Performance Bounds for Column-Block Partitioning of Parallel Gaussian Elimination and Gauss-Jordan Methods*

Apostolos Gerasoulis
Department of Computer Science
Rutgers University
New Brunswick, NJ 08903, USA
gerasoulis@cs.rutgers.edu

Tao Yang
Department of Computer Science
University of California
Santa Barbara, CA 93106, USA
tyang@cs.ucsb.edu

Abstract

Column-block partitioning is commonly used in the parallelization of Gaussian-Elimination(GE) and Gauss-Jordan(GJ) algorithms. It is therefore of interest to know performance bounds of such partitioning on scalable distributed-memory parallel architectures. In this paper, we use a graph-theoretic approach in deriving asymptotic performance lower bounds of column-block partitioning for both GE and GJ. The

*The work presented here was in part supported by ARPA contract DABT-63-93-C-0064, by the Office of Naval research under grant N000149310114, by a startup fund and faculty fellowship from University of California at Santa Barbara. The content of the information herein does not necessarily reflect the position of the Government and official endorsement should not be inferred.

new contribution is the incorporation of communication cost in the analysis which results in the derivation of sharper lower bounds. We use our scheduling system PYRROS to experimentally compare the actual run time performance with that derived by these lower bounds on the nCUBE-2 hypercube parallel machine.

1 Introduction

Gaussian-Elimination (GE) and Gauss-Jordan algorithms are widely used in the solution of linear algebraic systems and in the inversion of matrices. Parallel algorithms have been proposed for both GE and GJ [1, 3, 5, 8, 12, 14]. Given a linear algebraic system

$$Ax = b$$

where matrix A, where $A = (a_{i,j})$ is a non-singular $n \times n$ dense matrix and b is the right hand side. A very natural partitioning for GE and GJ, when solving the above system, is a block of columns, where operations are performed between blocks of columns of size r, e.g. Golub and Ortega [8], Robert et. al. [12]. Deriving lower performance bounds for such partitioning is of great interests since they can provide performance information for parallel programs, e.g. Cosnard [3], Ipsen et. al [11], Saad [14]. Most results in the literature for lower bound analysis ignore communication cost because it is quite difficult to incorporate such cost. In this paper, we use the graph scheduling theory and some of our recent results on the granularity of task graphs [7] to derive lower bounds that incorporate communication delays. We provide experimental evidence with our scheduling and code generation system PYRROS on the nCUBE-2 parallel machine to demonstrate the sharpness of the bounds. Cappello [1] does an asymptotic analysis of GE with block partitioning. Our work studies the optimal parallel time for any method with column-block partitioning, and our analysis is more exact.

2 Background

There are two fundamental steps in program parallelization:

1. Partitioning of program and data, and identifying the parallelism. The parallelism of a program can be modeled as a directed acyclic task graph (DAG).

2. Scheduling the task graph on a parallel machine.

We provide a summary of results and definitions that will be needed in the later analysis:

A directed acyclic weighted task graph (DAG) is defined by a tuple $G = (V, E, \mathcal{C}, \mathcal{T})$ where $V = \{n_j, j = 1 : v\}$ is the set of task nodes and $v = |V|$ is the number of nodes, E is the set of communication edges and $e = |E|$ is the number of edges, $\mathcal{C}$ is the set of edge communication costs and $\mathcal{T}$ is the set of node computation costs. The value $c_{i,j} \in \mathcal{C}$ is the communication cost incurred along the edge $e_{i,j} = (n_i, n_j) \in E$, which is zero if both nodes are mapped in the same processor. The value $\tau_i \in \mathcal{T}$ is the execution time of node $n_i \in V$.

A task is an indivisible unit of computation which may be an assignment statement, a subroutine or even an entire program. We assume that tasks are convex, which means that once a task starts its execution it can run to completion without interrupting for communications, Sarkar [13]. In the task computation, a task waits to receive all data in parallel before it starts its execution. As soon as the task completes its execution it sends the output data to all successors in parallel.

Scheduling is defined by a processor assignment mapping, $PA(n_j)$, of the tasks onto the p processors and by a starting time mapping, $ST(n_j)$, of all nodes onto the real positive numbers set. $CT(n_j) = ST(n_j) + \tau_j$ is defined as the completion time of task n_j in this schedule.

Fig. 1(a) shows a weighted DAG with all computation weights assumed to be equal to 1. Fig. 1(b) shows a processor assignment using 2 processors. Fig. 1(c) shows a *Gantt chart* of a schedule for this DAG. The Gantt chart completely describes the schedule since it defines both $PA(n_j)$ and $ST(n_j)$. The scheduling problem with communication delay has been shown to be NP-complete for a general task graph in most cases, Chretienne [2], Papadimitriou and Yannakakis [10], Sarkar [13],

Performance bounds. Let $PT_{opt}(G)$ be the length of an optimal schedule for task graph G. Then

$$PT_{opt}(G) \geq max(CP, \frac{Seq}{p}) \tag{1}$$

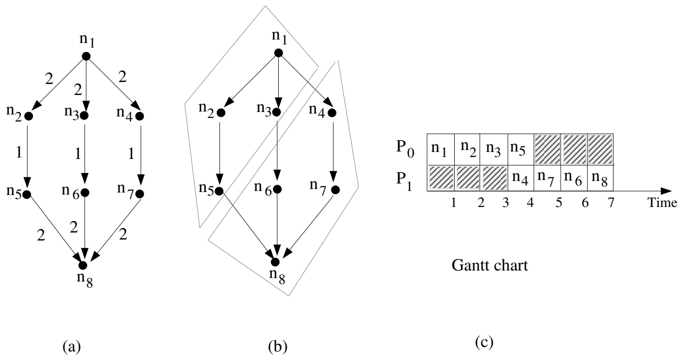

Figure 1: (a) A DAG with node weights equal to 1. (b) A processor assignment of nodes. (c) The Gantt chart of a schedule.

where p is number of processors, CP is the length of the critical path assuming zero communication between the edges, and Seq is the sequential execution time which is the summation of all task weights.

Granularity. To incorporate communication in the bounds we will need to use the concept of granularity. Let $SUCC(n_x)$ be the set of tasks as the successors of task n_x and $PRED(n_x)$ be the set of tasks as the predecessors of task n_x. In [7], we define the grain of a task as follows:

$$g(n_x) = \min\{\frac{\min_{n_k \in PRED(T_x)}\{\tau_k\}}{\max_{T_k \in PRED(T_x)}\{c_{k,x}\}}, \frac{\min_{n_k \in SUCC(T_x)}\{\tau_k\}}{\max_{n_k \in SUCC(T_x)}\{c_{x,k}\}}\}$$

Then the *granularity* of G is:

$$g(G) = \min_{n_x \in V}\{g(n_x)\}.$$

We call a DAG *coarse grain* if $g(G) \geq 1$. For a coarse grain DAG, each task receives or sends a small amount of communication compared to the computation of its neighboring tasks. When there is a sufficient number of processors, we say that a schedule uses *nonlinear clustering* if it assigns independent tasks in the same processor, otherwise *linear clustering*. In [7], we prove the following theorem:

Theorem 1 *If $g(G) \geq 1$, a schedule that uses nonlinear clustering can be transformed into a schedule that uses linear clustering with equal or shorter parallel time.*

Thus there exists a schedule with linear clustering that attains the optimal solution. This result will be useful in the derivation of lower bounds for GE and GJ later on.

In this paper, we consider *asymptotic* lower bounds instead of exact for the sake of simplicity. By asymptotic bounds we mean that certain lower-order terms of problem parameters, e.g. the size of the matrix n, are ignored when computing lower bounds. Thus the lower bounds are approximations which converge to the exact lower bound when the parameter, say the size of the matrix n, is sufficiently large.

We also assume that the processors are *fully connected.* This assumption implies that the performance bounds are valid for any processor topology.

3 Parallelization of the Gauss-Jordan Algorithm

We first discuss the column oriented partitioning of Gauss-Jordan algorithm and then a submatrix based column block partitioning.

3.1 Column partitioning

Gauss-Jordan kji form

for $k = 1 : n$
  for $j = k + 1 : n + 1$
    $T_k^j : \{ \quad a_{k,j} = a_{k,j}/a_{k,k}$
      for $i = 1 : n$ **and** $i \neq k$
        $a_{i,j} = a_{i,j} - a_{i,k} * a_{k,j}$
      end$\}$
  end
end

Figure 2: The kji form of GJ with interior loop task partitioning.

Figure 2 shows the kji form of the GJ algorithm [5], with interior loop partitioning. The interior loop defines task T_k^j which uses column k to operate (modify) column j of matrix A. We assume that the matrix is partitioned into columns data units to be consistent with the data accessing pattern of the tasks. The dependence task graph is given in Figure 3 [3, 9, 12]. Task T_k^{k+1} is a broadcasting node, sending the same column $k+1$ to all T_{k+1}^j, $j = k+2 : n+1$. This DAG has a degree of parallelism (the width of the DAG) equal to n.

The communication and computation weights can be estimated as follows. For T_k^j there are about $W = n\omega$ computation operations for each task, where ω is the time for performing "$a_{i,j} = a_{i,j} - a_{i,k} * a_{k,j}$". If α is the startup time for processor communication and β be the transmission speed, then the communication delay of sending a column between two neighbor processors is $C = \alpha + n\beta$, see Dunigan[6]. Therefore, the granularity of the GJ DAG in Figure 3 is

$$g(GJ) = \frac{W}{C} = \frac{n\omega}{\alpha + n\beta} = \frac{\omega}{\frac{\alpha}{n} + \beta} \approx \frac{\omega}{\beta}$$

where the "$\approx$" assumes n is large enough.

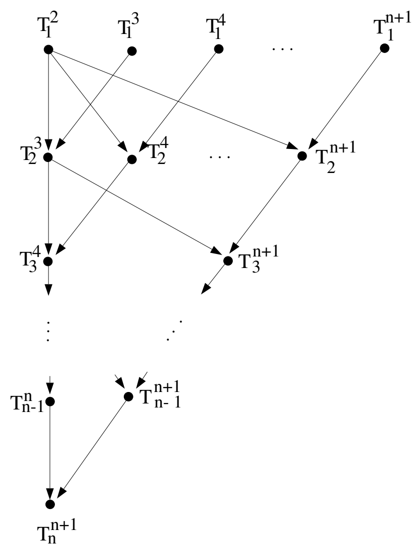

Figure 3: The GJ DAG.

We can easily see that the critical path length without communication is $CP = nW$, and the sequential time $Seq \approx \frac{Wn^2}{2}$. Then a general lower bound based on Inequality 1 is:

$$PT_{opt}(GJ) \geq nW \quad \text{If} \quad p \geq \frac{n}{2}, \qquad \text{and} \qquad PT_{opt}(GJ) \geq \frac{n^2 W}{2p} \quad \text{If} \quad p \leq \frac{n}{2}.$$

We now derive tighter bounds by considering communication delays and using our results for linear clustering given in Theorem 1.

Theorem 2

$$PT_{opt}(GJ) \geq nW \; + \; (n-1)C, \qquad \textit{If } W \geq C$$

$$PT_{opt}(GJ) \geq (2n-1)W, \qquad \textit{If } W < C.$$

Proof. If $W \geq C$, then $g(GJ) \geq 1$. We will derive this bound assuming that there is a unlimited number of processors. From Theorem 1. we can assume that an optimal solution uses linear clustering.

We define a *layer* k of the GJ DAG in Fig. 3 as the set of tasks $\{T_k^{k+1}, \ldots, T_k^{n+1}\}$. We will prove that the completion time of each task T_k^j at layer k satisfies

$$CT(T_k^j) \geq kW + \; (k-1)C \text{ where } j = k+1, \cdots, n+1.$$

This is trivial for $k = 1$. Suppose it is true for all tasks at layer $k-1$. We will prove that it is also true for the tasks at layer k. We examine the completion time for each task T_k^j at layer k. Since each task has two incoming edges from tasks at layer $k-1$, and linear clustering zeros at the most one edge, task T_k^j has to wait at least $C = \alpha + n\beta$ time to receive the message from one of its two predecessors, say T_{k-1}^r, at layer $k-1$. Therefore

$$CT(T_k^j) \geq CT(T_{k-1}^r) + C + W.$$

From the induction hypothesis we have that

$$CT(T_{k-1}^r) \geq (k-1)W + (k-2)C$$

which implies

$$CT(T_k^j) \geq kW + (k-1)C.$$

Since $PT_{opt}(GJ) = CT(T_n^{n+1})$, the statement is true.

If $W < C$, then we can construct a graph called GJ* with all computation and communication weights equal to W. Then any optimal solution for the

GJ DAG is a legal schedule for the GJ* DAG. And an optimal schedule of GJ* has a length shorter or equal to that for GJ.

For GJ* DAG, by using the same argument for case $W \geq C$, we have $PT_{opt}(GJ^*) \geq nW + (n-1)W$. This is also a lower bound for the GJ DAG. ■

Next we provide another lower bound in the case of a limited number of processors p.

Theorem 3

$$PT_{opt}(GJ) \geq \frac{n^2W}{2p} + \frac{(W+C)^2p}{2(W+2C)}, \qquad \textit{If } W \geq C.$$

$$PT_{opt}(GJ) \geq \frac{n^2W}{2p} + \frac{4Wp}{6}, \qquad \textit{If } W < C$$

Proof: Consider an optimum schedule and its parallel time PT_{opt}. In Fig. 3, we select an integer k $(1 < k < n)$. Assume that task T_k^{k+1} has a starting time ST_k in the optimal schedule. Let $H_k = PT_{opt}(GJ) - ST_k$.

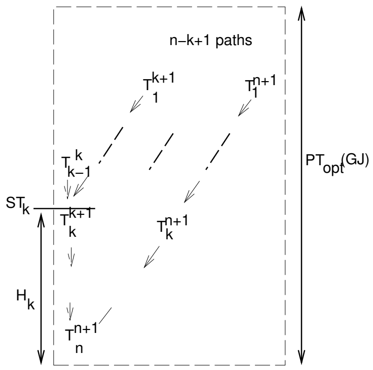

Figure 4: Examining $n-k+1$ paths. Each path has a computational length of nW.

The task T_{k-1}^k and all predecessors of T_{k-1}^k have to be executed in the interval between 0 to ST_k. The total amount of arithmetic work for these tasks is $\frac{k^2W}{2} + O(k)$.

Let us examine the $n-k+1$ paths starting from $T_1^{k+1}, T_1^{k+2}, \cdots$ and T_1^{n+1} as depicted in Fig. 4. These paths end at T_n^{n+1} and each of them has a computational length of nW. For each path, assume that an amount t of its computational work executes in the interval between ST_k and PT_{opt} and $nW - t$ in the interval between 0 and ST_k. Since t must be less than or equal to H_k, then the part of computation work at each path that must be finished before ST_k is at least $nW - H_k$ if $nW \geq H_k$ or 0 otherwise.

Thus

$$PT_{opt}(GJ) \geq H_k + \frac{k^2W/2 + (n-k+1)(nW - H_k)}{p}$$

$$\approx \frac{n^2W + (n-k)^2W}{2p} + H_k(1 - \frac{n-k}{p}).$$

If $W/C \geq 1$, then using the result of Theorem 2, we can show that

$$H_k \geq (n-k+1)W + (n-k)C \approx (n-k)(W+C).$$

Set $K_1 = n - k$, then

$$PT_{opt}(GJ) \geq B(K_1) = \frac{(n^2 + K_1^2)W}{2p} + K_1(W+C)(1 - \frac{K_1}{p}).$$

We find K_1 that maximizes $B(K_1)$ by differentiating $B(K_1)$ when $K_1 \leq p$. We get the maximum point achieved when $0 \leq K_1 = \frac{(W+C)p}{W+2C} \leq p$. Then

$$PT_{opt}(GJ) \geq B(\frac{(W+C)p}{W+2C}) = \frac{n^2W}{2p} + \frac{(W+C)^2p}{2(W+2C)}.$$

When $W < C$, we construct a DAG called GJ* from GJ by making all computation and communication weights in the GJ graph equal to W. Since any schedule for the GJ DAG is valid for the GJ* DAG then the lower bound is:

$$PT_{opt}(GJ) \geq PT_{opt}(GJ^*) \geq B(\frac{2Wp}{3W}) = \frac{n^2W}{2p} + \frac{2Wp}{3}).\blacksquare$$

Let us now look at the parameters of some commercially available message passing architectures to see for what values of n, $g(GJ) = W/C \approx \omega/\beta \geq 1$ holds true. Table 1 lists the parameters of several message passing architectures for single precision arithmetic.

Observe that for the first generation of hypercube machines (Intel iPSC/2 and nCUBE-1), the column-based partitioning is coarse grain for a reasonable

	iPSC/2	iPSC/860	nCUBE-1	nCUBE-2
α (microseconds)	697	136	383.6	200
β (microseconds/word)	1.6	1.6	10.4	2.4
ω (microseconds/flop)	11.4	0.2	35.1	1.6
$g(GJ) \geq 1$ when $n \geq$	72		16	
$\lim_{n\to+\infty} g(GJ) = \omega/\beta =$	7.1	0.125	3.4	0.667

Table 1: The granularity of the GJ DAG in different architectures.

size of n. For the second generation of hypercubes, nCUBE-2 and iPSC/860, the processor speed is very high compared to communication speed and column-based partitioning does not produce coarse grain tasks for any size of n. There are many approaches to increase the granularity of the GJ DAG and we describe one next.

3.2 Block partitioning

A popular approach for increasing the granularity of GJ is BLAS-3 block partitioning, see Dongarra et. al. [5]. The matrix of $n \times n$ is divided into $N \times N$ submatrices and each submatrix has size of $r \times r$ where $N = n/r$. Each task T_k^j in the block GJ DAG is operating on a block of columns composed of N submatrices.

The task definition for block GJ without pivoting can be described as:

$$\boxed{\begin{array}{ll} T_k^j : \{ & A_{k,j} = A_{k,k}^{-1} * A_{k,j} \\ & \textbf{for } i = 1 : N \textbf{ and } \ i \neq k \\ & \quad A_{i,j} = A_{i,j} - A_{i,k} * A_{k,j} \\ & \textbf{end}\} \end{array}}$$

where $k = 1, \cdots, N$ and $j = k+1, \cdots N+1$. Notice that the task T_k^{N+1} is operating on the right hand side column b. Thus the definition T_k^{N+1}, $k = 1, 2, \cdots$, involves only matrix-vector multiplication. For simplicity, we will exclude these tasks T_k^{N+1}, $k = 1, 2, \cdots$, when computing lower bounds. Such exclusion does not affect the correctness of asymptotic lower bound calculation.

The dependence structure of the block GJ DAG remains the same as in Figure 3 but the degree of parallelism is reduced from n to $N+1$ where $N = n/r$. Except tasks T_k^{N+1}, the computation size of T_k^j increases to about $W = Nr^3\omega$ and a message communicated between tasks is a block column of size Nr^2. Thus the communication delay is $C = \alpha + Nr^2\beta$.

We call the block GJ graph as the BGJ DAG. The granularity $g(BGJ)$ is asymptotically equal to $\frac{r\omega}{\beta}$, increases by a factor of r compared to that of the column GJ DAG. To make $g(BGJ) \geq 1$ the minimum submatrix size r is $\lceil \frac{\beta}{\omega} \rceil$, which becomes 2 for the nCUBE-2 and 8 for the Intel iPSC/860.

The lower bound analysis for the BGJ DAG is similar to that for the column-oriented GJ DAG. We summarize these results as follows:

Theorem 4 *Let $W = Nr^3\omega$ and $C = \alpha + Nr^2\beta$.*

$$PT_{opt}(BGJ) = max((N-1)W + (N-2)C,\ \frac{N^2W}{2p} + \frac{(W+C)^2p}{2W+2C}), \quad \textit{If } W \geq C.$$

$$PT_{opt}(BGJ) \geq max((2N-3)W,\ \frac{N^2W}{2p} + \frac{2pW}{3}), \quad \textit{If } W < C.$$

3.3 Gauss-Jordan with partial pivoting

for $k = 1$ **to** n
$\boxed{T_k^k : \{ \text{ Find } |a(l,k)| = max\{|a(k,k)|, \cdots, |a(n,k)|\}; \ \text{piv(k)=l; swap(a(l,k), a(k,k));}}$
  for $j = k+1$ **to** $n+1$
    T_k^j:{$swap(a(piv(k),j), a(k,j))$
      $a(k,j) = a(k,j)/a(k,k)$
      For $i = 1$ **to** n, $i \neq k$
        $a_{i,j} = a_{i,j} - a_{i,k} * a_{k,j}$
      end}
  end
end

Figure 5: GJP: GJ with partial pivoting and its task partitioning.

A column-based partitioning for the Gauss-Jordan algorithm with pivoting is shown in Fig. 5. Computing the lower bound for GJ with partial pivoting is complicated by the addition of pivoting task T_k^k ($k = 1, 2, \cdots$). We call this DAG as GJP. This graph has a structure similar to Fig 8. We take a simple approach in this paper by deleting the tasks T_k^k, $k = 1, 2, \cdots$ from the graph. We call this new graph as the reduced graph GJP/R. It can be shown that an optimal schedule for GJP is a legal schedule for the GJP/R graph and $PT_{opt}(GJP) \geq PT_{opt}(GJP/R)$.

Thus we can conduct the lower bound analysis on the GJP/R graph. Then the asymptotic lower bounds computed in the previous subsections are still valid for GJP/R and as a result for GJP.

4 Gaussian-Elimination

```
for k = 1 to n - 1
  | T_k^k : { Find |a(l,k) = max{a(k,k), ..., |a(n,k)|};
  | piv(k)=l; swap(a(l,k), a(k,k));
    for j = k + 1 to n
      | T_k^j:{swap(a(piv(k),j), a(k,j))
      |    a(k,j) = a(k,j)/a(k,k)
      |    For i = k + 1 to n
      |       a_{i,j} = a_{i,j} - a_{i,k} * a_{k,j}
      |    end}
    end
end
```

Figure 6: LU with partial pivoting and its column-based partitioning.

We only consider the LU part of the Gaussian Elimination algorithm. The column-oriented with pivoting LU factorization is shown in Fig. 6. The block LU factorization without pivoting is shown in Figure 7. The $n \times n$ matrix is divided into $N \times N$ submatrices and each submatrix has size of $r \times r$ where $N = n/r$. The resulting task graph is similar to GJ algorithm but the weight distribution differs. The dependence graph is shown in Fig. 8. Each task T_k^j is operating on a block column composed of N elements and each element is an $r \times r$ submatrix. Task T_k^k has a weight $(N - k + 1)r^3\omega/2$ and it communicates with T_k^j $(N - k + 1)$ submatrices. Task T_k^j has a weight $(N - k + 1)r^3\omega$ and it sends the part of column block j of size $(N - k)$

submatrices to T^j_{k+1}. This graph is coarse grain in the top part but becomes fine grain in the bottom. The block LU algorithm with partial pivoting is relatively complicated [5]. The performance lower bound of block LU without partial pivoting can be used for block LU with partial pivoting.

for $k = 1$ **to** N
  T_k^k : { Factorize $A_{k,k}$ as $L_k * U_k$
    for $i = k+1$ **to** N
      $A_{i,k} = A_{i,k} * U_k^{-1}$
    end}
  for $j = k+1$ **to** N
    T_k^j : { $A_{k,j} = L_k^{-1} * A_{k,j}$
      For $i = k+1$ **to** N
        $A_{i,j} = A_{i,j} - A_{i,k} * A_{k,j}$
      end}
  end
end

Figure 7: Block LU factorization and its task partitioning.

We first derive the following general lower bound for mapping this block GE/LU graph (called BLU): The sequential time: $Seq \approx N^3 r^3 \omega/3 = n^3\omega/3$. The critical path excluding communication is $\{T_1^1, T_1^2, T_2^2, \cdots T_N^N\}$. Thus

$$CP = \sum_{k=1}^{N} (\frac{N-k+1}{2} + (N-k+1))\omega r^3 \approx \frac{3N^2 r^3 w}{4}.$$

Thus

$$PT_{opt}(BLU) \geq max(\frac{n^3\omega}{3p}, \frac{3n^2 r\omega}{4}).$$

Next we incorporate communication overhead on the bound. If we delete all tasks T_k^k, $k = 1, \cdots, N$, the remaining graph has a structure similar to Fig. 3 except that weights are monotonically decreasing from the top to bottom. We first find the place the local granularity becomes less than 1. The new graph has a local grain value

$$g(T_k^j) \approx \frac{(N-k)r^3\omega}{\alpha + (N-k)r^2\beta}$$

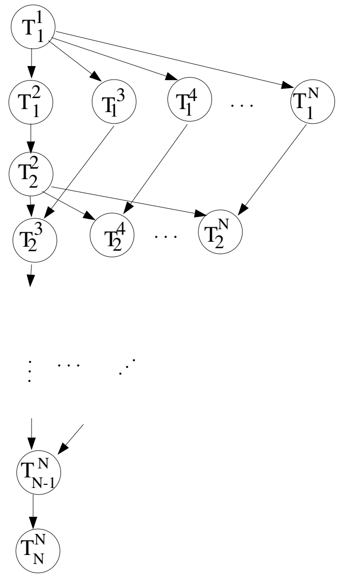

Figure 8: The task graph of block LU factorization with or without pivoting.

When N is very large, the condition to make $g(T_k^j) \geq 1$ is $r\omega/\beta \geq 1$ and $k \leq N - \frac{\alpha}{r^2(r\omega-\beta)}$. Let $s = N - \frac{\alpha}{r^2(r\omega-\beta)}$.

Lemma 1 *If* $r\omega/\beta \geq 1$,

$$PT_{opt}(BLU) \geq W_1 + \sum_{i=2}^{s}(W_i + C_i)$$

where $W_i = (N-i+1)r^3\omega$ *and* $C_i = \alpha + \beta(N-i+1)r^2$.

Proof. The proof is similar to that of Theorem 2. We exclude the bottom fine grain part of this graph, i.e., tasks T_k^j with $k \geq s+1$.

Since the granularity is greater than one, the optimum parallel time must be achieved by a linear clustering. We define a *layer* k in Fig 8 as the set of tasks $\{T_k^{k+1}, \ldots, T_k^N\}$. We will prove that the completion time of each task T_k^j at layer k satisfies

$$CT(T_k^j) \geq W_1 + \sum_{i=2}^{k}(W_i + C_i).$$

This is trivial for $k = 1$. Suppose it is true for tasks at layer $k-1$. We examine the completion time for each task T_k^j at layer k. Since each task has two incoming edges from tasks at layer $k-1$, and linear clustering zeros at the most only one edge, task T_k^j has to wait at least C_k to receive the message from one of its two predecessors, say T_{k-1}^r, at layer $k-1$. Therefore

$$CT(T_k^j) \geq CT(T_{k-1}^r) + W_k + C_k.$$

Then using the induction hypothesis we prove the statement. ■

Notice when N is very large or r is large, $s \approx N$, and then we can simplify the bound as

$$W_1 + \sum_{i=2}^{N-1}(W_i + C_i) \approx \frac{N^2 r^2}{2}(rw + \beta).$$

For the case of $r\omega < \beta$, we can transform the BLU graph to a graph BLU* by replacing weight C_i with W_i. Thus we have the following theorem:

Theorem 5 *If $r\omega \geq \beta$,*

$$PT_{opt}(BLU) \geq \frac{N^2 r^2}{2}(rw + \beta).$$

If $r\omega < \beta$,

$$PT_{opt}(BLU) \geq N^2 r^3 w.$$

We further estimate a lower bound for block LU when we are given a finite number of processors p.

Theorem 6 *Let $W = r^3\omega$ and $C = r^2\beta$.*

If $r\omega \geq \beta$

$$PT_{opt}(BLU) \geq \frac{N^3 W}{3p} + \frac{(W + C)^3 p^2}{6(W + 1.5C)^2}.$$

If $r\omega \leq \beta$

$$PT_{opt}(BLU) \geq \frac{N^3 W}{3p} + \frac{16Wp^2}{75}.$$

Proof: For Fig. 8, we select an integer k ($1 < k < N$) and use a technique similar to that for Theorem 3 to prove the statement. Assume that task T_k^{k+1} has a starting time ST_k in an optimal schedule. Let $H_k = PT_{opt}(BLU) - ST_k$.

Thus T_{k-1}^k and all predecessors of T_{k-1}^k have to be executed in the interval between 0 to ST_k. The total amount of work in these tasks is about

$$\sum_{i=1}^{k-1}(k-i+1)(N-i+1)W \approx \frac{Nk^2W}{2} - \frac{k^3W}{6}.$$

We examine $N-k$ paths starting from $T_1^{k+1}, \cdots T_1^N$ and ending in T_N^N. For each of those paths, the total amount of computational work has to be performed before time ST_k is:

$$\sum_{i=1}^{N}(N-i+1)W - H_k \approx \frac{N^2W}{2} - H_k.$$

Thus

$$PT_{opt}(BLU) \geq H_k + \frac{\frac{Nk^2W}{2} - \frac{k^3W}{6}}{p} + \frac{(N-k)(\frac{N^2W}{2} - H_k)}{p}.$$

Let $K_1 = N-k$, then we can simplify the above expression as:

$$PT_{opt}(BLU) \geq \frac{N^3W}{3p} + \frac{K_1^3W}{6p} + H_k(1 - \frac{K_1}{p}).$$

If $W \geq C$ (i.e. $r\omega \geq \beta$), we choose $K_1 < p$. By Lemma 1, $H_k \geq K_1^2(W+C)/2$ and

$$PT_{opt}(BLU) \geq B(K_1) = \frac{N^3W}{3p} + \frac{K_1^3W}{6p} + \frac{K_1^2(W+C)(1-\frac{K_1}{p})}{2}.$$

We perform the differentiation of $B(K_1)$

$$\frac{d\ B(K_1)}{d\ K_1} = \frac{0.5K_1^2W}{p} + K_1(W+C) - \frac{1.5K_1^2(W+C)}{p},$$

and

$$\frac{d^2\ B(K_1)}{d\ K_1^2} = W + C - \frac{(2W+3C)K_1}{p}.$$

To make $\frac{d^2\ B(K_1)}{d\ K_1^2} < 0$, we have $K_1 > \frac{(W+C)p}{2W+3C}$. Let $\frac{d\ B(K_1)}{d\ K_1} = 0$, then $K_1 = \frac{(W+C)p}{W+1.5C}$. Apparently $\frac{(W+C)p}{W+1.5C} > \frac{(W+C)p}{2W+3C}$. Thus

$$PT_{opt}(BLU) \geq \frac{N^3W}{3p} + \frac{(W+C)^3p^2}{6(W+1.5C)^2}.$$

When $r\omega \leq \beta$, we construct a new graph from BLU such that communication weights C_i are replaced with computation weights W_i. We derive

$$PT_{opt} \geq B(\frac{2Wp}{2.5W}) = \frac{N^3 W}{3p} + \frac{16Wp^2}{75}. \blacksquare$$

5 Summary and Applications

We summarize our analysis for block GJ and GE/LU algorithms in Table 2. We have ignored the lower order terms assuming that $N = n/r$ is large. Notice that when n is large, the asymptotic lower bounds for column-partitioning is a special-case of column block partitioning. The lower bounds for algorithms without partial pivoting are also valid for algorithms with partial pivoting.

	$r\omega \geq \beta$	$r\omega < \beta$
GJ	$max(n^2(r\omega+\beta), 0.5(\frac{n^3\omega}{p} + \frac{nr(r\omega+\beta)^2 p}{r\omega+2\beta}))$	$max(2n^2 r\omega, \frac{n^3\omega}{2p} + \frac{2nr^2 p}{3})$
LU	$max(\frac{3n^2 r\omega}{4}, 0.5n^2(r\omega+\beta), \frac{n^3\omega}{3p} + \frac{(r\omega+\beta)^3 r^2 p^2}{6(r\omega+1.5\beta)^2})$	$max(n^2 r\omega, \frac{n^3\omega}{3p} + \frac{16r^3\omega p^2}{75})$

Table 2: A summary of the lower bounds.

Next we compare these lower bound results with the performance of parallelizing GJ/GE programs on nCUBE-2 using PYRROS [15]. PYRROS is an automatic scheduling and code generation tool for mapping task graphs on message-passing machines. The table contains the results for two different block partitions, $r = 8$ and $r = 16$ which result in parallelism degree of $N = 128$ and $N = 64$ respectively when $n = 1024$. The *Speedup-Bound* column is the speedup upper bound Seq/B using our lower bound result B in Table 2. We use $\alpha = 160\mu s$, $\beta = 2.4\mu s/word$, and $\omega = 2.4\mu s$ [6].

Table 4 lists the results for block Gauss-Jordan (GJ) algorithm without pivoting, with $r = 10$ and $n = 1000$. The sequential time is about 1150. Again we calculate speedup bound Seq/B and B is from Table 2. We also include the $Speedup - Bound1$ value which is calculated based on Inequality 1. We can see for this case, our bounds that incorporate communication delay provide sharper results than that using Inequality 1.

#Proc.	$r = 8$ $N = 128$ Speedup-Bound	PYRROS-Speedup	$r = 16$ $N = 64$ Speedup-Bound	PYRROS-Speedup
p=4	3.99	3.8	3.99	3.7
p=8	7.99	7.1	7.99	6.7
p=16	15.98	12.5	15.87	11.2
p=32	31.75	19.4	28.44	21.2
p=64	56.89	29.7	28.44	23.6

Table 3: The block LU with pivoting performance of PYRROS for $n = 1024$.

From these two experiments, we can see that lower bounds performance prediction is quite good.

	$r = 10$ $N = 100$		
#Proc.	Speedup-Bound	PYRROS-Speedup	Speedup-Bound1
p=4	3.99	3.99	4
p=8	7.99	7.6	8
p=16	15.6	14.1	16
p=32	29.0	24.3	32
p=64	45.3	35.1	50

Table 4: The GJ PYRROS performance and the performance bounds.

References

[1] P. R. Cappello, Gaussian elimination on a hypercube automaton, *Journal of Parallel and Distributed Computing 4*, (1987), 288-308.

[2] P. Chretienne, *Task Scheduling over Distributed Memory Machines*, Proc. of the International Workshop on Parallel and Distributed Algorithms, (North Holland, Ed.), 1989.

[3] M. Cosnard, B. Tourancheau, and G. Villard, *Gaussian Elimination on Message Passing Architecture*, Lecture Notes in Computer Science 297, Berlin: Springer-Verlag, 1987, pp. 611-628.

[4] G. Davis, *Column LU Factorization with Pivoting on a Hypercube Multiprocessor*, SIAM J. Algebraic and Discrete Methods, vol. 7, pp. 538-550, 1986.

[5] J. J. Dongarra, L. S. Duff, D. Sorensen and H. A. van der Vorst. *Solving Linear Systems on Vector and Shared Memory Computers*, SIAM, 1991.

[6] T.H. Dunigan, Performance of the INTEL iPSC/860 and nCUBE 6400 Hypercube, ORNL TM-11790, Oak Ridge National Laboratory, Oak Ridge, TN, Nov. 1991.

[7] A. Gerasoulis and T. Yang, On the Granularity and Clustering of Directed Acyclic Task Graphs, *IEEE Trans. on Parallel and Distributed Systems*, June, 1993. Vol. 4, No. 6, June 1993, pp. 686-701.

[8] G. Golub and J.M. Ortega, *Scientific Computing : An Introduction to Parallel Computing*, Academic Press, 1993.

[9] R.E. Lord, J.S. Kowalik, and S. P. Kumar, *Solving Linear Algebraic Equations on an MIMD Computer*, Journal of the ACM, vol. 30, pp. 103-117, 1983.

[10] C. Papadimitriou and M. Yannakakis, *Towards on an Architecture-Independent Analysis of Parallel Algorithms*, SIAM J. Comput., vol. 19, pp. 322-328, 1990.

[11] I.C.F. Ipsen, Y. Saad and M. Schultz, Complexity of Dense Linear System Solution on a Multiprocessor Ring, *Linear Algebra and Appl.,* vol. 77, pp. 205-239, 1986.

[12] Y. Robert, B. Tourancheau and G. Villard, Data allocation strategies for the Gauss and Jordan Algorithms on a ring of processors. *Information Processing Letters* 31, (1989) 21-29.

[13] V. Sarkar, *Partitioning and Scheduling Parallel Programs for Execution on Multiprocessors*, The MIT Press, 1989.

[14] Y. Saad, *Gaussian Elimination on Hypercubes*, Parallel Algorithms and Architectures, Cosnard, M. et al. Eds., Elsevier Science Publishers, North-Holland, 1986.

[15] T. Yang and A. Gerasoulis, PYRROS: Static Task Scheduling and Code Generation for Message-Passing Multiprocessors, *Proc. of 6th ACM Inter. Confer. on Supercomputing,* Washington D.C., 1992, pp. 428-437.