Paralelização de *Solver* Intervalar para a Resolução de Sistemas Lineares Esparsos

Antonio Argeu Moreira de Lima[1], Prof. Dr. Carlos Amaral Hölbig

Curso de Ciência da Computação – Universidade de Passo Fundo (UPF)
Campus 1 – BR 285 – Bairro São José – CEP 91501-970 – Passo Fundo – RS – Brasil
{75956, holbig}@upf.br

1. Introdução

A resolução de sistemas lineares esparsos do tipo $Ax = b$, onde A é a matriz de coeficientes, b o vetor de termos independentes e x o vetor solução, é um dos principais problemas da álgebra linear e representa o núcleo de muitos problemas na engenharia e na computação científica. A importância do estudo de sistemas esparsos não se deve somente por ter aplicação em diversos problemas reais, mas também por envolver algoritmos e estruturas de dados mais complexas que em sistemas densos [Kumar 1994].

Este trabalho tem por meta apresentar uma versão paralela do *solver* BAND. O *solver* verificado BAND atua na resolução de sistemas lineares com matrizes esparsas do tipo banda e corresponde a um método numérico desenvolvido através da biblioteca de alta exatidão C-XSC. Além disso, os algoritmos utilizados no *solver*, possibilitam a geração de resultados com alta exatidão e com controle automático de erros.

Com a finalidade de melhorar o desempenho do *solver*, foi desenvolvida uma versão paralela do BAND para máquinas paralelas do tipo *clusters* de computadores, pois o *solver* apresentou um alto tempo de processamento na avaliação realizada. A partir desta avaliação, pôde-se observar que o desempenho do *solver* é prejudicado em um nível mais elevado pelo aumento da largura de banda da matriz do que pelo aumento da dimensão.

O desenvolvimento da versão paralela do BAND foi realizada com uso da biblioteca MPI e através de *threads* OpenMP. Com o auxílio da biblioteca MPI, foi implementado um algoritmo de divisão e conquista na fase de refinamento iterativo do *solver*. Para a realização da comunicação entre processos, foram utilizadas rotinas de comunicação coletiva da biblioteca MPI, mais especificamente de uma adaptação da biblioteca MPI desenvolvida por [Almeida 2008], com suporte para determinados tipos de dados do C-XSC. Além disso, algumas rotinas internas do *solver* foram paralelizadas com a biblioteca OpenMP.

Este trabalho é uma extensão da tese de doutorado de [Hölbig 2005] e corresponde a um dos trabalhos que fazem parte do projeto em cooperação internacional com as universidades alemãs Wuppertal e Karlsruhe para a otimização da biblioteca C-XSC.

---

[1] Bolsista PIBIC-CNPq

2. *Solver* BAND seqüencial

O *solver BAND* tem por objetivo a resolução de sistemas de equações lineares esparsos com matrizes do tipo banda. As matrizes do tipo banda são matrizes onde os elementos não-nulos são alocados em faixas (*bandwidths*) em torno da diagonal principal.

Para matrizes do tipo banda, o algoritmo utilizado para resolver sistemas densos não é eficiente, uma vez que é necessária uma grande quantidade de memória para alocação dos sistemas e, além disso, há um aumento no tempo de execução. Esse problema pode ser amenizado com a utilização de técnicas de armazenamento de matrizes esparsas. No *solver* BAND, pode ser visto a utilização de uma dessas técnicas, conhecida como *Compressed Diagonal Storage* (CDS).

Em [Lima 2008] foram realizados testes de desempenho da versão seqüencial do *solver*, com variação da dimensão da matriz de entrada e também com variação da largura de banda. A partir desses testes, pôde-se verificar o impacto do aumento da largura de banda em relação ao aumento da dimensão da matriz.

3. *Solver* BAND paralelo

A estrutura do programa original foi mantida, de forma a executar as rotinas seqüencialmente até a região do refinamento iterativo, apenas o cálculo do defeito da aproximação da solução é executado em paralelo através da criação de *threads* OpenMP.

Para a região do programa onde ocorre o refinamento iterativo foi determinada uma partição por domínio. A partir disto, o algoritmo implementado foi o de divisão e conquista.

Seguindo essa estratégia foi determinado que os dados necessários para a execução dessa região fossem distribuídos por meio de um particionamento bloco-linha, ou seja, cada processo recebe um bloco de linhas do conjunto de dados global para posteriormente realizar a sua parte da tarefa. Desta forma, o problema original foi divido em subproblemas menores de mesmo tamanho, onde cada subproblema representa um bloco-linha dos dados principais. Para isso, foram utilizados as funções de comunicação coletiva *MPI_Scatter* e *MPI_Gather* com suporte ao C-XSC.

3.1 Avaliação de Desempenho

O ambiente utilizado para a realização dos testes de desempenho da versão paralela do *solver* BAND corresponde a uma máquina paralela de arquitetura NOW (*Network of Workstations*), composta por 16 nodos homogêneos entre si, com a seguinte configuração: processador Intel Core 2 Duo E4500 (freqüência de 2.2 GHz , 2 MB de *cache* L2), com 2GB de memória principal, sistema operacional Ubuntu 7.04 com compilador GCC 4.2, biblioteca C-XSC versão 2.2.2 e LAM/MPI 7.1.2. Os nodos foram interligados por meio de uma rede *fast ethernet* (100 Mbps).

O padrão para os testes foi a definição de uma matriz de *Toeplitz* simétrica *A* com 5 (cinco) bandas contendo os valores 1 (um), 2 (dois), 4 (quatro), 2 (dois), 1 (um) e todos os valores de *b* iguais a 1 (um) [Hölbig 2005]. O tempo de execução do programa foi calculado para sistemas de ordem 50.000, 100.000, 200.000, 400.000, 800.000 e 1.600.000. Para cada ordem de matriz foram feitas 10 (dez) execuções e, além disso, foram alocados 2 (dois) processos por nodo, a fim de tirar um melhor proveito da

máquina paralela disponível. No gráfico da Figura 1, é apresentado o *speedup* obtido com a paralelização.

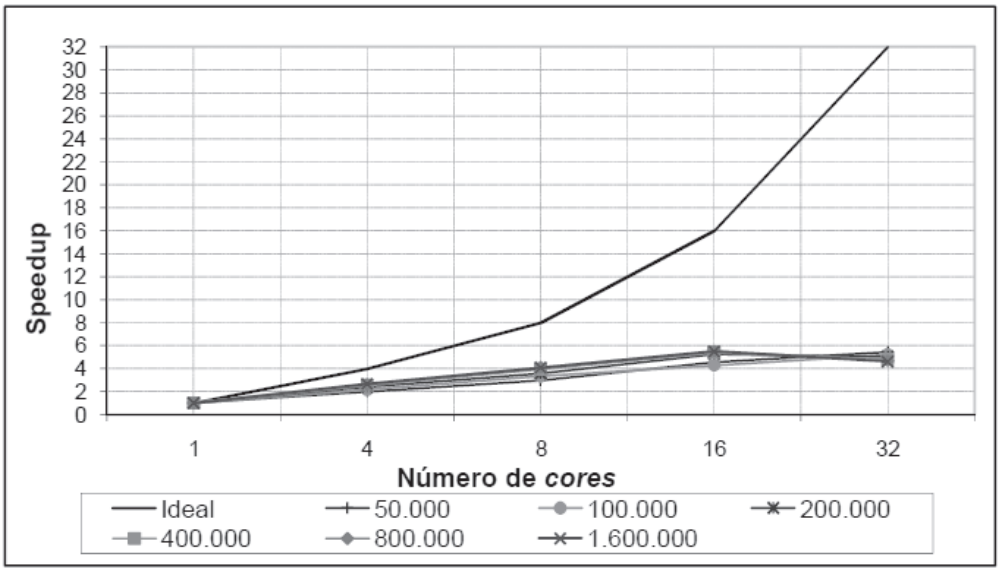

Figura 1. Gráfico de *speedup* da versão paralela do BAND

Através dos testes realizados, pode-se verificar que um ganho de desempenho foi obtido com a paralelização do *solver*. O melhor *speedup* foi alcançado com 16 *cores* nos testes realizados para matrizes de ordem igual a 1.600.000. Além disso, para matrizes de ordem 1.600.000, 800.000, 400.000 e 200.000 ocorreu uma queda na aceleração do *solver* paralelo, quando executado com 32 *cores*. Apenas, com as dimensões 50.000 e 100.000 ocorreu um crescimento constante do *speedup*.

A fim de verificar os resultados da paralelização realizada com *threads* OpenMP no cálculo do defeito da aproximação, foram realizados testes de desempenho de forma isolada, com a criação de 2 (duas) *threads* em um computador com a mesma configuração dos nodos do *cluster*. A Tabela 1 apresenta os tempos de processamento do método seqüencial e paralelo, e o *speedup* para cada dimensão de sistema.

Tabela 1. Valores de *speedup* da paralelização do cálculo do defeito

Dimensão	Seqüencial	Paralelo (duas *threads*)	Speedup
50.000	0,289611	0,164074	1,76512
100.000	0,586773	0,320783	1,82918
200.000	1,152632	0,632801	1,82147
400.000	2,236771	1,268517	1,76329
800.000	4,369245	2,483864	1,75905
1.600.000	8,918086	5,010675	1,77981

Através dos valores de tempo apresentados na Tabela 1, foi possível observar um ganho em desempenho de, aproximadamente, 44% com relação aos tempos seqüenciais do método do cálculo do defeito.

4. Considerações Finais e Trabalhos Futuros

A partir deste trabalho, verificou-se que o aumento no tempo de processamento do *solver* é mais acentuado à medida que o número de bandas da matriz aumenta, enquanto que à medida que a dimensão aumenta o tempo de processamento não sofre um aumento tão acentuado quanto o da variação do número de bandas.

Em relação à paralelização do *solver*, embora tenha sido obtido um ganho de desempenho, foi observado que o *speedup* e a eficiência ficaram distantes do que seria ideal. Nota-se também que a execução do *solver* paralelo mostrou-se mais eficiente com um número de *cores* igual a 4 (quatro), enquanto que o tempo de processamento diminuiu a medida que se aumentou o número de *cores*, porém não de forma proporcional. Um *speedup* mais próximo do ideal foi obtido na utilização de *threads* OpenMP na rotina que realiza o cálculo do defeito da aproximação.

Como trabalhos futuros, uma das possibilidades seria a adoção do algoritmo DotK [Ogita et al 2005 apud Krämer and Zimmer 2008], ao custo de uma possível perda de precisão nos resultados numéricos. Além disso, como somente parte do *solver* foi paralelizada, poderia ser realizada a paralelização da rotina que faz o cálculo da decomposição LU.

Uma alternativa de paralelização seria a utilização de bibliotecas externas no *solver* para a realização de alguns cálculos específicos. Neste caso, poderia ser utilizada a biblioteca numérica ScaLAPACK (*Scalable* LAPACK), para a realização tanto da decomposição LU quanto da fase onde são executadas a pós-substituição e retro-substituição intervalares.

Referências

Almeida, A. V. (2008). *Paralelização do Solver Intervalar LSS*. 2008. Trabalho de conclusão de curso (Graduação em Ciência da Computação) - Universidade de Passo Fundo, Passo Fundo.

Hölbig, C. A. (2005). *Ambiente de Alto Desempenho com Alta Exatidão para a Resolução de Problemas*. 2005. Tese (Instituto de Informática/Programa de Pós-Graduação em Computação) – Universidade Federal do Rio Grande do Sul, Porto Alegre.

Krämer, W. and Zimmer, M. (2008). *Fast (Parallel) Dense Linear System Solvers in C-XSC Using Error Free Transformations and BLAS*. In: Numerical Validation in Current Hardware Architectures, 08021, 2008, Dagstuhl, Anais eletrônicos… Dagstuhl: IBFI, 2008. http://drops.dagstuhl.de/opus/volltexte/2008/1436, Julho.

Kumar, V. et al (1994). *Introduction to Parallel Computing*: Design and Analysis of Algorithms. California: The Benjamin/Cummings Publishing Company.

Lima, A. A. M. (2008). *Paralelização de Solver Intervalar para a Resolução de Sistemas Lineares Esparsos*. 2008. Trabalho de conclusão de curso (Graduação em Ciência da Computação) - Universidade de Passo Fundo, Passo Fundo