O Uso da biblioteca de Alta Exatidão C-XSC no Cluster LabTeC

Paulo Sérgio Morandi Júnior, Bernardo Frederes Krämer Alcalde, Carlos Amaral Hölbig, Tiarajú Asmuz Diverio

Instituto de Informática e PPGC - Universidade Federal do Rio Grande do Sul
Caixa Postal 15064 – 90501-970 Porto Alegre, RS
{sergio, bfkalcalde, holbig, diverio}@inf.ufrgs.br
Universidade de Passo Fundo e PPGC - UFRGS
99001-970 Passo Fundo, RS

Introdução

A busca de qualidade numérica nos cálculos torna-se mais evidente em aplicações que necessitam da realização de uma grande quantidade de operações em ponto flutuante [DIV 96]. Uma alternativa para controlar esses erros é se fazer uso da Computação Verificada. Entende-se por Computação Verificada o processamento numérico de problemas, utilizando a aritmética de alta exatidão, os métodos intervalares de inclusão e a convergência garantida pelo Teorema de Ponto Fixo de Brouwer [KUL 81].

Esse ramo da Computação Científica ou da Matemática Computacional busca melhorar a qualidade numérica dos cálculos em ponto flutuante em computadores. Porém, o fato de se utilizar mais controles sobres os dados (Computação Verificada), torna as aplicações naturalmente mais lentas, o que pode ser decisivo em aplicações que exigem alto desempenho. Buscando uma alternativa de aliar essa qualidade numérica dos cálculos e a alta exatidão, essa pesquisa visa construir uma biblioteca de alta exatidão para agregados de computadores.

Neste artigo apresenta-se os primeiros passos em direção a esse objetivo, mostrando a integração do C-XSC com uma biblioteca paralela (biblioteca *MPICH*, nesse caso). Com a disponibilização dessa ferramenta no cluster será possível o desenvolvimento de métodos numéricos computacionais, necessários para a solução de problemas reais de grande porte, com alta exatidão e alto desempenho.

C-XSC

A Aritmética de alta exatidão é aritmética computacional de ponto flutuante baseada no padrão IEEE-754 [IEE 85], acrescida de arredondamentos direcionados e do cálculo do produto escalar e somatórios em registradores especiais, que permitem que valores parciais sejam armazenados sem arredondamentos, resultando que o valor final dessas operações difira do valor real por apenas um arredondamento (por isso o termo *máxima exatidão*). Para dar suporte a matemática intervalar e a computação verificada (aritmética de alta exatidão) foram criadas as linguagens XSC (e**X**tensions for **S**cientific

Computing). Desenvolvida na Alemanha pela Universidade de Karlsruhe e mantida atualmente pela Universidade de Wuppertal, o C-XSC ([KLA 93]) é uma biblioteca desenvolvida para C++ com todas as características citadas acima. O suporte a tipos abstratos de dados e sobrecarga de operadores e funções da linguagem C++ facilitou a implementação de novos tipos de dados (como *rmatrix*, matrizes de reais) e de operações sobre esses dados. O objetivo do C-XSC é servir de interface entre a computação científica e a programação para C++.

O C-XSC torna o computador mais podero aritmeticamente e reduz significativamente a carga de programação, com uma notação próxima da notação matemática usual. Os operadores que foram redefinidos na biblioteca possuem máxima exatidão. Existem também bibliotecas de resolução de problemas numéricos (chamados de *Toolboxes*) escritas em C-XSC. Essas bibliotecas são um conjunto de rotinas que resolvem problemas comuns da análise numérica, como resolução de sistemas lineares, diferenciação automática e cálculo de raízes de um polinômio, que garantem resultados de alta exatidão ([MOR 2003] e [HAM 95]).

Integrando o C-XSC e o MPI no Cluster LabTeC

Nesta pesquisa procurou-se uma maneira de integrar o C-XSC à biblioteca paralela *MPICH*. Para isso foi observado o comportamento do C-XSC no ambiente de alto desempenho (compilação e instalação) para depois realizar testes com a biblioteca *MPICH* enviando e recebendo *tipos primitivos* do C-XSC (como *imatrix*, matrizes de intervalos, ou *complex*, tipos complexos). Esses testes, que serão descritos mais a frente, foram realizados junto ao Cluster presente no LabTeC (Laboratório de Tecnologias em Cluster - DELL/UFRGS - *http://www.inf.ufrgs.br/labtec*). O cluster tem como configuração atual:

- Servidor de acesso ao cluster (front-end): Dual Pentium IV Xeon 1.8 GHz (Hyper Threading), 1 GB de memória RAM, HD SCSI de 36 GB, placa de rede Gigabit Ethernet;
- Configuração de cada Nó (total de 20 nós): Dual Pentium III 1.1 GHz, 1 GB de memória RAM, HD SCSI de 18 Gb, placa de rede Gigabit Ethernet;
- Switch da 3Com 3300 TM, com 24 portas Fast Ethernet (100 Mbps) e uma porta Gigabit Ethernet (1000 Mbps). É por meio deste switch que é feita a comunicação dos nós do cluster entre si (portas Fast Ethernet) e dos nós com o servidor (porta Gigabit Ethernet).

Testando a Integração

Foram realizados testes simples, enviando dados (através da primitiva *MPI_Send*) do servidor para um dos nodos (dois processos, um rodando no servidor e outro no nodo). Para tipos mais simples como matrizes e vetores do tipo real, intervalo e complexo, intervalos do tipo complexo e real e os próprios tipos complexo e real, os testes que seguiram a seguinte forma, foram suficientes para transmintir esses tipos de dados:

```
MPI_Send(&(var),sizeof(var),MPI_BYTE,dest,tag,comgroup)
```

onde **var** pode ser qualquer um dos tipos citados acima.

Porém, quando se realiza o mesmo teste para dados do tipo *dotprecision* (bem como para *idotprecision*, *cdotprecision*, *idotprecision* e *cidotprecision*) ocorre um erro quando tenta-se imprimir o resultado recebido (*Segmentation Fault*). Lembrando que esse tipo de erro ocorre quando um processo de usário tenta fazer leitura (ou escrita) em uma posição de memória inválida para o contexto do mesmo processo. O *dotprecision* trata-se de uma variável especial responsável pela realização de operações com máxima exatidão.

Diante disso, foi realizado um teste diferente do anterior, mais parecido ao que um programa mais sofisticado estaria fazendo, calculando um produto escalar ótimo, porém sem deixar de lado a simplicidade do teste para facilitar a depuração. O teste criava 4 processos, 1 principal e 3 escravos, onde, o processo principal dividia dois vetores entre os escravos, que, por sua vez, calculavam um produto escalar ótimo parcial e enviavam esse resultado parcial (sem arredondamento) para o processo principal.

Esse recebia e somava os três resultados parciais e arrendondava para uma resposta final. Obviamente, durante a execução desse problema, o erro persistiu. Utilizando algumas primitivas do C++ (*sizeof*), percebeu-se que os processos escravos estavam enviando apenas 4 bytes para o processo principal. Consultando [KLA 93], percebe-se que a variável *dotprecision* ocupa cerca de 529 bytes em memória. Logo, o que estava sendo enviando era um ponteiro para uma área de memória, o que não é o que se deseja, pois esse endereço é local não global. O que se deseja é efetivamente enviar o conteúdo da variável do tipo *dotprecision* (e suas variantes).

Utilizando um depurador paralelo desenvolvido no grupo GPPD da II-UFRGS (PADI - depurador de programas paralelos em fase de desenvolvimento), descobriu-se que a variável *dotprecision* é do tipo apontador para *unsigned longint*. Diante desse fato, foi implementado um *type casting* para que o MPI conseguisse enviar o conteúdo do *dotprecision*. Esse *type casting* teste seguiu o seguinte formato:

```
void *envia;
dotprecision accu;
envia = *(void **)(&accu);
```

Observe que o *void* foi utilizado no lugar de *unsigned longint* para facilitar ainda mais a programação. Durante o envio ocorre uma particularidade. Observando o código fonte do C-XSC, verificou-se que ele, para alocar o *dotprecision* na memória, utiliza uma constante chamada ***BUFFERSIZE***. Desta maneira a primitiva do *MPI_Send* ficou da seguinte forma:

```
MPI_Send(envia,BUFFERSIZE,MPI_BYTE,destino,tag,mpicomgroup)
```

Os procedimentos de recebimento são análogos e também exigem um *type casting* de recebimento:

```
void *recv;
dotprecision accu2;
recv = *(void **)(&accu2);
```

```
MPI_Recv(recv,BUFFERSIZE,MPI_BYTE,orig,tag,comgroup,mpistat)
```

Com isso conseguiu-se verificar que o C-XSC e o MPI funcionaram corretamente. Como podemos perceber, o tempo de execução não foi levado em consideração nesse momento, pois o objetivo do trabalho era verificar se o MPI enviava corretamente os tipos primitivos do C-XSC.

Conclusões e Trabalhos Futuros

A preocupação com a qualidade do resultado é necessária quando se esta resolvendo problemas numéricos. A biblioteca C-XSC mostra-se eficiente no uso da Computação Verificada. Porém o uso da computação verificada aumenta naturalmente o tempo de execução dos algoritmos, devido aos controles que são necessários para se obter a alta exatidão. Diante disso, a otimização dessa biblioteca será um fator importante para o transcorrer dessa pesquisa.

Esse trabalho mostrou que é possível se obter uma integração do C-XSC com a biblioteca MPICH. Atualmente se está adaptando o algoritmo do Gradiente Conjugado para funcionar com os tipos especiais do C-XSC visando, posteriormente, a paralelização do método para disponibilizá-lo no Cluster LabTeC. Também está em andamento o estudo de algoritmos que resolvem sistemas lineares esparsos com matrizes de coeficientes do tipo banda utilizando a Computação Verificada. Com o desenvolvimento dessas atividades procuraremos responder algumas questões do tipo: O que vale mais a pena: Paralelizar um método intervalar existente ou introduzir a Computação Verificada (ou algumas de suas técnicas) em um algoritmo paralelo? O quanto mais demorado é um algoritmo de C-XSC paralelo em relação ao algoritmo convencional (sem C-XSC)?

Referências

[DIV 96] DIVERIO, T. A.; FERNANDES, Ú. A. L.; CLAUDIO, D. M. Errors in vector processing and the library libavi.a. **Reliable Computing**, New York, v.2, n.2, p.103–109, 1996.

[HAM 95] HAMMER, R. et al. **C-xsc toolbox for verified computing i**: basic numerical problems. New York: Springer-Verlag, 1995.

[IEE 85] IEEE. IEEE Standart 754-1985 for Binary Floating-Point Arithmetic, IEEE 754. New York, 1985.

[KLA 93] KLATTE, R. et al. **C-xsc - a c++ class library for extended scientific computing**. New York: Springer-Verlag, 1993.

[KUL 81] KULISCH, U.; MIRANKER, W. **Computer arithmetc in theory and practice**. New York: Academic Press, 1981.

[MOR 2003] MORANDI JÚNIOR, P. S. et al. Implementação do módulo lss em c-xsc: solucionador de sistemas lineares com alta exatidão. In: III ESCOLA REGIONAL DE ALTO DESEMPENHO, 2003, Santa Maria, Brasil. **Anais...** Santa Maria: Universidade Federal de Santa Maria, 2003. v.1, p.256–268.