RESOLUÇÃO DE SISTEMAS DE EQUAÇÕES LINERARES UTILIZANDO UMA BIBLIOTECA DE OPERAÇÕES VETORIAIS/MATRICIAIS PARALELAS*

M.C.S. de Castro [1] C.L. de Amorim [2]

RESUMO

Neste trabalho procuramos a partir dos passos elementares disponíveis na biblioteca de operações paralelas, desenvolvida para multiprocessadores hipercúbicos baseado em *transputers*, solucionar um sistema de equações lineares algébricas, utilizando o método do gradiente conjugado. Os resultados experimentais demonstram o potencial e a simplicidade de aplicação da biblioteca para diversas aplicações numéricas.

ABSTRACT

In this work we use a library of parallel operators whith has been developed for hypercube multiprocessors based on transputers, to solve a system of linear equations, using the conjugate gradient method. The experimental results reveal the simplicity and the potential of the library for several numerical applications.

[1]Aluna de Mestrado da COPPE/Sistemas (UFRJ); áreas de interesse: Processamento Paralelo (arquiteturas e avaliação de desempenho);

[2]MSc (COPPE - 1979), PhD (Imperial College - 1984); áreas de interesse: Processamento Paralelo (arquiteturas e avaliação de desempenho); Professor Adjunto da COPPE/Sistemas;

COPPE/Universidade Federal do Rio de Janeiro
Caixa Postal 68511 - CEP 21945 - Rio de Janeiro - RJ
E - MAIL : COS99284@UFRJ (BITNET)

* Esse trabalho foi parcialmente financiado pela Finep, CNPq, Capes e Sociedade Cultural e Beneficente Guilherme Guinle.

1 Introdução

A solução de sistemas de equações lineares é fundamental para inúmeras aplicações numéricas, encontradas em problemas das Engenharias, Física, Meteorologia, entre outras.

A biblioteca de operações paralelas desenvolvida para multiprocessadores hipercúbicos baseado em *transputers* [2], contém algoritmos que são passos elementares, tais como alocação de vetores e matrizes, operações básicas da álgebra linear incluindo produto interno, produtos de uma matriz por um vetor e de matrizes e operações de difusão e convergência de escalares, vetores e matrizes.

A biblioteca torna transparente ao usuário a dependência da arquitetura da máquina e a comunicação entre os processadores realizada pelas rotinas. Bastando somente saber o que realiza cada uma delas individualmente, chamá-las com seus argumentos convenientemente e agrupá-las de modo a obter um algoritmo maior que solucione seu problema real, tendo a facilidade da modularidade para a depuração.

Apresentamos inicialmente o método do gradiente conjugado utilizado na solução de sistemas lineares algébricos e os passos do algoritmo. Na seção seguinte, mostraremos as rotinas que compõem a biblioteca de operações paralelas para multiprocessadores hipercúbicos baseado em *transputers*. Em seguida será visto o algoritmo implementado a partir da biblioteca. Concluiremos então, com comentários sobre o desempenho do algoritmo no computador paralelo NCP I, composto de oito processadores (*transputers*).

2 O Método do Gradiente Conjugado

Para uma matriz $A(n,n)$, simétrica e definida positiva, a solução do sistema linear algébrico [1]

$$Ax = b$$

é equivalente ao seguinte problema de otimização sem restrições

$$minQ(x) = \frac{1}{2}x^t Ax - b^t x.$$

A função $Q(x)$ tem um mínimo global onde o seu gradiente

$$\nabla Q(x) = Ax - b$$

é zero.

A forma geral dos métodos iterativos de minimização é dada por [1]

$$x^{k+1} = x^k - \alpha^k d^k, \quad k = 1, 2, \ldots$$

onde:

d^k é a direção vetorial de minimização;

α^k é o escalar que define a distância de movimento na direção d^k.

A forma de escolha de α^k e d^k, define uma grande variedade de métodos, entre eles, o método do gradiente conjugado [1] empregado neste trabalho.

O algoritmo do gradiente conjugado quando aplicado na solução de um sistema linear algébrico segue os seguintes passos:

1. Especificação de uma solução inicial x^0;
2. Cálculo da direção negativa do gradiente

$$r^0 = b - Ax; \tag{2.1}$$

3. Inicialização da direção vetorial

$$d^0 = r^0; \tag{2.2}$$

4. Enquanto o resíduo

$$\left\| r^{k+1} \right\|_2^2 \tag{2.3}$$

for maior que a precisão ε especificada, calcular

$$\alpha^k = -\frac{[r^k]^t.[r^k]}{[d^k]^t.[Ad^k]} \tag{2.4}$$

$$x^{k+1} = x^k - \alpha^k d^k \tag{2.5}$$

$$r^{k+1} = r^k + \dot{\alpha}^k d^k; \tag{2.6}$$

5. Cálculo do novo resíduo

$$\left\|r^{k+1}\right\|_2^2 \tag{2.7}$$

6. Se o resíduo for maior que a precisão ε, então calcula a nova direção conjugada d^{k+1}

$$\beta^k = \frac{[r^{k+1}]^t.[r^{k+1}]}{[r^k]^t.[r^k]} \tag{2.8}$$

$$d^{k+1} = r^{k+1} + \beta^k d^k \tag{2.9}$$

7. Retorna ao passo quatro.

3 A Biblioteca de Operações

As várias operações que compõem a biblioteca são realizadas com os processadores chamando a rotina apropriada com seus argumentos e trabalham sobre os seus segmentos locais ignorando a existência de seus vizinhos. A arquitetura hipercúbica [3], as topologias empregadas para tornar eficiente a aplicação da biblioteca e uma visão mais detalhada da biblioteca de operações paralelas, podem ser vistas em [2].

A biblioteca contém três grupos distintos de rotinas que atendem a comunicação, difusão e convergência de escalares, vetores e matrizes, e os operadores da álgebra linear. Existem além desses grupos principais, diversas rotinas auxiliares para se obter informações como a dimensão ou quantidade de processadores do hipercubo, identificação dos processos entre outras.A seguir são mostrados os passos básicos da biblioteca de operações paralelas.

3.1 Rotinas de comunicação

Toda a comunicação realizada pela biblioteca de operações paralelas é feita internamente as rotinas, tornado-as transparentes ao usuário. Nelas está

contida toda a dependência da arquitetura da máquina. Essas rotinas são apresentadas na tabela 3.1.

Tabela 3.1: Rotinas de Comunicação

canal_de_escrita ()	canal por onde a mensagem deve ser enviada
canal_de_leitura ()	canal por onde a mensagem deve ser recebida
roteamento ()	próximo processador a receber a mensagem [5]
envia_mensagem ()	
recebe_mensagem ()	

3.2 Rotinas de difusão e convergência

Os vetores e matrizes devem ser distribuídos adequadamente aos processadores segundo o código "Binary Reflected Gray Code" [4] e dependendo da aplicação do algoritmo têm diferentes tipos de alocação [4].

As rotinas de difusão tratam da distribuição de escalares, vetores e matrizes ao hipercubo,e as rotinas de convergência realizam o caminho inverso, coletam os dados do hipercubo.

3.3 Rotinas vetoriais e matriciais

Os vetores e matrizes são representados por estruturas de dados que contém todas as informações necessárias a sua manipulação dentro das rotinas tais como o tipo de alocação, tamanho da área de dados, ponteiros para a área de dados e quaisquer outras informações relevantes.

As rotinas que não envolvem comunicação são realizadas serialmente sobre o segmento local a cada processador e naquelas onde existem a comunicação, os segmentos locais são processados individualmente e transmitidos aos processadores vizinhos, segundo o mapeamento conveniente à aplicação [2].

As rotinas vetoriais são mostradas a seguir na tabela 3.2 e as matriciais na tabela 3.3.

Tabela 3.2: Rotinas Vetoriais

aloca_vetor	aloca segmento local de um vetor
deleta_vetor	desaloca memória de um vetor
converte_vetor	troca o tipo de alocação
vetor_nulo	$u_i = 0$
copia_vetor	$u_i = v_i$
soma_vet_a_esc_vet	$u_i = u_i + sv_i$
soma_esc_vet_a_vet	$u_i = su_i + v_i$
soma_vet_a_vet_vet	$u_i = u_i + v_i w_i$
desloca_vetor	$u_i = u_{(i-s) \bmod N}$
desloca_vetor_otimo	$u_i = u_{(i-s) \bmod N}$
maximo_vetor	$max_{0 \leq i < N} u_i$
soma_vetor	$\sum_{i=0}^{N-1} u_i$
produto_interno	$\sum_{i=0}^{N-1} u_i v_i$

4 Implementação do Método do Gradiente Conjugado

Para solucionar o sistema de equações lineares, foram definidos dois processos. Um processo de controle que realiza as operações de entrada e saída dos dados e um processo que efetivamente calcula a solução segundo o algoritmo do gradiente conjugado visto anteriormente.

O algoritmo do processo de controle realiza a captação, distribuição e coleta dos dados utilizados, e é mostrado a seguir:

Processo de controle

1. Para se obter os dados iniciais armazenados em arquivos, chamamos as rotinas de leitura tendo como argumentos os ponteiros dos arquivos;

   - existem duas rotinas para leitura de matrizes uma para matriz densa e outra para esparsa;

     $pte\_A = le\_matriz\_densa(pta\_A)$;

     $pte\_A = le\_matriz\_esparsa(pta\_A)$;

Tabela 3.3: Rotinas Matriciais

aloca_matriz	aloca segmento local de uma matriz
deleta_matriz	desaloca memória de uma matriz
converte_matriz	troca o tipo de alocação matricial
transposição_por_deslocamento	transposição matricial
transposição_por_bloco	transposição matricial
matriz_vetor	multiplicação de uma matriz por um vetor
matriz_matriz	multiplicação de matrizes
rank1_update	$A \leftarrow A + xy^T$
shuffle	troca os limites de área com os vizinhos

- para a leitura de um vetor temos:

  $pte\_b = le\_vetor(pta\_b);$

  $pte\_x^0 = le\_vetor(pta\_x^0);$

Essas rotinas retornam um ponteiro para a estrutura de dados que representam a matriz e os vetores.

2. A distribuição da matriz e dos vetores aos processadores que compõem o hipercubo, é realizada pelas rotinas de difusão:

   - os argumentos da rotina que distribui uma matriz são o ponteiro da estrutura de dados e o tipo de alocação conveniente à aplicação [2];

     $envia\_matriz\_ao\_cubo(pte\_A, tipo);$

   - o argumento da rotina de distribuição vetorial é o ponteiro da estrutura de dados;

     $envia\_vetor\_ao\_cubo(pte\_b);$

     $envia\_vetor\_ao\_cubo(pte\_x^0);$

3. Para receber o vetor solução chamamos a rotina de convergência vetorial que não contém argumentos;

   $pte\_x = recebe\_vetor\_do\_cubo().$

Assim, no processo de controle utilizamos sete rotinas da biblioteca de operações paralelas, três para leitura, três para o envio e uma para recepção, onde o usuário não teve qualquer preocupação com a arquitetura ou o modo como foram distribuídos e recebido a matriz e os vetores empregados na solução.

O algoritmo dos processos dos nodos está demostrado a seguir e é dividido em duas partes, uma de recepção e envio dos dados em relação ao processo de controle e outra do método do gradiente conjugado propriamente dito. A modularidade empregada é para facilitar a depuração do algoritmo.

Processo dos Nodos

1. Para receber a matriz e os vetores do processo de controle utilizamos as rotinas de recepção;

   - para a recepção matricial não há argumentos;

     $pte\_A = recebe\_matriz\_do\_controle();$

   - a recepção de vetores tem como argumento o tipo de alocação vetorial;

     $pte\_b = recebe\_vetor\_do\_controle(tipo);$

     $pte\_x^0 = recebe\_vetor\_do\_controle(tipo);$

   Essas rotinas retornam as estruturas de dados que representam a matriz e os vetores.

2. O cálculo do gradiente conjugado é realizado por um módulo separado para facilitar a depuração, cujos argumentos são os ponteiros das estruturas da matriz e dos vetores;

   $pte\_x = gradiente(pte\_A, pte\_b, pte\_x^0);$

   Esse módulo retorna o ponteiro do vetor solução.

3. Finalmente para enviar o vetor solução ao processo de controle utilizamos a rotina correspondente, tendo como argumento o ponteiro da estrutura de dados do vetor.

   $envia\_vetor\_ao\_controle(pte\_x);$

Aqui foram empregados quatro passos elementares da biblioteca, três correspondentes a recepção da matriz e dos vetores, e um de envio do vetor solução ao preocesso de controle.

Aplicando agora as rotinas da biblioteca de operações paralelas para compor o algoritmo do gradiente conjugado, temos:

Gradiente Conjugado

1. A especificação da solução inicial x^0 foi passada como argumento para a rotina e a precisão ε foi especificada com um valor fixo, interno a rotina;

2. O cálculo da direção negativa do gradiente da equação 2.1 é realizado com dois passos elementares:

   - primeiro é feito o produto entre a matriz A e o vetor solução inicial x^0, tendo como argumentos os ponteiros das estruturas de dados da matriz e do vetor;

     $$pte\_r^0 = matriz\_vetor(pte\_A, pte\_x^0);$$

     ao retornarmos dessa rotina temos $r^0 = Ax^0$;

   - a seguir chamamos a rotina de soma vetorial apropriada, cujos argumentos são as estruturas do vetor obtido pelo primeiro passo e do vetor b, e o escalar -1 para a subtração;

     $$soma\_esc\_vet\_a\_vet(pte\_r^0, pte\_b, -1);$$

   Ao término obtemos então r^0 da equação 2.1;

3. A inicialização da direção vetorial d^0 é conseguida com outro passo elementar que copia um vetor, enviando como argumento o vetor r^0;

   $$pte\_d^0 = copia\_vetor(pte\_r^0);$$

   Assim temos então a equação 2.2;

4. O resíduo da equação 2.3 é calculado através do produto interno, que recebe como argumentos a estrutura do vetor r^k;

   $$n^{k+1} = produto\_interno(pte\_r^0, pte\_r^0);$$

5. Para as sucessivas iterações verificamos se o resíduo obtido é maior que a precisão especificada, não sendo atendida a condição, calcula-se:

- a distância de movimento na direção d^k que corresponde a equação 2.4. São realizados uma atribuição a variável

  $n^k = n^{k+1}$;

  que corresponde ao numerador e mais duas chamadas aos passos elementares para se obter o denominador. Primeiro para se ter o produto entre a matriz A e o vetor d^k e depois o produto interno entre esse resultado e o vetor d^k. Temos assim:

  $pte\_Ad^k = matriz\_vetor(pte\_A, pte\_d^k)$;

  $d^k = produto\_interno(pte\_d^k, pte\_Ad^k)$;

  $\alpha^k = -n^k/d^k$;

- Para o cálculo do novo vetor solução (equação 2.5), utilizamos a rotina de soma vetorial tendo como argumentos as estruturas dos vetors x^k e d^k e o escalar α^k;

  $soma\_vet\_a\_esc\_vet(pte\_x^k, pte\_d^k, -\alpha^k)$;

- A equação 2.6 também é resolvida com a rotina de soma vetorial agora tendo como argumentos os vetores r^k, d^k e o escalar α^k

  $soma\_vet\_a\_esc\_vet(pte\_r^k, pte\_Ad^k, \alpha^k)$;

6. Para o cálculo do novo resíduo procedemos como da forma anterior empregando o produto vetorial, agora tendo o vetor r^k, calculado no passo acima, como argumento;

   $n^{k+1} = produto\_interno(pte\_r^k, pte\_r^k)$;

7. Verificamos novamente se o resíduo atende a precisão dada, ou em caso contrário, calculamos a nova direção conjugada através da soma vetorial e do valor de β^k da seguinte forma:

   - a equação 2.8 é obtida com os resultados já calculados do denominador de α^k e do novo resíduo n^{k+1};

     $\beta^k = n^{k+1}/d^k$;

   - e para a nova direção conjugada (equação 2.9) enviamos como argumentos as estruturas dos vetores d^k e r^k e o escalar β^k;

     $soma\_esc\_vet\_a\_vet(pte\_d^k, pte\_r^k, \beta^k)$;

Dessa forma, o algoritmo do gradiente conjugado foi implementado com cinco passos elementares da biblioteca de operações paralelas:

- Cópia de um vetor ($y = x$);
- Produto de uma matriz por um vetor ($y = Ax$);
- Soma do produto de um escalar por um vetor com outro vetor ($y = ey + x$) e
- Soma de um vetor com o produto de um escalar por outro vetor ($y = y + ex$).

5 Conclusão

A biblioteca de operações está desenvolvida na linguagem *C Paralela* do *transputer*. Testes estão sendo realizados para a sua avaliação. Tentamos demonstrar as facilidades permitidas ao usuário que não se ocupará com a arquitetura ou com a comunicação. Tendo somente que saber o que realiza cada passo individualmente, utilizá-los com seus argumentos convenientes, e agrupá-los de modo a obter a solução do seu problema, tendo a facilidade da modularidade para a depuração. Atualmente temos disponível o computador NCP 1 desenvolvido pela COPPE - UFRJ que tem uma arquitetura hipercúbica com oito processadores *transputers*. Futuramente teremos além dos *transputers*, um processador i860 em cada nodo do hipercubo que permitirá o processamento vetorial de cada segmento distribuído. Esperamos então que haja um considerável aumento no desempenho da biblioteca de operações paralelas.

Referências

[1] Ortega, James M., "Introduction to Parallel and Vector Solution of Linear Systems", Plenum Press, 1988.

[2] Castro, M.C.S. de e Amorim, C.L. de "Uma Biblioteca de Operações Vetoriais e Matriciais para Multiprocessadores Hipercúbicos Baseado em Transputers", Anais do IX Congresso da SBC, 1989.

[3] Hwang, Kai e Briggs, Fayé A., "Computer Architecture and Parallel Processing", McGraw Hill, Inc 1894.

[4] McBryan, Oliver A. e Velde, Eric F. Van de, "Hypercube Algorithms and Implementations", SIAM J. Sci, Statistic Comput. 8 (1987) pp 227-287.

[5] Dirk C. e Reed, Daniel A., "Networks For Parallel Processor Measurements and Prognostications", ACM 1988, pp 610 - 619.