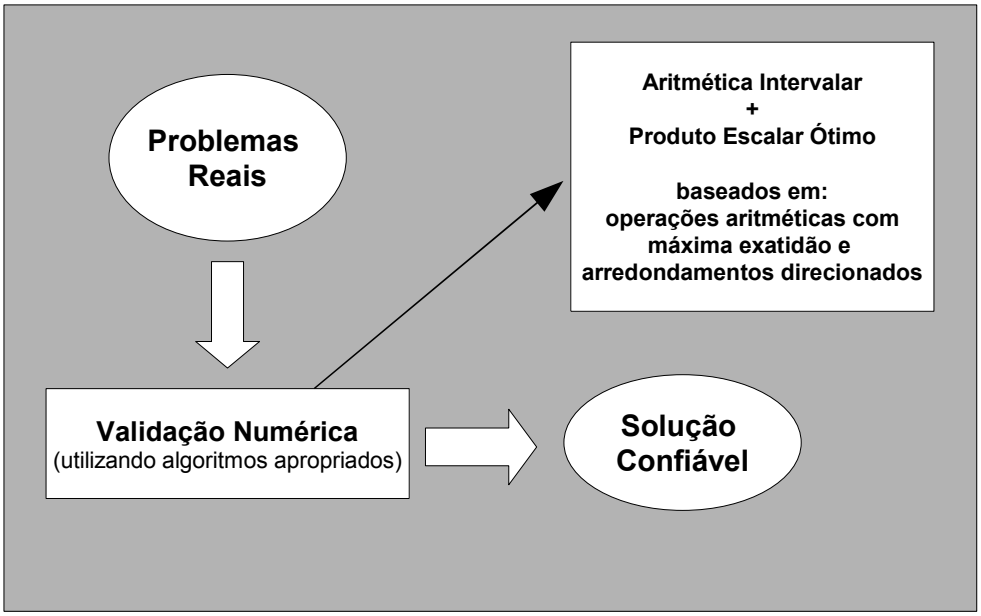

Figura 3.1. Requisitos para a Computação Verificada

3.2.1. Matemática Intervalar

A matemática intervalar tem sido utilizada em diversas áreas como 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 et al. 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 garantida. 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 diversos problemas numéricos. Várias das características do padrão IEEE-754 [IEEE 1985] 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 *under*flow e *over*flow 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 a consistência lógica de programas e o 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. 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 (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 definidos para argumentos intervalares.

$$X = [x_1, x_2] = \{x \mid x_1 \leq x \leq x_2\} \tag{1}$$

A partir da aritmética intervalar são desenvolvidos os conceitos que compõem a matemática intervalar e os métodos intervalares para resolução de problemas numéricos. Uma descrição completa das operações existentes sobre o conjunto de intervalos poderá ser encontrada em [Hölbig 2005].

3.2.2. Arredondamentos Direcionados

Os arredondamentos direcionados são funções de mapeamento utilizadas para representar os números reais em números de máquina [Kulish e 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 baixo (∇x) é a função que aproxima o número real x para o menor número de máquina que o contém, conforme (2);
- Arredondamento para cima (Δx) é a função que aproxima o número real x para o maior número de máquina que o contém, conforme (3).

$$\nabla x = min\{y \in F / y \geqslant x\} \forall x \in \Re \tag{2}$$

$$\triangle x = max\{y \in F / y \leqslant x\} \forall x \in \Re \tag{3}$$

$$Ox = \begin{cases} \nabla x & \text{se } |x - \nabla x| < |x - \triangle x| \\ \nabla x & \text{se } |x - \nabla x| = |x - \triangle x| \text{ e } \nabla x \text{ é par} \\ \triangle x & \text{em caso contrário} \end{cases} \tag{4}$$

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 o qual arredonda o número real *x* para o número de máquina mais próximo e é definido em função dos arredondamentos para cima e para baixo e produz um menor erro de aproximação, conforme (4).

3.2.3. 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 binária de ponto-flutuante ordinária, definida pelo padrão IEEE-754, a maioria dos erros ocorre em acumulações, isto é, pela execução de uma seqüência de operações de adição e de subtração. Na aritmética de ponto-fixo, contudo, a operação de acumulação é realizada sem erros. Assim, grande parte dos erros encontrados em ponto-flutuante poderia ser evitada se a acumulação fosse realizada em ponto-fixo, em um acumulador especial [Miranker e Toupin 1986].

Com a atual tecnologia pode-se realizar facilmente essa acumulação em 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 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 poderão 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 e com o arredondamento para o número mais próximo de máquina (definidos na Seção 3.2.2).

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, 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 que aborda esses tópicos [IMACS e GAMM 1990].

Os requisitos da aritmética de alta exatidão são: arredondamentos direcionados e operações aritméticas com máxima que 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: $x \boxdot y := \in(x \circ y)$ para todos $x, y \in F$ (onde $\in$ 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: $+, -, *, /, \sqrt{}$, 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 3.3.5 é apresentado como a aritmética de alta exatidão é implementada e utilizada na biblioteca C–XSC.

3.2.4. Produto Escalar Ótimo

O produto escalar entre dois vetores de dimensão n, definido da maneira tradicional, envolve $2n-1$ arredondamentos, definido em (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.

$$v \times w = \square \sum_{i=1}^{n} \square v_i \times w_i \tag{5}$$

$$v \times w = \square \sum_{i=1}^{n} v_i \times w_i \tag{6}$$

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 (6).

3.3. Ferramentas Computacionais

A resolução de sistemas de equações lineares é um dos problemas numéricos mais comuns em aplicações científicas. Tais sistemas surgem em conexão com a solução de equações diferenciais parciais, determinação de caminhos ótimos em redes (grafos) e interpolação de pontos, dentre outras aplicações [Bortoli et al. 2003]. Os problemas que podem ser resolvidos através destes 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, podem-se citar aplicações nas áreas de hidrodinâmica ([Rizzi 2002] e [Dorneles 2003]), transporte de massa (mecânica de fluídos), biomecânica, sistemas elétricos, fluxo de potências [Barboza, Dimuro e Reiser 2004] e a simulação de processos de aeração de grãos em silos de armazenagem [Borges e Padoin 2006].

Devido a esses fatores é que foram selecionados algumas ferramentas computacionais (tradicionais e com alta exatidão) que podem auxiliar na solução de sistemas lineares que são utilizados na solução de aplicações reais de grande porte.

Para efeitos didáticos procurou-se dividir estas ferramentas em de alta exatidão, ou seja, as que possuam pelo menos a característica da alta exatidão e, em alguns casos, todas as características necessárias para se ter a Validação Numérica, e em tradicionais que são as ferramentas seqüenciais e paralelas utilizadas mais tradicionalmente na área de Computação Científica.

3.3.1. Ferramentas Seqüenciais e Paralelas Tradicionais

Sobre as ferramentas seqüenciais e paralelas pode-se destacar:

- BLAS (*Basic Linear Algebra Subroutines*): é uma biblioteca matemática de sub-rotinas 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);
- LAPACK (*Linear Algebra PACKage*): É uma poderosa ferramenta para computação numérica de alto desempenho, utilizada para resolver problemas de álgebra linear como sistemas de equações lineares, problemas envolvendo mínimos quadrados, fatoração de matrizes e autovalores e autovetores. Suas rotinas foram construídas utilizando chamadas para as funções da BLAS (http://www.netlib.org/lapack);
- PETSc (*Portable, Extensible Toolkit for Scientific Computation*): 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++ (*Iterative Methods Library*): é uma biblioteca para 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 (*Parallel SPArse Symmetric dirEct Solver*): 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/).

A respeito de outras bibliotecas/solvers tradicionais, informações adicionais poderão ser encontradas em [Dongarra et al. 2003] e [Karniadakis e Kirby II 2003].

3.3.2. Ferramentas Computacionais de Alta Exatidão

As ferramentas de alta exatidão podem ser bibliotecas desenvolvidas sob algum tipo de software matemático (MatLab, Maple, ...) ou linguagem de programação (C, Pascal ou, 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. 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] (http://www.xsc.de);
- IntLab (*Interval Laboratory*): 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 - 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 1999] (http://www.ti3.tu-harburg.de/rump/intlab/);
- PROFIL/BIAS (*Programmer's Runtime Optimized Fast Interval Library*): é uma biblioteca de classes em C++ que suporta operações com números reais e intervalares (http://www.ti3.tu-harburg.de/keil/profil/index_e.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 [Zemke 1998] (http://www.ti3.tu-harburg.de/zemke/b4m/);

- IntPakX: o pacote IntPakX disponibiliza o desenvolvimento de algoritmos com verificação numérica no Maple ([Grimmer, Petras e Revol 2003] e [Krämer 2002]) (http://www.math.uni-wuppertal.de/~xsc/software/intpakX/);
- Sun *Studio* C/C++/Fortran *Compilers*: compiladores desenvolvidos pela Sun Microsystems que suporta o uso da aritmética intervalar (http://developers.sun.com/sunstudio/overview/topics/compilers_index.html)
- MKL (*Math Kernel Library*): é uma distribuição da BLAS desenvolvida pela INTEL e suas rotinas foram otimizadas para utilização em seus microprocessadores. Ela provê uma implementação da BLAS e do LAPACK, com todas suas rotinas para a resolução de problemas da álgebra linear e ainda algumas rotinas matemáticas novas, para a engenharia e aplicações financeiras. Sua versão em Fortran disponibiliza a aritmética intervalar (http://www.intel.com/support/performancetools/libraries/mkl/index.htm).

Informações adicionais sobre outras bibliotecas/ferramentas de alta exatidão poderão ser obtidas no endereço http://www.cs.utep.edu/interval-comp/intsoft.html, portal sobre a Validação Numérica.

3.3.3. Biblioteca C-XSC

Com base nas informações já apresentadas neste texto uma pergunta deve ser respondida: Qual o motivo da escolha da biblioteca C–XSC como a ferramenta que disponibilizará a alta exatidão visando a resolução de problemas numéricos em ambientes de alto desempenho? Essa escolha se deve, principalmente, a dois fatores: disponibilidade e usabilidade. Disponibilidade por ser uma biblioteca *freeware*; e usabilidade por o C–XSC ser uma biblioteca da linguagem C/C++, linguagem 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.

A seguir, resumidamente, serão apresentadas algumas características da biblioteca C-XSC destacando, principalmente, as relacionadas com a disponibilização das técnicas da Validação Numérica detalhadas na Seção 3.2.

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 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 2.2.2. 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.

Nas próximas subseções serão apresentados e exemplificados alguns dos mais importantes conceitos disponibilizados pela biblioteca C–XSC, em especial os referentes ao paradigma da Validação Numérica. Maiores detalhes sobre a biblioteca C–XSC são descritos em [Hammer et al. 1995], [Klatte et al. 1993], [Krämer 2002] e [Hofschuster e Krämer 2001].

3.3.3.1 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.

3.3.3.2 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×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 3.2 onde é apresentado um trecho de código em C–XSC que realiza a fatoração 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 3.2. Fatoração LU de uma matriz A(n×n) em C–XSC

Este exemplo da fatoração LU demonstra duas características importantes do C–XSC. Primeiro, se salva 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,n): ";
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 3.3. Alocação e redefinição dinâmica matrizes em C–XSC

No exemplo apresentado na Figura 3.3 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.

3.3.3.3 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 ($*$, $+$, $-$, $/$, $\square$) estão disponíveis para os tipos intervalares. Na Figura 3.4 é apresentado um trecho de programa que apresenta algumas operações aritméticas sobre o tipo de dado intervalar.

3.3.3.4 Arredondamentos Direcionados

Usando o conceito corrente e operadores sobrecarregados << e >> 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 manipular 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.

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

Figura 3.4. Uso de intervalos no C–XSC

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()`).

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

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

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

Figura 3.5. Arredondamentos Direcionados no C–XSC

Além dessas formas e baseado nestes 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 e que os resultados são armazenados internamente no computador já arredondados, conforme descrito na Figura 3.5.

3.3.3.5 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 é chamado 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 3.6 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 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.

```
...
real result;
dotpreciosion 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 << "Produto escalar otimo = " << result << end;
...
```

Figura 3.6. Produto Escalar Ótimo no C–XSC

3.3.6 A Estrutura de um Acumulador dotprecision real

Um acumulador `dotprecision` real ocupa um espaço de armazenamento (conforme Figura 3.7) com um tamanho total de *bits* de acordo com (7), 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].

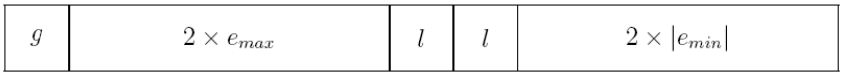

Figura 3.7. Estrutura de um acumulador dotprecision real

$$L = g + 2e_{max} + 2|e_{min}| + 2l \text{ dígitos da base } b \tag{7}$$

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 (8).

$$\begin{aligned} L &= 31 + 2 \times 1023 + 2 \times |-1022| + 2 \times 53 \\ &= 4227 \; bits \\ &= 529 \; bytes \\ &= 133 \text{ palavras de memória de } 32 \; bits \text{ cada} \end{aligned} \tag{8}$$

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.

3.4. Alta Exatidão em Cluster de Computadores

Essa seção trata do uso de agregados de computadores (*clusters* de computadores) em conjunto com aplicações que buscam alta exatidão. O uso dos clusters é empregado com o objetivo de aumentar a capacidade computacional disponível para as aplicações. Essa maior capacidade computacional busca amenizar o aumento do tempo de execução das aplicações, decorrente dos requisitos para obtenção alta exatidão como, por exemplo. a aritmética intervalar [Grimmer 2005, p. 3].

3.4.1. Biblioteca MPI

Conforme Cavalheiro (2006, p. 40) o *Messaging Passing Interface* (MPI) é uma ferramenta de programação que permite a programação concorrente de aplicações. Em especial a biblioteca MPI é um padrão de fato e de direito na implementação de aplicações computacionais para serem executadas em ambientes computacionais paralelos que não possuem memória compartilhada. Os agregados de computadores são máquinas paralelas que não dispõem de memória compartilhada[1].

[1] Uma exceção são os componentes dos agregados que porventura possam possuir memória compartilhada. Nesse caso, a memória seria compartilhada apenas entre as unidades processadoras de um mesmo nodo. Clusters que possuem nodos com memória compartilhada são os que são formados por máquinas de arquitetura SMP ou multi-core, por exemplo. Nesses casos, há memória compartilhada em

O MPI foi criado para resolver o problema de interoperabilidade entre as diferentes bibliotecas para exploração do paralelismo por meio de trocas de mensagens. Ele define um conjunto padrão de rotinas para trocas de mensagens que pode ser utilizado para escrever um programa paralelo portável utilizando C, C++ ou Fortran [Grama et. al. 2003].

O padrão MPI foi concluído no final do primeiro semestre de 1994 (versão 1.0) pelo MPI Forum e atualizado na metade do ano seguinte (versão 1.1) ([Cáceres e Song 2004] e [Message Passing Interface Forum 2004]). A versão 2 teve sua preliminar apresentada no *SuperComputing´96*, e em abril de 1997 o documento MPI-2 foi unanimemente votado e aceito [Ignácio e Ferreira Filho 2002]. Ele é o resultado de um esforço da comunidade para definir e padronizar a sintaxe e semântica de uma biblioteca de rotinas para trocas de mensagens que pudesse ser implementada numa ampla variedade de máquinas paralelas [Cáceres e Song 2004]. Alguns autores apontam sua utilização e aceitação atual é devida a colaboração dos membros que constituíram o MPI Forum. O MPI Forum foi aberto, constituído por pesquisadores, acadêmicos, programadores, usuários e fabricantes, representando aproximadamente 40 organizações.

O padrão MPI não define uma implementação, mas sim a lógica das operações para comunicação entre processos. Dessa forma, as implementações desse padrão podem ser realizadas por programadores especialistas em cada hardware a ser portado. A disponibilização de diversas implementações que funcionam em arquiteturas distintas, conduz o MPI a um beneficio adicional: a transparência. Uma aplicação desenvolvida num determinado sistema computacional homogêneo pode ser facilmente executada em outro sistema, com diferente arquitetura e até mesmo heterogêneo, contendo máquinas com diferentes arquiteturas [Cáceres e Song 2004].

Ainda que possua várias implementações, elas conseguem apresentar um bom desempenho uma vez que somente a lógica das operações é especificada. A implementação fica a cargo de desenvolvedores que otimizam o código de acordo com o hardware a ser portado.

Existem implementações especialmente desenvolvidas para famílias de computadores, computadores específicos e versões disponibilizadas por universidades e centros de pesquisa que funcionam em diversos tipos de máquinas. As principais implementações que funcionam em várias plataformas de hardware são:

- MPICH: Argonne National Laboratory/Mississipi State University;
- LAM: Ohio Supercomputer Center/Indiana University.

O *Local Area Multicomputer* (LAM) é uma versão que foi desenvolvida para funcionar tanto em clusters como em estações de trabalho ligadas através de rede (máquinas NOW). Dessa forma, disseminou-se rapidamente mesmo quando o número de clusters dedicados ao processamento de alto desempenho era reduzido. Essa será a versão que será utilizada nas demais referências e exemplos no decorrer do texto, sendo executada no SO Linux.

3.4.2. Integrando as Bibliotecas C-XSC e MPI

A escrita de aplicações com MPI segue o modelo de trocas de mensagens, onde processos são executados em espaços de endereçamento de memória distintos (multicomputadores). A comunicação ocorre quando uma região de memória de um

um componente do cluster (intra-nó) e o restante do agregado não possui memória compartilhada.

processo é copiada para outro. Esta operação é realizada quando um processo executa uma operação de envio (*send*) e o outro executa uma de recebe (*receive*) o conteúdo de memória a ser comunicado [Gropp, Lusk e Skjellum 1999, p. 13].

Nesse contexto, uma aplicação paralela escrita com MPI é constituída processos concorrentes que se comunicam por meio de rotinas do tipo envia/recebe para o compartilhamento dos dados. Cada um destes processos pode executar um fluxo de execução diferente do outro, caracterizando um modelo de execução com *Multiple Program Multiple Data (*MPMD) [Ignácio e Ferreira Filho 2002, p. 107]. Na prática, utiliza-se disparar um único programa gerando múltiplos processos onde cada processo irá executar uma porção delimitada do código. Esse modelo é conhecido como *Simple Program Multiple Data* (SPMD) e segundo Costa, Stringhini e Cavalheiro é "uma forma natural de programar" aplicações MPI (2002, p. 56).

As aplicações que utilizam a biblioteca C-XSC buscam soluções computacionais que propiciem resultados com alta exatidão e verificados automaticamente. Essas características são fornecidas por meio de diversos tipos de dados específicos e operadores pré-definidos, implementados em C++ [Hölbig 2005].

3.4.2.1 Programação com MPI e C-XSC

O MPI possui aproximadamente 120 funções (pode variar de acordo com a implementação), porém o número de seus conceitos chaves é menor. Para compreender o funcionamento da escrita de aplicações em MPI deve-se compreender:

- Processos MPI;
- Comunicadores;
- Mensagens.

A execução das aplicações escritas com MPI ocorre com o disparo em paralelo de seus processos, geralmente com o auxílio do script *mpirun*, fornecido por diversas implementações do MPI. Num cluster onde existem várias instâncias de S.O.s, uma em cada nodo, a identificação dos processos MPI é realizada por meio de seu *rank.*

Os processos MPI organizam-se em grupos ordenados para realizar uma comunicação eficiente. Para possibilitar a comunicação entre os processos do grupo, o MPI introduz o conceito de comunicadores, que nada mais são do que um conjunto de processos que podem ser conectados. Por padrão, todos os processos fazem parte de um grupo único associado ao comunicador pré-definido `MPI_COMM_WORLD`.

Para a comunicação dos processos através dos comunicadores o MPI definiu um formato de mensagens. Elas podem ser enviadas ou recebidas e são compostas de duas partes: dados (conteúdo comunicado) e envelope (informações de controle da comunicação). Os dados a serem comunicados compreendem a referência na memória onde eles devem ser copiados/acessados bem como o tipo de dado sendo comunicado. O envelope contém informações sobre a origem/destino das mensagens, representadas geralmente pelo *rank* de cada processo.

3.4.2.2 Tipos de dados suportados pelo MPI x tipos de dados do C-XSC

Os tipos de dados utilizados nas mensagens MPI não são tipos padrão da linguagem de programação utilizada. Ao invés, o MPI proporciona tipos próprios de dados para permitir a utilização de máquinas com arquiteturas diferentes na composição de sua máquina virtual paralela. Esse fato é devido a uma implementação não homogênea dos

tipos de dados na grande variedade de máquinas as quais o MPI é portado. A Tabela 3.2 ilustra os tipos de dados suportados no MPI e sua relação com os disponíveis na linguagem C/C++ e relaciona os tipos MPI associandos com os tipos padrão da linguagem C/C++. Os dois últimos, `MPI_BYTE` e `MPI_PACKED` não possuem correspondente direto na linguagem C/C++.

A biblioteca C-XSC dispõem de tipos de dados específicos para a computação verificada e de alta exatidão, diferentes dos tipos fornecidos em C/C++. São tipos de dados para suportar intervalos (`interval` e `cinterval`), números complexos (`complex`) e também para a aritmética de alta exatidão (`dotprecision`). Além desses tipos escalares, arranjos unidirecionais (vetores) e bidimensionais (matrizes) dos tipos diferenciados de dados são fornecidos.

Tabela 3.2. Tipos de dados do MPI relacionados aos fornecidos pela linguagem C/C++

Tipos de dados do MPI	Tipos de dados em C
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unisigned short
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	
MPI_PACKED	

Fonte: PACHECO, 1997, p. 48

A programação de aplicações computacionais envolvendo MPI e C-XSC deve permitir a comunicação dos dados manipulados em C-XSC através das estruturas de comunicação disponibilizadas pelo MPI. Para essas tarefas, os tipos não relacionados com nenhum tipo específico em C/C++ podem ser utilizados, além da criação de novos tipos de dados por meio de funções do MPI.

3.4.2.3 Mapeando tipos do C-XSC em MPI

Ambas as estratégias (criar novos tipos ou usar `MPI_BYTE`/`MPI_PACKED`) para mapear tipos de dados do C-XSC em formatos compatíveis com comunicações do MPI possuem vantagens e desvantagens. Dependendo do caso pode ser necessário uma ou outra e, a união de ambas por vezes é indispensável.

A criação de novos tipos de dados possui como principal vantagem a possibilidade de seu uso em grande parte das funções de comunicação do MPI, desde comunicação ponto a ponto, tanto síncronas como assíncronas, até comunicações coletivas. Sua principal desvantagem na integração com o C-XSC é a impossibilidade de tipos com tamanho variável, como é o caso dos vetores e matrizes do C-XSC. Porém, para os tipos de dados escalares é a forma recomendada [Grimmer 2005].

Dentre os tipos que não possuem ligação direta com tipos padrão do C/C++ o tipo `MPI_BYTE` pode ser usado para enviar uma seqüência de bytes contínua na memória. Dessa forma, em conjunto com o operador `sizeof`, onde se pode determinar o tamanho em *bytes* de um elemento em C/C++, o uso do `MPI_BYTE` pode ser considerado uma opção[2]. Essa solução pode ser diretamente empregada em tipos escalares, exceto o `dotprecision` que necessita de um tratamento especial pela característica do número [Hölbig 2005].

O `MPI_PACKED` pode ser usado para uma extensa lista de situações, desde mapeamento de tipos escalares até tipos de arranjos. A principal desvantagem dessa forma de mapeamento é que um tipo de dados `MPI_PACKED` não é um novo tipo de dados, mas sim uma cópia dos diferentes dados numa área de armazenamento auxiliar, destinada a servir como um *buffer* de comunicação. Dessa forma, o consumo de memória é duplicado, uma vez que os dados devem ser armazenados em seu formato original e também empacotados no *buffer*. Outra desvantagem é que o tipo de dados `MPI_PACKED` não pode ser usado com operações de computação global, como, por exemplo, o `MPI_Reduce`. Como principal vantagem, o uso do tipo `MPI_PACKED` permite a comunicação de dados de tamanho variável (como vetores e matrizes) e pode ter suas funções de empacotamento (`MPI_Pack`) e desempacotamento (`MPI_Unpack`) associadas a sobrecarga de rotinas de comunicação, facilitando seu uso [Grimmer 2005].

Os tipos escalares do C-XSC, exceto o tipo `dotprecision` podem ser relacionados diretamente dados do tipo `double` em C++. O que difere entre eles é a quantidade de valores `double` usados para o armazenamento. A Tabela 3.3 ilustra a relação dos tipos escalares do C-XSC com valores `double` em C/C++.

Tabela 3.3. Tipos de dados do C-XSC relacionados com tipo double em C/C++

Tipos de dados do C-XSC	Número de elementos
`Real`	1
`Interval`	2
`Complex`	2
`Cinterval`	4

Uma variável do tipo `real` pode ser diretamente mapeada como um `double` em C/C++. O tipo `interval`, por conter um intervalo de objetos do tipo *real*, é armazenado em 2 variáveis contínuas do tipo `double`. As variáveis `complex`, que armazenam números complexos (parte real e parte imaginária), são armazenadas em 2 variáveis do tipo `double`. O tipo `cinterval`, que representa um intervalo de números imaginários, contém 4 valores `double`.

3.4.3. Validando a Alta Exatidão em Clusters

Nesta seção é descrito como pode ser realizado o processo que possibilita o uso do C–XSC em *clusters* e a sua integração com a biblioteca de troca de mensagens MPI propiciando, com isso, a disponibilização da alta exatidão em *clusters* de computadores.

3.4.3.1 Uso de tipos escalares `real`, `interval`, `complex` e `cinterval`

Uma das formas de usar as propriedades do C-XSC em conjunto com o MPI é a criação de tipos de dados no MPI que mapeiem os dados do C-XSC. Para a criação de

[2] Assume-se que a arquitetura usada mapeia um char em um byte.

novos tipos de dados em MPI podem-se usar as seguintes funções ([Pacheco 1997] e [Gropp et. al 1999]):

- `MPI_Type_contiguous`: permite a criação de um novo tipo de dados com elementos do mesmo tipo continuamente alocados na memória;
- `MPI_Type_vector`: permite a criação de um novo tipo de dados com elementos do mesmo tipo parcialmente contínuos na memória, variando em blocos regulares;
- `MPI_Type_indexed`: permite a criação de um novo tipo de dados com elementos do mesmo tipo não continuamente alocados na memória;
- `MPI_Type_struct`: permite a criação de um novo tipo de dados com elementos de diferentes tipos não continuamente alocados na memória;
- `MPI_Type_commit`: disponibiliza um tipo de dados criado para ser utilizado em comunicações.

Uma vez que os dados dos objetos que representam tipos escalares estão continuamente alocados na memória, para a criação de tipos de dados em MPI que representem os tipos de dados escalares do C-XSC pode-se utilizar apenas o `MPI_Type_contiguous` seguido de um `MPI_Type_Commit`. A Figura 3.8 ilustra o protótipo dessas funções.

```
int MPI_Type_contiguous(
    int           count; /* in */
    MPI_Datatype  old_t; /* in */
    MPI_Datatype* new_t) /*out*/  (a)

int MPI_Type_commit(
    MPI_Datatype* new_t); /* in/out*/  (b)
```

Fonte: Pacheco 1997, p. 95 e p.97

Figura 3.8. Protótipos de funções do MPI para criação de tipos de dados

Com as funções descritas na Figura 3.8 pode-se criar tipos de dados do MPI para os tipos do C-XSC: `real` (`MPI_CXSC_REAL`), `interval` (`MPI_CXSC_INTERVAL`), `complex` (`MPI_CXSC_COMPLEX`) e `cinterval` (`MPI_CXSC_CINTERVAL`). O código descrito na Figura 3.9 exemplifica a criação de alguns desses tipos juntamente com uso de rotinas de comunicação ponto a ponto e de grupo.

Na Figura 3.9, até a linha 4 são estão inseridos os `includes` e diretivas que especificam o uso do MPI e de rotinas especificas do C-XSC. Para uma correta compilação recomenda-se inserir a diretiva de inclusão do MPI (`mpi.h`) antes das diretivas de inclusão de funções do C-XSC. Entre as linhas 6 e 10 são definidas as variáveis usadas pelo programa. São definidas 3 variáveis com tipos do C-XSC (linhas 6 e 7) e 3 variáveis com tipos de dados do MPI (linhas 9 e 10). Ressalta-se que na linha 10 são definidas variáveis do tipo `MPI_Datatype`, que serão usadas para a criação dos tipos do C-XSC.

A criação dos tipos de variáveis do C-XSC no MPI ocorre nas linhas 14 a 17. Inicialmente é criado o tipo `MPI_CXSC_REAL`, como sendo 1 elemento do tipo `MPI_DOUBLE` (linha 14). Antes de possibilitar que esse tipo possa ser usado em comunicações, ele serve como base para a criação do tipo `MPI_CXSC_INTERVAL`, criado como sendo 2 objetos do tipo `MPI_CXSC_REAL` (linha 15). Após os tipos

criados são liberados para serem usados em comunicações, por meio da função `MPI_Type_commit` (linhas 16 e 17).

```
1. #include <mpi.h>        // para uso do MPI
2. #include <interval.hpp> // biblioteca para uso de intervalos

3. using namespace std;
4. using namespace cxsc;

5. int main ( int argc, char ** argv ){
6.     interval a, b;
7.     real r;
8.     int meurank, procs, i, tag=10;
9.     MPI_Status status;
10.    MPI_Datatype MPI_CXSC_REAL, MPI_CXSC_INTERVAL;

11.    MPI_Init(&argc, &argv);
12.    MPI_Comm_rank(MPI_COMM_WORLD, &meurank);
13.    MPI_Comm_size(MPI_COMM_WORLD, &procs);

14.    MPI_Type_contiguous(1,MPI_DOUBLE,&MPI_CXSC_REAL);
15.    MPI_Type_contiguous(2,MPI_CXSC_REAL,&MPI_CXSC_INTERVAL);
16.    MPI_Type_commit(&MPI_CXSC_REAL);
17.    MPI_Type_commit(&MPI_CXSC_INTERVAL);

18.    if (meurank == 0){
19.       a = interval(-8.0,-2.0);
20.       cout << "Processo " << meurank << "Intervalo a = " << a << endl;
21.       for(i=1; i < procs; i++)
22.          MPI_Send(&a, 1, MPI_CXSC_INTERVAL, i, tag, MPI_COMM_WORLD);
23.       r = Sup(a) + Inf(a);
24.    }
25.    else{
26.       MPI_Recv(&b,1,MPI_CXSC_INTERVAL,0,tag,MPI_COMM_WORLD,&status);
27.       Inf(b) = Inf(b) + procs;
28.       Sup(b) = Sup(b) * meurank;
29.      cout << "Processo "<< meurank << "Intervalo b = " << b << endl;
30.    }

31.    MPI_Bcast(&r, 1, MPI_CXSC_REAL, 0, MPI_COMM_WORLD);

32.    if (meurank != 0)
33.       cout << "Processo " << meurank << " Valor real=" <<r<< endl;
34.    MPI_Finalize();
35.    return (0); }
```

Figura 3.9. Exemplo de uso de criação de tipos do C-XSC em MPI e comunicações

A partir da linha 18 inicia o código de execução. O programa segue a linha de programação SPMD onde a variável `meurank` irá diferenciar o código a ser executado por um ou outro processo. Essa variável recebe valor (linha 12) junto do bloco de inicialização do MPI (linhas 11 a 13).

O processo de `rank` zero irá definir valores para uma variável `interval` (linha 19) e mostrar esses valores na saída padrão (linha 20). Após, na linha 22 ele envia uma mensagem aos demais processos em execução com o conteúdo dessa variável `interval` (é enviado apenas 1 elemento), usando a função `MPI_Send` e o tipo de

dados explicitamente criado `MPI_CXSC_INTERVAL`. Após o(s) envio(s) ele atribui um valor para a variável real, usando função do C-XSC de manipulação de intervalos (linha 23).

Os processos com `rank` diferente de zero recebem (linha 26) a mensagem enviada usando o tipo de dados criado (`MPI_CXSC_INTERVAL`) em outra variável do tipo `interval`. Em seguida, os limites do intervalo recebido são modificados, usando as variáveis de controle do MPI e função de manipulação de intervalos do C-XSC (linhas 27 e 28). Após, os valores do intervalo são mostrados na saída padrão, redirecionada pelo MPI, seu conteúdo (linha 29).

Os processos em execução realizam um broadcast (`MPI_Bcast`) de uma variável do tipo `real` (linha 31). Essa operação de comunicação global utiliza o tipo criado (`MPI_CXSC_REAL`), possuindo seu nodo raiz apontando para o processo com `rank` zero (origem dos dados) que serão copiados para a variável "`r`" nos demais processos. Em seguida os processos que obtiveram valor para a variável real através de broadcast mostram na saída padrão o conteúdo dessa variável. A Figura 3.10 apresenta o resultado da execução da aplicação em 4 processos.

```
Processo 0 Intervalo a = [  -8.000000, -2.000000]
Processo 2 Intervalo b = [  -4.000000, -4.000000]
Processo 3 Intervalo b = [  -4.000000, -6.000000]
Processo 2 Valor real  =   -10.000000
Processo 3 Valor real  =   -10.000000
Processo 1 Intervalo b = [  -4.000000, -2.000000]
Processo 1 Valor real  =   -10.000000
```

Figura 3.10. Execução de aplicação com MPI e C-XSC que comunica real e interval

Para a compilação do programa da Figura 3.9 deve-se usar o *script* mpiCC ao invés do g++ (C-XSC) e do mpicc (MPI com c). Os parâmetros aceitos pelo mpiCC são os mesmos do g++. A Figura 3.10 apresenta o resultado dos comandos `cout`, que não necessariamente mantém a ordem dos processos em execução, uma vez que a execução é em paralelo.

```
22. MPI_Send(&a, sizeof(a), MPI_BYTE, i, tag, MPI_COMM_WORLD);

26. MPI_Recv(&b, sizeof(b), MPI_BYTE, 0, tag, MPI_COMM_WORLD, &status);

31. MPI_Bcast(&r, sizeof(r), MPI_BYTE, 0, MPI_COMM_WORLD);
```

Figura 3.11. Linhas modificadas para uso de real e interval com tipo MPI_BYTE

O mesmo exemplo da Figura 3.9 pode ser implementado sem a criação de tipos específicos do MPI, utilizando o tipo `MPI_BYTE` (seqüência continua de bytes). Nesse caso, não é necessária a criação dos tipos de dados `MPI_CXSC_*` (linhas 10 e 14 a 17). Para as comunicações, ao invés de usar um elemento do tipo criado, deve-se usar o operador `sizeof` para delimitar a quantidade a ser comunicada. A Figura 3.11 mostra as modificações nos códigos de comunicação para uso com o tipo `MPI_BYTE`.

A criação do tipo de dados, ainda que necessite a escrita de código extra, proporciona uma melhor legibilidade do programa, uma vez que a quantidade de elementos (segundo argumento das funções de comunicação) fica explicitamente definida além do tipo de dados sendo comunicado.

3.4.3.2 Uso com vetores e matrizes

O C-XSC oferece recursos para trabalhar com arranjos de dados unidimensionais ou bidimentionais com facilidades, como por exemplo, de verificação automática dos índices acessados. Dessa forma, não é aconselhável criar vetores de reais usando a definição "`real vet[N]`" e sim usar o tipo `rvector`.

Além do tipo `rvector`, o C-XSC oferece vetores para *interval* (*ivector*), `complex` (`cvector`) e `cinterval` (`civector`). Da mesma forma, tipos específicos para os tipos de dados diferenciados do C-XSC são fornecidos para matrizes. Esses tipos que compõem conjuntos de dados de mesmo tipo, também possuem a indicação dos limites de cada dimensão. Além dos limites, os tipos de dados possuem um elemento extra que armazena o tamanho de cada dimensão.

Uma vez que pode-se definir vetores e matrizes de tamanhos diferentes, a opção de criar um tipo de dados para cada tipo correspondente no C-XSC não tem sentido, uma vez teriam de ser definidos tipos de dados para cada tamanho de matriz ou vetor desejado. Então, para a comunicação de tipos de dados não escalares do C-XSC em MPI uma solução é usar o tipo de dados do MPI conhecido como `MPI_PACKED`.

O `MPI_PACKED` não é um novo tipo, mas sim um formato de dados "empacotado" para a transmissão/recepção dos dados. Antes do envio, os dados que necessitam ser transmitidos devem ser empacotados em um *buffer*. O processo receptor, após receber os dados "empacotados" deve antes de usá-los desempacotá-los, o que em outras palavras significa colocá-los no formato original. A Figura 3.12 descreve os protótipos das funções do MPI para empacotamento (`MPI_Pack`) e de desempacotamento (`MPI_Unpcak`).

```
int MPI_Pack (                          int MPI_Unpack (
    void* inbuf,           /* in */         void* inbuf,           /* in */
    int insize,            /* in */         int insize,            /* in */
    MPI_Datatype datatype, /* in */         int* position,         /*in/out*/
    void* outbuf,          /* out */        void* outbuf,          /* out */
    int outsize,           /* in */         int outsize,           /* in */
    int* position,         /*in/out*/       MPI_Datatype datatype, /* in */
    MPI_Comm comm)        /* in */          MPI_Comm comm)        /* in */
            (a)                                     (b)
```

Fonte: Pacheco 1997, p. 375-376

Figura 3.12. Protótipo das funções MPI_Pack e MPI_Unpack

Para empacotar/desempacotar dados deve-se usar uma (extensa) área de memória do tipo `char` como buffer. Após esse buffer deve preenchido com os dados por meio da função `MPI_Pack`. Cada dado empacotado deve ter um tipo do MPI definido. No preenchimento do buffer deve-se ter o cuidado de ajustar as posições, iniciado uma variável inteira com a posição zero e passando a mesma por referência. No desempacotamento, devem-se extrair do *buffer* os elementos, na mesma ordem em que foram empacotados, indicando a correta posição e o tipo do elemento, atribuindo a referência de variáveis específicas. Igualmente deve-se iniciar pela posição zero do buffer, passando a variável por referência. A Figura 3.13 ilustra um código fonte com envio e recebimento de vetores e matrizes de intervalos de reais.

Na figura 3.13 a porção inicial é semelhante ao início da Figura 3.9, com a inclusão de diretivas, declaração de variáveis, inicialização do MPI e criação de tipos. A exceção é a definição de algumas constantes. O modelo usado é o SPMD sendo o código contido entre as linhas 17 e 26 executado pelo processo com `rank` zero e o código entre as linhas 28 a 31 pelos demais processos.

O processo com `rank` zero atribui valores para a variável "`a`" que é do tipo `imatrix` (linhas 17 a 19). Após na linha 20, o conteúdo da matriz é empacotado no *buffer*, sendo empacotados `LIN * COL` elementos do tipo `MPI_CXSC_INTERVAL`. O empacotamento ocorre da referência do elemento `a[1][1]` uma vez que o C-XSC inicia os índices de vetores e matrizes em 1. Nessa operação, a variável `posicao` é passada por referência e atualizada, caso seja necessário empacotar mais elementos. Na linha 21 é realizado em *broadcast* do conteúdo empacotado.

```
1.  #include <mpi.h>        // para uso do MPI
2.  #include "imatrix.hpp"  // biblioteca para uso do tipo de dado matriz
3.  #define LIN 3           // de intervalos reais
4.  #define COL 2
5.  #define BUFFER 10000

6.  /* uso dos namespaces std e cxsc */

7.  int main ( int argc, char ** argv ){
8.     imatrix a(LIN,COL);
9.     ivector b(COL);
10.    int meurank, procs, i, j, tag=10, posicao=0;
11.    char buff[BUFFER];  // buffer para envio. Pode necessitar ser maior
12.    MPI_Status status;
13.    MPI_Datatype MPI_CXSC_REAL, MPI_CXSC_INTERVAL;

14.    /* Inicialização do MPI, obtenção de procs e meurank */
15.    /* criação dos tipos MPI_CXSC_REAL e MPI_CXSC_INTERVAL */

16.    if (meurank == 0){
17.       for(i=1; i <= LIN; i++)
18.          for(j=1; j <= COL; j++)
19.             a[i][j] = interval(-1.375 * i + j, 1.477 * i + j);
20.       MPI_Pack(&a[1][1],LIN * COL, MPI_CXSC_INTERVAL,
                   buff, BUFFER, &posicao, MPI_COMM_WORLD);
21.       MPI_Bcast(buff, posicao, MPI_PACKED, 0, MPI_COMM_WORLD);

22.       for(i=1; i < procs; i++){
23.           MPI_Recv(&b[1], COL * sizeof(interval), MPI_BYTE, i, tag, MPI_COMM_WORLD,
                     &status);
24.          cout << "Rank 0 recebeu de " << i << " vetor " << b << endl;
25.       }
26.    }
27.    else{
28.       MPI_Bcast(buff, BUFFER, MPI_PACKED, 0, MPI_COMM_WORLD);
29.       MPI_Unpack(buff, BUFFER, &posicao, &a[1][1], LIN * COL,
                     MPI_CXSC_INTERVAL, MPI_COMM_WORLD);
30.       b = a[meurank]; // verificar linhas da matriz x procs
31.       MPI_Send(&b[1], COL * sizeof(interval), MPI_BYTE, 0, tag, MPI_COMM_WORLD);
32.    }
33.    MPI_Finalize();
37.    return (0); }
```

Figura 3.13. Exemplos de comunicação de tipos ivector e imatrix

Os demais processos inicialmente realizam o *broadcast* (linha 28) para obter o buffer, realizando após o desempacotamento (linha 29) e conseqüente atribuição de valores a matriz. O desempacotamento é realizado a partir da referência do elemento `a[1][1]`, sendo colocados em seqüência LIN*COL elementos do tipo `MPI_CXSC_INTERVAL`. De forma semelhante ao empacotamento uma variável inteira de controle de posição do buffer é passada por referência e atualizada. A partir da execução do `MPI_Unpack`, todos os processos contêm a matriz a completa em seu espaço de endereçamento de memória. Este programa foi implementado considerando que se deve ter o cuidado de não especificar uma matriz com um número de linhas

menor que o número de processos em execução, sob pena de funcionamento incorreto do programa.

Na linha 30 é atribuída ao vetor (`ivector`) "`b`" uma linha da matriz "`a`". Essa atribuição é obtida por meio de funções específicas do C-XSC. A linha a ser colocada no vetor "`b`" é a equivalente ao `rank` do processo. O vetor é então enviado (linha 31) ao processo com `rank` zero, usando o tipo de dados `MPI_BYTE`. Para o computo do tamanho de bytes a serem enviados, foi usada a multiplicação do número de elementos de uma linha (COL) com o tamanho em bytes de um objeto `interval` (`sizeof(interval)`).

O processo com `rank` zero recebe os vetores enviados pelos demais processos em ordem (primeiro do processo 1, após do processo 2 e assim sucessivamente). O cálculo do número de elementos a serem recebidos segue a mesma lógica do envio, sendo após cada recebimento de vetor o mesmo mostrado na saída padrão.

O processo de envio e recebimento de vetores e matrizes segue a mesma seqüência: no envio primeiro empacota e depois envia; no recebimento: primeiro recebe e depois desempacota. Grimmer (2005) em seu trabalho implementa a sobrecarga de função de comunicação do MPI. As funções sobrecarregadas recebem os tipos de arranjos de dados do C-XSC (`rvector`, `ivector`, ...). Elas trabalham no empacotando os dados e após enviando (`MPI_Send`, por exemplo) e recebendo com desempacotamento logo em seguida (`MPI_Recv`). Como são funções de uso genérico, alguns cuidados são realizados, como o empacotamento/desempacotamento dos limites das dimensões e no caso específico do recebimento, ajuste por meio de funções do C-XSC de propriedades da matriz/vetor (limites e número de elemento das dimensões).

3.4.3.2 Uso com valores do tipo dotprecision

Segundo Grimmer, o tipo `dotprecision` foi projetado para armazenar o resultado exato de um produto escalar. Ele possui versões para as quatro bases e é armazenado em C por um vetor de elementos do tipo `unsigned long`. O espaço em memória destinado é dependente da arquitetura onde o C-XSC está instalado (2005, p. 7 e 8).

Uma vez que o tamanho de um `dotprecision` é dependente da arquitetura a opção de criação de tipo específico não é a mais adequada. Resta o uso do tipo `MPI_PACKED` ou do tipo `MPI_BYTE`. Para o uso do tipo `MPI_PACKED` deve ser apontado para o endereço inicial do ponteiro apontado pelo objeto. O uso do tipo `MPI_BYTE` nesse caso torna-se mais simples e direto, bastando realizar um *type casting* numa atribuição do ponteiro. O código da Figura 3.14 ilustra um exemplo de envio e recebimento de dados do tipo `dotprecision`.

Destaca-se na Figura 3.14 as linhas 9 e 10 onde são definidas as variáveis que irão possibilitar o uso de variáveis do tipo `dotprecision`. Na linha 9, é declarado um ponteiro para uma variável do tipo `unsigned long`. As comunicações das variáveis `dotprecision` serão realizadas a partir desse ponteiro, que recebe a referência da variável `dotprecision accu_p` na linha 14 (com uso de um *type casting* para um ponteiro de ponteiro do tipo `unsigned long`).

Entre as linhas 17 e 22 os processos com `rank` diferente de zero inicializam 2 vetores. Após, nas linhas 23 e 24 é realizado o produto escalar dos 2 vetores. Essas ações ocorrem em paralelo em todos os processos. Na linha 25 o valor do produto escalar (parcial) calculado no processo é enviado ao processo com `rank` zero. No envio, é usada a constante do C-XSC `BUFFERSIZE` para delimitar o número de

elementos a serem transmitidos e o tipo `MPI_BYTE`. Segundo Grimmer (2005) e Hölbig (2005) é por meio dessa constante que um objeto do tipo `dotprecision` é alocado.

```
1.  #include <mpi.h>        // para uso com o MPI
2.  #include <rvector.hpp>  // biblioteca para uso do tipo de dado vetor
3.  #define N 1000000       // de reais
4.  /* uso dos namespaces std e cxsc */
5.  int main(int argc, char ** argv){
6.     int i;
7.     real result;
8.     rvector v_a(N), v_b(N);
9.     unsigned long *pep;
10.    dotprecision accu, accu_p;
11.    int meurank, procs, tag=10;
12.    MPI_Status status;
13.    /* Inicialização do MPI, obtenção de procs e meurank */
14.    pep = *(unsigned long **)(&accu_p);
15.    accu_p = accu = 0.0;
16.    if (meurank != 0){
17.        for (i=1; i<=N ; i+=4){
18.            v_a[i] = meurank * 10E+50; v_a[i+1] = meurank * 1.1;
19.            v_a[i+2] = meurank * 10E+50; v_a[i+3] = meurank * 1.25;
20.            v_b[i] = 1.0; v_b[i+1] = 1.0;
21.            v_b[i+2] = -1.0; v_b[i+3] = -1.0;
22.        }
23.        for (i=1; i<=N; i++)
24.            accumulate(accu_p, v_a[i], v_b[i]);
25.        MPI_Send(pep, BUFFERSIZE, MPI_BYTE, 0, tag, MPI_COMM_WORLD);
26.    }
27.    else{
28.       for(i=1; i<procs; i++){
29.          MPI_Recv(pep, BUFFERSIZE, MPI_BYTE, i, tag, MPI_COMM_WORLD, &status);
30.          accu += accu_p;
31.       }
32.       result = rnd(accu);
33.       cout.precision(30);
34.       cout << "Produto Escalar Paralelo  = " << result << endl;
35.    }
36.    MPI_Finalize();
37.    return 0; }
```

Figura 3.14. Exemplo de produto escalar em paralelo com uso de dotprecision

O processo com `rank` zero recebe as mensagens dos demais processos, usando o ponteiro `pep` e a constante `BUFFERSIZE` além do tipo `MPI_BYTE` (linha 29). Após o recebimento dos produtos escalares parciais é calculado o produto escalar geral, acumulando os valores recebidos (linha 30). Ao final, o processo mostra o resultado do produto escalar calculado em paralelo na saída padrão.

3.5. Alternativas de Otimização

O desempenho computacional obtido pela biblioteca C-XSC nas mais diversas implementações demonstra a necessidade do estudo de novas alternativas de otimização para suas rotinas básicas. Um C-XSC “mais rápido” possibilitaria sua efetiva utilização na resolução de aplicações reais de grande porte que necessitassem de uma melhor exatidão (não conseguida com as ferramentas computacionais tradicionais). Entre estas

alternativas pode-se citar a otimização da biblioteca através da implementação de novos algoritmos (rápidos) de somatório e/ou de produto escalar, de alterações na forma de implementação das suas atuais rotinas, da inclusão em suas rotinas de rotinas da BLAS, das funções SSE (*Streaming SIMD Extensions*) e do uso das diretivas de otimização do compilador gcc. Algumas destas possibilidades de alternativas já estão sendo pesquisadas por alguns grupos de pesquisa do Brasil e Europa e serão citadas nas seções seguintes.

3.5.1. Otimizando através do uso de diretivas do compilador gcc

Abstraindo-se as características da máquina na qual o software será executado, o tempo de execução de um programa vai depender, principalmente, da quantidade e do tipo de instruções utilizadas. Ao optar pela codificação de seus programas usando uma linguagem de alto nível, porém, o programador delega ao compilador a geração do executável. Desta forma, a qualidade do binário produzido pelo compilador é um dos fatores que interferem no desempenho da aplicação.

Uma das formas de melhorar o desempenho de uma aplicação seja ela seqüencial ou paralela, é fazer uso das opções de otimização oferecidas pelo compilador. No caso do gcc/g++, estas opções estão disponíveis através da diretiva que indica um dos níveis de otimização (–O) ou de diretivas independentes (-f) que permitem ajuste fino nas opções. Na seqüência são apresentadas as diretivas que correspondem aos diversos níveis de otimização, conforme FSF (2007):

- –O0 (default): Realiza a compilação da forma mais simples e rápida possível, sem nenhuma otimização do código;
- –O1 (ou –O) : Primeiro nível de otimização. Ativa 19 diretivas independentes que tem por objetivo reduzir o tamanho do código gerado e o tempo de execução: -fdefer-pop, -fdelayed-branch, -fguess-branch-probability, -fcprop-registers, -floop-optimize, -fif-conversion, -fif-conversion2, -ftree-ccp, -ftree-dce, -ftree-dominator-opts, -ftree-dse, -ftree-ter, -ftree-lrs, -ftree-sra, -ftree-copyrename, -ftree-fre, -ftree-ch, -funit-at-a-time, -fmerge-constants;
- –O2 : Segundo nível de otimização. Adiciona 28 diretivas independentes àquelas da opção –O1: -fthread-jumps, -fcrossjumping, -foptimize-sibling-calls, -fcse-follow-jumps, -fcse-skip-blocks, -fgcse, -fgcse-lm, -fexpensive-optimizations, -fstrength-reduce, -frerun-cse-after-loop, -frerun-loop-opt, -fcaller-saves, -fpeephole2, -fschedule-insns -fschedule-insns2, -fsched-interblock -fsched-spec, -fregmove, -fstrict-aliasing, -fdelete-null-pointer-checks, -freorder-blocks -freorder-functions, -falign-functions -falign-jumps, -falign-loops -falign-labels, -ftree-vrp, -ftree-pre;
- –O3: Mais avançado nível de otimização, podendo resultar em executável maior. Adiciona outras três diretivas às ativadas por –O2: -finline-functions, -funswitch-loops, -fgcse-after-reload;
- –Os: Otimiza para o menor tamanho do binário gerado, em detrimento do tempo de execução.

Para ilustrar o efeito do uso destas opções na compilação de um programa, apresenta-se a descrição de algumas das diretivas individuais citadas, de acordo com FSF (2007):

- defer-pop: não desempilha imediatamente os argumentos passados para uma função após o retorno desta, de forma a minimizar modificações no *stack pointer*.
- loop-optimize: otimizações em laços incluindo a transferência de expressões constantes para fora do *loop* e simplificação nos testes de saída;
- tree-dce: Eliminação de código não atingível (*dead code elimination*) e código irrelevante para o programa;
- fthread-jumps: simplificação em casos de desvio condicional que possuem como alvo outro desvio baseado na mesma condição;
- finline-functions: insere o código de funções simples diretamente no ponto em que estão sendo chamadas;
-

Tabela 3.4. Tempo de execução de operações básicas

Otimização	*Operações*	`double` Ox	`real` Ox	`real` ∇x	`real` Δx
-O0	Somas	750	1850	8730	9050
	Subtrações	710	1950	8660	8670
	Multiplicações	830	1950	8570	42960
	Divisões	1950	3330	10050	10040
	TOTAL	**4240**	**9080**	**36010**	**70720**
-O1	Somas	840	750	7310	7210
	Subtrações	780	750	7480	7470
	Multiplicações	780	760	7470	30020
	Divisões	2020	2050	8600	8350
	TOTAL	**4420**	**4310**	**30860**	**53050**
-O2	Somas	760	720	7060	7060
	Subtrações	760	760	7070	7150
	Multiplicações	770	760	7280	30330
	Divisões	2030	2030	8330	8320
	TOTAL	**4320**	**4270**	**29740**	**52860**
-O3	Somas	720	720	7280	7350
	Subtrações	760	750	7290	7260
	Multiplicações	750	760	7280	30590
	Divisões	2030	2030	8350	8480
	TOTAL	**4260**	**4260**	**30200**	**53680**
-O3 -funroll-loops	Somas	330	330	7240	7200
	Subtrações	330	320	7240	7230
	Multiplicações	320	330	7310	30180
	Divisões	1580	1580	8540	8590
	TOTAL	**2560**	**2560**	**30330**	**53200**

Para verificar o tempo de execução de um programa que faz uso do C-XSC, realizou-se a compilação da biblioteca e do programa de *benchmark* fornecido com ela, empregando-se cada uma das opções de otimização acima citadas, com exceção da -Os, e adicionando-se o *flag* -funroll-loops à opção -O3. Estes testes foram realizados em computador com processador AMD Athlon XP-M 2400+ (1.6 Ghz) com 512 MiB de

RAM e 512 KiB de cache L2, utilizando sistema operacional GNU/Linux (*kernel* 2.6.22), compilador g++ versão 4.1.2 e biblioteca C-XSC versão 2.2.2. Todos os tempos estão representados em milisegundos.

A Tabela 3.4 exibe o tempo de execução de operações básicas de soma, subtração, multiplicação e divisão, num total de 100000000 repetições para cada operação, utilizando-se o tipo nativo `double` e o tipo do C-XSC `real`. Neste último caso, utilizou-se arredondamento para o número mais próximo de máquina ($\bigcirc x$), arredondamento para baixo (∇x) e arredondamento para cima (Δx).

Percebe-se que a otimização do código geralmente resulta em redução do tempo de execução do *benchmark*. No caso das operações efetuadas com o tipo real do C-XSC, usando-se as opções -O3 e -funroll-loops para a compilação, observa-se uma redução de aproximadamente 25% do tempo de execução, quando comparado com a execução do código gerado sem nenhuma otimização. Uma exceção encontrada foi nas execuções da operação de divisão, quando empregado o tipo nativo `double`, caso onde houve um pequeno acréscimo no tempo de execução com a otimização.

Tabela 3.5. Tempo de execução de operações intervalares

Otimização	*Operações*	`double`	`real`	`interval`
-O0	Somas	710	2000	20160
	Subtrações	810	1960	22780
	Multiplicações	750	1880	63260
	Divisões	2120	3080	28760
	TOTAL	**4390**	**8920**	**134960**
-O1	Somas	830	750	16710
	Subtrações	760	750	19220
	Multiplicações	1010	870	41100
	Divisões	2080	2080	18910
	TOTAL	**4680**	**4450**	**95940**
-O2	Somas	720	730	17640
	Subtrações	1100	1000	17170
	Multiplicações	730	830	47290
	Divisões	2020	2020	18960
	TOTAL	**4570**	**4580**	**101060**
-O3	Somas	760	760	17280
	Subtrações	790	830	18110
	Multiplicações	920	930	48240
	Divisões	2050	2010	19090
	TOTAL	**4520**	**4530**	**102720**
-O3 -funroll-loops	Somas	750	760	16700
	Subtrações	730	760	17020
	Multiplicações	730	730	47410
	Divisões	1980	1980	19660
	TOTAL	**4190**	**4230**	**100790**

Na Tabela 3.5 são apresentados os resultados da execução de 100000000 de operações intervalares, executadas nas mesmas condições do teste anterior. Nestes

testes se destaca a redução no tempo de execução para o tipo `interval`, quando utilizada a opção –O1, que ficou em torno de 29%.

A Tabela 3.6 exibe os tempos de execução exigidos para a execução de 1000 repetições do cálculo do produto escalar para um vetor de 20000 elementos.

Percebe-se novamente o ganho no tempo de execução resultante da otimização do código, em qualquer uma das opções, quando comparado com o programa gerado sem nenhuma otimização.

No caso do cálculo empregando-se o tipo `real`, compilado com as opções -O3 e -funroll-loops, o tempo de execução foi de aproximadamente 50% daquele obtido sem otimizações.

3.5.2. Otimizando através do uso de instruções SIMD

Uma segunda alternativa para aumentar o desempenho das aplicações que utilizam a biblioteca C-XSC está sendo investigada: A otimização da biblioteca por meio de recursos oferecidos no nível da arquitetura de conjunto de instruções, particularmente instruções SIMD (*Single Instruction Multiple Data*).

Tabela 3.6. Tempo de execução de operações intervalares

Otimização	`double`	`real`	`interval`
-O0	210	10800	21550
-O1	160	6150	13100
-O2	180	5890	12530
-O3	150	5920	12430
-O3 -funroll-loops	150	5470	11690

No momento estão sendo analisadas as instruções disponibilizadas pela arquitetura IA-32 (x86), em função da popularidade desta e visto que a IA-32 possui algumas extensões arquiteturais que oferecem suporte à operações SIMD: MMX, SSE, SSE2, SSE3 e SSE4.

A tecnologia MMX introduziu um conjunto de oito registradores de 64 bits e instruções para realizar operações SIMD em *bytes*, *words* e *doublewords* contidos nestes registradores, segundo Intel (2005).

A tecnologia SSE (*Streaming SIMD Extension*) acrescentou oito registradores de 128 bits (XMM) com instruções para manipulação de quatro valores empacotados, representados em ponto-flutuante com precisão simples, além outras instruções diversas como para controle da memória *cache*. Já a extensão SSE2 suporta operandos empacotados como dois valores em precisão dupla, tanto em registradores XMM como em memória, aritmética SIMD em inteiros de 64 *bits*, além de algumas instruções variadas adicionais. Posteriormente, a tecnologia SSE3 acrescentou 13 novas instruções, permitindo, inclusive computação horizontal sobre os operandos. Estas novas instruções são úteis, segundo Intel (2005), para aplicações científicas, de processamento de vídeo e multi-threaded. A tecnologia SSE4, recentemente lançada com os novos processadores Intel construídos com tecnologia de 45 nm, adicionou 54 novas instruções à arquitetura (Intel, 2007). Dentre estas, destaca-se as duas instruções que tem por objetivo calcular o produto escalar de dois valores em precisão dupla

empacotados em um registrador XMM ou armazenados em memória (instrução DPPD), ou quatro valores em precisão simples (DPPS).

3.6. Exemplos de Aplicações de Alta Exatidão em Ambientes Paralelos

A resolução de aplicações reais com alta exatidão é um desafio que vem sendo enfrentado a algumas décadas. O principal obstáculo, como comentado no início da Seção 3.5, diz respeito ao desempenho computacional das ferramentas que disponibilizam esta característica no computador. Devido a este fator várias pesquisas foram desenvolvidas visando, em um primeiro momento, a integração destas ferramentas de alta exatidão em ambientes paralelos e, atualmente, as pesquisas estão direcionando-se a busca de melhorias no desempenho destas ferramentas, tanto em suas versões seqüenciais como nas versões para ambientes paralelos.

Pode-se dizer que, principalmente a partir das décadas de 80 e 90, algumas pesquisas começaram a destacar a importância e o uso da Validação Numérica (ou de algumas de suas características) em ambientes paralelos. Por exemplo, em 1994, a revista *Interval Computations* publicou um número especialmente dedicado ao desenvolvimento de algoritmos paralelos utilizando a Validação Numérica. Nesta revista, Kreinovich e 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 se pode concluir a importância do estudo deste paradigma 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 Validação Numérica.

Além disso, alguns grupos de pesquisa estão desenvolvendo trabalhos que podem ser relacionados ao tema deste curso. 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 *workstations* (http://www.nd.edu/~cgau/hpc.html). Na Escola Normal Superior 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://perso.ens-lyon.fr/nathalie.revol/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://interval.louisiana.edu/GlobSol/download_GlobSol.html). No site Interval Computation é uma área específica que descreve várias aplicações reais que estão sendo solucionadas com a ajuda da alta exatidão e das demais técnicas da Validação Numérica (http://www.cs.utep.edu/interval-comp/appl.html).

Do ponto de vista da integração de ferramentas computacionais com alta exatidão em ambientes paralelos é importante comentar as pesquisas que foram desenvolvidas na Universidade de Karlsruhe que abordaram aspectos da Validação Numérica e da resolução de problemas numéricos em ambientes paralelos. Este levantamento foi apresentado em agosto de 2005 pelo professor Gerd Bohlender da Universidade de Karlsruhe (Alemanha) durante sua palestra na Pontifícia Universidade

Católica do Rio Grande do Sul. Desta apresentação podem-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 e Davidenkoff 1992];
- Otimização global verificada em computadores paralelos (Wiethoff – 1997);
- Solução verificada de sistemas lineares usando a linguagem C#++ (Kersten – 1998).

Além destes trabalhos pode-se incorporar a esta lista as seguintes pesquisas que comprovam a eficiência do MPI em aplicações com C-XSC:

- Uso da aritmética intervalar no cálculo do fluxo de potências [Barboza, Dimuro e Reiser 2004];
- Integração do C-XSC em *clusters* de computadores (nos testes foram utilizados os *clusters* da UPF, do II-UFRGS e da Universidade de Wuppertal - Alemanha) – 2005 [Hölbig 2005];
- Criação de uma *interface* no MPI para uso do C-XSC – 2005 [Grimmer 2005];
- Paralelização de solvers intervalares para a resolução de sistemas lineares densos e esparsos – em andamento ([Hölbig 2006], [Carmo 2007] e [Kolberg 2006]).

Atualmente o grupo de Pesquisa de Computação Paralela e Sistemas Distribuídos (ComPaDi) da UPF desenvolve pesquisas, em conjunto com pesquisadores das Universidades de Wuppertal e Karlsruhe, relacionadas com a otimização da biblioteca C-XSC, seu uso em clusters de computadores e sua integração com a biblioteca MPI e com a resolução paralela de sistemas de equações lineares densos e esparsos utilizando as características da Validação Numérica ([Hölbig et. al 2002, 2005a, 2006]).

3.7. Referências Bibliográficas

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.

BORGES, P.A.P.; PADOIN, E.L. Exemplos de Métodos Computacionais Aplicados a Problemas na Modelagem Matemática. In: ESCOLA REGIONAL DE ALTO DESEMPENHO, 6., 2006, Ijuí, Brasil. **Anais. . .** Ijuí: SBC, 2006. p.5–20.

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.

CÁCERES, E.N.; SONG, S.W. **Algoritmos Paralelos usando CGM/PVM/MPI: Uma Introdução**. Texto preparado para o XXI Congresso da Sociedade Brasileira da Computação, Jornada de Atualização em Informática. Disponível em <http://www.ime.usp.br/~song/papers/jai01.ps.gz>. Acesso em 14 Fev. 2004.

CARMO, A.B.; HÖLBIG, C.A.; REBONATTO, M.T. Paralelização do Solver Intervalar Linear System Solver. In: WSCAD-CTIC, I, 2007, Gramado. **Anais...** Porto Alegre: Universidade Federal do Rio Grande do Sul, 2007.

CAVALHEIRO, Gerson G. H. Princípios da Programação Concorrente. In: **Caderno dos Cursos Permanentes das Escolas Regionais de Alto Desempenho**. Porto Alegre: Instituto de Informática – UFRGS. 2006, 74p.

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

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

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.

FSF. **Optimize Options - Using the GNU Compiler Collection (GCC)**. Disponível em http://gcc.gnu.org/onlinedocs/gcc-4.1.2/gcc/Optimize-Options.html Acesso em Dez. 2007.

GRAMA, Ananth et. al. **Introduction to Parallel Computing**. Second Edition, 2003. Redwood City: Addison Wesley. 2003, 656p.

GRIMMER, M. **An MPI Extension for the Use of C-XSC in Parallel Environment**. Preprint 2005/3, Universität Wuppertal, 2005. Disponível por http em <http://www.math.uni-wuppertal.de/wrswt/preprints/prep_05_3.pdf>. Acesso em 27 dez. 2007.

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

GROPP, Wiliam; LUSK, Ewing; SKJELLUM, Anthony. **Using MPI: portable parallel programming with the message-passing interface**. 2nd ed. Cambridge: The MIT Press, 1999. 371p. ISBN: 0-262-57134-X.

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

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

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).

HÖLBIG, C. A. et al. 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. et. al. 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. A. **Ambiente de Alto Desempenho com Alta Exatidão para a Resolução de Problemas**. 2005. Tese (Doutorado 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. Use of C-XSC Interval Library on Cluster Computers, In: INTERNATIONAL CONFERENCE ON NUMERICAL ANALYSIS AND APPLIED MATHEMATICS, 4., 2006. Hersonisos, Greece. **Proceedings…** Weinhein:Wiley-VCH Verlag, 2006, v.1, pp.151-154.

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

IGNÁCIO, Aníbal Alberto Vilcapona; FERREIRA FILHO, Virgílio José Martins. **MPI: Uma ferramenta para implementação paralela**. Pesquisa Operacional. Rio de Janeiro, v.22, n.1, p.105-116, janeiro a junho de 2002. ISSN: 0101-7438.

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

INTEL **IA-32 Intel® Architecture Optimization Reference Manual**. Intel Corporation. 2005.

INTEL **Intel® SSE4 Programming Reference**. Intel Corporation. Disponível em <http://www.developers.net/intelisdshowcase/view/2550>. Acesso em Dez. 2007.

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.

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

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

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

KOLBERG, M. L. ; BALDO, L.; VELHO, P.; WEBBER, T.; FERNANDES, L.G.; FERNANDES, P.; CLAUDIO, D.M. Parallel Selfverified Method for Solving Linear Systems. In: INTERNATIONAL MEETING OF HIGH PERFORMANCE COMPUTING FOR COMPUTATIONAL SCIENCE, 7., 2006, Rio de Janeiro, Brazil. **Proceedings. . .** Rio de Janeiro, 2006. p. 179-190.

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**. 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. **A New Approach to Scientific Computation**. New York: Academic Press, 1983.

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

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.

MESSAGE PASSING INTERFACE FORUM. **MPI-2: Extensions to the Message-Passing Interface**. Disponível em <http://www.mpi-forum.org/docs/mpi-20.ps.Z>. Acesso em 1 Mar. 2004.

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.

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.

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

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.

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

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.

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).