SIMULATION OF SEMICONDUCTOR DEVICES AND PROCESSES Vol. 5
Edited by S. Selberherr, H. Stippel, E. Strasser – September 1993

The Application of Sparse Supernodal Factorization Algorithms for Structurally Symmetric Linear Systems in Semiconductor Device Simulation

A. Liegmann and W. Fichtner

Integrated Systems Laboratory, ETH-Zürich
Gloriastraße 35, CH-8092 Zürich, SWITZERLAND

1. Introduction

It is well known that the solution of sparse linear systems, generally expressed in the form `Ax = b`, is a core task of numerical simulation. In case of semiconductor device simulation the coefficient matrix `A` is unsymmetric, but structurally symmetric ([2]). The solution of linear systems can be achieved by iterative or direct methods. While iterative methods do not always lead to a solution due to matrix conditions, direct methods usually consume more time and memory.

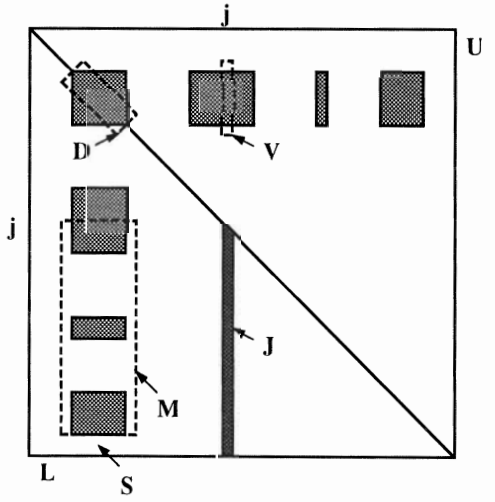

Figure 1: Updating column `J` by supernode `S`

2. Computational considerations

The problem of reducing the memory needed for a direct solver is closely related to the problem of minimizing the size of the LU decomposition as a result of reordering the coefficient matrix. In this respect the Minimum Degree reordering algorithm proved to be very successful for general sparse systems. This algorithm has been enhanced in terms of execution speed to what is referred to as the **Multiple Minimum Degree** algorithm ([5]).

On the other hand, speeding up a direct solver basically means a faster computation of the LU factorization. Extensive research in this area has lead to so-called *supernodal* techniques. The key concept of these techniques is what is nowadays referred to as a **supernode** [1]. During symbolic factorization, supernodes are identified as a set of consecutive columns in the factor L of the LU decomposition with the following structural properties. A supernode formed by, s say, adjacent columns consists of two blocks: a dense diagonal block of size $s \times s$ and a block of width s below the diagonal block where all columns share the same sparsity pattern. A sample supernode is depicted in Figure 1 denoted with the letter S. Computing column J involves the following steps: for all supernodes S updating column J determine vector V1 = M*(D·V) and then subtract it from the contents of J, i.e. J = J − V1 (see Fig. 1). The determination of vector V1 involves dense matrix-vector multiplications which run at vector speed on today's supercomputers. The subtraction step requires gather/scatter operations which are mostly hardware supported on many supercomputers. As a result, supernodal techniques make excellent use of the hardware features and thus are highly powerful.

The computational power of supernodal techniques applied to symmetric positive definite linear systems has been documented in papers like [1]. Since in semiconductor device simulation the linear systems are usually unsymmetric but structurally symmetric, we could apply supernodal techniques to both factors L (columns) and U (rows) simultaneously ([3]). Further enhancements have been added and more recently we implemented a whole collection of supernodal factorization algorithms which divide into supernode-node and supernode-supernode or block supernode methods ([4]).

```
for  J = 1 to  N_s  do                                  (1)
   for  j ∈ J (in order) do                             (2)
      for all K updating j do                           (3)
         determine V1 and update j
      end for
      scale j with diagonal element                     (4)
   end for
end for
```

Figure 2: General supernodal algorithm

3. Supernode-node algorithms

Supernode-node updating describes a technique where only one column/row of the factors L and U is computed at a time, although the corresponding supernode might

consist of several columns/rows. Figure 2 depicts the general framework for algorithms implementing this technique. A first glance at the algorithm already reveals the general form of supernode-node updating algorithms: a triple-nested for-loop (indicated with indices 1 through 3). The outermost loop runs over all supernodes J that were generated in the reordering and symbolic factorization steps. The next for-loop (2) goes one level deeper and scans over all nodes j of the current supernode J starting with the smallest index. The innermost loop (3) handles the contribution of all updating supernodes K to the current node j. Finally, column/row j has to be scaled by its diagonal element (4).

4. Block supernode algorithms

Block supernode factorization operates on groups of columns/rows or even a whole supernode at the same time instead of merely focusing on a single column/row. Doing so does not reduce the number of references to memory by any means, but by grouping them together memory fetch and store can be made more efficiently, i.e. using the same index map only once throughout a loop cycle ([6]).

On the other hand, supernode-supernode factorization increases storage overhead significantly, since the intermediate results for more than one column/row have to be kept and other data structures had to be added to support this technique. In our tests we have seen memory increase between at worst 6 to 20 times over our supernode-node implementations. Furthermore, the time necessary to do the set up and administration of these data structures cannot be neglected.

5. Benchmark

We present the timing results for a medium sized linear system stemming from a simulation of a MOSFET. The linear system has 12,000 unknowns and about 250,000 non-zero entries in the coefficient matrix. The benchmark was run on a Convex C220, a Cray-2, a Cray Y-MP, a NEC SX-3, and a Cray C98. The numbers shown in Table 1 are those of the best performing factorization algorithm in CPU seconds. For none of

	supernode-node	block supernode
Convex C220	12.93	18.93
Cray-2	3.38	3.48
Cray Y-MP	1.39	1.39
NEC SX-3	1.23	1.38
Cray C98	.73	.74

Table 1: Timing results for the best performing algorithm (seconds)

the machines used in the benchmark we found block supernode algorithms to perform better than the supernode-node algorithms. Mainly, there are two reasons for that:

- For all of the block supernode algorithms implemented we noticed a significant increase of scalar memory references. This increase is stemming from the additional data structure handling built into the block supernode algorithms. Obviously, this hurts especially on machines with scalar data caches like the Convex and the NEC. Here, block supernode methods loose performance by suffering from scalar data cache misses.

- Block supernode techniques are most effective when the supernodes contain many columns/rows, i.e. when the supernode partitioning consists of a small number of supernodes. A small supernode partitioning provides for bigger blocks during supernode update. In our test cases supernodes contain 5 to 6 columns/rows on average. Additionally, our factor columns/rows are very sparse (about 200 non-zero entries maximum) so that there are only a few cases during the factorization where we can exploit the potential of the block supernode algorithms.

6. Conclusion

In this paper we presented supernodal techniques suitable for structurally symmetric linear systems as they appear in semiconductor device simulation. Among these, supernode-node updating schemes perform best for this type of application. We have shown that block supernode methods cannot be exploited to their full potential which is due to the extreme sparsity of the linear systems and small supernode sizes.

7. Acknowledgement

The authors highly appreciate the support of Cray Research (Switzerland) for providing the original source code. Also, we thank the staff of the computer centers of the Swiss Institutes of Technology in Zurich and Lausanne as well as the Swiss Scientific Computing Center in Manno for providing us access to their supercomputers. Additionally, we are grateful to J.F. Bürgler and S. Müller (both from the Integrated System Laboratory) for providing the set of test cases. Our special thanks go to C. Pommerell (now AT&T Bell Labs) for a number of interesting discussions on the hallway of the laboratory, and to B.W. Peyton (ORNL) for his help on understanding the original code.

References

[1] C.C. Ashcraft, R.R. Grimes, J.G. Lewis, B.W. Peyton, and H.D. Simon. Progress in sparse matrix methods for large linear systems on vector supercomputers. *The International Journal of Supercomputer Applications*, 1(4):10–30, 1987.

[2] G. Heiser, C. Pommerell, J.Weis, and W. Fichtner. Three dimensional numerical semiconductor device simulation: Algorithms, architectures, results. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 10(10):1218–1230, 1991.

[3] A. Liegmann. The application of supernodal techniques on the solution of structurally symmetric systems. Technical Report 92/5, Institut für Integrierte Systeme (ETH Zürich), 1992.

[4] A. Liegmann and W. Fichtner. The application of supernodal factorization algorithms for structurally symmetric linear systems in semiconductor device simulation. Technical Report 92/17, Institut für Integrierte Systeme (ETH Zürich), 1993. Submitted to *The International Journal of Supercomputer Applications.*

[5] J.W.H. Liu. Modification of the Minimum-Degree algorithm by multiple elimination. *ACM Transactions on Mathematical Software*, 11(2):141–153, 1985.

[6] E.G. Ng and B.W. Peyton. Block sparse Cholesky algorithms on advanced uniprocessor computers. Technical Report TM-11960, Oak Ridge National Laboratory, 1991.