PERFORMANCE STUDY OF *LU* FACTORIZATION WITH LOW COMMUNICATION OVERHEAD ON MULTIPROCESSORS

F. Desprez[*†], J. J. Dongarra[‡§] and B. Tourancheau[*¶]
The University of Tennessee
Computer Science Department
107, Ayres Hall
Knoxville, TN 37996-1301
USA
E-mail: [desprez,btouranc,dongarra]@cs.utk.edu

ABSTRACT

In this paper, we make efficient use of asynchronous communications on the LU decomposition algorithm with pivoting and a column-scattered data decomposition to derive precise computational complexities. We then compare these results with experiments on the Intel iPSC/860 and Paragon machines and show that very good performances can be obtained on a ring with asynchronous communications.

Keywords: LU factorization, Pipelining, Parallel Complexity.

1. Introduction

This paper presents an analytical estimation of the LU factorization algorithm on a distributed-memory message-passing multiprocessor.

Numerous methods have been proposed for LU factorization (see [PBKP92] and the related works of [Saa86b,Cap87,CTV87,RT88,CRT89,DO90,Rob90]). For example, [CG87] advocates partial pivoting and load balancing in row-wise methods with a straightforward parallel triangular solve algorithm, but [LC89] shows that the parallel triangular solve algorithm can have the same performance with column-wise storage. In [RTV89], the panel-wrapped column-distribution method is proved to be efficient because it leads to a good load-balancing between computation and communication; and, in practice, this method has given good results with blocked computations [RT88,DO90]. Block-wise distribution is introduced in [CDW92], and [BDL91,DGW92] show

[*] On leave from LIP, CNRS URA 1398, ENS Lyon, 46 allée d'Italie, 69364 Lyon Cedex 07, France.

[†] This work was supported by MRE grant No. 974, the CNRS-NSF grant No. 950.22/07 and the research program C3.

[‡] and also Oak Ridge National Laboratory, Mathematical Sciences Section, P.O. Box 2008, Bldg. 6012, Oak Ridge, TN 37831-6367, USA

[§] This work was supported in part by the National Science Foundation under grant ASC-8715728, the National Science Foundation Science and Technology Center Cooperative Agreement CCR-8809615, the DARPA and ARO under contract DAAL03-91-C-0047

[¶] This work was supported in part by CNRS-NSF grant 950.223/07, Archipel SA and MRE under grant 974, the National Science Foundation under grant ASC-8715728, the National Science Foundation Science and Technology Center Cooperative Agreement CCR-8809615, the DARPA and ARO under contract DAAL03-91-C-0047, PRC C^3, and the DRET.

that the communication can be reduced by using a block-wrapped data decomposition of the data.

To understand the effectiveness of these various methods and to be able to *predict* performance on a target machine, we must carry out an analytical computational complexity analysis.

We focus on column-scattered data distribution and the column-wise kji, or "right looking" elimination method with pivoting. Our work is divided in two parts and we introduce different cases depending on the target machine parameters that correspond to different critical paths of the execution. In the first part we study the algorithm with synchronous communication and propose its computational complexity. We then compare the model predictions with the execution timings[a] for a complete network and a ring topology. In the second part, we introduce the possibility of overlapping the communication by the computation (i.e., taking advantage of asynchronous communication to hide their cost by doing computations at the same time). Using our earlier analysis, we propose the computational complexities for both algorithms and compare them to the corresponding experiments and the results obtained previously. This shows that a nearly complete overlap of communications can be achieved. Thus, we obtain quasi-optimal performance with a simple column-scattered data distribution on a ring of processors.

In section 2 and 3, we describe the LU decomposition and the used parallel version. In section 4, the model for the computational complexity analysis and its parameters is introduced. Section 5 and 6 correspond to the synchronous and asynchronous communication parts respectively. In section 8, we present the experiments and we conclude and present our future works in section 9.

2. Gaussian Elimination and LU decomposition

Gaussian elimination can be used in the solution of a system of equations $Ax = b$. This process transforms the matrix A in a triangular form with an accompanying update of the right-hand side, so that the solution of the system $Ax = b$ is straightforward by a triangular solve.

LU factorization uses the same algorithm and converts the matrix A in two matrices L and U, where $A = LU$ and L and U are lower and upper triangular, respectively. Hence, many systems can be solved by two triangular solves, $Ly = b$ and $Ux = y$.

There are many versions of the LU algorithm depending on the way the three internal loops are nested [Rob90,GL89]. We study the kji, or "right looking" form, which is most suitable for parallel implementation (since the matrix can be distributed by columns, the pivoting is done inside each processor without communication [PBKP92]).

3. Parallel kji Version of the LU Algorithm with Column-scattered Data Distribution

[a] All the experiments are ran on the Intel iPSC/860 and Paragon machines.

```
$me = my\_proc\_id()$
for $k = 0$ to $n-2$
   if $(data\_alloc(k) == me)$
      $S(k)$ /* scaling of column $k$ of $A$.  */
   endif
   $C(k)$ /* broadcast of column $k$ */
   for $(j \geq k$ and $data\_alloc(j) == me)$
      $U(k,j)$ /* updating of column $j$ */
   endfor
endfor
```

Figure 1: Parallel LU decomposition algorithm ($data\_alloc(i)$ returns the id. of the processor that owns the column i of A).

```
Task $S(k)$
   $pivot = index\_of\_max(k)$
   $interchange((k,k),(pivot,k))$
   $c = \frac{1}{a_{kk}}$
   for $i = k+1$ to $n-1$
      $a_{ik} = a_{ik} * c$
   endfor
```

```
Task $U(k,j)$
   $interchange((k,j),(pivot,j))$
   for $i = k+1$ to $n-1$
      $a_{ij} = a_{ij} - a_{ik} * a_{kj}$
   endfor
```

Figure 2: Algorithms for the tasks S and U in the LU decomposition ($index\_of\_max(i)$ returns the index of the maximum absolute value in the column i of A, $interchange((i,j),(m,n))$, interchanges elements $A(i,j)$ and $A(m,n)$).

Parallel versions of the LU algorithm with column-scattered data distribution of the matrix A are described in [Saa86a,Saa86b,RTV89,Rob90] and summarized in Figure 1[b]. Depending on the topology, the difference between the methods is the manner in which the elimination column is sent to all the processors. Two well-known methods for broadcast are the minimum spanning tree broadcast and the pipeline ring (unidirectional broadcast along the ring) algorithms [Saa86a,RTV89,Rob90].

In the following parallel algorithm, we assume that the data are equally distributed in a wrapped manner along a virtual ring. Thus, with our algorithm, the same number of columns is assigned to each processor, and, as the column distribution is wrapped, a given processor has a scattered collection of columns. This approach ensures a good balance of the computational load between the processors.

For the parallel kji LU decomposition, we selected the following set of computational tasks: S is the scaling[c] (Figure 2), U is the updating[d] of the column j (Figure 2), and C is the communication broadcast (Figure 3) of the elimination

[b] Note that the multipliers, L, are saved in the array A in place of the elements that would become zero in task S.

[c] It comprises the pivot search,the interchange, and the scaling of the elimination column.

[d] It comprises the interchange of pivot elements and the updating of the column j.

```
Task $C(k)$
  if (data_alloc(k) == me)
    $mess := concatenate(pivot, [k+1 : n-1, k])$
  endif
  $broadcast(mess, n-k)$
```

Figure 3: Algorithm for the task C of LU decomposition ($concatenate(i, [a:b,c])$, concatenates in the same message buffer i and elements $a, a+1, \ldots, b$ of column c of A, $broadcast(X, i)$, broadcasts array X of length i).

column k to the processors using message passing primitives[e].

Thus, the LU decomposition of an $n*n$ matrix on p processors is $(n-1)$ tasks S_j, and assuming n and p are even, $(np - \frac{p^2}{2} + \frac{p}{2})$ tasks C_j and $(\frac{n^2}{2} + \frac{n}{2})$ tasks U.

4. Models

Our target machines are distributed-memory parallel multicomputers. We assume that the p processors $(0..p-1)$ are identical and that the same program runs (at the same time) on each processor. The communication network topology is assumed to be a fully connected network or a ring; the results for the two configurations will be compared.

In the first part, we assume that the communication protocol is synchronous. On the other hand, in the second part, we assume that the communication protocol is asynchronous; thus, each processor has independent units for communication (which manage direct-memory access) and computation. It is therefore possible to perform in parallel on the same node at least unidirectional data transfers on each link (half duplex and two-port assumptions) and arithmetic operations.

The time needed to communicate a message of size m between two neighboring processors is modeled like in [SS89] as the startup time β_C plus the length m times the transfer time[f] τ_C. We assume that the messages are sent in one block.

For arithmetic operations, the arithmetic logical unit of the CPU is assumed to be super-scalar. We also use a linear $\beta + m\tau$ model for its performance[g] on vector of size m. This model corresponds roughly to the currently available commercial machines, such as the Intel iPSC/860 [Dun90] and Paragon [EK93,Int93].

In the remainder of this paper, we call β_S and τ_S the parameters corresponding to the cost of the whole S task (which contains the inversion of the pivot element, the search for the maximum of the vector, the interchange of these two elements, and the scale of the vector by a scalar using the Level 1 BLAS DSCAL[DCDH90] function.

We define β_U and τ_U as the parameters for the cost of the U task (which contain the pivot interchanges and the multiply and add operations on the vector using the DAXPY function of the Level 1 BLAS).

[e] Note that C can be sends, receive and sends, or a receive call, depending of the broadcast strategy and the processor where it is executed.

[f] i.e. the inverse of the throughput of the communication channels.

[g] Note that in all our experiments we use 64-bit double precision floating-point arithmetic.

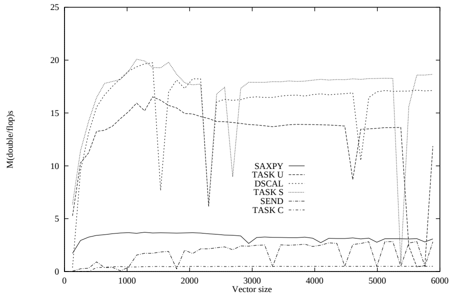

Figure 4: Intel Paragon execution performances as a function of the vector length for the tasks U, S and C compared with their corresponding basic kernels (`daxpy, dscal, send`) in Mflops or in Mwords per second.

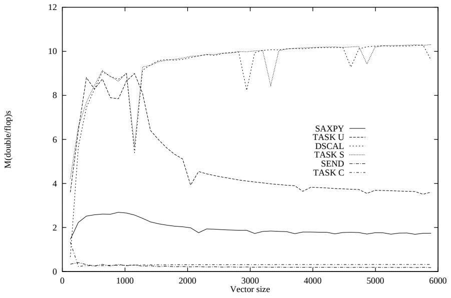

Figure 5: Intel iPSC/860 execution performances as a function of the vector length for the tasks U, S and C compared with their corresponding basic kernels (`daxpy, dscal, send`) in Mflops or in Mwords per second.

These parameters of the target machine are determined by experiments shown in Figures 4 and 5[h]. The corresponding asymptotical values of the τ parameters are given in Table 1. Notice that the pivot search and interchange are costly (task S is far from the peak performance of the `DSCAL` function) and that the target machine presents its best performance for vector of size around 1 K for the iPSC/860 and 1.5 K for the Paragon (because of memory and cache management).

Machine	$\tau_U(\mu sec/flop)$	$\tau_S(\mu sec/flop)$	$\tau_C(\mu sec/double)$
iPSC/860	0.097	0.57	5
Paragon	0.058	0.32	2

Table 1: Values of the machine parameters for the different tasks.

For our critical path analysis of LU decomposition, we use the notation of [LKK83], where a task is an indivisible unit of computational activity and the precedence binary relation between tasks is denoted by $\prec$, where if S and U belong to the set of tasks, then $S \prec U$ means that task S must complete execution before U commences execution.

5. Synchronous Communication

We begin with an analysis of the algorithm with synchronous communications and the complete network that will be compared with the ring topology.

5.1. Analysis of the algorithm on the Complete Network

With our model and when no overlap is allowed, the critical path of LU decom-

[h] Remark that the x-scale is different.

position is given by the precedence constraints of the sequential execution. Hence, the critical path described by edges in the time diagram of Figure 6 follows the following schema (where j represents the remaining local columns to update regarding the data allocation, the superscripts are the processors on which the task is executed):

$S(0) \prec C(0)^0 \prec C(0)^1 \prec U(0,j)^1 \prec S(1) \prec C(1)^1 \prec C(1)^2 \prec U(1,j)^2 \prec \ldots \prec$
$S(n-2) \prec C(n-2)^(p-2) \prec C(n-2)^(p-1) \prec U(n-2,j)^(p-1)$

Following this critical path, we compute the execution time analytically[i].

Proposition 1 *The parallel execution time of the LU algorithm on the complete network with no overlap of communication and computation is given by Equations 1 and 2 (where T_A meas the total execution time of task A).*

$$T_{LU}^{complete} = T_S^c + T_C^c + T_U^c \tag{1}$$

where,

$$\begin{aligned}
T_S^c &= \sum_{k=0}^{n-2} (\beta_S + (n-k-1)\tau_S) = (n-1)\beta_S + \frac{n(n-1)}{2}\tau_S \\
T_C^c &= \sum_{k=0}^{n-2} (\beta_C + (n-k)\tau_C) = (n-1)\beta_C + \frac{(n+2)(n-1)}{2}\tau_C \\
T_U^c &= \sum_{j=0}^{\frac{n}{p}-1} \left[\sum_{k=1}^{p} \left((\frac{n}{p} - j)(n - (j(p-1)+k))\tau_U + \beta_U \right) - (\beta_U + \frac{n^2}{p}\tau_U) \right] \\
&= (n-1)\beta_U + \left(\frac{n^3}{3p} + \frac{n^2}{4} - \frac{3n^2}{4p} + \frac{n}{4} - \frac{np}{12} \right) \tau_U
\end{aligned}$$

Thus,

$$\begin{aligned}
T_{LU}^{complete} &= (n-1)(\beta_U + \beta_S + \beta_C) + \frac{n(n-1)}{2}(\tau_S + \tau_C) \\
&+ \left(\frac{n^3}{3p} + \frac{n^2}{4} - \frac{3n^2}{4p} + \frac{n}{4} - \frac{np}{12} \right) \tau_U
\end{aligned} \tag{2}$$

5.2. Analysis of the LU algorithm on the Ring

Figure 6 shows the critical path of the pipeline LU algorithm corresponding to the ring topology on the left. The ring introduces idle times because of the precedence edges between tasks $C(i-1)$ (send) and $C(i)$ (receive), which are now executed one by one from processor to processor. Depending on the machine parameters, these dependencies introduce various periods of idle time in the critical path. If we examine the "zoom" in Figures 7 and 8, we can distinguish two cases depending if $T_{S(k)} > T_{C(k-2)}$ (case 1) or not (case 2).

[i] Note that the critical path must follow the data allocation of matrix on the processors between the S_j task (on processor $data\_alloc(A(j))$) and S_{j+1} task (on processor $data\_alloc(A(j+1))$).

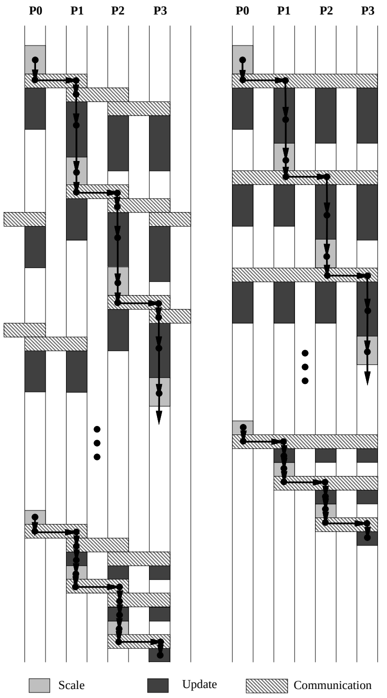

Figure 6: Time diagrams of synchronous algorithms (left on the ring, right on the complete network).

In order to determine into which case the algorithm fells, we study its corresponding computational complexity in case 1 :

$$\begin{aligned} T^r_{S(k)} &> T^r_{C(k-2)} \\ \beta_S+(n-k+1)\tau_S &> \beta_C+(n-k+2)\tau_C \end{aligned} \tag{3}$$

Generally (and it is true with our target machines), τ_C is at least one order bigger than τ_S. Thus $\tau_C - \tau_S > 0$ and inequality 3 can be rewritten as :

$$\begin{aligned} k &> \frac{\beta_C-\beta_S+(n+2)\tau_C-(n-1)\tau_S}{\tau_C-\tau_S} \\ k &> (n+2)+o(1), \end{aligned} \tag{4}$$

As $0 \le k \le n-2$, inequality 4 is never true and this proves that on most machines the critical path of the LU algorithm follows the description of case 2.

The computational complexity of the algorithm is then computed following the critical path of case 2, taking into account the corresponding idle times (note that we do not find the same total time of ^{PBKP92} since we take into account all the idle times introduced by the pipeline strategy). The T^r_C is 2 times the T^c_C because the pivot column needs to be received and then transmitted to the next processor. The idle time T^r_{idle} is expressed as a function of T^r_S and T^r_C.

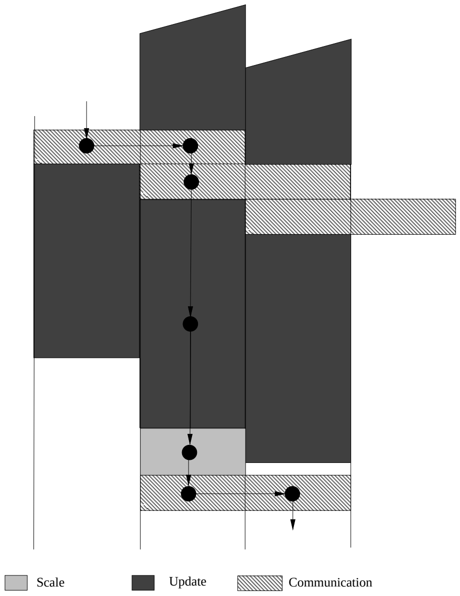

Figure 7: Zoom on the the critical path in case 1 (no idle time).

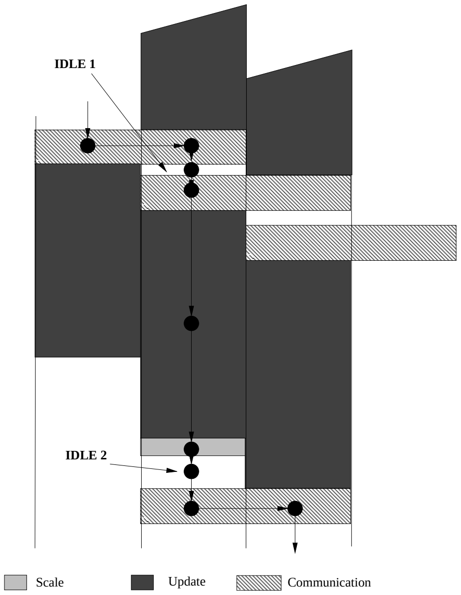

Figure 8: Zoom on the idle times in the critical path of case 2.

Proposition 2 *The parallel execution time of the LU algorithm on the ring with no overlap of communication and computation is given by Equations 5 and 6.*

$$T_{LU}^{ring} = T_S^r + T_C^r + T_U^r + T_{idle}^r \qquad (5)$$

where

$$\begin{aligned}
T_S^r &= T_S^c \\
T_C^r &= \sum_{k=0}^{n-2} 2(\beta_C + (n-k)\tau_C) = 2T_C^c \\
T_U^r &= T_U^c \\
T_{idle}^r &= \sum_{k=1}^{n-3} \left((C^{k-1} - S^k)^+ + C^{k-1} - C^k\right) \\
&= (n-3)\left(\beta_C - \beta_S + \frac{(n+2)}{2}\tau_C - \frac{n}{2}\tau_S\right)^+ + (n-3)\tau_C
\end{aligned}$$

where $(x)^+$ is the function which return x if $x > 0$ and 0 otherwise. Thus, we obtain in case 2 ($T_{S(k)} \leq T_{C(k-2)}$) :

$$\begin{aligned}
T_{LU}^{ring} &= T_{LU}^{complete} + (2n-4)\beta_C + (n^2 - n - 3)\tau_C - (n-3)\left(\beta_S - \frac{n}{2}\tau_S\right) \qquad (6) \\
&= T_{LU}^{ring}(case1) + (n-3)\beta_C + (n^2/2 + 3n/2 - 3)\tau_C - (n-3)\left(\beta_S - \frac{n}{2}\tau_S\right)
\end{aligned}$$

6. Asynchronous Communications

In this section, we present a strategy for improving the execution times, based on the reduction of the communication time by its partial overlap with computation. This overlap is realized inside a processor; it does not correspond to the overlap between processors steps described in [PBKP92], which is the effect of pipeline algorithms. We will see that the execution time of the complete network can be obtained with the pipeline algorithm on a ring. Hence, on any topology (including a ring), the execution time will remain the same.

6.1. Complete Network

Figure 9 shows that the critical path on the complete network is not changed while using asynchronous communications. Although a slight improvement occurs at the beginning of the task $U(k,k)$ on the pivot processor, the execution time of the critical path is not affected. This is because the tasks C must wait for the receive to complete before the execution of the U tasks.

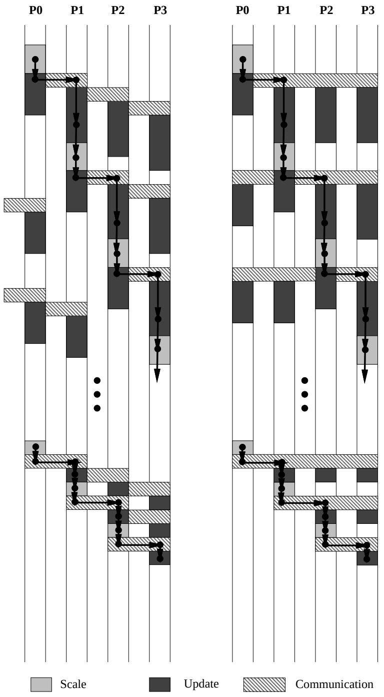

Figure 9: Time diagrams of nonblocking algorithms (ring on the left, complete network on the right).

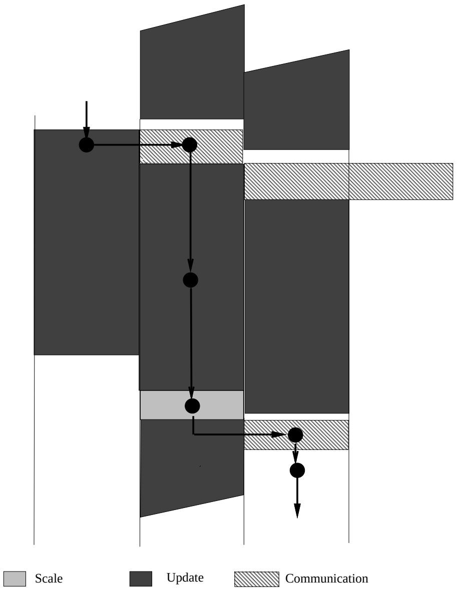

Figure 10: Zoom on the critical path for the ring topology (no idle time as in the complete network case).

Proposition 3 *The parallel execution time of the LU algorithm on the complete network with overlap of communication and computation (asynchronous* `sends`*) (overlap of communication and computation) is given by Equation 7.*

$$T^{complete}_{LU,async} = T^c_S + T^c_C + T^c_U, \tag{7}$$

where T^c_S, T^c_C, and T^c_U are the same as in the synchronous section.

6.2. Ring Topology

When asynchronous communications are available, the processor holding the next pivot column is not delayed by the sending of the current pivot column. Thus, it starts its U tasks as soon as possible.

As the data are distributed equally (see Section 4), the tasks U are nearly of the same duration; thus, the ring communications cannot perturb the critical path execution time (see Figure 10).

We follow the critical path of Figure 9, to determine the total duration of the algorithm execution. It leads to the following results[j].

Proposition 4 *The parallel execution time of the LU algorithm on the ring with overlap of communication and computation (asynchronous* SENDS*) (overlap of communication and computation) is given by Equation 8.*

$$T^{ring}_{LU,async} = T^r_S + T^{a,r}_C + T^r_U = T^{complete}_{LU,async} \tag{8}$$

where T^r_S and T^r_U are the same as in the synchronous case and $T^{a,r}_C$ is half of the synchronous one (i.e. the same as the complete network one).

It is remarkable that, from an analytical point of view, as soon as the targeted architecture is able to run asynchronous send communications, the classical pipeline *LU* decomposition on a ring has the best execution time on any topology.

7. Experiments

The experiments were performed on the Intel machines. On the iPSC/860, we used 2 to 64 processors with a ring embedded in the hypercube. On the Paragon, we used up to 64 nodes and a virtual ring. Obviously, we were not able to conduct experiments on a complete network of processors.

To have real synchronous sends, we used the `sendrecv` function (from the vendor library), which makes possible the acknowledgment of the reception of the message.

We plotted on Figures 11 and 12 the experiments and the computational complexities curves computed with the parameter values of section . The computational complexity curves are a little bit optimistic for the asynchronous case (we think this is due to the overhead of the program management and the setup of the asynchronous calls) and a little bit pessimistic for the synchronous case (probably because of an overlap between communication and idle time resulting of the successive communication calls[k]. The experimental curves remain between the 2 computed curves, except when OS delays occurs on the Paragon[l].

[j] We assume that the $T^r_C << T^r_U$, which is true since small vector sizes n because $T^r_C = O(n^2)$ while $T^r_U = O(n^3)$.

[k] An example of such an effect for the spanning binomial tree broadcast in hypercube is studied in [DFT93].

[l] We were using version 1.1 of the operating system which was doing some perturbing work for some memory size requirement.

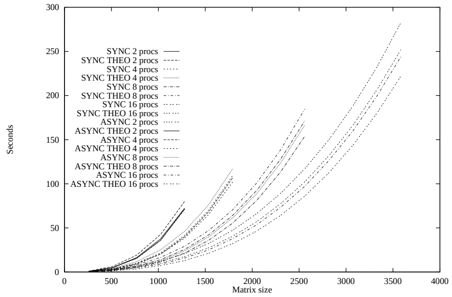

Figure 11: Intel iPSC/860 execution timings and complexity curves as a function of the matrix size.

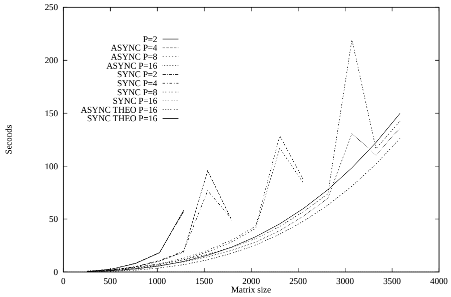

Figure 12: Intel Paragon execution timings and complexity curves as a function of the matrix size.

As shown by the complexity analysis, the asynchronous method perform always better than the synchronous one, and the difference increases slightly with the matrix size.

We show that such analysis can predict the behavior of the LU algorithm for any number of processors, by giving a timing interval in which the experimental data will fit.

With the assembly-coded Level 1 BLAS routines, the total performance is in the range of the BLAS kernels. The maximum performance obtained while decomposing a 2K matrix with 8 processors is 64 Mflops on the iPSC and 136 Mflops on the Paragon (representing, respectively, 8 and 17 Mflops per processor[m]). For larger matrices and bigger machines, our experimental results indicate that our method is efficient: we achieved 0.42 Gflops on the iPSC860 and 0.47 Gflops on the Paragon for a 7K matrix on 64 processors, compared with 1.34 Gflops of the ScaLAPACK implementation of the LU decomposition in the benchmark of the iPSC860 [DGW92].

8. Conclusion and Future Work

We have described an approach for analyzing the parallel LU decomposition with a scattered-column data distribution in both synchronous and asynchronous communication protocols cases. We have presented a computational complexity analysis for each algorithm. In addition, we have described experiments to determine the values of the target machines parameters. We thus have a tool that can be used for predicting the execution time.

We have showed that in the case of synchronous communications, idle times have to be taken into account in the ring topology complexity analysis and that they are of the same order ($O(n^2)$) as the communication time. On the other hand, in the case of asynchronous communications, we have showed that no idle time is introduced by the topology restriction to the ring.

[m] Note that because we are not using a blocked version of the LU decomposition, we not could use the assembly-coded Level 3 BLAS routines and reach the performance of [CDW92].

Our analytical analysis has been corroborated by efficient experiments on the Intel iPSC/860 and Paragon machines.

Our focus has been on complete networks and on ring topologies. The analysis can be extended, however, to grid topologies with scattered block data decomposition.

References

[BDL91] R.H. Bisseling, L. Daniel, and J.C. Loyens. Towards Peak Parallel LINPACK Performance on 400 Transputers. *Supercomputer*, VIII-5(45):20–27, September 1991.

[Cap87] P.R. Capello. Gaussian Elimination on a Hypercube Automaton. *Journal of Parallel and Distributed Computing*, 4:288–308, 1987.

[CDW92] J. Choi, J.J. Dongarra, and D.W. Walker. The Design of Scalable Software Libraries for Distributed Memory Concurrent Computers. In J.J. Dongarra and B. Tourancheau, editors, *Environments and Tools For Parallel Scientific Computing.* Elsevier, 1992.

[CG87] E. Chu and A. George. Gaussian Elimination with Partial Pivoting and Load Balancing on a Multiprocessor. *Parallel Computing*, 5:65–74, 1987.

[CRT89] M. Cosnard, Y. Robert, and B. Tourancheau. Evaluating speedups on distributed memory architectures. *Parallel Computing*, 10:247–253, 1989.

[CTV87] M. Cosnard, B. Tourancheau, and G. Villard. Gaussian elimination on message passing architecture. In *International Conference on Supercomputing*, Patras, Grece, June 1987. Springer-Verlag.

[DCDH90] J.J. Dongarra, J. Du Croz, I. Duff, and S. Hammarling. A Set of Level 3 Basic Linear Algebra Subprograms. *ACM Transaction on Mathematical Software*, 16(1):1–17, 1990.

[DFT93] F. Desprez, P. Fraigniaud, and B. Tourancheau. Successive Broadcast on Hypercube. Technical Report CS-93-210, The University of Tennessee - Knoxville USA, 1993.

[DGW92] J.J. Dongarra, R. Van De Geijn, and D.W. Walker. A Look at Dense Linear Algebra Libraries. Technical Report ORNL/TM-12126, Oak Ridge National Laboratory, July 1992.

[DO90] J.J. Dongarra and S. Ostrouchov. LAPACK Working Note: LAPACK Block Factorization Algorithms on the Intel iPSC/860. Technical Report 24, Department of Computer Science - University of Tennessee, 1990.

[Dun90] T.H. Dunigan. Performance of the Intel iPSC/860 Hypercube. Technical Report TM-11491, Oak Ridge National Laboratory, September 1990.

[EK93] R. Esser and R; Knetcht. Intel Paragon XP/S - Architecture and Software Environment. Technical Report KFA-ZAM-IB-9305, Zentralinstitut fur Angewandte Mathematik - Forschungszentrum Julich, April 1993.

[GL89] G.H. Golub and C.F. Van Loan. *Matrix Computation.* The John Hopkins University Press, 1989. Second edition.

[Int93] Intel Corporation - SSD, Beaverton, OR. *PARAGON OSF/1 - User's guide*, April

1993.

[LC89] G. Li and T.F. Coleman. A New Method for Solving Triangular Systems on Distributed Memory Message Passing Multiprocessors. *SIAM Journal on Science and Statistical Computing*, 10:382–396, 1989.

[LKK83] R.E. Lord, J.S. Kowalik, and S.P. Kumar. Solving Linear Algebraic Equations on a MIMD Computer. *Journal of the ACM*, 30(1):103–117, January 1983.

[PBKP92] B.V. Purushotham, A. Basu, P.S. Kumar, and L.M. Patnaik. Performance Estimation of LU Factorisation on Message Passing Multiprocessors. *Parallel Processing Letters*, 2(1):51–60, 1992.

[Rob90] Y. Robert. *The Impact of Vector and Parallel Architectures on the Gaussian Elimination Algorithm.* Manchester University Press - Manchester - UK, 1990.

[RT88] Y. Robert and B. Tourancheau. Block gaussian elimination on a hypercube vector multiprocessor. *Revista de Mathematicas Aplicadas*, 10:77–89, 1988. Universidad de Chile.

[RTV89] Y. Robert, B. Tourancheau, and G. Villard. Data allocation strategies for the gauss and jordan algorithms on a ring of processors. *Information Processing Letters*, 31:21–29, 1989.

[Saa86a] Y. Saad. Communication Complexity of the Gaussian Elimination Algorithm on Multiprocessors. *Linear Algebra and Applications*, 77:315–340, 1986.

[Saa86b] Y. Saad. Gaussian Elimination on Hypercubes. In M. Cosnard, Y. Robert, P. Quinton, and M. Tchuente, editors, *Parallel Algorithms and Architectures.* North-Holland, 1986.

[SS89] Y. Saad and M.H. Schultz. Data Communication in Parallel Architectures. *Journal of Parallel and Distributed Computing*, 6:115–135, 1989.