Efficient Hybrid Solution of Sparse Linear Systems

A. Liegmann[a], K. Gärtner[b], W. Fichtner[a,b]

[a]Integrated Systems Engineering AG,
Im Dornacher 8, 8127 Forch, SWITZERLAND
[b]Institut für Integrierte Systeme,
ETH-Zentrum, Gloriastr. 35, 8092 Zürich, SWITZERLAND

Abstract

During a numerical simulation usually many linear systems have to be solved. Using a direct method requires to factor each coefficient matrix separately. In this paper we present a hybrid approach which combines our supernodal direct solver with an iterative solver such that the iterative solver is called as often as possible to avoid the computationally expensive factorizations.

1. Motivation

In the last few years the research efforts towards efficient solutions of sparse linear systems using direct or iterative methods have made significant progress. Unfortunately, it seems impossible to find an universal method which is optimal with respect to memory consumption and computational speed for all types of problems. Therefore, so-called *hybrid* methods have been considered which use a combination of usually stand-alone solution techniques. The idea behind such hybrid methods is that during the solution of a problem one can select the method which is known to work best on a particular phase of the solution process whereas the normal approach uses only one method for the whole solution process. Our approach uses a combination of iterative and direct techniques to solve sparse structurally symmetric linear systems as they appear in numerical semiconductor simulation.

2. The hybrid approach

Numerical semiconductor device simulation involves the solution of the discretized device equations usually by a Newton approximation algorithm where each Newton step requires the solution of a linear system describing the coupled device equations. Especially during transient simulations a large number of linear system solves are necessary in order to complete a simulation. The usual direct approach requires a factorization for each linear system to be solved. Since factoring the coefficient matrix is the most time consuming part of the solution process, it has to be avoided as often as possible. Unfortunately, direct methods provide no means to avoid the factorization. Consequently, we apply a preconditioned iterative method using a given factorization

```
call SUPER(A^(0),x^(0),b^(0))
LU = L^(0)U^(0)
x̄ = x^(0); b̄ = b^(0)
for i = 1,2,... do
  z = ||b̄ − A^(i)x̄||/||b̄||
  if z > threshold do
    call SUPER(A^(i),x^(i),b^(i))
    LU = L^(i)U^(i)
    x̄ = x^(i); b̄ = b^(i)
  else
    call CGS(A^(i),x^(i),b^(i))
    if CGS fails do
      call SUPER(A^(i),x^(i),b^(i))
      LU = L^(i)U^(i)
      x̄ = x^(i); b̄ = b^(i)
    end if
  end if
end for
```

Algorithm 1: The supernodal hybrid solution approach.

as a preconditioner. On the other hand, we are willing to invest a factorization, if convergence of the iterative procedure is slow.

Algorithm 1 outlines the strategy of our hybrid approach which combines our sparse linear solver *SUPER* [1] with the preconditioned *conjugate gradient squared* (CGS) iterative method by Sonneveld [2] which is known to converge fast. As a preconditioner the most recent factorization is used. Initially, our sparse solver *SUPER* is called which solves the first linear system of the simulation process. Upon return from the direct solver the LU factorization of the coefficient matrix, the solution vector, and the right-hand side are saved in the data structures $\overline{LU}$, $\bar{x}$, and $\bar{b}$, respectively. There are used to estimate the norm of the matrix difference $\|A^{(i)} - \overline{LU}\|$. All further linear systems are solved by either *SUPER* or CGS. The iterative process is started only if condition

$$threshold \geq \|\bar{b} - A^{(i)}\bar{x}\|/\|\bar{b}\| \tag{1}$$

is satisfied. Parameter *threshold* denotes an upper limit for the relative residual norm of the current coefficient matrix and the solution and right-hand side vectors from the last exact solution. This relative residual norm is a measure how much the two vectors $\bar{b}$ and $A^{(i)}\bar{x}$ differ. If the relative residual norm is small, one can conclude that the numerical values of $A^{(i)}$ and $\overline{LU}$ do not differ too much so that CGS is expected to converge fast using $\overline{LU}$ as the preconditioner. If condition (1) does not hold, *SUPER* is called; otherwise CGS is invoked. On the other hand, even if condition (1) is satisfied, CGS is not guaranteed to succeed. In this case *SUPER* is called to compute an exact solution.

3. Preconditioned CGS

Because we want to solve sets of linear equations with continuous parameter dependent coefficient matrices, we expect the iterative procedure to be invoked only for cases with eigenvalues of the iteration matrix sufficiently close to zero. If this is true, CGS will converge fast; otherwise, one observes large oscillations in the residual norm [3]. This fact is used to interrupt the iterative procedure and to calculate a new factorization. Furthermore, we restrict the number of iterations by the parameter *maxiter* which is computed as the quotient of the measured times for a factorization and for the first CGS iteration (initially, *maxiter* is set to 1). After the first iteration *maxiter* is adjusted according to the measured times and the following formula:

$$maxiter = \left\lfloor \frac{T_{factorization}}{T_{first\_CGS\_iteration}} \times c \right\rfloor, \; c \leq 1. \tag{2}$$

procedure *CGS*
  $x = 0;\ r = b$
  for $i = 1$ **to** *maxiter* **do**
    $\rho_1 = b^T r$
    if $(\rho_1 = 0)$ return failure
    if $i = 1$ **do**
      $u = r;\ p = r$
    else
      $\beta = \rho_1/\rho_2;\ u = r + \beta q$
      $p = u + \beta q + \beta^2 p$
    end if
    $\rho_2 = \rho_1$
    solve $\overline{LU}\hat{p} = p$
    $\hat{v} = A\hat{p};\ \alpha = \rho_1/(b^T\hat{v})$
    $q = u - \alpha\hat{v}$
    solve $\overline{LU}\hat{v} = u + q$
    $u = A\hat{v};\ x = x + \alpha\hat{v}$
    $r = r - \alpha v$
    call *check_convergence*(i)
  end for
end procedure

Algorithm 2: The preconditioned CGS algorithm.

In other words, *maxiter* is set to reflect how many CGS iterations can be executed, not exceeding the time required for a full factorization.

After the necessary vectors and scalars have been computed, CGS requires to solve two linear systems. At this point, the preconditioner comes into play. Recall from the previous section that our preconditioner is the LU factorization of the most recent exact solve which is denoted as $\overline{LU}$. Consequently, only forward and backward substitution are required to solve the systems $\overline{LU}\hat{p} = p$ and $\overline{LU}\hat{v} = u + q$. This improves the computational efficiency of the CGS algorithm significantly.

Eventually, the solution vector x and the residual r are updated. The relative residual tolerance $z = \|r\|/\|b\|$ is then used to check convergence of the preconditioned CGS method. Depending on this value, it is decided whether CGS is considered to have failed, the solution is found, or another CGS iteration has to be performed.

At this point of the CGS algorithm, the method is considered to have failed in the following cases:

1. $z > maxnorm$
   If the relative residual norm exceeds the predefined limit given by the parameter *maxnorm*, the CGS process is expected to converge too slowly.

2. $i = maxiter/2 \wedge z > \sqrt{mintol}$
   Parameter *mintol* specifies the accuracy z has to achieve so that the linear system is considered as solved. If we have not reached an accuracy of $\sqrt{mintol}$ after half of the iterations allowed, convergence is considered to be too slow.

3. $i \geq maxiter \wedge z > mintol$
   If CGS has not been able to solve the system with accuracy *mintol* after *maxiter* iterations, the iterative process is stopped.

These failure criteria are used to optimize the behavior of CGS in a way that performance loss is minimized if the CGS process has to be stopped and replaced by a direct solve.

4. Performance of the hybrid solver

Figure 1 displays the effect of our hybrid approach on the overall execution time for a 3D transient simulation of an IGBT. The simulation was performed on an

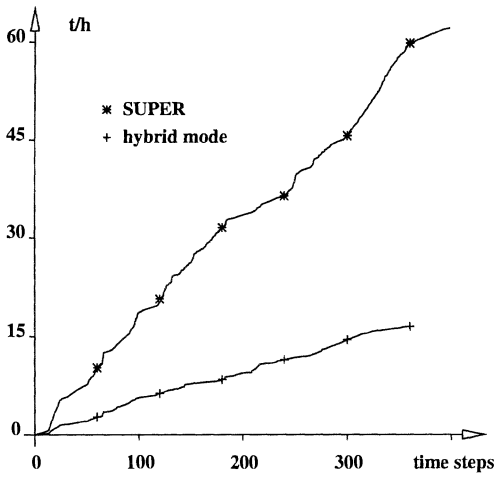

Figure 1: The effect of using a hybrid approach on the overall execution time for a 3D transient simulation of an IGBT (grid size: 7790 points) on an IBM RS/6000-590. The upper curve denoted with $*$, depicts the accumulated time when *SUPER* is used exclusively. The lower curve marked with $+$ shows the accumulated simulation time using the hybrid approach.

IBM RS/6000-590 workstation. The grid size of this device was 7790 points (i.e. one coupled linear system has $n = 3 \times 7790 = 23370$ unknowns). The upper curve, marked with $*$, shows the accumulated execution time of the simulation when only the direct sparse supernodal solver *SUPER* was used. Here, the simulation required more than 60 hours wall clock time to complete. The lower curve, marked with $+$, displays the significantly smaller execution time of the same simulation using the hybrid approach. In this case, the simulation was finished after 16 hours wall clock time. This means, using the hybrid approach we were able to speed up the simulation by almost a factor of four! The reason for this speedup is that out of the 4819 linear systems solved during the simulation only 446 factorizations were required. The remaining systems could be solved by CGS.

References

[1] A. Liegmann and W. Fichtner, "The application of sparse supernodal factorization algorithms for structurally symmetric linear systems in semiconductor device simulation", In S. Selberherr, H. Stippel, and E. Strasser, editors, *Simulation of Semiconductor Devices and Processes*, vol. 5, pp. 77–80, Springer-Verlag, 1993.

[2] P. Sonneveld, "CGS, a fast Lanczos-type solver for nonsymmetric linear systems", *SIAM Journal on Scientific and Statistical Computing*, vol. 10, pp. 36–52, 1989.

[3] H.A. van der Vorst, "Lecture notes on iterative methods", Technical Report No. 838, Department of Mathematics, University, Utrecht, 1993.