A Fast Parallel Algorithm for Special Linear Systems of Equations using Processor Arrays with Reconfigurable Bus Systems

Technical Report B-2-99
January 29, 1999

Rajeev Wankar[1]
N.S.Chaudhari*
Elfriede Fehr

Freie Universität Berlin
Institut für Informatik
{wankar, fehr}@inf.fu-berlin.de
*School of Computer Science, DAVV Indore, India

Abstract

A parallel algorithm using Processor Arrays with Reconfigurable Bus Systems has been designed to solve dense Symmetric Positive Definite (SPD) systems of equations Ax = b. The key content of this report is the parallelisation of the algorithm by Delosme & Ipson [8]. In order to design a parallel algorithm for PARBS, many procedures involved in [8] are handled in a slightly different way. The parallel time and processor's complexity of each step of the algorithm is calculated. The parallel time complexity is O(n) using $2n \times 2n \times 5n$ number of Processing Elements.

Introduction: There are many direct and iterative methods for solving system of equations $Ax = b$ with $A_{n \times n}$ SPD matrix [13,14,25,28,30]. One of the numerically stable approaches to solve a system of equations is LU decomposition[13]. Besides factorization this method involves a forward elimination and a back substitution process which cannot be parallelised easily. In order to construct a parallel algorithm for this problem, we need to investigate a method which does not involve these steps. One such method was proposed by Delosme and Ipson. Other than these two parallel steps the algorithm presented in [8] includes many steps which can be exploited for parallelism to reduce the complexity of the algorithm. Such work has been carried out and a parallel algorithm based on the PRAM model of computation has been published in [38]. Implementing this method on processor arrays with a reconfigurable bus system requires a special technique with modifications to the existing algorithm. In Section 3 this implementation has been discussed. The report is designed in the following way: In Sec tion 1 we give some important results from [8], in Section 2 we present a brief description of the architecture used for designing the problem. In Section 3 we give a PARBS solution of the problem and discuss the time complexity of each of the steps.

1. This work is supported by the German Academic Exchange Services (DAAD) under the "Sandwich Model" fellowship with the author who is permanently working in the Department of Computer Science, North Maharashtra University, Jalgaon (MS), India.

Section 1: Algorithm based on hyperbolic Cholesky factorization.

In this section a brief discussion on the method of Delosme and Ipsen [8], based on hyperbolic rotations, for the solution of linear systems of equations

$$Ax = b \tag{1.1}$$

with symmetric positive definite coefficient matrix A, is given. Cholesky factor of A is first determined by essentially pre-multiplying A with appropriate hyperbolic rotations, whose product is called Q. Next simple matrix vector multiplication involving Q to the right hand side of (1.1) provides a novel way of solving the system. Avoiding forward elimination and back substitution is a very desirable feature for a parallel implementation.

1.1 The hyperbolic Cholesky factorization.

The computation of the Cholesky decomposition,

$$A = U^T U,\ U_{n \times n} \text{ upper triangular} \tag{1.2}$$

of real symmetric positive definite(spd) $n \times n$ matrix $A = a_{ij}$, by means of hyperbolic rotations is called **hyperbolic Cholesky factorization**. Its derivation is based on a particular decomposition of the matrix A as given in [8]. Since A is spd, $a_{k,k}$ is strictly positive and $A^{(k)}$ can be written as the difference of the outer product.

$$A^{(k)} = v_k^T v_k - w_k^T w_k \tag{1.3}$$

where v_k and w_k are row vectors with elements

$$v_{k,j} = \begin{bmatrix} a_{k,k}^{-1/2}\ a_{k,j} & j \ge k \\ 0 & \text{otherwise} \end{bmatrix} \tag{1.4}$$

$$w_{k,j} = \begin{bmatrix} v_{k,j} & j \ne k \\ 0 & j = k \end{bmatrix} \tag{1.5}$$

That is, v_k consists of non zero rows of $A^{(k)}$ scaled by the square root of the diagonal element, while w_k differs from v_k only in its k^{th} entry. Stacking the v_k and w_k respectively in upper triangular matrices.

$$V = \begin{bmatrix} v_1 \\ v_2 \\ \cdot \\ \cdot \\ v_n \end{bmatrix},\ W = \begin{bmatrix} w_1 \\ w_2 \\ \cdot \\ \cdot \\ w_n \end{bmatrix} \tag{1.6}$$

one has A = V^TV-W^TW. (1.7)

***Lemma 1.1*:** Let R and S be upper triangular $n \times n$ matrices such that R^TR-S^TS is positive definite, and let $\mu_{k,k} = r_{k,k}^{-1} s_{k,k}$, $1 \le k \le n$.

$$\begin{bmatrix} \tilde{R} \\ \tilde{S} \end{bmatrix} = \hat{Q} \begin{bmatrix} R \\ S \end{bmatrix} \qquad \text{where } \hat{Q} = \tilde{Q}^{(n)} \dots \tilde{Q}^{(1)} \text{ and}$$

$$\tilde{q}_{ij}^{(k)} = \begin{cases} 1 & i = j \neq k \text{ or } i = j \neq n+k \\ (1-\mu_k^{-2})^{-1/2} & i = j = k \text{ or } i = j = n+k \\ -(1-\mu_k^2)^{-1/2}\mu_k & (i,j) = (k, n+k) \text{ or } (i,j) = (n+k, k) \\ 0 & \text{otherwise} \end{cases} \tag{1.8}$$

then $\hat{Q}$ is pseudoorthogonal, $\tilde{R}$ is upper triangular, and $\tilde{S}$ is strictly upper triangular (upper triangular with zero diagonal).

Remarks:

1. Since the matrix $\tilde{Q}^{(k)}$ constitute disjoint rotations, they commute and can be applied in any order. Their product $\hat{Q}$ has a very simple expression:

$$\hat{q}_{k,k} = \hat{q}_{n+k,n+k} = (1-\mu_k^2)^{-1/2}, \quad 1 \le k \le n \tag{1.9}$$

$$\hat{q}_{k,n+k} = \hat{q}_{n+k,k} = -(1-\mu_k^2)^{-1/2}\mu_k, \quad 1 \le k \le n \tag{1.10}$$

$$\hat{q}_{ij} = 0, \qquad i \neq j (\text{mod n}) \tag{1.11}$$

2. The diagonal elements of $\tilde{R}$ have same sign as the corresponding diagonal elements of R; thus if R has a positive diagonal, $\tilde{R}$ also has a positive diagonal.

***Theorem 1.1* (The Hyperbolic Cholesky Algorithm):**

Let A be a $n \times n$ spd matrix and V and W be upper triangular matrices as defined in (1.4) and (1.5), so that A = V^TV-W^TW. Set

$$\begin{bmatrix} R^{(0)} \\ S^{(0)} \end{bmatrix} = \begin{bmatrix} V \\ W \end{bmatrix} \tag{1.12}$$

and apply the sequence of operations

$$\begin{bmatrix} R^{(l+1)} \\ S^{(l+1)} \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & P \end{bmatrix} \hat{Q}^{(l)} \begin{bmatrix} R^{(l)} \\ S^{(l)} \end{bmatrix} \tag{1.13}$$

$$l = 0, \ldots, n-1$$

Where $\hat{Q}^{(l)}$ is obtained from $R^{(l)}$ and $S^{(l)}$ in the same way as $\hat{Q}$ is constructed from R and S in lemma 1.2, and where P is the $n \times n$ circular permutation matrix with $P_{1,n} = 1$ and $P_{i,i-1} = 1$, $2 \le i \le n$, then $R^{(n)} = U$, the Cholesky factor of A, and $S^{(n)} = 0$.

Remarks:
The transformation performed by the hyperbolic Cholesky algorithm is denoted by

$$Q = \begin{bmatrix} I & 0 \\ 0 & P \end{bmatrix} \hat{Q}^{(n-1)} \ldots \begin{bmatrix} I & 0 \\ 0 & P \end{bmatrix} \hat{Q}^{(l)} \ldots \begin{bmatrix} I & 0 \\ 0 & P \end{bmatrix} \hat{Q}^{(0)} \tag{1.14}$$

The matrix Q is pseudoorthogonal, since it is the product of pseudoorthogonal matrices.

Application of hyperbolic rotations to the solution of linear systems:

The hyperbolic Cholesky algorithm determines simultaneously the Cholesky factor U of an spd matrix A and a set of n(n-1)/2 parameters $\mu_k^{(l)}$, $1 \le l \le n-1$, $1 \le k \le n$, which defines the hyperbolic rotations that make up the matrix Q. For the solution of the spd system $Ax = b$, the above algorithm could be used to find U followed by forward elimination to solve system $U^T y = b$ and by back substitution to solve $Ux = y$. Instead the above algorithm can also be used to find parameter $\mu_k^{(l)}$ followed by the application of the hyperbolic rotation to the **b** in a particular way to get the solution vector **x**.

The algorithm for the solution of $Ax = b$ can be summed up as follows: Let A_α be the upper part of A and A_β its strictly upper triangular part ($A_\alpha = D^2 + A_\beta$). Using the hyperbolic Cholesky algorithm as specified in the theorem, express the matrix Q as a product of hyperbolic rotations such that-

$$Q\Delta \begin{bmatrix} A_\alpha \\ A_\beta \end{bmatrix} = \begin{bmatrix} U \\ 0 \end{bmatrix} \tag{1.15}$$

where

$$\Delta = \begin{bmatrix} D^{-1} & 0 \\ 0 & D^{-1} \end{bmatrix} \tag{1.16}$$

and D = diag $(a_{11}^{1/2}, \ldots, a_{nn}^{1/2})$, so apply the same operation to $\begin{bmatrix} b \\ b \end{bmatrix}$ to obtain

$$Q\Delta\begin{bmatrix} b \\ b \end{bmatrix} = \begin{bmatrix} U^{-T}b \\ L^{-T}b \end{bmatrix} \tag{1.17}$$

and then apply these operations essentially in reverse order to get form

$$x = (I, I)\Delta Q^T\begin{bmatrix} \alpha U^{-T}b \\ (1-\alpha)L^{-T}b \end{bmatrix} \tag{1.18}$$

where α is an arbitrary real number, which can be equal to 0 or 1 for convenience.
and Q^T is given by

$$Q^T = \hat{Q}^{(0)}\begin{bmatrix} I & 0 \\ 0 & P^{-1} \end{bmatrix}\hat{Q}^{(1)}\begin{bmatrix} I & 0 \\ 0 & P^{-1} \end{bmatrix}\ldots\hat{Q}^{(n-1)}\begin{bmatrix} I & 0 \\ 0 & P^{-1} \end{bmatrix} \tag{1.19}$$

Section 2: The Computational Model.

Recently the Processor Array with Reconfigurable Bus Systems (PARBS) has drawn lot of attention in the scientific community for its high performance computing with general purpose processors. Various models of PARBS appear in the literature[5]. Common models are Bus automaton, Polymorphic torus network[16], Reconfigurable Meshes[29,19], Bypass capabilities etc.

An N_1 x N_2 x ...x N_r PARBS consist of an array of N_1 x N_2 x ...x N_r processors that is connected to a r-dimensional grid shaped reconfigurable bus system. The processing elements are connected to a bus through a fixed number of I/O ports. The ability to change the configuration of the bus system by adjusting local or global switches makes this architecture interesting to obtain various computational configurations like row, zig-zag, staircase & diagonal at run time. A two dimensional processor array with a reconfigurable bus system of size N^2, consisting of identical processors, connected to a NxN rectangular mesh system, is called reconfigurable mesh. In figure 1, we see processing elements connected to a grid of buses and a PE with its four I/O ports and connection pattern.

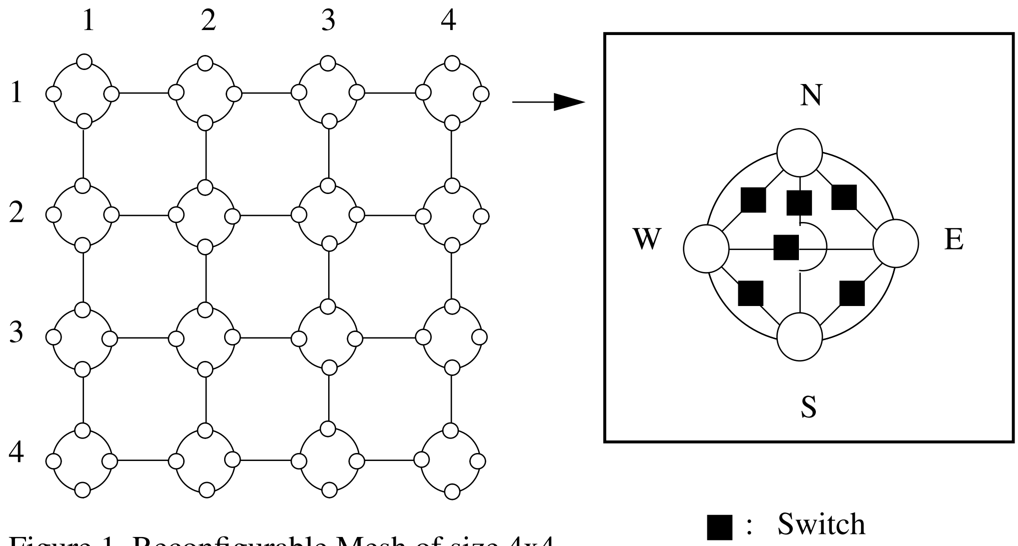

Figure 1. Reconfigurable Mesh of size 4x4

The basic computational unit of the reconfigurable mesh is the Processing Element (PE) which consists of switches, small storage and an ALU. A PE is capable of performing following operations in one unit of time

1. Setting up a connection pattern.
2. Read from a or write onto a sub bus or memory
3. Performing logical and arithmetic operations
4. Disconnecting itself from the bus.

Higher dimensional PE's can be formed in the same way. Figure 2 (b) shows a PE in 3-dimension with six ports labelling (U)pper, (L)ower, (F)ront, (B)ack, (R)ight and (L)eft.

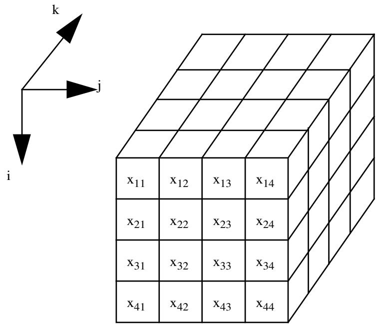

Figure 2(a). A 3-D configuration.

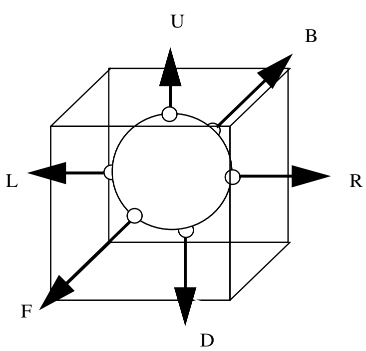

Figure 2(b). A PE in 3-D.

We assume that constant time is needed to broadcast values through the established buses irrespective of their distance from the first processor. Although it is a theoretical assumption and somewhat unrealistic on the current architecture, with the advancement in the fibre optics communication technology, this architecture is expected to gain wide popularity.

Section 3:

The parallel algorithm for spd systems works in stages, we first present a pseudocode for the algorithm and then present the PARBS implementation.

Begin

[1] Read *A,b*

[2] **for all** k, $1 \le k \le n$ **in parallel do**

for all j, $1 \le j \le n$ **in parallel do**

begin

$$V_{k,j} = \begin{cases} A_{k,j} / \sqrt{A_{k,k}} & \text{if } j \ge k \\ 0 & \text{otherwise} \end{cases}$$

$$W_{k,j} = V_{k,j} \text{ if } j \ne k$$

end;

[3] $\Delta = \begin{bmatrix} D^{-1} & 0 \\ 0 & D^{-1} \end{bmatrix}$

[4] **for all** k, $1 \le k \le n$ **in parallel do**

for all j, $1 \le j \le n$ **in parallel do**

begin

$$R_{i,j} = V_{i,j}$$

$$R_{i+n,j} = W_{i,j}$$

end;

[5] **for all** i, $1 \le i \le 2n$ **in parallel do**

for all j, $1 \le j \le 2n$ **in parallel do**

begin

$Q_{i,j}^{tot} = 1$ if $i = j$ else 0

$Q_{i,j} = 1$ if $i = j$ else 0

end;

[6] p:=1

while $p \le n$ **do**

begin

[6.1] E = interchange(Q^{tot})

[6.3] **if** ($p < n$) **then**

begin

[6.3.1] F = qmult(Q,R)

[6.3.2] R = interchange(F)

[6.3.3] Q = qconstruct(R)

[6.3.4] Q^{tot} = qmult(Q,E)

end;

p = p + 1;

end;

[7] **in parallel do**

begin

[7.1] Y = vmult(E, (bb = dmult(Δ,b)))

[7.2] Z = emult(H= (umult((I,I), Δ)),E^T)

end;

[8] x = vmult(Z, $\begin{pmatrix} \alpha \\ 1-\alpha \end{pmatrix} Y$)

End.

The important results used in the design process are presented next.

Lemma 3.1. A semi group computation of $a_1+a_2+...+a_n$ where '+' denotes an associative operator can be performed in O(log n) time on a linear PARBS of n processors. Initially each processor owns a value. The computed value is stored in the leftmost processor. [29]

Lemma 3.2. On a 2-D, $n \times n$ mesh, n data items d_1, d_2, ..., d_n can be moved from the first column to the first row in O(1) time. [5]

Theorem 1. The transpose of a given square matrix A_{nxn} can be obtained in constant time on $n \times n \times n$ PARBS. Initially P(i,j,1) owns the values $a_{i,j}$, $1 \le i \le n$, $1 \le j \le n$ after the computation is over P(i,j,1) contains the transpose of A.

Proof:

Step1. Establish straight bus in *k*-direction. Each processor connects port F to B. After this step n^2 straight buses are established, each one for n^2 $PARBS_{i,j}$, $1 \le i \le n$, $1 \le j \le n$.

Step 2. Processor P(i,j,1) broadcast values A[i,j] on the buses such that A[i,j] is broadcasted on P(i,j,j), $1 \le i \le n$, $1 \le j \le n$. (figure 3(a)).

Step 3. Break-up the established sub buses (i.e. disconnect port F to B).

Step 4. Establish sub buses in *j*-direction, i.e. connect port L to R.

Step 5. Broadcast values of A from P(i,j,k) to P(i,i,k), $1 \le i \le n$, $1 \le j \le n$, $1 \le k \le n$. (figure 3(b)).

Step 6. Disconnect port L to R and establish new sub buses in *i*-direction and transmit data items of A from processor P(i,i,k) to processor P(k,i,k), $1 \le i \le n$, $1 \le k \le n$, (figure 3(c)).

Step 7. Disconnect established sub buses and establish new sub buses in *k*-direction and broadcast values back to P(i,j,1), (figure 3(d)).

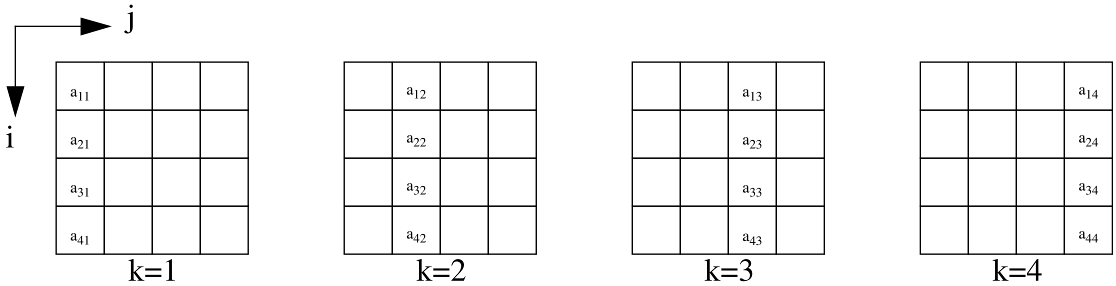

Figure 3(a): After steps 1-3

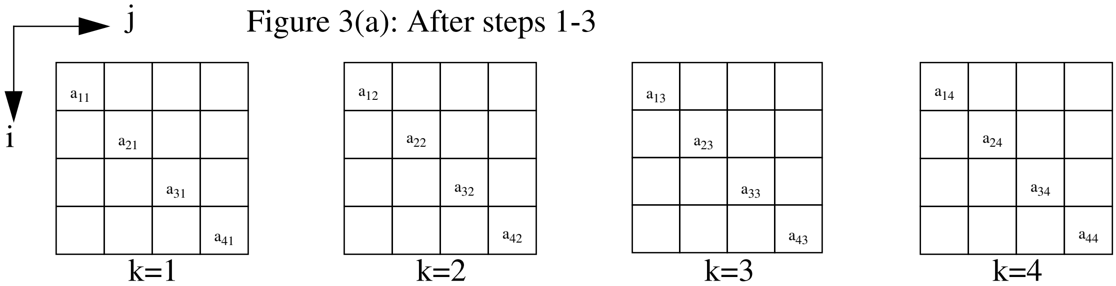

Figure 3(b): After steps 4-5

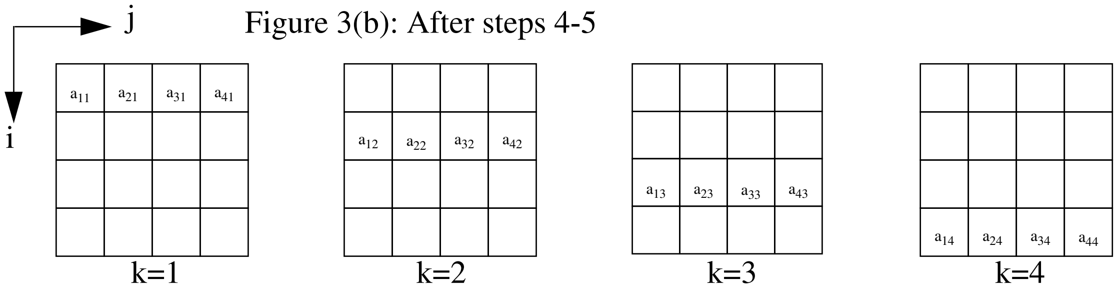

Figure 3(c): After step 6

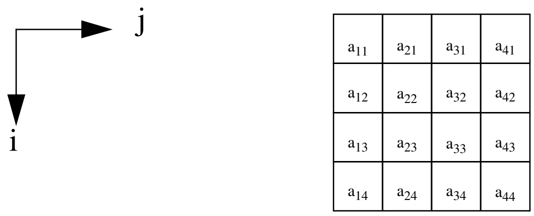

Figure 3(d): After step 7

It is not difficult to see that the transpose of a matrix A_{nxn} can be obtained in O(1) time with $n \times n \times n$ processors.

3.2 PARBS implementation of SPD solver.

Now we present an algorithm for SPD system of equations on PARBS.

Algorithm:

Phase I: We assume the existence of two working spaces where most of the operations are performed.We assume that the values of A, b and Q^{tot} are initially stored at the **first layer** of a $2n \times 2n \times 5n$ PARBS. *A[i,j]* is stored at processor P[i,j,1] and P[i+n,j,1], $1 \le i \le n$, $1 \le j \le n$,

b is stored at processors P[i+n,n+1,1], $1 \le i \le n$. We assume that the identity matrix is stored at the processors P[i,j+n,1], $1 \le i \le n,\ 1 \le j \le n$, as initial value for D.

Step 1: Establish sub buses in j-direction i.e. connect port L to R, broadcast values of *A[i,i]* to P[i,j,1] on the established buses for the first n rows and *A[i+n,i]* on the remaining n rows.

Step 2: For the first n rows on the established sub buses, divide values of *A[i,j]*, *D[i,j]* at P[i,j,1] by the value received in step 1, i.e. by *A[i,i]* if the index $j \ge i$, otherwise convert *A[i,j]* = 0, $1 \le i \le n,\ 1 \le j \le n$.

Step 3: For the remaining n rows on the established sub buses, perform the same operation as in step 2, if the index *(j+n) > i*. After completion of this step we get initial values of R_{nxn} at the position P[i,j,1], $1 \le i \le 2n,\ 1 \le j \le n$ and a diagonal matrix D at P[i,j+n,1], $1 \le i \le n,\ 1 \le j \le n$.

Phase II: The processes involved in this phase are repeated n times.

In step **6.1** of the algorithm we multiply a $2n \times 2n$ matrix with another $2n \times 2n$ matrix $\begin{bmatrix} I & 0 \\ 0 & P \end{bmatrix}$, where P is a $n \times n$ circular permutation matrix. The process of the multiplication is a sequence of row interchange operations. The $2n^{th}$ row of the second matrix becomes $n+1^{st}$ row, $n+1^{st}$ becomes $n+2^{nd}$ and so on. $2n-1^{st}$ row becomes $2n^{th}$ row. P-RAM based algorithm for the procedure computation is given as a procedure 3.1.

```
for i:= 1 to n in parallel do
    for j:= 1 to c1 in parallel do
        E[i,j] := mat[i,j];

for j:= 1 to c1 do in parallel do
    E[n+1,j] := mat[2*n,j];

for i:= (n+1) to r1-1 in parallel do
    for j:= 1 to c1 do in parallel do
        E[i+1,j]:= mat[i,j];
```

Where c_1, r_1 denotes column, row index of the first matrix.

Procedure 3.1 Row interchange procedure

This operation can be performed on a $2n \times 2n \times n$ PARBS in constant time. We establish a sub bus in k-direction and send values of Q^{tot} to the processors P[i,j,1] of working space I in constant time.

Step 1: Establish sub buses in k-direction and send the row vectors of Q^{tot} from the $n+1^{st}$ row to the $2n^{th}$ row to processor P[i+n,j,k], $1 \le i \le n$, $1 \le j \le n$, $1 \le k \le n$.

Step 2: Establish sub buses in *i*-direction. Processors P[i+n,j,i], $1 \le i \le n$, broadcast values of $Q^{tot}[i+n,j]$ on the established sub buses. After the process is over $Q^{tot}[i+n,j]$ is stored on processors P[i+n,j,i].

Step 3: Disconnect established connections and establish new sub buses in *k*-direction and broadcast values of these row vectors from P[i+n,j,k], $2 \le i \le n$, $1 \le k \le n-1$ to P[i+n,j,1] and values at P[n+1,j,n] to P[n+1,j,1].

Step 4: At the end of the step 3, required result is available at processors P[i,j,1] of working space I. Establish sub buses in *k*-direction and broadcast values back to the **first layer.**

In step **6.3.1** of the algorithm, the matrix *Q* which is initially an identity matrix and stored at the **first layer**, is multiplied by *R*, obtained in the step **4**. We establish sub buses in *k*-direction and broadcast the values of *Q* and *R* to the working space I. Initially we assume that the values are stored at P[i,j,1] of this working space. Due to the special structure of matrix *Q,* we design a procedure which takes constant time to multiply two $2n \times 2n$ matrices on a $2n \times 2n \times 2n$ PARBS. We first present P-RAM based algorithm for the procedure, followed by a PARBS mapping of it. Here r_1 denotes row index of the first matrix and c_2 denote column index of the second matrix as given in procedure 3.2.

```
for i:= 1 to (r1 div 2) in parallel do
    for j:= 1 to c2 in parallel do
      begin
        F[i,j]:= Q[i,i] * R[i,j] + Q[i,i+n] * R[i+n,j];
        F[i+n,j] := Q[i+n,i] * R[i,j] + Q[i+n,i+n]*R[i+n,j];
      end;
```

Procedure 3.2 Multiplication by *Q*

Sub phase I:

Step 1: Establish the straight sub bus in *k*-direction. Each processor connects port F to B.

Step 2: Processor P[i,j,1] broadcast values *Q[i,j]* on the connected sub buses. After this step, processor P[i,j,k] owns the value of *Q[i,j]*, (Figure 4(a)).

Step 3: Break-up the established sub buses (each processor disconnect port F to B).

Step 4: Establish straight buses in *j*-direction.

Step 5: Processors P[i,k,k] broadcast the values *Q[i,k]* on the bus which is connected to $1 \le i \le n$, $1 \le k \le n$. After step 5 processor P[i,j,k], $1 \le j \le n$ owns the value of *Q[i,k]*, (Figure 4(b)).

Sub phase II.

Step 6: Establish sub buses in *k*-direction. Processors P[i,j,1], of working space I, broadcast values of *R[i,j]* on the established sub buses.

Step 7: Establish straight buses in *i*-direction. (Figure 4(c)).

Step 8: Processors P[k,j,k] broadcast values *R[k,j]* on the sub buses. After this step processor P[i,j,k] owns the value of *R[k,j]*, (Figure 4(d)).

Step 9: Break-up established sub buses.

Multiplication Phase: Each processor P[i,j,k] computes the value of *Q[i,k] * R[k,j]*, which takes constant time.

Summing Phase: Establish a straight bus in *k*-direction and broadcast values, obtained after the multiplication phase, from P[i,j,i+n] to P[i,j,1] and P[i+n,j,i+n] to P[i+n,j,i], $1 \le i \le n$, $1 \le j \le n$. Broadcasting values in this way takes constant time. Perform the summation at the respective processors and send values back to the second layer by establishing sub buses in *k*-direction.

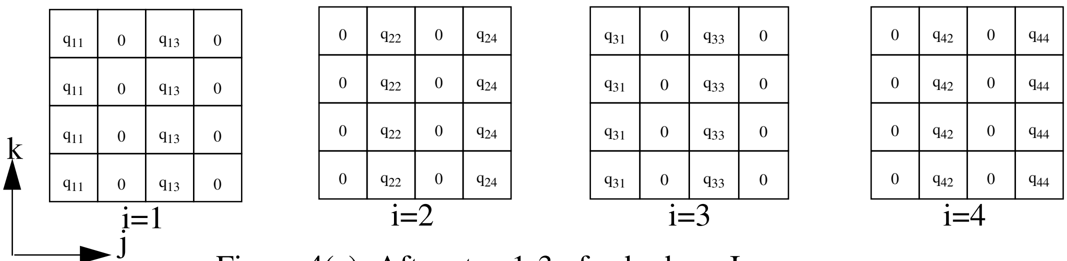

Figure 4(a): After step 1-3 of sub phase I.

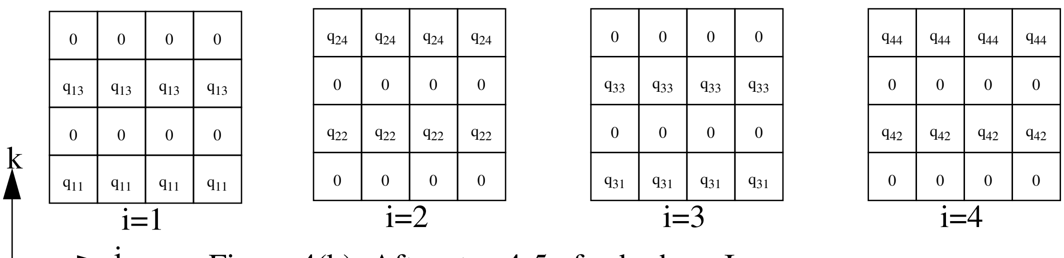

Figure 4(b): After step 4-5 of sub phase I.

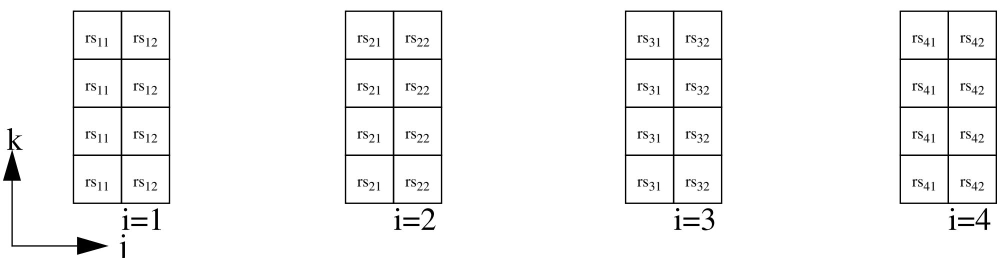

Figure 4(c): After step 6-7 of sub phase II.

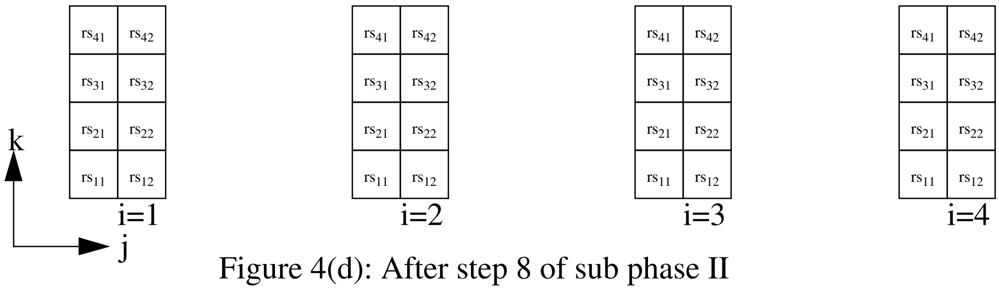

Figure 4(d): After step 8 of sub phase II

The next step **6.3.2** of the algorithm is again an interchange operation which has been described in step **6.1.** After the process is over we obtain the new value of *R* and store it at the **first layer** in constant time.

In step **6.3.3** we construct *Q*, we first present P-RAM algorithm (procedure 3.3) for the process and then PARBS implementation of it.

```
for i:= 1 to 2*n in parallel do
  for j:= 1 to 2*n in parallel do
     Q[i,j] := 0.0;

for k:= 1 to n in parallel do
  begin
       qq:= mat1[n+k,k] / mat1[k,k];
      Q[k,k] := 1/sqrt(1-sqr(qq));
     Q[n+k,n+k] := Q[k,k];
     Q[k,n+k] := -(qq/sqrt(1-sqr(qq)));
    Q[n+k,k] := Q[k,n+k];
 end;
```

Procedure 3.3 Construction of *Q*

Step 1: Establish a sub bus in *k*-direction, processors P[i,j,1], $1 \le i \le 2n$, $1 \le j \le n$ and broadcast values of *R* as the initial value for *Q* to working space I.

Step 2: Establish sub buses in *i*-direction (Connect port U and D). Processors P[i,i,1] broadcast values of *Q[i,i],* $1 \le i \le n$ on the established sub buses.

Step 3: Divide the values of *Q* by the value of *Q[i,i]*, received in the last phase i.e. ρ_k = *Q[n+k,k]/Q[k,k],* $1 \le k \le n$**.**

Step 4: Processors P[i+n,i,1], $1 \le i \le n$ broadcast the values of ρ_k on the established sub buses.

Step 5: Disconnect port U to D**.**

Step 6: Connect port L to R. Processors P[i,i,1] and P[i+n,i,1], $1 \le i \le n$, broadcast the values of ρ_k on the established sub buses.

Step 7: Processors P[k,k,1] and P[n+k,n+k,1], $1 \le k \le n$ perform the operation $Q[k, k] = Q[n+k, n+k] = 1/(\sqrt{1-\rho_k})$ and Processors P[k,n+k,1] and P[n+k,k,1], $1 \le k \le n$, perform $Q[k, n+k] = Q[n+k, k] = -\rho_k/(\sqrt{1-\rho_k^2})$.

Step 8: Disconnect the established sub buses. After the process is over, we get the value of Q in constant time.

The step **6.3.4** is a multiplication of matrix Q with the matrix E obtained in the step **6.1.** As we described in step **6.3.1**, the procedure **qmult** multiplies two matrices in constant time. The resultant matrix Q^{tot} is stored at **first layer**.

Once the values are received after step **6** of pseudocode, we can use them to avoid the forward elimination and back substitution process. The two processes **7.1** and **7.2**, given below are executed **parallely**. We broadcast values of b to the working space I, and D to working space I and II both, by establishing the sub buses in k-direction. The two steps **7.1** and **7.2** are described separately. The step **7.1** consists of two steps **dmult**(Δ,b) and **vmult**(E, result of procedure dmult). We first describe **dmult**.

Phase III.

Step 1: The broadcasted $n \times n$ diagonal matrix D at the processor P[i,j+n,1], $1 \le i \le n$, $1 \le j \le n$ of working space I and b at P[i+n,n+1,1], $1 \le i \le n$. We establish sub buses in j-direction so there are n sub buses which are connected to the ports L and R of the processors.

Step 2: Broadcast the values of b from the processors P[i+n,n+1,1] to processors P[i+n,i+n,1], $1 \le i \le n$ on the established sub buses.

Step 3: Disconnect the established sub buses and connect new sub buses in i-direction. Processors P[i+n,i+n,1], $1 \le i \le n$ send values of b to processors P[i,i+n,1], $1 \le i \le n$.

Step 4: Perform the multiplication operation of the received values on the respective processors.

Step 5: On the established sub buses send the result back from processors P[i,i+n,1] to P[i+n,i+n,1], $1 \le i \le n$

Step 6: Disconnect ports U and D and establish sub buses in j-direction i.e. connect port L to R, broadcast values from P[i,i+n,1] to P[i,1,1] and from P[i+n,i+n,1] to P[i+n,1,1], $1 \le i \le n$.

The second step of **7.1** is a procedure **vmult**(Q * column vector resulted from the last step).

This is a simple matrix vector multiplication which can be done in O(log n) time on a $2n \times 2n \times 2n$ PARBS using a standard procedure described in [10]. The resultant $2n \times 1$ matrix Y is stored at processors P[i,1,1], $1 \le i \le 2n$ of working space I.

The step **7.2** consists of two multiplication steps. First the Δ, obtained in the form of D from the first layer, is multiplied by $n \times 2n$ matrix *(I, I)*, made up of $n \times n$ identity matrix ***I*,** followed by a procedure **emult.** It can be observed from the property of such matrix that the resultant matrix is a matrix with it's rows 1 to n are added to n+1 to 2n. Here we describe a procedure which takes constant time for such row additions. This operation can be done in only one step.

Step 1: The values of D are available in P[i,i+n,1], $1 \le i \le n$ of working space II. We establish a sun bus in *j*-direction by connecting port L to R, broadcast the value of D from processor P[i,i+n,1] to processors P[i,i,1], $1 \le i \le n$.

We have already proved that the transpose of a matrix $A_{n \times n}$ can be obtained in constant time using $n \times n \times n$ PARBS, the second procedure **emult** is a simple vector matrix multiplication and it can be executed on PARBS of size $2n \times 2n \times 2n$ in O(log n) time [10].

In step **8** of the algorithm the resultant matrix of the last step, Z, is multiplied by Y by taking appropriate value of α as 0 or 1. This is a vector matrix multiplication which can be executed on PARBS of size $2n \times 2n \times 2n$ in O(log n) time [10].

It can be observed that all steps in the loop of step **6** takes constant time and repeated n times, the algorithm presented above can obtain the solution of spd system in O(n) time using $2n \times 2n \times 5n$ processors, where $2n \times 2n \times 4n$ processors are required for working space.

Concluding Remarks: For the solution of linear systems of equations $Ax = b$ with symmetric positive definite matrix $A_{n \times n}$, an algorithm based on hyperbolic Cholesky factorization on processor arrays with reconfigurable systems is presented and its parallel time and processors complexity is calculated. The processors complexity is on the upper bound and with the clever use, can be further reduced. Since the matrix multiplication, due to the summation of n values in an inner vector product, is bounded by O(log n) parallel time, fast and efficient matrix manipulation on a linear array with a reconfigurable pipeline bus system using optical buses has been proposed recently [19]. One of the fast algorithm for solving linear system of equations with this latest pipeline architecture, takes $O(\log N)^{1+\delta}$ parallel time with $O\left(N^4 / \left(\frac{8}{7}\right)^{(\log N)^{\delta}}\right)$ processors, where $0 \le \delta \le 1$ [19]. The algorithm presented in section 3 has parallel time complexity of O(n). It can be considered as a fast algorithm from the practical point of view but not the efficient parallel algorithm as it does not belongs to Nick's class[12,35]. The work can be further extended to find the efficient parallel algorithm for the solution of spd systems of equations using a pipelined bus architecture.

References:

1. Aggarwal A., "Optimal Bounds for Finding Maximum on Array of Processors with k Global Buses", **IEEE Transaction on Computers**, pp. 62-64, Vol. c-35, No. 1, January 1986.
2. Aho A., J. Hopcroft and J. Ullman, **The Design and Analysis of Computer Algorithms**, Addison-Wesley, 1974.
3. Alaghband Geeta, "Parallel sparse matrix solution and performance", **Parallel Computing**, pp.1407-1430, 21(1995).
4. Bauer B. E., **Practical Parallel Programming**, Academic Press Inc., 1992.
5. Biing-Feng Wang and Gen-Huey Chen, "Constant Time Algorithms for the Transitive Closure and Some Related Graph Problems on Processor Arrays with Reconfigurable Bus Systems", **IEEE Transaction on Parallel and Distributed Systems**, pp. 500-507, Vol. 1, No. 4, October 1990.
6. Biing-Feng Wang, Gen-Huey Chen and Ferng-Ching Lin, "Constant Time Sorting on a Processor Array with Reconfigurable Bus System", **Information Processing Letters**, pp. 187-192, 34(1990).
7. Bokhari S. H., "Finding Maximum on an Array Processor with a Global Bus", **IEEE Transaction on Computers**, pp. 133-139, Vol. c-33, No. 2, February 1984.
8. Delosme J.M., Ilse C.F. Ipsen, "Parallel Solution of Symmetric Positive Definite Systems With Hyperbolic Rotations", **Linear Algebra and its Applications**, Special Volume on Parallel Computing, North Holland, NY, pp. 75-111, Vol. 77, May 1986.
9. Duff I.S., A.M. Erisman and J.K. Ried, **Direct Methods for Sparse Matrices**, Clarendon Press Oxford, 1986.
10. G-H Chen, B-F Wang and C.J. Lu, "On the parallel computation of the algebraic path problem", **IEEE Transaction on Parallel and Distributed Systems**, pp. 251-256, Vol. 3, No. 2, March 92.
11. G-H Chen, S. Olariu et al., "Constant Time Tree Algorithms on Reconfigurable Meshes on Size n x n", **Journal of Parallel and Distributed Computing**, pp. 137-150, 26 (1995).
12. Gibbons A. And Wojciech Rytter, **Efficient Parallel Algorithms**, Cambridge University Press, Cambridge, May 1988.
13. Golub G.H. and Charles and F. Van Loan, **Matrix Computations**, John Hopkins Press, Baltimore, MA, 1983.
14. Golub G.H. and James Ortega, **Scientific Computing: An Introduction with Parallel**

Computing, Academic Press Inc., 1993.

15. Horowitz E. And S.Sahni, **Fundamentals of Computer Algorithms**, Addison-Wesley, Computer Science Press, NY, 1978.
16. Hungwen Li and Massimo Maresca, “Polymorphic Torus Network”, **IEEE Transaction on Computers**, pp. 1345-1351, Vol. 38, No. 9, September 1989.
17. Hwang Kai and F.A.Briggs, **Computer Architectures and Parallel Processing**, McGraw-Hill International Edition, 1985.
18. Jain Abhay, N.S. Chaudhari, “Efficient parallel recognition of context-free languages”, **Parallel Computing**, pp. 1303-1321, 20(1994).
19. Ju-Wook J. and V.K. Prasanna, “An Optimal Sorting Algorithm on Reconfigurable Mesh”, **Journal of Parallel and Distributed Computing**, pp. 31-41, 25 (1995).
20. K. Li, Y. Pan and S.Q. Zheng, “Fast and Efficient Parallel Matrix Manipulation on a Linear Array with a Reconfigurable Pipelined Bus System”, **High Performance Computing Systems and Application**s, J.Schaeffer and R.Unrau eds. , Kluwer Academic Press, 1998.
21. Koji Nakano and Koichi Wada, “Integer Summing Algorithms on Reconfigurable Meshes”, **IEEE First International Conference on Algorithms and Architectures for Parallel Processing**, Brisbane, Australia, pp. 19-21, April 1995.
22. Koji Nakano and Stephan Olariu, “An Optimal Algorithm for the Angle Restricted all Nearest Neighbour Problem on Reconfigurable Mesh with Applications”, **Proc. 10th International Parallel Processing Symposium**, Honolulu, Hawaii, April 15-19, 1996.
23. Koji Nakano, “Optimal Initializing Algorithms for a Reconfigurable Mesh”, **Journal of Parallel and Distributed Computing**, pp. 218-223, 24 (1995).
24. Koji Nakano, “Prefix sums algorithms on Reconfigurable Meshes”, **Parallel Processing Letters**, pp. 23-35, Vol.5 No.1 (1995).
25. Lancaster P.,M. Tismenetsky, **The Theory of Matrices**, Academic Press Inc., (1985).
26. Lester B. P., **The Art of Parallel Programming**, Prentice Hall, Englewood Cliffs, N.J. (1993).
27. Linda Kaufman, “The Generalized Householder Transformation and Sparse Matrices”, **Linear Algebra and its applications**, pp. 221-234, 90 (1987).
28. Louis A. Hageman and David M. Young, **Applied Iterative Methods**, Academic Press Inc., NY, 1981.
29. Miller R., V.K. Prasanna Kumar, Dionisios Reisis and Quentin F. Stout, “Meshes with Reconfigurable Buses”, **Proc. 15th MIT Conference on Advance Research in VLSI**, pp. 163-178, March 1988.

30. Stewart G.W., **Introduction to Matrix Computations**, Academic Press Inc., SanDiego California 1973.
31. Susanne E.H., “VLSI Algorithms for the Connected Component Problems”, **SIAM Journal Computing**, pp. 355-365, Vol. 12, No. 2, May 1983.
32. Tatsuya Hoyashi, Koji Nakano and Stepahn Olariu, “Efficient List Ranking on Reconfigurable Meshes with Applications”, **Hayashi Technical Report**, May 1996.
33. Ten-Hwang Lai, M.J.Shwng, “Triangulation on Reconfigurable Meshes: A Natural Decomposition Approach”, **Journal of Parallel and Distributed Computing**, 38-51, 30 (1995).
34. V.Bokka, H.Gurla, S. Olaru and J.L. Schwing, “Constant Time Convexity Problems on Reconfigurable Meshes”, **Journal of Parallel and Distributed Computing**, pp.86-99, 27 (1995).
35. Vipin Kumar et at., **Introduction to Parallel Computing, Design and Analysis of Algorithms**, The Benjamin Cummings Publishing Company Inc. , California, 1994.
36. Voevodin V.X., **Mathematical Foundations of Parallel Computing**, World Scientific Publishing Co., 1986.
37. Wankar R. and N.S.Chaudhari, “Parallel Cholesky Factorization Algorithm”, **National Conference on current trends in Information Technology**, pp. 23-24, June 1995, Bhopal.
38. Wankar R., “Parallel Algorithms for Solving Symmetric Positive Definite System (SPD) of Equations”, **International Journal of Management and Systems**, pp. 311-324, Vol. 11, No. 3, Sept-Dec 95.
39. Wei Shu, “Parallel implementation of a sparse simplex algorithm on MIMD distributed memory computers”, **Journal of Parallel and Distributed computing**. pp. 25-40, 31(1995).
40. Yakowitz S., F. Szidarovszky, **An Introduction to Numerical Computations**, MacMillan Publishing Co., 1989.
41. Yen-Cheng Chen and Wen-Tsuen Chen, “Constant Time Sorting on Reconfigurable Meshes”, **IEEE Transaction on Computers**, pp. 749-751, Vol. 43, No. 6, June 1994.
42. Yen-Cheng Chen et al., “Designing Efficient Parallel Algorithms on Mesh-Connected Computers with Multiple Broadcasting”, **IEEE Transaction on Parallel and Distributed Systems**, pp. 241-245, Vol. 1, No. 2, April 1990.
43. Yosi Ben-Asher, Dan Gordon and Assaf Schuster, “Efficient Self-Simulation Algorithms for Reconfigurable Arrays”, **Journal of Parallel and Distributed Computing**, pp. 01-22, 30 (1995)

44. Yue-Li Wang, H.C.Chen and Chen-Yu Lee, “An O(log n) parallel algorithm for constructing a spanning tree on permutation graphs”, **Information Processing Letters**, pp. 83-87, 56(1995).