Using Postordering and Static Symbolic Factorization for Parallel Sparse LU

Michel Cosnard
LORIA - INRIA Lorraine
Nancy, France
Michel.Cosnard@loria.fr

Laura Grigori
LORIA - Univ. Henri Poincaré
Nancy, France
Laura.Grigori@loria.fr

Abstract

In this paper we present several improvements of widely used parallel LU factorization methods on sparse matrices. First we introduce the LU elimination forest and then we characterize the L, U factors in terms of their corresponding LU elimination forest. This characterization can be used as a compact storage scheme of the matrix as well as of the task dependence graph. To improve the use of BLAS in the numerical factorization, we perform a postorder traversal of the LU elimination forest, thus obtaining larger supernodes. To expose more task parallelism for a sparse matrix, we build a more accurate task dependence graph that includes only the least necessary dependences.

Experiments compared favorably our methods against methods implemented in the S environment on the SGI's Origin2000 multiprocessor.*

1. Introduction and background

In general, solving an unsymmetric linear system of equations $Ax = b$ by a direct method requires an initial transforming of a matrix in its LU form, which is the most time-consuming step before solving two triangular systems. In the case of a sparse matrix, the parallel LU factorization is particularly important since sparsity offers several opportunities to speed up a normal factorization: storage requirements are smaller, computation requirements are reduced and more parallelism can be exploited.

Two approaches have been developed for the LU factorization of a sparse matrix: submatrix-based methods [1] and column-based (more recently supernodal-based) methods [2]. Our LU factorization research was based on two implementations of supernodal methods. The SuperLU [2] package is designed for an existing threading model on the shared memory machines. On the other hand, the S* [5] and S+ [10] implementations are built for distributed memory machines, giving a better scalability.

In order to present the contributions of this paper we first present the sequence of steps for a sparse LU factorization, the definition of elimination tree and we particularize these definitions for SuperLU, S* and S+. Factoring an unsymmetric sparse matrix in its LU form is a sequence of chained steps. Even if the exact separation of steps and the operations done in each step are treated differently by different authors, there is a general agreement on the following scheme: (1) compute fill-reducing ordering, (2) perform symbolic factorization, (3) compute numerical factorization and (4) solve triangular systems of equations.

Notations: the matrix obtained after the static symbolic factorization was applied to A is noted $\bar{A} = \bar{L} + \bar{U} - I$. The element a_{ij} is the element on row i and column j of A. The number $|A_{i*}|$ equals the number of elements in the ith row of A. An important data structure is the *elimination tree* (etree for short) [9]; first introduced for the sparse Cholesky factorization, it has proven to be very useful in the symmetric case and it has later been adapted for the unsymmetric case. The SuperLU package employs the *column elimination tree*, which is the etree of $A^T A$. In the case of S* and S+, the elimination tree (forest) is based on the matrix $\bar{A}$, and is given by the following definition:

Definition 1 [10] An LU elimination forest (LU eforest for short) for an $n \times n$ matrix $\bar{A}$ has n nodes, and k is the parent of j ($parent(j) = k$) in the eforest if and only if $k = \min\{r > j | \bar{u}_{jr} \neq 0\}$ and $|\bar{L}_{*j}| > 1$.

The first step, compute fill-reducing ordering, is completely separated from the factorization. We shall not discuss this step, just mention that we use the minimum degree algorithm on $A^T A$.

The symbolic factorization step gives the structures of L and U, allowing the next step to know the elements on which to compute numerical factorization and evaluate memory requirements. A variation is the *dynamic symbolic factorization* which consists to interleave symbolic factorization with numerical factorization steps (SuperLU [2]). Note a larger overhead, since symbolic steps take $20 - 45\%$ from the total factorization time.

A faster approach exists for symbolic factorization: generate a larger data structure which contains the nonzeros in L and U for all possible row permutations which can later appear in the numerical factorization, due to pivoting (pivoting may be required if A unsymmetric; this structure can be determined by an efficient static symbolic factorization scheme [6]. This way, the LU factorization is computed on $\bar{A}$ instead of A. Even if some operations will involve zero elements (S* [5], S+ [10]), recent developments show that some of the zero blocks can be eliminated from the computation (LazyS+.)

At the numerical factorization step, an important goal is to group columns with the same structure, obtaining a structure similar to a dense matrix (from the storage and computation points of view). This concept was introduced in the SuperLU package and is called "unsymmetric supernode". The benefits of supernodes are that most of the numerical factorization can be done using the dense BLAS-2, improving cache hierarchy usage and reducing communication latencies. In SuperLU, supernodes are enlarged by permuting the matrix according to a postorder on its column elimination tree. This is needed since the sizes of supernodes actually occurring in practice are rather small (2 or 3 columns.) In the S+ and S* approaches [10] the authors show that, by using L/U supernode partitioning after the static symbolic factorization, it is possible to identify dense structures in both L and U factors, thus maximizing the use of BLAS-3 subroutines. Overall, we consider that the S* and S+ approaches for static symbolic factorization can lead to competitive results and we tried to optimize them by applying a postorder step that maximizes the supernode sizes.

Scheduling algorithms are used to partition and distribute blocks to be factored on different processors. S* uses a *static scheduling algorithm*, since data and control flows are known in advance. The task dependence graph is built using the structure of $\bar{A}$, and the run-time system RAPID assigns tasks to processors in an optimal way. We improve this technique by building the task dependence graph with the least number of necessary dependences.

The outline of the article is: section 2 presents the utilization of the LU eforest to characterize the L, U factors. In section 3, we perform a postorder traversal on this elimination forest, and obtain a block upper triangular form. In section 4, we obtain the task dependence graph for the sparse LU factorization. In section 5 we present the experimental results, and finally, section 6 concludes the paper.

2. Using LU eforest for characterization of $\bar{L}$, $\bar{U}$ factors

In this section we show that the row and column structures of the $\bar{L}$ and $\bar{U}$ factors of an unsymmetric sparse matrix $\bar{A}$ can be characterized in terms of $\bar{A}$'s LU eforest. This characterization will help prove that postordering does not change the static symbolic factorization. As an aside, this characterization leads also to the definition of a compact storage scheme for an unsymmetric sparse matrix.

Notations: as presented in [9], the subtree $T[x]$ is the subtree of the LU eforest of $\bar{A}$ (noted $T(\bar{A})$) rooted at the node x which includes all descendants of x in the tree T. The structure $T_r[i]$ of the ith row of the $\bar{L}$ factor is $T_r[i] = \{j \mid \bar{l}_{ij} \neq 0\}$ and is called the ith row subtree of $\bar{L}$. The column subtree of the $\bar{U}$ factor is defined as $T_c[i] = \{j \mid \bar{u}_{ji} \neq 0\}$. We suppose that the matrix A is nonsingular. Then, we can consider that A has a zero-free diagonal (it's always possible to permute A's rows using a transversal thus transforming A's diagonal to a zero-free diagonal [3]). A characterization of the rows of $\bar{L}$ using the elimination tree for unsymmetric matrices was proposed in [7]. In that paper, every row $\bar{L}_{i*}$ is represented by a branch of the etree, branch that belongs to $T[i]$.

A characterization of the columns of $\bar{U}$ is not known in the surveyed literature. The next theorems will help define the structure of every column $\bar{U}_{*j}$.

Theorem 1 *If $\bar{u}_{ij} \neq 0$, then $\bar{u}_{kj} \neq 0$ for every ancestor k of i such that $k < j$.*

PROOF If k is an ancestor of i such that $k < j$ then the following holds: $\bar{U}_{i*} - \{i, i+1, \ldots, k-1\} \subseteq \bar{U}_{k*}$ ([10]). As $\bar{u}_{ij} \neq 0$ and $k < j$, then $j \in \bar{U}_{i*} - \{i, i+1, \ldots, k-1\}$, and thus $j \in \bar{U}_{k*}$, and hence $\bar{u}_{kj} \neq 0$.

Theorem 2 *If $\bar{u}_{ij} \neq 0$, then $i \in T[j]$ or $i \in T[k]$, where $parent[k] = 0$ and $k < j$.*

PROOF We will prove by contradiction. Suppose that $i \in T[k']$, where $parent[k'] = 0$ and $k' > i, j$. Let m_1, m_2 being ancestors of i in $T[k']$ st $parent(m_1) = m_2$, $m_1 < j$ and $m_2 > j$. As $\bar{u}_{ij} \neq 0$ and $m_1 < j$, theorem 1 holds, and so $\bar{u}_{m_1 j} \neq 0$. This is a contradiction, because by the definition of the LU eforest, $\bar{u}_{m_1 m_2}$ is the first element on the m_1 row of $\bar{U}$.

We can conclude that the jth column subtree of $\bar{U}$ is defined by the following subtrees:

- $T_{c_j}[j] \subseteq T[j]$ which is a subtree rooted at j.
- $T_{c_k}[j] \subseteq T[k]$ which is a subtree rooted at k, where $parent[k] = 0$ and $k < j$.

Here is an example of LU eforest characterization of both $\bar{L}$ and $\bar{U}$ factors. Consider the matrix in figure 1(a). Then the corresponding extended LU eforest is represented in figure 1(b) (branches and digits close to the nodes). Italics at the left of each node denote the first nonzero in the row subtree corresponding to $\bar{L}$, while italics at the right of each node represent the leaf nodes of column subtrees corresponding to $\bar{U}$.

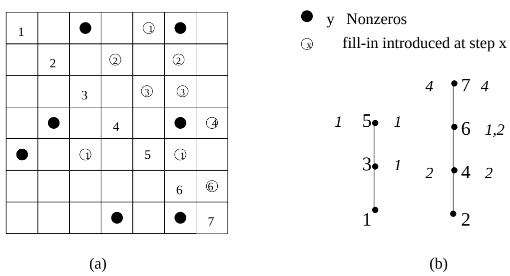


Figure 1. Matrix example $\bar{A}$ (a) and its extended LU eforest (b).

3. Column ordering and supernode partitioning

In [2], after computing the column elimination tree, the matrix columns are permuted according to a postorder on this etree. This ordering is equivalent to the original ordering in terms of fills and computation and is referred to as a topological ordering on the column etree, its aim being to bring together unsymmetric supernodes. Using the column elimination tree has the disadvantage that it substantially overestimates the structures of L and U, and implicitly the supernodes which will actually occur in practice.

For using the postordering with the LU eforest, we have to prove that the former does not change the static symbolic factorization. We show that this postorder does more than just bringing together unsymmetric supernodes: it also offers a decomposition in a block upper triangular form. We will renumber the columns such that any node is numbered before its parent in the LU eforest. This generates a column ordering P for $\bar{A}$ and a new elimination forest. Since the static symbolic factorization will not change, the structure of the extended LU eforest will be the same, and only the nodes labels will change. The reordering is applied to both rows and columns of $\bar{A}$ such as to preserve the nonzero diagonal and to obtain the decomposition in a block upper triangular form. We will define a matrix obtained after the postordering step as one that satisfies the following: let $x_1 < \ldots < x_n$ be nodes in the LU eforest of $\bar{A}$ st $parent(x_1) = \ldots = parent(x_n) = x$. Then $\forall m, i$ st $m \in T[x_i], m > x_{i-1}$.

Next, we give an algorithm which, from a matrix $\bar{A}$ and its LU eforest, computes a matrix satisfying the above property. This function is called with the roots of the etrees in an ascending order and the number of the etrees in the LU eforest as parameters.

void $postorder(R_1, \ldots, R_n, n)$
$R_0 = 0$;
for $i = n$ downto 1 **do**
  loop
    if $\exists x \in T[R_i]$ st $x < R_{i-1}$ **then**
      let x be the biggest number with this property;
      interchange rows and columns $(x, x+1)$;
    else
      break out of the loop;
    end if
  end loop
  $n' =$ number of sons of R_i;
  if $n' \neq 0$ **then**
    let $R'_1 < \ldots < R'_n$ be st $parent(R'_1) = \ldots = parent(R'_{n'}) = R_i$; $postorder(R'_1, \ldots, R'_n, n')$;
  end if
end for

Theorem 3 *Let $\bar{A}$ be a square matrix obtained after the static symbolic factorization. If the matrix is permuted using the above algorithm, then the static symbolic factorization will not change.*

PROOF First we prove that the candidate pivot rows at each step of the symbolic factorization are not changed (figure 2(a)). The only step which can change them is the interchanging of rows/columns $x, x+1$ (interchange $(x, x+1)$). As x is the biggest number in $T[R_i]$ which fills the condition $x < R_{i-1}$, then we are sure that $x+1$ does not belong to $T[R_i]$ and thus, by using the characterization of $\bar{L}$ rows in section 2, we can say that $\bar{L}_{x,1:x} \cap \bar{L}_{(x+1),1:x} = \emptyset$.

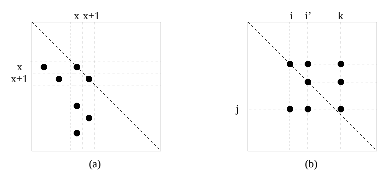


Figure 2. Illustration for Th 3 (a) and Th 4 (b).

Finally, we have to show that $\bar{l}_{x,x+1} = 0$ before interchanging x with $x+1$. As $parent(x) \neq x+1$, the only situation when $\bar{l}_{x,x+1} \neq 0$ can appear is when $|\bar{L}_{*x}| = 0$ and so x is the root of a tree. In all cases, as the trees are ordered in a descending order of their roots, x cannot be the root of the tree, and so $|\bar{L}_{*x}| \neq 0$. We conclude that there are no candidate pivot rows introduced.

This algorithm only helped to prove that the postorder will not change the static symbolic factorization. For the ease of implementation, we preferred to code the postorder depth-first search.

An example is given in the figure 3 in which the extended LU eforest is obtained from the one in figure 1 by relabeling the nodes using a postordering on the LU eforest. The permuted matrix P^TAP is block upper triangular. This can be explained by using the characterization of $\bar{L}, \bar{U}$ factors in section 2. In the following we will focus only on the factorization of the diagonal blocks.

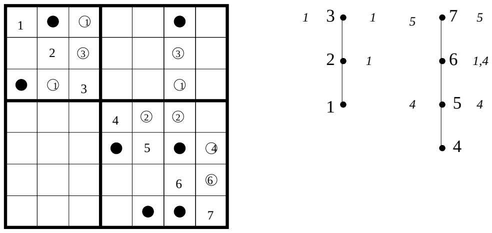


Figure 3. The decomposition in block upper triangular form corresponding to $\bar{A}$ in figure 1.

Given an unsymmetric matrix $\bar{A}$ after static symbolic factorization and postordering were applied, the next step is to identify supernodes in order to use dense computations. We do this by using the L/U supernode partitioning method described in [10]: the columns are partitioned using the definition of a supernode. After that, the same partitioning is applied to the rows of the matrix to further break each supernode into submatrices. As the average size of a supernode is very small, amalgamation is applied to further increase the supernode size.

4. Task dependence graphs

We suppose that after applying the supernode partitioning and amalgamation, we obtain $N \times N$ submatrix blocks. We will denote by $\bar{B}_{kj}$ a submatrix of $\bar{A}$ at row block index k and column block index j. We use a 1D block mapping scheme: an entire column block k is assigned to one processor. For each column block we identify two types of tasks [5] : *Factor(k)* and *Update(k,j)*. Task *Factor(k)* (*F(k)* for short) factorizes the column k, including finding the pivoting sequence associated with that column. Task *Update(k,j)* (*U(k,j)* for short) exists for $k < j$ and $\bar{B}_{kj} \neq 0$ and consists in updating column j by column k. An outline of the sparse LU factorization algorithm with partial pivoting is following:

for $k = 1$ to N **do**
  Perform task $Factor(k)$;
  for $j = k+1$ to N with $\bar{B}(k,j) \neq 0$ **do**
    Perform task $Update(k,j)$;
  end for
end for

The S* approach of building the task dependence graph is based on the factored matrix structure: starting from it, the tasks *F(k)* and *U(k,j)* are deduced and the dependences between *U(k,j)* tasks are given by the ascending order of the indices k.

We will show that using a method based on the factored matrix structure and on the corresponding LU etree can lead to a better task dependence graph. This graph exposes more parallelism by eliminating *false dependences* and replacing them with the least necessary dependences. Next we present a method of building this graph starting from the structure of $\bar{B}$ and from its LU eforest. The following theorem gives the dependences between two columns i, i' which are on a same path of the LU etree and which update the same third column.

Theorem 4 *Let $i,\ i'$ and k be nodes of $T(\bar{B})$ such that $parent(i) = i'$ and the tasks $U(i,k),\ U(i',k)$ exist. Then the task $U(i,k)$ has to be completed before the task $U(i',k)$.*

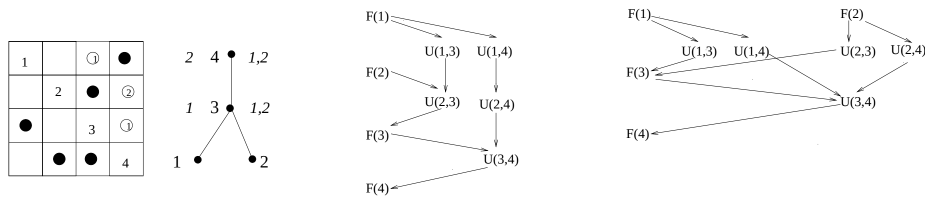
PROOF Knowing that $parent(i) = i'$ then i' is the first nonzero in the row $\bar{U}_{i*}$. As illustrated in figure 2(b), knowing that $|\bar{L}_{*i}| > 1$ and $\bar{B}_{ii'}$ is the first nonzero in the row $\bar{U}_{i*}$, then there is a $j \geq i'$ such that $\bar{B}_{ji} \neq 0$. By using the principle of the static symbolic factorization, we can conclude that $\bar{B}_{ji'} \neq 0$ and $\bar{B}_{jk} \neq 0$. It can be seen that $\bar{B}_{ji'}$ is a candidate pivot for the factorization $F(i')$, and thus $\bar{B}_{jk}$ has to be updated by $U(i,k)$ before $U(i',k)$ is computed.

Columns in independent subtrees of the elimination tree can be computed without referring to any common elements, because the source columns in the updates have completely disjoint row indices [8].

Let $i,\ i'$ be nodes of $T(\bar{B})$, neither of which is an ancestor of the other, and k another node such that the tasks $U(i,k),\ U(i',k)$ exist. Then there is no dependence between the two tasks, as they are computed without referring to any common element (as above.) From these results, the structure of a sparse LU task dependence graph can be defined as follows :

1. There is a task *F(i)* for each $1 \leq i \leq N$.
2. There is a task *U(i,k)* for each $\bar{B}_{ik} \neq 0$ and $1 \leq i < k \leq n$.
3. There is a dependence edge from *F(i)* to task *U(i,k)*, for any k such that $\bar{B}_{ik} \neq 0$.
4. There is a dependence edge from *U(i,k)* to *U(i',k)* if i' is the parent of i in $T(\bar{B})$.
5. There is a dependence edge from *U(i,k)* to *F(k)* if k is the parent of i in $T(\bar{B})$.

We illustrate the above definition by computing the sparse LU task dependence graph for the matrix in the figure 4(a). The resulting LU etree is in figure 4(a). From this etree we obtain the sparse LU DAG shown in figure 4(c). In figure 4(b) we show the dependence graph for the same matrix which was used in the S* package. In order to schedule the LU task dependence graph we used the RAPID runtime system [4]. It creates a schedule in two steps: first, it analyzes data accesses for each task, thus obtaining a task dependence graph and second it efficiently distributes these tasks on a distributed memory machine. RAPID uses an inspector–executor approach and optimizes the interleaving of communications and computations.



(a) Matrix $\bar{B}$ and its extended LU eforest (b) S* task dependence graph (c) New task dependence graph

Figure 4. Matrix example $\bar{B}$, its extended LU eforest and its dependence graphs.

Table 1. Benchmark matrices.

Matrix	Name	Order	Nonzeros $\|A\|$	$\|\bar{A}\|/\|A\|$
1	sherman3	5005	20033	31.20
2	sherman5	3312	20793	15.70
3	lnsp3937	3937	25407	27.33
4	lns3937	3937	25407	27.92
5	orsreg1	2205	14133	41.44
6	saylr4	3564	22316	44.19
7	goodwin	7320	324784	10.80

Table 2. Time performance (in seconds) of our algorithm on 195MHz Origin 2000.

Mat	P=1	P=2	P=4	P=8
1	5.79	3.48	2.18	1.51
2	0.83	0.51	0.37	0.37
3	5.98	3.51	2.11	1.62
4	13.22	7.38	4.4	2.9
5	5.93	2.9	1.78	1.4
6	14.59	8.62	5.3	3.37
7	4.93	2.99	1.95	1.35

5. Experimental studies

In this section, we show experimental results obtained when applying the proposed factorization method on real-world matrices. We tried to use several matrices of small or medium sizes from a variety of application domains. These matrices[1] and their characteristics are presented in table 1.

From oil reservoir modeling matrices we used sherman3, sherman5, orsreg1 and saylr4. The matrices lnsp3937 and lns3937 are used in fluid flow modeling. The finite element matrix goodwin is used in a nonlinear solver also for a fluid mechanics problem. The third column in table 1 is the order of the matrix, and the fourth column contains the number of nonzeros $|A|$ before symbolic factorization. We also list the total number of factor entries obtained in $\bar{A}$ divided by $|A|$.

We choose to test our factorization method on a Silicon Graphics cache-coherent distributed shared memory architecture, the Origin 2000. The configuration contains 64 R10000 processors clocked at 195Mhz. The total physical memory size is 24Gbytes. The cache hierarchy has two-levels: L1 is a separated 32kbytes data and instruction on-chip cache while L2 is a 4Mbytes on-board unified cache. The interconnection network is a 4-ary hypercube, achieving a peak bandwidth between nodes of 140Mbytes/sec. We used the MIPSPro C compiler v7 with the second optimization level enabled (`-O2`) the scientific library SCSL (BLAS levels 1, 2, 3) and the SHMEM communication library. All floating point operations are double precision.

[1]Matrices were obtained from the Harwell-Boeing Collection and from the ftp site maintained by Tim Davis of the University of Florida, ftp.cise.ufl.edu/pub/faculty/davis/matrices.

Table 3. Difference between the supernode sizes without/with postordering.

Name	1	2	3	4	5	6
NoBlks	2110	1675	39	26	1	1
SN	3082	1857	1365	1357	630	1109
SNPO	2696	1815	1010	1012	406	705
1-SN/SNPO	.13	.02	.26	.25	.35	.36

Next, we report the overall performance of our code. Then, to better understand the impact of using postordering and of the new task scheduling graph, we separately measure the effectiveness of each of them.

Overall performance In table 2 we show the performance of our implementation for the numerical factorization step. The code scales well up to 8 processors and the speedups range from 2.3 to 4.6.

Effectiveness of postordering The methodology for evaluating the impact of postordering is the following: first, for the different matrices we measure the supernode sizes obtained after the L/U supernode partitioning and the amalgamation were applied. After that, we permute the rows and columns of the matrix according to a postorder on its elimination forest, before applying the supernode L/U par-

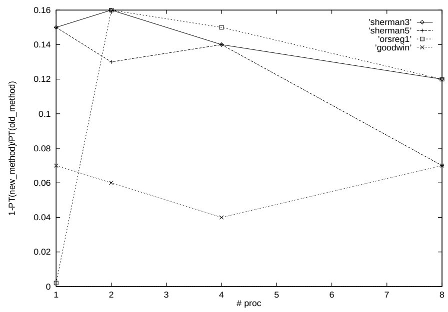


Figure 5. Performance improvement by using the new task dependence graph.

titioning and the amalgamation of the supernodes. Then we measure again the obtained supernode sizes. Results are shown in table 3, where SN and SNPO represent the number of supernodes obtained without and with postordering. It can be observed a decrease in the number of supernodes (an average of 24%). One exception is the sherman5 matrix, for which the size improvement ratio is not substantial. We can explain this behavior by the large sparsity and lack of structure which will make supernode identification difficult even when postordering is applied.

When measuring the number of blocks obtained (NoBlks in table 3), we notice a large number of blocks for the first four matrices. The size of the first blocks on the diagonal is 1 and only the last block has a significant size.

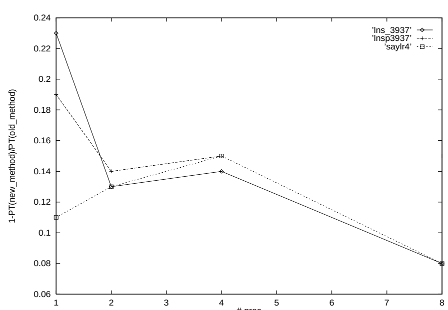


Figure 6. Performance improvement by using the new task dependence graph.

Effectiveness of the new task dependence graph The gain obtained by the introduction of the new task dependence graph is evaluated by comparing our code speed with the speed of a modified version which uses the task dependence graph defined in S* [5]. The performance improvement ratio of our approach is listed in figures 5 and 6. Experiments confirm our assumptions. Execution times obtained when using LU etrees to deduce the task graph are 4% to 23% faster than execution times obtained when not using etrees to build the task graph.

6. Conclusions and future work

In this paper we propose a number of techniques to improve existing methods of parallel sparse LU factorization. We evaluate the actual performances by applying those methods in a program implemented on a 64 processor SGI Origin2000 machine. These techniques enable the usage of the postordering and of the static symbolic factorization methods together. Also, we build a more accurate task dependence graph that includes only the least necessary dependences, thus exposing more task parallelism for a sparse matrix.

Future work consists to experiment our code on larger matrices, as well as to extend our methods for a 2D partitioning of the matrix. Another direction will be to use the automatic task scheduling techniques for dynamically building the task dependence graph at run time. Yet another direction will be to use the extended LU eforest for more effective task dependence representation.

References

[1] P. R. Amestoy, I. S. Duff, and J.-Y. L'Excellent. Mumps multifrontal massively parallel solver version 2.0. Technical Report TR/PA/98/02, CERFACS, Toulouse, France, 1998.

[2] J. W. Demmel, J. R. Gilbert, and X. S. Li. An Asynchronous Parallel Supernodal Algorithm for Sparse Gaussian Elimination. *SIAM J. Mat. Anal. Appl.*, 20(4):915–952, 1999.

[3] I. S. Duff. On algorithms for obtaining a maximum transversal. *ACM Trans. Math. Softw.*, 7:315–330, 1981.

[4] C. Fu. *Scheduling and Runtime Support for Irregular Computations*. PhD thesis, UCSB, 1997.

[5] C. Fu and T. Yang. Efficient Sparse LU Factorization with Partial Pivoting on Distributed Memory Architectures. Technical Report TR CS96-18, UCSB, 1996.

[6] A. George and E. Ng. Symbolic factorization for sparse gaussian elimination with partial pivoting. *SIAM J. Sci. Stat. Comput.*, 8(6):877–898, 1987.

[7] A. George and E. Ng. Parallel Sparse Gaussian Elimination with Partial Pivoting. *Annals of Operations Research*, 22:219–240, 1990.

[8] J. R. Gilbert. An efficient parallel sparse partial pivoting algorithm. Technical Report 88/45052-1, CMI, 1988.

[9] J. W. H. Liu. The role of elimination trees in sparse factorizations. *SIAM J. Matrix Anal. Appl.*, 11(1):134–172, 1990.

[10] K. Shen, X. Jiao, and T. Yang. Elimination Forest Guided 2D Sparse LU Factorization. Technical Report TR CS98-08, UCSB, 1998.