UFRPE

(Universidade Federal Rural de Pernambuco)

ALUNO: *D'Emery, Richarlyson Alves* *MAT.: 20011250223*

Filho, Pedro Henrique de Souza *MAT.: 20022250116*

PROF.º: *Jones Albuquerque*

CURSO: *Licenciatura em Computação*

DISCIPLINA: *Programação Paralela e Distribuída*

Projeto: ***Análise do Algoritmo***

Matrix Multiplication (Hypercube SIMD)

2º semestre /23.08.2004

Recife – PE

Multiplicação de Matriz no Model Hipercubo SIMD

Richarlyson Alves D'Emery[1] , aluno de Licenciatura em Computação –UFRPE
Pedro Henrique de Souza Filho[2] , aluno de Licenciatura em Computação –UFRPE[3]

Resumo: Este trabalho visa fazer um estudo sobre algoritmos que resolvem o problema da multiplicação de matrizes, face as diversas possibilidades de algoritmos existentes, descreveremos a aplicação que utiliza o modelo hipercubo SIMD presente na computação paralela, decorrendo com análises e eventuais simulações, assim como, um paralelo com outros algoritmos que tentam resolver o problema visando um menor tempo computacional. Também descreve acerca da computação paralela distribuída, sugere leituras para estudo e sugestões para futuros trabalhos.

Palavras-chave:.Algoritmo, multiplicação de matrizes, hipercubo SIMD, computação paralela e distribuída.

Abstract:. This paper aims to make a study on algorithms that solve problem of the multiplication of matrices, facing the diverse possibilities of existing algorithms, we will describe the application that uses the hypercube SIMD model present in the parallel computation, elapsing with analyses and eventual simulations, as well as, a parallel with other algorithms that try to solve the problem aiming at a less computational time. Also, it describes concerning the distributed parallel computation, suggests readings for study and suggestions for future works.

Key-Words:. Algorithm, matrix multiplication, hypercube SIMD, parallel and distributed computation.

1. Introdução

Este trabalho tem por objetivo descrever o problema de multiplicação de matrizes, onde diante das soluções seqüenciais fizemos um estudo de uma solução utilizando paralelismo de dados, além de realizar uma pequena análise acerca do algoritmo proposto.

Em computação, de forma geral, deseja-se, no processamento de dados, obter resultados de maneira rápida e eficiente, e a multiplicação de matrizes esta presente em cálculos científicos e educacionais.

[1] E-mail de contato: rico_demery@yahoo.com.br
[2] E-mail de contato: phsfilho@bol.com.br
[3] Universidade Federal Rural de Pernambuco, CEP: 52171-900, Dois Irmãos, Recife – PE, Brasil.

A complexidade para realizar a multiplicação de duas matrizes utilizando um algoritmo seqüencial é de $O(n^3)$, como pode ser visto no algoritmo abaixo:

```
...
for (i = 0; i < n; i++){
        for (j = i; j < n; j++){
                c [i][j] = valor;
                for (k = 0; k < n; k++){
                        c[i][j] = c[i][j] + a[i][k] * b[k][j] ;
                }
        }
}
...
```

O algoritmo acima possui três estruturas, o for, e cada um com complexidade O(n), como os três estão um dentro do outro, e ignorando os custos de atribuição, logo temos n x n x n, ou seja, sua complexidade é $O(n^3)$. Esta análise ignora diversos fatores como, por exemplo, a tecnologia de processamento da CPU.

Algumas soluções são possíveis para otimizar o algoritmo acima, mas ainda assim a solução seria seqüência, e a complexidade desses novos algoritmos não sofreriam grandes ganhos, como levar a solução as ser logarítmica, por exemplo e, ao nosso ver, não seriam de grande relevância.

Algumas soluções são proposta para otimizar o algoritmo seqüencial acima, mas tais soluções utilizam o paralelismo de dados, como por exemplo, paralelizar um dos loops "for", onde pode-se notar que os dois loops externos não são dependentes de nenhuma outra iteração, podendo executar o loop interno em paralelo. Daí se utilizássemos uma matriz n x n e n processadores a complexidade do problema passaria a ser $O(n^2)$. E ainda sim poderíamos obter, com a paralelização, resultados ainda melhores, por exemplo, utilizando n^2 processadores a complexidade do problema passaria a ser O(n) e com n^3 processadores (paralelizando o loop interno) obtêm-se uma complexidade de O(log n), a qual segundo Dan I. Moldovan[4] seria a menor solução possível para o problema.

Não sendo o nosso propósito fazer um estudo de possíveis soluções seqüenciais, e simplesmente fazer referencias a soluções utilizando o paralelismo de dados, analisaremos uma implementação, ao nosso ver, pouco estudada, utilizando uma estrutura do paralelismo de dados, a estrutura de um hipercubo.

2. Computação Paralela Distribuída

Com a computação paralela podemos ter um alto desempenho para aplicações específicas. Mas há o fator custo, onde inviabiliza a aquisição de equipamentos cuja arquitetura seja paralela, assim como, a implantação desses sistemas que dificultam a sua disseminação de diversas formas.

Diante desta problemática surge os sistemas distribuídos, que possibilita o compartilhamento de hardware fisicamente separados. Logo têm-se a solução para o problema mencionado, ou seja, o baixo custo

[4] Dan I. Moldovan. "Parallel Processing from Applications to Systems - 1993

oferecidos pelos sistemas distribuídos e o alto desempenho fornecido pelo processamento paralelo. Na Universidade Federal Rural de Pernambuco, tivemos a oportunidade de experimentar, o estudo da computação paralela distribuída, na disciplina Programação Paralela e distribuída.

Hoje, com essa nova tecnologia, possuímos ambientes que viabilizam tal idéia. Tais ambientes, conhecidos como ambientes paralelos virtuais, são ambientes de passagem de mensagens que permitem aos sistemas distribuídos desempenharem a função de uma máquina paralela (virtual).

3. Multiplicação de Matrizes no Modelo Hipercubo SIMD

Segundo o teorema de Dekel[5], Dado o modelo hipercubo SIMD com $n^3 = 2^{3q}$ processadores, duas matrizes n x n pode ser multiplicado num tempo $\Theta(\log n)$.

Demonstração: A chave do algoritmo de Dekel et al. (1981) é a estratégia de distribuição de dados;

$5q = 5 \log n$ passos de distribuição são suficientes para transmitir os valores iniciais pela lista de processadores e combinar os resultados.

Os elementos processados deverão ser tidos num preenchimento de uma treliça n x n x n. Processador P (x), onde $0 \leq x \leq 2^{3q} - 1$, com a, b, c, s e t armazenados numa memória local.

Quando o algoritmo paralelo inicia sua execução, os elementos da matriz $a_{i,j}$ e $b_{i,j,}$ para $0 \leq i, j \leq n - 1$, são armazenados nas variáveis a e b do processador $P(2^q i + j)$. Depois do algoritmo paralelo está completo, os elementos da matriz $c_{i,j}$, para $0 \leq i, j \leq n - 1$, deverão ser armazenados na variável c do processador $P(2^q i + j)$.

O algoritmo proposto é:

MATRIX MULTIPLICATION (Hypercube SIMD)

```
Parameter q       (Matrix size is 2^q x 2^q)
Global ℓ
Local a, b, c, s, t

begin
      (Phase 1: Broadcast matrices A and B)

      for ℓ ← 3q – 1 downto 2q do
            for all P_m, where BIT (m, ℓ)=1 do
                  t ← BIT.COMPLEMENT (m, ℓ)
                  a ⇐ [t]a
                  b ⇐ [t]b
            endfor
      endfor
      for ℓ ← q – 1 downto 0 do
            for all P_m, where BIT (m, ℓ) ≠ BIT(m, 2q + ℓ) do
                  t ← BIT.COMPLEMENT (m, ℓ)
                  a ⇐ [t]a
```

[5] Dekel et al. 1978

```
        endfor
endfor
for ℓ ← 2q – 1 downto q do
        for all P_m, where BIT (m, ℓ) ≠ BIT(m, q + ℓ) do
                t ← BIT.COMPLEMENT (m, ℓ)
                b ⇐ [t]b
        endfor
endfor
[Phase 2: Do the multiplications in parallel]
for all P_m do
        c ← a x b
endfor
[Phase 3: Sum the products]
for ℓ ← 2q to 3q -1 do
        for all P_m do
                t ← BIT.COMPLEMENT (m, ℓ)
                s ⇐ [t]c
                c ← c + s
        endfor
endfor
end
```

O algoritmo apresentado possui três distintas fases. Durante a primeira fase os elementos ai,j e bi,j precisarão ser distribuídos para o resto dos processadores. Depois deste loop,

$$\left.\begin{array}{l}[2^{2q}k + 2^{q}i + j]a = a_{i,j}\\ [2^{2q}k + 2^{q}i + j]b = b_{i,j}\end{array}\right\} \text{ para } 0 \le k \le n - 1$$

Depois do segundo loop,

$$[2^{2q}k + 2^{q}i + j]a = a_{i,k} \qquad \text{para } 0 \le j \le n - 1$$

Depois do terceiro loop,

$$[2^{2q}k + 2^{q}i + j]b = b_{k,j} \qquad \text{para } 0 \le i \le n - 1$$

O algoritmo possui n^3 multiplicações para serem executadas e n^3 processamentos de elementos. Como a distribuição é feita na primeira fase, todas as multiplicações $a_{i,k}$ x $b_{k,j}$, podem ser feitas simultaneamente durante a segunda fase. A terceira fase do algoritmo direciona e soma os produtos.

O algoritmo utiliza duas funções a BIT e a BIT.COMPLEMENT. A função BIT, recebe dois argumentos como parâmetros, m e ℓ, e retorna o valor do ℓ-ésimo bit na representação binária de m, ou seja:

Se BIT (m, ℓ) tal que m = 5, ℓ = 2 então BIT (5,2) = **1**

Simulação função BIT: Como m = 5 e na representação binária 5 é igual a 101, e como ℓ = 2 representa o terceiro bit (bit de índice 2) do número binário, então temos:

A função BIT.COMPLEMENT, também recebe dois argumentos como parâmetros, m e ℓ, porém retorna o inteiro formado pelo complemento do valor do bit ℓ na representação binária de m, ou seja:

Se BIT (m, ℓ) tal que m = 5, ℓ = 1 então BIT.COMPLEMENT (5,1) = **7**

Simulação função BIT.COMPLEMENT: Como m = 5 e na representação binária 5 é igual a 101, e como ℓ = 1 representa o segundo bit (bit de índice 1) do número binário, então temos:

7 6 5 4 3 2 1 0
0 0 0 0 0 1 **0** 1

Complemento de 0 é 1, na base 2

7 6 5 4 3 2 1 0
0 0 0 0 0 1 **1** $1_{(2)}$

$= 7_{(10)}$

Agora iremos analisar como multiplicar duas matrizes 2 x 2 numa máquina hipercubo SIMD de oito processadores.

ANÁLISE DO ALGORITMO:

MATRIX MULTIPLICATION (Hypercube SIMD)

Partindo do fato de que a multiplicação a ser realizada seja:

$$\underset{A}{\begin{pmatrix}1 & 2\\ 3 & 4\end{pmatrix}} \times \underset{B}{\begin{pmatrix}-5 & -6\\ 7 & 8\end{pmatrix}} = \underset{C}{\begin{pmatrix}9 & 10\\ 13 & 14\end{pmatrix}}$$

Então temos os seguintes dados:

Matrix size is 2^q x 2^q
Como A e B são 2 x 2 $\Rightarrow$ q = 1

Para a implementação na estrutura a ser utilizada precisaremos de P(x) processadores, onde:

P(x), onde $0 \leq x \leq 2^3q - 1$

Como q = 1, logo teremos 8 processadores, ou seja:

P(x), onde 0 £ x £ $2^3$1 – 1
\ P(x), onde 0 £ x £ 7

E cada processador P(x) tem suas variáveis local a, b, c, s e t.

Quando o algoritmo paralelo inicia sua execução, os elementos da matriz $a_{i,j}$ e $b_{i,j}$, para $0 \leq i, j \leq n-1$, são carregados nas variáveis a e b do processador $P(2^q i + j)$.

Como n = 2, pois a matriz a ser utilizada é do tipo n x n, teremos:

$$j \leq n-1 \therefore j \leq 2-1 \therefore \mathbf{j \leq 1}$$

Como a matriz é n x n, logo o intervalo de i será igual ao de j, ou seja:

$$\mathbf{0 \leq i \leq 1}$$

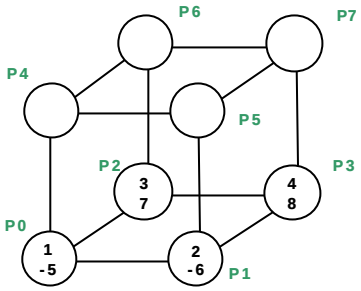
Assim temos:

$P(2^q i + j)$
i=0 j=0 $P(2^0 0 + 0) = P(0)$
i=0 j=1 $P(2^0 0 + 1) = P(1)$
i=1 j=0 $P(2^1 1 + 0) = P(2)$
i=1 j=1 $P(2^1 1 + 1) = P(3)$

Local os elementos das matrizes distribuídos entres os processadores será:

Variáveis Local			
i, j	**P(m)**	**$a_{i,j}$**	**$b_{i,j}$**
i=0 j=0	P(0):	$a_{0,0}$	$b_{0,0}$
i=0 j=1	P(1):	$a_{0,1}$	$b_{0,1}$
i=1 j=0	P(2):	$a_{1,0}$	$b_{1,0}$
i=1 j=1	P(3):	$a_{1,1}$	$b_{1,1}$
	P(4):		
	P(5):		
	P(6):		
	P(7):		

Ou seja:



Agora vamos analisar e simular o algoritmo Matrix Multiplication (Hypercube SIMD):

1) Análise do Algoritmo

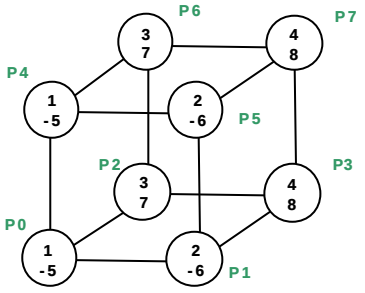
```
Parameter q       (Matrix size is 2^q x 2^q)
Global ℓ
Local a, b, c, s, t

begin
      (Phase 1: Broadcast matrices A and B)

      for ℓ ← 3q – 1 downto 2q do
            for all P_m, where BIT (m, ℓ)=1 do
                  t ← BIT.COMPLEMENT (m, ℓ)
                  a ⇐ [t]a
                  b ⇐ [t]b
            endfor
      endfor
...
***************

***************
Análise 1:

...
      for ℓ ← 2 downto 2 do
            for all P_m, where BIT (m, ℓ)=1 do
                  t ← BIT.COMPLEMENT (m, ℓ)
                  a ⇐ [t]a
                  b ⇐ [t]b
            endfor
      endfor
...
***************

***************
Simulação 1
```

ℓ = 2
m = 0 P0: BIT(0,2) = 0 * //0≠1

```
m=1   P1:
      BIT(1,2) = 0  * //0≠1

m=2   P2:
      BIT(2,2) = 0  * //0≠1

m=3   P3:
      BIT(3,2) = 0  *//0≠1

m=4   P4:
      BIT(4,2) = 0  //1=1
      t ←BIT.COMPLEMENT(4,2) ∴ t←0
      a ⇐ [t]a  ∴ a ⇐ [0]a
      b ⇐ [t]b  ∴ b ⇐ [0]b

m=5   P5:
      BIT(5,2) = 0  //1=1
      t ←BIT.COMPLEMENT(5,2) ∴ t← 1
      a ⇐ [t]a  ∴ a ⇐ [1]a
      b ⇐ [t]b  ∴ b ⇐ [1]b

m=6   P6:
      BIT(6,2) = 0  //1=1
      t ←BIT.COMPLEMENT(6,2) ∴ t← 2
      a ⇐ [t]a  ∴ a ⇐ [2]a
      b ⇐ [t]b  ∴ b ⇐ [2]b

m=7   P7:
      BIT(7,2) = 0  //1=1
      t ←BIT.COMPLEMENT(7,2) ∴ t← 3
      a ⇐ [t]a  ∴ a ⇐ [3]a
      b ⇐ [t]b  ∴ b ⇐ [3]b
```

Análise da Simulação 1:

Os processadores recebem uma cópia do conteúdo de seu processador complementar, ou seja:



****

****

Continuação do Algoritmo

...

for $\ell \leftarrow q - 1$ downto 0 do

for all P_m, where BIT $(m, \ell) \neq$ BIT$(m, 2q + \ell)$ do

t ← BIT.COMPLEMENT (m, ℓ)

a ⇐ [t]a

endfor

endfor

...

****

****

Análise 2

Como q=1 então $\ell \leftarrow 0$

Então analisando o algoritmo:

...

for $\ell \leftarrow 0$ downto 0 do // entrará no laço uma única vez

for all P_m, where BIT (m, 0) ≠ BIT(m, 2) do // só habilitará os Proc. que atenda a condição

t ← BIT.COMPLEMENT (m, ℓ)

a ⇐ [t]a

endfor

endfor

...

****

****

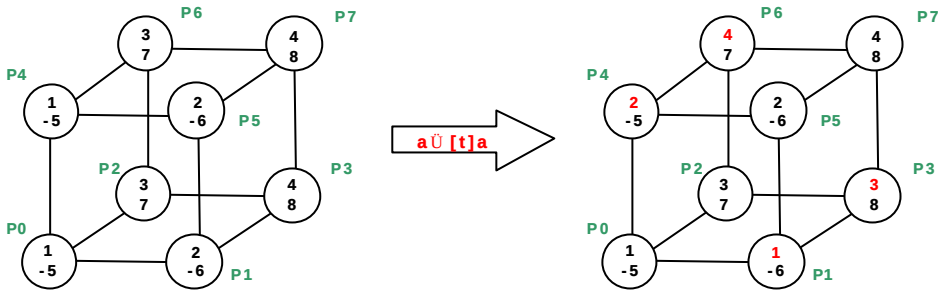
Simulação 2

ℓ = 0

m=0 P0:
BIT(0,0) ≠ BIT (0,2) * //0=0
 	
m=1 P1:
BIT(1,0) ≠ BIT (1,2) //1≠0
t←BIT.COMPLEMENT(1,0) ∴ t← 0
a ⇐ [t]a ∴ a ⇐ [0]a //
 	
m=2 P2:
BIT(2,0) ≠ BIT (2,2) * //0=0
 	
m=3 P3:
BIT(3,0) ≠ BIT (3,2) //1≠0
t←BIT.COMPLEMENT(3,0) ∴ t← 2
a ⇐ [t]a ∴ a ⇐ [2]a //
 	
m=4 P4:
BIT(4,0) ≠ BIT (4,2) //0≠1
t←BIT.COMPLEMENT(4,0) ∴ t← 5
a ⇐ [t]a ∴ a ⇐ [5]a //
 	
m=5 P:5
BIT(5,0) ≠ BIT (5,2) */1=1
 	
m=6 P6:
BIT(6,0) ≠ BIT (6,2) //0≠1
t←BIT.COMPLEMENT(6,0) ∴ t← 7
a ⇐ [t]a ∴ a ⇐ [7]a //
 	
m=7 P:7
BIT(7,0) ≠ BIT (7,2) * //1=1

Análise da Simulação 2:

Os processadores trocam o conteúdo de a, ou seja:



****

****

Continuação do Algoritmo

...

for $\ell \leftarrow 2q - 1$ downto q do

for all P_m, where BIT (m, ℓ) ≠ BIT(m, q + ℓ) do

t ← BIT.COMPLEMENT (m, ℓ)

b ⇐ [t]b

endfor

endfor

...

****

****

Análise 3

Como q=1 então $\ell \leftarrow 1$

Então analisando o algoritmo:

...

for $\ell \leftarrow 1$ downto 1 do // entrará no laço uma única vez

for all P_m, where BIT (m, 1) ≠ BIT(m, 2) do // só habilitará os Proc. que atenda a condição

t ← BIT.COMPLEMENT (m, ℓ)

b ⇐ [t]b

endfor

endfor

...

****

****

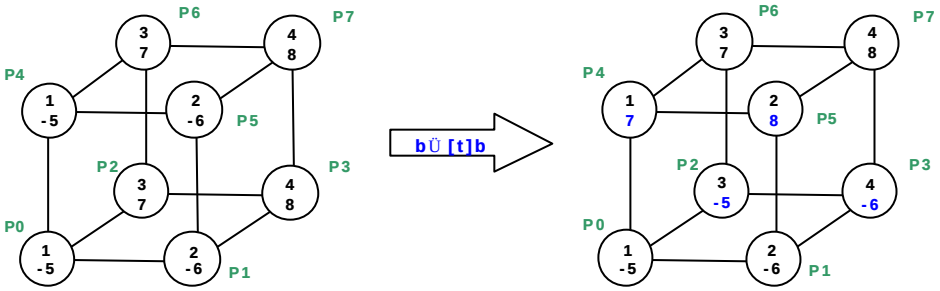
Simulação 3

$\ell = 1$
m = 0 P0: BIT(0,1) ≠ BIT (0,2) * //0=0 m = 1 P1: BIT(1,1) ≠ BIT (1,2) * //0=0 m = 2 P2: BIT(2,1) ≠ BIT (2,2) //1≠0 t ←BIT.COMPLEMENT(2,1) ∴ t ← 0 b ⇐ [t]b ∴ b ⇐ [0]b //

```
m=3   P3:
      BIT(3,1) ≠ BIT (3,2)   //1≠0
      t←BIT.COMPLEMENT(3,1) ∴ t←1
      b ⇐ [t]b  ∴ b ⇐ [1]b //

m=4   P4:
      BIT(4,1) ≠ BIT (4,2)   //0≠1
      t←BIT.COMPLEMENT(4,1) ∴ t←6
      b ⇐ [t]b  ∴ b ⇐ [6]b //

m=5   P:5
      BIT(5,1) ≠ BIT (5,2)   //0≠1
      t←BIT.COMPLEMENT(5,1) ∴ t←7
      b ⇐ [t]b  ∴ b ⇐ [7]b //

m=6   P6:
      BIT(6,1) ≠ BIT (6,2)   * //1=1

m=7   P:7
      BIT(7,1) ≠ BIT (7,2)   *//1=1
```

Análise da Simulação 3:

Os processadores trocam o conteúdo de a, ou seja:

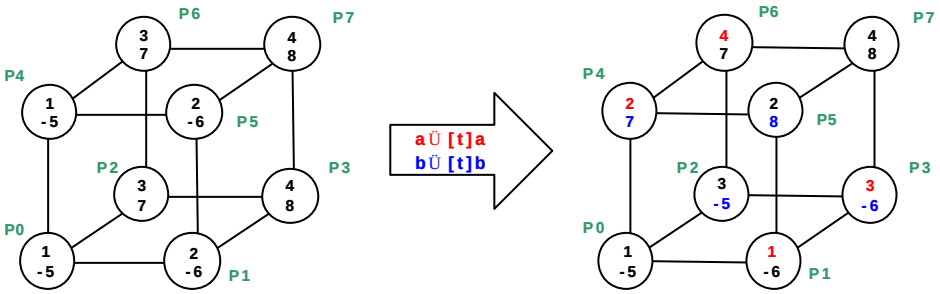


****

****

Análise simultânea das simulações 2 e 3

Após os conteúdos dos processadores terem sido submetidos ao algoritmo o resultado é:



****

****
Continuação do Algoritmo

...
[Phase 2: Do the multiplications in parallel]
for all P_m do
c ← a x b
endfor
...
****

****
Análise 4

Todos os processadores farão a multiplicação do seu conteúdo, ou seja, c recebe o produro de a por b (c= a x b)

****

****
Simulação 4

P_m=0..7

m=0 P0:
c ← a x b ∴ c ←1 x (-5) ∴ c ← -5

m=1 P1:
c ← a x b ∴ c ← 1x (-6) ∴ c ← -6

m=2 P2:
c ← a x b ∴ c ←3 x (-5) ∴ c ← -15

m=3 P3:
c ← a x b ∴ c ←3 x (-6) ∴ c ← -18

m=4 P4:
c ← a x b ∴ c ← 2 x 7 ∴ c ← 14

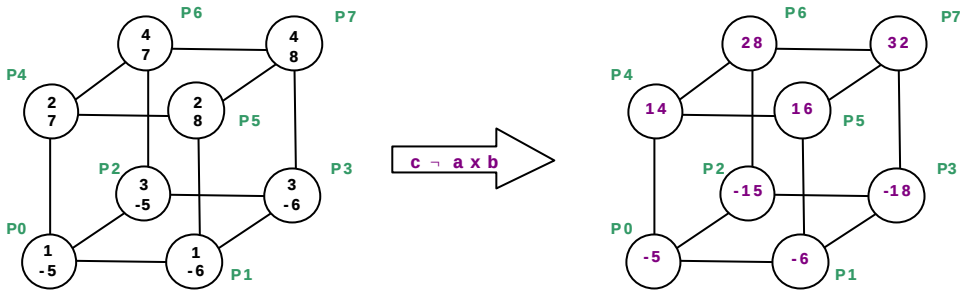
m=5 P:5
c ← a x b ∴ c ← 2 x 8 ∴ c ← 16

m=6 P6:
c ← a x b ∴ c ← 4 x 7 ∴ c ← 28

m=7 P:7
c ← a x b ∴ c ← 4 x 8 ∴ c ← 32

Análise da Simulação 4:

Os processadores substituem seu conteúdo (a e b) pela multiplicação de a por b, ou seja, c:



****

****

Continuação do Algoritmo

...

```
        [Phase 3: Sum the products]
        for ℓ ← 2q to 3q -1 do
                for all Pm do
                        t ← BIT.COMPLEMENT (m, ℓ)
                        s  ⇐ [t]c
                        c ← c + s
                endfor
        endfor
end
```

****

****

Análise 5

Como q=1 então: $\ell \leftarrow 2 \times 1 = 2$ e $\ell \leftarrow 3 \times 1 - 1 = 2$

Então analisando o algoritmo:

...

```
        [Phase 3: Sum the products]
        for ℓ ← 2q to 3q -1 do  // entrará no laço uma única vez
                for all Pm do   // habilita todos os processadores
                        t ← BIT.COMPLEMENT (m, ℓ) // calcula os processadores correspondentes
                        s  ⇐ [t]c // coloca o conteúdo do processador remoto no processadores corrente
                        c ← c + s // soma o conteúdo do processador corrente
                endfor
        endfor
end
```

****

****

Simulação 5

ℓ = 2

m = 0 P0:

t ←BIT.COMPLEMENT(0,2) ∴ t ← 4

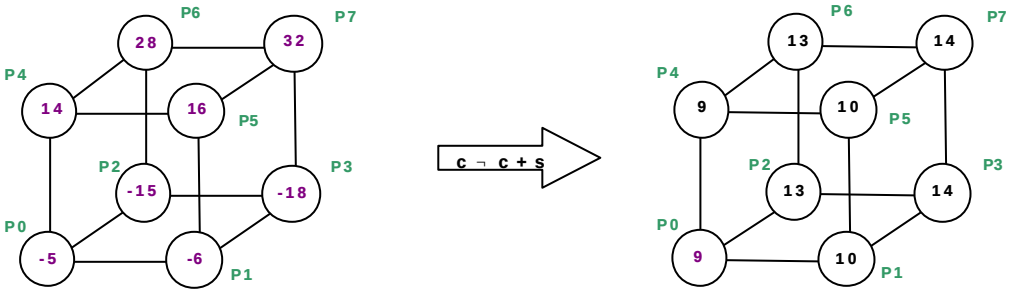
s ⇐ [t]c ∴ s ⇐ [4]c //

c ← c + s ∴ c ← (-5) + 14 ∴ c ← 9

```
m=1  P1:
     t ←BIT.COMPLEMENT(1,2) ∴ t ← 5
     s ⇐ [t]c  ∴ s ⇐ [5]c //
     c ← c + s  ∴ c ← (-6) + 16  ∴ c ←10

m=2  P2:
     t ←BIT.COMPLEMENT(2,2) ∴ t ← 6
     s ⇐ [t]c  ∴ s ⇐ [6]c //
     c ← c + s  ∴ c ← (-15) + 28  ∴ c ← 13

m=3  P3:
     t ←BIT.COMPLEMENT(3,2) ∴ t ← 7
     s ⇐ [t]c  ∴ s ⇐ [7]c //
     c ← c + s  ∴ c ← (-18) + 32  ∴ c ← 14

m=4  P4:
     t ←BIT.COMPLEMENT(4,2) ∴ t ← 0
     s ⇐ [t]c  ∴ s ⇐ [0]c //
     c ← c + s  ∴ c ← (14)+(-5)  ∴ c ← 9

m=5  P:5
     t ←BIT.COMPLEMENT(5,2) ∴ t ← 1
     s ⇐ [t]c  ∴ s ⇐ [1]c //
     c ← c + s  ∴ c ← 16 + (-6)  ∴ c ← 10

m=6  P6:
     t ←BIT.COMPLEMENT(6,2) ∴ t ← 2
     s ⇐ [t]c  ∴ s ⇐ [2]c //
     c ← c + s  ∴ c ← 28 + (-15)  ∴ c ← 13

m=7  P:7
     t ←BIT.COMPLEMENT(7,2) ∴ t ← 3
     s ⇐ [t]c  ∴ s ⇐ [3]c //
     c ← c + s  ∴ c ← 32 +(-18)  ∴ c ← 14
```

Análise da Simulação 5:

Os processadores calculam seus correspondentes no hipercubo e substitui seu valor atual pela soma do seu conteúdo pelo conteúdo do seu correspondente, ou seja:



****

****

Conclusão das Análises e Simulações

Para se calcular duas matrizes do tipo n x n, pode-se obter o resultado utilizando a programação paralela, neste caso utilizamos a estrutura de um

hipercubo, que após análises e simulações de um algoritmo dado, verificou-se que é possível fazer tal multiplicação e obter o resultado na própria estrutura utilizada.

Tal estudo obedece ao seguinte teorema:

***"Dado um modelo de um Hipercubo SIMD com $n^3=2^{3q}$ processadores, duas matrizes n x n pode ser multiplicado num tempo $Q(\log n)$.* (Dekel et al. 1978)"**

4. Conclusões

Após analisado o algoritmo que utiliza o paralelismo de dados, verificamos que no primeiro loop requer 2q passos de distribuição de dados. Cada um dos últimos três loops requer q passos de distribuição de dados. Um total de 5q passos de distribuição de dados são suficientes para multiplicar duas matrizes utilizando o modelo hipercubo SIMD. O algoritmo também possui um passo de multiplicação e q passos de adição. Claramente, a complexidade de multiplicação de matrizes no modelo hipercubo SIMD é $\Theta(q)=\Theta(\log n)$, dado n^3 elementos processados.

Agora fazendo uma comparação com algumas soluções para o problema da multiplicação de matrizes, podemos perceber que o algoritmo proposto por Dekel et al. (1981) tem uma complexidade de tempo $\Theta(\log n)$, assim como propõe Dan I. Moldovan.

E achamos que vale a pena ressaltar o fato de ambos os teóricos utilizarem técnicas diferentes, mas no que diz respeito ao número de processadores, utilizam igual número, ou seja, n^3 processadores para o exemplo proposto.

5. Sugestões para Futuros Estudos

Após discussão do item 2 deste trabalho e estudo dos demais tópicos, proponho a trabalhos futuros a implementação do algoritmo estudado utilizando a biblioteca PVM (máquina paralela virtual), ou até mesmo o MPI (Message Passing Interface), e verificar sua complexidade a nível de implementação.

Ambas as bibliotecas permitem que o usuário escreva programas paralelos usando uma mensagem explícita que passa o modelo

6. Bibliografia Complementar

Para quem deseja utilizar a biblioteca PVM, é fundamental ter à mão o tutorial [Gue94], publicado pelo MIT Press, que também pode ser obtido através da rede em versão html através da URL http://www.netlib.org/pvm3/pvm-book.html. Para os que desejam se aprofundar um pouco mais, [Ben90] faz uma introdução sobre programação distribuída, apresentando outros mecanismos de programação que vão além da troca explícita de mensagens. Uma visão geral sobre os diversos paradigmas existentes, e linguagens de programação distribuída pode ser vista em [Bal93]. Outro aspecto importante em programação distribuída, é a elaboração dos algoritmos, onde a questão da geração das mensagens constitui um fator preponderante na obtenção de alto desempenho [Kum94]. Um grupo de bibliotecas e ferramentas são citados e referenciados por [Duk93], que também fornece um tutorial sobre a área de processamento distribuído baseado em redes de estações.

Uma breve discussão entre o PVM e o MPI pode ser vista em Experiments in Parallel Computing de David Biermann [2000], esta referência pode ser obtida através da rede em versão html através da URL http://www.ee.duke.edu/Academics/Undergraduate/IndStudy00/BiermannGwD/BiermannGwDIntro.html.

7. Referências Bibliográficas

Dan I. Moldovan. "Parallel Processing from Applications to Systems – 1993