Implementação Paralela do Método dos Gradientes Conjugados para Solução de Sistemas Esparsos de Equações Lineares

Ademar Muraro Jr., Airam J. Preto, Stephan Stephany
Laboratório Associado de Computação e Matemática Aplicada - LAC/INPE
ademarm@ieav.cta.br, airam@lac.inpe.br, stephan@lac.inpe.br

Angelo Passaro, Onofre F. Lima, Nancy M. Abe, Roberto Y. Tanaka
Laboratório de Engenharia Virtual, Instituto de Estudos Avançados - IEAv/CTA
angelo@ieav.cta.br, onofre@ieav.cta.br, nancy@ieav.cta.br, roberto@ieav.cta.br

Resumo

O artigo apresenta uma comparação de diferentes esquemas de comunicação aplicados na paralelização do Método dos Gradientes Conjugados (CGM), um método numérico iterativo para a resolução de sistemas de equações lineares (A.x = b). As implementações paralelas usam o paradigma de troca de mensagens (MPI) e são baseadas em uma versão seqüencial implementada em C++ e orientada a objetos, desenvolvida no Laboratório de Engenharia Virtual (LEV/IEAv/CTA).

Palavras-chave: *MPI, matriz esparsa, métodos numéricos iterativos, gradientes conjugados, computação de alto desempenho.*

1. Introdução

Uma das funções da análise numérica é a resolução de sistemas lineares. Muitos problemas em Ciências e Engenharia, tais como, a resolução de equações diferenciais associadas ao estudo de dinâmica dos fluidos, simulações de plasma, engenharia elétrica, civil, mecânica e outros, recaem em grandes sistemas lineares. Esses sistemas lineares são gerados quando equações diferenciais parciais são resolvidas usando aproximações por elementos finitos ou diferenças finitas, em problemas lineares e não lineares. Freqüentemente, a resolução desses sistemas é a que apresenta o maior consumo de tempo computacional.

A solução de sistemas lineares pode ser obtida pela aplicação de métodos diretos, tais como, a decomposição LU, a qual obtém a solução por meio da fatorização dos coeficientes da matriz, ou a partir de métodos iterativos, os quais obtêm a solução em aproximações sucessivas.

Os métodos da família do subespaço de Krylov são métodos iterativos poderosos para resolver grandes sistemas lineares esparsos, os quais envolvem os coeficientes da matriz somente na forma de produtos matriz-vetor. Esses métodos consistem em gerar bases adequadas em cada iteração para um espaço vetorial, denominado espaço de Krylov. Os métodos mais populares são o dos Gradientes Conjugados (CGM), o dos Gradientes Biconjugados (BiCGM) e suas modificações como BiCGSTAB e LSQR.

Os métodos numéricos, diretos (Murthy, 2001) e iterativos, têm sido implementados em versões paralelas para serem executados, principalmente, em computadores massivamente paralelos de memória distribuída (Yang, 2003, Yang, 2002 e Zhang, 2002). Porém, o uso crescente de *clusters* (conjuntos de computadores interconectados por meio de uma rede padrão Fast Ethernet ou, mais recentemente, Gigabit Ethernet) para processamento paralelo, impõe a necessidade de se avaliar o custo da comunicação envolvido, o qual representa o gargalo no desempenho desse tipo de arquitetura. Alguns trabalhos recentes abordando o paralelismo em *clusters* têm sido apresentados na literatura (Kumar, 2004 e Newman, 1999).

Nesse trabalho são analisados o tempo consumido na comunicação global envolvida em uma implementação paralela de resolução de equações lineares, derivada de uma versão seqüencial que foi desenvolvida no Laboratório de Engenharia Virtual (LEV/IEAv/CTA) (Almeida, 2004).

As implementações avaliadas utilizam o protocolo de comunicação MPI (*Message Passing Interface*) (Gropp, 1999). O trabalho apresenta versões com rotinas de comunicação ponto a ponto (MPI_Send e MPI_Recv) e para comunicação coletiva, tais como *broadcast*, *allgather* e *reduction*.

Esse artigo está organizado da seguinte maneira: a seção 2 faz uma descrição da implementação paralela do CGM, com detalhes para a distribuição de dados e esquema de comunicação; a seção 3 apresenta as implementações; a seção 4 faz uma comparação dos resultados e a seção 5 apresenta as conclusões.

2. Método dos Gradientes Conjugados (CGM)

O Método dos Gradientes Conjugados (CGM) é um algoritmo para encontrar o mínimo local de uma função de *n* variáveis, supondo que o gradiente da função possa ser calculado. Ele usa direções conjugadas (ortogonais) ao invés do gradiente local para buscar o mínimo. Se a vizinhança de um mínimo tem o aspecto de um vale suave, o mínimo é alcançado em poucos passos, de maneira mais rápida do que usando o Método da Máxima Descida (*Steepest descent*).

O CGM é um método efetivo para a resolução de sistemas matriciais com matrizes simétricas definidas positivas. Ele é o mais antigo e o mais conhecido método não estacionário. O método caminha gerando vetores a cada iteração. Os resíduos $r^{(i)}$ correspondendo às iterações e as direções de busca $p^{(i)}$ são utilizados para atualizar a iteração seguinte. A cada iteração do método, produtos internos são calculados a fim de atualizar os escalares que são definidos para satisfazer certas condições de ortogonalidade. Isso pode ser interpretado como a busca pela mínima energia *E* de um sistema linear $A.x = b$. A energia do sistema é mínima quando o resíduo $r = b - A.x$ se anula. Os vários passos do algoritmo são descritos na Figura 1. No algoritmo, itc_{max} corresponde ao número máximo de iterações, ϵ a tolerância e α é o valor que minimiza o resíduo

$$
\begin{aligned}
&Set\ \ r^{(0)}=b,\ \ \rho^{(0)}=(r^{(0)},r^{(0)}),\ \ itc=0\\
&\quad Do\quad itc=itc+1\\
&\quad\ \rho^{(i-1)}=(r^{(i-1)},r^{(i-1)})\\
&\quad\ if\ \ it=1\\
&\quad\quad p^{(i)}=r^{(0)}\\
&\quad\ else\\
&\quad\quad \beta^{(i-1)}=\rho^{(i-1)}/\rho^{(i-2)}\\
&\quad\quad p^{(i)}=r^{(i-1)}+\beta^{(i-1)}\,p^{(i-1)}\\
&\quad\ endif\\
&\quad\ q^{(i)}=Ap^{(i)}\\
&\quad\ \alpha^{(i)}=\rho^{(i-1)}/(p^{(i)},q^{(i)})\\
&\quad\ x^{(i)}=x^{(i-1)}+\alpha^{(i)}p^{(i)}\\
&\quad\ r^{(i)}=r^{(i-1)}-\alpha^{(i)}q^{(i)}\\
&\quad while\ \ \rho^{(i-1)}>\epsilon\ \ or\ \ itc<itcmax
\end{aligned}
$$

Figura 1 - **Algoritmo CGM seqüencial**

2.1. Estratégias de paralelização para o CGM

As estratégias discutidas nessa seção consideram uma arquitetura de múltiplos computadores com memória distribuída e interconectados por meio de uma rede fast ethernet.

As etapas do algoritmo que podem ser beneficiadas com o mecanismo de paralelização são (Kumar, 2004 e Golub, 1993):
(a) produto interno de dois vetores e
(b) multiplicação matriz-vetor.
A seguir, será discutido o processo de paralelização desses componentes básicos.

Divisão por blocos: supondo que se tenha uma matriz A_{nxn}, um vetor b_{nx1} e *k* processadores disponíveis, a matriz A_{nxn} pode ser dividida em blocos de linhas com segmentos de tamanho *n/k* (o tamanho do segmento para o último processador será $n-(k-1)n/k$) e distribuída aos *k* processadores (Figura 2). O vetor *b* pode ser dividido e distribuído de maneira similar.

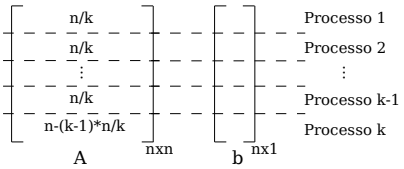

Figura 2 - **Particionamento de uma matriz (por linhas)**

Produto interno: supondo dois vetores *u* e *v* com dimensões *nx1*em um sistema com *k* processadores. Cada um dos *k* processadores possui vetores locais que são *local_u* e *local_v* de tamanho *n/k*. O produto interno desses segmentos residentes nos vários processadores é determinado concorrentemente, executando o código seqüencial nos respectivos processadores (Figura 3).

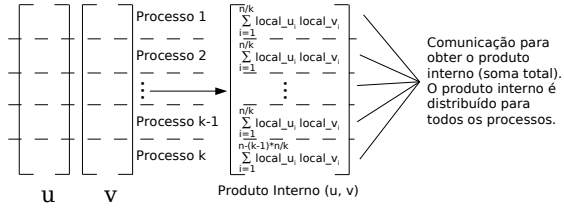

Figura 3 - **Produto interno de dois vetores (execução paralela)**

Produto matriz-vetor: para executar o produto de uma matriz *A* (dividida por bloco de linhas entre *k* processadores) por um vetor *p*, cada processador deve enviar sua parte do vetor p_{nx1} para todos os outros. Após esse procedimento, cada processador executa a operação de produto com o seu respectivo bloco da matriz pelo vetor *p* completo (Figura 4).

Sobreposição de comunicação com processamento: no cálculo do produto interno de dois vetores, cada processador executa os cálculos com os seus vetores locais. Para máquinas de memória distribuída, cada vetor local deve ser enviado para os outros processadores a fim se obter o produto interno global. Isso pode ser feito com o envio de todos para todos (*all-to-all send*) e cada processador efetua a soma dos vetores locais ou, por

meio de uma centralização global em um só processador, seguido por um *broadcast* do resultado final. Claramente, essa etapa requer comunicação. Na formulação usual do CGM, os produtos internos induzem a uma sincronização dos processadores, pois não é possível prosseguir com os cálculos até que o resultado final seja alcançado: a atualização de $x^{(i+1)}$ e $r^{(i+1)}$ somente começa após a conclusão do produto interno para $\alpha^{(i)}$.

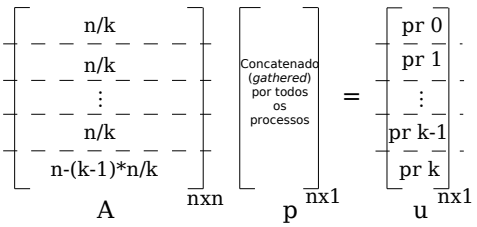

Figura 4 - **Execução paralela de um produto de uma matriz *A* por um vetor *p***

Desde que a comunicação em máquinas de memória distribuída é necessária para o produto interno, não é possível sobrepor essa comunicação com o cálculo. Essa observação também se aplica na atualização de $p^{(i)}$, a qual somente começa após a conclusão do produto interno para $\beta^{(i-1)}$.

$$
\begin{aligned}
&Set\ r^{(0)}=b,\ \rho^{(0)}=(r^{(0)},r^{(0)}),\ itc=0\\
&\ Do\\
&\quad itc=itc+1\\
&\quad p^{(i)}=r^{(i)}+\beta^{(i-1)}\,p^{(i-1)}\\
&\quad q^{(i)}=Ap^{(i)}\\
&\quad \gamma=(p^{(i)},q^{(i)})\\
&\quad x^{(i)}=x^{(i-1)}+\alpha^{(i-1)}\,p^{(i-1)}\\
&\quad \alpha^{i}=\rho^{(i)}/\gamma\\
&\quad r^{(i+1)}=r^{(i)}-\alpha^{(i)}q^{(i)}\\
&\quad \rho^{(i+1)}=(r^{(i+1)},r^{(i+1)})\\
&\quad if\quad \rho^{(i+1)}<\epsilon\quad then\\
&\qquad x^{(i+1)}=x^{(i)}+\alpha^{(i)}\,p^{(i)}\\
&\qquad quit\\
&\quad endif\\
&\ while\,(itc<itcmax)
\end{aligned}
$$

Figura 5 - **CGM paralelo com sobreposição**

A Figura 5 apresenta uma variação do CGM, na qual todas as comunicações poder ser sobrepostas aos cálculos. Essa versão envolve uma reorganização das instruções do CGM, que apresenta a mesma estabilidade da versão original (Barrett, 1994). Outra vantagem é que não há necessidade de operações adicionais. Esse rearranjo é baseado em um artifício. O artifício é a atualização da iteração retardada para encobrir o estágio de comunicação do produto interno $p^{(i)T}Ap^{(i)p}$. Dado A, b, itc_{max} (número máximo de iterações) e ε (tolerância).

Com essas considerações, o CGM pode ser eficientemente paralelizado:
(1) A comunicação requerida para a redução do produto interno para γ pode ser sobreposta com a atualização de $x^{(i)}$, a qual poderia ter sido feita na iteração anterior;
(2) A redução do produto interno para $\rho^{(i+1)}$ pode ser sobreposta com a parte remanescente da operação de pré-condicionamento no começo da iteração seguinte;
(3) O cálculo do segmento de $p^{(i)}$ pode ser seguido imediatamente pelo cálculo do segmento de $q^{(i)}$ e seguido pelo cálculo da parte do produto interno.

3. Estratégias de comunicação

A comunicação coletiva é um recurso do MPI usado com freqüência. Implementações rudimentares das funções que implementam esse tipo de comunicação podem ter um impacto negativo no desempenho das aplicações que as utilizam. Esforços têm sido feitos com o objetivo de melhorar o desempenho dessas rotinas (Benson, 2003 e Thakur, 2004).

Nessa seção comparamos o efeito de alguns mecanismos de comunicação no desempenho de um código computacional paralelo para o CGM. Para tanto, um código implementado anteriormente em C++, com orientação a objetos, foi adaptado para execução paralela, utilizando o MPICH 1.2.5 (Argonne, 2003). a paralelização foi realizada com base no algoritmo apresentado na Figura 1. Os sistemas matriciais resolvidos envolvem matrizes esparsas, as quais são armazenadas na forma compacta, sem sobreposição entre comunicação e cálculo, por linha ou por coluna. Este armazenamento é realizado em classes de matrizes esparsas que permitem a conversão do armazenamento por coluna em armazenamento por linha, e vice-versa. As classes também permitem a realização de particionamentos, fornecendo partições da matriz original, conforme solicitação de objetos clientes. As classes de resolução foram construídas para utilizar operações definidas nas classes de matrizes esparsas.

Todos os casos foram executados em um *cluster* formado por 12 máquinas com processadores AMD Athlon XP 2500+ e interconectados por meio de rede padrão Fast Ethernet (100 Mbps), equipada com uma *switch*.

Na primeira implementação foram utilizadas as funções para comunicação ponto a ponto, MPI_Send e MPI_Recv, onde todos os processadores enviam seus dados, provenientes de cálculos locais para uma máquina central que realiza os cálculos globais e redistribui os novos dados por meio de uma operação de *broadcast* (MPI_Bcast).

Outra implementação utilizou a função MPI_Allgather, que é uma operação de concatenação, na

qual os dados gerados por cada processadores são distribuídos para todos os outros processadores, de maneira que todos os processadores passam a possuir os dados globais. Versões do MPICH anteriores a 1.2.5 utilizavam o esquema em anel, em que os dados de cada processador são enviados para os outros, percorrendo um anel virtual de processadores. No primeiro passo, cada processador *i* envia sua contribuição para o processador *i+1* e recebe a contribuição do processador *i-1* . No segundo passo, cada processador *i* repassa para o processador *i+1* os dados recebidos do processador *i-1* do passo anterior. Se *p* é o número de processadores, então o algoritmo inteiro perfaz *p-1* passos. A versão 1.2.5 introduziu modificações na função Allgather, com o algoritmo *recursive doubling*. No primeiro passo desse algoritmo, os processadores que estão a distância 1 trocam seus dados. No segundo passo, os processadores que estão a distância 2 entre si trocam seus próprios dados, bem como os dados recebidos no passo anterior. No terceiro passo, os processadores que estão a distância 4 trocam seus dados juntamente com os recebidos anteriormente. Desse modo, para um número de processadores de potência de 2, todos os processadores obtêm todos os dados em *lg p* passos.

Uma terceira implementação também utilizou as funções MPI_Send e MPI_Recv, mas em um esquema de distribuição em anel, dispensando o envio dos dados para uma máquina centralizadora. Essa implementação só foi executada com a matriz de 22226 equações, pois com as matrizes de 71644 e 182948 equações o programa congelou, não fornecendo resultados ou mensagens de erro. O *deadlock* possivelmente está associado a problemas de sincronismo de mensagens do *send* e *receive* ou com o *buffer* de armazenamento das mensagens. O esquema em anel foi rearranjado, de maneira solucionar o problema, com um alternação do envio das mensagens entre os processadores com identificadores (*rank*) pares e ímpares, ou seja, em uma etapa todos os processadores pares enviam suas mensagens (*send*) e os processadores ímpares recebem essas mensagens (*receive*) e na etapa seguinte ocorre o inverso, os processadores ímpares enviam as mensagens e os pares a recebem. Isso é feito até que todos os dados seja recebidos por todos os processadores. Com essa mudança no esquema em anel, foi possível obter os dados para as matrizes de 71644 e 182948 equações e são os resultados dessa implementação modificada que estão apresentados na seção 4.

4. Resultados

As tabelas mostram os resultados obtidos para as várias implementações e com duas dimensões de matrizes: 71644 equações e 182948 equações. Os tempos exibidos são as somas dos tempos de comunicação e de cálculo acumulados nas iterações. Todos os casos executados tiveram a tolerância ϵ fixada em 1.10^{-12} e quantidade máxima de iteração *itcmax* de 5000.

As Tabelas 1 e 2 mostram os tempos e desempenhos (*speed up*) onde a coluna Send/Recv/Bcast significa que todos os processadores enviam seus dados locais para uma máquina centralizadora (denominada **master**), que realiza os cálculos necessários e reenvia os novos dados, por meio de uma operação de *broadcast*. A coluna Send/Recv/Anel utiliza o esquema de anel no qual os dados não são centralizados, mas as parcelas calculadas em cada processador são trocadas com os outros, até que todos obtenham os dados completos.

A Tabelas 3 mostra os tempos de execução para as matrizes de 71644 e 182948 equações, utilizando a função Allgather e sem a participação da máquina **master** no processo de comunicação. Os resultados obtidos com o **master** participando do processo foi um desempenho pior para todos os casos executados.

Em todos os casos, a execução para um só processador (seqüencial) foi mais rápida do que a execução paralela devido a quantidade de comunicação realizada entre os processadores e ao método utilizado. Em nenhuma das implementações paralelas conseguiu-se um ganho de desempenho, com relação à execução seqüencial, mas observou-se uma ligeira vantagem para o esquema em anel, principalmente quando se aumenta o número de processadores. Esse comportamento é melhor visualizado no formato gráfico, conforme mostra a Figura 6, para um sistema de 182948 equações.

A título de comparação, a resolução das matrizes de 71644 e 182948 também foi executada utilizando a biblioteca matemática SUPERLU *(LU decomposition*), que é uma biblioteca de resolução de sistemas de equações lineares esparsas não simétricas para uso em máquinas de arquitetura paralela, e utiliza o método direto para obter a solução (Tabela 4). Apesar de não ter sido obtido ganho, de maneira geral pode-se observar que os *speed up* da versão paralela do CGM são superiores aos da SUPERLU. O gráfico da Figura 7 compara o desempenho do esquema em anel (melhor caso do CGM) com a SUPERLU.

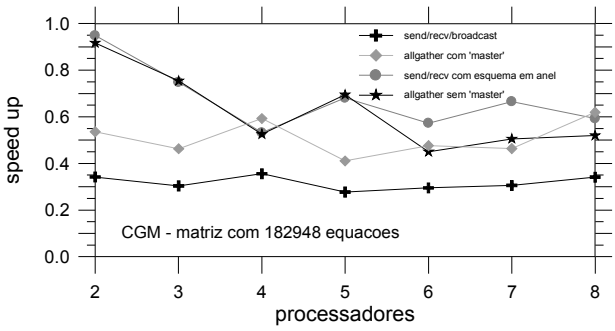

Figura 6 - **Comparação de desempenho entre várias implementações paralelas do CGM, para um sistema de 182948 equações**

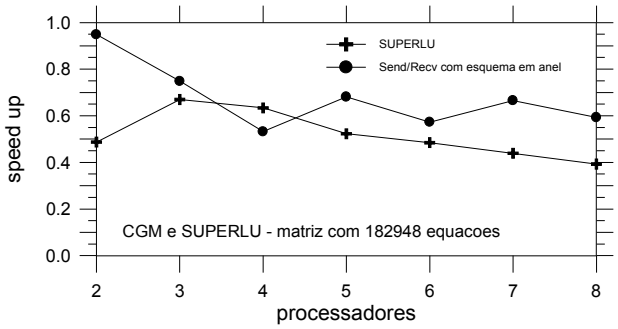

Figura 7 - **Comparação de desempenho entre a SUPERLU e o esquema de anel do CGM**

Tabela 1 - **Tempos de comunicação para matriz de 71644 equações, utilizando os esquemas *Send/Recv/Bcast* e *Send/Recv/Anel***

	Send/Rec/Bcast		Send/Rec/Anel	
process.	tempo (s)	speed up	tempo (s)	speed up
1	76,84	1	75,96	1
2	238,93	0,32	80,86	0,94
3	277,45	0,28	104,89	0,72
4	236,17	0,33	148,83	0,51
5	299,24	0,26	118,02	0,64
6	281,62	0,27	202,10	0,38
7	274,18	0,28	120,61	0,63
8	241,41	0,32	270,55	0,28

Tabela 2 - **Tempos de comunicação para matriz de 182948 equações, utilizando os esquemas *Send/Recv/Bcast* e *Send/Recv/Anel***

	Send/Rec/Bcast		Send/Rec/Anel	
process.	tempo (s)	speed up	tempo (s)	speed up
1	419,67	1	403,17	1
2	1228,78	0,34	425,17	0,95
3	1382,67	0,30	538,98	0,75
4	1180,76	0,35	757,48	0,53
5	1515,40	0,28	592,01	0,68
6	1423,16	0,29	703,75	0,57
7	1374,06	0,31	606,39	0,65
8	1231,54	0,34	680,35	0,59

Tabela 3 - **Tempos de comunicação para as matrizes de 71644 e 182948 equações, utilizando a função de comunicação coletiva *allgather***

	matriz 71644 Allgather		matriz 182948 Allgather	
process.	tempo (s)	speed up	tempo (s)	speed up
1	76,33	1	406,06	1
2	83,76	0,91	443,17	0,92
3	103,45	0,74	538,64	0,75
4	151,36	0,50	773,14	0,52
5	113,75	0,67	584,78	0,69
6	178,26	0,42	901,97	0,45
7	158,79	0,48	803,46	0,51
8	152,01	0,51	782,03	0,52

Tabela 4 - **Tempos de comunicação com a biblioteca SUPERLU**

	matriz 71644		matriz 182948	
process.	tempo (s)	speed up	tempo (s)	speed up
1	7,35	1	33,61	1
2	14,65	0,50	68,98	0,49
3	15,61	0,47	50,19	0,67
4	17,52	0,42	53,02	0,63
5	20,57	0,36	64,28	0,52
6	23,32	0,32	69,36	0,48
7	24,89	0,29	76,56	0,44
8	29,64	0,25	85,65	0,39

5. Conclusão

Os resultados obtidos mostraram que o esquema de comunicação empregado tem impacto direto no desempenho do programa paralelo implementado assim, esses esquemas de comunicação esta sendo explorados de forma a se obter melhor desempenho.

Algumas estratégias poderiam até fazer uso de diferentes métodos de comunicação em um mesmo programa, selecionando aqueles mais adequados para um específico tamanho de mensagem e número de processadores.

O armazenamento compacto das matrizes esparsas reduz o tamanho dos blocos que devem ser trocados entre os processos. Por outro lado, também reduz

bastante as operações matemáticas realizadas em cada processo, ocasionando uma granularidade muito fina em termos de processamento paralelo. Na implementação corrente do CGM não se obteve ganho de desempenho com a paralelização.

Em continuação a esse trabalho, pretende-se obter um desempenho paralelo satisfatório otimizando a comunicação e utilizando outras estratégias tais como sobreposição de comunicação com processamento.

6. Referências

Almeida, L. A., Passaro, A., Abe, N. M., 2004. Iterative Numerical Methods for Solving Large Sparse Linear Systems of Equations. *XXVII Congresso Nacional de Matemática Aplicada e Computacional (XXVII CNMAC)*. Porto Alegre, RS.

Argonne National Laboratory, 2003. *MPICH - A Portable Implementation of MPI*.
http://www-unix.mcs.anl.gov/mpi/mpich

Barrett, R., Berry, M. et al., 1994. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. Esse livro está disponível, em formato postscript, no sitio: ftp.netlib.org/templates/templates.ps.

Benson, G. D, Chu, Cho-Wai et al., 2003. A Comparison of MPICH Allgather Algorithms on Switched Networks. *10th Euro PVM/MPI 2003 Conference*. p. 335-345.

Golub, G. H., Van Loan, C. F., 1993. *Matrix Computations*. 2nd edition. The Johns Hopkins University Press.

Gropp, W., Lusk, E., Skejellum, A., 1999. *Using MPI: Portable Parallel Programming with the Message Passing Interface*. MIT Press.

Kumar, B. V. R., Kumar, B., 2004. On the development of parallel sparse CGM solver for CFD computations on Anu-cluster. *Applied Mathematics and Computation*, **article in press**.

Murthy, C. S. R., Murthy, K. N. B., Aluru, S., 2001. *New Parallel Algorithms for Direct Solutions of Linear Equations*. John Wiley & Sons, Inc.

Newman, M., 1999. Parallelization of a Preconditioned Conjugate Gradient Method Algorithm using the Message-Passing Paradigm.
http://quattro.me.uiuc.edu/~newman/parallel/ppcgm.pdf,
último acesso 28/09/2004.

Thakur, R., Rabenseifner, R., Gropp W., 2004. Optimization of Collective Communication Operations in MPICH. *Submetido ao International Journal of High Performance Computing Applications*.

Yang, L. T., Brent, R. P., 2003. The Improved Krylov Subspace Methods for Large and Sparse Linear Systems on Bulk Synchronous Parallel Architectures. *International Parallel and Distributed Processing Symposium (IPDPS'03)*.

Yang, L. T., Brent, R. P., 2002. The Improved BiCGStab Method for Large and Sparse Unsymmetric Linear Systems on Parallel Distributed Memory Architectures. *Fifth International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP'02)*. p. 324-328.

Zhang, J., Maple, C., 2002. Parallel Solutions of Large Dense Linear Systems Using MPI. *International Conference on Parallel Computing in Electrical Engineering (PARELEC'02)*. p. 312-317.