

Minimizing Communication Overhead for Matrix Inversion Algorithms on Hypercubes *

Xiaodong Wang

Vwani P. Roychowdhury

Abstract

We propose novel parallel Gauss-Jordan inversion algorithms (with or without partial pivoting) under different data partitioning strategies. The machine model we assume is a MIMD hypercube, using asynchronous message passing, with the software supporting user specified interrupt handling. These algorithms achieve almost optimal overlapping of communication delays by computation, leading to a minimization of communication overhead. Furthermore, it is shown that the optimal data layout for our algorithms is different from that for hypercubes that do not overlap communication and computation. Rigorous analytical and numerical performance analysis of our parallel algorithms are presented as well.

1 Introduction

Parallel Gauss-Jordan matrix inversion algorithms on the hypercube multiprocessors have been extensively studied in the literature. Two common data partitioning strategies for matrix algorithms are row-wise partitioning and submatrix partitioning. It has been claimed that for the parallel GJ inversion algorithm, submatrix partitioning scheme exhibits communication overhead advantages not shared by partitions limited to rows or columns [1,2]. Most parallel algorithms proposed in the literature, however, do not attempt to mask inter-processor communication by computation. As a result, during most of the communication time, the processors are actually idle, waiting for the data to arrive.

By utilizing the interrupt handling capability of the parallel machine, we believe the communication overheads of many parallel matrix algorithms can be re-

*The authors are with the School of Electrical Engineering, Purdue University, West Lafayette, IN 47907.
xiaodong@ecn.purdue.edu, vwani@ecn.purdue.edu.
This work was supported in part by the NSF Grant No. ECS-9308814.

Partition	Total Comm. Overhead
row	$N(t_s + t_w N) \log p$
submatrix	$N(t_s + t_w \frac{N}{\sqrt{p}}) \log p$

Table 1: Communication Overhead, No Overlapping

duced by masking interprocessor communication by computation. In this paper, we propose and analyze new parallel GJ inversion algorithms under different data partitioning strategies, with or without partial pivoting.

The communication overhead of a processor consists of the setup time, when the processor sets up the communication channel, and the idle time, when the processor is idle, waiting for the message to arrive. The strategy for overlapping communication and computation is to let each processor compute and send out the data needed by the other processors as early as possible, so that these data would have arrived at the destinations when they are to be used. The best we can do to this end is to make the idle time of each processor zero, and to make the total setup overhead of each processor as small as possible.

2 Summary of Results

Suppose an $N \times N$ matrix is to be inverted on a p -processor hypercube. Let t_s be the channel setup time, and t_w be the data transmission rate. Table 1 lists the communication complexities of the parallel GJ inversion algorithm when no communication and computation overlapping is attempted [1]. Table 2 summarizes the results in this paper on the total communication overheads when full overlapping is achieved.

Figure 1 depicts some numerical results on the total communication overheads of different algorithms. The vertical axis is log scaled. It is clear from this figure that overlapping data communication and computa-

Partition	Total Comm. Overhead
row	$\frac{1}{2}Nt_s$
submatrix, w/o pivoting	Nt_s
submatrix, w/ pivoting	$\frac{1}{2}N(1 + \log p)t_s$

Table 2: Communication Overhead, Full Overlapping

tion can greatly reduce the communication overhead. Furthermore, when the matrix size is large enough, *row partitioning is superior to submatrix partitioning when communication and computation are overlapped.*

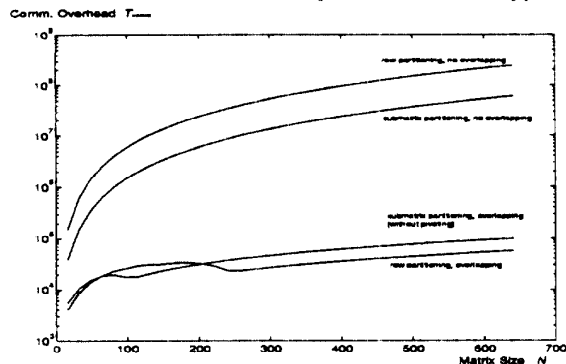


Figure 1: Comparison of the communication overheads of four parallel GJ inversion algorithms (Number of processors $p=16$)

3 Algorithm I: Row Partitioning, With Pivoting

We give our algorithm of parallel GJ inversion with column interchanges, using row partitioning. Suppose we want to invert a $N \times N$ matrix A on a d dimensional hypercube containing $p = 2^d$ processors. Assume $N = np$. We partition the matrix by rows and use wrap-mapping to distribute the rows of A among processors, i.e., rows $k, p+k, 2p+k, \dots, N-p+k$ are assigned to processor $P_k, 1 \leq k \leq p$. We use notation $[k]$ to represent $((k-1) \bmod p) + 1$, thus $1 \leq [k] \leq p$.

When the matrix is partitioned by rows, each row is entirely within one processor. Thus the search for the pivot element can be done by the single processor which has the pivot row. In the algorithm, during the k -th iteration, each processor first gets the k th pivot row of A and the pivot element from processor $P_{[k]}$, and then updates all the rows assigned to it. In order to mask the communication by computation, the

pivot row should be sent out at the earliest possible time, so that when a processor is to use it, it would have already arrived at that processor. Since processor $P_{[k+1]}$ is assigned row $(k+1)$ of A , it first updates this row during the k -th iteration. Then it normalizes this row and finds the pivot element (*compute-ahead*). After that it broadcasts this $k+1$ st pivot row and the pivot element to all the other processors, before it resumes to update the other $n-1$ rows using the k -th pivot row (*send-ahead*). By using this *compute-and-send-ahead* strategy, the $k+1$ st pivot row is transmitted through the communication channels while all the processors are still updating in the k th iteration. Thus the overlapping of communication and computation is achieved.

The following is the pseudo code of Algorithm I:

```

if ( $P = P_{[k+1]}$ ) then
  read in the  $k$ th pivot row and pivot element;
  update the  $k+1$ st row of  $A$ ;
  find the  $k+1$ st pivot element;
  normalize the  $k+1$ st pivot row;
  broadcast the  $k+1$ st pivot row;
  update the rest  $n-1$  rows of  $A$ ;
if ( $P = P_{[k]}$ ) then
  update the  $n-1$  rows of  $A$  (other than the  $k$ th row);
if ( $P \neq P_{[k+1]}$  and  $P \neq P_{[k]}$ ) then
  read in the  $k$ th pivot row and pivot element;
  update the  $n$  rows of  $A$ ;
Interrupt handling:
  forward the incoming message to appropriate processors, if needed;

```

4 The SBT, Broadcasting Algorithm

As discussed in the previous section, the $k+1$ st row of matrix A is broadcast by processor $P_{[k+1]}$ immediately after it is updated and normalized in the k th stage of the parallel GJ inversion algorithm, before $P_{[k+1]}$ starts updating the other rows using the k th pivot row. To make the $k+1$ st row being sent out at the earliest possible time, processor $P_{[k+1]}$ should receive the k th pivot row from processor $P_{[k]}$ at the earliest possible time. This implies that $P_{[k+1]}$ and $P_{[k]}$ should be adjacent on the hypercube. On the other hand, after receiving the k th pivot row, $P_{[k+1]}$ should not involve forwarding the message to some other processors so that its computation is not delayed by the startup time overhead and the $k+1$ st row can be sent out at the earliest possible time. This implies that $P_{[k+1]}$ should be an end node in the k th broadcasting tree.

Since the logically consecutively numbered processors need to be physically adjacent, i.e. processors $P_{[k+1]}$ and $P_{[k]}$ be adjacent on the hypercube, we address the processors in a binary-reflected Gray code [3] order with starting address as 0. Let the d -bit code of $p = 2^d$ integers be $Gray(d)$. The binary-reflected Gray code on d bits is recursively defined as following.

Let $Gray(d) = (g_1^d, g_2^d, \dots, g_{2^{d-1}}^d, g_{2^d}^d)$. Then

$$Gray(d+1) = (0g_1^d, 0g_2^d, \dots, 0g_{2^{d-1}}^d, 1g_{2^{d-1}}^d, \dots, 1g_1^d)$$

The physical address of processor P_i is g_i^d . For the rest of the paper, we will just use P_i to denote the physical address g_i^d .

In [3] the spanning binomial tree (SBT) of a d -cube rooted at node s was defined as follows. Let i be any node, and let $c = i \oplus s$, where \oplus denotes bitwise exclusive OR operation. Let q be such that $c_q = 1$ and $c_m = 0, \forall m \in M(c) = \{q+1, q+2, \dots, d-1\}$, and let $q = -1$ if $c = 0$. The set $M(c)$ is the set of leading zeros of c . Then in $SBT(s)$, the child nodes and the parent node of a given node i are

$$\text{children}(i) = \{(i_{d-1}i_{d-2}\dots\bar{i}_m\dots i_0), \forall m \in M(c),$$

$$\text{parent}(i) = \begin{cases} \phi, & i = s \\ (i_{d-1}i_{d-2}\dots\bar{i}_q\dots i_0), & i \neq s \end{cases}$$

At the k th broadcasting in Algorithm I, the root of the SBT is $P_{[k]}$. And because of the binary-reflected Gray code mapping, $P_{[k+1]}$ is adjacent to $P_{[k]}$ and therefore is at the first level of the SBT. However, with the SBT defined above, it is not guaranteed that $P_{[k+1]}$ is an end node, since by the definition, $P_{[k+1]}$ is an end node in $SBT(P_{[k]})$ only if they differ by the high order $(d-1)$ st bit.

Suppose $P_{[k]}$ and $P_{[k+1]}$ differ by the j th bit, we need to construct a SBT rooted at $P_{[k]}$ and with $P_{[k+1]}$ as an end node. We denote such a SBT as $SBT_j^k(s)$, meaning it is rooted at s and the j th neighbor node of s is an end node. Note the $SBT(s)$ defined in [3] is $SBT_{d-1}(s)$ here. We next show that $SBT_j^k(s)$ can be obtained by applying shuffle and inverse shuffle operation on $SBT_{d-1}(s)$.

Definition 1: The shuffle operation on a node $i = (i_{d-1}i_{d-2}\dots i_1i_0)$ is defined as $S(i) = (i_{d-2}i_{d-3}\dots i_1i_0i_{d-1})$. The inverse shuffle operation on i is defined as $S^{-1}(i) = (i_0i_{d-1}i_{d-2}\dots i_1)$. Moreover, $S^k(i)$ represents applying shuffle operation k times on i . and $S^{-k}(i)$ represents applying inverse shuffle operation k times on i . The shuffle operation on a graph $G(V, E)$ is defined as $S(G) = G(S(V), S(E))$, where $S(V) = \{S(i) | \forall i \in V\}$ and

$S(E) = \{(S(i), S(j)) | \forall (i, j) \in E\}$. Similarly we define the inverse shuffle operation on a graph $S^{-1}(G) = G(S^{-1}(V), S^{-1}(E))$.

Theorem 1: [5]

$$SBT_k(s) = S^{-(d-1-k)} [SBT_{d-1}(S^{d-1-k}(s))]$$

Definition 2: $SBT(Gray(d))$ is a family of $p = 2^d$ SBT's:

$$SBT(Gray(d)) = (SBT^{(1)}, SBT^{(2)}, \dots, SBT^{(2^d)}),$$

where $Gray(d) = (g_1^d, g_2^d, \dots, g_{2^{d-1}}^d, g_{2^d}^d)$, $SBT^{(i)} = SBT_j^k(g_i^d)$, where g_i^d and $g_{[i+1]}^d$ differ by the j th bit.

Theorem 2: [5] In $SBT(Gray(d))$, each node appears at the l th level of $\binom{d}{l}$ SBT's. Each node appears as an end node in 2^{d-1} SBT's.

Corollary 1: [5] In Algorithm I (row partitioning, with pivoting), when $SBT(Gray(d))$ is used to broadcast the pivot rows, the total setup overhead of each processor is $\frac{1}{2}Nt_s$.

5 Analysis of Algorithm I

Let $T_{start}^k(P)$ be the time processor P starts the k th iteration. $T_{complete}^k(P)$ is the time processor P completes the k th iteration. $T^k(P)$ is the computation load at the k th iteration of processor P , i.e. $T^k(P) = T_{comp}^k(P)$. T_{send}^k is the time that the k th pivot row is sent out. $T_{arrive}^k(P)$ is the time that the k th pivot row arrives at P . $s^k(P)$ is the setup overhead of P at the k th stage. $t^k(P)$ is the time between the message containing the k th pivot row is sent out by $P_{[k]}$ and the time it arrives at processor P , assuming there is no delay at each intermediate processor. $s^k(P)$ is either 0 or t_s , depending on whether or not P is an end node in the k th broadcasting tree. Let $H(P_{[k]}, P)$ be the Hamming distance between the physical addresses of $P_{[k]}$ and P . Then the message will take $H(P_{[k]}, P)$ hops to reach P from $P_{[k]}$. Therefore $t^k(P) = H(P_{[k]}, P)(t_s + t_w N)$.

Lemma 1: [5] We have the following recurrent relationships:

$$T_{start}^k(P) = \begin{cases} \max\{T_{complete}^{k-1}(P), T_{arrive}^k(P)\} + s^k(P), & P \neq P_{[k]} \\ T_{complete}^{k-1}(P) + s^k(P), & P = P_{[k]} \end{cases}$$

$$T_{complete}^k(P) = T_{start}^k(P) + T^k(P)$$

$$T_{send}^k = T_{start}^{k-1}(P_{[k]}) + T_2$$

Theorem 3: [5] In Algorithm I, when the SBT; broadcast algorithm is used, then

$$T_{arrive}^k(P) = T_{send}^k + t^k(P)$$

Remark: Theorem 3 states that in Algorithm I whenever a message arrives at an intermediate processor, the processor can forward this message immediately without delay. A delay is incurred if when the message arrives at the processor, the processor is setting up the communication channel for a previous message. Then only after the processor finishes setting up the communication channel for the previous message can it start forward this message. In the worst case the message can be delayed for a time up to t_s .

Definition 3: We say that the communication is fully overlapped by computation for processor P in Algorithm I if

$$T_{complete}^{k-1}(P) \geq T_{send}^k + t^k(P), \quad 2 \leq k \leq N$$

or

$$T_{start}^k(P) = T_{complete}^{k-1}(P) + s^k(P), \quad 2 \leq k \leq N$$

That is, the k th pivot row has already arrived at processor P when P completes computation of the $k-1$ st stage of the parallel GJ inversion algorithm.

Definition 4: Let $Q^k(P)$ be the message queue length of processor P at stage k , then

$$Q^k(P) = l$$

if

$$T_{send}^{k+l} + t^{k+l}(P) \leq T_{complete}^k(P) < T_{send}^{k+l+1} + t^{k+l+1}(P)$$

That is, after processor P completes stage k and is to start stage $k+1$, messages for stage $k+1, k+2, \dots, k+l$ have already arrived at P .

Theorem 4: [5] When $N \geq N_0$, full masking communication by computation can be achieved, where N_0 is the positive root of the quadratic equation

$$\left(\frac{N}{p} - 3\right)Nf = \frac{1}{2}pt_s + 2(t_s + t_w N) \log p.$$

Theorem 5: [5] When $N \geq N_0$, the message queue length is at most 2, i.e. $Q^k(P) \leq 2$.

Corollary 2: [5] When $N \geq N_0$, the communication overhead of processor P in Algorithm I is $d_0(P) + \frac{N}{2}t_s$.

Next we give some simulation results on the parallel overheads of Algorithm I, obtained by numerically evaluating the recurrent relationships given in

Lemma 1. Assume the time to update each element of A is f . The time is scaled to f , i.e. let $f = 1$. We use the machine parameter $t_s = 150$ and $t_w = 3$. These parameters are close to that of the NCUBE 2 machine. Figure 2 depicts the communication overhead T_{comm} vs the matrix size N when the machine size $p = 16$. Curve 1 is the total communication overheads; curve 2 is the total communication overheads minus the initial delays; for the case of curve 3, we let each processor initially holds the first row of A , such that all processors can start without the initial delay.

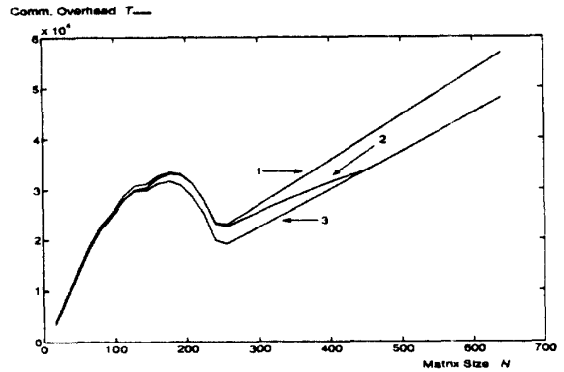


Figure 2: Communication overheads of Algorithm I (number of processors $p = 16$).

6 Algorithms Using Submatrix Partitioning

6.1 Algorithm II: Submatrix Partitioning, Without Pivoting

The processors are configured as a $\sqrt{p} \times \sqrt{p}$ array. The physical address of each processor is determined by the 2-d Gray code mapping. We still use the *compute-and-send-ahead* strategy to achieve communication and computation overlapping. In Algorithm II, there are concurrent broadcasts within subcubes at each iteration: first there are \sqrt{p} concurrent broadcasts of segments of multipliers within the subcube rows; then there are \sqrt{p} concurrent broadcasts of segments of the pivot row within the subcube columns.

Theorem 6: [5] In Algorithm II, the total setup overhead of each processor is Nt_s .

Theorem 7: [5] When $N \geq N_0$, the communication can be fully overlapped by computation, where N_0 is the positive root of the following quadratic equation:

$$\left(\frac{N^2}{p} - \frac{3N}{\sqrt{p}}\right) f = 2\sqrt{p}t_s + 2\left(2t_s + t_w \frac{N}{\sqrt{p}}\right) \log p$$

Figure 3 depicts the communication overheads T_{comm} vs the matrix size N when the machine size $p = 16$. As a comparison, we put the corresponding curve of Algorithm I (curve 1 in Figure 2) in the same figure. We can see that when the matrix size N is small, Algorithm II incurs less overhead than Algorithm I; but when N becomes large, Algorithm I incurs much less overhead than Algorithm II.

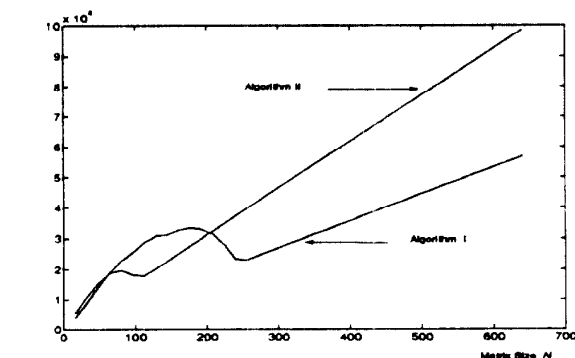


Figure 3: Communication overheads of Algorithm II (number of processors $p = 16$).

6.2 Algorithm III: Submatrix Partitioning, With Pivoting

We consider the parallel GJ inversion algorithm with column interchanges. Since the matrix is partitioned by submatrix, the searching of pivot element can not be done by a single processor. Instead, a row of processors each determines the local pivot element and then find the pivot element through recursive doubling. In Algorithm III, we let each processor updates its share of the submatrix column by column. As soon as the message containing the pivot row elements arrives, it will be interrupted and find the local pivot element. After that it will first update the column segment corresponding to this local pivot element and then participate permutation among the processors in the same subcube-row. After $\frac{d}{2}$ write-read pairs, each processor will get the pivot element and the corresponding segments of multipliers. Then it will normalize the pivot row elements using the pivot element.

Theorem 8: [5] In Algorithm III, the setup overhead for each processor is $\frac{1}{2}(1 + \log p)Nt_s$.

Thus Algorithm III incurs much larger setup overhead than Algorithm I and Algorithm II. Furthermore, since the recursive doubling essentially causes synchronization barrier for all the processors, although we can use the interrupt handling capability to let the proces-

sor do some computation while waiting for the message, it is difficult to achieve full overlapping.

7 Concluding Remarks

In this paper, we show that by utilizing the interrupt handling capability of the parallel machine, an asynchronous parallel GJ inversion algorithm can be designed such that the data transmission can be partially or fully masked by computation. We propose a new one-to-all broadcast algorithm on hypercube that helps to achieve communication and computation overlapping for the parallel GJ inversion algorithm. We give the new parallel GJ inversion algorithms under different data partitioning strategies (i.e. row partitioning and submatrix partitioning), with or without partial pivoting. For each algorithm, we prove a lower bound on the matrix size such that data transmission is fully overlapped by computation, and the total communication overhead when full overlapping is achieved. Our conclusion is that when the matrix size is large enough such that data transmission is fully overlapped by computation, the row partitioning strategy has the lowest communication overhead.

References

- [1] E. Chu, A. George and D. Quesnel, "Parallel matrix inversion on a subcube-grid", *Parallel Computing*, 19, 1993, pp.243-256.
- [2] M. Angelaccio and M. Colajanni, "Unifying and optimizing parallel linear algebra algorithms", *IEEE Transactions on Parallel And Distributed Systems*, Vol.4, No.12, Dec 1993, pp.1382-1397.
- [3] S.L. Johnsson and C-T Ho, "Optimum broadcasting and personalized communication in hypercubes", *IEEE Transactions on Computers*, Vol.38, No.9, Sept 1989, pp.1249-1268.
- [4] Anshul Gupta and Vipin Kumar, "Scalability of FFT on parallel computers", *IEEE Transactions on Parallel and Distributed Systems*, Vol.4, No.8, Aug 1993, pp.922-932.
- [5] Xiaodong Wang and V.P. Roychowdhury, "Minimizing communication overhead for matrix inversion/factorization algorithms on hypercubes", Technical Report, School of Electrical Engineering, Purdue University