ISSN 1984-820X
Anais do CNMAC v.3
PUBLICAÇÃO DA SBMAC
CNMAC2010
XXXIII CONGRESSO NACIONAL DE MATEMÁTICA APLICADA E COMPUTACIONAL

Solver Verificado com Alta Exatidão para a Resolução de Sistemas Lineares Densos[1]

Carlos Amaral Hölbig
Universidade de Passo Fundo - ICEG
99001-970, Passo Fundo, RS
E-mail: holbig@upf.br

***Resumo**: Este trabalho realiza um estudo de versões paralelas do solver verificado Linear System Solver (LSS) e, com base neste estudo, apresenta uma nova versão paralela do LSS baseada em otimizações realizadas na sua versão sequencial original. Estas otimizações na versão sequencial e o consequente desenvolvimento de uma nova versão paralela visaram uma melhora no desempenho computacional do solver LSS sem a perda da característica da alta exatidão. O solver LSS, nas suas mais diversas versões e/ou adaptações, é utilizado para a resolução de sistemas lineares densos do tipo Ax = b, tratando a instabilidade numérica da aritmética de ponto-flutuante de máquina por meio da biblioteca C-XSC com o uso da matemática intervalar.*

1. Introdução

A resolução de sistemas de equações lineares do tipo $Ax = b$, onde A representa uma matriz densa de coeficientes, é uma operação muito comum na área da Análise Numérica devido à sua alta aplicabilidade nas ciências e engenharias como, por exemplo, na análise tensorial de estruturas elétricas ou na análise de vibrações em sistemas mecânicos [4]. Entretanto, quando a tarefa de resolver um sistema linear é atribuída a um computador, a qualidade numérica dos resultados e o alto custo computacional tornam-se fatores críticos que, se não devidamente tratados, podem tornar o uso de uma solução computacional inviável, como pode ser observado em [10].

A instabilidade numérica é um problema inerente a todas as máquinas digitais, em parte devido à necessidade de representar o conjunto dos números reais em registradores com uma mantissa de precisão finita. Essa limitação no tamanho da mantissa pode gerar erros de arredondamento, além de tornar as operações aritméticas passíveis de erros de cancelamento. Dessa forma, tais problemas são capazes de conduzir uma dada aplicação a retornar resultados numéricos não confiáveis.

A Validação Numérica é um paradigma que pode ser utilizado para tratar a instabilidade numérica de máquina e garantir a qualidade dos resultados. Através desse paradigma de programação, o próprio computador torna-se capaz de decidir por executar um algoritmo alternativo, ou repetir o processo utilizando uma maior precisão, caso seja constatado que o cálculo realizado não resulte em valores suficientemente exatos para o problema que se está resolvendo [1].

O solver LSS (abreviação de *Linear System Solver*) é um programa para a resolução de sistemas de equações lineares com matrizes densas que, por meio da biblioteca de alta exatidão C-XSC, implementa o paradigma da Validação Numérica para que possa garantir a qualidade numérica de suas operações.

[1] Projeto de pesquisa financiado pelo CNPq (processo 475425/2009-0).

Alguns trabalhos já foram desenvolvidos objetivando a paralelização do solver LSS embora utilizando abordagens um pouco diferentes. Por exemplo, o solver paralelo proposto por Zimmer [9] não realiza o cálculo do produto escalar utilizando os registradores de alta precisão da biblioteca C-XSC, o que acarreta um ganho de desempenho mas apresenta uma pequena perda de exatidão. Já no trabalho de Kolberg [8], o solver LSS foi implementado em C++ sem o uso da biblioteca C-XSC não sendo assim um solver verificado (que possui todas as características da validação numérica). Já o solver desenvolvido neste trabalho foi baseado na versão paralela desenvolvida por Do Carmo [3]. A diferença entre as duas implementações é que para o solver apresentado neste trabalho foi realizada uma otimização da versão sequencial do LSS e, após esta otimização, uma nova abordagem na paralelização foi aplicada, conforme será detalhado nas próximas seções.

2. Overview do solver LSS

O solver LSS é um programa composto, basicamente, por dois algoritmos principais: uma etapa de precisão simples e uma etapa de precisão dupla. A etapa de precisão dupla é executada apenas quando o solver é incapaz de determinar uma aproximação para o sistema linear por meio do algoritmo de precisão simples, como acontece, por exemplo, em sistemas mal-condicionados.

Quando um sistema linear $Ax = b$ é submetido ao solver, o algoritmo de precisão simples calcula uma inversa A^{-1} da matriz densa de coeficientes A, por meio do método de Gauss-Jordan. Após o cálculo de A^{-1}, o solver determina um resultado aproximado x' através de $x' = A^{-1} \times b$. Em sequência, o resultado aproximado x' é submetido a uma série de subtarefas, que são responsáveis pelo refinamento do resultado x'. As subtarefas de refinamento são realizadas por meio de valores residuais de erro, determinados com o auxílio dos acumuladores de alta exatidão da biblioteca C-XSC. Caso a etapa de precisão simples seja incapaz de encontrar a solução do sistema linear, o solver executa a etapa de precisão dupla, onde é obtida uma matriz inversa de maior precisão. A partir disso, as subtarefas da etapa de precisão simples são executadas novamente, porém utilizando-se a nova matriz inversa obtida. O detalhamento do algoritmo do solver LSS, o *background* teórico de seu funcionamento, e o código-fonte em linguagem C++ podem ser encontrados em [6]. Alternativamente, pode-se encontrar uma descrição do algoritmo do solver em [2].

2.1. Tempos de processamento do LSS

As Tabelas 1 e 2 descrevem, respectivamente, as subtarefas das etapas de precisão simples e dupla, apresentando ainda o tempo de processamento de cada subtarefa do solver de forma isolada. O sistema linear utilizado para a obtenção dos tempos foi de ordem 2048. Os testes foram realizados em um computador com processador Intel Core 2 Duo E4500 à frequência de 2.2 GHz, front-side bus (FSB) de 800 MHz e com 2 GB de memória principal (como a memória trabalha a 667 MHz, a frequência do FSB limita-se à esse valor).

O sistema linear de entrada foi inicializado com uma matriz de Hilbert. Os valores dos elementos de uma matriz de Hilbert são determinados por $H_{i,j} = \frac{1}{i+j-1}$, onde i e j são índices da matriz. Dessa forma, pôde-se garantir que as etapas de precisão simples e precisão dupla do solver LSS fossem executadas, fazendo com que o teste atingisse, portanto, toda a extensão do algoritmo.

Seq.	Subtarefas do Algoritmo	Tempos(s)
1	Cálculo da inversa A^{-1} de A	2763.631
2	$x' = A^{-1} \times b$	0.142
3	$E = \Diamond(b - (A \times x'))$	1.533
4	$x'' = \Diamond(x' + A^{-1} \times E))$	1.424
5	$E = \Diamond(b - (A \times x''))$	1.533
6	$[F] = \Diamond((b - (A \times x'')) - E)$	1.536
7	$[G] = \Diamond(E \times A^{-1} + [F] \times A^{-1})$	5.408
8	$[C] = \Diamond(I - A^{-1} \times A)$	3901.698
	Total:	**6676.905**

Tabela 1: Tempos de processamento das subtarefas da etapa de precisão simples

Seq.	Subtarefas do Algoritmo	Tempos(s)
1	$I - (A^{-1} \times A)$	3462.095
2	Cálculo da inversa I^{-1} de I	2743.501
3	$Ai^{-1} = I^{-1} \times A^{-1}$	3460.048
4	$Ea = \Diamond(I^{-1} \times A^{-1}) - Ai^{-1}$	3420.006
5	$x'' = \Diamond(A^{-1} \times b + Ea \times b)$	0.162
6	$x' = \Diamond((A^{-1} \times b + Ea \times b) - x'')$	0.161
7	$E = \Diamond(b - (A \times x') - (A \times x''))$	3.045
8	$x'' = \Diamond(x' + (A^{-1} \times E) + (Ea \times E))$	2.788
9	$E = \Diamond(b - (A \times x''))$	1.536
10	$[F] = \Diamond((b - (A \times x'')) - E)$	1.539
11	$[G] = \Diamond(Ai^{-1} \times E + Ea \times E + Ai^{-1} \times [F] + Ea \times [F])$	7.834
12	$[C] = \Diamond(I - Ai^{-1} \times A - Ea \times A)$	7562.380
	Total:	**20665.095**

Tabela 2: Tempos de processamento das subtarefas da etapa de precisão dupla

3. Otimização do solver LSS

Antes de realizar o desenvolvimento da versão paralela do solver LSS, o programa sequencial foi otimizado por meio da identificação e eliminação de subtarefas que efetuavam computações redundantes nos algoritmos das etapas de precisão simples e dupla (Tabelas 1 e 2, respectivamente). Além da eliminação de recálculos presentes no algoritmo, o método de Gauss-Jordan foi otimizado para que se pudesse reduzir o tempo de processamento das subtarefas 1 e 2 das etapas de precisão simples e dupla, respectivamente.

Os algoritmos otimizados das etapas de precisão simples e dupla obtiveram tempos de execução de 4086.296s e 11581.886s, respectivamente. Para possibilitar tal comparação entre grandezas, tanto o tamanho do sistema linear de entrada (de ordem 2048) quanto os recursos computacionais utilizados foram mantidos os mesmos da sessão 2. Os tempos de processamento das subtarefas 1 e 7 da etapa de precisão simples, e das subtarefas 1 e 2 da etapa de precisão dupla foram obtidos com base na média de 2 execuções. Para as demais subtarefas, os tempos de execução foram calculados através da média de 10 execuções. O ganho em desempenho da versão otimizada do solver é representado através do histograma da Figura 1, que expressa a diferença percentual entre os tempos de processamento da versão original com relação à versão otimizada do LSS.

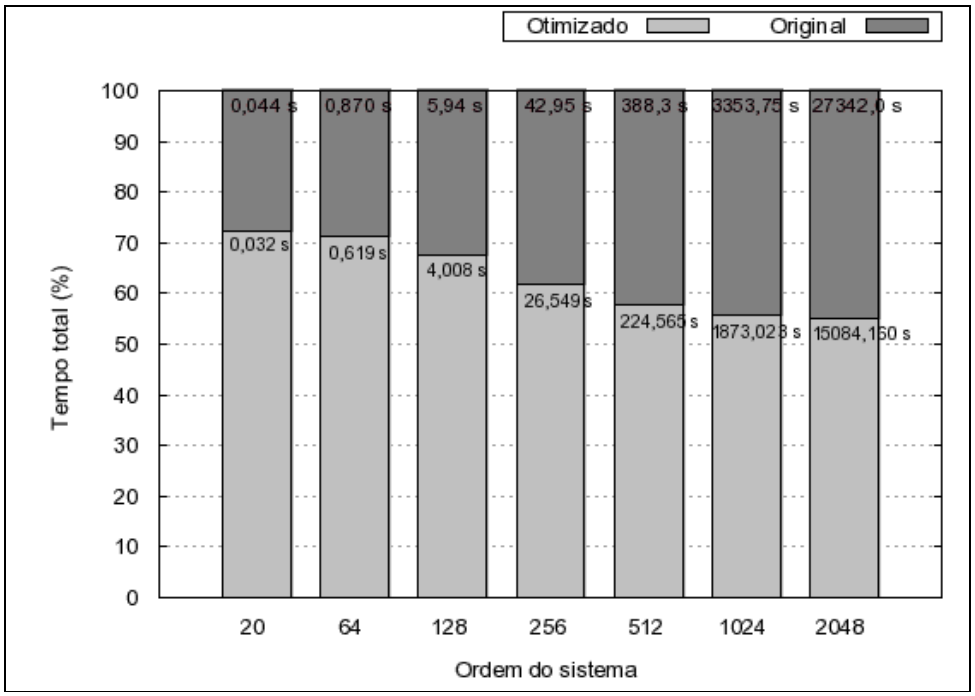

Figura 1: Ganho em desempenho do LSS otimizado com relação ao tempo do solver original

Através dos testes de desempenho apresentados, foi possível observar o considerável ganho em desempenho obtido no solver, decorrente tanto da eliminação de etapas redundantes do algoritmo, quanto da re-escrita da função que implementa o método de Gauss-Jordan, responsável pela subtarefa de inversão de matrizes. Entretanto, o tempo de execução ainda manteve-se relativamente alto para sistemas lineares de ordem mais elevada. Por exemplo, para um sistema linear de ordem 2048, a versão otimizada do LSS leva em torno de 4 horas para retornar uma solução na máquina descrita na sessão 2. O desenvolvimento de uma versão paralela do solver para execução em máquinas de memória distribuída é uma alternativa para reduzir o tempo de processamento de sistemas lineares que possuem uma ordem maior, sendo apresentada na sessão seguinte deste trabalho.

4. Solver LSS paralelo

Com base nos tempos de processamento da versão seqüencial otimizada do LSS, foi possível constatar que a degradação em desempenho dos algoritmos de precisão simples e dupla é causada por pontos bem definidos no código, sendo, portanto, os trechos do programa que devem ser decompostos para execução em paralelo. Esta sessão apresenta a identificação dos pontos críticos de tempo do algoritmo do LSS, que serão decompostos na versão paralela do

solver. Além disso, é descrito o esquema de particionamento a ser adotado, e a estruturação do programa paralelo, no que diz respeito ao fluxo de execução e alocação de tarefas aos processos da execução paralela.

4.1. Estruturação do programa paralelo

Como apenas determinadas partes do solver LSS foram paralelizadas, o solver pode ser visto como um sistema seqüencial constituído de módulos paralelos, pois a maior parte do código será executada no computador que invocará o programa. Seguindo essa idéia, foi adotado um modelo de algoritmo por divisão e conquista, onde a execução do programa principal é realizada por um processo gerenciador que, ao atingir as subtarefas paralelas, divide o domínio entre *p* processos trabalhadores (inclusive ele próprio), sendo estes responsáveis pelo cálculo e devolução dos resultados parciais ao processo gerenciador. Para a divisão do domínio entre os processos, foi adotado o esquema de particionamento por blocos de linhas (do inglês, *block-striped partitioning*). Nesse esquema de particionamento, as linhas das matrizes são alocadas de forma contígua nos processos que constituem a execução paralela, onde cada processo conterá um bloco com uma ou mais linhas completas da matriz [9, p. 152].

O desenvolvimento do solver paralelo foi feito utilizando a biblioteca MPI (distribuição LAM/MPI 7.1.2), sendo adotado um modelo de comunicação coletiva para efetuar a distribuição dos dados das matrizes entre os processos paralelos. Como as matrizes utilizadas no LSS são dos tipos rmatrix e imatrix, específicos da biblioteca C-XSC, foi necessário implementar as funções de comunicação coletiva MPI Bcast, MPI Scatter, MPI Gather e MPI Allgather para suportar esses tipos de dados. Uma vez que as funções de comunicação coletiva tenham sido implementadas, o particionamento por blocos de linhas pode ser feito com uma chamada para a MPI Scatter, informando-se a matriz que se deseja particionar. De forma semelhante, a junção dos blocos de linhas no processo gerenciador pode ser realizado através da função MPI Gather. É necessário ressaltar que o desenvolvimento dessas funções foi baseado na implementação de Grimmer [5], que apresenta funções da biblioteca MPI para comunicação ponto-a- ponto dos tipos de dados da biblioteca C-XSC. Na próxima sessão são apresentados os resultados de tempo obtidos da versão paralela com relação à versão sequencial otimizada (*speedup*).

4.2. Análise de desempenho

O solver LSS paralelo desenvolvido foi executado em um agregado de máquinas do tipo NOW (abreviação de *Network of Workstations*) no Laboratório Central de Informática da Universidade de Passo Fundo. O agregado foi constituído de 32 nodos interligados por uma rede *fast ethernet* (100 Mbps). Cada computador que constitui o agregado possui um processador *dual-core* Intel Core 2 Duo E4500 (64 KB de *cache* L1, e 2 MB de *cache* L2 compartilhada), trabalhando a frequência de 2,2 GHz e 800 MHz de *front-side bus*. Cada nodo possui ainda 2 GB de memória principal (dois módulos de 1 GB em *dual channel*) operando `a frequência de 667 MHz. A medição dos tempos de processamento foram feitos tanto para as subtarefas paralelas de forma isolada, quanto para o solver como um todo, com base na média aritmética de 10 amostras de tempo para sistemas de ordem 64, 128, 256, 512, 1024 e 2048. A execução de cada ordem de sistema foi dividida entre 2, 4, 8, 16, 32 e 64 processos MPI, alocando-se sempre dois processos por nodo do agregado, dessa forma tirando proveito dos processadores duais presentes em cada nodo. O sistema operacional utilizado nos testes foi o GNU/Linux Ubuntu 7.04 (kernel 2.6.20-smp) com compilador GCC 4.1. As versões das bibliotecas utilizadas na execução dos testes foram as seguintes: biblioteca C-XSC 2.2.2, biblioteca ATLAS 3.6.0, e biblioteca LAM/MPI 7.1.2.

Embora a aceleração das subtarefas paralelas tenha sido razoável, foi possível contatar pela Figura 2 que o *speedup* obtido da execução do solver LSS paralelo foi relativamente baixo, principalmente pelo fato de que a etapa de inversão de matrizes ainda é executada de forma sequencial, comprometendo assim a eficiência do solver.

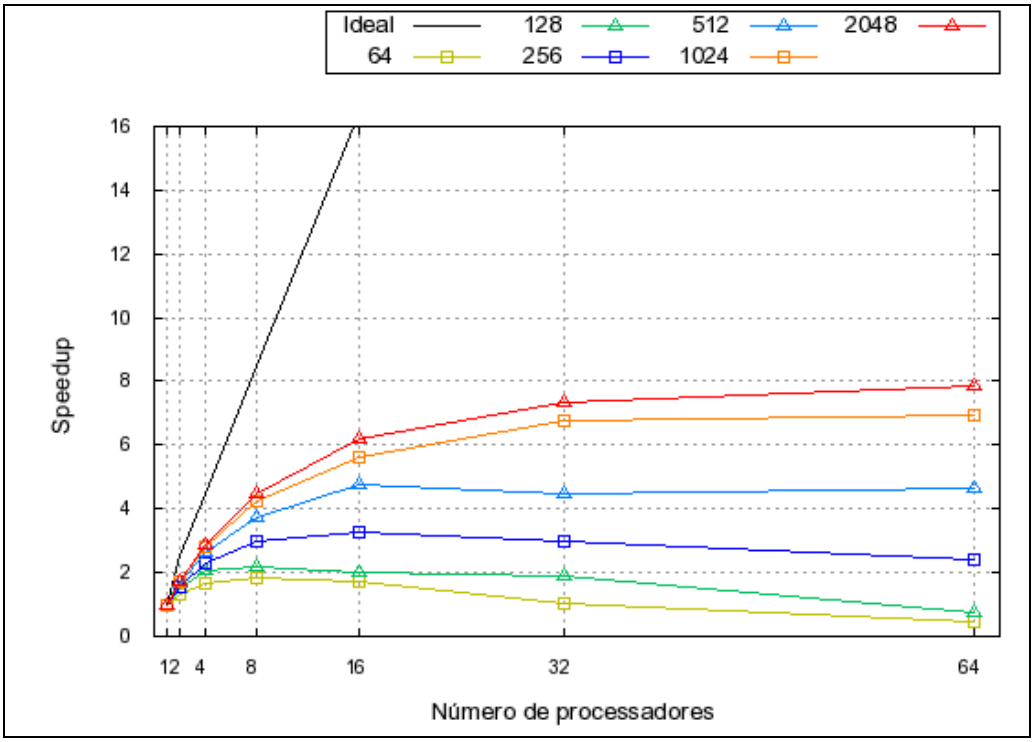

Figura 2: *Speedup* do solver paralelo LSS

A subtarefa 8 paralela demonstrou ser a mais eficiente dentre as subtarefas paralelizadas, onde o crescimento do *speedup* manteve-se próximo ao ideal (ou acima, em alguns casos) para todas as ordens de sistema. Essa maior eficiência de execução, com relação às outras subtarefas, pode ser explicada pelo fato da subtarefa 8 possuir duas operações de multiplicação de matrizes, o que reduz a relevância do custo da comunicação frente à necessidade por um maior uso do poder de processamento dos nodos. Além do uso mais intenso de CPU pela subtarefa 8, o número de operações de trocas de dados pela rede de comunicação é menor nessa subtarefa em comparação com as outras rotinas paralelas, o que dá um maior peso à sua característica de ser CPU *bound*, fato que contribui ainda mais para seu *speedup* e eficiência. Portanto, como as outras subtarefas paralelas possuem mais operações de I/O, é natural que sejam menos eficientes devido ao maior acesso à rede de comunicação para troca de dados.

5. Considerações finais

O estudo e detalhamento das subtarefas que compõe o algoritmo do solver LSS possibilitaram identificar computações redundantes no código que causavam um impacto significativo no desempenho do sistema. Através da eliminação de tais recálculos aliada com a nova função do método Gauss-Jordan, foi obtida uma versão sequencial otimizada do solver, que expressou um ganho em desempenho considerável em relação à versão original, como foi constatado pelo histograma da Figura 1. Embora a redução do tempo de resposta da versão sequencial otimizada tenha sido considerável, ele ainda mostrou-se insatisfatório para sistemas de ordem mais alta, fato que manteve o objetivo pela otimização do sistema através da

paralelização das subtarefas que mais consumiam tempo de processamento. A aceleração das subtarefas paralelas mostraram-se razoáveis para a maior parte dos casos, com destaque para a subtarefa 8, cujo *speedup* manteve-se bastante próximo do ideal. Entretanto, os testes de desempenho do solver LSS paralelo como um todo mostraram um crescimento mais tímido para o speedup. Esses resultados foram obtidos devido ao fato de que a subtarefa de inversão de matrizes pelo método de Gauss-Jordan foi mantida em sua forma sequencial, sendo que essa subtarefa representa um dos gargalos do sistema.

Em relação ao uso da biblioteca C-XSC em um ambiente paralelo, foi possível perceber que o uso de funções da biblioteca MPI para comunicação coletiva reduz consideravelmente o custo da comunicação de dados pela rede, se comparado com funções de comunicação ponto-a-ponto. Através do desenvolvimento dessas funções, estendeu-se ainda o conjunto de funções MPI para comunicação ponto-a-ponto dos tipos de dados do C-XSC, disponibilizadas por [5]. Como as funções de comunicação coletiva implementadas suportam apenas os tipos de dados da biblioteca CXSC que foram necessários ao desenvolvimento do solver paralelo, pode-se considerar ainda a implementação de funções MPI que suportem os outros tipos de dados da biblioteca.

Referências

[1] G. Alefeld e J. Herzberger, “An Introduction to Interval Computations”, Academic Press, New York, 1983.

[2] A. Almeida, “Paralelização do Solver Intervalar LSS”, Trabalho de Conclusão de Curso, Curso de Ciência da Computação-Universidade de Passo Fundo, 2009.

[3] A. B. do Carmo, C. A. Hölbig e M. T. Rebonatto, Paralelização do solver intervalar linear system solver, em “I Concurso de Trabalhos de Iniciação Científica em Arquitetura de Computadores e Computação de Alto Desempenho, WSCAD-CTIC 2007, SBAC-PAD 2007”, Gramado, Brasil, 2007.

[4] D. M. Cláudio e J. M. Marins, “Cálculo Numérico Computacional: Teoria e Prática”. Editora Atlas, São Paulo, 2000.

[5] M. Grimmer, “An MPI extension for the use of C-XSC in parallel environments”, Relatório Técnico, UniversitätWuppertal, 2005.

[6] C. Hölbig e W. Krämer, “Selfverifying solvers for dense systems of linear equations realized in C-XSC”, Relatório Técnico, Bergische Universität Wuppertal, 2003.

[7] C. A. Hölbig, “Ambiente de Alto Desempenho com Alta Exatidão para a Resolução de Problemas”, Tese de Doutorado, PPGC-Universidade Federal do Rio Grande do Sul, 2005.

[8] Kolberg, M. L., Fernandes, L. G. e Claudio, D. M., Dense Linear System: A Parallel Self-verified Solver, International Journal of Parallel Programming, vol. 36, pp. 412–425, (2008).

[9] V. Kumar et al., “Introduction to Parallel Computing: Design and Analysis of Parallel Algorithms”, Benjamin-Cummings Pub Co, 1994.

[10] M. Zimmer e W. Krämer, Fast (parallel) dense linear interval systems solvers in C-XSC using error free transformations and blas, em “A. Cuyt, W. Krämer, W. Luther e P. Markstein (editors) - Numerical Validation in Current Hardware Architectures, number 08021 in Dagstuhl Seminar Proceedings”, Dagstuhl, Alemanha, 2008.