Using Postordering and Static Symbolic Factorization for Parallel Sparse LU

Michel Cosnard, Laura Grigori

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

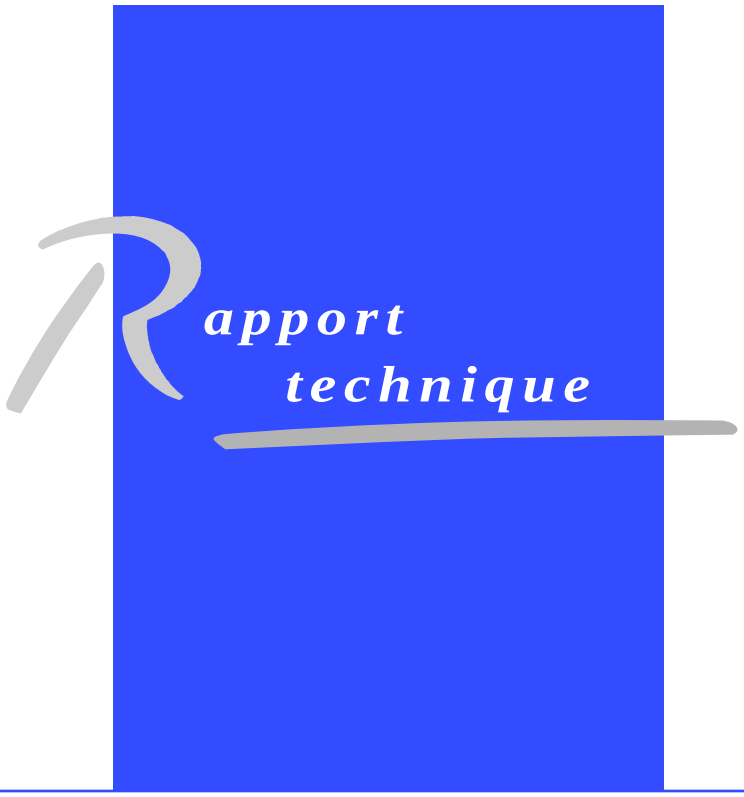
Using Postordering and Static Symbolic Factorization for Parallel Sparse LU

Michel Cosnard - Laura Grigori

N˚0237

November 3, 1999

THÈME 1



ISSN 0249-0803

Using Postordering and Static Symbolic Factorization for Parallel Sparse LU

Michel Cosnard - Laura Grigori

Thème 1 — Réseaux et systèmes
Projet RÉSÈDAS

Rapport technique n ° 0237 — November 3, 1999 — 13 pages

Abstract: In this report we present several improvements of widely used parallel LU factorization methods on sparse matrices. First 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 eforest 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 dependencies.

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

Key-words: sparse unsymmetric matrices, Gaussian elimination, partial pivoting, static symbolic factorization, elimination tree, postordering, SHMEM

(Résumé : tsvp)

Unité de recherche INRIA Lorraine
Technopôle de Nancy-Brabois, Campus scientifique,
615 rue de Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY (France)
Téléphone : 03 83 59 30 30 - International : +33 3 3 83 59 30 30
Télécopie : 03 83 27 83 19 - International : +33 3 83 27 83 19
Antenne de Metz, technopôle de Metz 2000, 4 rue Marconi, 55070 METZ

Factorisation LU parallèle sur des matrices creuses en utilisant des méthodes symboliques statiques et le parcours postfixé

Résumé : Le sujet de ce rapport est lié aux méthodes de factorisation LU en parallèle sur des matrices creuses. Tout d'abord, nous nous concentrons sur une caractérisation des facteurs L, U en fonction de leurs forêt d'élimination LU. Cette caractérisation peut être utilisée comme une méthode efficace de stockage de la matrice et de son graphe de dépendences de tâches. Ensuite, afin d'améliorer l'emploi des routines BLAS durant la factorisation numérique, nous appliquons un parcours postfixé sur la forêt d'élimination. Ceci nous permet d'augmenter la taille des supernoeuds. Pour mieux paralléliser les tâches, nous construisons un graphe de dépendances de tâches qui inclut les dépendances absolument nécessaires.

A partir de nos expériences, nos méthodes sont plus efficaces que celles implantées par S*, dans l'environnement SGI Origin2000 64 processeurs.

Mots-clé : matrices non-symmetriques creuses, élimination de Gauss, pivotage numérique, factorisation statique symbolique, arbre d'élimination, parcours postfixé, SHMEM

1 Introduction

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 [ADL98] and column-based (or, more recently, supernodal-based) methods [DEG+95]. Our LU factorization research was based on two recent implementations of supernodal methods. The SuperLU [DGL97] package is designed for an existing threading model on top shared memory machines. On the other hand, the S* [FY96] and S+ [SJY98] implementations are built for distributed memory machines, thus achieving a better scalability.

Our first contribution is a characterization of L, U factors in terms of their corresponding LU elimination forest. We will use this characterization for proving necessary theoretical results. Moreover, this characterization can be used as a compact storage scheme of the matrix as well as of the task dependence graph.

This report's second contribution is to combine advantages of SuperLU's postordering method and S*'s static symbolic factorization methods. One of the advantages of permuting a matrix according to a postorder on its elimination tree is that we obtain larger supernodes, and thus more numerical computation is done in dense matrix kernels. On the other hand, using the static symbolic factorization [GN87], as proposed by Yang and al. [FY96] is important to eliminate the structure variations of L, U factors during the numerical factorization. In doing so, there is no postordering step, due to authors' expectancy that row/column permutations could introduce incorrect results to the outcome of static symbolic factorization. However, we demonstrate that a postordering step does not change the static symbolic factorization. Thus, the advantages of static symbolic factorization and of the postordering are combined.

As a last contribution, we develop the task dependence graph, using the matrix obtained after the static symbolic factorization and the LU eforest. This graph is scheduled using the RAPID run-time system. We compare our results with the task dependence graph previously used in S*, and we obtain gains in speed performance that range from 4% to 23%.

The outline of the report is the following: in section 2, we briefly review notions related to parallel LU factorization as well as packages that implement them. Section 3 presents the utilisation of the LU elimination forest to characterize the L and U factors. In the next section (4), we perform a postorder traversal on this elimination forest, and obtain a block upper triangular form. In section 6, we obtain the task dependence graph for the sparse LU factorization. In section 7 we present the experimental results, and finally, section 8 concludes the report.

2 Background

In this section we present the general 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.

A widely used data structure is the *elimination tree* (etree for short) [Liu90]; it was first introduced for the sparse Cholesky factorization. Since the etree concept has proven to be very useful in the symmetric case, it has been adapted for the unsymmetric case.

As a general notation, $\bar{A} = \bar{L} + \bar{U} - I$ is the matrix obtained after the static symbolic factorization was applied to A. 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.

The SuperLU package employs the *column elimination tree*, which is the etree of A^TA. In the case of S* and S+, the elimination tree is based on the matrix A which was already statically simbolically factored (noted $\bar{A}$), given by the following definition:

Definition 1 [SJY98] 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 $|\bar{L}_{*k}| > 1$[1] (if j is a root of one etree in the LU eforest, then $parent(j) = 0$).

The first step, compute fill-reducing ordering, is completely separated from the factorization. In this report we shall not discuss this step and we use the minimum degree algorithm on $A^T A$ for this step.

The symbolic factorization step is necessary in order to compute the numeric factorization. This step gives structures of L and U thus allowing the next step to know exactly what are the elements on which to compute numerical factorization and what are the memory requirements. Calculating the L and U structures depends on the structure of A and on the row interchanges induced by pivoting (pivoting is required if A nonsymmetric). Most software packages interleave symbolic factorization steps with numerical factorization steps (SuperLU [DEG+95], [DGL97]), from where the term *dynamic symbolic factorization*. A trivial inconvenient of a dynamic symbolic factorization approach is that an overhead is introduced before each numerical step (approximately $20 - 45\%$ from the total factorization time).

A better time-efficient approach exists for symbolic factorization, which consists to generate a large data structure which contains the nonzeros in L and U for all possible row permutations which can later appear in the numerical factorization; it can be determined by an efficient static symbolic factorization scheme [GN87]. Some packages compute the LU factorization on $\bar{A}$ instead of A even if some operations will involve zero elements (S* [FY96], S+ [SJY98]). Clearly, this is the static symbolic factorization's most important disadvantage, but recent developments show that some zeros blocks can be eliminated from the computation (LazyS+).

At the numerical factorization step, an important goal is to group together columns with the same structure, thus obtaining a structure similar to a dense matrix (storage and computation viewpoints). 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' sizes are made larger by permuting the matrix according to a postorder on its column elimination tree. This operation is necessary since the sizes of supernodes actually occurring in practice are very reduced, only 2 or 3 columns.

In S+ and S* approaches, it is shown in [SJY98] that by using an L/U supernode partitioning technique after the static symbolic factorization, the authors identify dense structures in both L and U factors, thus maximising the use of BLAS-3 level subroutines. It is usual that highly optimized BLAS-3 subroutines outperform BLAS-2 subroutines, this being a main reason for achieving better performances than SuperLU [].

Overall, we consider that the S* and S+ approaches can lead to competitive results, especially if the level of the fill-in introduced by the static symbolic factorization is not very high, and we tried to optimize it 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. In S*, the fact that the flow of data and control is known in advance is exploited by using a *static scheduling algorithm*. For this, the task dependence graph is built by using the structure of $\bar{A}$, and the run-time system RAPID assign tasks to processors in a predefined, optimal way.

We will use this technique, and improve it by constructing the task dependence graph with the least number of necessary dependencies.

3 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 a unsymmetric sparse matrix $\bar{A}$ can be characterized in terms of its LU eforest. This characterization will help proving a previously mentioned goal: postordering does not change the static symbolic factorization. As an aside, this characterization leads also to the definition of a compact storage scheme for a unsymmetric sparse matrix.

Several necessary notations follow. As presented in [Liu90], the subtree $T[x]$ is the subtree of the LU etree 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 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 $\bar{U}$ factor is defined as $T_c[i] = \{j \mid \bar{u}_{ji} \neq 0\}$. We suppose that the matrix A is reducible and 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).

[1] The LU eforest should be distinguished from the eforest defined by George and Ng in [GN90] for the unsymmetric Gaussian factorization. The important difference is that in an LU forest the condition $|\bar{L}_{*k}| > 1$ is imposed.

A characterization of the rows of $\bar{L}$ using the elimination tree for unsymmetric matrices was proposed in [GN90]. If f_i is the column subscript of the first nonzero in row i of $\bar{L}$ (and hence of A) then the nonzeros in row i of $\bar{L}$ are given by nodes on the path f_i, $parent[f_i]$, ..., $i = parent^r[f_i]$. In other words, the row $\bar{L}_{i*}$ is in fact represented by a branch of the etree, branch that belongs to the subtree rooted at i.

However, a characterization of the columns of $\bar{U}$ is not known in the surveyed literature. We will characterize the structure of the columns of $\bar{U}$ using the LU eforest.

The next theorems will help define the structure of every column $\bar{U}_{*i}$.

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

PROOF If k is an ancestor of i such that $k < j$ then the following holds: $\bar{U}_{i*} - \{i, i+1, \ldots, k-1\} \subseteq \bar{U}_{k*}$ ([SJY98]). As $\bar{u}_{ij} \neq 0$ and $k \leq j$, 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 $k \leq j$.*

PROOF We will prove by contradiction. Suppose that $i \in T[k']$, where $parent[k'] = 0$ and $k' > i, j$. Let $m_1, m_2 \in T[k']$ be 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 $\bar{U}$.

Theorem 3 *The node i is a leaf in the column subtree $T_c[j]$ if and only if $\bar{u}_{ij} \neq 0$, and for every proper descendant k of i, $\bar{u}_{kj} = 0$.*

We can conclude that the ith column subtree of $\bar{U}$ (noted by $T_c[i]$) is defined by the following subtrees:

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

Remark that the etree characterization of $\bar{U}$ is different than the etree characaterization of $\bar{L}$. The leaves of the column subtrees corresponding to $\bar{U}$ are elements of $\bar{U}$ itself while the leaves of the row branches corresponding to $\bar{L}$ are elements of A.

As already mentioned, this characterization also gives a compact storage scheme of the L, U factors of a matrix, analogous with a compact storage scheme deduced for the Cholesky factors. In other words, we can represent the ith row of $\bar{L}$ by storing only the first nonzero of the ith row of A. Respectively, we can represent the ith column of $\bar{U}$ by storing only the leaf nodes of the subtrees $T_{c_i}[i]$, $T_{c_{k_1}}[i]$, ..., $T_{c_{k_p}}[i]$.

Here is an example of LU eforest characterization of both $\bar{L}$ and $\bar{U}$ factors. Consider the $\bar{A}$ 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 $\bar{U}$. From this elimination forest, called an extended LU eforest, we can find all information needed to obtain the structure of $\bar{A}$.

4 Column ordering

In packages as [DEG+95], [DGL97], [GN90], 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. The aim of this ordering is to bring together unsymmetric supernodes. The inconvenient of using the column elimination tree is that it often 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 elimination forest, we have to prove that this postordering does not change the static symbolic factorization. At the same time, we will show that this postorder on the LU elimination forest does more than just bringing together unsymmetric supernodes: it offers also 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 labelling of the nodes will

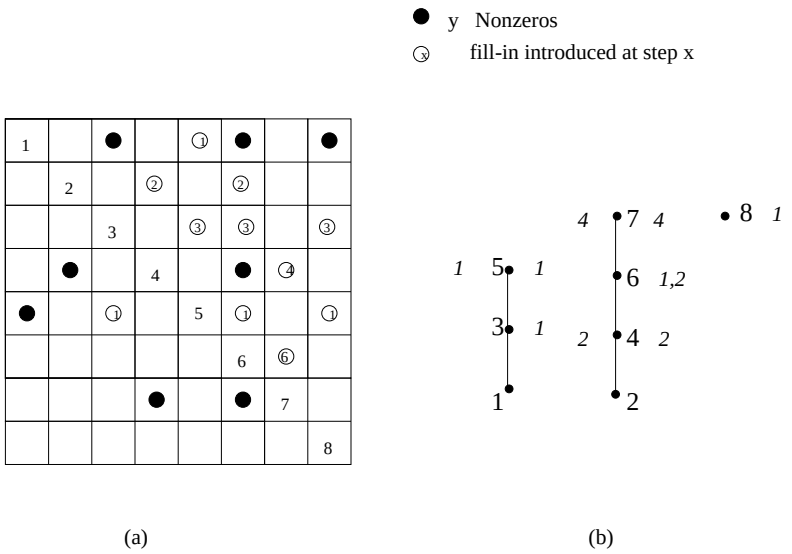


Figure 1: A matrix example $\bar{A}$ (a) and its extended LU eforest (b).

be changed. 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 a matrix satisfying the following property: 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.

```
1  void postorder (R_1, ..., R_n, n)
2  begin  for i = 1 to n
3      while (true)
4        if exist x ∈ T[R_i] st x > R_{i-1} then
5          let x be the biggest number with this property;
6          interchange (x, x+1);
7        else break;
8      end while
9      n = number of sons of R_i;
10     let R'_1 < ... < R'_n be st parent(R'_1) = ... = parent(R'_n) = R_i
11     postorder (R'_1, ..., R'_n, n);
12 endfor
13 end
```

Theorem 4 *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 To prove this, we have to prove that the candidate pivot rows at each step of the symbolic factorization do not change (figure 2). The only step which can change the candidate pivot rows 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 3, we can say that $\bar{L}_{x,1:x-1} \cap \bar{L}_{(x+1),1:x-1} = \emptyset$. By interchanging rows/columns $x, x+1$, this intersection is still empty.

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

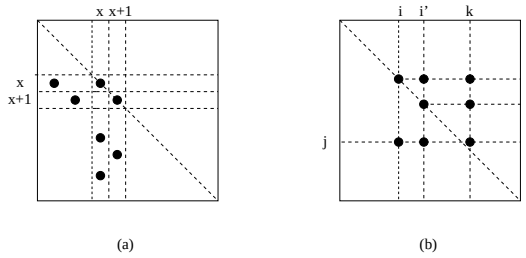


Figure 2: Illustration for the proof of Theorem 4 (a) and Theorem 5 (b)

An example is given in the figure 3 in which the extended LU eforest is obtained from the one in figure 1 by relabelling the nodes using a postordering on the LU eforest. The permuted matrix $P^T AP$ is block upper triangular.

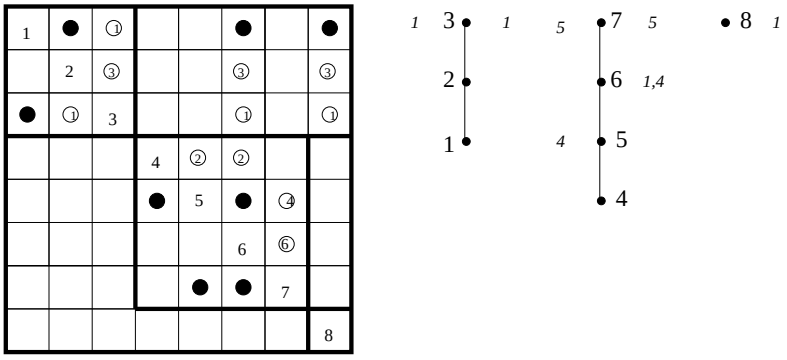


Figure 3: The decomposition in block upper triangular form corresponding to $\bar{A}$ in 1

In the common case, after having permuted the collumns in a postorder of an LU eforest containing N trees, a block upper triangular form will be obtained, as in the figure:

$$P^T \bar{A} P = \begin{bmatrix} \bar{A}_{11} & \bar{A}_{12} & \dots & \bar{A}_{1N} \\ & \bar{A}_{22} & \dots & \bar{A}_{2N} \\ & & \dots & \dots \\ & & & \bar{A}_{NN} \end{bmatrix}$$

This conclusion (obtaining a block upper triangular form) is explained by using the characterization of $\bar{L}, \bar{U}$ factors in 3. For example, considering the block $\bar{A}_{22}$ and a node i in the LU etree defining this block. The ith line is defined by a chain in the subtree rooted at $T[i]$, so all the $\bar{l}_{ij} \neq 0$ belong to this block, and the block $\bar{A}_{12}$ is null.

The advantage of using this upper triangular form is that the linear equations system can be solved by performing a block backsubstitution. In the following, we will focus only on the factorization of the diagonal blocks.

5 L/U Supernode partitioning

Given a nonsymmetric matrix $\bar{A}$ after static symbolic factorization and postordering was applied, the next step is to identify supernodes in order to use dense matrix computations.

We do this by using the L/U supernode partitioning method described in [SJY98]: first we partition the columns using the definition of a supernode. A second partitioning is applied to the rows of the matrix, thus obtaining submatrices for each supernode. A few nice properties hold for the resulting blocks: each diagonal matrix is dense, each nonzero submatrix in the L part contains only dense subrows and each nonzero submatrix in the U factor contains only dense subcolumns [FY96]. This way, the use of BLAS-3 subroutines is maximised

in S+, S* packages. As the average size of a supernode is very small, amalgamation is usually apply to further increase the supernode size.

6 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 [FY96] : *Factor(k)* and *Update(k,j)* for some $k < j$ and $\bar{B}_{kj} \neq 0$. Task *Factor(k)* (*F(k)* for short) factorize the column i, including finding the pivoting sequence associated with that column. Task *Update(k,j)* (*U(k,j)* for short) consists in updating column j by column k.

An outline of the partitioned sparse LU factorization algorithm with partial pivoting is following:

```
1  for k = 1 to N
2    Perform task Factor(k);
3    for j = k + 1 to N with B̄(k, j) ≠ 0
4      Perform task Update(k,j);
5    endfor
6  endfor
```

The S* approach of building the task dependence graph is based on the factored matrix structure: starting from this structure the tasks *F(k)* and *U(k,j)* are deduced. Also, simple rules based on the matrix structure give dependences between the tasks. There are implementation considerations that make this simple method very attractive.

However, we show that using a method based on the factored matrix structures and on the corresponding LU etree can lead to a better task dependence graph. This graph exposes more parallelism by eliminating *false dependencies* and replacing them with the least necessary dependencies.

Now we present a method of building this graph starting from the structure of $\bar{B}$ and from its LU eforest.

The first result we want to remind is a well known result in the LU factorization, which says that if column j updates column k during LU factorization, then j is a descendant of k in $T(\bar{B})$.

One of the most important results in [Gil88] shows that columns in independent subtrees of the elimination tree can be computed (the tasks *U(k,i)* and *F(i)*) without referring to any common memory, because the source columns in the updates have completely disjoint row indices.

The next theorem gives the dependencies between two columns i, i' which are on a same path of the LU etree and which update the same third column.

Theorem 5 *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.

Now we show the dependencies between two columns which are on different branches of the LU etree and which update a same third column. 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 memory. This is because the source columns i, i' in the updates have completely disjoint row indices.

Using all the above results, the structure of a sparse LU task dependence graph can be defined as follows :

1. There are N tasks *F(i)* where $1 \leq i \leq N$
2. There is a task *U(i,k)* if $\bar{B}_{ik} \neq 0$, where $1 \leq i < k \leq n$.
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 $T(\bar{B})$.

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 1: Benchmark matrices

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 DAG for the matrix in the figure 4 (a). The resulting LU etree is in figure 4 (b). From this etree we obtain the corresponding sparse LU task scheduling graph shown in figure 5(b). In figure 5 (a) we show the dependence graph for the same matrix 4 which was used in the S* package.

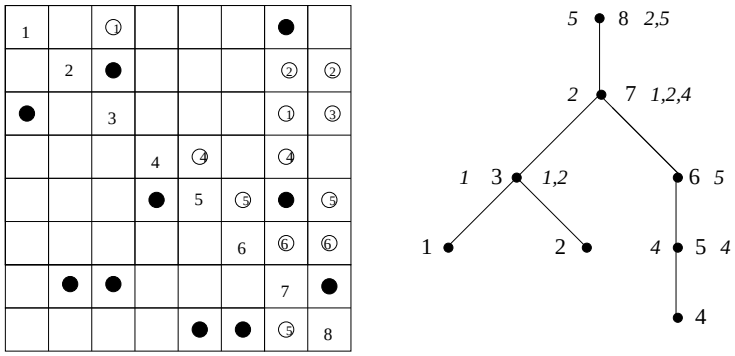


Figure 4: A matrix example $\bar{B}$ and its extended LU etree after the static symbolic factorization and the L/U supernode partitioning was applied

In order to schedule the LU task dependence graph we used the RAPID runtime system [Fu97]. This tool creates a schedule in two steps: first it analyses data accesses for each task, thus obtaining a task dependence graph and second, it distributes efficiently these tasks on a distributed memory machine. RAPID uses an inspector (analyse access patterns) / executor (execute tasks) approach and uses several techniques to optimize interleaved communications and computations.

7 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 were obtained from the Harwell-Boeing Collection and from the ftp site maintained by Tim Davis of the University of Florida[2] and their characteristics are presented in table 1.

From oil reservoir modelling matrices we used sherman3, sherman5, orsreg1 and saylr4. The matrices lnsp3937 and lns3937 are used in fluid flow modelling. 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 chose to test our factorization method on a Silicon Graphics general-purpose cache-coherent distributed shared memory (DSM) architecture, the Origin 2000. The test configuration contains 64 R10000 processors

[2] ftp.cise.ufl.edu/pub/faculty/davis/matrices.

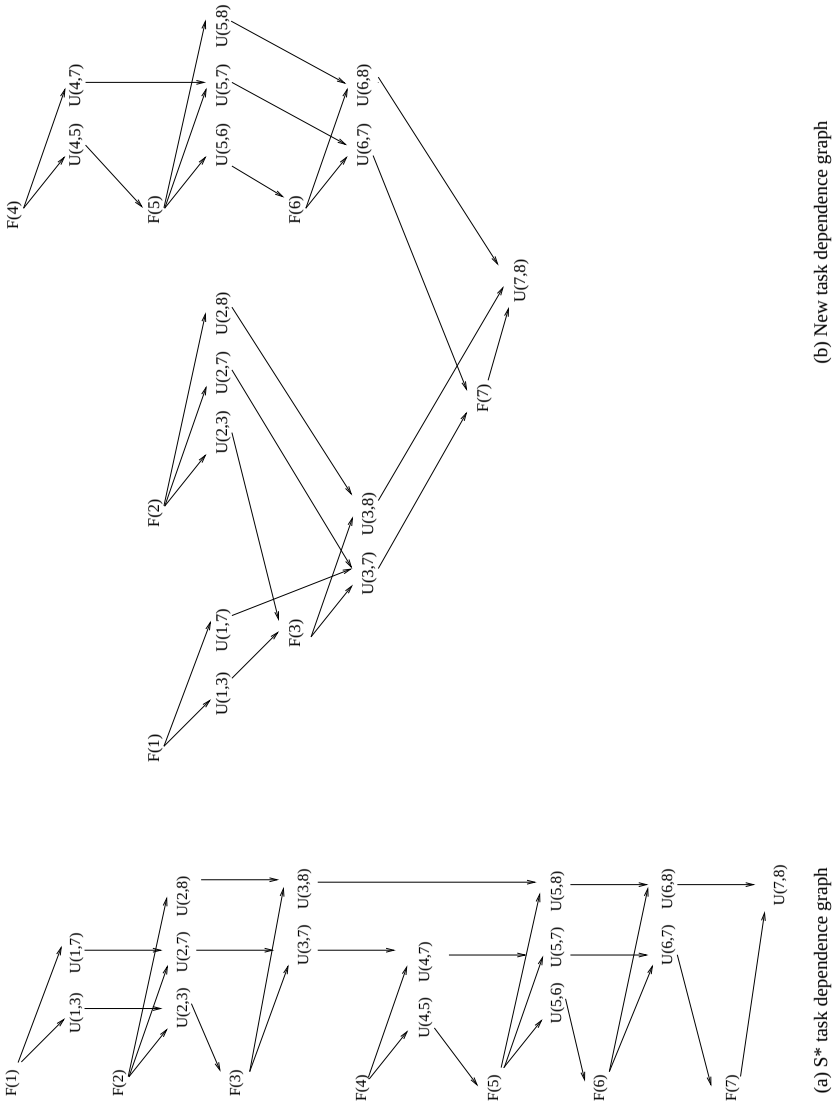


Figure 5: The LU task dependence graph corresponding to $\bar{B}$ in fig 4

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

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

Name	1	2	3	4	5	6	7
NoBlks	2110	1675	39	26	1	1	1
#SN	3082	1857	1365	1357	630	1109	223
#SNPO	2696	1815	1010	1012	406	705	178
1-(#SN/#SNPO)	0.13	0.02	0.26	0.25	0.35	0.36	0.20

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

each 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 SGI's MIPSPro C compiler v7.3.1m with the second optimization level option enabled (`cc -O2`) and scientific software libraries provided by SGI: BLAS levels 1, 2 and 3 optimized implementations (SCSL) and a recent Irix port of Unicos SHMEM for distributed memory access.

In the following, we first 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 efectiveness of each of them.

Overall performance In order to always have 3 blocks in the cache at the same time while doing the updates, we have limited the upper supernode size to 30. All floating point computations are in double precision. We have observed that when block size is 30, DGEMM achieves 106 MFLOPS.

When using the SHMEM library we can obtain a maximum bandwidth of $130 Mbytes/sec$ and a minimum communication overhead (the `shmem_put()` function is also used as a common communication primitive by the RAPID implementation).

In table 7 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, we measure for the different matrices the supernode sizes (without imposing any upper limit) obtained after the L/U supernode partitioning and the amalgamation was 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 partitioning and the amalgamation of the supernodes. Then we measure again the obtained supernode sizes. Results are shown in the 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 behaviour by the large sparsity and lack of structure which will make supernode identification difficult even when postordering is applied.

In figure 4 (a) we present cumulative sum of supernodes occurrences for the saylr4 matrix. For clarity sake, in 4 (b), we remove the tail of the graph corresponding to $x > 67$. We can observe that by using postordering we obtain larger supernodes, thus decreasing the total number of supernodes.

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.

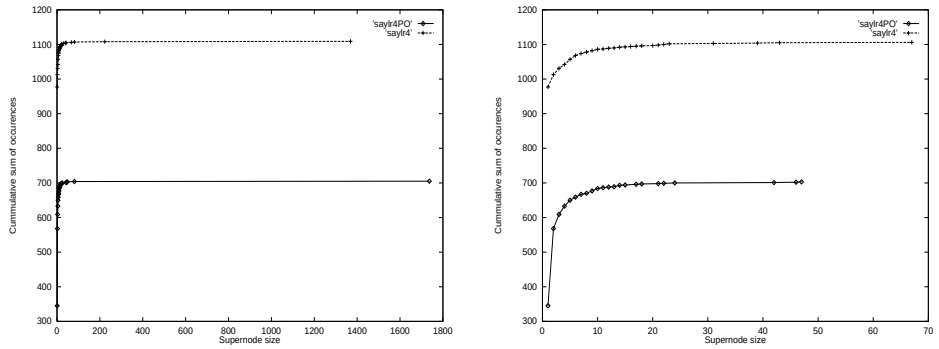


Table 4: Cumulative sum of supernodes occurrences for matrix 6.

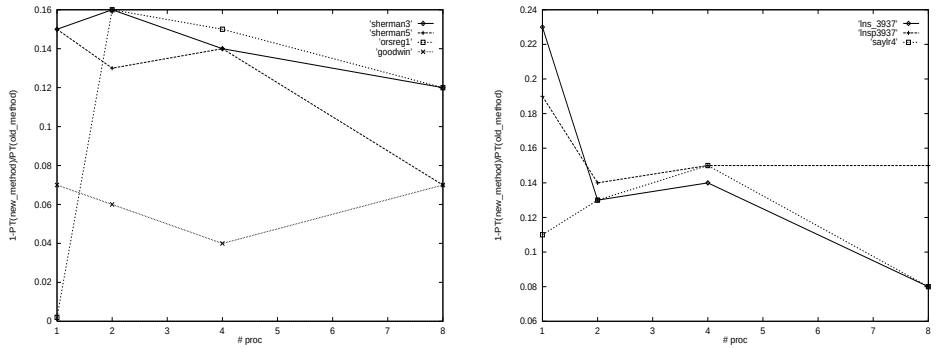


Table 5: Performance improvement by using the new task dependence graph

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* [FY96]. The performance improvement ratio of our approach is listed in figure 5.

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.

8 Conclusions and future work

In this report 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 dependencies, thus exposing more task parallelism for a sparse matrix.

Future work includes the experiment of 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

[ADL98] 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.

[DEG+95] J. W. Demmel, S. C. Eisenstat, J. R. Gilbert, X. S. Li, and J. W. H. Liu. A Supernodal Approach to Sparse Partial Pivoting. Technical Report TR UCB//CSD95-883, U.C. Berkeley, 1995.

[DGL97] J. W. Demmel, J. R. Gilbert, and X. S. Li. An Asynchronus Parallel Supernodal Algorithm for Sparse Gaussian Elimination. Technical Report TR UCB//CSD97-943, U.C. Berkeley, 1997.

[Fu97] Cong Fu. *Scheduling and runtime support for Irregular Computations.* PhD thesis, UCSB, 1997.

[FY96] Cong Fu and Tao Yang. Efficient Sparse LU Factorization with Partial Pivoting on Distributed Memory Architectures. Technical Report TR CS96-18, UCSB, 1996.

[Gil88] John R. Gilbert. An efficient parallel sparse partial pivoting algorithm. Technical Report 88/45052-1, Christian Michelsen Institute, 1988.

[GN87] Alan George and Esmond Ng. Symbolic factorization for sparse gaussian elimination with partial pivoting. *SIAM J. Sci. Stat. Comput.*, 8(6):877–898, 1987.

[GN90] Alan George and Esmond Ng. Parallel Sparse Gaussian Elimination with Partial Pivoting. *Annals of Operations Research*, 22:219–240, 1990.

[Liu90] Joseph W. H. Liu. The role of elimination trees in sparse factorizations. *SIAM J. Matrix Anal. Appl.*, 11(1):134–172, January 1990.

[SJY98] Kai Shen, Xiangmin Jiao, and Tao Yang. Elimination Forest Guided 2D Sparse LU Factorization. Technical Report TR CS98-08, UCSB, 1998.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-0803