PARALLEL SOLUTION OF ELLIPTIC BOUNDARY VALUE PROBLEMS

Babak Bagheri, Andrew Ilin, L. Ridgway Scott, and Dexuan Xie
The Texas Center for Advanced Molecular Computation
and the Department of Mathematics, University of Houston

Abstract

We describe the development of some parallel iterative techniques for solving boundary value problems for elliptic partial differential equations. Using domain decomposition techniques, we modify standard sequential iterative techniques to obtain effective parallel methods. We contrast implementations on distributed-memory and shared-memory scalable parallel processors. We describe the use of two different programming paradigms, one involving explicit parallelism in a distributed-memory model and the other utilizing simple loop decompositions in a shared-memory model. Our primary conclusion is that parallel computing on existing commercial parallel supercomputers makes it routine to do three-dimensional modeling of semiconductor devices using drift-diffusion models. The implications this has for the use of more realistic models of submicron devices using Boltzmann-type equations will be mentioned.

I. Introduction

We discuss several techniques for solving elliptic boundary value problems via iterative methods which have a high degree of parallelism. These techniques are being developed to solve as broad a class of problems as possible, but our primary motivation has come from computing the electrostatic potential around molecules of biological significance [8]. Moreover, implementation of the methods has been done as part of an existing code UHBD [5]. This makes the code development more complex but also provides an assessment more realistic than would be available by looking only at computational kernels. In addition, we have applied some of the computational techniques to solve prototypical problems related to semiconductor device simulation [3].

We have studied several variants of standard iterative methods which we have designed to have good parallelism. These include variants of the well known ICCG and SOR iterative methods. In addition, we have proposed new types of iterations especially suitable for parallel computation [12]. We anticipate that all of these methods will be useful as coarse grid solvers for parallel multigrid methods [9].

In addition to studying different parallel iterative methods, we have used different parallel programming paradigms. Two of these are (1) IPfortran [1] and (2) shared memory constructs supported by Kendall Square's KSR-1 Fortran [10]. Both approaches have proved adequate for implementing the parallel algorithms presented here, due to the high degree of regularity of the loops involved. Less regular loops in UHBD, related to its Brownian dynamics phase, have been easier to parallelize using shared-memory constructs [4].

II. PSOR

The Jacobi method for approximating the solution of a linear system is naturally parallel, but the typically more efficient Gauss-Seidel method is essentially sequential. In the Jacobi method, each component X_i of the approximate solution vector $X = (X_1, \ldots, X_N)$ can be computed separately of all others, which we can write schematically as

$$X_i^{k+1} = F_i(X_1^k, \ldots, X_N^k), \quad \text{for} \quad i = 1, 2, \ldots, N, \tag{2.1}$$

where the F_i are functions of N variables. For example, $F = (F_1, \ldots, F_N)$ is an affine function in the case of solving a linear system. Typically F is sparse, depending only on entries near the diagonal, which we indicate by $F_i(\ldots, X_{i-1}, X_i, X_{i+1}, \ldots)$. With Gauss-Seidel, it is frequently the case that X_{i+1}^k depends on X_i^k: schematically it is

$$X_i^{k+1} = F_i(\ldots, X_{i-1}^{k+1}, X_i^k, X_{i+1}^k, \ldots) \quad \text{for} \quad i = 1, 2, \ldots, N. \tag{2.2}$$

The same applies for the SOR method, which is just a relaxed (or accelerated) version of Gauss-Seidel.

One approach taken to deal with the sequential nature of SOR is to reorder the unknowns so that one group of components X_i can be computed independently of others. This is often referred to as a coloring of the index set. The most well known case is that of two colors, usually called "red-black" ordering since it is similar to a chess board in simple cases. While this can be quite effective, it requires communication to be done for each color as opposed to just once for each iteration, as is the case for the Conjugate Gradient (CG) method. The number of colors required depends on the extent of the sparcity of F.

A simple technique used in practice is to decompose the index domain (the set of indices i) in a way to minimize the communication (either the number of messages required, or the size) among neighboring domains. Gauss-Seidel (or SOR) is used within each domain, without updating using the appropriate neighboring values. In the two-processor case, it takes the form

$$\begin{aligned} X_i^{k+1} &= F_i(\ldots, X_{i-1}^{k+1}, X_i^k, X_{i+1}^k, \ldots) \quad \forall\, i,\ 1 \le i \le N/2, \\ X_i^{k+1} &= F_i(\ldots, X_{N/2}^k, X_{N/2+1}^{k+1}, \ldots, X_{i-1}^{k+1}, X_i^k, X_{i+1}^k, \ldots) \quad \forall\, i,\ \frac{N}{2} + 1 \le i \le N. \end{aligned} \tag{2.3}$$

Once the local Gauss-Seidel (or SOR) sweep is done, neighboring values are exchanged, similarly to what would be done in the Jacobi iteration. For this reason, we refer to this method as the Jacobi-Gauss-Seidel (JGS) algorithm (or JSOR for its accelerated or relaxed variant). While appealing for its simplicity, this algorithm frequently requires a much larger number of iterations than the sequential case.

Remarkably, a simple alternative [14] to JGS and JSOR has convergence properties similar to the sequential case, but with communication features similar to JGS/JSOR. We will not attempt a complete description of the most general case, but will simply describe an example and present

numerical results. Consider the following algorithm:

$$\begin{aligned} X_i^{k+1} &= F_i(\ldots, X_{i-1}^{k+1}, X_i^k, \ldots, X_{N/2}^k, X_{N/2+1}^{k+1}, \ldots) \quad \forall\, i,\ 1 \le i \le N/2, \\ X_i^{k+1} &= F_i(\ldots, X_{N/2}^k, X_{N/2+1}^{k+1}, \ldots, X_{i-1}^{k+1}, X_i^k, X_{i+1}^k, \ldots) \quad \forall\, i,\ \frac{N}{2}+1 \le i \le N. \end{aligned} \tag{2.4}$$

This algorithm, which we call PGS (and PSOR for its accelerated or relaxed variant) is parallel for sparse F to the extent that the values $X_{N/2}^{k+1}, \ldots, X_N^{k+1}$ which are produced by the processor computing the second line can be computed and made available to the processor computing the first line before they are needed. In the case that the functions F_i are suitably sparse, this constraint poses no practical limitation to parallelism.

Figure 2.1 shows performance analysis for calculations done with the 5-point discretization of Laplace's equation using a strip decomposition (algorithm (2.4) in the case of two processors). We use this type of performance analysis graph to isolate different parts of a code. The computation time decreases even superlinearly [4] whereas the communication time (due to the use of a strip decomposition) remains nearly constant. The category "other time" simply reflects the part of the total time that cannot be accounted for in either of these categories; in this case it is quite small (being less that a second for two and four processors).

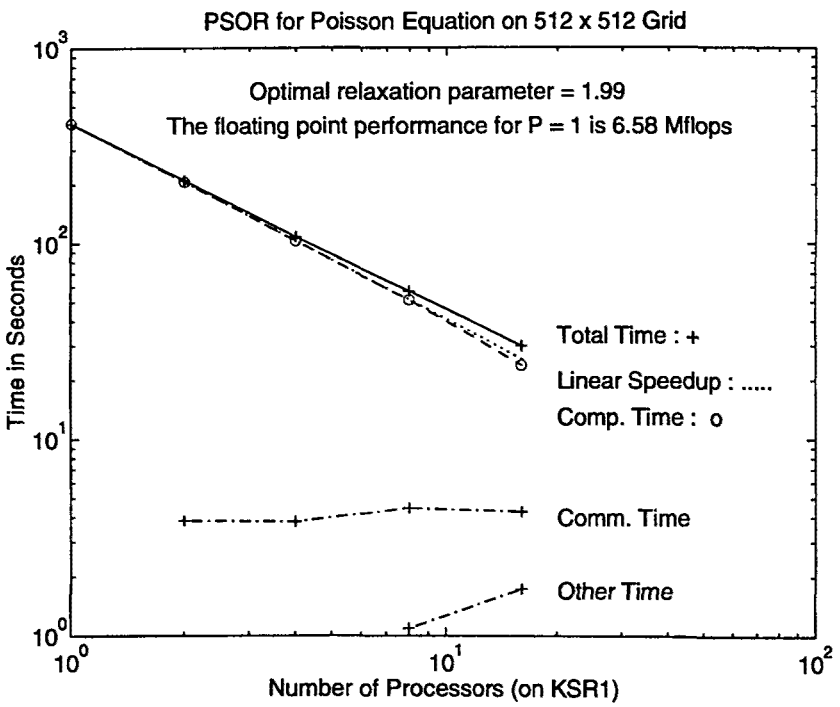

Figure 2.1. Performance analysis for PSOR for the 5-point discretization of Laplace's equation using a strip decomposition on the KSR-1.

We note that the code for this test was implemented in IPfortran and compiled separately for the Delta and KSR-1, without change of source code. The resulting speedup is almost identical for

both systems. In fact, the computation and communications times are largely the same for both systems. Although it is certainly possible to optimize performance for these distinct architectures, this shows that a single programming paradigm can provide efficient execution across a variety of different parallel architectures.

III. PICCG

Our parallel variants of ICCG have been implemented as part of the code UHBD [5,6] which was developed to study the interaction of two molecules of biological significance. One phase involves computing the electrostatic potential around the dominant molecule, and the second phase simulates Brownian motion of the second molecule in this electrostatic force field. The first phase solves the nonlinear Poisson-Boltzmann (NLPB) equation for the electrostatic potential.

We have modified the electrostatic solver to be able to model semiconductor devices [3]. This has provided a stronger test both of the linear and nonlinear parts of the solver, but the principal conclusion is that semiconductor devices can be modeled quite effectively on massively parallel computers. For example, the following table shows that the solver is scalable in the sense that larger problems can be solved without increasing the execution time, by increasing the number of processors used.

Total CPU time in seconds for a MOSFET simulation
on the Intel Delta for P nodes and mesh of size N^3

N^3	$P=1$	2	4	8	16	32	64	128	256
30^3	*22*	14	9	6	4	4	4	5	8
60^3	**192**	99	52	35	*20*	15	12	13	16
90^3			**214**	94	62	36	28	*26*	28
140^3					**184**	127	90	91	76
200^3							**252**	228	197
260^3									**440**

One particular case of interest is the so-called *memory constrained* scaling, the times for which are indicated in bold face. This is the case using the smallest number of processors which can run the problem, i.e., can fit the problem in local memory. We note nearly constant run times for this case. The slanted numbers indicate a different scaling which corresponds to a number of processors yielding an execution time that is an order of magnitude smaller. In this case, local memory is not utilized fully.

Most importantly, this table indicates that very large problems can be solved in just a few minutes (or just a few seconds, depending on resources available), allowing repeated designs to be tested or even optimized. We note also that the best decomposition has not been used for the case of large P and moderate N. If a block decomposition were used in this case, even better performance would be realized for the times away from the diagonal in the table.

One striking conclusion of our work so far [2, 3, 4] is that the total execution time for the elliptic solver portion of UHBD is essentially the same for quite disparate computer architectures and programming paradigms, as shown in Figure 3.1. The computations on each machine have quite distinct internal characteristics. For example, each calculation in done in each machine's single precision, which is 8-bytes on the KSR-1 and 4-bytes on the Delta. Due to the shorter word length, more iterations actually are done to reach the prescribed tolerance (the same for both machines).

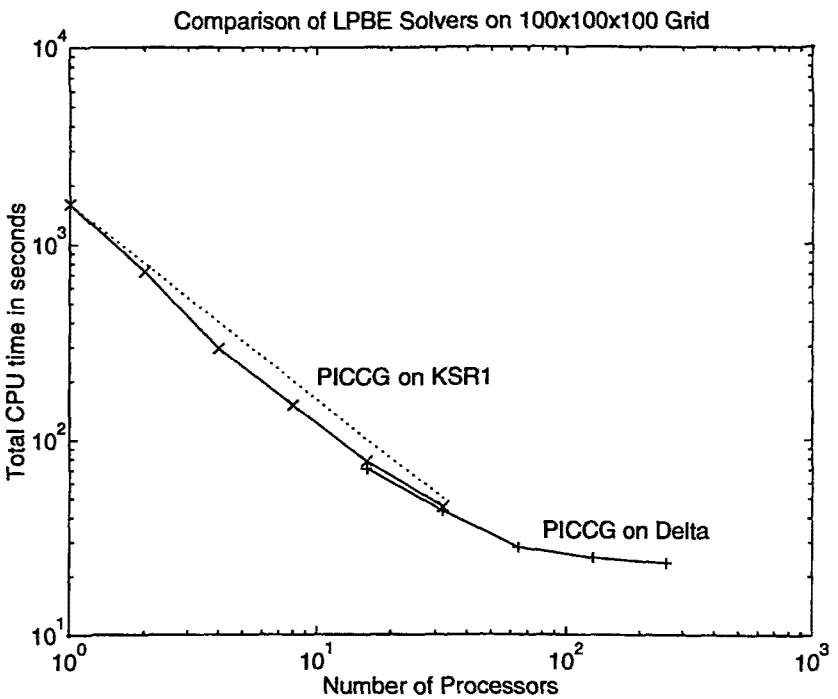

Figure 3.1. Timing for the linear and nonlinear solvers in UHBD on a test problem with a single atom.

In addition, quite different programming paradigms are being used in each case. For the Delta computations, we used IPfortran [1], an explicitly parallel language. For the KSR computations, we used the KSR "tiling" directives [10]. However, the total time is almost identical for 16 and 32 processors for a uniform mesh of size 100^3.

IV. Conclusions and future work

We view the current state of affairs in our work as incomplete. We have identified a number of promising parallel iterative methods, but we have not yet begun to quantify their domains of applicability (and superiority). Moreover, we anticipate these will ultimately find their best application as coarse grid solvers in a parallel multigrid technique.

On the other hand, just using these parallel variants of standard iterative methods, we are able to solve two and three dimensional problems of substantial industrial interest remarkably quickly. For example, the simulation shown in Figure 3.1 solves a three dimensional problem with a million

unknowns in less than a minute using 32 processors. For this reason, it seems appropriate to consider more accurate models of semiconductors, e.g., the Boltzmann equation [7], together with methods for accelerating such calculations using a diffusion approximation [13]. The understanding of such methods in the context of neutral particles (photons, neutrons, etc.) has advanced dramatically recently [11]. However, application of these ideas to electron transport is still in a formative stage. We hope to address this at a later date.

References

[1] Bagheri, B., Clark, T. W., and Scott, L. R. IPfortran: a parallel dialect of fortran. *Fortran Forum 11* (Sept. 1992), 20–31.

[2] Bagheri, B., Ilin, A., and Scott, L. R. Parallelizing UHBD. Research Report UH/MD 167, Dept. Math., Univ. Houston, 1993.

[3] Bagheri, B., Ilin, A., and Scott, L. R. Parallel 3-D MOSFET simulation. In *Proceedings of the* 27^{th} *Annual Hawaii International Conference on System Sciences* vol. 1, T.N.Mudge and B.D. Shriver, ed's, IEEE Computer Soc. Press, 1994, pp. 46–54.

[4] Bagheri, B., Ilin, A., and Scott, L. R. A Comparison of Distributed and Shared Memory Scalable Architectures. 1. KSR Shared Memory, accepted for the Scalable High Performance Computing Conference, May, 1994.

[5] Davis, M. E., Luty, J. D., Allision, B. A., and McCammon, J. A. Electrostatics and diffusion of molecules in solution: Simulations with the University of Houston Brownian Dynamics program. *Computer Physics Communications 62* (1990), 187–197.

[6] Davis, M. E., and McCammon, J. A. Solving the finite difference linearized Poisson-Boltzmann equation: A comparison of relaxation and conjugate gradient methods. *Journal of Computational Chemistry 10* (1989), 386–391.

[7] P. Degond and F.J. Mustieles, A deterministic partilce method for the kinetic model of semiconductors: the homogeneous field model, *Solid-State Electronics* **34** (1991), 1334-1345.

[8] Gilson, M. K., Straatsma, T. P., and McCammon, J. A. Open 'back door' in a molecular dynamics simulation of acetylcholinesterase. *Science March 4* (1994), 386–391.

[9] Holst, M., and Saied, F. Multigrid solution of the Poisson-Boltzmann equation. *Journal of Computational Chemistry 14* (1993), 105–113.

[10] Kendall Square Research Corporation. *KSR Fortran Programming.* Kendall Square Research, Waltham, MA, 1992.

[11] K.M. Khattab and E.W. Larsen, Synthetic acceleration methods for linear transport problems with highly anisotropic scattering, *Nuclear Sci. & Eng.* **107** (1991), 217–227.

[12] Scott, L. R. Elliptic preconditioners using fast summation techniques, Proc. Domain Decomposition 7, Penn State, October, 1993. To appear in Contemporary Mathematics, Americam Math. Soc., Providence.

[13] Scott, L. R. Computer design: a new grand challenge, In *Proceedings of the* 27^{th} *Annual Hawaii International Conference on System Sciences* vol. 1, T.N.Mudge and B.D. Shriver, ed's, IEEE Computer Soc. Press, 1994, pp. 3–6.

[14] Xie, D. Research Report UH/MD, Dept. Math., Univ. Houston, to appear.