Journal of Computer Science 3 (12): 956-964, 2007
ISSN 1549-3636

Some P-RAM Algorithms for Sparse Linear Systems

[1]Rakesh Kumar Katare and [2]N.S. Chaudhari
[1]Department of Computer Science, A.P.S. University, Rewa (M.P.) 486003, India
[2]Nanyang Technological University, Singapore

Abstract: PRAM algorithms for Symmetric Gaussian elimination is presented. We showed actual testing operations that will be performed during Symmetric Gaussian elimination, which caused symbolic factorization to occur for sparse linear systems. The array pattern of processing elements (PE) in row major order for the specialized sparse matrix in formulated. We showed that the access function in^2+jn+k contains topological properties. We also proved that cost of storage and cost of retrieval of a matrix are proportional to each other in polylogarithmic parallel time using P-RAM with a polynomial numbers of processor. We use symbolic factorization that produces a data structure, which is used to exploit the sparsity of the triangular factors. In these parallel algorithms number of multiplication/division in $O(\log^3 n)$, number of addition/subtraction in $O(\log^3 n)$ and the storage in $O(\log^2 n)$ may be achieved.

Key words: Parallel algorithms, sparse linear systems, cholesky method, hypercubes, symbolic factorization

INTRODUCTION

In this research we will explain the method of representing a sparse matrix in parallel by using P-RAM model. P-RAM model is a shared memory model. According to Gibbons and Rytter[6], the PRAM model will survive as a theoretically convenient model of parallel computation and as a starting point for a methodology. There are number of processors working synchronously and communicating through a common random access memory. The processors are indexed by the natural number and they synchronously execute the same program (through the central main control). Although performing the same instructions, the processors can be working on different data (located in a different storage location). Such a model is also called a single-instruction, multiple-data stream (SIMD) model. The symmetric factorization is in logarithmic time of the hypercube SIMD model[2,10].

To develop these algorithms we use method of symbolic factorization on sparse symmetric factorization. Symbolic factorization procedures a data structure that exploits the sparsity of the triangular factors. We will represent an array pattern of processing elements (PE) for sparse matrix on hypercube. We have already developed P-RAM algorithms for linear system (dense case)[7,8,9]. These algorithms are implemented on Hypercube architecture.

BACKGROUND

We consider symmetric Gaussian elimination for the solution of the system of linear Eq.

$$A x = b$$

Where A is an n×n symmetric, positive definite matrix. Symmetric Gaussian elimination is equivalent to the square root free Cholesky method, i.e., first factoring A into the product U^TDU and then forward and back solving to obtain x and we often talk interchangeably about these two methods of solving A x = b. Furthermore, since almost the entire of A, we restrict most of our attention to that portion of the solution process.

Sherman[13] developed a row-oriented U^TDU factorization algorithm for dense matrices A and considers two modifications of it which take advantage of sparseness in A and U. We also discuss the type of information about A and U which is required to allow sparse symmetric factorization to be implemented efficiently.

Here we will present a parallel algorithm for sparse symmetric matrix and will be implemented to hypercube.

Previously we have already developed matrix multiplication algorithm on P-RAM model[7] and implementation of back-substitution in Gauss-

Corresponding Author: Rakesh Kumar Katare, Department of Computer Science, A.P.S. University, Rewa (M.P.) 486003 India

elimination on P-RAM model[8]. Now we further extend this idea for sparse linear systems of Eq.s.

ANALYSIS OF SYMMETRIC GAUSS ELIMINATION

In this section we suggest the actual testing operations that will be performed during Symmetric Gaussian elimination. Basically the Gaussian elimination is applied to transform matrices of linear Eq. to triangular form. This process may be performed in general by creating zeros in the first column, then the second and so forth. For k = 1,2,...,n-1 we use the formulae

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - \left(\frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}\right) a_{kj}^{(k)}, \quad i, j > k \tag{1}$$

and

$$b_{i}^{(k+1)} = b_{i}^{(k)} - \left(\frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}\right) b_{k}^{(k)}, \quad i > k \tag{2}$$

where, $a_{ij}^{(1)} = a_{ij}$, i, j = 1, 2,…,n. The only assumption required is that the inequalities $a_{kk}^{(k)} \neq 0$, k = 1,2,..., n hold. These entries are called pivot in Gaussian elimination. It is convenient to use the notation,

$$A^{(k)}x = b^{(k)}$$

For the system obtained after (k-1) steps, k = 1, 2,..., n with $A^{(1)} = A$ and $b^{(1)} = b$. The final matrix $A^{(n)}$ is upper triangular matrix[3]

As we know that in $a_{ij}^{(k+1)} = a_{ij}^{(k)} - \left(\frac{a_{ik}^{(k)}}{a_{ik}^{(k)}}\right) a_{jk}^{(k)}$ we are eliminating position $a_{ik}^{(k)}$

Case 1: If $i \neq j$ then the above expression can be written as follows:

$$\left(a_{ij}^{(k)} - a_{ij}^{(k+1)}\right) = \left(\frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}\right) a_{jk}^{(k)} \tag{3}$$

Lemma 1: if $(a_{ij}^{(k)} - a_{ij}^{(k+1)}) < 0$ then $a_{ik}^{(k)}$ or $a_{jk}^{(k)}$ is negative

Proof: $(a_{ij}^{(k)} - a_{ij}^{(k+1)}) < 0$

$$\Rightarrow \left(\frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}\right) a_{jk}^{(k)} < 0$$

$\Rightarrow a_{ik}^{(k)}$ or $a_{jk}^{(k)}$ is negative

$$\Rightarrow a_{ij}^{(k+1)} = a_{ij}^{(k)} + \left(\frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}\right) a_{jk}^{(k)}$$

Lemma 2: If $(a_{ij}^{(k)} - a_{ij}^{(k+1)}) > 0$ then $a_{jk}^{(k)}$ or $a_{ik}^{(k)}$ both negative or both positive and for pivot element $a_{ij}^{(k)}$ is always positive.

Proof: -

$$\left(a_{ij}^{(k)} - a_{ij}^{(k+1)}\right) > 0 \Rightarrow \left(\frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}\right) a_{jk}^{(k)} > 0$$

$\Rightarrow a_{ij}^{(k)} > a_{ij}^{(k+1)}$ then $a_{ij}^{(k+1)}$ is positive

$\Rightarrow a_{jk}^{(k)}$ or $a_{ik}^{(k)}$ both negative or both positive and for pivot element $a_{ij}^{(k)}$ is always positive.

Lemma 3: If $(a_{ij}^{(k)} - a_{ij}^{(k+1)}) = 0$ then no elimination will take place and one of the following condition will be satisfied.

(i) $a_{ik}^{(k)} = 0$ or (ii) $a_{jk}^{(k)} = 0$ or both (i) and (ii) are zero.

Proof:

if $(a_{ij}^{(k)} - a_{ij}^{(k+1)}) = 0$ then $a_{ij}^{(k)} = a_{ij}^{(k+1)}$ and $\left(\frac{a_{ij}^{(k)}}{a_{kk}^{(k)}}\right) a_{jk}^{(k)} = 0$

$\Rightarrow (a_{ik}^{(k)}.a_{jk}^{(k)}) = 0$

$\Rightarrow a_{ik}^{(k)} = 0$ or $a_{jk}^{(k)} = 0$ or both are zero.

$\Rightarrow a_{ik}^{(k)}$ will remain same after elimination i.e. no elimination is required

Case 2: if i = j then

$$a_{ii}^{(k+1)} = a_{ii}^{(k)} - \left(\frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}\right) a_{ik}^{(k)}$$

$$\Rightarrow a_{ii}^{(k+1)} = a_{ii}^{(k)} - \left(\frac{\left(a_{ik}^{(k)}\right)^2}{a_{kk}^{(k)}}\right)$$

Lemma 4: If $a_{ii}^{(k+1)} = 0$ then $a_{ik}^{(k)} = \sqrt{a_{ii}^{(k)}.a_{kk}^{(k)}}$

Proof: If $a_{ii}^{(k+1)} = 0$ then $a_{ii}^{(k)} = \left(\frac{\left(a_{ik}^{(k)}\right)^2}{a_{kk}^{(k)}}\right) \Rightarrow a_{ii}^{(k)}.a_{kk}^{(k)} = \left(a_{ik}^{(k)}\right)^2 \Rightarrow a_{ik}^{(k)} = \sqrt{a_{ii}^{(k)}.a_{kk}^{(k)}}$

Lemma 5: If $a_{ik}^{(k)} = 0$ then $a_{ii}^{(k+1)} = a_{ii}^{(k)}$ no iteration will take place.

Proof: The result is obvious.

Lemma 6: If $\left(a_{ii}^{(k)} - a_{ii}^{(k+1)}\right) > 0$ then $a_{ik} > 0$

Proof:

$$\left(a_{ii}^{(k)} - a_{ii}^{(k+1)}\right) > 0 \Rightarrow \left(\frac{\left(a_{ik}^{(k)}\right)^2}{a_{kk}^{(k)}}\right) > 0 \Rightarrow \left(a_{ik}^{(k)}\right)^2 > a_{kk}^{(k)}$$

$$\Rightarrow \left(a_{ik}^{(k)}\right)^2 > 0$$

Lemma 7: If $(a_{ii}^{(k)}-a_{ii}^{(k+1)}) = 0$ then $a_{ik} = 0$

Proof: $(a_{ii}^{(k)}-a_{ii}^{(k+1)}) = 0 \Rightarrow a_{ii}^{(k)} = a_{ii}^{(k+1)}$

$$\Rightarrow \frac{\left(a_{ik}^{(k)}\right)^2}{a_{kk}^{(k)}} = 0$$

$$\Rightarrow a_{ii}^{(k)} = 0$$

Lemma 8: if $(a_{ii}^{(k)}-a_{ii}^{(k+1)}) < 0$ then $a_{kk}^{(k)}$ will be eliminated.

Proof:
$(a_{ii}^{(k)}-a_{ii}^{(k+1)}) < 0 \Rightarrow a_{ii}^{(k)} > a_{ii}^{(k+1)} \Rightarrow a_{ii}^{(k+1)} = a_{ii}^{(k)} - \left(\frac{\left(a_{ik}^{(k)}\right)^2}{a_{kk}^{(k)}}\right)$

In Gauss elimination, a nonzero position in position jk, implied that there was also a nonzero position in kj, which would cause fill to occur in position i, j, when row k is used to eliminate the element in position ik[4].

Here we present the following analysis when the matrix is symmetric and positive definite.

When $i \neq j$ in Eq. 3 and $\left(a_{ij}^{(k)} - a_{ij}^{(k+1)}\right)$ is less than 0 then the elements $a_{ik}^{(k)}$ or a_{jk}^{k} is negative before elimination and when in Eq. 3 $\left(a_{ij}^{(k)} - a_{ij}^{(k+1)}\right)$ is greater than 0 then $a_{jk}^{(k)}$ or $a_{ik}^{(k)}$ both are negative or both are positive and for pivot element $a_{ij}^{(k)}$ is always positive and when in Eq. 3 $\left(a_{ij}^{(k)} - a_{ij}^{(k+1)}\right)$ is equal to zero then no elimination will take place and one of the following condition will be satisfied

(a) $a_{ik}^{(k)} = 0$ or (b) $a_{jk}^{(k)} = 0$ or both (a) and (b) will be zero.

When i = j in the above explanation of Eq. 3 then for the first case the pivot element will be eliminated and for the second case the element which is to be eliminated will be greater than zero and for the third case elimination will not take place.

We suggested testing operation for variation occurring in the elements of Symmetric Gaussian elimination. There are more testing operations rather than arithmetic operations, so that the running time of the algorithms could be proportional to the amount of testing rather than amount of arithmetic operation, which will cause symbolic factorization to occur. To avoid this problem Sherman pre computed the sets ra_k, ru_k, cu_k and showed that in implementation of this scheme, the total storage required is proportional to the number of non zeroes in A and U and that the total running time is proportional to the number of arithmetic operations on nonzero. In addition to this the preprocessing required to compute the sets { ra_k} { ru_k} and {cu_k} can be performed in time proportional to the total storage. To see how to avoid these problems, let us assume that for each k, $1 \leq k \leq N$, we have pre-computed:

- The set ra_k of columns $j \geq k$ for which $a_{kj} \neq 0$;
- The set ru_k of columns $j > k$ for which $u_{kj} \neq 0$;
- The set cu_k of columns $i < k$ for which $u_{ik} \neq 0$;

Line

1. For k ← 1 to N do
2. [m_{kk} ←1;
3. d_{kk} ← a_{kk};
4. For j ϵ ru_k do
5. [m_{kj}← 0];
6. For j ϵ {n ϵ ra_k : n > k} do
7. [m_{kj} ← a_{kj}];
8. For i ϵ cu_k do
9. [t ← m_{ik};
10. m_{ik} ← m_{ik} / d_{ii};
11. d_{kk} ← d_{kk}-t. m_{ik};
12. For j ϵ {n ϵ ru_i : n >k} do
13. [m_{kj} ← m_{kj}-$m_{ik.}$ m_{ij}]]];

Comment: Now M = U

Row-oriented Sparse U^TDU Factorization (with pre-processing) [Source 13].

Algorithm 1

In Algorithm 1 only entries of M, which are used are those corresponding to nonzeroes in A or U. An implementation for Algorithm 1, it is already shown that the total storage required is proportional to the number of nonzeroes in A and U and that the total

running time is proportional to the number of arithmetic operations on nonzeroes. In addition, A.H. Sherman[13] showed that the pre-processing required to compute the sets $\{ra_k\}$, $\{ru_k\}$, $\{cu_k\}$ can be performed in time proportional to the total storage.

SYMBOLIC FACTORIZATION

The sets $\{ra_k\}$, which describe the structure of A, are input parameters and the symbolic factorization algorithm, computes the sets $\{ru_k\}$ from them. The sets $\{cu_k\}$ could be computed from the sets $\{ru_k\}$, at the k-th step of the symbolic factorization algorithm, ru_k is computed from ra_k and the sets ru_i for $i < k$. An examination of Algorithm 1 shows that for $j > k$, $u_{kj} \neq 0$ if and only if either

i. $a_{kj} \neq 0$ or
ii. $u_{ik} \neq 0$ for some $i \in cu_k$.

Thus letting

$ru^k_i = \{j \in ru_i : j > k\}$,

we have $j \in ru_k$ if and only if either

iii. $j \in ra_k$ or
iv. $j \in ru^k_i$ for some $i \in cu_k$.

Algorithm 2, 3 are a symbolic factorization algorithm based directly on Algorithm 1. At the k-th step, ru_k is formed by combining ra_k with sets $\{ ru^k_i\}$ for $i \in cu_k$. However, it is not necessary to examine ru^k_i for all rows $i \in cu_k$. Let l_k be the set of rows $i \in cu_k$ for which k is the minimum column index in ru_i. Then we have the following result, which expresses a type of transitivity condition for the fill-in in symmetric Gaussian elimination.

There are some important reasons why it is desirable to perform such a symbolic factorization.

- Since a symbolic factorization produce a data structure that exploits the sparsity of the triangular factors, the numerical decomposition can be performed using a static storage scheme. There is no need to perform storage allocations for the fill-in during the numerical computations. This reduces both storage and execution time overheads for the numerical phase[4,5].
- We obtained from the symbolic factorization a bound on the amount of sparse we need in order to solve the linear system. This immediately tells us if the numerical computation is feasible. (of course, this is important only if the symbolic factorization can be performed both in terms of storage and execution time and if the bound on space is reasonably light)[4,5]

```
1.    For k ← 1 to N do
2.        [ru_k ←0];
3.    For k ← 1 to N-1 do
Comment: Form ru_k by set Unions
4.            [For j ∈ {n ∈ ra_k : n > k} do
5.                [ru_k ← ru_k ∪ {j}];
6.            For i ∈ cu_k do
7.                [For j ∈ ru_i^k do
8.                [if j ∉ ru_k then
9.                ru_k ← ru_k ∪ {j}]]]];
```

$O(\theta_A)$ symbolic factorization [Source 13]

Algorithm 2

```
1.    For k ← 1 to N do
2.        [ru_k ← 0;
3.        l_k ← 0];
4.    For k ← 1 to N-1 do
Comment: Form ru_k by set Unions
5.        [For j ∈ {n ∈ ra_k : n > k} do
6.            [ru_k ← ru_k ∪ {j}];
7.        For i ∈ l_k do
8.            [For j ∈ ru_i^k do
9.                [if j ∉ ru_k then
10.                   [ru_k ← ru_k ∪ {j}]]];
11.               m ← min {j: j∈ ru_k ∪ {N+1}}
12.               If m < N+1 then
13.                   [l_m ← l_m ∪ {k}]];
```

$O(\theta_s)$ symbolic factorization Algorithm[Source 13]

Algorithm 3

PARALLEL MATRIX ALGORITHM

By an efficient parallel algorithm we mean one that takes polylogarithmic time using a polynomial number of processors. In practical terms, at most a polynomial number of processors is reckoned to be feasible[6]. A polylogarithmic time algorithm takes $O(\log^k n)$ parallel time for some constant integer k, where n is the problem size. Problems which can be solved within these constraints are universally regarded as having efficient parallel solutions and are said to belong to the class NC(Nick Pippenger's Class).

Representation of Array Pattern of Processing Elements (P.Es.): Consider a case of three dimensional array pattern with $n^3 = 2^{3q}$ (Processing Elements) PEs.

Conceptually these PEs may be regarded as arranged, in n×n×n array pattern. If we assume that the PEs are row major order, the PE (i,j,k) in position (i,j,k) of this array has index in^2+jn+k (note that array indices are in the range[0, (n-1)]. Hence, if $r_{3q-1},.....,r_0$ is the binary representation of the PE position (i,j,k) then i = $r_{3q-1},....,r_{2q}$, j = $r_{2q-1},...,r_q$, k = $r_{q-1},....,r_0$ using A(i,j,k), B(i,j,k) and C(i,j,k) to represent memory locations in P(i,j,k), we can describe the initial condition for matrix multiplication as:

$$A(0,j,k) = A_{jk}$$
$$B(0,j,k) = B_{jk},\ 0 <= j < k,\ 0 <= k < n$$

A_{jk} and B_{jk} are the elements of the two matrices to be multiplied. The desired final configuration is

$$C(0,j,k) = C(j,k),\ 0 <= j < n,\ 0 <= k < n$$

Where,

$$C_{jk} = \sum_{l=0}^{n-1} A_{jl} B_{lk} \qquad (4)$$

This algorithm computes the product matrix C by directly making use of (4). The algorithm has three distinct phases. In the first, element of A and B are distributed over the n^3 PEs so that we have $A(l,j,k) = A_{jl}$ and $B(l,j,k) = B_{lk}$. In the second phase the products $C(l,j,k) = A(l,j,k) * B(l,j,k) = A_{jl}B_{lk}$ are computed. Finally, in third phase the sum $\sum_{l=0}^{n-1} C(l,j,k)$ are computed.

The details are spelled out in Dekel, Nassimi and Sahni 1981. In this procedure all PE references are by PE index (Recall that the index of PE(i,j,k) as in^2+jn+k). The key to the algorithm of Dekel, Nassimi and Sahni[2] in the data routing strategy 5q = 5 log n routing steps are sufficient to broadcast the initial value through the processor array and to confine the results.

The array pattern of processing elements (PE) in row-major order for the specialized sparse matrix (symmetric)[12] can be formulated in the following manner.

- **Representation of lower-triangular matrix:**
  Index of (a_{ij}) = Total Number of elements in first i-1 rows + Number of elements up to j^{th} column in the i^{th} row

  = i (i+1) / 2 + j (1≤i, j≤n)

- **Representation of upper-triangular matrix:**
  Index of (a_{ij}) = Number of elements up to a_{ij} element = (i-1)×(n-i/2)+j (1≤i, j≤n)

- **Representation of diagonal matrix:**
  In the sparse matrices having the elements only on diagonal following points are evident:
  Number of elements in a n×n square diagonal matrix = n
  Any element a_{ij} can be referred as processing element using the formula
  Address (a_{ij}) = i[or j]

- **Representation of tri-diagonal matrix:**
  Index of (a_{ij}) = Total number of elements in first (i-1) rows + Number of elements up to j^{th} Column in the i^{th} Row = 2+2 x (i-2)+j (1≤i, j≤n) (1≤i, j≤n)

- **Representation of αβ-band matrix:**

Case 1: 1≤i≤β
  Index of (a_{ij}) = Number of elements in first $(i-1)^{th}$ row + Number of element in i^{th} row up to j^{th} column

  = α × (i-1) + ((i-1) (i-2)) / 2 + j

Case 2: β < i ≤ (n-α+1)
  Index of (a_{ij}) = Number of elements in first β row + Number of elements between $(\beta+1)^{th}$ row and $(i-1)^{th}$ row + Number of elements in i^{th} Row

  = αβ + (β(β-1))/2+(α+β-1)(i-β-1)+j-i+β

Case 3: n-α+1<i
  Index of (a_{ij}) =
  Number of elements in first (n-α+1) rows + Number of elements after $(n-\alpha+1)^{th}$ row and up to $(i-1)^{th}$ row + Number of elements in i^{th} Row and unto j^{th} column

= αβ+(β(β-1))/2+(α+β-1) (n-α-β+1)+(α+β) (i-n+α−1)-((i-n+α-1)×(i-n+α-2)) / 2+ 1

Representation of array pattern of processing elements (PE) for lower triangular matrix is presented on a hypercube model. Hypercubes are loosely coupled parallel processors based on the binary n-cube network. A n-cube parallel processor consists of 2^n identical processors, each provided with its own sizable memory and inter connected with n neighbors[1,14,15]. This architecture consists of a large number of identical

processors inter connected to one another according to some convenient pattern. In a shared memory system, processors operate on the data from the common memory, each processor reads the data it needs, performs some processing and writes the results back in memory. In a distributed memory system inter processor communication is achieved by message passing and computation of data driven (although some designs incorporate a global bus, this does not constitute the main way of inter communication). By message passing it is meant that data or possibly code are transferred from processor A to processor B by traveling across a sequence of nearest neighbor nodes starting with node A and ending with B, synchronization is driven by data in the sense that computation in some node is performed only when its necessary data are available. The main advantage of such architectures, often referred to as ensemble architectures, is the simplicity of their design. The nodes are identical, or are of a few different kinds and can therefore be fabricated at relatively low cost. The model can easily be made fault tolerant by shutting down failing nodes.

The most important advantages of this class of design is the ability to exploit particular topologies of problem or algorithms in order to minimize communication costs. A hypercube is a multidimensional mesh of nodes with exactly two nodes in each dimension. A d-dimensional hypercube consists of K nodes, where $K = 2^n$.

- A hypercube has n special dimensions, where n can be any positive integer (including zero) [1],[15].
- A hypercube has 2^n vertices [11].
- There are n connections (lines) that meet at each vertex of a hypercube [11].
- All connections at a hypercube vertex meet at right angles with respect to each other[1],[14].
- The Hypercube can be constructed recursively from lower dimensional cubes.
- An architecture where the degree and diameter of the graph is same than they will achieve a good balance between, the communication speed and the complexity of the topology network. Hypercube achieve this equality, which explains why they are one of the today's most popular design (e.g. i psc of intel corp., T-series of FPS, n-cube, connection machine of thinking machines corp.)[11,15].

When the lower triangular matrix is presented in three dimension then the PE's are indexed in the following manner.

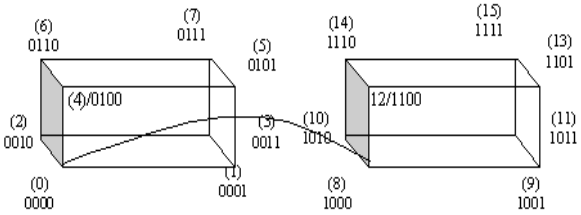

Fig. 1: Mapping of lower triangular matrix on hypercube

$$a_{(i,j,k)} = \frac{i(i+1)}{2} n^2 + jn + k$$

For different values of i and j we can map Hypercube. Here we are representing mapping of Hypercube for a single value of i and j by using functions BIT and BIT- COMPLIMENT. (Fig. 1).

A. i = 0, j = 0 a(i,j) = 0,k = 0

BIT (0000, 0) = 0, BIT-COMPLEMENT (0000, 0) = 0001 = (1)
BIT (0000, 1) = 0, BIT-COMPLEMENT (0000, 1) = 0010 = (2)
BIT (0000, 2) = 0, BIT-COMPLEMENT (0000, 2) = 0100 = (4)
BIT (0000, 3) = 0, BIT-COMPLEMENT (0000, 3) = 1000 = (8)

Now we conclude the following results

Lemma 9: The routing or access function in^2+jn+k contains topological properties.

Proof: This access function is a polynomial of 3 dimensional discrete space. Where i, j, k are 3 dimensions and n is fixed. It gives a relationship of processing elements (i.e. there are 2^3 connections) that meet at each vertex of a hypercube means that the algorithm can be evaluated in polylogarithmic time using a polynomial (in^2+jn+k) of three dimensional discrete space. We can easily construct hypercube successively from lower dimensional cubes by using polynomial in^2+jn+k. A discrete space is a topological space in which all sets are isolated. We conclude that the access function by which we are mapping matrix elements will be pairwise continuous. It is shown in the implementation that because of the hamming distance between the processes the hypercube is modeled as a discrete space with discrete time. This access function is also used to map a matrix of three dimensions into RAM sequentially.

Lemma 10: Cost of storage and cost of retrieval of a matrix are proportional to each other in polylogarithmic parallel time using P-RAM with a polynomial number

of processor.

Proof: For storage and retrieval of a matrix we use parallel iteration. Parallel iteration has the property of convergence in log n in parallel. It converge geometrically or at the rate of geometric progression therefore they are proportional to each other for a single value. From the above fact we can write that cost of retrieval is proportional to cost of storage

$\Rightarrow$ cost of retrieval = k x cost of storage, (Where k is a constant)

$\Rightarrow$ if $k \geq 1$ then it is a Dense matrix and if $k < 1$ then it is a sparse matrix.

Here we are representing PRAM-CREW Algorithm.

Row oriented sparse U^TDU factorization (With pre-processing) PRAM-CREW Algorithm:

Begin
Repeat log n times do
For all (ordered) pair (i, j, k), $0< k \leq n$, $0<i = j = k \leq n$
And q = log n in parallel do
$m(2^{2q}i+2^qj+k) = m(k, k)$
$d(2^{2q}i+2^qj+k) = d(k, k)$
$a(2^{2q}i+2^qj+k) = a(k, k)$
$m(k, k) = 1$
$d(k, k) \leftarrow a(k, k)$
end for
For all (ordered) pair (i, j,. k) $0 < k \leq n$, $i > 0$
$j \in u(k, j)$ and q = log n
in parallel do
$m(2^{2q}k+2^qj+k) = m(k, j)$
$m(k, j) \leftarrow 0$;
end for
for all (ordered) pair (i, j, k), $0 <k\leq n$
$j \in (n \in a(k,j) : n > k)$ and q = log n in parallel do
$a(2^{2q}i+2^qj+k) = a(k, j)$
$m(k, j) \leftarrow a(k, j)$
end for
For all (ordered) pair (i, j, k) $0 <k\leq n$
$i \in (u(i, k))$ do
$m(2^{2q}k+2^qj+k) = m(i, k)$
$m(2^{2q}k+2^qj+k) = d(i, i)$
$m(2^{2q}k+2^qj+k) = m(i, j)$
$t \leftarrow m(i, k)$;
$m(i, k) \leftarrow m(i, k)/d(i, i)$;
$d(k, k) \leftarrow d(k, k)$-t. m(i, k);
for all (ordered) pair (i, j, k), $0 <k\leq n$
$j \in (n \in ru_i : n > k)$ and q = log n in parallel do
$m(k, j) \leftarrow m(k, j)$-m(i, k). m(i, j)
End for
End for
End

$O(\theta_A)$ symbolic factorization PRAM-CREW Algorithm:

Begin
Repeat log n times do
For all (ordered) pair (i, j, k), $0< k \leq n$, $(j > k : u_{k,j} \neq 0)$
and q = log n in parallel do
u (k, j) $\leftarrow$ 0
End for
For all (ordered) pair (i, j, k), $0 < k < n$ and q = log n
in parallel do
For all (ordered) pair (i, j, k), $0< k < n$,
$j \in (n \in a(k, j): n > k)$ do
$u(k, j) \leftarrow u(k, j) \cup \{i\}$
end for
for all (ordered) pair (i,j, k), $0 <k\leq n$
$i \in cu_k$ and q = log n in parallel do
For all (ordered) pair (i. j. k) $0 <k\leq n$
$j \in ru_i$ and q = log n in parallel do
if $j \notin u(k, j)$ then
$u(k, j) \leftarrow u(k, j) \cup \{i\}$
End if
End for
End for
End.

Since we have developed parallel algorithm for sparse linear systems therefore it is not required to discuss storage for sequential algorithms. Although the storage requirements has been discussed in very short here in this research. The storage schemes used for sparse matrices consists of two facts, primary storage used to hold the numerical values and overhead storage, used for pointers, subscripts and other information needed to record the structure of the matrix. The data structures involved for these two different strategies may be compared. The elements of the upper triangle (excluding the diagonal) are stored row by row in a single array with a parallel array holding their column subscripts. A third array indicates the position of each row and a fourth array contains the sophistication of the storage scheme increases. Exploiting more and more zeros, the primary storage decreases, but the overhead usually increases. There is usually a point where it pays to ignore some zeros, because the overhead storage required to exploit them far more than the decrease in primary storage[5].

CONCLUSION

The above discussion proves the values of the elements of the matrix. The Gauss elimination for

symmetric, positive definite matrix for share memory has been studied. The classical problem for sequential algorithm for U^TDU factorization of a matrix A was computed by A. Sherman. The results are given as follows: $\theta_S(A) \approx O(n^2), \theta_M(A) \approx O(n^3), \theta_A(A) \approx O(n^3)$ where θ_S, θ_M and θ_A denote the storage, the number of multiplication/division and addition/subtraction respectively. The matrix multiplication (SIMD-Hypercube) example of Dekel, Nassimi and Sahni 1981 is extended to sparse linear systems now. Consider a cube connected computer with n^3 PEs. Conceptually, these PEs may be regarded as arranged in an n×n×n array pattern. If it is assumed that the PEs are indexed in row-major order, the PE, PE(i,j,k) in position (i,j,k) of this array has index in^2+jn+k (note that array indices are in the range[0, n-1]). Hence, if $r_{3q-1},...,r_0$ is the binary representation of the PE position (i,j,k) then i = $r_{3q-1},...,r_{2q}$, j = $r_{2q-1},...,r_q$ and k = $r_{q-1},...,r_0$. In mapping of data into the hypercube it was indicated that the data is mapped to its all possible neighbor processors in the n-cube which has hamming distance exactly by one bit, which makes like a tree structure of having leaf of all its possible dimensions (i.e. for n-cube the tree has n leaf). The complexity of these parallel algorithm is $O_S(A) = O(\log^2 n)$, $O_M(A) = O(\log^3 n)$ and $O_A(A) = O(\log^3 n)$ and the number of PE are as shown in Fig. 2 for the case of lower triangular matrix (i.e. only (i(i+1)/2) PE are required). Same way for upper triangular matrix only (i-1) (n-i)/2) + j, processors for diagonal matrix on i (or j) number of processors, for tri-diagonal matrix only 2 + 2 x (i-2) + j no. of processors and for αβ-band matrix only no of processors for Case 1 is $\alpha \times (i-1) + ((i-1)(i-2)) / 2 + j$ for Case 2 is $\alpha\beta+(\beta(\beta-1))/2+(\alpha+\beta-1)(i-\beta-1)+j-i+\beta$ and for Case 3is$\alpha\beta+(\beta(\beta-1))/2+(\alpha+\beta-1)(n-\alpha-\beta+1)+(\alpha+\beta)(i-n+\alpha-1)-((i-n+\alpha-1)\times(i-n+\alpha-2))/2+1$ is required to calculate U^TDU factors. Based on the above concept we developed the following Algorithm. Row-oriented dense U^TDU factorization PRAM-CREW Algorithm, Row oriented sparse U^TDU factorization (with zero testing). PRAM-CREW Algorithm, Row oriented sparse U^TDU factorization (with pre-processing PRAM-CREW Algorithm and $O(\theta_A)$ symbolic factorization PRAM-CREW Algorithm respectively. It has been shown that the access function or routing function to map data on hypercube contains topological properties. This function is convergent in the finite interval. The hypercube is modeled as a discrete space with discrete time because the processor's are in Hamming distances, where as hypercube is an undirected graph consisting of $n = 2^k$ vertices, if and only if the binary representation of their labels differ by one and only one bit.

j i	0	1	2	3
0	0 0000			
1	1 0001	2 0010		
2	3 0011	4 0100	5 0101	
3	6 0110	7 0111	8 1000	9 1001

Fig. 2: Array view of a 16 PE for a lower triangular matrix. Each square matrix represents a PE. The number in a square is the PE index (both decimal and binary representation are provided)

The most important property is that the degree of the graph and the diameter are always equal, which will achieve a good balance between the communication speed and the complexity of the topology network. These structures are restricted to having exactly 2^k nodes. Because structure sizes must be a power of 2, there are large gaps in the sizes of the system that can be built with the hypercubes. This severely restricts the number of possible nodes. A hypercube architecture has a delay time approximately equal to 2 log n and has a skew, i.e. different delay times for different inter connecting nodes[11].

REFERENCES

1. Ammon, J., Hypercube Connectivity within cc NUMA architecture Silicon Graphics, 201LN. Shoreline Blvd. Ms 565, Mountain View, CA 94043
2. Dekel E.D., Nassimi and Sahni, S. 1981. Parallel matrix and graph algorithms. SIAM. J. Comput., 10 (4): 657-675.
3. Duff, I., A. Erisman and Reid , J.K. 1987. Direct Methods for Sparse Matrices. Oxford University Press. Oxford, UK.
4. George, A. and Esmond, N.G. 1987. Symbolic factorization for sparse guassion elimination with partial pivoting. SIAM. J. Sci. Stat., Comput., 8 (6) 877-898.

5. George, A., J. Liu and Ng, E. 2003. Computer solution of sparse positive definite systems, unpublished Book. Department of Computer Science, University of Waterloo.
6. Gibbons, A. and Rytter, W. 1988. Efficient Parallel Algorithms. Cambridge University Press, Cambridge.
7. Katare, R.K. and Chaudhari, N.S. 1999. Formulation of matrix multiplication on P-RAM model. N. Delhi, Proc. of International Conference on Cognitive Systems.
8. Katare, R.K. and Chaudhari, N.S. 1999. Implementation of back substitution in Gauss-elimination on P-RAM model. Rewa, Indian Mathematical Society, Proc of 65th Annual Conference.
9. Katare, Rakesh Kumar 2000. Some P-RAM algorithms for linear systems. M. Tech Dissertation, Devi Ahilya University, Indore.
10. Quinn, M.J., 1994. Parallel computing. Mc Graw-hill INC.
11. Saad, Y. and Schultz, M.H. 1988. Topological properties of Hypercubes. IEEE Trans. Comput., 37 (7): 867-872.
12. Samanta, D., 2001. Classic data structures. PHI, New Delhi.
13. Sherman, A., 1975. On the efficient solution of sparse systems of linear and nonlinear Eq.s. Ph.D. Thesis, Yale University, New Haven, CT.
14. Tamiko, Teil, 1994. The design of the connection machine. Design Issues, 10 (1).
15. YVES ROBERT, The Impact of Vector and Elimination Algorithm. Halsted press, A division of John Wiley and sons, New York.