RESOLUÇÃO PARALELA DE SISTEMAS ESPARSOS DE EQUAÇÕES LINEARES

R. D. Arantes[†] e **C. L. de Amorim**[‡]

RESUMO

Neste trabalho considera-se a resolução de sistemas lineares esparsos em arquiteturas paralelas do tipo Hipercubo. O método de Fatoração de Cholesky do tipo "Fan-In" Distribuído é tomado como base para as comparações de desempenho. Uma análise preliminar do desempenho do método em ambientes paralelos para matrizes extraídas de aplicações reais é apresentada, contando-se atualmente com os resultados de um simulador desenvolvido especificamente para o processo de fatoração paralela.

ABSTRACT

In this work we consider the solution of sparse linear systems in parallel architectures of Hypercube type. The Distributed "Fan-In" Sparse Cholesky Factorization method is adopted as the basis for comparisons. A preliminar analysis of the performance of the method in parallel environments for the solution of real world problems is presented. The actual results were obtained with a simulation program developed specifically for the parallel factorization process.

1. Introdução

O problema a ser considerado neste trabalho é o da solução de sistemas lineares da forma $A\,x = b$, onde a matriz de coeficientes A é esparsa, simétrica e positivo definida.

O método de solução adotado consiste na fatoração de Cholesky $L\,L'$ da matriz do sistema, onde L representa uma matriz triangular inferior e L' a sua transposta.

[†] BSc Engenharia Elétrica (PUC/RJ - 1989)
Aluno de Doutorado da COPPE/Sistemas (UFRJ)
Areas de interesse: Esparsidade, Otimização e Processamento Paralelo
E-mail: arantes@rio.cos.ufrj.br

[‡] MSc (COPPE - 1979), PhD (Imperial College - 1984)
Professor Adjunto da COPPE/Sistemas (UFRJ)
Areas de interesse: Supercomputação e Processamento Paralelo
COPPE/UFRJ, Caixa Postal 68511, CEP 21945-970, Rio de Janeiro, RJ
E-mail: amorim@rio.cos.ufrj.br

De posse da matriz de fatores L, e' possivel resolver o problema original, mediante a solução dos seguintes sub-sistemas:

$$L\ y = b \qquad (\text{"}forward\text{"})$$

$$L'\ x = y \qquad (\text{"}backward\text{"})$$

Para o problema particularmente sendo considerado neste trabalho, não é necessário o pivotemento em valor numérico durante a fase de fatoração, uma vez que pelo fato da matriz do sistema ser positivo definida, a fatoração de Cholesky ser um processo estável por natureza.

Assim o problema recai em se obter uma seqüência "estatica" de pivoteamentos efetuados uma unica vez durante uma fase inicial de pré-processamento, com o objetivo de se reduzir o esforço computacional (evitando a criação de novos elementos não nulos durante o processo) e de aumentar-se a independência entre as sub-tarefas (correspondentes a eliminação simultânea de porções distintas da matriz alocadas entre os varios processadores).

Os dois objetivos acima são muitas vezes conflitantes entre si, pois a minimização do número de fill-in's (elementos novos introduzidos durante o processo) nem sempre acarreta a solução com o maior grau de liberdade entre as tarefas, e vice-versa (caso um reordenamento baseado apenas na maximização da independência entre as sub-tarefas seja adotado), pois invariavelmente esta estratégia isolada pode vir a ocasionar um número maior de operações aritméticas do que seria obtido com um ordenamento pelo critério de "menor grau" (minimizando-se apenas o número de fill-in's introduzidos).

Uma solução "definitiva" para este problema, ainda encontra-se em aberto, pois o problema de reordenamento por si só (independentemente do tipo de critério de decisão utilizado), é um problema NP-Completo.

Assim o que se utiliza na prática são métodos "heuristicos", (consagrados ao longo das últimas décadas), e voltados normalmente para uma determinada classe de problemas em particular (como por exemplo as heuristicas do tipo "nested dissection" para problemas de elementos finitos).

Uma solução para o problema do reordenamento normalmente adotada consiste em se empregar uma heuristica para minimização de fill-in's (como as de "minimum degree" ou "nested dissection"), e a seguir aplicar modificações sobre o ordenamento básico, com o objetivo de se aumentar a independência entre

as tarefas.

Outra solução consiste em se usar como critério de desempate, uma heurística de menor grau "modificada", em que o desempate durante a fase de decisão do processo, é feito tomando-se como base o aumento da independência entre as tarefas.

Dependendendo do tipo de reordenamento empregado, um sub-produto típico desta fase, é a estrutura de elementos não nulos da matriz de fatores que resultará ao final do processo de eliminação, uma vez que o critério de "menor grau" consiste na simulação (a nível estrutural) do processo de eliminação, partindo-se do grafo original de representação da matriz A, e aplicando-se ao mesmo sucessivas transformações (correspondentes a eliminação das linhas associadas), até se chegar no final do processo ao grafo representante da matriz de fatores L.

Nos casos em que esta informação não pode ser obtida "gratuitamente" (como sub-produto do ordenamento), um processo de "fatoração simbólica" (operando apenas com a informação estrutural dos elementos) pode ser aplicado em tempo na maioria das vezes proporcional ao número total de fatores resultantes (e portanto consideràvelmente inferior ao esforço da fase numérica em que o sistema será efetivamente solucionado).

A determinação da estrutura da matriz de fatores a priori, traz uma série de benefícios, (como a redução dos overheads dinâmicos decorrentes criação de fill-in's que seriam inevitáveis de outra forma), bem como possibilita o uso de ferramentas bem mais poderosas na fase posterior de análise e divisão de tarefas entre os processadores.

Uma vez determinada uma sequência de reordenamento (simétrica) das linhas e colunas da matriz e a estrutura da matriz de fatores resultante, pode-se lançar mão de ferramentas auxiliares como a "árvore de caminhos de eliminação", obtida com facilidade a partir da estrutura da matriz de fatores.

A "árvore de caminhos de eliminação", fornece como informação básica a topologia (hierarquia) de dependência entre as diversas linhas da matriz. Nós (linhas) que estejam em sub-árvores distintas podem ser alocadas em processadores distintos e operados de forma independente.

A estrutura de hierarquia vai das folhas para a raiz, onde cada nó só pode ser operado depois que todos os seus antecessores na árvore tiverem sido operados.

A informação contida na "árvore de caminhos de eliminação" encontra-se num nível de granularidade "médio", pois não leva em conta cada um dos elementos da linha associada a cada nó em particular, e que se fossem explorados em termos de uma abordagem com baixa granularidade, permitiriam a liberação de processadores a medida que as operações entre porções de elementos de cada linha, viabilizassem um avanço no "grafo de tarefas" associado ao processo como um todo.

Uma abordagem a este nível mais baixo de granularidade é muito onerosa e não é utilizada na prática, pois requer um espaço adicional considerável (da ordem do número de operações aritméticas a serem realizadas ao longo de todo o processo).

Assim, o que se utiliza na prática é a "árvore de caminhos de eliminação" como base para o mapeamento das diversas linhas da matriz original entre os processadores, (efetuado antes de se iniciar a fase de processamento numérico propriamente dito).

Uma vez que não é necessário pivoteamento "dinâmico" durante do processo, o problema de "balanceamento da carga", é concentrado apenas na fase inicial de mapeamento e na determinação da seqüência de alocação de tarefas associadas a cada um dos processadores.

De posse dessas informações, o algoritmo de fatoração esparsa baseado no método "Fan-In Distribuido" (e que será apresentado em detalhe em seções subseqüentes), pode então ser aplicado para se obter a matriz de fatores L. Uma vez concluida a fatoração, as fases de "forward" e "back substituition" (mencionadas no início desta seção) permitem a obtenção do vetor solução x do problema original.

2. Método de Cholesky (caso Esparso)

A única diferença em relação a variante "densa" do método é a exploração da existência de um número significativo de elementos nulos em cada linha da matriz, o que possibilita uma redução considerável no esforço computacional necessario durante o processo de eliminação, se forem evitadas operações desnecessárias sobre os elementos nulos.

Assim uma versão equivalente do algoritmo de Cholesky, para o caso esparso é a seguinte:

```
para j de 1 até n faça
    para cada k < j tal que a_jk ≠ 0 faça
        cmod (j, k)
    cdiv (j)
```

Onde no caso esparso as operações **cmod** e **cdiv** são expressas:

```
cmod (j, k):   para cada i entre j e n tal que a_ik ≠ 0 faça
                   a_ij ← a_ij - a_ik * a_jk

cdiv (j):      a_jj ← √a_jj
               para cada k entre j + 1 e n tal que a_jk ≠ 0 faça
                   a_kj ← a_jk / a_jj
```

Levando-se em conta as informações contidas na "árvore de caminhos de eliminação", e introduzindo-se a notação **Tcol (j)** para designar a tarefa de eliminação (de todos os elementos associados) de cada linha j, percebe-se que no caso esparso, existe intrinsecamente um maior grau de independência entre as tarefas, pois operações **cmod** (j_1, k) e **cmod** (j_2, k) para colunas j_1 e j_2 em sub-árvores distintas, podem ser efetuadas concorrentemente.

Em termos desta nova notação, um pseudo-código para o caso esparso fica sendo:

```
para cada coluna j de 1 a n tal que as tarefas Tcol (.)
de todos os seus antecessores na arvore de eliminação
tenham sido completadas faça
    Tcol (j)
```

Sendo a tarefa **Tcol (j)** expressa por:

```
Tcol (j):      para cada k < j tal que a_jk ≠ 0 faça
                   cmod (j, k)
               cdiv (j)
```

Este algoritmo fornece uma primeira forma de abordagem para a solução paralela do problema, pois as operações **Tcol (j)** para colunas j em ramos distintos da árvore podem ser efetuadas em paralelo.

Esta é uma informação "básica" (obtida atraves da arvore de caminhos) e que pode ser explorada plenamente na fase de mapeamento, como será detalhado ao longo do texto.

3. Método "Fan-In" Distribuido

Nas seções anteriores não se considerou a divisão de tarefas entre os vários processadores, de modo a explorar o paralelismo inerente ao processo de fatoração esparsa.

O método "Fan-In", apresenta como pontos favoráveis, um menor volume de comunicação que os demais métodos paralelos como o "Fan-Out", e uma simplicidade de código em relação aos métodos "Multifrontais Distribuidos".

Desse modo, a escolha do método "Fan-In", parece ser a mais concisa e elegante para uma primeira abordagem paralela do problema, e nesta seção, portanto será considerado apenas o caso "geral" do método "Fan-In" Distribuido.

A essência do método de eliminação de Cholesky (com geração dos fatores por colunas), consiste em se gerar sucessivamente as colunas de L, que por sua vez serão utilizadas para gerar demais colunas que venham a depender destas, e assim sucessivamente.

Mais precisamente as colunas anteriores a uma dada coluna j e que serão subraídas desta, correspondem as colunas associadas aos elementos não nulos da j'ezima linha da matriz de fatores sendo gerada.

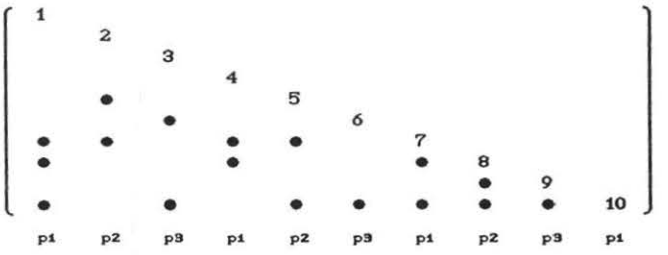
Assim é comum na pratica o uso de "vetores de contribuições" associados a cada coluna e que são posteriormente agregados entre si (compondo o somatório de todas as contribuições de colunas que afetam a coluna correntemente sendo gerada).

O método "Fan-In" tem como principal caracteristica, a eficiente exploração desta propriedade, reduzindo desta forma o volume de comunicação entre os processadores, mediante o acúmulo de todas as contribuições das colunas (em cada processador) que afetam a coluna corrente, enviando ao processador de destino (contendo a j'ezima coluna sendo gerada), o resultado final agregado de todas as contribuições relativas a cada processador.

Nos métodos anteriormente implementados na literatura, a medida que as colunas vão sendo geradas, estas tem de ser repassadas a todas os demais processadores contendo colunas que dependam da j'ezima coluna recém gerada. Isso sem dúvida acarreta um volume adicional desnecessário de comunicação, pois cada linha acaba sendo envidada isoladamente como uma contribuição, ao passo que no método "Fan-In", apenas um vetor de contribuições agregadas é enviado como mensagem.

Um exemplo prático do que foi dito, pode ser notado a seguir.

Considere-se a seguinte matriz de fatores L, resultante da aplicação do processo de eliminação até a coluna 6,



onde foi suposto um mapeamento do tipo "wrap" entre 3 processadores, alocando-se desta forma um conjunto de colunas a cada processador (de forma ciclica).

Considere agora a eliminação da coluna 7 e que pode ser esquemáticamente representada na forma:

$$\begin{pmatrix} l_{77} \\ l_{87} \\ 0 \\ l_{10,7} \end{pmatrix} = \begin{pmatrix} a_{77} \\ a_{87} \\ 0 \\ a_{10,7} \end{pmatrix} - l_{71}\begin{pmatrix} \bullet \\ \bullet \\ \bullet \end{pmatrix} - l_{72}\begin{pmatrix} \bullet \\ \end{pmatrix} - l_{74}\begin{pmatrix} \bullet \\ \bullet \end{pmatrix} - l_{75}\begin{pmatrix} \bullet \\ \bullet \end{pmatrix}$$

No esquema "Fan-In", ao invés de receber cada contribuição isoladamente, cada processador se encarrega de acumular as contribuições a serem enviadas ao processador contendo a coluna 7 (em questão).

Assim, cada processador irá préviamente calcular

$$u[7,p1] = l_{71}\begin{pmatrix} \bullet \\ \bullet \\ \bullet \end{pmatrix} + l_{74}\begin{pmatrix} \bullet \\ \bullet \end{pmatrix}$$

$$u[7,p2] = l_{72}\begin{pmatrix} \bullet \\ \end{pmatrix} + l_{75}\begin{pmatrix} \bullet \\ \bullet \end{pmatrix}$$

e enviar o resultado a p1

permitindo ao processador 1 (contendo a linha 7) montar definitivamente esta coluna mediante

$$\begin{pmatrix} l_{77} \\ l_{87} \\ 0 \\ l_{10,7} \end{pmatrix} = \begin{pmatrix} a_{77} \\ a_{87} \\ 0 \\ a_{10,7} \end{pmatrix} - u[7,p1] - u[7,p2]$$

Em termos mais formais, assumindo-se que um mapeamento **map[.]** das colunas entre os **p** processadores disponiveis seja a

priori conhecido (o que será abordado com maiores detalhes na próxima seção), pode-se descrever um pseudo-código para o algoritmo "Fan-In" Distribuido (a ser executado concorrentemente com uma versão idêntica de código em cada um dos processadores).

Algoritmo Fan-In Distribuído

```
Def: proc_cols ≡ { j | map[j] = proc_id }

para cada coluna j de 1 até n faça
    se lin[j,proc_id] ≠ ∅ ou se j ∈ proc_cols então
        u[j,proc_id] ← 0
        para k ∈ lin[j,proc_id] faça
            u[j,proc_id] ← u[j,proc_id] + l_jk (l_jk,...,l_nk)'
        se j ∉ proc_cols então
            envie contribuição agregada u[j,proc_id]
            para o processador map[j]
        senão
            L_*j ← (a_jj,...,a_nj)' - u[j,proc_id]
            enquanto todas as contribuições não forem
            recebidas faça
                receba as contribuições agregadas u[j,π]
                para a coluna j (provenientes dos demais
                processadores π) e subtraia cada uma das
                contribuições u[j,π] de L_*j mediante
                L_*j ← L_*j - u[j,π]
            cdiv (j)
```

No código acima, **proc_id** representa a identificação associada a cada processador, **proc_cols** o conjunto de colunas a ele associadas, **lin[j,π]** o conjunto de colunas a serem agregadas no processador π e que contribuem para a formação da coluna j, ou mais explicitamente:

$$\text{lin}[j,\pi]: \quad \{ k \in \text{Struct}(L_{j*}) \mid \text{map}[k] = \pi \}$$

onde $\text{Struct}(L_{j*})$ denota a "estrutura" da linha j (o conjunto de índices das colunas associadas a linha j, correspondentes aos elementos não nulos da linha em questão).

Finalmente, as contribuições agregadas $u[j,\pi]$ consistem no acúmulo de todas as contribuições relativas a coluna j das colunas que se encontram alocadas no processador π, ou seja:

$$u[j,\pi]: \quad \sum_{k \in \text{lin}[j,\pi]} l_{jk} (l_{jk},\ldots,l_{nk})'$$

4. Mapeamento das Linhas entre os Processadores

O problema de mapeamento das linhas da matriz entre os diversos processadores é de vital importância para a eficiência final do método, pois uma vez que o balanceamento das operações é "estático" e realizado durante esta fase particular do processamento, quaisquer benefícios ou acréscimos de informações obtidas nesta fase são merecedoras de atenção, na esperança de com isso melhorar o balanceamento da carga ou reduzir os atrasos de comunicação entre processadores.

Experiências práticas comprovaram que mapeamentos do tipo "wrap", tomando como base níveis sucessivos da "árvore de camninhos de eliminação" não produzem bons resultados pelo excessivo espalhamento de tarefas entre os processadores, obrigando os mesmos a um volume maior de comunicação, pelo fato das linhas a serem acessadas num curto intervalo de tempo após a linha correntemente sendo operada (e que porventura venham a sofrer contribuições desta), não estarem alocadas ao mesmo processador, tornando inevitavel deste modo, uma forçosa troca de informações entre processadores.

Tal deficiência pode ser contornada, se forem alocados num mesmo processador, todas as linhas envolvidas numa mesma "sub-árvore" durante o processo de eliminação.

Este tipo de mapeamento, originalmente denominado de "sub-árvore para sub-cubo", foi inicialmente idealizado para problemas de elementos finitos, com matrizes de estruturas regulares do tipo grid, onde a árvore de fatoração resultante (mediante um ordenamento do tipo "nested-dissection") possui na maioria dos casos "simetria perfeita" de alturas (sendo portanto já balanceada por construção).

Para problemas irregulares sem uma estrutura tipica como a de elementos finitos, a generalização deste esquema de mapeamento torna-se bem mais complexa, pois as sub-árvores a serem consideradas passam a ser de topologias variáveis e portanto com um peso computacional e volume de comunicações distinto o que sem dúvida dificulta a implementação da idéia original, baseada em sub-árvores de mesma altura (com topologias idênticas).

Assim, para este trabalho no qual se objetiva a implementação de um método genérico de solução, outros esquemas de mapeamento, baseados na "essência" do mapeamento "sub-árvore para sub-cubo" devem ser considerados.

A abordagem mais natural para a questão é tentar obter o balanceamento de carga, mantendo a maior porção de linhas em cada sub-árvore alocadas a um mesmo processador.

Como critério adicional para se decidir a alocação das linhas aos processadores, pode-se lançar mão de uma estimativa a priori do volume de comunicação, bem como do esforço de cálculo, procurando se estabelecer durante a fase de mapeamento, uma primeira aproximação do que será dispendido durante a execução da fase numérica propriamente dita.

Assim, começando-se das folhas da árvore, pode-se ir gradualmente determinando o esforço computacional necessário para se chegar até cada um dos nós, sendo o volume de cálculo necessário para se completar a eliminação até um dado nó, igual ao esforco computacional da eliminação do próprio nó, acrescido dos volumes de cálculos necessários para se chegar a cada um dos seus predecessores diretos na arvore.

A idéia básica consiste em uma vez determinado o esforço computacional necessário para se chegar até cada nó da arvore, tentar dividir o mais igualmente possivel, o esforço total, partindo-se da raiz em direção as folhas.

Assim, o que se objetiva é uma divisão proporcinal de todo o esforço computacional, pelo número disponivel de processadores, buscando preservar ao máximo a estrutura de sub-árvores (alocadas sempre que possivel) para cada um dos processadores isoladamente.

Um esquema de divisão como este, não leva em conta os atrasos inevitáveis decorrentes da comunicação entre os processadores.

Uma abordagem incluindo esta informação vem sendo idealizada, mas até o momento ainda não veio a ser concretizada como uma forma alternativa de ataque ao problema.

Muito ainda pode ser feito nesta área e a atual abordagem sendo proposta é passível de sofrer revisões e aprimoramentos ao longo do tempo.

5. Implementação Computacional

Para a implementação do método de fatoração esparsa em paralelo, contou-se com um extenso conjunto de sub-rotinas auxiliares, executadas seqüencialmente na fase inicial de pré-processamento dos dados.

O que foi codificado em paralelo foi somente o Método "Fan-In" distribuido (apresentado na seção 3).

As rotinas auxiliares visam entre outras tarefas, a obtenção de uma seqüência de reordenamentos estáticos das linhas e colunas da matriz original para se minimizar a introdução de novos elementos (fill-in's) no processo, bem como aumentar-se a independência entre as diversas linhas a serem eliminadas.

De posse da matriz reordenada, a estrutura de fatores da matriz L resultante pode ser obtida, bem como a "arvore de dependências de eliminação", e que permite uma análise criteriosa de quais conjuntos de linhas alocar a cada processador, correspondendo a fase de "mapeamento" (particionamento das linhas da matriz original entre os varios processadores).

Toda esta etapa de pré-processamento (dedicada e efetuada especificamente para cada problema a ser solucionado) foi codificada em FORTRAN (pois a maioria dos códigos de reordenamento esparso ainda se encontram disponiveis nesta linguagem).

Como esta etapa inicial teve como principal objetivo de projeto uma maior flexibilidade e modularidade, permitindo o fácil intercâmbio dos diferentes módulos de ordenamento e particionamento disponiveis, e como atualmente não se considerou a implementação paralela desta fase, o uso de FORTRAN não veio a se constituir um entrave ao pleno andamento do projeto.

Um dado adicional e que de novo justifica o uso de FORTRAN para esta fase é que o formato de entrada dos dados dos problemas teste (extraidos da NETLIB [8], contendo problemas reais na área de otimização), encontram-se no formato padrão MPS, e para o qual rotinas de leitura em FORTRAN já se encontram disponiveis e amplamente testadas.

Para a parte paralela do código no entanto, o uso de FORTRAN se constituiria uma severa limitação, razão pela qual adotou-se a linguagem C paralela disponivel no sistema operacional Helios [7].

O código paralelo assume portanto como entrada, a estrutura já particionada da matriz original, e enviada a cada um dos processadores pelo processador central. Cada processador é responsável pela eliminação de um dado conjunto de linhas, e fornece ao final do processo as linhas correspondentes da matriz de fatores L, enviando-as de volta ao processador central.

Para se obter a solução definitiva do sistema linear

original, é preciso efetuar-se as fases de substituição forward e backward, operando com os sistemas triangulares resultantes. Esta fase no trabalho atual não se encontra paralelizada e é executada pelo processador central após receber as linhas da matriz de fatores de cada um dos processadores.

Além do codigo de fatoração paralela, desenvolveu-se também como ferramenta auxiliar, um simulador do processo, e que permite uma fácil "reconfiguração" dos parametros arquiteturais como os tempos e taxas de startup, de comunicação de mensagens e de cálculo em ponto flutuante e em aritmética inteira, e que permitiu levantar conclusões preliminares interessantes sobre o desempenho da implementação em paralelo de métodos de fatoração esparsa em ambientes baseados (especificamente) no sistema operacional Helios.

6. Resultados Preliminares

Os resultados apresentados nesta seção foram obtidos mediante a simulação do processo de fatoração paralela em uma arquitetura como a de Transputers T-800 e tomando como base o tempo de cálculo e de comunicação entre processadores utilizando-se o sistema operacional paralelo Helios [7].

Na tabela a seguir são apresentadas algumas estatísticas básicas sobre os problemas e alguns resultados da simulação a nível do número de mensagens recebidas (RD) e enviadas (WR) e de operações aritméticas inteiras (Nint) e de pto. flutuante (Nflop).

Os dados abaixo referem-se a um ordenamento para redução de fill-in's baseado na heurística do "menor grau", adotando-se uma estratégia de mapeamento do tipo "sub-árvore p/ sub-cubo" para 3 processadores. Os tempos estimados tomaram como base uma arquitetura como a do T-800, cujos parâmetros de tempo de cálculo (inteiro e ponto flutuante) e de troca de mensagens (comunicação e startup) são apresentados logo após a tabela de resultados.

Inicialmente é listado o nome de cada problema MPS da Netlib utilizado nas comparações, bem como os parâmetros caracteristicos básicos de cada problema como dimensão, densidade da matriz original, número de elementos, fatores não nulos e % de fill-in's.

A seguir apresenta-se resultados como o volume e tamanho das mensagens transmitidas, numero de operações (inteiras + pto. flutuante), bem como o tempo total gasto na troca de mensagens (Tcom + Tstartup), globalmente e para cada um dos processadores.

```
================================================================
Prob: AFIRO  Dim: 27  Dens: 21%  NzA: 63  NzL: 80  Fil: 21%

TotMsg  TotLen  TotOper  TimMsg   TimOper
  53      191     2655    0.077    0.023

Proc #  NMsg   LenMsg   NInt   NFlops   TimMsg   TimOper
   1     25      86     889     279     0.036    0.010
   2     13      50     574     140     0.019    0.006
   3     15      55     618     155     0.022    0.006

TotMsgRD  TotMsgWR  TotLenMsgRD  TotLenMsgWR  TotTstart  TotTcom
  33        20         119          72         0.077     0.0001
================================================================
Prob: ADLITTLE  Dim: 56  Dens: 23%  NzA: 328  NzL: 355  Fil: 8%

TotMsg  TotLen  TotOper  TimMsg   TimOper
 106     860    13381    0.156    0.124

Proc #  NMsg   LenMsg   NInt   NFlops   TimMsg   TimOper
   1     40     267     3197    1114     0.059    0.039
   2     42     383     3837    2147     0.062    0.057
   3     24     210     2287     799     0.035    0.028

TotMsgRD  TotMsgWR  TotLenMsgRD  TotLenMsgWR  TotTstart  TotTcom
  67        39         545          315        0.154     0.0004
================================================================
Prob: SC50A  Dim: 50  Dens: 10%  NzA: 101  NzL: 180  Fil: 44%

TotMsg  TotLen  TotOper  TimMsg   TimOper
  94     476     6190    0.138    0.055

Proc #  NMsg   LenMsg   NInt   NFlops   TimMsg   TimOper
   1     47     232     1978     689     0.069    0.024
   2     17      83     1196     270     0.025    0.012
   3     30     161     1545     512     0.044    0.018

TotMsgRD  TotMsgWR  TotLenMsgRD  TotLenMsgWR  TotTstart  TotTcom
  63        31         319          157        0.137     0.0002
================================================================
Prob: SC105  Dim: 105  Dens: 5%  NzA: 226  NzL: 471  Fil: 52%

TotMsg  TotLen  TotOper  TimMsg   TimOper
 187     1175    16497    0.275    0.150

Proc #  NMsg   LenMsg   NInt   NFlops   TimMsg   TimOper
   1     67     401     3874    1228     0.098    0.045
   2     56     364     4143    1568     0.082    0.052
   3     64     410     4144    1540     0.094    0.051

TotMsgRD  TotMsgWR  TotLenMsgRD  TotLenMsgWR  TotTstart  TotTcom
 128        59         804          371        0.273     0.0006
================================================================
Prob: SHARE2B  Dim: 96  Dens: 18%  NzA: 775  NzL: 960  Fil: 19%

TotMsg  TotLen  TotOper  TimMsg   TimOper
 166     1707    39797    0.246    0.378

Proc #  NMsg   LenMsg   NInt   NFlops   TimMsg   TimOper
   1     94     950     12916    8065     0.139    0.202
   2     29     311     5738     2202     0.043    0.072
   3     43     446     7201     3675     0.063    0.102

TotMsgRD  TotMsgWR  TotLenMsgRD  TotLenMsgWR  TotTstart  TotTcom
 126        40        1330          377        0.242     0.0009
```

```
================================================================
Prob: SC205  Dim: 205  Dens: 3%  NzA: 451  NzL: 979  Fil: 54%

TotMsg  TotLen  TotOper  TimMsg   TimOper
 340     2324    34297   0.501    0.313

Proc #  NMsg    LenMsg   NInt    NFlops  TimMsg   TimOper
   1    177     1190    12054    5510    0.261    0.164
   2     66      440     4972     618    0.097    0.047
   3     97      694     8039    3104    0.143    0.102

TotMsgRD  TotMsgWR  TotLenMsgRD  TotLenMsgWR  TotTstart  TotTcom
  239       101       1620          704       0.496     0.0013
================================================================
Prob: BLEND  Dim: 74  Dens: 28  NzA: 743  NzL: 940  Fil: 21%

TotMsg  TotLen  TotOper  TimMsg   TimOper
 192     2818    46876   0.286    0.455

Proc #  NMsg    LenMsg   NInt    NFlops  TimMsg   TimOper
   1     82      1073    10466    6428    0.122    0.163
   2     58       972    10297    7973    0.086    0.181
   3     52       773     7631    4081    0.077    0.111

TotMsgRD  TotMsgWR  TotLenMsgRD  TotLenMsgWR  TotTstart  TotTcom
  126       66        1838          980       0.280     0.0015
================================================================
```

A tabela a seguir ilustra os dados relativos aos tempos de comunicação num T-800 rodando a 20 Mhz e foram obtidos de [7], com os tempos **Tstartup** e **Tcom** relativos ao startup e a transmissão de mensagens de 1 byte (sendo expressos em micro-segundos).

	Nivel de comunicação	Tstartup	Tcomm
	Direto	2.2	0.56
	Primitivo	124.4	0.56
	SYSTEM	1445.0	0.57
(*)	POSIX (Helios)	1459.7	0.57

De ambas as tabelas pode-se concluir que o tempo de startup e' o fator determinante da comunicação, a menos que mensagens de tamanho elevado venham a ser transmitidas o que não ocorre na prática no caso esparso, onde em média as mensagens costumam ser da ordem de no máximo 1 K bytes em casos "extremos", (para linhas com mais de 200 elementos nao nulos e de densidade elevada).

Em casos típicos de matrizes de potência, o tempo de startup é dominante não apenas sobre a comunicação como sobre o tempo de cálculo, pois nestes casos é comum encontrar-se apenas de 2 a 3 elementos por linha o que acarreta um volume muito baixo de operações aritméticas face ao tempo total gasto com startup.

7. Conclusões

Dos resultados preliminares obtidos observa-se que na maioria dos casos, o tempo de startup para uma arquitetura baseada no sistema operacional Helios é dominante frente ao tempo de cálculo, e que para o caso considerado (com apenas 3 processadores), a eficiência de uma implementação paralela pode ser bastante questionavel, se comparada com uma implementação seqüencial para o problema, pois nessa análise preliminar não se levou em consideração os atrasos na comunicação por canais ou processadores ocupados com a transmissão de mensagens e que acabam reduzindo ainda mais a eficiência do processo paralelo.

A grande vantagem da fatoração em paralelo está na possibilidade de se atacar problemas de dimensão mais elevada, pois cada processador opera apenas sobre um conjunto restrito de linhas. O grande desafio para que isto se torne viável é a implementação paralela de todas as fases desde o reordenamento (atualmente executadas seqüencialmente em um único processador).

BIBLIOGRAFIA

[1] M.Heath, E.Ng & B.Peyton
Parallel Algorithms for Sparse Linear Systems
em Parallel Algorithms for Matrix Computations, SIAM Publ., 1990

[2] C.Ashcraft, S.Eisenstat, J.Liu & A.Sherman
A comparison of 3 Column-based Distributed Sparse Factorization Schemes, Tech Rep. YALEU/DCS/RR-810, Yale University, 1990

[3] C.Ashcraft, S.Eisenstat, J.Liu, B.Peyton & A.Sherman
A Compute-Ahead Implementation of the Fan-In Sparse Distributed Factorization Scheme
Oak Ridge National Laboratory, 1990, Tech. Report ORNL/TM-11496

[4] G.Geist & E.Ng
Task Scheduling for Parallel Sparse Cholesky Factorization
Int. J. Parallel Programming, vol 18 no 4, 1989, pp 291-314

[5] J.Liu
Computational Models and Task Scheduling for Parallel Sparse Cholesky Factorization
Parallel Computing, 3 (1986), pp 327-342

[6] J.Liu
Reordering Sparse Matrices for Parallel elimination
Parallel Computing, 11 (1989), pp 73-91

[7] I.Davies et all.
The Helios Parallel Operating System
Perihelion Software Ltd., 1991, Prentice-Hall Inc.

[8] D.Gay
Electronic Mail Distribuition of Linear Programming Test Problems
Math. Programming Society COAL Newsletter, 13 (1985), pp 10-12