Parallel Symbolic Factorization for Sparse LU Factorization with Static Pivoting

L. GRIGORI[1] J.W. DEMMEL[2] X.S. LI[3]

February 20, 2005

[1] *INRIA Rennes, Campus Universitaire de Beaulieu, Avenue du General Leclerc Rennes, 35042, France. email: Laura.Grigori@irisa.fr*

[2] *U.C. Berkeley, 737 Soda Hall,. Berkeley, CA 94720-1776, USA. email: demmel@cs.berkeley.edu*

[3] *Lawrence Berkeley National Laboratory, One Cyclotron Road, Mail Stop 5 0F-1650 Berkeley, CA 94720-8139, USA. email: XSLi@lbl.gov*

In this paper we consider a direct method to solve a sparse unsymmetric system of linear equations $Ax = b$, which is the Gaussian elimination. This elimination consists in explicitly factoring the matrix A into the product of L and U, where L is a unit lower triangular matrix, and U is an upper triangular matrix, followed by solving $LUx = b$ one factor at a time. One of the main characteristics of the sparse LU factorization is the notion of fill-in. This notion denotes an element that was zero in the original matrix A, but becomes nonzero during the factorization. As these fill-ins can be computed without referring to the numerical values of the matrix, and to be able to allocate memory and to organize computations before calculating the numerical values of the factors, the resolution of a sparse system is divided into several phases. We present here the specific phases of SuperLU_DIST [4], a widely used algorithm to solve large sparse unsymmetric systems on distributed memory computers. The first step consists in choosing a permutation matrix P_1 and diagonal matrices D_1 and D_2 so that $P_1 D_1 A D_2$ has large entries on the diagonal. This helps assure accuracy of the final solution. The second step orders equations and variables by choosing a permutation matrix P_2 so that the factors L and U of $P_2^T P_1 D_1 A D_2 P_2$ are as sparse as possible. The third step performs a *symbolic analysis*, that is it identifies the locations of nonzero entries of L and U. And finally, the fourth step computes the numerical values of the factors L and U.

We discuss the design and the implementation of a memory scalable symbolic factorization algorithm for unsymmetric matrices on distributed memory machines. Its integration in SuperLU_DIST will transform this solver into a fully parallel solver. Earlier work has addressed the parallelization of the numerical factorization (step 4 in SuperLU), because its complexity is generally of higher order compared to the other steps. This is now a well understood problem and the algorithm implemented in SuperLU proved to be highly parallel and efficient. Techniques were proposed for computing fill-reducing ordering in parallel (step 2 in SuperLU), and we will review them briefly later in this section. More recent research focused on the development of efficient parallel algorithms for permuting large entries on the diagonal (step 1 in SuperLU). All these algorithms use distributed data structures, and in particular the input matrix A is distributed over the processors. They offer an overall good scalability. This includes memory scalability, i.e. if both the problem size and the number of processors is increased by the same factor, then the same amount of memory is used per processor.

The symbolic factorization is the only step that is sequential and that needs the input matrix to reside on one processor. Thus it currently represents a memory bottleneck. This paper presents the design and the implementation of a parallel symbolic factorization algorithm, which is suitable for general sparse unsymmetric matrices. Its main goal is to decrease the memory requirements of the symbolic factorization step.

From an algorithmic point of view, Rose and Tarjan [5] showed that the symbolic factorization problem is related to the transitive closure problem. That is, any algorithm that computes the fill-in for a specific ordering can be used to compute the transitive closure of a graph.

The challenge in formulating a scalable parallel algorithm lies not only in the small amount of computation to be distributed among processors, but also in the sequential data dependency. The relation to the transitive closure algorithm implies that an improved algorithm for fill-in computation would give an improvement over the best transitive closure algorithms. Since the parallelization of the transitive closure problem has been thoroughly studied, it seems unlikely that there is an easy way to find a formulation of the symbolic factorization, in which the coarse grain parallelism would be inherent.

Guided by this observation, our goal is to develop an algorithm that provides memory scalability. Speedup does not play the principal role in this development, but our purpose is to obtain enough speedup to prevent this step from being a computational bottleneck in a parallel solver. For this, the algorithm exploits two types of parallelism. The first type of parallelism relies on the usage of a graph partitioning approach to reduce the fill-in. In particular, the partition of the matrix is suitable for parallel execution. The second type of parallelism is based on a block cyclic distribution of the data, a standard technique in parallel scientific computing.

The choice of relying on a graph partitioning approach is in accordance with the state of the art in the parallelization of reordering techniques. These techniques fall into two basic categories. One consists in using a local strategy to minimize the amount of fill-in, and the second one consists in using a graph partitioning approach to permute the matrix to some particular form and so confining the fill-in.

The algorithms belonging to the first category proved to be difficult to parallelize. The minimum degree ordering is one such example. This algorithm minimizes at each elimination step the fill that can be introduced locally, without considering the effects on the later steps. The parallelization of one of its variants is discussed for example by Chen et al [1]. They have designed a parallel algorithm for shared memory machines, and they have observed a limited speedup. As main sources of inefficiency, they discuss lack of parallelism and load imbalance.

Nested dissection is an algorithm that belongs to the second category. The key concept is the computation of a vertex separator, that splits the matrix into two disconnected parts. The variables corresponding to the first part are ordered, followed by those of the second part, and finally by those of the separator. The disconnected parts can be themselves further divided by the computation of new separators, with the recursion continuing to any depth. The main advantage of this partitioning is that the form of the matrix is suitable for parallel execution. Following a similar graph partitioning approach, the multilevel schemes [2, 3] proved to be well parallelized on distributed memory machines, while producing comparable quality partitioning to the sequential techniques.

The above discussion has outlined the advantages of using a graph partitioning approach in a fully parallel solver. This motivates our choice to rely on a corresponding partitioning of the input matrix in our parallel algorithm.

We will first review several sequential symbolic factorization algorithms. Then we will discuss the parallel symbolic factorization algorithm and we will present the experimental results obtained when applying the algorithm on real world matrices.

References

[1] T.-Y. Chen, J. Gilbert, and S. Toledo. Toward an efficient column minimum degree code for symmetric multiprocessors. *Proceedings of the 9th SIAM Conference on Parallel Processing for Scientific Computing*, 1999.

[2] B. Hendrickson and R. Leland. A Multilevel Algorithm for Partitioning Graphs. *Proceedings of SuperComputing*, 1995.

[3] G. Karypis and V. Kumar. A parallel algorithm for multilevel graph partitioning and sparse matrix ordering. *Journal of Parallel and Distributed Computing*, 48(1):71–95, 1998.

[4] X. Li and J. Demmel. SuperLU_DIST: A Scalable Distributed-memory Sparse Direct Solver for Unsymmetric linear systems. *ACM Transactions on Mathematical Software*, 29(2), 2003.

[5] D. J. Rose and R. E. Tarjan. Algorithmic aspects of vertex elimination on directed graphs. *SIAM J. Appl. Math.*, 34(1):176–196, 1978.