IMPLEMENTAÇÃO DO MÉTODO DOS GRADIENTES CONJUGADOS EM MULTIPROCESSADORES COM ARQUITETURA HIPERCÚBICA

Roberto G. CABRAL e Eugenius KASZKUREWICZ
UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
COPPE - LAB. COMPUTAÇÃO PARALELA - C.P. 68516
21945-970 Rio de Janeiro - RJ - BRASIL
Fax: 55 21 2906626
E. Mail: coel0027@ufrj.bitnet

Resumo

Neste trabalho, são apresentadas implementações paralelas do algoritmo do Gradiente Conjugado (GC) Precondicionado para resolução de sistemas lineares de equações esparsas, em multiprocessadores com arquitetura hipercúbica. As computações do algoritmo GC paralelo consistem primordialmente de operações matriciais que são implementadas em cada nó do hipercubo, utilizando esquemas com paralelismo de dados. A implementação paralela síncrona (no iPSC/860) apresenta razoável desempenho, por outro lado uma versão assíncrona preliminar apresentou problemas de convergência em sistemas de equações de porte médio provenientes de cálculo estrutural.

Abstract

In this paper, parallel implementations of the Preconditioned Conjugate Gradient (CG) algorithm for the solution of large sparse linear systems of equations, on a hypercube multiprocessor (iPSC/860), are presented. Computations with the parallel CG algorithm mainly consist of matrix operations that are implemented on each node of the hypercube via data parallel schemes. The synchronous implementation presented reasonable performance, however a preliminary asynchronous version presented convergence problems when applied to a medium size structural calculation problem.

1 INTRODUÇÃO

A resolução de problemas científicos e de engenharia, tais como: análise de sistemas de energia elétrica, previsão meteorológica, exploração de petróleo, pesquisa em energia de fusão, simulações aerodinâmicas, sistemas especialistas, automação industrial, defesa militar, análise estrutural, entre outros, conduz a sistemas de equações algébricas lineares de grande porte da forma $Ax = b$. Esses sistemas requerem grandes recursos computacionais, tanto no aspecto da capacidade de cálculo, como de velocidade de processamento.

Métodos numéricos para solução dessas equações lineares podem ser classificados, de uma maneira geral, em duas classes: iterativos e diretos. Qual dessas classes é mais eficiente, é uma questão que não pode ser respondida de uma forma geral, uma vez que depende de como definimos " eficiência", e também do problema particular ou classe de problemas a serem resolvidos. Os métodos iterativos são atraentes em termos de requisitos de armazenagem de dados, desde que sua implementação requer somente A (esparsa), b, $x(i)$ e talvez um ou dois vetores a serem armazenados (sendo $x(i)$ o valor da aproximação da solução na iésima iteração). De outra parte quando A é fatorada, nos métodos chamados diretos, podem surgir elementos não nulos em posições que anteriormente eram ocupadas por elementos nulos. Assim, é muitas vezes verdade que os métodos

diretos para sistemas esparsos podem exigir mais armazenagem que as implementações de métodos iterativos [1]. Como mencionado por George e Liu, a sequência de operações pode afetar sobremaneira a quantidade de cálculos no processo de eliminação Gaussiana [6]. Já num esquema iterativo o número de iterações executadas, depende muito das características da matriz A [1], [11]. Uma comparação mais profunda de métodos iterativos e diretos em termos de requisitos computacionais, todavia, está fora do escopo deste trabalho.

Por outro lado, as observações acima tornam claro que, a menos que se explicite um contexto bem definido, é complicado ou mesmo impossível de se responder à questão sobre que classe de métodos iterativos ou diretos deve ser utilizada.

No presente trabalho, tratamos de um método da classe dos iterativos, ou seja, das versões síncronas e assíncronas do método do Gradiente Conjugado (GC), uma vez que no contexto paralelo, esse método em aplicações a sistemas de energia elétrica tem se mostrado competitivo, como mostra o trabalho de Decker [18]. Apresenta-se, para uma particular combinação algoritmo/arquitetura, a implementação do método dos gradientes conjugados para resolução de grandes sistemas esparsos de equações lineares, utilizando multiprocessadores interligados em uma arquitetura hipercúbica.

Na Sec. 2 descreve-se a versão sequencial do método do Gradiente Conjugado, apresenta-se uma proposta de paralelização do algoritmo, na sua versão síncrona, de modo a diminuir o número de trocas de mensagens entre os processadores utilizando esquemas de paralelismo de dados.

Na Sec. 3 expõe-se as características de comunicação síncrona e assíncrona no iPSC/860, na Sec. 4 apresenta-se a versão assíncrona do GC. Na Sec 5 apresenta os resultados computacionais do algoritmo, obtidos na máquina iPSC/860 com sistema operacional Unix-System V, quando aplicado a um exemplo de cálculo estrutural [4].

2 PARALELIZAÇÃO DO MÉTODO DO GRADIENTE CONJUGADO

O foco desta seção é a descrição da paralelização do método do Gradiente Conjugado visando o máximo ganho de velocidade de processamento, mais especificamente numa máquina de topologia hipercúbica e mais especificamente o iPSC/860 da Intel. A paralelização envolve vários problemas de software e hardware, que são expostos mais abaixo:

Na subseção 2.1 apresenta-se o método do Gradiente Conjugado (GC) na sua versão tradicional (sequencial). Na subseção 2.2 discorre-se sobre a influência da arquitetura da máquina escolhida na reformulação do algoritmo e dá-se uma rápida visão do sistema iPSC/860. Na subseção 2.3 aborda-se a concepção geral que conduz à estruturação do algoritmo, além de considerações sobre o particionamento do problema. Na subseção 2.4 descreve-se a estruturação do algoritmo paralelo, visando a obtenção de um máximo ganho de velocidade.

2.1 Método do Gradiente Conjugado

O método do gradiente conjugado é um método clássico [11] e as etapas básicas do algoritmo iterativo (para $k = 0, 1, ...$) são apresentadas abaixo, onde:

$$< x, y >:= x^T y \tag{1}$$

é o produto interno dos vetores x e $y \in \Re^n$ e o resíduo correspondente à k-ésima iteração é definido por:

$$r(k) := b - Ax(k) \tag{2}$$

Inicialmente, escolha $x(0)$ e faça

$$r(0) \;=\; p(0) = b \;-\; Ax(0)$$

então calcule $< r(0), r(0) >$ e $< b, b >$.

A seguir, para $k = 0, 1, 2, \ldots$ faça:

- $q(k) = Ap(k)$
- Calcule $< p(k), q(k) >$
- $\alpha(k) = \frac{< r(k), r(k) >}{< p(k), q(k) >}$
- $r(k+1) = r(k) - \alpha(k) q(k)$
- $x(k+1) = x(k) + \alpha(k) p(k)$
- Calcule $< r(k+1), r(k+1) >$
- $\beta(k) = \frac{< r(k+1), r(k+1) >}{< r(k), r(k) >}$
- $p(k+1) = r(k+1) + \beta(k) p(k)$

onde o resíduo $r(k)$ é associado ao vetor $x(k)$, isto é,

$$r(k) = b - Ax(k)$$

que deve tender a zero, quando $x(k)$ tende à solução. Sendo $p(k)$ o vetor direção na k-ésima iteração. Um critério adequado para interromper o processo iterativo é quando

$$\left(\frac{< r(k), r(k) >}{< b, b >} \right)^{\frac{1}{2}} < \epsilon \tag{3}$$

onde ϵ, no nosso caso, é da ordem de 10^{-5}.

A taxa de convergência do algoritmo GC depende sobremodo do número de condição κ da matriz A. Uma estimativa desta taxa é dada em [9] e [13]:

$$\| x(k) - x^* \|_2 \leq 2\sqrt{\kappa} \ \alpha(k) \ \| x(0) - x^* \|_2 \tag{4}$$

Assim, quanto menor for o número de condição, maior a taxa de convergência, já que neste caso,

$$\kappa := \frac{\lambda_{max}(A)}{\lambda_{min}(A)} \tag{5}$$

$\lambda_{max}(A) :=$ maior autovalor da matriz$A$, $\lambda_{min}(A) :=$ menor autovalor da matriz$A$,

Ou seja, quanto mais próximos estiverem, entre si, os auto-valores, mais rapidamente convergirá o algoritmo, conforme se pode verificar da equação (4). Caso os auto-valores da matriz A estejam excessivamente separados, o pré condicionamento pode ser utilizado, ou seja através de uma matriz não singular e simétrica C, tal que

$$\tilde{A} = (C^{-1})^T A C^{-1} \tag{6}$$

De modo que a nova matriz $\tilde{A}$ tenha um número de condição $\kappa(\tilde{A})$ menor, e assim, diminua o tempo de processamento, para a determinação da solução. A matrix C é denominada *"pré-condicionador"*; existem vários tipos de pré-condicionadores na literatura, tais como, o escalamento pela inversa da diagonal principal, a decomposição Incompleta de Cholesky, dentre outros, são os mais utilizados [10]. No presente trabalho utiliza-se a forma mais simples de pré-condicionamento, o escalamento das linhas e colunas da matriz A pela inversa da sua diagonal ou seja

$$C = \text{diag}(a_{11}, a_{22}, \ldots, a_{nn}). \tag{7}$$

Com esse tipo de condicionador o método é conhecido como GCED (gradiente conjugado escalado pela diagonal). No restante do trabalho, para facilidade de notação, o sistema já escalado será expresso por $Ax = b$, ao invés de $\tilde{A}\tilde{x} = \tilde{b}$.

2.2 Influência do tipo de Arquitetura utilizada

O conhecimento da arquitetura da máquina, onde será implementado o algoritmo paralelo é fundamental para um projeto eficiente. Assim, são fundamentais as informações de que se deve dispor sobre o multiprocessador, a topologia da conexão entre os processadores, o custo adicional de comunicação, a velocidade de computação, a interconexão entre o hospedeiro e os nós, os acessos de entrada/saída, e os recursos de memória.

O iPSC/860 da Intel é um computador de topologia hipercúbica, com 2^d nós (processadores), onde d é a dimensão do hipercubo. Nessa topologia, a distância máxima entre dois nós quaisquer, ao longo das interconexões, é d. Desde que, cada processador tem sua memória local particular e opera independentemente, a maneira dos nós se comunicarem entre si é via troca de mensagens. Neste contexto, também, são fundamentais um sistema operacional distribuído, e um coprocessador de comunicação em cada nó.

2.3 Paralelização de Algoritmos

De uma maneira genérica, o projeto de algoritmos paralelos pode ser realizado sob diferentes abordagens, como se infere dos trabalhos de Bertsekas [1], Gustafson [3] e mais especificamente em relação ao GC, nos trabalhos de Fox [2] e Aykanat [12].

- uma forma natural é a partir de um algoritmo serial já existente, após adaptações, efetuar sua paralelização, sem torná-lo demasiado específico a um particular tipo de máquina.
- em uma outra forma, a escolha do algoritmo e da máquina paralela são interdependentes ao ponto em que o projeto de um tenha uma forte influência no projeto do outro.

Este trabalho adota uma combinação das duas abordagens realizando a paralelização de um algoritmo sequencial existente (o Gradiente Conjugado), numa particular topologia paralela, a hipercúbica, buscando manter uma portabilidade para máquinas da mesma classe, ou seja, máquinas de memória distribuída.

O presente trabalho está centrado na ideía de tornar a versão paralela do GC em um algoritmo com paralelismo de dados, como definido no trabalho de Hillis [14]. Algoritmos com paralelismo de dados são adequados para problemas caracterizados pelo fato de que as mesmas operações são repetidamente e independentemente executadas sobre um conjunto diferente de dados. Desta forma, a paralelização é efetuada particionando-se o conjunto de dados, ao invés da partição da estrutura do algoritmo, e no caso, opera-se sobre os subconjuntos de dados que são alocados aos diferentes processadores.

Algoritmos paralelos estruturados sob a forma de paralelismo de dados foram originalmente planejados para utilização em sistemas com milhares de processadores [14], como o sistema Connection Machine, que possui uma topologia matricial de processadores SIMD. Contudo, seu paralelismo inerente também os faz adequados para execução em multicomputadores com memória compartilhada ou distribuída.

O propósito do particionamento é a divisão do problema em subproblemas de modo que estes possam ser distribuidos a diferentes processadores para serem resolvidos de maneira concorrente. Existem dois fatores importantes que podem afetar o acoplamento entre os subproblemas, e portanto a eficácia da partição, são eles: as dependências entre os dados e a limitação de recursos do sistema (quantidade de memória e velocidade de comunicação entre os processadores).

A dependência entre os dados determina o grau de paralelismo disponível no problema. Contudo, tanto o paralelismo realizável no multicomputador é restringido pelos recursos disponiveis do sistema como pela velocidade do processador, e rede de interconexão dos processadores.

O problema da partição pode ser simplificado quando a estrutura de dados for estática e regular, como mencionado por Ni e colaboradores [15]. No caso, a equação matricial $Ax = b$ estruturada em duas dimensões, permite que a partição seja ao longo das linhas, colunas, ou ambas, i.e., linhas e colunas.

2.4 Estruturação do algoritmo paralelo a partir do método do Gradiente Conjugado

A idéia básica do algoritmo com paralelismo de dados é de que em cada um dos processadores, sejam executadas, de forma concorrente, todas as operações básicas de matrizes e vetores do GC serial, utilizando porém, os dados particionados atribuídos a cada um desses processadores. Por outro lado, para se obter maiores ganhos de velocidade em um algoritmo com paralelismo de dados é necessário minimizar:

- o tempo ocioso dos processadores, através da distribuição equilibrada do conjunto de dados, entre os processadores de modo que as iterações neles executadas sejam concluídas simultaneamente (balanceamento de carga). Assim, quando houver necessidade de troca de mensagens entre os processadores, eles estarão praticamente sincronizados, reduzindo os tempos de espera.

- os tempos de comunicação entre os processadores através da utilização de funções globais da biblioteca de troca de mensagens.

Assim sendo, na primeira fase da estruturação sob a forma de um algoritmo do tipo dados paralelos, a cada nó do hipercubo são atribuídas todas as operações do método do Gradiente Conjugado, bem como as correspondentes partições do conjunto de dados. No caso, para um hipercubo de dimensão d, para $i = 0,1,\ldots,2^d - 1$, ao iésimo nó serão alocadas A_i, x_i, r_i, p_i, q_i, como sendo as iésimas partições de, respectivamente: matriz A e dos vetores x, r, p e q.

A seguir, mostramos como foi efetuada a minimização, tanto do número de ocasiões em que se faz necessária a comunicação, como do tempo utilizado em cada uma dessas comunicações.

Trata-se de identificar as ocasiões nas quais se faz necessária a comunicação, assegurando-se que, em cada nó do hipercubo, seja executado o algoritmo básico com paralelismo de dados, na sua versão síncrona, conforme escrito abaixo, para o i-ésimo processador:

- Inicialmente , para $i = 0,1,\ldots,2^d - 1$

  escolha $x_i(0) = 0$ e faça:

  $r_i(0) = p_i(0) = b_i - A_i x_i(0)$

  Então calcule: $< r_i(0), r_i(0) >$ e $< b, b >$

  A seguir para $k = 0,1,2,\ldots$

- $q_i(k) = A_i p(k)$; (observe que o vetor p está particionado)

- Calcule: $< p(k), q(k) >$ (os vetores p e q estão particionados, 1ª **comunicação**).

- Calcule:

$$\alpha(k) = \frac{< r(k), r(k) >}{< p(k), q(k) >} \tag{8}$$

- $r_i(k+1) = r_i(k) - \alpha(k) q_i(k)$

- $x_i(k+1) = x_i(k) + \alpha(k) p_i(k)$

- Calcule: $< r(k+1), r(k+1) >$ (o vetor r está particionado, 2ª **comunicação**).

- $\beta(k) = \frac{< r(k+1), r(k+1) >}{< r(k), r(k) >}$

- $p_i(k+1) = r_i(k+1) + \alpha(k) p_i(k)$

- Monte $p(k+1)$ (o vetor p está particionado, 3ª **comunicação**).

- Até $(\frac{< r(k+1), r(k+1) >}{< b, b >})^{\frac{1}{2}} < 10^{-5}$

Examinando-se o algoritmo básico acima e levando-se em consideração que cada processador somente possui os dados particionados que a ele foram alocados, há necessidade de comunicação entre os processadores para a passagem dos resultados parciais para o cálculo das variáveis globais (como, por exemplo $<p,q>$ e α), utilizadas no processamento.

As operações em que se faz necessária a comunicação, com passagem de dados parciais, são:

- Calculo de:

$$<p(k),q(k)> = \sum_{i=0}^{2^d-1} <p_i(k),q_i(k)> \quad (9)$$

$$<r(k+1),r(k+1)> = \sum_{i=0}^{2^d-1} <r_i(k+1),r_i(k+1)>$$

ou seja, $< p(k),q(k) >$ vem a ser o somatório dos produtos internos dos vetores particionados $p_i(k)$ e $q_i(k)$ que estão distribuídos entre os i processadores e analogamente $< (r(k+1),r(k+1) >$.

- Montagem do vetor $p(k+1)$ a partir das partições $p_o(k+1), \ldots, p_i(k+1), \ldots, p_{2^d-1}(k+1)$.

Cada uma das etapas acima gera a necessidade de comunicação, tanto para o cômputo do valor global, resultado de um somatório, quanto para a montagem do vetor p a partir dos subvetores p_i. Uma vez identificadas as três etapas de comunicação do algoritmo no restante da seção mostramos a viabilidade de redução do número dessas etapas, de modo a melhorar o ganho de velocidade de processamento. O cálculo dos produtos internos que, na versão original, demanda duas etapas de comunicação (das 3 etapas utilizadas em cada iteração) foi reduzido para apenas uma.

O problema básico é encontrar as expressões para os escalares globais $\alpha(k)$ e $\beta(k)$ em termos de produtos internos, de forma a reduzir o número de etapas de comunicação. O produto interno $<r(k+1),r(k+1)>$, substituindo o segundo termo pela relação de recorrência $r(k+1) = r(k) - \alpha(k)q(k)$, pode ser expresso por:

$$<r(k+1),r(k+1)> = -\alpha(k) <r(k+1),q(k)> \quad (10)$$

onde $<r(fk+1),r(k)>= 0$, já que os resíduos gerados em duas iterações sucessivas do GC são mutuamente ortogonais, (conforme Luenberger [16]).

Por isso, com a aplicação da equação (10), no conjunto de equações (8) temos:

$$\beta(k) = -\frac{\alpha(k) <r(k+1),q(k)>}{<r(k),r(k)>} \quad (11)$$

Com o valor de $\alpha(k)$ das equações (8) aplicado em (11) resulta:

$$\beta(k) = -\frac{<r(k+1),q(k)>}{<p(k),q(k)>} \quad (12)$$

Utilizando a relação de recorrência definida por $r(k+1) = r(k) - \alpha(k)q(k)$, obtem-se:

$$<r(k+1),q(k)> = <r(k),q(k)> -\alpha(k) <q(k),q(k)> \quad (13)$$

ou ainda :

$$<r(k+1),Ap(k)> = <r(k),Ap(k)> -\alpha(k) <Ap(k),Ap(k)> \quad (14)$$

Reescrevendo a relação de recorrência definida por $p(k+1)$ do conjunto de equações (8) resulta:

$$p(k+1) = r(k+1) + \beta(k)p(k)$$

e consequentemente:

$$p(k) = r(k) + \beta(k-1)p(k-1) \tag{15}$$

Tirando-se o valor de $r(k)$ da equação (15) temos:

$$r(k) = p(k) - \beta(k-1)p(k-1) \tag{16}$$

Desse modo, pode-se determinar $< r(k), Ap(k) >$, utilizando a equação (16) ou seja:

$$\begin{aligned} < r(k), Ap(k) > &= < p(k) - \beta(k-1)p(k-1), Ap(k) > \\ &= < p(k), q(k) > \end{aligned} \tag{17}$$

Observe-se que

$$< p(k-1), Ap(k) > = 0 \tag{18}$$

desde que os vetores de direção gerados durante o algoritmo GC sejam ortogonais em A (Luenberger [16]).

Inserindo a equação (17) em (13), resulta:

$$< r(k+1), q(k) > = < p(k), q(k) > - \alpha(k) < q(k), q(k) > \tag{19}$$

Aplicando a equação (19) em (12) temos :

$$\beta(k) = \alpha(k) \times \frac{< q(k), q(k) >}{< p(k), q(k) >} - 1 \tag{20}$$

Portanto, a partir dos produtos internos $< r(k), r(k) >$, $< q(k), q(k) >$ pode-se determinar os escalares globais $\alpha(k)$ e $\beta(k)$, a partir do conjunto das equações (8) e (20) respectivamente.

Dessa forma, a partir da equação (20) pode-se reestruturar o algoritmo paralelo GC básico de modo a reduzir de três para duas as etapas de comunicação em cada iteração. Segue-se abaixo a nova sequência de cálculos:

- Escolha $x(0)$, faça $r(0) = p(0) = b - Ax(0)$ e calcule $< r(0), r(0) >$.
  Então para $k = 0,1,2,\ldots$:
- Calcule $q_i(k) = A_i p(k)$;
- Calcule $< p_i(k), q_i(k) >$ e $< q_i(k), q_i(k) >$
- Calcule (via comunicação entre processadores):

$$\begin{aligned} < r(k), r(k) > &= \sum_{i=0}^{2^d-1} < r_i(k), r_i(k) > \\ < p(k), q(k) > &= \sum_{i=0}^{2^d-1} < p_i(k), q_i(k) > \\ < q(k), q(k) > &= \sum_{i=0}^{2^d-1} < q_i(k), q_i(k) > \end{aligned} \tag{21}$$

- Calcule $\alpha(k) = \frac{<r(k), r(k)>}{<p(k), q(k)>}$;
- Calcule $\beta(k) = \frac{\alpha(k) <q(k), q(k)>}{<p(k), q(k)>} - 1$

  $<r(k+1), r(k+1)> = \beta(k) <r(k), r(k)>$

  $r_i(k+1) = r_i(k) - \alpha(k) q_i(k)$

  $x_i(k+1) = x_i(k) + \alpha(k) p_i(k)$

  $p_i(k+1) = r_i(k+1) + \beta(k) p_i(k)$
- Atualize $p(k+1)$, pela concatenação dos subvetores $p_i(k+1)$, via troca de mensagens.
- Até:

$$(\frac{<r(k+1), r(k+1)>}{<b, b>})^{\frac{1}{2}} < \epsilon$$

Como se pode verificar das equações (21), o número de etapas de comunicação foi reduzido de três para duas.

Cabe ainda, observar que o volume de comunicação não é alterado, desde que os três valores, correspondentes aos produtos internos, sejam acumulados numa única etapa de comunicação nas equações (21). O acréscimo de carga computacional por iteração, após a estruturação proposta é de somente uma multiplicação escalar e uma subtração escalar.

3 A COMUNICAÇÃO SÍNCRONA E ASSÍNCRONA NO iPSC/860

A troca de mensagens é o meio de comunicação entre os diferentes nós do Sistema iPSC/860. Cada processador tem sua memória exclusiva, requerendo a explícita transferência dos dados. Quando a informação armazenada em um dado nó é necessária em outro processador, o nó de origem deve transmitir a informação e o nó que a necessita, deve estar pronto para recebê-la.

Em geral, o tempo necessário para transmitir as mensagens é dependente da velocidade do sistema de comunicação da máquina.

Cada um dos nós tem um módulo de comunicação separado do processador, que utiliza um hardware de chaveamento especial. Nenhum tempo de processamento é utilizado nos processadores intermediários durante a troca de mensagens. Uma vez que o circuito de conexão é estabelecido, a mensagem é transmitida numa velocidade de 2,8 Megabytes por segundo.

As trocas de mensagens podem ser síncronas ou assíncronas. e podem ter qualquer comprimento, mas as mensagens entre o hospedeiro e um dado nó não podem exceder 256 Kbytes.

3.1 Manipulação de Mensagens Síncronas

As rotinas de troca de mensagens síncronas incluem o bloqueio das transmissões e recepções. Quando um processo emite um pedido de recepção de mensagem, deixa de executar instruções (permanece bloqueado) até que uma mensagem do tipo esperado seja recebida. Se a mensagem não chega ao nó, no momento em que o pedido é feito, o processo não realiza nenhuma tarefa até que essa mensagem chegue. Se a mensagem chega ao nó antes que o processo tenha efetuado o pedido, o processo espera até que a mensagem seja copiada para a memória do processo, a partir do "buffer" do sistema no qual foi temporariamente armazenada.

Similarmente, quando um processo faz um pedido de transmissão de uma mensagem sincronamente, o processo é bloqueado até que a mensagem seja copiada, pelo sistema operacional, da memória do processo transmissor para o sistema de transferencia de mensagens.

As transferencias síncronas de mensagens são essenciais para a correta implementação de uma grande classe de algoritmos, e são também úteis para o desenvolvimento inicial e depuração de aplicações que serão, eventualmente, utilizadas sob a forma de transferencia assíncrona.

3.2 Manipulação de Mensagens Assíncronas

No contexto da comunicação assíncrona; as recepções assíncronas permitem aos processos alertarem o sistema operacional de que certas mensagens estão sendo aguardadas e deverão ser distribuídas a esses processos tão logo cheguem, mesmo que o processo esteja ocupado realizando outras tarefas naquele momento. O processo não aguarda ociosamente pela mensagem que deve chegar, pelo contrário, continua a executar instruções até que a informação existente na mensagem seja solicitada. Então, o processo deve verificar se a mensagem já foi distribuída; em caso afirmativo, o processo pode utilizá-la imediatamente; caso contrário, o processo, nesta ocasião, utiliza o valor mais atualizado existente.

A transmissão ou recepção assíncronas não bloqueiam o processo. O envio assíncrono é feito através da função" isend()", e a recepção assíncrona é efetuada com " irecv()". Para se utilizar a comunicação assíncrona não se pode utilizar as operações globais do sistema iPSC, como: "gcol()", "gdsum()", etc, devido ao seu carater síncrono. Deve-se utilizar as instruções " isend()" e "recv()", além de designar os nós de destino utilizando códigos de Gray (conforme proprõe Chen [8]) de modo a reduzir os tempos de comunicação entre os nós.

4 IMPLEMENTAÇÃO DA VERSÃO ASSÍNCRONA

Na implementação da versão síncrona do Método do Gradiente Conjugado o tempo consumido em comunicação foi maior em relação ao tempo de processamento paralelo, na medida em que foi aumentado o número de processadores com a dimensão do problema mantida fixa. Esta foi uma das razões que motivou a implementação da versão assíncrona, buscando reduzir o sobre-custo de comunicação.

A principal vantagem na utilização das versões assíncronas dos algoritmos decorre justamente da redução da necessidade de sincronização, com a redução do sobre-custo de tempo devido a esperas ociosas. Em contrapartida aparecem restrições mais fortes nas condições de convergência do algoritmo como mencionado no trabalho [17], e uma maior dificuldade na detecção da terminação, além de uma maior dificuldade de implementação devido à complexidade do gerenciamento das mensagens pendentes, assim como na fase de depuração do programa.

Na versão assíncrona o algorítmo do Gradiente Conjugado, procedeu-se as alterações na versão síncrona, definida nas equações (21), chegando-se ao algoritmo descrito abaixo. As operações foram escritas para o i-ésimo processador:

- Escolha $x(0)$, faça $r(0) = p(0) = b - Ax(0)$ e calcule $< r(0), r(0) >$.
  Então para $k = 0,1,2,\ldots$:
- Calcule $q_i(k) = A_i p(k)$;
- Calcule (através da comunicação assíncrona entre processadores):

$$
\begin{aligned}
< r(k), r(k) >_{(i)} &= \sum_{i=0}^{2^d-1} < r_i(k - e_i^j(k)), r_i(k - e_i^j(k)) > \\
< p(k), q(k) >_{(i)} &= \sum_{i=0}^{2^d-1} < p_i(k - e_i^j(k)), q_i(k - e_i^j(k)) > \\
< q(k), q(k) >_{(i)} &= \sum_{i=0}^{2^d-1} < q(k - e_i^j(k)), q(k - e_i^j(k)) > \qquad (22)
\end{aligned}
$$

- $\alpha(k)_{(i)} = \frac{<r(k), r(k)>_{(i)}}{<p(k), q(k)>_{(i)}}$;
- $\beta(k)_{(i)} = \frac{\alpha(k) <q(k), q(k)>_{(i)}}{<p(k), q(k)>_{(i)}} - 1$

  $<r(k+1), r(k+1)>_{(i)} = \beta(k)_{(i)} <r(k), r(k)>_{(i)}$

  $r_i(k+1) = r_i(k) - \alpha(k) q_i(k)$

  $x_i(k+1) = x_i(k) + \alpha(k) p_i(k)$

  $p_i(k+1) = r_i(k+1) + \beta(k) p_i(k)$
- Atualize $p(k+1)_{(j)}$, através a comunicação assíncrona entre os processadores e passagem dos subvetores $p_i(k+1)$, que foram calculados em cada nó.
- Até:

$$(\frac{<r(k+1), r(k+1)>}{<b, b>})^{\frac{1}{2}} < \epsilon \qquad (23)$$

Observação 1: $e_i^j(k)$ corresponde ao retardo determinado pelo número de iterações em atraso (com referencia ao processador mais rápido) existente no i-ésimo processador em sua k-ésima iteração que recebe a informação do jésimo processador; normalmente, é utilizado o último valor disponível em cada processador.

Observação 2: para o cálculo dos produtos internos $<r(k), r(k)>_{(i)}$, $<p(k), q(k)>_{(i)}$ e $<q(k), q(k)>_{(i)}$, há a necessidade de comunicação entre os processadores para, através a soma desses resultados parciais obter-se as variáveis globais para o nó i. Exemplificando, $< p_j(k - e_i^j(k)), q_j(k - e_i^j(k)) >$ representa o produto interno dos subvetores particionados (p_j e q_j) que estão distribuídos pelos j-ésimos processadores e que são utilizados na k-ésima iteração do i-ésimo nó, e que correspondem a valores de iterações com atraso igual a $e_i^j(k) \in \{0, 1, \ldots, l\}$ para algum número inteiro $l > 0$.

Observação 3: as partições $p_0(k+1), \ldots, p_i(k+i), \ldots, p_{2^d-1}(k+1)$, que se encontram distribuídas entre os processadores são passadas, através de mensagens aos nós vizinhos, e em seguida ordenadas para atualização do vetor global $p(k+1)_{(i)}$.

Assim, em duas ocasiões, nesse algoritmo (a cada iteração) são efetuadas as comunicações assíncronas entre os processadores, utilizando-se o último resultado atualizado disponível em cada processador.

Na versão assíncrona os produtos escalares $<r, r>$, $<p, q>$ e $<q, q>$ são determinados da seguinte forma:

- alocação das partições da matriz A e dos vetores r, p e q aos nós, utilizando códigos de Gray;
- utilização das funções de comunicação isend(), irecv() em cada etapa de comunicação ao invés da função global gopf();
- a primeira iteração é efetuada com transmissão e recepção de mensagens entre os nós, aguardando-se a conclusão através msgwait();
- na segunda iteração posta-se a recepção e efetua-se a transmissão, mas utiliza-se na operação de soma o resultado que é passado anteriormente, sem aguardar a mensagem do nó vizinho, de modo a evitar tempo de espera;
- na terceira e demais iterações verifica-se se a mensagem foi recebida, através de msgdone(), em caso afirmativo ela é utilizada, caso contrário, prossegue-se utilizando o último resultado disponível, de modo a evitar as perdas de tempo enquanto se aguarda a atualização.

Na última etapa do algoritmo em que se faz a atualização do vetor $p(k+1)$, ela é realizada da seguinte maneira:

- efetua-se a primeira iteração com irecv(), isend() e msgwait(), aguardando-se a recepção das partições do vetor $p(k)$ existentes em cada nó, concatenando-se de modo a obter o vetor $p(k)$ atualizado.
- na segunda iteração posta-se a recepção com irecv(), efetua-se a transmissão com isend(), mas não se aguarda a recepção; na concatenação utiliza-se os resultados anteriores;
- na terceira e demais iterações verifica-se através msgdone() se a mensagem do nó adjacente chegou, em caso afirmativo utiliza-se este resultado para a concatenação e obtenção do vetor $p(k)$, caso contrário utiliza-se o último resultado recebido do nó adjacente para efetuar a concatenação; nestas iterações também é verificado, através msgdone() se foi concluída com êxito a comunicação, em caso positivo efetua-se isend(), em nenhum caso, há perda de tempo aguardando recepção ou transmissão;

O processamento é interrompido naquele processador em que é constatado um erro menor do que 10^{-5}, sendo que este mesmo nó fica responsável pela interrupção dos processos dos nós vizinhos bem como pela transmissão do resultado final ao hospedeiro.

5 RESULTADOS COMPUTACIONAIS

A fim de avaliar o desempenho da versão paralela síncrona do método do Gradiente Conjugado (escalado pela diagonal principal) no iPSC/860, foi utilizado inicialmente, para teste do algoritmo um problema de cálculo estrutural.

Trata-se de um sistema de equações matriciais oriundas de utilização do método de elementos finitos na análise de tensões em uma estrutura anelar, com 1290 graus de liberdade. Este problema está associado à solução de um sistema $Ax = b$, onde A é uma matriz simétrica A(1290 x 1290), esparsa, com uma largura de banda de 359 elementos. Sendo essa matriz esparsa A armazenada em duas outras: (a_compac) e (n_col). A primeira (a_compac), contem apenas os elementos não nulos, armazenados em linhas correspondentes de A, a segunda (n_col), contem o número das colunas que relacionam, entre si, os elementos de (a_compac) e de A. Desta forma, economiza-se o espaço de memória local em cada processador.

Na tabela (V.1) são apresentados os tempos de processamento em um VAX-11/785, do modelo de elementos finitos dessa estrutura anelar [4], são mostrados os tempos (em segundos) correspondentes à utilização de três rotinas de métodos diretos, (Skyline, Frontal e Esparso) da solução multi-frontal do sistema de equações.

	SKYLINE	FRONTAL	ESPARSO
TEMPOS (seg)	62,00	67,68	65,48

Tabela V.1 Tempos de processamento em um VAX-11/785 (segundos)

Na avaliação foram utilizados dois índices de desempenho: ganho de velocidade e eficiência no paralelismo [19]. O ganho de velocidade (speed-up) aqui utilizado é definido como a relação entre o tempo consumido quando da execução em um único processador, e o tempo gasto na execução em n processadores. A eficiência do paralelismo é definida como a relação entre o speed-up e o número de processadores.

Na análise do desempenho do método do Gradiente Conjugado Paralelo implementado no iPSC/860, foram obtidos os seguintes tempos:

Processadores	Tempos (seg)	N^o de iterações	Speed-up	Eficiência
1	24,369	1167	1,0	100,00%
2	17,117	1167	1,423	71,15%
4	13,754	1167	1,771	44,27%
8	13,527	1167	1,801	22,51%

Tabela V.2 Tempos, iterações, speed-up e eficiências

5.1 Análise dos resultados

O tempo de processamento numa topologia hipercúbica, na versão síncrona, pode ser expresso da seguinte forma [5]:

$$T_{pp} = \frac{T_{ps}}{N_{proc}} + T_{com} + T_{bloqueio} \tag{24}$$

sendo,

T_{pp} := Tempo de processamento paralelo
T_{ps} := Tempo de processamento sequencial
N_{proc} := Número de processadores
T_{com} := Tempo de comunicação entre os processadores
$T_{bloqueio}$:= Tempo de bloqueio para sincronização de processadores
$\frac{T_{ps}}{N_{proc}}$:= Tempo de processamento se a comunicação fosse instantânea e sem bloqueio

Em muitos algoritmos paralelos que utilizam comunicação intensa, como é o caso do Gradiente Conjugado paralelo, o tempo consumido na troca de mensagens entre os processadores representa uma fração significativa do tempo total necessário para a resolução do problema. Dessa forma, pode-se admitir, neste caso, um sobre-custo devido a atrasos de comunicação, que pode ser expresso através da relação:

$$C_{atraso} := \frac{T_{pp} - \frac{T_{ps}}{N_{proc}}}{T_{pp}} \tag{25}$$

que representa a relação entre o tempo consumido em comunicação entre os processadores mais o tempo de bloqueio, e o tempo de processamento paralelo total. Este sobre-custo ou atraso, a partir dos dados obtidos para o GC está tabulado abaixo:

Processadores	C_{atraso}
2	28,81%
4	55,70%
8	77,48%

Tabela V.3 Percentual consumido em comunicação

Assim, o tempo consumido em comunicação é considerável e proporcionalmente mais elevado em relação ao tempo de processamento, quando a dimensão do hipercubo aumenta e a dimensão do problema permanece fixa. Assim sendo, a versão do algoritmo do Gradiente Conjugado (GC), na sua versão síncrona, mostra-se mais adequado quando o número de incógnitas é elevado (excede mil variáveis). Sendo que para o caso de mil incógnitas, para o iPSC/860 com 8 processadores, o ganho de velocidade (speed-up) ainda é inferior a 2. Para explorar essaa característica de melhores

eficiências, estão sendo avaliados (em fase de testes) sistemas de dimensão mais elevada, bem como esquemas alternativos de implementação do GC paralelo.

A implementação preliminar da versão assíncrona, acima descrita, utilizada para resolver o mesmo exemplo considerado na aplicação síncrona, apresentou problemas de convergência. O problema ocorre, quando a partir da centésima iteração, o processo entra em um ciclo oscilatório quando o erro, atinge a ordem de 10^{-2}, impedindo dessa forma que se chegue a uma solução com a precisão desejada. Todavia a velocidade de convergência, antes que esse regime oscilatório fosse atingido, encoraja implementações alternativas do esquema assíncrono com diferentes estratégias de controle de troca de mensagens, que atualmente estão em investigação.

Agradecimentos:

Ao Prof. Álvaro L. G. A. Coutinho, do Grupo de Computação Paralela da COPPE/UFRJ pelos comentários e sugestões relativos a este trabalho.

Referências

[1] Bertsekas, D.P., Tsitsiklis, J.N.: 'Parallel and Distribuited Computation: Numerical Methods'. Prentice-Hall Inc., Englewood, N.Y., 1989.

[2] Fox, G.C., Johnson, M.A., Lyzenga, G.A., Otto, S.W., Salmon, J.K., Walker, D.W.: 'Solving Problems on Concurrent Processors'. Prentice-Hall, Englewood Cliffs, N.J., 1988.

[3] Gustafson, J.L., Montry, G.R, Benner, R.E.: 'Development of Parallel Methods for a 1024-Processor Hypercube'. Siam J. Sci. Stat. Comput., vol. 9, no 4, jul 1988.

[4] Justino, M.R.F.:'Solução Multi-frontal de Sistemas de Equações Algébricas Lineares Esparsas Simétricas Esparsas', Seminário de Doutorado do Programa de Engenharia Civil – COPPE/UFRJ, 19 de dezembro de 1990.

[5] Cabral, R.G.: 'Avaliação do Desempenho do Método dos Gradientes Conjugados em Multiprocessadores com Arquitetura Hipercúbica'. Tese de Mestrado COPPE/UFRJ. Março de 1991, Rio de Janeiro.

[6] George A. e Liu,J.W.H.: 'Computer Solution of Large Sparse Positive Definite Systems. Prentice-Hall, Inc. Engewood Cliffs, New Jersey 1981.

[7] Chan, T.F. e Saad, Y.: 'Multigrid Algorithms on the Hipercube Multiprocessor'. IEEE Transactions on Computers, vol-35, no 11, novembro de 1986.

[8] Chen, M.S., Shin, K.G.:'Processor Allocation in an N-Cube Multiprocessor Using Gray Codes'. IEEE Trans. on Computers, vol 36, no 12, dez 1987.

[9] Abe, G.B. e Hane, K.: 'The Preconditioned Conjugate Gradient Method on the Hypercube'. The Third Conference on Hypercube Concurrent Computers and Applications, Pasadena, California, jan 1988.

[10] Van Der Vorst, H.K., e Dekker: 'Conjugate Gradiente Type Methods and Preconditioning'. Journal of Computational and Mathematics 24, pp 73-87, North-Holland, novembro de 1988.

[11] Golub, G.H. e Van Loan, C.F.: 'Matrix Computations'. The Johns Hopkins Universty Press, Baltimore, Maryland 1983.

[12] Aykanat, C., Ozguner, F., Ercal, F., Sadayapan, P.: 'Iterative Algorithms for Solution of Large Sparse Systems of Linear Equations on Hypercubes'. IEEE Transactions on Computers, vol-37, no 12, dezembro 1988.

[13] Ortega, J.M.: 'Introduction to Parallel and Vector Solution of Linear Systems'. Plenum Press, New York and London, 1988.

[14] Hillis, W.D., Steele, G.L.Jr.:'Data Parallel Algorithms'. Communications of the ACM, vol 29, no 12, dez 1986, pg 1170-1183.

[15] Ni, L.M., King, C.T.:'On Partitioning and Mapping for Hypercube Computing'. International Journal of Parallel Programming, vol 17, no 5, pg 475-495, 1988.

[16] Luenberger,D. : "Introdution to Linear and Nonlinear Programming". Reading , MA : Addison Wesley, 1973.

[17] Kaszkurewicz,E., Bhaya,A. e Siljak, D.D.: 'On the Convergence of Parallel Asynchronous Block-Iterative Computations'. Linear Algebra and Its Applications pg 139-160. Elsevier Science Publishing Co., Inc., 1990.

[18] Decker, I.C., Falcão, D.M., Kaszkurewicz, E.: 'Parallel Implementation of a Power System Simulation Methodology using the Conjugate Gradient Method'. IEEE-Transactions on Power System Computation Conference, pp 509-519, 1991.

[19] Eager, D.L., Zahorjan, J., Lazowska, E.D.: 'Speedup Versus Efficiency in Parallel Systems'. IEEE-Transactions on Computers, vol 38, no 3, mar 1989.