Thirteenth International Conference on Domain Decomposition Methods
Editors: N. Debit, M.Garbey, R. Hoppe, J. Périaux, D. Keyes, Y. Kuznetsov

49 Domain decomposition methods in semiconductor device modeling

L. Giraud[1], J. Koster[2], A. Marrocco[3], J.-C. Rioual[4]

Introduction

In this paper, we present some parallel implementations of domain decomposition techniques for the solution of the drift diffusion equations involved in 2D semiconductor device modeling. The model describes the stationary state of a device when biases are applied to its bounds. The mixed dual formulation is retained. Therefore, we have to deal with a system of six totally coupled nonlinear partial differential equations. This system is decoupled and discretized in time by a semi-implicit nonlinear scheme using local time stepping. At each time step, we have to solve three systems of two nonlinear partial differential equations. The first system is associated with electrostatic potential, the second with the negative charges (electrons) and the third with the positive charges (holes). Each pair of equations is naturally discretized in space by mixed finite elements defined on 2D unstructured meshes and then solved by a Newton-Raphson method [HM94]. At each step of the Newton-Raphson method, a linear system of equations has to be solved. Depending upon which nonlinear system is being solved, these linear systems can be either symmetric positive definite or unsymmetric. These systems are sparse with a maximum of five nonzero entries per row due to the mixed finite element triangulation. A complete simulation is decomposed into two phases: first the solution of the equilibrium problem, then the solution of the static problem. The equilibrium problem consists of applying a zero potential to the bounds of the device and its numerical solution only involves the solution of symmetric positive definite linear systems. In this paper, we consider only the solution of the equilibrium problem.
Our objective is to obtain a fully parallel code in a distributed memory environment with MPI as message-passing library.
The formulation is mainly vectorial and is naturally parallelizable. The main difficulty consists of the efficient implementation of suitable linear solvers. This numerical kernel is the most time consuming part of the code. In this respect, we investigate substructuring approaches using either direct or iterative linear solvers.

Substructuring techniques

We assume that the domain Ω with boundary $\partial\Omega$ is partitioned into N non-overlapping subdomains $\Omega_1, \ldots, \Omega_N$ with boundaries $\partial\Omega_1, \ldots, \partial\Omega_N$. After discretization, we obtain a linear system $Au = f$, where the matrix A is sparse, unstructured, and symmetric positive definite. Let B be the set of all the indices of the discretized points which belong to the interfaces between the subdomains. Grouping the points corresponding to B in the vector u_B and the

[1]CERFACS, 42 Av. G. Coriolis, 31057 Toulouse, France. giraud@cerfacs.fr
[2]Parallab, University of Bergen, N-5020 Bergen, Norway. jak@ii.uib.no
[3]INRIA, Rocquencourt, 78153 Le Chesnay Cedex, France. Americo.Marrocco@inria.fr
[4]CERFACS, 42 Av. G. Coriolis, 31057 Toulouse, France. rioual@cerfacs.fr

points corresponding to the interior I of the subdomains in u_I, we get the reordered problem

$$\begin{pmatrix} A_{II} & A_{IB} \\ A_{IB}^T & A_{BB} \end{pmatrix} \cdot \begin{pmatrix} u_I \\ u_B \end{pmatrix} = \begin{pmatrix} f_I \\ f_B \end{pmatrix} . \tag{1}$$

The matrix A_{II} can be reordered to a block diagonal matrix in which the i-th diagonal block A_{ii} corresponds to the internal variables of subdomain Ω_i. Eliminating u_I from the second block row of Equation (1) leads to the reduced problem

$$S u_B = f_B - A_{IB}^T A_{II}^{-1} f_I , \quad \text{where} \quad S = A_{BB} - A_{IB}^T A_{II}^{-1} A_{IB} \tag{2}$$

is the Schur complement matrix of the matrix A_{II} in A. The matrix S is symmetric positive definite,

Let Γ be the interface between the subdomains. The local interface of a subdomain can be defined as $\Gamma_i = \partial\Omega_i \backslash \partial\Omega$. Let $R_i : \Gamma \to \Gamma_i$ be the canonical pointwise restriction which maps vectors defined on Γ onto vectors defined on Γ_i, and let $R_i^T : \Gamma_i \to \Gamma$ be its transpose. For a stiffness matrix A arising from a finite element discretization, the Schur complement matrix can be written as the sum of N (local) smaller Schur complement matrices

$$S = \sum_{i=1}^{N} R_i^T S^{(i)} R_i, \quad \text{where} \quad S^{(i)} = A_{\Gamma_i}^{(i)} - A_{i\Gamma_i}^T A_{ii}^{-1} A_{i\Gamma_i} , \tag{3}$$

where the local Schur complement matrix $S^{(i)}$ associated with subdomain Ω_i is computed from the subdomain stiffness matrix $A^{(i)}$, defined by

$$A^{(i)} = \begin{pmatrix} A_{ii} & A_{i\Gamma_i} \\ A_{i\Gamma_i}^T & A_{\Gamma_i}^{(i)} \end{pmatrix} . \tag{4}$$

In a parallel (multi-processor) computing environment, each subdomain matrix $A^{(i)}$ is assigned to one processor. In this way, the local Schur complement matrices $S^{(i)}$ can be computed simultaneously. The (global) Schur complement matrix S is available through Equation (3) and is never assembled explicitly.

Implementation

The implementation of substructuring methods usually consists of three steps. They are summarized in Algorithm 1.

Algorithm 1 : Substructuring algorithm	
Step 1 :	Factorize the matrix A_{II} and compute the Schur complement matrix S.
Step 2 :	Solve the Schur complement system $S u_B = f_B - A_{IB}^T A_{II}^{-1} f_I$, for u_B.
Step 3 :	Solve the system $A_{II} u_I = f_I - A_{IB} u_B$, for u_I.

Since the matrix A_{II} is a block diagonal matrix, steps 1 and 3 each consist of N independent smaller steps (one for each subdomain). Furthermore, since each subdomain matrix $A^{(i)}$ is assigned to a different processor, these N steps can be performed in parallel. When the number of subdomains increases, the amount of parallelism naturally increases in steps 1 and 3 (but the total amount of work decreases). The solution of the Schur complement problem in step 2 becomes more complex when the number of subdomains gets larger; the Schur complement matrix S is not assembled and therefore efficient parallel direct or iterative schemes are required for step 2.

Direct substructuring

The three steps of Algorithm 1 are implemented as follows.

Computation of the local Schur complement matrices

Since the discretization of the local problem is a sparse matrix, a direct (factorization) approach can be used in step 1 to compute the local Schur complement matrices $S^{(i)}$ explicitly. In fact, computing $S^{(i)}$ for subdomain Ω_i is equivalent to a partial Cholesky factorization of the matrix $A^{(i)}$ of Equation (4). In our implementation, we use the parallel multifrontal sparse direct solver MUMPS [ADLK01] that can compute a Cholesky factorization of the sparse submatrix A_{ii} and return the Schur complement matrix $S^{(i)}$. As mentioned before, the local Schur complement matrices are computed simultaneously (one on each processor). Therefore, we use one instance of the MUMPS solver on each processor.

Solution of the interface problem

The global Schur complement matrix S is the sum of the local Schur complements matrices $S^{(i)}$, see Equation (3). After computing the matrices $S^{(i)}$ explicitly, the matrix S is available as a sparse matrix that is distributed over the processors. MUMPS can accept such matrices in distributed form and can therefore be used to solve the interface problem directly and in parallel to obtain vector u_B. For this step, we use only one instance of MUMPS over all processors.

Solution of the interior problems

Step 3 consists of the solution of N independent systems $A_{ii}u_i = f_i - A_{i\Gamma_i}R_i u_B$ where $R_i u_B$ is the restriction of the interface solution vector u_B to interface Γ_i and f_i is the restriction of f_I to the internal variables of subdomain Ω_I. The systems can be solved by using the factorizations of the matrices A_{ii} that were computed in step 1.

Iterative substructuring

Iterative substructuring differs from direct substructuring in the solution of the interface problem (step 2 of Algorithm 1). Instead of a parallel Cholesky factorization, preconditioned conjugate gradient (PCG) iterations are used. Two important components of the preconditioned iteration are the matrix-vector product and the construction (and use) of the preconditioner.

Matrix-vector product

As the local Schur complements have been computed explicitly in the first step of the algorithm, it is straightforward to implement the matrix-vector product with the global Schur complement system.

For 2D problems, this implementation is more efficient than a classical implicit formulation where the local Schur complements are not known explicitly and where step 1 of Algorithm 2

Algorithm 2 : Computation of $y \rightarrow Sx$

Step 1 : $y_i = S^{(i)} R_i x$

Step 2 : $y = \sum_{i=1}^{N} R_i^T y_i$

requires a forward and back substitution and three sparse matrix-vector products (see Equation (3)). We display in Table 1 a comparison between the explicit and the implicit version of the Schur matrix-vector products for subdomains with 16000 unknowns each. For this example, the computation of the explicit local Schur complement is twice as expensive as the factorization of the local internal problem, but the subsequent gain for each Krylov iteration is very important (without preconditioning, the matrix-vector product remains the most expensive part of a Krylov iteration). For 2D problems, the extra storage cost for the local Schur complement is not prohibitive.

	implicit	explicit
Numerical factorization	14.2	27.2
Matrix-vector product	2.2	0.1
Time for factorization + 20 products	58.2	29.2

Table 1: Time comparison between implicit and explicit Schur matrix-vector product for subdomains of 16000 unknowns and with interfaces of size 1600. The computation has been performed on an SGI Origin 2000.

Preconditioners for the Schur complement

The rate of convergence of the conjugate gradient method depends on the condition number of the matrix of the linear system. In this section, we present two preconditioners for the Schur complement system.

Balanced Neumann-Neumann preconditioner

This two-level preconditioner was first introduced in [Man93]. It can be formulated as

$$M^{-1}S = P + (I - P)M_1 S(I - P), \quad \text{and} \quad M_1 = \sum_{i=1}^{N} R_i^T D_i^T (S^{(i)})^{-1} D_i R_i.$$

Here, M_1 is the one-level Neumann-Neumann preconditioner that was originally proposed in [DRLT91]. The weight matrices D_i form a decomposition of unity. P denotes the S-orthogonal projection onto the coarse space defined by $\{\sum_{i=1}^{N} R_i^T D_i^T Z_i v_i\}$ for some local subspaces Z_i and arbitrary vectors v_i.
In the experiments for this paper, we used a software package developed by Parallab (University of Bergen). One of the options that is available in that package is to construct the coarse space from local subspaces Z_i that are spanned by the eigenvectors associated with selected (small) eigenvalues of the local Schur complement matrices $S^{(i)}$. In particular, each subspace Z_i must contain the null space of $S^{(i)}$. We refer to [BKK00] for more details on this preconditioner.

Assembled Schur preconditioner

We consider another preconditioning technique for the solution of the Schur complement system. Let $\overline{S^{(i)}}$ be the local assembled Schur complement associated with a subdomain Ω_i. The assembled Schur preconditioner [CGM01] is defined by

$$\sum_{i=1}^{n} R_i^T (\overline{S^{(i)}})^{-1} R_i.$$

$\overline{S^{(i)}}$ corresponds to the restriction of the global Schur matrix to the boundary $\partial\Omega_i$ of the subdomain Ω_i and can be algebraically written as $\overline{S^{(i)}} = R_i S R_i^T$. A second level can be added to this preconditioner to define

$$M^{-1} = \sum_{i=1}^{n} R_i^T (\overline{S^{(i)}})^{-1} R_i + R_0^T (R_0 S R_0^T)^{-1} R_0,$$

where R_0 is a restriction operator from the global interface Γ onto a coarse space V_0. The second term of the equation defines a second level for the preconditioner. In our experiments, we consider a coarse space where we have one degree of freedom per edge of the decomposition, that is one degree of freedom per interface between two subdomains [CGT01].

Numerical experiments

We have performed experiments on various meshes, but here we only report results on a problem with 154892 unknowns. The size of the interface Γ varies from 440 unknowns for the 4-subdomain decomposition to 2446 unknowns for the 32-subdomain decomposition. The size of this problem is modest but it is representative for many problems that are solved in device modeling. Indeed, due to the complexity of the underlying physical problem, the complete code is memory consuming. We used a 32-processor SGI Origin 2000 for our experiments.

Iterative substructuring methods

In this section, we use the following notation:

1. AS : Assembled Schur preconditioner without coarse space.
2. NN : Neumann-Neumann preconditioner without coarse space.
3. AS-edge : Assembled Schur preconditioner with a coarse space defined by one degree of freedom per interface of the decomposition.
4. BNN1 : Balanced Neumann-Neumann preconditioner with a coarse space with one degree of freedom per subdomain.
5. BNN3 : Balanced Neumann-Neumann preconditioner with a coarse space with three degrees of freedom per subdomain.

For the sake of comparison between direct and iterative substructuring approaches, we stop the preconditioned conjugate gradient iterations only when the 2-norm of the residual of the current iterate normalized by the 2-norm of the right hand side is less than 10^{-15}. Note that such accuracy is often obtained by direct methods, without iterative refinement.
The condition number of the preconditioned Schur complement matrix varies from a few tens to a few hundreds. It is obtained from the eigenvalues of the tridiagonal matrix of the PCG coefficients although this may not be very reliable when the number of iterations is small.
We display in Table 2 the average number of conjugate gradient iterations that are needed to solve a linear problem during nonlinear solution of the equilibrium problem. The table shows

	Preconditioner					
No. of subdomains	none	AS	NN	AS-edge	BNN1	BNN3
4	67	4	15	6	15	15
8	87	20	22	23	21	20
16	117	31	29	33	27	25
32	125	37	33	40	32	29

Table 2: Average number of conjugate gradient iterations per linear system. 'none' denotes that no preconditioner is used. 19 linear systems were solved during the simulation.

that all the preconditioners improve the convergence of the conjugate gradient iteration. We observe that one-level preconditioner AS is more efficient than NN on a small number of subdomains. NN becomes more efficient when the number of subdomains increases. For all methods, the number of iterations of the preconditioned conjugate gradient increases moderately when the number of subdomains increases. The use of two-level preconditioners does not eliminate this increase. The AS-edge preconditioner performs worse than the AS preconditioner on this example.

Time measurements

In this section, we compare iterative substructuring with direct substructuring from an elapsed time point of view. We also make a comparison with MUMPS used as a black-box parallel sparse direct solver on the complete original problem, where the stiffness matrices are considered as in a distributed format (a feature of the MUMPS software).
In Table 3, we display the elapsed times to solve the complete equilibrium problem, by using direct substructuring (DSS), MUMPS as a black box direct parallel solver (MS) or iterative substructuring with preconditioned conjugate gradient (using AS, NN or BNN3),

	Algorithm				
no. of subdomains	AS	NN	BNN3	DSS	MS
4	49.65	49.50	53.01	57.47	63.97
8	26.11	31.04	28.53	31.59	44.24
16	14.79	16.74	17.89	22.81	43.66
32	10.65	11.87	16.31	16.98	44.07

Table 3: Times (in s) for solving the equilibrium problem (19 linear systems).

Table 3 shows that the substructuring algorithms are more efficient than the multifrontal parallel solver that does not scale very well because of the relative modest size of the linear

systems. We also observe that iterative substructuring is slightly more efficient than direct substructuring. Iterative substructuring will perform even better when we use a relaxed accuracy of the iterative linear solvers, while maintaining the same nonlinear path. However, this behaviour might not be true for other equations and to make a fair comparison, we required the same accuracy for all the linear solvers.

Table 3 shows that the one-level algorithms (AS and NN) perform better than the two-level algorithms, even though the latter require slightly fewer iterations for larger number of subdomains. This shows that a smaller number of iterations is not always an indication for better overall efficiency of a preconditioned Krylov solver.

Conclusion

We compared some efficient parallel linear solvers based on direct or iterative substructuring methods that enable us to solve problems that cannot be tackled on a single processor computer. For a set of linear systems that arise in semiconductor device modeling, we have shown that the two-level Balanced Neumann-Neumann preconditioner converges faster, but is less efficient in elapsed time, than the one-level preconditioners (Neumann-Neumann and Assembled Schur). The overall efficiency of the iterative substructuring approaches for these 2D simulations is due to the multifrontal direct sparse solver MUMPS that is for example able to compute the local Schur complement matrices efficiently.

Finally, we mention that domain decomposition techniques to parallelize the solution of the unsymmetric systems arising from the solution of the continuity equations associated with the electrons and the holes concentrations in the domain is still an ongoing work.

References

[ADLK01]Patrick R. Amestoy, Iain S. Duff, Jean-Yves L'Excellent, and Jacko Koster. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM J. Matrix Analysis and Applications*, 23(1):15–41, 2001.

[BKK00]Petter E. Bjørstad, Jacko Koster, and Piotr Krzyżanowski. Domain decomposition solvers for large scale industrial finite element problems. In *PARA2000 Workshop on Applied Parallel Computing*. Lecture Notes in Computer Science 1947, Springer-Verlag, 2000.

[CGM01]Luiz M. Carvalho, Luc Giraud, and Gérard Meurant. Local preconditioners for two-level non-overlapping domain decomposition methods. *Numerical Linear Algebra with Applications*, 8(4):207–227, 2001.

[CGT01]Luiz M. Carvalho, Luc Giraud, and Patrick Le Tallec. Algebraic two-level preconditioners for the schur complement method. *SIAM J. Scientific Computing*, 22(6):1987–2005, 2001.

[DRLT91]Yann-Hervé De Roeck and Patrick Le Tallec. Analysis and test of a local domain decomposition preconditioner. In Roland Glowinski, Yuri Kuznetsov, Gérard Meurant, Jacques Périaux, and Olof Widlund, editors, *Fourth International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 112–128. SIAM, Philadelphia, PA, 1991.

[HM94]Frédéric Hecht and Americo Marrocco. Mixed finite element simulation of hetero-

junction structures including a boundary layer model for the quasi-fermi levels. *COMPEL*, 13(4):757–770, 1994.

[Man93]Jan Mandel. Balancing domain decomposition. *Comm. Numer. Meth. Engrg.*, 9:233–241, 1993.