A Parallel Matrix Inversion Algorithm on Torus with Adaptive Pivoting

Javed I Khan, Woei Lin, & David Y. Y. Yun
East West Center & Department of Electrical Engineering
University of Hawaii at Manoa
2540 Dole Street, Honolulu, HI-96822
javed@wiliki.eng.hawaii.edu
Tel:808-956-7249
Fax:808-941-1399

ABSTRACT

This paper presents a parallel algorithm for matrix inversion on a **torus** interconnected MIMD-MC^2 multi-processor. This method is faster than the parallel implementations of other widely used methods namely Gauss-Jordan, Gauss-Seidal or LU decomposition based inversion. This new algorithm also introduces a novel technique, called **adaptive pivoting**, for solving the zero pivot problem at no cost. Our method eliminates the costly **row interchange** used by the existing elimination based parallel algorithms. This paper presents the design, analysis and simulation results (on a 32 Node Meiko Transputer) of this new and efficient matrix inversion algorithm.

August 14, 2001

Published in the Proceedings of the
21st International Conference on Parallel Processing, ICPP'92
St. Charles, Illinois, August 1992, pp69-72

A PARALLEL MATRIX INVERSION ALGORITHM ON TORUS WITH ADAPTIVE PIVOTING

Javed I. Khan, Woei Lin & David Y. Y. Yun
East West Center & Department of Electrical Engineering
University of Hawaii at Manoa, Honolulu, HI-96848
javed@wiliki.eng.hawaii.edu

This paper presents a parallel algorithm for matrix inversion on a ***torus*** *interconnected MIMD-MC^2 multi-processor. This method is faster than the parallel implementations of other widely used methods namely Gauss-Jordan, Gauss-Seidal or LU decomposition based inversion. This new algorithm also introduces a novel technique, called* ***adaptive pivoting****, for solving the zero pivot problem at no cost. Our method eliminates the costly* ***row interchange*** *used by the existing elimination based parallel algorithms. This paper presents the design, analysis and simulation results (on a 32 Node Meiko Transputer) of this new and efficient matrix inversion algorithm.*

1 Introduction

Csanky [1] has shown the best possible bound of parallel matrix inversion. Given a finite number of processors, the best we can do is to compute inverse in $O(\log^2 n)$ time using $O(n^4/\log n)$ processors. However, such processor requirement is unrealistic. Among the O(n) algorithms, using $O(n^2)$ processors, QR method using Givens transformations [4] takes more than 8n computational steps (because of decomposition), LU decomposition requires 3n steps (because of forward and backward substitutions), Gauss-Jordan requires 2n steps (because of backward substitution) and Gauss-Seidal requires n steps, but its step-width is 4flops (because it maintains two matrices) [4]. We here propose a new parallel algorithm, which is superior to all of the above methods, and is able to compute the inverse by performing n^3 computations in n sequential steps on n^2 processors with 2flops step-width. This algorithm is based on Faddeeva's [2] method for the computing determinant. We have reduced the matrix inversion problem to the problem of computing determinants, and mapped the transformed computations on a torus architecture with overlapped computational phases. As we will see, the perfect match between the algorithm and the homogeneous torus structure resulted in a highly localized and uniform communication and computation pattern.

Two of the most critical problems in matrix manipulation are zero pivots and propagation of rounding off error [5]. Some researchers suspect the existence of a law of conservation in linear systems which states that if a stability criterion is to be met then a certain number of arithmetic steps must be performed [4]. Existing matrix algorithms use row column interchange for stability. But, in parallel computation such interchange is prohibitively expensive due to communication cost. Our algorithm solves the problem of zero pivot by **dynamic adaptation of pivots** and re-labelling of elements using local informations. Thus, it substitutes the costly row column interchange at no cost.

Sections 2 and 3 respectively present the theoretical derivation of the computational procedure and the supporting theorems for adaptive pivoting. Sections 5 and 6 respectively present the algorithm and the simulation study performed on a 32 node Meiko Transputer.

2 Inversion Method

Computing Determinants: We shall start from the Faddeev's original procedure [2]. Given the nxn matrix A, if $a_{11} \neq 0$ then the determinant is,

$$|\mathbf{A}| = \begin{vmatrix} a_{11}a_{12}a_{13}\dots a_{1n} \\ a_{21}a_{22}a_{23}\dots a_{2n} \\ \dots\dots \\ \dots\dots \\ a_{n1}a_{n2}a_{n3}\dots a_{nn} \end{vmatrix} = a_{11} * \begin{vmatrix} a_{22.1}a_{23.1}\dots a_{2n.1} \\ a_{32.1}a_{33.1}\dots a_{3n.1} \\ \dots \\ a_{n2.1}a_{n3.1}\dots a_{nn.1} \end{vmatrix} = a_{11} * A_{.1}$$

Here, the subscript .n refers to the successive stages after each Gaussian elimination. i.e., $a_{ij.1}=a_{ij.0}-a_{i1.0}\ a_{1i.0}/a_{11.0}$. i,j= 2,3,...n. One such elimination along the rows (or columns) of matrix **A** reduces the original determinant to a product of pivot a_{11} and a determinant of (n-1)th order matrix $\mathbf{A}_{.1}$. Repeated application of the above transformation generates matrices of decreasing orders and corresponding coefficients a_{11}, $a_{22.1}$ ...$a_{nn.n-1}$. Finally, the determinant of the matrix is given by the scalar equation (1) provided all the pivots are non-zero.

$$|\mathbf{A}| = a_{11} * a_{22.1} * a_{33.2} * \dots a_{nn.n-1} \tag{1}$$

Computing Inverse: The cofactor A_{ij} of the element a_{ij} is the determinant of a new (n-1)x(n-1) matrix formed by the elements of A except the ith row and jth column. However, this same cofactor A_{ij} can be defined as the determinant of another matrix E of order (n+1)x(n+1), which is constructed by augmenting (instead of eliminating) a new row and a new column with A in the following way,

$$E = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} & e_{1(n+1)} \\ a_{21} & a_{22} & \dots & a_{2n} & e_{2(n+1)} \\ . & . & & . & \\ . & . & & . & \\ a_{n1} & a_{n2} & \dots & a_{nn} & e_{3(n+1)} \\ e_{(n+1)1} & e_{(n+1)2} & \dots & e_{(n+1)n} & e_{(n+1)(n+1)} \end{pmatrix} \quad \begin{aligned} & where, e_{pq} \\ & = a_{pq} \quad when\, 1 \le p \le n, \quad 1 \le q \le n \\ & = 1 \quad when \quad p = (n+1), \quad q = i \\ & = 1 \quad when \quad p = j, \quad q = (n+1) \\ & = 0 \quad else \end{aligned}$$

Now, n times repeated application of the Faddeeva's procedure (just described above in Sec-2.1) on **E** generates a new matrix $\mathbf{A}_{.n} = [a_{ij.n}]$ such that:

$$-A_{ij} = a_{11} * a_{22.1} * a_{33.2} \dots * a_{nn.n-1} * a_{ij.n} = |\mathbf{A}|\, a_{ij.n} \tag{2}$$

Equation-(2) and Cramer's rule directly shows that $-\mathbf{A}_n^{\mathbf{T}}$ is the inverse of **A** as shown below:

$$\mathbf{A}^{-1} = \frac{1}{|A|} * \begin{pmatrix} A_{11} & A_{12} & A_{13} & \dots & A_{1n} \\ A_{21} & A_{22} & A_{23} & \dots & A_{2n} \\ . & . & . & & . \\ . & . & . & & . \\ . & . & . & & . \\ A_{n1} & A_{n2} & A_{n3} & \dots & A_{nn} \end{pmatrix} = - \begin{pmatrix} a_{11.n} & a_{12.n} & a_{13.n} & \dots & a_{1n.n} \\ a_{21.n} & a_{22.n} & a_{23.n} & \dots & a_{2n.n} \\ . & . & . & & . \\ . & . & . & & . \\ . & . & . & & . \\ a_{n1.n} & a_{n2.n} & a_{n3.n} & \dots & a_{nn.n} \end{pmatrix}^{\mathbf{T}}$$

Computational Model & Mapping The following procedure computes Equation-(2):

1. Enter the first phase k=1. Take the original matrix $[a_{ij}]$ and create the extended matrix E^1 with $e_{ij} = a_{ij}$ for i,j=1,2..n.
2. Add a new row and a new column with all zero elements except, $e_{(n+1),1.k} = -1$ and $e_{1,(n+1).k} = 1$.
3. Calculate $e_{i,j.1} = e_{i,j.1} - e_{i,1.1} * e_{1,j.1} / e_{1,1.1}$ for all i,j= 2,..(n+1).
4. Consider the elements $e_{ij.k}$ of E^k with i and j varying from 2 to (n+1) as the elements $e_{i,j.k+1}$ of the new extended matrix E^{k+1}.
5. Repeat steps 2,3 & 4 until k=n.

At the end of nth phase, the inverted matrix is given by $[e_{ii.n}]^T$. Fig-1 presents the propagation of computation plane by the above model for a 4x4 matrix. In each of the phases, the elimination is performed on the shaded elements. The lower right 4x4 elements at the end of 4th phase contain the transpose of the inverse.

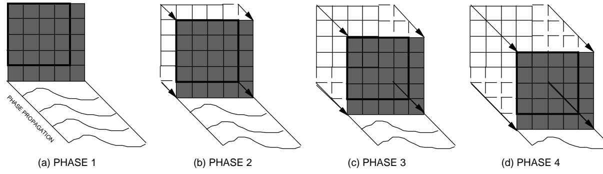

Fig-1

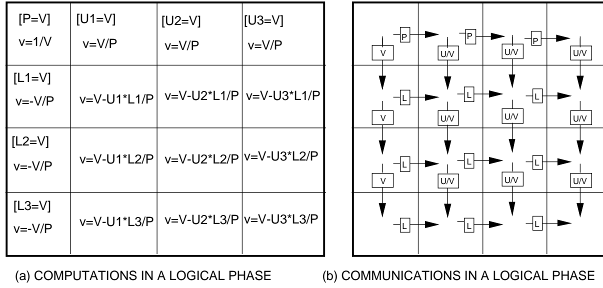

Fig-2

Fig-2(a) shows the required computations in each of the phases. Capital letters denote the previous value and small letters denote the new value of the local variables. To perform these computations, each of the top row and left column nodes requires only one external data (the pivot), while all others need three external data i) the pivot, ii) the left most element of the row and iii) the topmost element of the column. We have reduced the number of communication from three to two by making the pivot transfer implicit. The topmost nodes, upon receiving P, transmit U/P instead of U and the nodes compute v=V-L*[U/V]. Fig-2(b) shows the resulting communication pattern.

Now, we will map the phases. It is evident from the computational model (Fig-1) that the phase computation directly maps onto a mesh. However, between the phases, the corner of the computation frame slides down diagonally. A straightforward mapping on the mesh architecture therefore requires that all of the elements be pulled up diagonally one step in between phases. Algorithmically, this mapping is sound, but pulling the whole matrix up is expensive, and fortunately, can be avoided.

At the end of each phase, the data of some nodes are no longer being used (the first row and the first column) while, on the other hand, some new nodes are required (in the bottom). We have decided to map these new elements directly on the emptied nodes. The consequences are, i). the diagonal transfer of the entire matrix is no longer required, ii). the

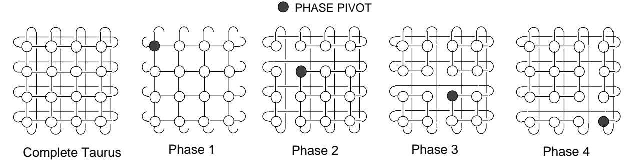

Fig-3

logical mesh containing the active matrix is wrapped and iii). the phase procedure is required to slide down the diagonal at every iteration.

We have used torus to accommodate all the phases as shown in Fig-3. Because of its boundary free property, it maps uniformly all the wrapped around meshes. The phase computation and communication pattern presented in Fig-2, applied recursively on the logical meshes of Fig-3 provides a compact mechanism for matrix inversion.

3 . Dynamic Adaptive Pivoting

In this section we show that it is possible to compute the inverse using more flexible pivot ordering than the *principle-diagonal* pivot order used earlier in section 2 (or in conventional algorithms). We know, a complete computation requires n pivots. In this section, using the following two theorems, we will show that any element can be a part of those n pivots so far no two are from the same row or same column. Thus, there can be $n^2 * (n^2 - 2n + 1) * (n^2 - 4n + 4) * (n^2 - 6n + 7) * \ldots * 2 * 1$ ways of selecting the pivots. Below, **a** $\Leftarrow$ **b** refers to **a 'maps' b**.

***Theorem 1**: If **T** is a torus where individual processors are* t_{ij} ***and B** is A^{-T} with elements a_{ij} and P is a set of n pivots, then the sequence of selecting individual pivots from the same set in successive phases does not affect the final mapping **of the elements of B on the processors of the torus T** where P is as follows and x_i and y_i are the pivot position selected in i^{th} iteration:*

$$P = \left\{ P_{x_1 y_1}, P_{x_2 y_2}, \ldots P_{x_n y_n} \ni x_1 \neq .. \neq x_n, 1 < x_j < n, y_1 \neq .. \neq y_n, 1 < y_j < n \right\}$$

Proof: To prove the above theorem we will show that the interchange of the lth and $(l+1)^{th}$ pivots, i.e. the sequences (i). $\ldots P_{x_l y_l} \rightarrow P_{x_{l+1} y_{l+1}} \rightarrow ..$ and (ii). $\ldots P_{x_{l+1} y_{l+1}} \rightarrow P_{x_l y_l} \rightarrow \ldots$, both generates the same result at the end of $(l+1)^{th}$ phase. Let, at the end of $(l-1)^{th}$ phase, the matrix be as follows where P and Q are the next two successive pivots and T_{ij} is an arbitrary element of the torus.

$$A_{l-1} = \begin{pmatrix} \cdot & \cdot & \cdot & & \cdot & & & & & \\ \cdot & \cdot & P_{x_l y_l} & \cdot & \cdot & U_{x_l j} & \cdot & \cdot & N_{x_l y_{l+1}} & \cdot & \cdot \\ \cdot & \cdot & \cdot & & \cdot & & & & & \\ \cdot & \cdot & L_{i y_l} & \cdot & \cdot & T_{ij} & \cdot & \cdot & R_{i y_{l+1}} & \cdot & \cdot \\ \cdot & \cdot & \cdot & & \cdot & & & & & \\ \cdot & \cdot & M_{x_{l+1} y_l} & \cdot & \cdot & V_{x_{l+1} j} & \cdot & \cdot & Q_{x_{l+1} y_{l+1}} & \cdot & \cdot \\ \cdot & \cdot & \cdot & & \cdot & & & & & \end{pmatrix}$$

The sequence (i) $\ldots P_{x_l y_l} \rightarrow P_{x_{l+1} y_{l+1}} \rightarrow ..$computes the following at the end of $(l+1)^{th}$ phase on Torus node $T_{ij.(l+1)}$:

$$T_{ij.l+1} \Leftarrow \left(T - \frac{UL}{P}\right) - \left(R - \frac{NL}{P}\right) \frac{\left(V - \frac{UM}{P}\right)}{\left(Q - \frac{MN}{P}\right)} = T - \frac{VRP + LUQ - LVN - RUM}{PQ - MN}$$

The other sequence (ii) $\ldots P_{x_{l+1} y_{l+1}} \rightarrow P_{x_l y_l} \rightarrow \ldots$at the end of $(l+1)$th phase computes on the same node $T_{ij.(l+1)}$:

$$T_{ij.l+1} \Leftarrow \left(T - \frac{VR}{Q}\right) - \left(L - \frac{RM}{Q}\right) \frac{\left(U - \frac{VN}{Q}\right)}{\left(P - \frac{MN}{Q}\right)} = T - \frac{LUQ + VRP - LVN - RUM}{PQ - MN}$$

These two values are same. Thus, if pivot set P is fixed, the sequence does not effect the final mapping.

***Theorem 2**: If **B** is A^{-T}, and initially the matrix **A** is mapped onto torus **T** in such a way that a_{ij} is mapped on T_{ij} for i,j=1,2,3....n and P_{lm} and P_{xy} are two members of the pivot set P, then at the end of the nth phase, T_{mx} contains b_{ly} and T_{yl} contains b_{xm}.*

Proof: Let us consider the general pivot order,

$$P = \{P_{x_1y_1}, P_{x_2y_2}, \ldots P_{x_ny_n} \ni x_1 \neq .. \neq x_n, 1 < x_j < n, y_1 \neq .. \neq y_n, 1 < y_j < n\}$$

Since, the internal sequencing of the pivots does not effect the final mapping (according to Theorem 1), we will sort the pivots along row dimension. Let the resulting equivalent sequence be:

$$\dot{P} = \{P_{1z_1}, P_{2z_2}, \ldots P_{nz_n} \ni z_1 \neq z_2 \neq .. \neq z_n, 1 < z_j < n\} \equiv P$$

Since, at each of the successive phases a new column and a new row of the final matrix $\mathbf{A_{.n}}$ are introduced. Thus, the kth column of the final matrix will contain the values of the cofactor of the rows calculated at the kth phase. Therefore, the ith column in $A_{.n}$ will contain the cofactor of the elements of the z_i^{th} row.

Now, to find out the content of a row, we again sort the pivots along the column dimension. Let the resulting equivalent sequence be the successive elements of the pivot set $\ddot{\mathbf{P}}$

$$\ddot{P} = \{P_{w_11}, P_{w_22}, \ldots P_{w_nn} \ni w_1 \neq w_2 \neq .. \neq w_n, 1 < w_j < n\} \equiv P$$

The above sequence maps the cofactor of the w_j^{th} column in the jth row. Thus, if p_{lm} and p_{xy} are two pivots, then,

i. l^{th} column of T $\Leftarrow$ m^{th} row of $A^{-1} \equiv m^{th}$ column of B.
ii. x^{th} column of T $\Leftarrow$ y^{th} row of $A^{-1} \equiv y^{th}$ column of B.
iii. m^{th} row of T $\Leftarrow$ l^{th} row of $A^{-1} \equiv l^{th}$ column of B.
iv. y^{th} row of T $\Leftarrow$ x^{th} row of $A^{-1} \equiv x^{th}$ column of B.

The following required mappings are the direct consequence of the above:

$$T_{ml} \Leftarrow b_{lm} \equiv A_{ml}; \quad T_{mx} \Leftarrow b_{ly} \equiv A_{yl}; \quad T_{yl} \Leftarrow b_{xm} \equiv A_{mx}; \quad T_{xy} \Leftarrow b_{xy} \equiv A_{yx}$$

4 . The Algorithm

The algorithm proceeds through successive overlapped wavefronts. At each of the execution phases, the algorithm selects a pivot and then the computational wave, originating at the pivot, follows the computation and communication pattern explained in Fig-2. Pivots are selected according to a flexible *default order*. If a zero pivot is encountered, the *default order* is interrupted and a new pivot position is tried according to a *search order*.

__Default Order__: According to Theorem 1 & 2 any n pivots can be selected in any sequence to compute the inverse elements as long as no two of them are from the same row or same column. The performance of the wave algorithm depends on the *inter phase gap*. Therefore, the pivot sequence for the *default order* should be chosen such that, this gap is minimum. Therefore, any of the *diagonal* orders is a good choice as a *default pivot* order.

__Search Order__: The search for a non zero pivot can proceed along the i) same row, ii) same column or iii) same diagonal. Any three of the strategies can be adopted. If it proceeds through the same row (or column) than the singularity can be detected efficiently. On the other hand, if it proceeds through the same diagonal, according to Theorem 1, the mapping expected from the forward diagonal pivot order will be preserved.

__Tracing__: Flexible pivoting destroys the expected element-processor mapping. Since, all the processors know both the pivot selection strategies algorithmically, therefore, all can determine the physical position of the current pivot locally. Thus, according to theorem 2, a processor in the same row (or same column) as with a pivot, records the phase number as the final column (or row) position. This scheme provides the final mapping. Interestingly this mapping is done using only local information.

Below we present a simplified pseudo-code of the algorithm. For clarity we have assumed the actual pivot order is supplied by the **xphase[]** and **yphase[]** arrays.

```
int k=0, s=n, p_count=n;
int xphase[],yphase[];
getpid(i,j);

/* Loop for n phases*/
while(p_count) {
    px= xphase[k];
    py= yphase[k++];

/* If Phase pivot*/
   if(px=i and py=j) {
     if(v <= thres) {
        v= ABORT;
        xphase[s]=px;
        yphase[s++]=yx;
        p_count++;}
     send(v:south);
     send(v:east);
     v=1/v;}

/* If Upper Row Elements*/
   elseif(py==j) {
     recvb(p:north);
     if(p==ABORT) {
        xphase[s]=px;
        yphase[s++]=yx;
        v=ABORT;
        p_count++;}
     else v=v/p;
     send(v:east);
     send(p:south); }

/* If Left Column Elements*/
   elseif(px==i) {
     send(v:south);
     recvb(p:west);
     if(p==ABORT) {
        xphase[s]=px;
        yphase[s++]=yx;
        p_count++; }
     send(p:east);
     v=-v/p;}

/* For Other Elements*/
   else {
     recvb(u:north);
     if(u==ABORT) {
        xphase[s]=px;
        yphase[s++]=yx;
        p_count++;}
     send(u:south);
     recvb(l:west);
     send(l:east);
     v=v-u*l;}
p_count--; }
```

__Complexity of The Algorithm__: The performance of the algorithm lies on the time between the initiations of two successive phases. If the pivots are along the principle diagonal, then the successive phases can be initiated at 1,7,13,19.. time instances. The distance 6 consists of 4 communication and 3 computations(1 division+1 multiplication+ 1 subtraction). There are n-1 phase gaps. The last phase propagates to the logical end in time 2(n-1) communication steps and n computations. Thus,

$$T_{par}=(4t_{comm}+t_{sub}+t_{mul}+t_{div})(n-1)+2(n-1)t_{comm} + n.t_{comp}$$

If we consider the **sequential execution time**, then in each phase, there are 2*(n-1) divisions, $(n-1)^2$ multiplications, 1 inversion and $(n-1)^2$ subtractions. Thus, for n phases:

$$T_{seq}=(2n-1).n.t_{div}+n(n-1)^2(t_{mul}+t_{sub})$$

The **scalability** of our algorithm is given by the following equation where **m** is the order of matrix and **n** is the order of torus space.

$T_{par}=(4t_{comm}+(m/n)(t_{sub}+t_{mul}+t_{div}))(m-1)+2(n-1)t_{comm} + m.t_{comp}$

This expression shows that smaller block size does not significantly increase the communication cost (which is still proportional to the matrix size) but reduces the computation cost drastically.

5 . Performance Analysis

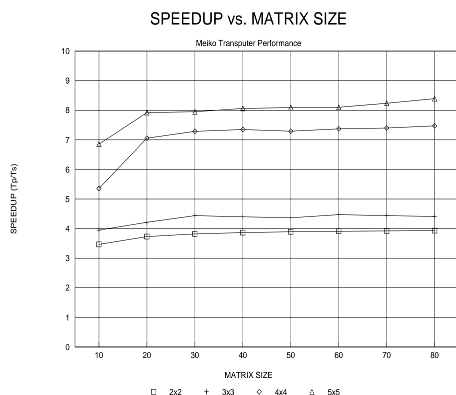

Fig-4

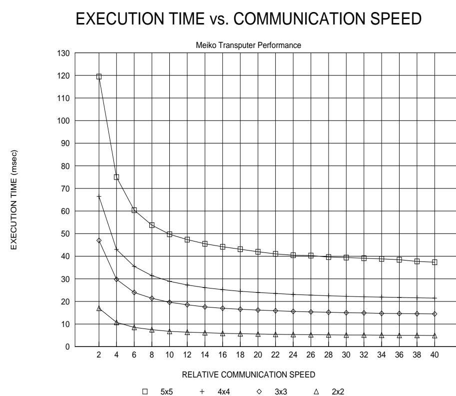

Fig-5

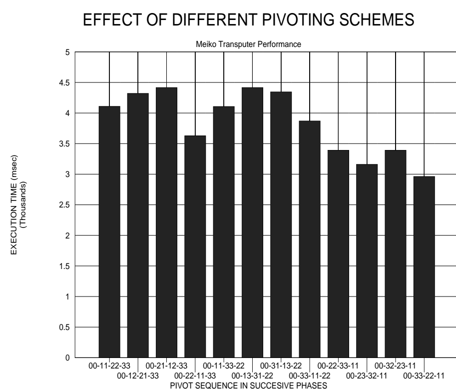

Fig-6

We have implemented the algorithm with adaptive flexible (any general) order pivoting and tested its performance on a 32 node Meiko Transputer. We used randomly generated matrices of various sizes up to 80x80. Block decomposition strategy has been adopted to map large matrix on a smaller torus. Fig-4 shows the relative speedup with the variation of the matrix size for different torus configurations. Fig-5 shows the variation of absolute execution time with respect to the variation of relative communication speed with various torus size. We varied the torus sizes from 2x2 to 5x5. To keep the computations per node constant, we used a block size of 5x5 on each torus node. As expected, the performance improved with large matrices, higher communication speed and with more processors.

To study the effect of zero pivots we picked up some non regular pivot sets. Fig-6 shows the effect of selecting such arbitrary pivots on execution cost. For a 4x4 torus, there are 576 ways of selecting the pivots. In this figure we have considered 12 from this set (so that the first pivot is 00 and the last pivot is any element on the principle diagonal). The subset shows 50% variation in the execution cost due to pivoting.

6 . Conclusion

In this paper we have presented a parallel algorithm for matrix inversion. The algorithm uses an nxn torus connected parallel processor to invert an n by n matrix in O(n) time. The resulting algorithm is one of the most compact and efficient method in terms of the number of computations and communication pattern.

Here we will put some comments and a comparison on the performance of our method in **solving linear systems**. To solve the linear system AX=B, our method requires additional n steps to multiply the inverse with the right hand side B. On the other hand, LU decomposition method (cholesky for example), requires 2n additional steps to complete the forward and backward substitutions for each column of B. The bottle neck of LU decomposition method is the inherently sequential forward and backward substitution. Thus, our method will be faster than LU decomposition method in solving linear systems in parallel, although, in sequential case the LU decomposition will perform better.

On the issues of **numerical stability**, we have shown that our method can take care of zero pivots by adaptive pivoting. However, to improve stability against catastrophic cancellation and finite precision arithmetic, our method suffers similar disadvantages [3,5] like the LU decomposition or any other elimination methods. The sorting required for partial or complete pivoting requires O(n.logn) parallel comparison steps and a global search in each iteration. However, our adaptive pivoting method will save the traditional interchange required after sorting.

7 . References

[1] L. Csanky, *Fast Parallel Matrix Inversion Algorithms*, SIAM J. Vol 5, 1976, p618-623.

[2] V. N. Faddeeva, *Computational Methods of Linear Algebra*, Chapter 2, Translated by C. D. Benster, Dover Pub., New York, 1959.

[3] G. H. Golub & C. F. V. Loan, *Matrix Computations*, The Johns Hopkins University Press, Maryland, 1985.

[4] D. Heller, *A Survey of Parallel Algorithms In Numerical Linear Algebra*, SIAM Review, Vol 20, no 4, October 1978, p740-777.

[5] J. H. Wilkinson, *Error Analysis of Direct Methods of Matrix Inversion*, J. Assoc. Comp. Mach. 8, 1961, p281-330.