COST-OPTIMAL GAUSSIAN ELIMINATION ALGORITHM ON RCC-FULL

Ashutosh Vyas, Raghvendra Maloo and P. Gupta
email: {ashu, rmaloo, pg}@iitk.ac.in
Department of Computer Science & Engineering
Indian Institute of Technology,
Kanpur-208 016, India
Phone & Fax No: 0091 512 597647

ABSTRACT

In this paper, we have proposed a cost-optimal parallel algorithm to find the solution of n simultaneous linear equations on RCC-FULL $\lambda(N, L)$ network. The algorithm is designed following the strategy of the sequential Gaussian Elimination algorithm. The algorithm utilizes strength of the RCC-FULL network which helps to reduce the time complexiety and has the time complexity of $O(n)$ with $O(n^2)$ processors.

1 INTRODUCTION

Gaussian Elimination is classical well known method for solving system of linear equations. The method requires $O(n^3)$ number of operations where n is number of unknowns to be determined. Number of operations can be reduced significantly in case of a sparse system [2] by suitable data structure taking advantage of the sparsity. If the system is dense it requires $O(n^2)$ divisions, $O(n^3)$ multiplications and additions for computing the upper triangular matrix. However, computation time can be reduced by parallel processing. But the phenomenon of cost optimality will come into picture. A cost optimal parallel algorithm can be designed by selecting a proper model. In this paper we have proposed a parallel algorithm to solve a system of simultaneouse n linear equations on RCC-FULL network [3]. The algorithm is designed based on the classical sequential Gaussian Elimination algorithm. The algorithm has the time complexity of $O(n)$ time with $O(n^2)$ processors and is cost-optimal.

In section 2 the model of computation RCC-FULL network has been discussed. Section 3 gives overview of sequential algorithm for Gaussian Elimination. In section 4 the cost-optimal parallel algorihm has been presented. An example has been considered in section 5 to illustrate the example. Finally conclusion is given in section 6.

2 MODEL OF COMPUTATION : RCC-FULL

In this section we describe the model of computation which is a recusively compounded graph proposed by Hamid and Hall [3]. RCC-FULL is constructed incrementally by systematically connecting together a number of basic atoms. A basic atom is a set of fully interconnected nodes. An RCC-FULL network (λ) is defined by two parameters: number of processing elements in basic atom (N) and level of recursion (L). Let $\lambda(N, L)$ be a network having N processing elements in a basic atom and L level of recursion. The network $\lambda(N, 0)$ is a complete graph of N processing elements e.g. $\lambda(2, 0)$ is shown in Figure 1.

The network $\lambda(N, L)$ is constructed using $N^{2^{L-1}}$ copies of $\lambda(N, L-1)$ network.It can be viewed as $N^{2^{L-1}} \times N^{2^{L-1}}$ matrix, by treating each $\lambda(N, L-1)$ as a row. Each node in $\lambda(N, L)$ is specified by an m bit binary number where $m = 2^L \log N$ bits. The most significant $2^{L-1} \log N$ bits identify the $\lambda(N, L-1)$ network that this node belongs to and the least significant $2^{L-1} \log N$ bits identify the node within the $\lambda(N, L-1)$. The links between these $\lambda(N, L-1)$ networks are called as L-transpose

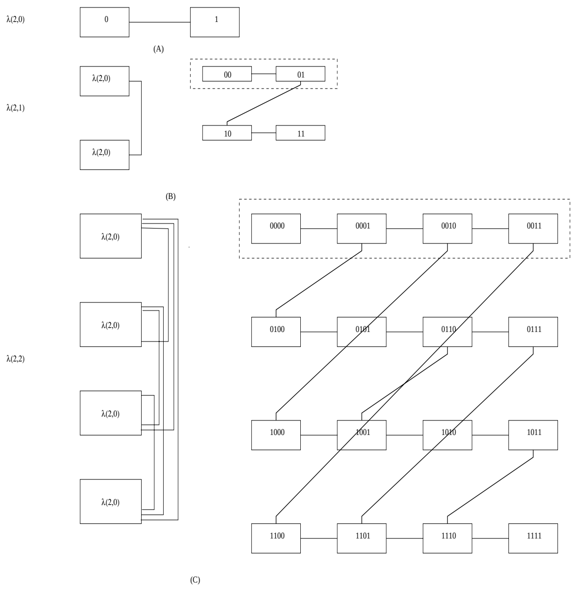

Figure 1. RCC-FULL for different N and L

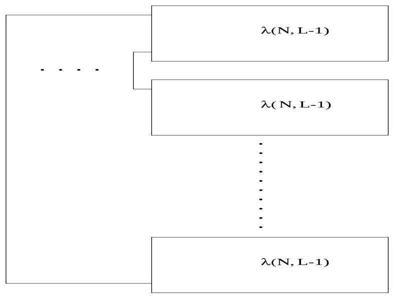

Figure 2. Recursive RCC-FULL $\lambda(N, L)$

links as shown in Figure 2 and are formed by connecting PE_{ij} to $PE_{j,i}$, with $i \neq j$, where i and j are binary numbers of length $2^{L-1} \log N$. Figure 2 shows view of $\lambda(N, L)$ network in terms of $\lambda(N, L-1)$ and interconnection among them. The diameter of the network $\lambda(N, L)$ is $\phi(N, L) = 2^{L+1} - 1$ while the degree of the network is $\Delta(N, L) = L + N - 1$. It is shown in [3] that the RCC-FULL network is equally powerful like hypercube with less interconnection.

3 SEQUENTIAL GAUSSIAN ELIMINATION ALGORITHM

Suppose we have sytem of n equations each consists of same n variables as follows:

$$\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \ldots\ldots + a_{1n}x_n &= a_{1(n+1)} \\
a_{21}x_1 + a_{22}x_2 + \ldots\ldots + a_{2n}x_n &= a_{2(n+1)} \\
&. \\
&. \\
a_{n1}x_1 + a_{n2}x_2 + \ldots\ldots + a_{nn}x_n &= a_{n(n+1)}
\end{aligned}$$

Gaussian Elimination method does solution in two phases. The first phase reduces the system of linear equtaions to an equivalent upper triangular form **Ux**=**c**, where **U** is an $n \times n$ upper triangular matrix and **c** is a vector of length n. In the second phase the system of equations of this form is solved by Back Substitution. The sequential Gaussian Elimination algorithm [4] may be written as follows.

Phase I- Upper Triangularization

```
for ($k = 1$; $k < n$; $k$++)
    for ($i = k$+1; $i \leq n$; $i$++)
        $u = a_{ik}/a_{kk}$;
        for ($j = k$; $j \leq n + 1$; $j$++)
            $a_{ij} = a_{ij}$ - $u * a_{kj}$;
```

Phase II- Back Substiution

```
$x_n = a_{n(n+1)}/a_{nn}$
for ($i = n - 1$; $i \leq 1$; $i$- -)
    $sum = 0$;
    for ($j = i + 1$; $j \leq n$; $j$++)
        $sum = sum + a_{ij}$ * $x_j$;
        $x_i = (a_{i(n+1)} - sum)/a_{ii}$;
```

Algorithm: Gaussian Elimination

4 COST-OPTIMAL ALGORITHM ON RCC-FULL

In this section we describe a cost-optimal parallel algorithm to solve the simultaneous linear equations on RCC-FULL $\lambda(N, L)$. It is based on the strategy used in sequential Gaussian Elimination algorithm which has been discussed in the previous section. The algorithm uses the power of interconnection network of RCC-FULL in such a way that the algorithm becomes cost optimal. Algorithm exploits some characteristic operations of RCC-FULL such as *broadcast* and *transpose* to acheive the desired efficiency. Transpose on this network takes O(1) time and complexity of broadcast depends on the diameter of the network.

We assume, RCC-FULL $\lambda(N, L)$ network with $(n+1)^2$ processors such that $n+1 = N^{2^{L-1}}$. All these processors can be thought in the form of a two dimensional $(n+1) \times (n+1)$ array. Let the processor P_x lie in the j-th column of the i-th row and be logically indexed as P_{ij}. Observe that x can be expressed by $2\log(n+1)$ bits. Most significant $\log(n+1)$ bits form i while least significant $\log(n+1)$ bits give j. Each processor P_{ij} has at least two registers R_s and R_r. Initially, the coefficient a_{ij} is available in R_s register of P_{ij}.

4.1 Some Basic Operations

Some important operations used in the main algorithm are described below. In all these operations it is assumed that n equations are available to solve and each equation has $n+1$ coefficients, hence set of equations form $(n+1) \times (n+1)$ matrix (with one dummy equation). Each row of this matrix represent one equation.

4.1.1 Row Broadcast Operation

Broadcast (register R) operation is used to broadcast the content of the register R of a processor to all the processors lying in that row. Since a row of RCC-FULL $\lambda(N, L)$ is also a RCC-FULL $\lambda(N, L-1)$, so the broadcast in a row is same as broadcast in a whole RCC-FULL with just one less level.

To broadcast the content of a specified register of processor P_{ij}, the processor first broadcasts the data in its basic atom, then all nodes in basic atom send data on their level 1 link. Now all the processors receiving on link 1 broadcast the data in their own basic atoms. This completes broadcast of P_i's register in 1 level RCC-FULL. Figure 3 shows broadcast operation from processor P_{00} in $\lambda(4, 1)$. Numbers on the links show the sequence in which data transfer happens. To broadcast in multilevel RCC-FULL following broadcast algorithm can be used recursively. The algorithm to broadcast data from P_{ij} in to its L level network is given below:

1. Processor P_{ij} broadcasts data in its L-1 level network.

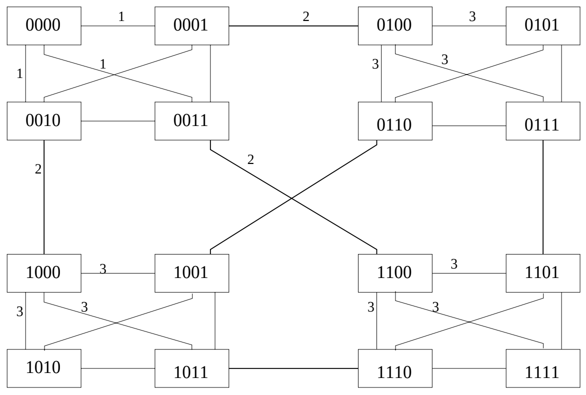

Figure 3. Broadcast operation in $\lambda(4, 1)$

2. All the processors lying in that L-1 level network of P_{ij} send the data on their respective L level links.

3. All processors receiving data from the L level links broadcast data in their respective L-1 level network.

Steps 1 and 3 broadcast data on one less level of network. So by recursive calls we arrive to level 0 and then we need only to broadcast data into the basic atoms. This operation is required to broadcast the value of a_{ii} in i-th column, to broadcast multiplication factor $\frac{a_{ii}}{a_{ji}}$ in j-th row and so on. It is easy to show that the complexity of this operation depends on the diameter of the network i.e. $O(2^L)$.

4.1.2 Parallel Exchange Operation

Parallel_exchange (register R) operation is a characteristic operation of the RCC-FULL. In this operation processors P_{ij} and P_{ji} interchanges contents of their specified register R. More clearly, this operation transposes the $(n+1) \times (n+1)$ matrix constructed by coefficients of the n equations. Since in RCC-FULL there exist a direct link between P_{ij} and P_{ji}, this operation can be performed in $O(1)$ time.

Structure of RCC-FULL is such that some operations like broadcast can be performed easily on row but not on column. Thus in order to perform some operation on the elements of column, one can follow the following steps:

1. Perform Parallel_exchange operation to bring elements of columns into their corresponding rows

2. Perform the desired operation on the elements of the row

3. Finally again perform Parallel_exchange operation to send back the elements into the column.

For example, in order to broadcast a_{ij} in j-th column, we follow the following steps:

1. Perform Parallel_exchange operation to bring a_{ij} at P_{ji}

2. Broadcast P_{ji} into j-th row

3. Finally perform Parallel_exchange operation to send back a_{ij} to P_{ij} and also to send the broadcasted a_{ij} into j-th column from j-th row

4.1.3 Right Neighbour Fetch Operation

Fetch_right_neighbour operation is used to get data by a processor from its right neighbouring processor. Processor P_{ij} gets data from its right neighbourP_{ij+1}. To perform Fetch_right_ neighbour operation, following *routing* algorithm can be used recursively. *Routing* algorithm to send data from $P_{i_1 j_1}$ to $P_{i_2 j_2}$ is given below:

1. Send data from $P_{i_1 j_1}$ to $P_{i_1 i_2}$ (in same row)

2. Send data from $P_{i_1 i_2}$ to $P_{i_2 i_1}$ (Send on L level link)

3. Send data from $P_{i_2 i_1}$ to $P_{i_2 j_2}$ (in same row)

Steps 1 and 3 send data between elements of same row and since row is nothing but a RCC-FULL of one less level, hence *routing* algorithm must be followed recursively till level reduces to 0 i.e. row reduces to basic atom. This operation is required to

1. Get computed constant from its neighbour to calculate the value of x_i's at P_{ii}. Note that the value of x_i is -(computed constant of right neighbour)/coefficient of x_i's available at P_{ii}).

2. Get computed constant from its right neighbour to calculate its own computed constant.

Computed constant of P_{ij} is $\sum_{k=j}^{n} C_{ik} * X_k + Const_i$ where C_{ik} is the coefficient of k-th variable of i-th equation, x_k is the value of the k-th variable computed at P_{kk} and $Const_i$ is the constant term of the i-th equation.

4.2 Cost-optimal Algorithm

The parallel algorithm for RCC-FULL network works as follows. First it finds the upper tringular matrix of the system of n equations in $n-1$ iterations. In i-th iteration coefficients of x_i are made zero from the last $n-i$ equations. Finally, it performs back substitution to calculate the values of variables. Back substitution phase requires n iterations.

To eliminate x_i's coefficient from rest $n-i$ equations, a_{ii} is broadcasted in i-th column ($S-1.1, S-1.2$ of the Phase I algorithm). Processors in i-th column i.e.P_{ji} for all $j > i$ calculates multiplying factor $(\frac{a_{ii}}{a_{ji}})$ for its row and this multiplying factor is broadcasted in j-th row ($S-1.3$). This factor is multiplied with coefficient of P_{jk} $\forall$ k ($S-1.4$). Now the coefficient of P_{ji} for all j is equal to a_{ii}, hence subtracting corresponding coefficient of i-th equation from j-th equation eliminates the x_i's coefficient from later equation. So to subtract i's coefficient from the last $n-i$ equations, coefficient a_{ij} is broadcasted in j-th column and subtracted from the last $n-i$ rows in that column ($S-1.5$ to $S-1.8$).

Phase I: Upper Triangularization

```
          for $i = 1$ to $n-1$ do
          /* $P_{ii}$ broadcasts $a_{ii}$ in the $i$-th row */
$S-1.1$ : Processor $P_{ii}$ does the following:
              - Initiate to broadcast $R_s$ to all
              processors $P_{ij}$ $\forall$ $j$
$S-1.2$ : Perform parallel_exchange operation to
          interchange $R_r$ contents between $P_{kl}$
          and $P_{lk}$ $\forall$ $k$, $l$
          /*Multiplying factor $\frac{a_{ii}}{a_{ji}}$ calculated for
          each row greater than $i$ and broadcast
          the factor in the corresponding row */
$S-1.3$ : For all processors $P_{ki}$,$k > i$ do the
          following:
              - Modify $R_r$ by $\frac{R_r}{R_s}$
              - Initiate broadcast $R_r$ all processor
              $P_{kl}$ for all $l$
              /* Multiply the factor to coefficients */
$S-1.4$ :     Forall processors $P_{kj}, k > i$ and forall $j$ do
                  - Modify $R_s$ by $R_s * R_r$
              /*Transpose so that $i^{th}$ row becomes
              $i$-th column*/
$S-1.5$ :     Perform parallel exchange operation to
              interchange $R_s$ contents between $P_{kl}$
              and $P_{lk}$ $\forall$ $k, l$
              /* Broadcast $j$-th coefficient of $i$-th
              equation into $j$-th row */
$S-1.6$ :     Processor $P_{ji}$ does the following:
                  - Initiate to broadcast $R_s$ to all
                  processors $P_{jk}$ $\forall$ $j$ and $k$
              /*Transpose to restore the equation into
              rows with $a_{ij}$ available in $j$-th row*/
$S-1.7$ :     Perform parallel exchange operation to
              interchange $R_s$ and $R_r$ contents between
              $P_{kl}$ and $P_{lk}$ for all $k, l$
              /*Now coefficients of $i$-th equation are
              available in corresponding columns*/
              /*Subtract $i$-th equation from rest $n-i$
              equations*/
$S-1.8$ :     Forall processors $P_{kj}, k > i$ and forall $j$ do
                  - Modify $R_s$ by $R_s$-$R_r$
```

Back-substitution is used to calculate the values of variables. In $(n-i+1)$-th iteration, P_{ii} performs Fetch_right_neighbour to get computed constant of its right neighbour and calculates the value of x_i which is - (computed constant of right neighbour)/ a_{ii} and is calculated in $S-2.1$ of the following algorithm. This value is broadcasted in i-th column so that P_{ji} for all $j < i$ can calculate its computed constant by adding a_{ji} with the computed constant of its right neighbour ($S-2.1$ to $S-2.3$). After n iterations, values of x_i, for all i become available at P_{ii}.

Phase II: Back Substitution

```
              for $i = n$ to 1 do
              /* $P_{ii}$ fetches data from its right
              neighbour and calculate $x_i$'s value*/
              /* broadcast computed value into the
              $i$-th row*/
$S-2.1$ :     Processor $P_{ii}$ does the following:
                  - Get $R_s$ of it's right neighbour
                  using Fetch_right_neighbour operation
                  - Modify $R_r$ by - $R_r/R_s$
                  - Initiate broadcast $R_r$ all
                  processors $P_{ij}$ $\forall$ $j$
              /*Transpose $i$-th row so that value of $x_i$
              becomes available in $i$-th column*/
$S-2.2$ :     Perform parallel exchange operation to
              interchange $R_r$ contents between $P_{ij}$
              and $P_{ji}$ $\forall$ $j$
              /*Calculate computed constant of $P_{ji}$
              for all $j < i$ */
$S-2.3$ :     For all processors $P_{ji}$, where $j < i$ do
              the following:
$S-2.3.1$         - Modify $R_s$ by $R_s * R_{R_r}$
                  - Get $R_s$ of its right neighbour
                  using Fetch_right_neighbour
                  - Modify $R_s$ by $R_s + R_r$
```

4.3 Time Complexity

The dominating operations in the main algorithm are: Broadcast in a row, Parallel exchange, Fetch_right_neighbour and their complexities are $O(2^L - 1)$,$O(1)$, $O(2^L - 1)$ respectively. So the complexity of main algorithm is $O(n * 2^L)$. Now if L is kept constant then the time complexity of the algorithm becomes $O(n)$. Hence the cost of the algorithm is $O(n^3)$ which is comparable to the optimal sequential algorithm [1].

5 AN EXAMPLE

To illustrate the execution of the algorithm let us consider a a system of 3 linear equations and a RCC-FULL $\lambda(4,1)$ with 16 processors. Initially, coefficients are in R_s registers. Steps used to reach the next stage are mentioned on the $\rightarrow$. Data stored in R_r register at any stage are encircled and shown only whenever they are needed.

6 CONCLUSION

In this paper, a cost-optimal parallel algorithm to find the solution of n simultaneous linear equations has been presented. The algorithm uses RCC-FULL $\lambda(N,L)$ network and is designed following the strategy of the sequential Gaussian Elimination algorithm. It has the time complexity of $O(n)$ with $O(n^2)$ processors and has been tested under the simulated environment using PVM 3.0.

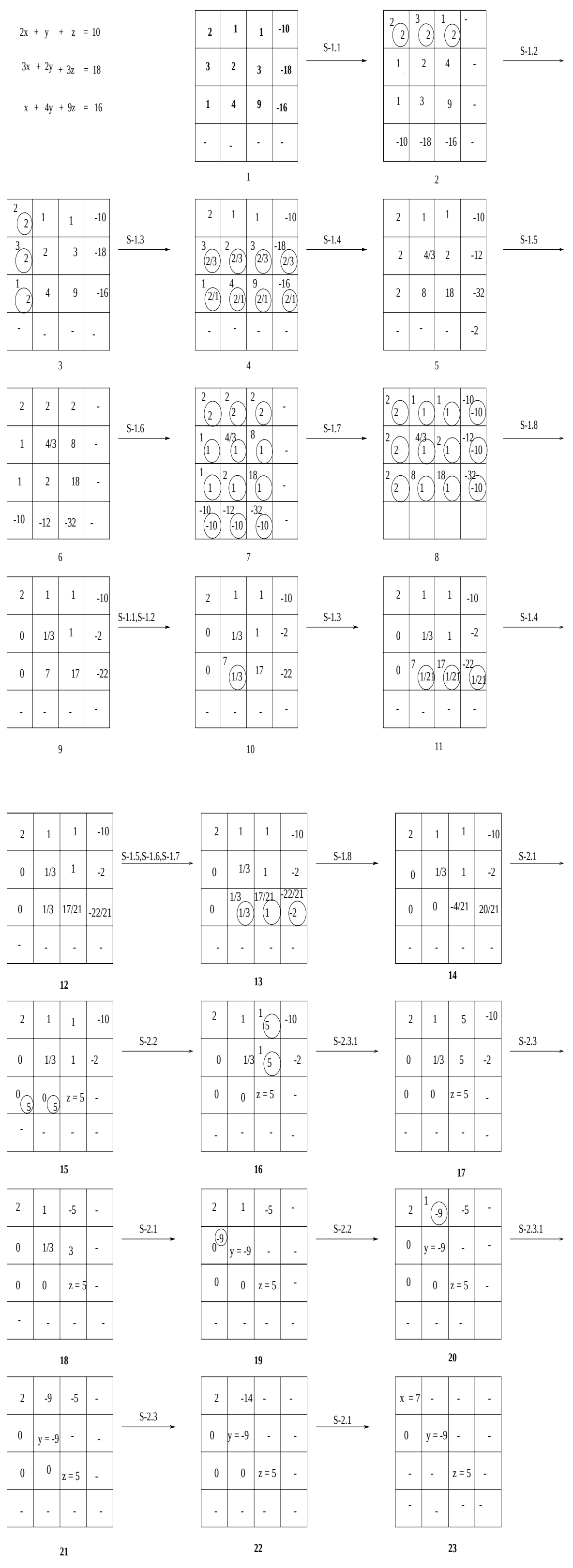

Figure 4. Different stages of solutions

References

[1] Akl, S. G., *The Design and Analysis of Parallel Algorithms,* Prentice Hall, Englewood Cliffs, NJ, 1989.

[2] Duff L. S., et.al., *Direct Methods for Sparse Matrices*, Oxford Science Publications, 1986.

[3] Hamid,M. and Hall,R.W., *RCC-FULL: An Effective Network for Parallel Computations,* Journal of Parallel and Distributed Computing, 1997,139-155.

[4] Hilderbrand, F. B., *Introduction to Numerical Analysis,* McGraw Hill, NY, 1956.

[5] Leighton, T., *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercube,* Morgan Kaufman, San Mateo, C.A., 1993.