Solving Sparse Systems of Linear Equations on the Connection Machine

Charles E. Leiserson[1,3]
Jill P. Mesirov[1]
Lena Nekludova[1]
Stephen M. Omohundro[1]
John Reif[1,2]
Washington Taylor[1]

[1]Thinking Machines Corporation
245 First Street
Cambridge, Massachusetts

[2]Aiken Computation Laboratory
Harvard University
Cambridge, Massachusetts

[3]Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts

Abstract—The Connection Machine is a 65,536-processor computer which was designed for artificial intelligence applications. This paper shows that the machine is suitable for numerical computations as well. We describe a program for solving sparse systems of linear equations based on parallel nested dissection which has been implemented on the Connection Machine.

Summary

A commonly occurring problem in the physical sciences is to solve a system $Ax = b$ of linear equations, where A is an n-by-n symmetric, positive-definite matrix, and b is a vector of length n. If the matrix A is dense, then the worst-case time required by most direct methods is $\Theta(n^3)$ on a sequential machine. For many important applications, however, the matrix A is sparse, and its adjacency graph is planar or nearly planar. For such problems, *nested dissection* [3,7] can reduce the running time to $O(n^{3/2})$. Recently, Pan and Reif [9] described a parallel nested-dissection algorithm that runs in $O(\log^3 n)$ time, but the space requirement is $\Theta(n^{3/2})$, which makes it infeasible for large

Approved for External Distribution
Thinking Machines Corporation

systems of equations. In this paper, we describe an implementation of their algorithm that runs in $O(\sqrt{n})$ time and uses $O(n \log n)$ space for planar graphs, but it can be applied more generally to any class of graphs that has a good *separator* [8], although the performance bounds may vary. The program has been implemented on the 65,536-processor Connection Machine [4].

The parallel nested-dissection algorithm has three parts. First, we recursively compute a decomposition tree for the adjacency graph of the matrix A based on a separator theorem for that class of graphs. This step is quickly performed in $O(n \log n)$ time on a sequential machine. Second, we use the Connection Machine to compute a Cholesky-like decomposition of the matrix. Finally, we use the Connection Machine to solve the linear system for a given righthand side b.

To implement the first step, we use a sequential computer to compute the decomposition tree for the adjacency graph of the matrix A. If the graph is planar, we use the separator property [8] to decompose the graph into two subgraphs such that the number of common variables is $O(\sqrt{n})$. (Otherwise, we use bisection heuristics [2,5] to find a good separation.) We then decompose each of these two subgraphs, and so on, until each subgraph contains only a few variables. We thus obtain a decomposition tree of the original graph, each node of which is a subgraph. The portion of the matrix that corresponds to variables in a given node are stored in that node, and in that form, the matrix is transferred to the Connection Machine.

The second phase of the algorithm runs on the Connection Machine. It consists of computing a Cholesky-like decomposition of the matrix based on the decomposition tree. In parallel, all leaf nodes perform a partial factorization by Givens rotations [10, pp. 206] on their portions of the matrix. The factorization is performed by a parallel systolic algorithm [1,6], and in addition, the matrices in all leaves are operated on simultaneously. The partially factored matrices in each pair of siblings are then merged into the matrix of their parent. We once again perform partial factorization on all matrices at the parents' level in parallel, and continue level by level until we reach the root. This process produces a sparse representation of the Cholesky decomposition with matrix elements distributed through the nodes of the tree. Conceptually, each tree node contains portions of both the lower and upper triangular matrices in the LU-decomposition.

In the final phase we solve the matrix system for a vector b. The vector is distributed through the tree nodes so that the matrix variables in the nodes correspond to the variables of the vector. On the way up the tree, we use the pieces of b in a given node to solve the equation $Ly = b$ by forward substitution, and on the way down, we solve the equation $Ux = y$ by back

substitution to obtain the solution for the original matrix equation.

This parallel nested-dissection algorithm is well suited for implementation on the Connection Machine. The architecture of this machine currently has 65,536 bit-serial processors interconnected by a routing network. All processors receive the same instruction broadcast from a central controller, but whether a processor executes the instruction depends on an internal flag. The broadcast instruction can cause a processor to manipulate internal data, or it can direct communication among the processors either by mailing messages through a general-purpose routing network or by sending data to a nearest neighbor via a grid-like NEWS (North-East-West-South) network.

Rather than cycling a small number of processors through all the problem elements as might be done on many multiprocessors, it is natural on the Connection Machine to devote a separate processor to each problem element. We take advantage of this simplicity of representation in the parallel nested-dissection algorithm in two ways. First, we are able to use standard systolic algorithms to perform the matrix computations in each node of the decomposition tree. Second, all the computations at each level of the decomposition tree can be performed simultaneously. We use the general-purpose routing network to send data from one level of the tree to the next.

Another not-so-apparent advantage of the Connection Machine architecture is the flexibility of the bit-serial processors. As an example, we can use the somewhat more numerically stable Givens rotations instead of Gaussian elimination without pivoting. The times required by the two methods are comparable because the square-root operation needed by the Givens method is no more expensive than the other bit-serial floating-point operations.

The Connection Machine architecture is well suited to the development of scientific computing software, even though it was originally conceived as an AI machine. It offers an abstract model based on "data" parallelism rather than "process" parallelism. In other words, the programmer is responsible for planning the movement of data, but not for processor coordination. This abstract model leads to straightforward implementations of efficient parallel algorithms with predictable performance over a range of problem sizes.

References

[1] A. Bojanczyk, R. P. Brent, and H: T. Kung, Numerically stable solution of dense systems of linear equations using mesh-connected processors, Technical Report CMU–CS–81–119, Department of Computer Science, Carnegie-Mellon University, January 1931.

[2] C. M. Fiduccia and R. M. Mattheyses, A linear-time heuristic for

improving network partitions, *Proceedings of 19th Design Automation Conference,* IEEE, (1982) pp. 175–181.

[3] J. A. George, Nested dissection of a regular finite element mesh, SIAM J. Numer. Anal., 10, (1973) pp. 345–363.

[4] W. Daniel Hillis, *The Connection Machine,* MIT Press, 1985.

[5] B. W. Kernighan and S. Lin, An efficient heuristic procedure for partitioning graphs, *Bell System Technical Journal,* 49, (1970) pp. 291–307

[6] A. H. Sameh and D. J. Kuck, On stable parallel linear system solvers, *J. ACM,* 25(1) (1978) pp. 81–91.

[7] R. J. Lipton, D. J. Rose and R. E. Tarjan, Generalized nested dissection, *SIAM J. Numer. Anal.,* 16(2), (1979) pp. 346–358.

[8] R. J. Lipton and R. E. Tarjan, A separator theorem for planar graphs, *SIAM J. Appl. Math.,* 36, (1979) pp. 177–199.

[9] V. Pan and J. Reif, Efficient parallel solutions of linear systems, *Proceedings of the 17th Annual Symposium on Theory of Computing,* ACM, Providence, Rhode Island, May 1985, pp. 143–152.

[10] S. Pissanetzky, *Sparse Matrix Technology,* Academic Press, 1984.