Domain Decomposition Methods 7
October 1993

Parallel Domain Decomposition Applied to Coupled Transport Equations

PETTER E. BJØRSTAD, W. M. COUGHRAN, JR. AND

ERIC GROSSE

ABSTRACT. Modeling semiconductor devices is an important technological problem. The traditional approach solves coupled advection-diffusion carrier-transport equations in two and three spatial dimensions on either high-end scientific workstations or traditional vector supercomputers. The equations need specialized discretizations as well as nonlinear and linear iterative methods. We will describe some of these techniques and our preliminary experience with coarse-grained domain-decomposition techniques applied on a collection of high-performance workstations connected at a 100Mb/s shared network.

1. Introduction

Simulation is widely used in the semiconductor industry to explore new manufacturing techniques, to characterize novel device technologies, and to validate and characterize circuit designs.

The transport physics of carriers in semiconductor devices has been studied for many years. Most of the existing model equations are derived from the Boltzmann transport equation (BTE) although the ideal set of equations would include Maxwell's and Schrödinger's equations [**10**]. Usually, Maxwell's equations are replaced by a Poisson equation via an assumption of locally constant dielectric behavior. Schrödinger's equation is obviated by making assumptions about the electronic states and band structure of the semiconductor. A simple model, called drift-diffusion, is effective for silicon devices down to 0.5μm or even

1991 *Mathematics Subject Classification.* Primary 65M55, 65N55; Secondary 68M10, 68N25.

This work was done while the first author was visiting the Department of Computer Science at Stanford Univ. and RIACS. Support from NFR is also acknowledged.

This paper is in final form and will not be published elsewhere.

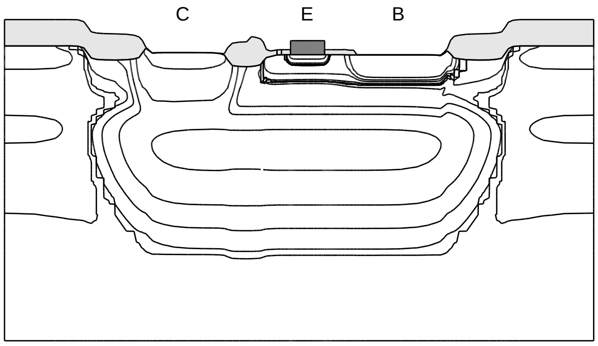

FIGURE 1. This is a two-dimensional cross-section of a three-dimensional 0.8μm BiCMOS bipolar transistor [**13**]. The contour lines represent impurity concentrations. The shaded gray areas are silicon oxide. The darker shaded region is a polysilicon wire. C, E, and B represent the collector, emitter, and base, respectively.

0.25μm feature sizes. It requires the solution of a Poisson equation coupled to two, formally parabolic, advection-diffusion equations

$$-\nabla \cdot (\epsilon \nabla \psi) = -q(n - p - N), \tag{1.1}$$

$$\frac{\partial n}{\partial t} = \nabla \cdot J_n - R, \tag{1.2}$$

$$\frac{\partial p}{\partial t} = -\nabla \cdot J_p - R, \tag{1.3}$$

where the current densities are

$$J_n = -\mu_n n \nabla \psi + D_n \nabla n, \tag{1.4}$$

$$J_p = -\mu_p p \nabla \psi - D_p \nabla p. \tag{1.5}$$

In silicon oxide (insulating) regions, these coupled equations are replaced by $-\nabla \cdot (\epsilon_0 \nabla \psi) = 0$. Here

- ϵ and ϵ_0 are dielectric constants;
- q is the magnitude of an elementary charge;
- $x \in \mathsf{R}^d$ and $t \in \mathsf{R}$ represent space and time, respectively;
- $N(x)$ is the impurity or doping concentration;
- μ_n, μ_p are carrier mobilities; D_n, D_p are diffusion coefficients;
- $R(n, p)$ represents electron-hole recombination-generation;
- $\psi(x, t)$ is the electrostatic potential;
- $n(x, t)$ and $p(x, t)$ are the electron and hole concentrations, respectively.

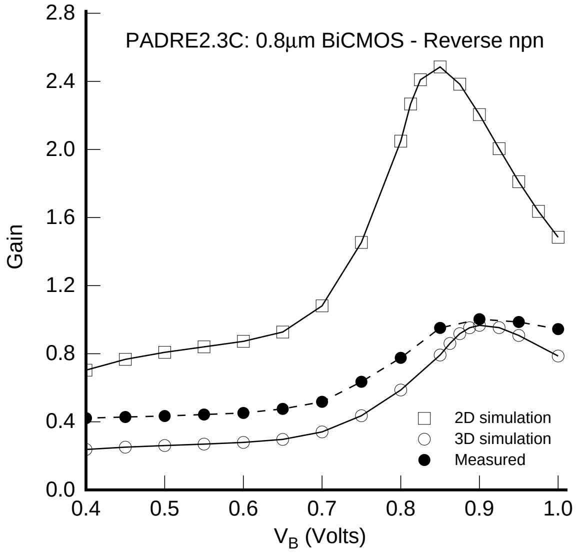

FIGURE 2. This represents the results of two- and three-dimensional simulations as well as measurements [**13**].

Geometry of the device structure and material interfaces are important; Figure 1 shows a cross-section of a bipolar transistor. Boundary conditions are commonly applied voltages at contacts along the perimeter of the device specified via ψ. Variables n and p range over several orders of magnitude while ψ varies less. The recombination-generation term, R, couples n and p. For power devices, it is sometimes necessary to add a coupled heat equation to model substrate heating.

Since the mid-1960s, numerous algorithms have been developed to solve the drift-diffusion equations. Specialized exponential upwinding schemes were invented for one-dimensional problems. These were generalized to tensor-product grids in two and three dimensions. The discretizations were also adapted to general finite volumes and elements. There are significant nonlinearities in this model so considerable research on Newton-like and nonlinear Gauss-Seidel methods has been done. Static grid adaption and continuation techniques have been used to explore devices with complex responses. Specialized sparse direct and iterative methods have been developed as well.

The state-of-the-art is such that complex two-dimensional problems are solved routinely. Sparse direct methods with triangular grids are effective and efficient for two-dimensional problems on conventional supercomputer architectures, like a Cray Y-MP. A snapshot of the state-of-the-art in 1990 is given by [**11**]. For drift-diffusion modeling, three-dimensional simulation is becoming more important (see Figure 2). However, grid generation and adaption as well as linear iterative methods are still inadequate for routine simulation.

There are more advanced partial-differential equation (PDE) models for semiconductor devices. The so-called energy-balance or energy-transport equations are similar to the drift-diffusion equations in mathematical character. Additional equations for carrier energy variables have to be solved. The "hydrodynamic" model involves formally hyperbolic equations, which call for alternate algorithmic approaches. Our motivation comes from systems of a form analogous to the drift-diffusion or energy-transport models, rather than the hydrodynamic model.

Over the years, AT&T Bell Laboratories has relied on a series of Cray Research computers to do its large-scale simulation. We currently employ a Cray Y-MP M92/256 as a production workhorse. Our evaluations of parallel computer systems have been mixed, particularly since the specialized MIMD architectures carry a significant development and communication-infrastructure overhead. However, high-performance microprocessors in the form of workstations are changing things. We embarked on a prototype PDE solver designed to run on workstations connected via a "high-speed" network interface. Our hope is that such an approach will lead to cost-effective computing for two- and especially three-dimensional simulation. From our brief discussion of device modeling, it is apparent that the ability to solve general PDEs on complex geometries is crucial.

2. Domain decomposition approach

We are developing a collection of C codes to deal with general elliptic and advection-diffusions PDEs on composite grids made up of simplices, prisms, pyramids, and bricks. Each of the following has to be dealt with:

- geometry specification;
- grid generation;
- temporal discretizations;
- spatial discretizations, including a variety of upwinding techniques suitable for advection-diffusion equations;
- Newton-like and other iterative methods for dealing with the associated nonlinear systems;
- continuation methods for characterization studies;
- automatic differentiation techniques for linearization;
- sparse direct and iterative methods for the associated nonsymmetric linear systems of equations.

Each of these subproblems is significant by itself, but all of them must be dealt with to produce a generically useful simulation toolbox. Too often researchers allow themselves to use simplified approaches only suitable for model problems on trivial geometries.

For two-dimensional, single device, simulation problems with well-adapted grids (typically less than 10^4 unknowns), it is difficult to beat sparse direct factorization methods for solving the linear systems. However, numerous possibilities exist for iterative methods. Domain decomposition appears to us as a natural means to exploit coarse-grained parallelism.

"Domain decomposition" refers to any method that divides the original problem domain into pieces and solves locally on each subdomain [**12**]. The particular variant that we use, multiplicative overlapping Schwarz [**4, 6**], defines subdomains with overlap and solves local problems in succession, propagating data from the solution on the interior of one subdomain to the boundary values of the next subdomain. To allow information to propagate more rapidly, a coarse grid solution is also used. Finally, rather than cycling through all the subdomains repeatedly, as in a pure alternating Schwarz method, we use one sweep as a preconditioner for an iterative linear system solver.

To be more explicit, suppose we wish to solve $Lu = f$ on Ω, $u = b$ on $\partial\Omega$, which we discretize as $Au = Rf - Bb$ using a finite volume or "box" approach. Without loss of generality, we will call the modified right hand side in this system f and assume $b = 0$. The domain Ω is subdivided into k overlapping subdomains Ω_i with local operators that are restrictions of the global ones in the sense that at an interior point j of the global grid corresponding to interior point ι in subdomain i, we have $[f - Au]_j = [f_i - A_i u_i - B_i w_i]_\iota$. Here u_i means the local interior unknowns and w_i means the local boundary values, which are 0 or u values from a neighboring subdomain.

Define a preconditioner $v = M^{-1}g$ for solving $Av = g$ with a Krylov space method by the steps

(2.1) $\quad g_c = I_f^c g \quad$ *(restrict to coarse grid)*

(2.2) $\quad A_c v_c = g_c \quad$ on Ω

(2.3) $\quad v_f = I_c^f v_c \quad$ *(prolongate to fine grid)*

(2.4) $\quad$ from v_f send values into w_i

(2.5) $\quad$ for $\quad 1 \le i \le k$

(2.6) $\quad\quad A_k v_k = g_k - B_k w_k \quad$ *(local solve on Ω_k)*

(2.7) $\quad\quad$ from v_k send values to w_j for neighboring Ω_j

The final v is obtained by assembling values from the subdomains.

The multiplicative Schwarz method is attractive to us because we want a good serial code, and in our workstation cluster environment we seek only coarse parallelism. In contrast, the additive Schwarz variant provides more parallelism at the cost of roughly twice as many iterations [**2, 3**]. So, while other computer configurations would call for other methods, for the next several years we see workstation networks as the best price/performance choice.

Although in theory one must increase the number of rows of overlap between subdomains as the grid is refined in order to keep the same rate of convergence, previous experience has shown that this is balanced by the increased work [**7, 14**]. Following those earlier conclusions, we simply take one row of unknowns for the overlap.

3. Communication structure

As mentioned in the previous section, our interest in this project has been to explore whether an inexpensive network of workstations is capable of the VLSI simulation modeling that had previously required Cray supercomputer resources. Can one get by without shared memory or specialized communication fabric? The cautionary words of Baskett and Hennessy [13 Aug 1993, Science, p.868] make this seem unlikely.

> "One potentially attractive approach to building parallel processors is to use workstations connected on a local area network, often called a workstation cluster. This approach, however, has proved suitable only for applications where the parallel computations are so coarse-grained as to be essentially independent."

Surely the coupling between neighboring subdomains keeps the subproblems from being "essentially independent"?

Our laboratory was purchasing Silicon Graphics Indigo 50MHz MIPS R4000 workstations connected by a Fiber Distributed Data Interface (FDDI) network for a multimedia experiment; we decided to see if we could put them to use solving differential equations when they weren't otherwise needed. FDDI is nominally a 12 megabyte per second (MB/s) ring with a maximum length of 100km using a timed token-passing protocol, in which each packet goes all the way around the ring. The SGI implementation we use has been found to be reasonably easy to administer, and we consistently measure 7MB/s or better for large transfers (user space on one machine to user space on another). In contrived circumstances in which few pages of memory have to be touched, we measure as much as 11MB/s. The measured latency of about 1ms is far higher than the intrinsic hardware latency of 10us around the ring, but as we shall see is sufficient for domain decomposition.

We expect the FDDI network to be replaced in the next year or so by Asynchronous Transfer Mode (ATM), but are happy with "dull but deliverable" FDDI for now. The number of workstations we expect to be cooperatively solving one problem is limited more by sociological and economic constraints than by scaling limits of the network. Also, while the transfer rates needed by domain decomposition are high compared to the typical loaded campus Ethernet, the traffic is rather bursty and can be accommodated adequately on a shared medium. This is in contrast to the demands of the same set of workstations if they are transmitting video signals to each other.

One specific disappointment with the hardware as seen at the user level is that, although each packet transits each workstation, there appeared to be no feasible way to use this for data broadcast. We were using TCP as the transport protocol, not wishing to bear the burden of adding reliability onto UDP [**5**]. A sufficiently sophisticated communications package might do hardware broadcast, but as far as we could tell none of the popular implementations is yet up to this.

Certainly the one we tried, **pvm** [9], was far from using the full FDDI bandwidth even for direct process-to-process communication.

The communication needs of domain decomposition turn out to be so simple that a trivial communication package built directly on sockets and read/write is sufficiently expressive and efficient. One "coarse-grid process" formulates, computes, and distributes the coarse grid solution; multiple "fine-grid processes" talk to the coarse-grid process and to fine-grid processes for neighboring domains. All processes execute the same program and hold the coarse grid and subdomain topology, but only store and compute fine grid values for the subdomains "owned" by that process. This leads to send/receive pairs that are close together in the source code, with a static matching known at compile time, so correct sequencing and typing of messages is immediate.

There are three classes of communication: dot products for the outer Krylov iteration, neighbor communication between fine-grid processes, and communication between fine-grid processes and the coarse-grid process.

Our implementation of BICGSTAB [15] is shown in Figure 3. In order to bring to the surface the essential global synchronization points, we have written this in a rather unusual style. The function `B` takes as its first argument a string representing in readable form an operation that can be completed with, at most, one communication. Our C statement

```
omega = B("dot{r*MAr}/dot{MAr*MAr}");
```

would ordinarily be written as

```
omega = dot(r,MAr)/dot(MAr,MAr);
```

an unusually clever communications package might be able to batch the two sum operations together, but probably not without further hints. Alternatively, one might write the vector primitives to use lazy evaluation, but the reader has to trust that sophisticated operations behind the scenes are doing what is intended. Our coding shows the reader immediately that one communication is involved, without any need to understand and debug scheduling policies.

For neighbor communication, each process loops through all the subdomains in order. If a process "owns" a subdomain, then it performs a local solve, using **y12m** [16] for the experiments reported below, and sends boundary data to its neighbor processes; if instead its neighbor owns the subdomain, then the process waits to receive data from its neighbor; otherwise, the process examines the next subdomain. This arrangement is self-synchronizing, with no explicit task management. Subdomains are colored by a greedy algorithm and different color subdomains assigned statically to separate processes. If four colors are needed, four processes are assigned to each physical processor. Obviously, we attempt to place neighbors on the same processor when possible. Assuming the work can be predicted well enough, and partitioned evenly enough, this static assignment has given good overlap of communication and computation, as will be seen in the next section.

```
#include "subdom.h"
void bicgstab( void (*A)(int,int,void*), void *Auser,
void (*M)(int,int,void*), void *Muser,
int (*done)(int,double,Chk*),Chk *conuser){
  int n;
  double rho, rho_old, omega, alpha, beta, sigma;
  B("set userdata",Auser);
  A(Gu,Gw,Auser);
  B("w=f-w");
  M(Gw,Gr,Muser);
  B("r0=p=r");
  rho = B("dot{r0*r}");
  for(n = 1; n<1000; n++){
     if(n>1){
      rho_old = rho;
      rho = B("dot{r0*r}");
      beta = (rho/rho_old) * (alpha/omega);
      B("p=beta*(p-omega*MAp)+r",beta,omega);
     }
     A(Gp,Gw,Auser);  M(Gw,GMAp,Muser);
     alpha = rho/B("dot{r0*MAp}");
     B("r -= alpha*MAp",alpha);
     A(Gr,Gw,Auser);  M(Gw,GMAr,Muser);
     omega = B("dot{r*MAr}/dot{MAr*MAr}");
     sigma = B("u+=alpha*p+omega*r; r-=omega*MAr; norm2{r}",
              alpha,omega);
     if(done(Amults,sigma,conuser)) break;
  }
}
```

FIGURE 3. An implementation of BICGSTAB in C, using ad hoc vector primitives ("BAHS") rather than BLAS in order to reveal communication.

The amount of overlap is known at setup time, so there would be no trouble allocating fixed buffers. The local solves are expensive enough that one or two buffers provide sufficient queuing.

The neighbor communication, though it involves scatter/gather references to memory, deals with fairly small amounts of data. Communication between the coarse grid process and the fine grid processes for the exchange of coarse grid data in prolongation and restriction is just the opposite: bulky data in contiguous memory. In the experiments described below, we transmitted the entire coarse grid to each fine grid process, to avoid the complication of determining exactly which coarse grid points were really needed.

It may be worth mentioning the size of the communication code, to dispel the notion that dealing with bare sockets is an overwhelming burden. About 200 lines of C code are involved in providing network connection primitives; the domain decomposition program proper has about another 300 lines to set up the connections in the right order and to issue sends and receives. All this is dwarfed by the grid and discretization modules.

4. Results

Consider a model system of two equations in two space dimensions, derived from the linearized drift-diffusion equations with one carrier

$$\Delta\psi + n = -r \tag{4.1}$$

$$n_o\Delta\psi + \nabla\psi_0\nabla n - \Delta n = 0. \tag{4.2}$$

An appropriate upwinding scheme was used [**1**]. This captures some aspects of the coupled transport without the full complexity. For convenient benchmarking, we discretize on a square domain Ω divided into an 8 by 8 mesh of subdomains Ω_i. The typical subdomain has 7 by 7 interior grid points, though subdomains near $\partial\Omega$ have fewer grid points; we start with a uniform subdivision of Ω and define Ω_i by "growing" each cell. Neighbors overlap by one mesh line of unknowns. This yields a total of 4418 fine unknowns. Finally, we overlay a coarse grid with 7 by 7 interior points, for a total of 98 coarse grid unknowns.

Using adjoint as the restriction operator and BICGSTAB as the accelerator, we measured the wall clock time of program execution on a relatively unloaded network of uniprocessor workstations, taking the best of three runs. The timings here are for an experimental code compiled with `-g`, but we did not observe any dramatic changes with `-O`. All workstations had enough memory to avoid swapping.

A serial version of the program ran in 38.12 seconds elapsed, 37.7 cpu. With two workstations (and hence one coarse grid process and eight fine grid processes) our socket-based parallel code ran in 23.6 seconds elapsed time, for a speedup of 1.6. The corresponding time for a `pvm`-based version of the code was 34.8 seconds, for a speedup of 1.1. With 4 workstations, our code ran in 13.2 seconds, for a

TABLE 1. Convergence rate for model problem.

Amults	$\log_{10} \|\|u - u_*\|\|_2$	$\log_{10} \|\|M(f - Au)\|\|_2$
3	1.1	-0.4
5	-2.7	-2.9
7	-2.7	-2.9
9	-4.4	-4.4
11	-5.6	-5.6
13	-7.2	-7.2
15	-8.8	-8.9
17	-11.1	-11.1

speedup of 2.9. Because of a bug we were unable to isolate, our `pvm`-based version hung after initialization in the four workstation (17 processes) case.

Although we are using relatively small subdomains in this experiment, because we are solving a system of differential equations and use a complicated discretization, there is enough computation to mask most of the communication delay. For a simple Poisson equation that would ordinarily be used in primitive benchmarks like this, the situation is reversed. On the same grid described above but with one equation (49 coarse and 2209 fine unknowns) with the serial code we measure 4.5 seconds elapsed; on two workstations we measure 4.0 seconds (speedup 1.13) with the socket-based code, 8.2 seconds (speedup 0.6) with the `pvm`-based code; on four workstations, the socket-based code we measure 2.4 seconds (speedup 1.9).

Another way in which these timings are conservative is that for this model, the cost of matrix setup is rather low. In real codes, it may account for half of the total execution time, and fortunately scales linearly with processors.

Exactly the same computations are being done in the serial and parallel codes, and as Table 1 shows the program is converging satisfactorily. Total cpu times for the 4 workstations were only a few seconds more than the serial cpu time.

Because the domain decomposition preconditioner is so effective, the preconditioned residual $||M(f - Au)||_2$ is an excellent estimate for the true error $||u - u_*||_2$, as is apparent from Table 1.

For a more detailed look at the degree of overlap of communication and computation, we profiled the program by inserting `gettimeofday` calls before and after each read and write and recorded the size of the message. Elapsed time increased by factor 1.075. For a small run on two workstations lasting 6 seconds, there were 6112 messages transmitting 372 kilobytes, for an average of 7.6 doubles per message. A schematic drawing of when each process was busy confirmed our expectations of how processes would be naturally scheduled by the operating system, and we did not feel the need to tune this further by using threads instead of separate processes on each processor.

To help the reader avoid one blind alley, we give in Table 2 a cautionary set

TABLE 2. Number of iterations for different restriction operators and Krylov methods on a simple Poisson equation on a square.

BICGSTAB				TFQMR			
n_{fine}	fine	adjoint	interp	n_{fine}	fine	adjoint	interp
25	7	5	5	25	6	4	5
81	11	5	7	81	11	4	6
289	15	5	7	289	17	4	6
1089	29	3	7	1089	34	4	6
289	9	7	7	289	9	6	8
841	11	5	13	841	14	5	13
2401	23	5	29	2401	23	4	22
9409	37	5	39	9409	44	5	39
25	7	7	7	25	7	6	6
49	9	7	9	49	9	6	8
225	15	5	17	225	16	5	15
961	27	5	25	961	30	4	31

of intermediate results from our testing. Iterations to reduce the residual by a fixed factor are given for 1) only fine grid solves; 2) coarse and fine grid with bilinear interpolation as prolongation and the adjoint as restriction; 3) bilinear interpolation for both prolongation and restriction. As expected, the number of iterations increases with n for case 1) and is nearly constant for case 2). The variational theory depends on using the adjoint as the restriction operator, but experience from multigrid and elementary considerations of approximation on unequally spaced grids led us to guess that interpolation as in 3) would be a good choice. The numerical evidence dashed this hope.

The experiments were all done on uniform fine grids on a square, with three slightly different ways of defining subdomains and slightly different coarse grids. The experiments were also repeated with another well-respected Krylov method, TFQMR [**8**]. There were no significant differences for this example between BICGSTAB and TFQMR.

We are pleased with the speedup of 2.9 on four workstations, and look forward to expanding to a dozen workstations, larger subdomain solves, and additional work in matrix setup and solution rendering that is known to scale linearly. We are satisfied with these speedups, and have plenty of other work for any spare cycles left over because of load imbalance. Happily, domain decomposition appears to be one of those "essentially independent parallel computations" that works well for a modest number of workstations on modern networks.

Acknowledgements. We have benefited greatly from collaborations with Randy Bank, Mark Pinto, Don Rose, and Kent Smith on algorithms for advection-diffusion equations. Claude Pommerell introduced us to ways of structuring communication for parallel iterative methods. Dave Presotto and Phil Winterbottom were welcome sources of advice on TCP. Linda Kaufman and Margaret Wright improved the presentation.

References

1. R. E. Bank, J. F. Bürgler, W. Fichtner, and R. K. Smith, *Some upwinding techniques for finite element approximations of convection-diffusion equations*, Numer. Math. **58** (1990), 185–202.
2. P. E. Bjørstad, R. Moe, and M. Skogen, *Parallel domain decomposition and iterative refinement algorithms*, (W. Hackbush, ed.), Vieweg Verlag, Braunschweig, 1991.
3. J. H. Bramble, J. E. Pasciak, J. Wang, and J. Xu, *Convergence estimates for product iterative methods with applications to domain decomposition*, Math. Comp. **57** (1991), 1–21.
4. X.-C. Cai and O. Widlund, *Multiplicative Schwarz algorithms for some nonsymmetric and indefinite problems*, SIAM J. Numer. Anal. **30** (1993), no. 4, 936–952.
5. D. E. Comer and D. L. Stevens, *Internetworking with TCP/IP, Vol. III: client-server programming and applications, BSD socket version*, Prentice-Hall, 1993.
6. M. Dryja, *An additive Schwarz algorithm for two- and three-dimensional finite element elliptic problems*, Domain Decomposition Methods (Tony Chan, Roland Glowinski, Jacques Périaux, and Olof Widlund, eds.), SIAM, 1989.
7. M. Dryja and O. B. Widlund, *Domain decomposition algorithms with small overlap*, SIAM J. Sci. Stat. Comput. **15** (1994).
8. R. W. Freund, *A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems*, SIAM J. Sci. Comput **14** (1993), 470–482.
9. A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, *PVM3 user's guide and reference manual*, Tech. Report ORNL/TM-12187, Oak Ridge National Laboratory, Oak Ridge TN 37831, 1993, *http://www.netlib.org/pvm3/ug.ps*.
10. K. Hess, *Advanced theory of semiconductor devices*, Prentice Hall, 1988.
11. K. Hess, J. P. Leburton, and U. Ravaioli, *Computational electronics: Semiconductor transport and device simulation*, Kluwer, 1991.
12. P. L. Lions, *On the Schwarz alternating method. I.*, First International Symposium on Domain Decomposition Methods for Partial Differential Equations (R. Glowinski, G. H. Golub, G. A. Meurant, and J. Périaux, eds.), SIAM, 1988.
13. M. R. Pinto, D. M. Boulin, C. S. Rafferty, R. K. Smith, W. M. Coughran, Jr., I. C. Kizilyalli, and M. J. Thoma, *Three-dimensional characterization of bipolar transistors in a submicron BiCMOS technology using integrated process and device simulation*, Int. Electr. Dev. Meeting Technical Digest '92, 1992, pp. 923–926.
14. M. D. Skogen, *Schwarz methods and parallelism*, Ph.D. thesis, Department of Informatics, University of Bergen, Norway, February 1992.
15. H. A. van der Vorst, *Bi-CGSTAB: a fast and smoothly converging variant of BI-CG for the solution of nonsymmetric linear systems*, SIAM J. on Scientific and Statistical Computing **13** (1992), 631–644.
16. Z. Zlatev, *Computational methods for general sparse matrices*, Kluwer, 1991.

Institutt for Informatikk, Universitetet i Bergen, N-5020 Bergen, Norway
E-mail address: Petter.Bjorstad@ii.uib.no

AT&T Bell Laboratories, Murray Hill, New Jersey 07974
E-mail address: wmc@research.att.com, ehg@research.att.com