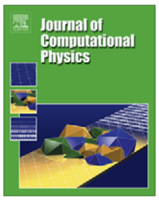
Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp



Performance of a parallel algebraic multilevel preconditioner for stabilized finite element semiconductor device modeling [☆]

Paul T. Lin [a,*], John N. Shadid [a], Marzio Sala [b], Raymond S. Tuminaro [c], Gary L. Hennigan [a], Robert J. Hoekstra [a]

[a] Sandia National Laboratories, P.O. Box 5800 MS 0316, Albuquerque, NM 87185-0316, USA
[b] BMW-Sauber, Hinwil, Switzerland
[c] Sandia National Laboratories, P.O. Box 969 MS 9159, Livermore, CA 94551-9159, USA

A R T I C L E   I N F O

Article history:
Received 1 November 2008
Received in revised form 15 April 2009
Accepted 12 May 2009
Available online 21 May 2009

Keywords:
Multilevel preconditioners
Multigrid
Nonsmoothed aggregation
Newton–Krylov
Schwarz domain decomposition
Graph partitioning
Drift-diffusion
Semiconductor devices
Finite element

A B S T R A C T

In this study results are presented for the large-scale parallel performance of an algebraic multilevel preconditioner for solution of the drift-diffusion model for semiconductor devices. The preconditioner is the key numerical procedure determining the robustness, efficiency and scalability of the fully-coupled Newton–Krylov based, nonlinear solution method that is employed for this system of equations. The coupled system is comprised of a source term dominated Poisson equation for the electric potential, and two convection–diffusion-reaction type equations for the electron and hole concentration. The governing PDEs are discretized in space by a stabilized finite element method. Solution of the discrete system is obtained through a fully-implicit time integrator, a fully-coupled Newton-based nonlinear solver, and a restarted GMRES Krylov linear system solver. The algebraic multilevel preconditioner is based on an aggressive coarsening graph partitioning of the nonzero block structure of the Jacobian matrix. Representative performance results are presented for various choices of multigrid V-cycles and W-cycles and parameter variations for smoothers based on incomplete factorizations. Parallel scalability results are presented for solution of up to 10^8 unknowns on 4096 processors of a Cray XT3/4 and an IBM POWER eServer system.

1. Introduction

The predictive computational modeling of advanced semiconductor devices for science and technology applications requires high resolution simulations. A base computational method for these devices is the drift-diffusion model [1,2]. This coupled system of nonlinear partial differential equations (PDEs) relate the electric potential to the electron and hole concentrations in these devices. The most common discretization approach for these equations is based on a finite volume (FV) method along with the Scharfetter–Gummel upwinding technique [1–3]. This approach applies an analytically derived, limiting case, one-dimensional solution to produce an exponential type of upwinding at each face of the FV cell. The resulting nonlinear algebraic system of equations is then commonly linearized by a nonlinear block Gauss–Seidel iteration (commonly referred to as Gummel's method) [4–7] or by a fully-coupled Newton type method [5,6,8,9], resulting in the generation of

☆ Partially supported by the DOE NNSA ASC program and the DOE Office of Science ASCR Applied Math Research program at Sandia National Laboratory. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.
* Corresponding author.
E-mail address: ptlin@sandia.gov (P.T. Lin).

0021-9991/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.jcp.2009.05.024

large, sparse linear systems. As a result efficient and robust parallel iterative solution methods are required to make such simulations tractable for large-scale problems. Currently preconditioned Krylov iterative methods are among the most robust and fastest iterative solvers over a wide range of applications (see Ref. [10] as well as its list of references). For these methods the preconditioner is the key numerical procedure influencing the robustness, efficiency and scalability of the linear solution.

The focus of this study is the initial evaluation of a recently developed fully-coupled algebraic multilevel preconditioning method [11–14] in the context of the drift-diffusion system. This fully-coupled preconditioner is based on aggregation techniques [15,16]. The aggregation technique is based on a sparse graph with connectivity and vertices defined by the nonzero block structure of the discrete approximation to the Jacobian. This graph therefore roughly corresponds to the mesh node connectivity of the underlying mesh of the discretization. An automated graph-based partitioning tool [17,18] is then applied to form subgraph partitions that aggregate a target number of vertices into a each aggregate. With a suitable choice of interpolation operator on the coarse aggregate a grid transfer operator can be defined. These grid transfer operators can then be used to produce approximate coarse operators through projection techniques and thereby enable the development of multilevel algebraic preconditioning methods.

Previous work that has considered the multilevel solution of the drift-diffusion system has included geometrically based multigrid methods applied to FV discretizations [7,19], geometrically based multigrid methods applied to finite element (FE) discretizations [9,20] and algebraic multigrid methods applied to FV discretizations [8]. In this study, the initial evaluation of the algebraic multilevel preconditioner for two-dimensional drift-diffusion systems is carried out in the context of a particular unstructured stabilized finite element discretization as described in Section 3. A FV Scharfetter–Gummel technique is currently under development, and a future study will evaluate the performance of the multilevel preconditioner on this discretization technique.

The remainder of the paper is structured as follows. In Section 2 the drift-diffusion equations are presented. Section 3 then presents the stabilized finite element formulation for this system. In Section 4 a very brief overview of the preconditioned Newton–Krylov solver is presented followed by a discussion in Section 5 of the graph-based fully-coupled multilevel preconditioner. The results of a set of numerical studies are presented in Section 6. These studies consider the effect of the key algorithmic parameters: the choice of the aggregate size, the selection of V-cycle vs. W-cycle, the number of multigrid cycles, the number of relaxation sweeps of the smoothers, and the choice of fill and overlap for the incomplete lower/upper (ILU) factorization smoother. Weak scaling studies are also presented comparing the one-level domain decomposition (DD) ILU preconditioner with a three-level multilevel preconditioner.

2. Governing equations for semiconductor device modeling

The equations governing the transport of charge carriers within a semiconductor device can be approximated using the standard drift-diffusion equations given by [1,2]:

$$\nabla \cdot (\epsilon \mathbf{E}) - q(p - n + C) = 0, \tag{1}$$

$$q\frac{\partial n}{\partial t} - \nabla \cdot \mathbf{J}_n + qG = 0, \tag{2}$$

$$q\frac{\partial p}{\partial t} + \nabla \cdot \mathbf{J}_p + qG = 0, \tag{3}$$

where

$$\begin{aligned}
\mathbf{E} &= -\nabla\psi,\\
\mathbf{J}_n &= qn\mu_n\mathbf{E} + qD_n\nabla n,\\
\mathbf{J}_p &= qp\mu_p\mathbf{E} - qD_p\nabla p.
\end{aligned}$$

The unknowns are: ψ, the electrostatic potential, n, the electron concentration (number of electrons per volume), and p, the hole concentration (number of holes per volume), with ϵ, the permittivity of the semiconductor material, q, the fundamental electron charge, μ_n and μ_p, the electron and hole mobilities, respectively, D_n and D_p, the electron and hole diffusion coefficients, respectively, C the doping profile, and G the generation/recombination source term.

Typically, (1)–(3) are scaled prior to discretization [1]. The scale factors used to non-dimensionalize this system, in this study, are given in Table 1. After scaling (1)–(3) the PDEs become in residual form

$$R_\psi = -\lambda^2\nabla^2\psi - (p - n + C) = 0, \tag{4}$$

$$R_n = \frac{\partial n}{\partial t} + \nabla \cdot (\mu_n n \nabla\psi) - \nabla \cdot (D_n \nabla n) + G = 0, \tag{5}$$

$$R_p = \frac{\partial p}{\partial t} - \nabla \cdot (\mu_p p \nabla\psi) - \nabla \cdot (D_p \nabla p) + G = 0, \tag{6}$$

Table 1
Scaling factors used to non-dimensionalize the semiconductor drift-diffusion equations. Here L_x, L_y are length scales for a rectangular 2D domain and the quantity V_0 is the thermal voltage [1]. †The intrinsic electron concentration is used as a scaling parameter. Other choices such as $\max(C(\mathbf{x}))$ over the domain are also common [1].

Quantity	Scaling factor symbol	Value of scaling factor
$\mathbf{x}$	x_0	Maximum length scale: $\max(L_x, L_y)$
ψ	V_0	$\frac{k_B T}{q}$
n, p, C	C_0	$n_i^\dagger$
D_n, D_p	D_0	$\max(D_n(\mathbf{x}), D_p(\mathbf{x}))$
μ_n, μ_p	μ_0	$\frac{D_0}{V_0}$
G	G_0	$\frac{D_0 C_0}{x_0^2}$
t	t_0	$\frac{x_0^2}{D_0}$
$\mathbf{E}$	E_0	$\frac{V_0}{x_0}$
J_n, J_p	J_0	$\frac{q D_0 C_0}{x_0}$

where

$$\lambda^2 = \frac{V_0 \epsilon}{q x_0^2 C_0}.$$

λ is the minimal Debye length of the device.

In this study these governing PDEs are discretized by a particular stabilized finite element method.

3. Stabilized finite element discretization

In this section a brief description of an initial stabilized finite element discretization of the drift-diffusion system is provided. This discretization is associated with a simple extension of an SUPG-type approach to the drift-diffusion system with the inclusion of a nonisotropic discontinuity capturing type term. The stabilized FE weak form is

$$F_\psi = \int_\Omega R_\psi \phi \, d\Omega = 0, \tag{7}$$

$$F_n = \int_\Omega R_n \phi \, d\Omega - \sum_e \int_{\Omega_e} \tau_n [\mu_n \mathbf{E} \cdot \nabla \phi] R_n \, d\Omega + \int_\Omega \nu(R_n) \nabla \phi \cdot \widehat{\mathbf{E}}^\perp \nabla n \, d\Omega = 0, \tag{8}$$

$$F_p = \int_\Omega R_p \phi \, d\Omega + \sum_e \int_{\Omega_e} \tau_p [\mu_p \mathbf{E} \cdot \nabla \phi] R_p \, d\Omega + \int_\Omega \nu(R_p) \nabla \phi \cdot \widehat{\mathbf{E}}^\perp \nabla p \, d\Omega = 0, \tag{9}$$

where

$$\widehat{\mathbf{E}}^\perp = \left[\mathbf{I} - \frac{\mathbf{E} \otimes \mathbf{E}}{\|\mathbf{E}\|^2} \right].$$

The stabilized FE strategy is used to control instability in the Galerkin FE formulation for the drift-diffusion system. The terms in the weak form include, the standard Galerkin term (first term in (7)–(9)), an SUPG-type term (second term in (8) and (9)) and finally a nonisotropic discontinuity capturing (DC) type term (third term in (8) and (9)). This methodology is based on a variation of the streamline upwind Petrov–Galerkin (SUPG) type FE formulations of Hughes et al. [21,22] and Shakib [23] for convection–diffusion systems and has similarities to the flux upwind Petrov–Galerkin method for the drift-diffusion equations of Carey et al. [24,25]. The stabilized FE method allows solution of convection–diffusion type systems by decreasing numerical oscillations due to convection effects. In addition, this stabilization improves the conditioning of, and therefore the iterative solution of, the Jacobian matrices in the linear subproblems generated by Newton's method. The addition of a nonlinear discontinuity capturing type operator [26,27] also attempts to control oscillations by adding diffusion in the direction perpendicular to the electric field. A more thorough discussion of stabilized FE methods for the drift-diffusion system, a new variational multiscale formulation and a comparison with a more standard Scharfetter–Gummel finite volume approach will be presented in [28].

3.1. Electron and hole stabilization parameters

The simplified stabilized FE formulation presented in this study uses a multidimensional generalization of the optimal stabilization parameter for solving a one-dimensional convection–diffusion equation with constant coefficients. The stabilization parameters are given by the formula

$$\tau_l = \frac{1}{\mu_l\sqrt{\mathbf{E}\mathbf{G}_c\mathbf{E}^T}}\left(\frac{1}{\tanh(\alpha_l)} - \frac{1}{\alpha_l}\right),$$

where

$$\alpha_l = \frac{\mu_l\sqrt{\mathbf{E}\mathbf{G}_c\mathbf{E}^T}}{D_l\|\mathbf{G}_c\|},$$

and $l = n, p$ for electrons and holes respectively. With this choice for (τ_n, τ_p) and no DC term this method can be shown to be equivalent to a box integration method using a Scharfetter–Gummel exponential upwinding technique in one dimension [28] that is considered a standard discretization on regular finite volume meshes [1,2]. The stabilizing terms have the property that the true solution also satisfies the weak variational form since the DC coefficients, ν_n and ν_p, are either linearly or quadratically proportional to the PDE residuals [22,23].

The multidimensional effect of convection (i.e. drift) is incorporated into the stability parameters by the use of the contravariant metric tensor, $\mathbf{G}_c$, of the transformation from local element coordinates $\{\zeta_\alpha\}$ to physical coordinates $\{x_i\}$:

$$[\mathbf{G}_c]_{ij} = \frac{\partial\zeta_\alpha}{\partial x_i}\frac{\partial\zeta_\alpha}{\partial x_j}.$$

Shakib [23] considers the one-dimensional limiting case of this multidimensional definition for the advection–diffusion equation and presents a comparison with the original SUPG technique.

Finally, it should be noted that this formulation, as presented, has some limitations. First this formulation is a simplification of proposed formulations for multiple advection–diffusion type equations that also couple the various equations in the definition of the least squares operators [29]. In addition a full variational multiscale analysis produces additional terms that affects both the stability and accuracy of these stabilized FE formulations. In [28] we consider various forms and contrast the stability and robustness of these methods in more detail. Finally, it is noted that in the parallel scaling studies that are the focus of this paper there is sufficient resolution so that the discontinuity capturing terms above are not employed.

4. Preconditioned Newton–Krylov method

Discretization of the drift-diffusion equations produces a large sparse, strongly coupled nonlinear system. This system is solved via a Newton–Krylov algorithm [30,31], where the linear systems (generated at each Newton step) are solved using a Krylov accelerator [32,33]. The nonlinear iteration is referred to as an *inexact* Newton–Krylov method when linear systems are solved inaccurately, which is computationally and theoretically justified during early Newton iterations [34].

Briefly, the convergence of Krylov methods is connected to the linear system condition number. For example, the convergence rate of the conjugate gradient (CG) method for symmetric positive definite problems, $Ax = b$, can be bounded by

$$\|e^{(k)}\|_A \leqslant 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k \|e^{(0)}\|_A, \tag{10}$$

where $e^{(k)} = x^* - x^{(k)}$ denotes the error at step k and κ is the condition number of the linear system (the ratio of largest to smallest eigenvalue in the symmetric case). Thus, CG converges rapidly when $\kappa \approx 1$. Unfortunately, even for straightforward equations that are dominated by Laplacian-type operators, $\kappa \propto h^{-2}$ where h is the mesh spacing on a uniform mesh. Thus the error decreases slowly for each iteration for high resolution meshes (i.e. when $h \ll 1$). Similar results exist for applying Krylov solvers to nonsymmetric systems (e.g. GMRES) [32].

For the problems considered here, convergence is not achieved without preconditioning [32]. Formally a preconditioner defines a nonsingular matrix M that approximates the original matrix A and is easily inverted. In the case of right preconditioning the original linear system is transformed to $AM^{-1}My = b$ where $AM^{-1} \approx I$. For the transformed system CG convergence is then bounded by (10) where κ is replaced by the condition number of the preconditioned linear system AM^{-1}. Thus, a preconditioning goal is to reduce the condition number. Ideally, it should be independent of mesh spacing to guarantee that convergence does not deteriorate as the mesh is refined. Finally, Krylov methods do not require A, or M^{-1} to be formed. Instead, procedures for applying A to a vector and efficient solution methods for $Mz = r$ must be provided.

5. Domain decomposition preconditioners

5.1. One-level additive schwarz methods

Domain decomposition (DD) preconditioners subdivide the computational domain into subdomains, *solve* the local subdomain problems, and then combine subdomain solutions. One DD method is the one-level Schwarz preconditioner [35,36]. To describe the Schwarz method utilized in this paper consider

$$Ax = b,$$

where A is an $n \times n$ matrix having a symmetric nonzero pattern. Define a graph $G = (\mathcal{V}, \mathcal{E})$ where the vertex set $\mathcal{V} = \{1, \ldots, n\}$ represents unknowns and the edge set $\mathcal{E} = \{(i,j)\ s.t.\ a_{ij} \neq 0\}$ represents vertex pairs coupled by a nonzero element in A. Graph partitioning is applied resulting in N subsets (or subdomains) $\mathcal{V}_i^0$ where $\mathcal{V}_i^0 \bigcap \mathcal{V}_j^0 = \emptyset (\forall i \neq j)$ and $\mathcal{V} = \bigcup_{i=1}^N \mathcal{V}_i^0$. Overlapping subsets are then constructed. Define $\mathcal{V}_i^1$ as the one-overlap decomposition of $\mathcal{V}$ where $V_i^1 \supset \mathcal{V}_i^0$ is obtained by including all immediate neighboring vertices to those in $\mathcal{V}_i^0$. The δ-overlap partition $\mathcal{V}_i^\delta$ is constructed by recursively applying this procedure.

Corresponding to each $\mathcal{V}_i^\delta$ define an $n_i^\delta \times n$ matrix $\boldsymbol{R}_i^\delta$ where $n_i^\delta = |\mathcal{V}_i^\delta|$ and $|\cdot|$ denotes cardinality. $\boldsymbol{R}_i^\delta$ restricts vectors in $\mathcal{R}^n$ to $\mathcal{R}^{n_i^\delta}$ by choosing vector entries corresponding to $\mathcal{V}_i^\delta$. The matrices $A_i^\delta = \boldsymbol{R}_i^\delta A (\boldsymbol{R}_i^\delta)^T$ are principal submatrices of A. In a parallel distributed environment each subdomain is assigned to one processor and a Schwarz preconditioner is defined by

$$(M_{AS}^\delta)^{-1} = \sum_{i=1}^{N} (\boldsymbol{R}_i^0)^T (A_i^\delta)^{-1} \boldsymbol{R}_i^\delta. \tag{11}$$

This is somewhat different from the classical additive Schwarz preconditioner which would use $(\boldsymbol{R}_i^\delta)^T$ instead of $(\boldsymbol{R}_i^0)^T$ in (11). The use of M_{AS}^δ typically converges a bit faster and avoids one communication step as each unknown is updated by only one subdomain even within overlapping regions [37,38].

Parallel implementation requires a factorization of a Dirichlet problem on each processor in the setup phase. Due to the large memory and floating point expense, a direct factorization can be replaced by an $ILU(k)$ incomplete factorization [32]. While this typically causes modest deterioration in the convergence rate, CPU time is reduced as each iteration is less expensive. In general, larger overlap leads to faster convergence up to a point where almost no further improvement is observed. Unfortunately, even slight overlap increases can lead to significant communication and computation requirements. Thus, we typically restrict $\delta \leqslant 2$.

The one-level Schwarz method is not scalable as all information exchange between subdomains only occurs through overlapping regions while for elliptic problems the domain of dependence is global. The condition number can be bounded by

$$\kappa((M_{AS}^\delta)^{-1} A) \leqslant C \frac{1}{H\delta},$$

(see for instance [39]) where H is the approximate subdomain diameter and C denotes a constant independent of H, h, and δ. The $1/H$ term implies that the condition number could increase as the number of subdomains increases which is consistent with an observed rise in the number of Krylov iterations. To rectify this, the preconditioner must incorporate some type of coarse system.

5.2. Multilevel schwarz

The primary idea of a multilevel Schwarz method is to introduce a coarse subproblem which captures the approximate coarse scale behavior. This coarse subproblem provides global coupling and can lead to optimal methods, condition numbers independent of the number of subdomains and either independent or logarithmically dependent on the number of unknowns per subdomain. Optimality in terms of cost requires that the coarse grid not be too fine if something like a direct solver is used to "solve" the coarse problem. Otherwise, a coarse solution may be approximated with the use of additional coarse levels. When coarse levels are incorporated, there are strong ties between domain decomposition and multigrid (see for example [40]).

There is a lack of agreement on the characteristics that separate domain decomposition methods with coarse solves from multigrid methods. Very rough guidelines to characterize the two approaches are shown in the left side of Fig. 1. We now briefly review some multigrid concepts.

Multigrid methods utilize multiple resolutions. Generally speaking, oscillatory components are reduced through simple relaxation such as a Gauss–Seidel iteration. The residual equation is then transferred to a coarse level under the assumption that the error should now be smooth. The coarse solution is then used to correct the fine level approximation. The idea is

domain decomposition	multigrid
1 to 3 levels	many levels
significant coarsening	modest coarsening
more expensive smoothers	inexpensive smoothers

```
MGV($A_\ell, u, b, \ell$) :
    if $\ell \neq \ell_{max}$
        $u \leftarrow \mathcal{S}_\ell^{pre}(A_\ell, u, b)$
        $r \leftarrow b - A_\ell u$
        $c \leftarrow 0$
        $c \leftarrow$ MGV($A_{\ell+1}, c, R_\ell r, \ell+1$)
        $u \leftarrow u + P_\ell c$
        $u \leftarrow \mathcal{S}_\ell^{post}(A_\ell, u, b)$
    else $u \leftarrow A_\ell^{-1} b$
```

Fig. 1. Left: domain decomposition vs. multigrid. Right: multigrid V-cycle to solve $A_\ell u = b$.

then applied recursively. Algebraic multigrid (AMG) methods automatically generate coarse levels and level transfer operators.

Fig. 1 presents a multigrid V-cycle. Here, A_1 is the linear system matrix that one is interested in solving. The A_ℓ define a hierarchy of different resolutions. P_ℓ is an interpolation operator that transfers solutions from level $\ell+1$ to level ℓ. R_ℓ restricts vectors from level ℓ to level $\ell+1$. Finally, $\mathcal{S}_\ell^{pre}$ and $\mathcal{S}_\ell^{post}$ define presmoothing and postsmoothing (sometimes referred to as relaxation) on level ℓ. These are local iterations which reduce errors that are oscillatory compared to A_ℓ's resolution. A W-cycle is obtained by adding a second $c \leftarrow \mathrm{MGV}()$ invocation immediately after the current one in Fig. 1. A multigrid method is defined once the $A_\ell, P_\ell, R_\ell, \mathcal{S}_\ell^{pre}$, and $\mathcal{S}_\ell^{post}$ are specified. Within geometric multigrid there is also a mesh $\mathcal{G}_\ell$ on each level. In this case, P_ℓ and R_ℓ are defined using geometric information (e.g., linear interpolation to transfer information at coarse nodes to fine nodes) and often the A_ℓ's are obtained by applying the same discretization technique (and code) to the different $\mathcal{G}_\ell$.

We consider algebraic multigrid techniques where level transfers and coarse resolution problems are automatically defined based on A_1. There are many trade-offs between geometric and algebraic multigrid in terms of robustness and convergence. The primary motivation for algebraic methods, however, has been their relative ease when interfacing with applications as complex mesh and geometric information is not needed within the solver. The coarse resolution systems are defined by the projection:

$$A_{\ell+1} = R_\ell A_\ell P_\ell.$$

Additionally, we take

$$R_\ell = P_\ell^T.$$

This is almost always done for symmetric problems though for nonsymmetric systems alternatives may be warranted (see [41]). We intend to explore other possibilities in a future paper based on the AMG algorithm described in [42]. For this paper, we take $\ell_{max} = 3$ and use one-level Schwarz smoothers. In particular,

$$\mathcal{S}_\ell^*(A_\ell, u, b): \ \textit{repeat } \nu \textit{ times } u \leftarrow u + \widetilde{M}_{AS}(b - A_\ell u)$$

where $\nu > 1$ corresponds to multiple sweeps, $*$ indicates either *pre* or *post*, and $\widetilde{M}_{AS}$ denotes (11) with inverses replaced by $ILU(k)$.

Once the P_ℓ are defined, the entire multigrid cycle is specified. Here, we want P_ℓ to accurately interpolate constants. Constructing P_ℓ consists of deriving its sparsity pattern and then assigning nonzero values. The sparsity pattern is determined by decomposing the set of A_ℓ's nodal blocks into *aggregates* $\mathcal{A}_\ell^i$ such that

$$\bigcup_{i=1}^{N_{\ell+1}} \mathcal{A}_\ell^i = \{1, \ldots, N_\ell\}, \quad \mathcal{A}_\ell^i \cap \mathcal{A}_\ell^j = \emptyset, \quad 1 \leqslant i < j \leqslant N_{\ell+1},$$

where N_ℓ denotes the number of nodal blocks on level ℓ. A nodal block refers to the submatrix which couples all degrees of freedom defined at the same grid node. In our case, the nodal block dimension is $m = 3$ corresponding to the electrostatic potential, electron concentration, and hole concentration unknowns. Each aggregate $\mathcal{A}_\ell^i$ on level ℓ gives rise to one node on level $\ell+1$. Fig. 2 gives an illustration of aggregates on an unstructured grid. Each aggregate normally corresponds to a set of connected nodes. Typically, one wants aggregates to be approximately the same size and roughly spherical in shape (at least for isotropic problems). The $\mathcal{A}_\ell^i$ are formed based on the connectivity and the strength of the connections between A_ℓ's nodal blocks (e.g. see [15,43] for aggregation algorithms). In domain decomposition aggregates are normally large. One possibility is to have aggregates correspond to subdomains (e.g. $\mathcal{A}_1^i = \mathcal{V}_i^0$). In this work, we use METIS and ParMETIS to generate aggregates [17,18]. These packages partition graph vertices into disjoint sets of approximately equal size in a way that essentially minimizes cuts between the disjoint sets. In our case, a graph is constructed corresponding to the nodal block matrix. One nice aspect is that we can experiment with different coarsening rates by varying the number of partitions requested from METIS (or ParMETIS).

P_ℓ is populated to correspond to piecewise-constant interpolation over each aggregate for each of the three solution components. This is described by first defining an operator I_ℓ^s which is a row partitioning of the identity, $I_\ell \in \mathcal{R}^{n_\ell \times n_\ell}$ where n_ℓ is the dimension of A_ℓ. That is, $I_\ell^s \in \mathcal{R}^{|\mathcal{A}_\ell^s| \times n_\ell}$ contains a subset of the rows of the identity matrix corresponding to the aggregated indices in $\mathcal{A}_\ell^s$.

The interpolation is then given by

$$\widehat{P}_\ell = \begin{pmatrix} I_\ell^1 B_\ell & & & \\ & I_\ell^2 B_\ell & & \\ & & \ddots & \\ & & & I_\ell^{N_{\ell+1}} B_\ell \end{pmatrix}, \tag{12}$$

where B_ℓ is an $n_\ell \times m$ matrix. The jth column contains ones for all rows associated with the jth degree of freedom within each block corresponding to a grid node. Otherwise, the jth column contains zeros. The matrix B_1 represents modes (the constants) that must be accurately interpolated. It is common to choose B_1 to represent rigid body modes (translations in

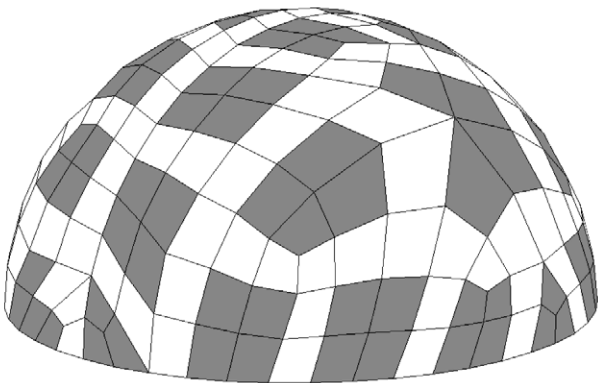


Fig. 2. Each aggregate corresponds to all nodes in and adjacent to one grey region.

each coordinate direction and rotations about each coordinate axis) for elasticity applications. The actual interpolation is slightly different from (12) in that columns of P_ℓ are normalized. In particular, a QR factorization is applied to each $I_\ell^s B_\ell$. The Q matrix is used in place of $I_\ell^s B_\ell$ to define the interpolation and the R matrix is used to define $B_{\ell+1}$ so that $B_\ell = P_\ell B_{\ell+1}$.

This prolongator corresponds to nonsmoothed aggregation. For symmetric elliptic problems, it is well known that multigrid methods using grid transfers based on piecewise constants are not scalable in that the convergence rate deteriorates as the mesh is refined. While smoothing the grid transfer basis functions can rectify this for symmetric elliptic problems (see [15,16]), the situation is less clear for a highly nonsymmetric system. Nonsmoothed aggregation has relatively inexpensive setup costs and has the nice property that when A_1 is an M-matrix, then the coarse level matrices inherit this property. This implies that smoothers whose efficiency relies on M-matrix properties are effective on coarse levels [44]. M-matrices arise frequently in PDE discretizations and have desirable characteristics in the context of iterative methods. They have the property that all diagonal entries are positive, all off-diagonal nonzeros are negative, the matrix is nonsingular, and all matrix entries in the inverse are greater than or equal to zero [45,46].

In terms of software, fine-level nonoverlapping subdomains are obtained using Chaco [47]. IFPACK provides overlapping subdomains and incomplete factorizations [48]. Krylov methods are implemented in AztecOO [38,49]. A sparse KLU factorization is used for the coarse direct solver [50]. The multigrid cycles and grid transfers are provided by ML [51]. ML also provides aggregation routines though METIS and ParMETIS [17,18] that are used in this paper. Access to this software is obtained via the Trilinos framework [52].

6. Results and discussion

The numerical studies presented in this paper are intended to briefly examine the effect of important algorithmic choices for the nonsmoothed aggregation multilevel method presented above. Since there are a large number of specific algorithmic choices and parameter settings that can influence the performance of multilevel-type methods it is beyond the scope of this study to present an exhaustive investigation of these issues. Instead a set of reasonably representative results are presented that attempt to illuminate the essential behavior of the techniques. The issues that are considered include the effects of

- choice of the aggregate size,
- choice of V-cycle vs. W-cycle,
- number of multigrid cycles,
- total number of relaxation sweeps (presmoothing and postsmoothing),
- choice of ILU smoother parameters (fill and overlap).

In order to assess the relative performance of the multilevel preconditioner, algorithmic scaling studies for the one-level and multilevel Schwarz preconditioner were performed for the following three test cases:

- A steady-state drift-diffusion solution for a two-dimensional 2×1.5 μm NPN bipolar junction transistor (BJT).
- A transient drift-diffusion solution for a 1×0.5 μm two-dimensional diode.
- And a pseudo one-dimensional steady-state drift-diffusion solution for a 1×0.125 μm diode that considers the effect of varying levels of doping.

The results present a comparison of the weak scaling and relative performance of the one-level and multilevel preconditioner for steady-state and transient simulations. Performance of these preconditioners when one or both cores of a dual core processor on a Cray XT3/4 is also examined. Finally a comparison of these methods on two different compute platforms, a

Cray XT3/4 system (ASC Red Storm) and an IBM POWER5-based eServer system (ASC Purple) is also presented. (ASC denotes the US Department of Energy Advanced Simulation and Computing Program.)

The Cray XT3/4 supercomputer architectures are the result of the Sandia-Cray Red Storm collaborative venture. Each compute node of Red Storm contains 2 GB RAM memory and one AMD dual core 2.4 GHz Opteron processor. Red Storm uses the Cray SeaStar custom chip for the interconnect and runs the Catamount lightweight kernel on the compute nodes. The semiconductor simulation code ASC Charon [53] that implements the drift-diffusion model and the iterative solution methods was compiled using the PGI 6.2.5 compiler. Unless otherwise stated, the Cray XT3/4 simulations were run using only one core per compute node. The Lawrence Livermore National Laboratories (LLNL) ASC Purple machine is an IBM POWER eServer system with p5 575 compute nodes. Each compute node has 30 GB RAM memory and eight 1.9 GHz POWER5 processors. The compute nodes are connected by the IBM High Performance Switch. Purple runs AIX 5.3 and the AIX compilers were used to build Charon.

Although the semiconductor device simulator described in this paper handles unstructured meshes, the test cases presented actually use uniform quadrilateral meshes. Ref. [11] provides an example of the application of this multilevel nonsmoothed aggregation preconditioner to a geometry with tetrahedral meshes for a different set of governing equations, the incompressible Navier–Stokes equations for fluid flow. High aspect ratio elements for anisotropic problems can have a significant detrimental effect on the performance of the multilevel nonsmoothed aggregation preconditioner. Ref. [14] discusses our semi-coarsening approach to handling anisotropic problems with stretched meshes for the incompressible Navier–Stokes equations for fluid flow.

6.1. *Numerical studies on steady-state solution to the drift-diffusion system*

This first set of numerical studies involves the solution of the two-dimensional drift-diffusion equations for a 2×1.5 μm silicon NPN BJT (Fig. 3). “NPN” denotes a transistor with *n*-type (material with additional negative charge carriers), *p*-type (material with excess holes), and *n*-type material. This geometry has three contacts. The base at the top left corner, the emitter at the top right corner, and the collector along the entire bottom. Both the emitter and base are 0.1 μm wide. The steady-state calculation is performed with a voltage bias of 0.3 V: the base and collector are held at ground and the emitter is assigned a voltage of -0.3 V. The initial guess for the drift-diffusion equations is taken as the solution to the nonlinear Poisson (NLP) problem [54]. Fig. 3(a) shows the signed logarithm of the doping for the device. The signed logarithm $slog(x)$ is defined as $slog(x) = sign(x)log_{10}(1 + |x|)$. The maximum donor doping is 10^{19} and the maximum acceptor doping is 10^{16}. Fig. 3(b) shows the corresponding steady-state electric potential solution.

As mentioned earlier, the fine-level nonoverlapping subdomains are obtained using Chaco as a preprocessing step. Each subdomain will be assigned to one processor (or one processor core for dual core processors), so naturally the size of the subdomain is limited by the amount of memory a processor can access. From a practical point of view, the size of the subdomain is usually determined by the trade-off between reducing the number of processors required to run a simulation and reducing the total run time of the simulation. Larger subdomains (larger number of matrix rows) means fewer processors are required to run a simulation but as each processor has more work, the total run time of the simulation will increase. For the steady-state BJT test cases that will be discussed, a subdomain with about 30,000 unknowns tends to be a good balance between the number of processors required and total run time.

6.1.1. *Effect of fill and overlap for ILU smoother*

We first consider the effect of fill and overlap for an ILU(k) smoother for a three-level preconditioner with one W(1,1) cycle and coarser levels generated with 85 nodes per aggregate. We use the conventional notation of “W(number of presmoothing sweeps, number of postsmoothing sweeps)” to denote the type of multigrid cycle and the number of relaxation sweeps. METIS and ParMETIS are used by the aggregation technique to generate the medium and coarse levels respectively.

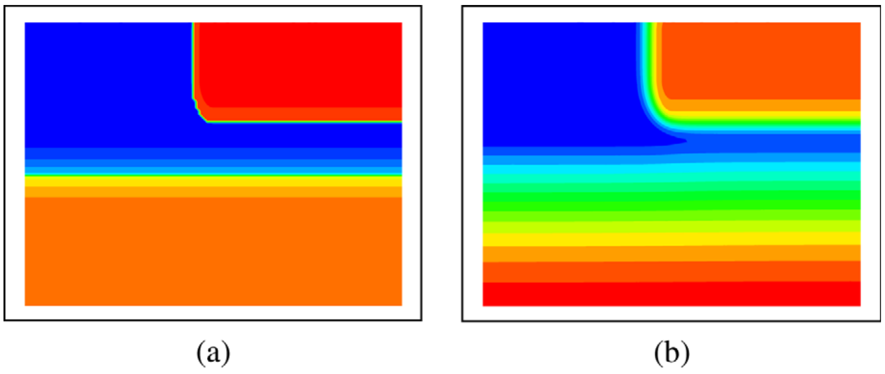


Fig. 3. (a) Signed logarithm of doping for 2×1.5 μm 2D NPN BJT; (b) Corresponding steady-state electric potential for 2×1.5 μm 2D NPN BJT at 0.3 V bias. (For interpretation to colours in Figs. 3–7, the reader is referred to the web version of this paper.)

ILU(k) is the smoother on the fine and medium levels with KLU direct solver on the coarse level. A fine mesh with 3520×2640 elements was used for the steady-state drift-diffusion solution for the 2×1.5 μm NPN BJT with 0.3 V bias and run on 2048 cores (both cores on 1024 nodes) of Red Storm. The coarsening algorithm used 85 nodes per aggregate to generate the coarser levels. The fine, medium, and coarse level have 27.9 million, 325,000 and 3828 unknowns respectively, so there were approximately 14,000 unknowns per core. Table 2 presents this comparison. The three values for each entry denote: average Krylov iterations per Newton step, average time to construct the preconditioner (preconditioner set-up) per Newton step, and average linear solve time per Newton step (does not include time to construct the Jacobian which was about 2.5 s per Newton step). All runs required seven Newton steps.

In the context of the level of fill, there is an advantage in terms of reduction of iteration count and CPU time for increasing fill from 0 to 1, followed by a more modest improvement increasing fill from 1 to 2. Further increases quickly reach the point of diminishing returns. Although further increases in the fill slightly reduce the iteration count and CPU time, the increased requirement in memory does not justify continued increases in fill. In terms of the choice of level of overlap, there is an advantage in terms of iteration count and CPU time of increasing overlap from zero to one-level, with a substantially smaller reduction increasing the levels to two. Further increase in levels of overlap has reached the point of diminishing returns for CPU time, and the increased memory requirement make continued increases unwarranted. Based on these results a reasonable choice for fill and overlap is two levels of fill and one-level of overlap. These parameter setting will be used to obtain the results in the remainder of the paper. Although the choice of fill and overlap does have an effect on solution time, the results in the next section will demonstrate that the choice of aggregate size has a significantly greater effect.

6.1.2. *Effect of aggregate size*

The rate at which the coarser level is coarsened can have a large effect on the number of linear solver iterations as well as linear solution time. Table 3 shows a comparison of aggregate size for a three-level preconditioner using METIS and ParMETIS to generate aggregates for the medium and coarse levels respectively. The fine mesh has 111.6 million unknowns. An ILU(2) with one-level overlap DD smoother is used on the fine and medium levels and KLU solver on the coarse level. Both presmoothing and postsmoothing (one relaxation sweep) are used in the multigrid cycle. Runs were performed on 4096 nodes (one core per node) of Red Storm. For both the V-cycle and W-cycle, the columns denote: average number of Krylov iterations per Newton step, time to construct the preconditioner per Newton step, and linear solution time per Newton step (includes time to construct the preconditioner). Times are given in seconds. The cost to construct the Jacobian is about 7 s for all the cases (not included in the "linear solve time" in the table). From the results it is clear that in this aggressively coarsened algorithm sufficient coarsening is required to balance the cost of the coarse grid solve (that correlates with the coarse

Table 2
Comparison of fill and overlap for an ILU(k) smoother for a three-level preconditioner with one W(1,1) cycle for the 2D NPN BJT; entries present average values per Newton step: iterations/preconditioner setup time/linear solve time (times in seconds); fine level with total of 27.9 million unknowns; run on 2048 cores of Red Storm.

Fill	Levels of overlap				
	0	1	2	3	4
ILU(0)	309/1.39/26.7	277/1.46/24.1	268/1.54/23.5	262/1.66/23.7	261/1.74/23.5
ILU(1)	287/1.46/25.0	246/1.58/21.4	228/1.72/20.2	221/1.81/19.8	217/1.92/19.8
ILU(2)	275/1.54/24.3	233/1.72/21.0	210/1.83/19.0	200/1.97/18.5	190/2.04/17.9
ILU(3)	266/1.66/23.9	221/1.79/20.6	199/1.91/18.5	187/2.08/17.8	179/2.22/17.5
ILU(4)	260/1.74/23.8	214/1.92/20.0	191/2.07/18.2	180/2.20/17.5	170/2.36/17.4

Table 3
Comparison of effect of aggregate size for the three-level preconditioner for the 2D NPN BJT with 111.6 million unknowns on the fine level. Results are for average values per Newton step and all runs required seven Newton steps. "Prec time" and "lin sol time" are the preconditioner setup and linear solve times respectively (times are given in seconds). 4096 nodes of the Red Storm machine were used (one core per node).

Agg	unknowns		V(1,1)-cycle			W(1,1)-cycle		
	Medium	Coarse	Ave iter	Prec time	Lin sol time	Ave iter	Prec time	Lin sol time
50	2.225M	44,499	317	32.3	98.3	245	32.0	108
60	1.853M	30,885	340	17.5	78.3	265	18.0	79.4
70	1.587M	22,674	353	10.6	66.6	275	11.1	64.5
80	1.388M	17,352	367	6.9	62.3	288	7.2	55.2
85	1.306M	15,363	381	6.3	63.9	295	5.9	52.8
90	1.233M	13,695	386	6.0	64.3	297	5.9	51.7
100	1.108M	11,082	402	4.6	64.7	310	4.4	50.4
125	885.9K	7086	426	3.1	68.7	334	3.0	50.0
150	737.6K	4917	445	2.6	71.6	349	2.6	53.9
175	631.1K	3606	465	2.4	72.2	362	2.5	54.4

problem size), and the decreased convergence rate as the coarse problem becomes smaller. For the three-level preconditioner applied to this particular example, aggregate sizes of about 80–100 and 100–125 appear to provide the minimal solution times for the $V(1,1)$ and $W(1,1)$ cycles, respectively.

Although a three-level preconditioner was considered here, one could certainly use preconditioners with more levels. The trade-off between the number of levels and sizes of the coarser levels has a significant impact on the performance of the preconditioner. Less aggressive coarsening means more levels and larger coarser levels. The larger coarser levels will provide more accurate corrections to the finer meshes and reduce the number of linear solve iterations. However, the smoothers on the larger coarser meshes will be more expensive, so even with the reduction in the number of linear solve iterations the CPU time may increase. More aggressive coarsening means fewer levels and smaller coarser levels. The smaller coarser levels will provide less accurate corrections to the finer meshes and increase the number of linear solve iterations. However, the smoothers on the smaller coarser meshes will be less expensive, so even with the increase in the number of linear solve iterations, the CPU time may decrease.

The drift-diffusion equations require a heavyweight smoother such as ILU. Without aggressive coarsening, the next coarser level will be large, and ILU will also be expensive on this level. Because a heavyweight smoother such as ILU is required, one can use an aggressive coarsening scheme to obtain convergence reasonably quickly. And the coarser level produced from the aggressive coarsening will be sufficiently smaller than the fine level so that the expense of the ILU for the coarser level is not significant compared with the fine level. We are pursuing physics-based preconditioning methods which will allow us to use less expensive smoothers than ILU (for examples of physics-based preconditioners, see [10]). With less expensive smoothers, less aggressive coarsening may be better than aggressive coarsening.

For the largest problems in this study (112 million unknowns), a three-level preconditioner performs well. For significantly larger problems, for example on the order of one billion unknowns, a four-level preconditioner would be needed.

6.1.3. Effect of multigrid parameters

Table 4 presents a comparison of performance of the multigrid preconditioner as a function of the various multigrid parameters. These include the choice of V-cycle vs. W-cycle, the number of presmoothing and postsmoothing (relaxation) sweeps, and the number of multigrid cycles. All runs used a three-level preconditioner with ILU(2) smoother with one-level of overlap on fine and medium levels and with a KLU direct solver on the coarse level. The coarsening algorithm used 85 nodes per aggregate to generate the coarser levels. The fine, medium, and coarse level problems have 27.9 million (3520×2640 elements), 325,000 and 3828 unknowns, respectively. The runs were performed on 2048 cores (both cores on 1024 nodes) of the Red Storm machine.

Each entry in the table lists the average number of Krylov iterations per Newton step and the average time to perform the linear solve per Newton step in seconds (includes time to construct the preconditioner but not the Jacobian). All calculations required seven Newton steps. Unrestarted GMRES was used for all the calculations. The linear solve tolerance was 10^{-6} and the matrix was scaled so that the sum of the absolute values of the nonzeros within each matrix row is one. The "presmooth only" portion of the tables presents the effect of increasing the number of presmoothing sweeps while the number of postsmoothing sweeps remains zero. The "postsmooth only" portion of the tables presents the effect of increasing the number of

Table 4
Comparison of multigrid parameters: V-cycle vs. W-cycle, smoothing sweeps, and number of multigrid cycles; entries are average values per Newton step: iterations/linear solve time; 28 million unknowns on fine mesh; 2048 cores on Red Storm.

	Multigrid cycles	Relaxation sweeps					
		1	2	3	4	5	6
V-cycle							
Presmooth only	1	443/36.9	311/25.9	281/26.0	253/26.0	239/27.4	226/28.8
	2	271/26.1	218/25.5	191/26.9	178/29.4	166/31.8	158/34.8
Postsmooth only	1	421/33.8	287/23.2	255/23.1	229/23.4	216/24.4	206/25.9
	2	254/24.2	200/23.1	176/24.7	161/26.6	151/29.0	144/31.3
Presmooth and postsmooth	1	290/23.4	232/24.0	206/25.9	187/28.1		
	2	202/23.4	162/26.9	143/31.1	130/34.9		
	3	165/25.9	132/30.8	115/36.0	104/40.4		
	4	141/28.1	112/33.8	97/39.9	88/45.2		
W-cycle							
Presmooth only	1	330/27.8	246/22.4	224/22.9	202/23.6	193/24.6	183/26.2
	2	199/23.0	172/24.4	152/25.5	142/27.8	133/29.8	127/32.0
Postsmooth only	1	312/26.3	228/20.4	205/20.7	184/21.0	173/22.0	163/22.9
	2	187/21.6	160/22.7	140/23.5	128/25.2	120/26.7	114/28.7
Presmooth and postsmooth	1	233/21.0	187/21.3	166/23.2	153/25.6		
	2	163/23.1	131/25.5	116/29.1	107/32.6		
	3	132/26.0	105/29.4	92/33.5	84/37.8		
	4	112/28.7	89/32.7	78/37.2	71/42.1		

postsmoothing sweeps while the number of presmoothing sweeps remains zero. For the "both presmoothing and postsmoothing" portion of the tables, one relaxation sweep means one presmoothing sweep and one postsmoothing sweep, two relaxation sweeps means two presmoothing sweep and two postsmoothing sweeps, etc.

The following trends in Table 4 can be observed:

- If memory is an issue, one can always decrease the Krylov iterations by either increasing the number of multigrid cycles or increasing the number of relaxation sweeps. For the example considered, this can reduce the average iterations by a factor of five. However, depending on the problem, this may or may not reduce the CPU time.
- W-cycles are more effective at reducing the number of iterations than V-cycles, and also tends to reduce the amount of CPU time. Use of a W-cycle will also have a memory advantage as the size of the Krylov subspace can be smaller for an equal level of linear solve convergence.
- Postsmoothing appears to be more effective than presmoothing in reducing the number of iterations. It is unclear why this should be the case. Normally, for a given number of smoothing sweeps, the number of iterations should not change too much depending on how they are distributed between presmoothing and postsmoothing. Postsmoothing costs a little less than presmoothing (savings of one matrix–vector product).
- For the case with both presmoothing and postsmoothing, while increasing the number of relaxation sweeps reduces the number of iterations, the amount of CPU time tends to increase. The optimal number of relaxation sweeps depends on whether the coarse grid correction (smooth modes) or the fine grid relaxation (oscillatory modes) dictate convergence. If the smooth modes are the bottleneck, then there is little benefit in performing more relaxation.
- In terms of number of multigrid cycles, for unrestarted GMRES, one would not expect it to be advantageous to perform more than one multigrid cycle. For example, if 100 unrestarted GMRES iterations are required to obtain a certain level of convergence using one multigrid cycle, then one would expect more than 50 GMRES iterations using two multigrid cycles per iteration to be required to obtain the same level of convergence. From the table, one can see that increasing the number of multigrid cycles from one to two per iteration does not reduce the number of iterations by half. However, whether there is an advantage in CPU time by increasing the number of multigrid cycles from one to two depends on the number of iterations necessary when one multigrid cycle is being used. When the number of iterations is high, the cost of the Krylov orthogonalization can be very expensive. This is evident for the case with either one presmoothing only or one postsmoothing only which requires a large number of iterations. Increasing the number of multigrid cycles from one to two leads to such a substantial reduction in iterations that it also reduces the CPU time. However, for almost all the other cases in the table, the reduction in iterations is not sufficient to reduce the CPU time.

6.1.4. Algorithmic scaling study comparing one-level and three-level Schwarz preconditioner for the 2D NPN BJT

This section presents a "weak scaling study" for the solution of the 2D steady-state drift-diffusion equations where the problem is scaled up in a manner that approximately keeps the amount of work per processor constant. The test case considered is the 2×1.5 μm NPN BJT at 0.3V bias. Each larger mesh is a uniform refinement of the previous mesh. The problem was scaled up to a mesh of 7040×5280 elements and 112 million unknowns. The coarsening algorithm uses 85 nodes per aggregate for the first and second levels of aggregation. The three-level preconditioner used a W(1,1)-cycle with ILU(2) with one-level of overlap as smoothers for the fine and medium level and KLU direct solver on the coarse level. Table 5 shows the results run on Red Storm (one core per compute node was used). "Avg iter/N" is the average Krylov iterations per Newton step. "Time" is the time to perform the linear solve per Newton step in seconds (includes time to create the preconditioner but not construct the Jacobian). The time to construct the Jacobian is about seven seconds per Newton step. Because this time is about the same for all the calculations and would just translate the curves vertically, it is not included in the values in the table. All runs required seven Newton steps. Table 5 clearly indicates the reduced growth in the number of Krylov iterations for the multilevel method as compared to the one-level DD technique. For the one-level DD preconditioner the theoretical scaling of $N^{1/2}$ appears evident (N denotes the number of unknowns in the problem). For the multilevel method the reduction in iteration count growth is also reflected in the reduced CPU time relative to the one-level method. In the case of the large-scale 112 million unknown problem on 4096 processors the three-level preconditioner performs significantly better than the

Table 5
Weak scaling study comparing one-level and three-level preconditioners for the 2D NPN BJT on Red Storm. "Time/N" denotes the linear solve time per Newton step.

Proc	Fine grid (elements)	Fine grid unk	1-level DD ILU		3-level W(1,1) agg85			
			Avg iter/N	Time/N (s)	Medium unk	Coarse unk	avg iter/N	Time/N (s)
4	220×165	110K	68	3.0	1290	15	36	3.8
16	440×330	438K	142	7.4	5124	60	66	5.8
64	880×660	1.75M	287	21	20454	240	111	9.5
256	1760×1320	6.98M	571	73	81699	960	156	15
1024	3520×2640	27.9M	1145	278	327K	3843	219	25
4096	7040×5280	112M	2264	1073	1.31M	15363	295	53

one-level preconditioner with about a factor of 20 reduction in CPU time. These results are also presented graphically in Fig. 4. While the three-level method is clearly not scaling optimally, both in iteration count and CPU time, these results are encouraging and indicate the value of the multilevel method in reducing the iteration count and CPU time for large-scale problems. In the context of the CPU time, the growth in time is due to both the time to complete the coarse problem direct solve as well as the increase in the number of iterations. For example, when increasing the problem size from 28 million to 112 million, the average linear solve time increases from 25 to 53 s. Roughly half this time is due to the large increase in factorization and solve time of the 15,400 row coarse matrix compared with the 3800 row coarse matrix. To move towards optimal *h* independent convergence, currently specialized coarsening techniques that respect the material junction of devices at which large jumps in material properties occur, along with Petrov–Galerkin type operator projections to form the coarse operator are being pursued. A future study will consider these attempts at improving the convergence rates of the multilevel method.

Table 6 compares the results between using one core per compute node and using both cores per compute node but with half the number of compute nodes on Red Storm. The two cores share a 1MB L2 cache on chip. Use of the second core is handled by the operating system, and from the point of view of the application is handled like a second MPI process. From the point of view of the application, whether an MPI task needs to communicate with another MPI task that is either running on the second core or on another node is immaterial and actually the MPI implementation was not shared-memory aware. The code currently uses a single-level MPI-only approach. We will be exploring the issue of threading with shared memory cores in the future. The most obvious effect of using both cores is that each MPI process has about 1 GB RAM memory per process, due to the two cores sharing the 2 GB RAM per compute node (note that after September 2008 the amount of RAM was doubled). The first column of the table lists the number of cores (which is also the number of MPI tasks). The four columns for each preconditioner are: average Krylov iterations per Newton step, time to perform linear solve in seconds when

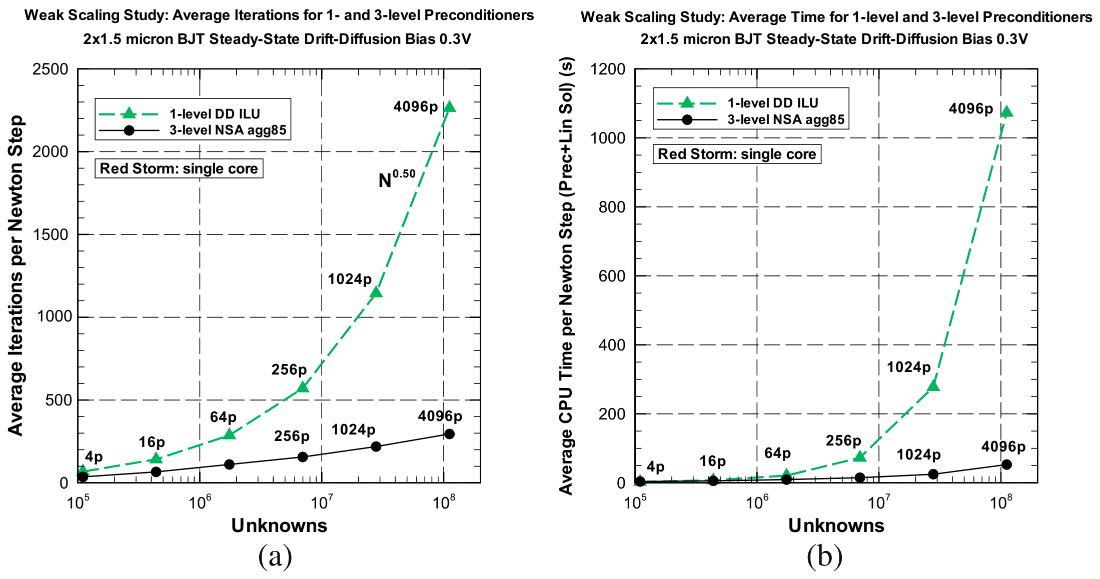


Fig. 4. (a) Weak scaling study comparing iteration count as a function of problem size for the one-level and three-level precondtioners for the 2D NPN BJT (b) Comparison of CPU time per Newton step as a function of problem size for the one-level and three-level precondtioners for the 2D NPN BJT.

Table 6
Weak scaling study comparing one-level and three-level preconditioners for the 2D NPN BJT on Red Storm using one or both cores on a compute node. "Time/Newt" denotes the linear solve time per Newton step.

Cores	Fine grid unk	1-level DD ILU				3-level W(1,1) agg85			
		Avg	Time/Newt (s)		Eff (%)	Avg	Time/Newt (s)		Eff (%)
		Iter/N	Single	Dual		Iter/N	Single	Dual	
4	110K	68	2.98	3.36	89	36	3.77	4.13	91
16	438K	142	7.36	8.44	87	66	5.80	6.54	89
64	1.75M	287	21.5	24.9	86	111	9.53	10.9	87
256	6.98M	571	73.0	84.1	87	156	14.8	17.6	84
1024	27.9M	1145	278	317	88	219	25.3	29.6	85
4096	112M	2264	1073	1192	90	295	52.8	59.9	88

only one out of the two cores is used, time to perform linear solve in seconds when both two cores are used but with half the number of compute nodes, and the efficiency of using the second core and reducing the number of nodes by half. For example, for the row in the table with 4096 cores, "single" denotes that 4096 compute nodes were used, and only one of out the two cores per node was used while "dual" denotes that 2048 computes nodes were used, and both cores per node were used. "Efficiency" is the quotient of the "single" time and the "dual" time and is not to be confused with parallel efficiency. 100% efficiency means that the dual core time is equal to the single core time (which means that rather than run one calculation and use only one core per node, one can use half the nodes and run two calculations simultaneously in the same amount of wall time). A W(1,1) multigrid cycle was used with ILU(2) with one-level of overlap as smoothers for the fine and medium level and KLU direct solver on the coarse level. Eighty-five nodes per aggregate were used to generate the coarser levels. All runs required seven Newton steps.

6.2. Numerical studies of a transient two-dimensional diode

The previous studies considered the performance of the multilevel preconditioner for steady-state solutions; the following example considers the transient case. The geometry for this test case is a two-dimensional 1×0.5 μm diode with two contacts on the top. Each contact is 0.3 μm wide with one in the upper left corner and the other in the upper right corner. A transient sinusoidal electric potential is applied to the upper left contact with amplitude of 0.5 and period of 1.0 s. Fig. 5 shows the signed logarithm of the doping as well as the electric potential for three different times.

Table 7 shows a comparison of the one-level preconditioner with a three-level preconditioner for this transient drift-diffusion problem. For the three-level preconditioner, METIS and ParMETIS with 60 nodes per aggregate were used to generate the coarser levels, and one V(1,1) multigrid cycle was used. ILU(2) with one-level of overlap was used as the smoother on the fine and medium levels with KLU direct solver on the coarse level. A sequence of three meshes was used: 650×325 elements, 1300×650 elements, and 2600×1300 elements, run on 64, 256, and 1024 cores on Red Storm, respectively (both cores on a compute node were used). The initial condition is the steady-state drift-diffusion calculation at zero voltage bias. For the transient simulations, a fixed time step was used for the entire calculation with the final simulation time taken to be equal to half the period of the sinusoidal boundary condition, i.e. 50, 10, and 5 time steps were used for the $\Delta t = 0.01, 0.05, 0.1$ cases respectively. The first order backward Euler time integration was used. Columns in Table 7 for each preconditioner present the average number of Newton steps per time step, average GMRES iterations per Newton step, average time to perform linear solve (including preconditioner setup) per Newton step, and average total time per Newton step (includes time for everything during a Newton step). Times are reported in seconds.

In Table 7 it is clear that the multilevel method shows a very significant benefit in iteration count reduction and CPU time reduction for all mesh resolutions and time steps considered in this study. While for a fixed time step an h independent convergence rate is not achieved, only a moderate increase is evident. For the largest mesh (10.2 million unknowns), the three-level preconditioner is about an order of magnitude faster than the one-level preconditioner. For a given resolution the iteration count is relatively constant as a function of time step size.

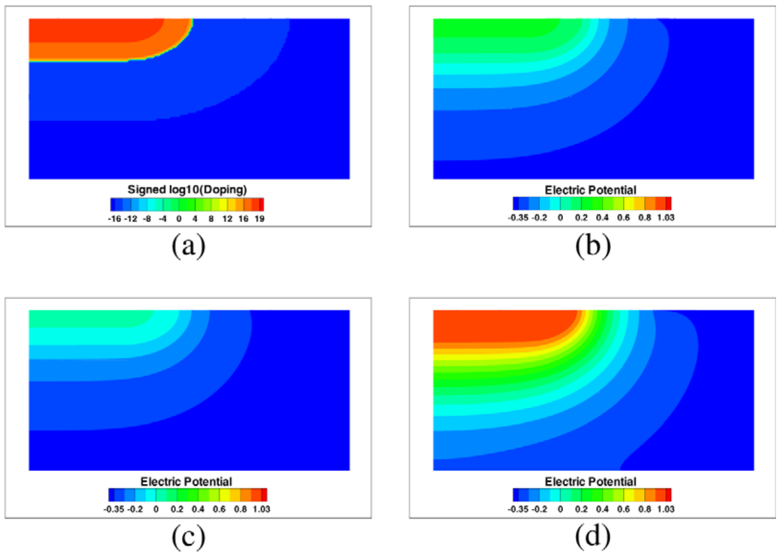


Fig. 5. (a) Signed logarithm of doping for the 1×0.5 μm diode with the two contacts on top; (b) electric potential at $t = 0.1$ (after the first time step); (c) Electric potential at $t = 0.3$; and (d) electric potential at $t = 0.8$.

Table 7
Comparison of one-level and three-level preconditioners for the transient 2D diode for three different mesh sizes and three different time steps, run on Red Storm. The four columns for each preconditioner list: average Newton steps per time step, average iterations per Newton step, average linear solve time per Newton step and average total time per Newton step.

Δt	Cores	Fine	1-level DD ILU				3-level: ILU/ILU/KLU agg60			
		unk	Newt/Δt	Iter/Newt	Lin sol time/N	Time/Newt	Newt/Δt	Iter/Newt	Lin sol time/N	Time/Newt
0.01	64	637K	2.9	251	8.6	12	2.9	89	3.9	7.1
	256	2.54M	2.9	488	28	31	2.9	106	5.3	8.5
	1024	10.2M	2.9	947	97	101	2.9	154	10	13
0.05	64	637K	4.2	247	8.4	12	4.1	83	3.6	6.7
	256	2.54M	4.2	488	28	31	4.1	100	5.0	8.1
	1024	10.2M	4.1	957	97	100	4.1	136	8.9	12
0.1	64	637K	4.8	271	9.9	13	4.8	84	3.7	6.8
	256	2.54M	4.8	537	33	36	4.8	98	4.9	8.0
	1024	10.2M	4.8	1052	115	119	4.8	129	8.4	12

6.3. *Effect of doping*

In this discussion the effect of an increasing material doping level on the performance of the multilevel preconditioner is briefly presented. In general as the material doping level increases the magnitudes of the dependent variables (ψ, n, p) and the gradients of the dependent variables increase for a given device [1,2]. However, due to the structure of the drift-diffusion system (4)–(6) where the doping, $C(\mathbf{x})$, appears as a source term in the potential equation, it is not readily apparent how this behavior will affect the conditioning of the associated linearized system that is to be iteratively solved in the Newton iteration. To intuitively illustrate this effect we consider a simplified linearized from of the electron transport Eq. (5) along with the potential Eq. (4) about the solution $(\bar{\psi}, \bar{n}, \bar{p})$. Assuming a one-dimensional problem with constant coefficients (λ, μ_n, D_n), carrying out an expansion of the drift term in the electron Eq. (5) by the chain rule, and substituting in $\nabla^2\psi$ from the potential Eq. (4) into the electron equation, we obtain

$$\frac{\partial n}{\partial t} + \bar{u}_n \frac{\partial n}{\partial x} - D_n \frac{\partial^2 n}{\partial x^2} + \left(\frac{\partial G}{\partial n}|_{\bar{n},\bar{p}} - \frac{\mu_n}{\lambda^2}[\bar{p} - \bar{n} + C]\right) n = 0$$

where $\bar{u}_n \equiv \mu_n \frac{\partial \bar{\psi}}{\partial x} = -\mu_n \bar{E}_x = \frac{-\mu_n}{\lambda^2} \int [\bar{p} - \bar{n} + C] dx$ is the drift velocity. In this formulation the linearized electron equation is in a standard transient convection–diffusion-reaction form. As the gradient of $\bar{\psi}$ increases the drift component of the transport increases relative to diffusion. In addition the reaction source term dependence on C is now readily apparent. Clearly, the material doping C will directly affect the eigenvalue structure and thereby the condition number of the linear system. However the nonlinearity and the coupling of the original drift-diffusion system would make a complete analytical understanding of this system difficult to obtain. For this reason a numerical study will be presented on the effect of the material doping for two test cases in order of increasing difficulty. The first is a one-dimensional NP diode with symmetric Gaussian doping on both sides of the junction, and the second is a two-dimensional NPN BJT. These test cases are used to qualitatively demonstrate the robustness and sensitivity of the multilevel preconditioner to the magnitude of the material doping. For the first test case, the position of the single isolated junction is fixed as the magnitude of Gaussian distribution of doping varies. For the second test case the position of the junctions in the 2D BJT can move as the doping magnitude is varied.

6.3.1. *One-dimensional diode*

The geometry for this test case is a 1×0.125 μm NP diode with the junction in the middle. n-doped material is to the left of the junction and p-doped material is to the right of the junction. The magnitudes and distribution of the n-doped material and the p-doped material are defined by symmetric Gaussians on both sides of the junction located at $x = 0.5$ μm, the first one centered at $x = 0.45$ μm and the second centered at $x = 0.55$ μm. This test case provides similar character to the 2D BJT since the BJT also uses a Gaussian-based doping profile. The diode is a more straightforward study since the junction location does not move and it is a single 1D isolated junction. Fig. 6 presents the material doping and a steady-state solution for the electric potential.

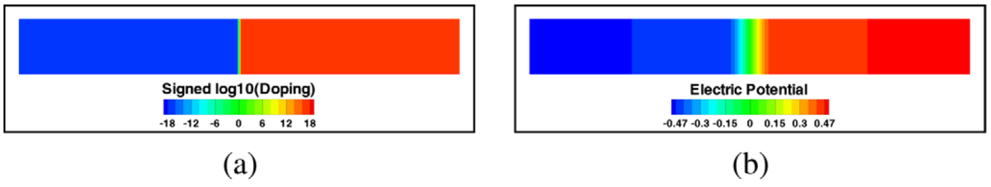


Fig. 6. (a) Signed logarithm of doping for 1×0.125 μm diode with symmetric Gaussian doping; and (b) corresponding steady-state electric potential for 1×0.125 μm diode for zero-bias case.

Table 8
Effect of doping on the three-level preconditioner for the 1D diode; each entry: average iterations [total Newton steps]/(average linear solve time); run on Red Storm machine.

Maximum	Unknowns on finest mesh/number of cores			
Doping	397K/16	1.58M/64	6.31M/256	25.2M/1024
10^{16}	55[2]/7.5	95[2]/13	138[2]/20	198[2]/33
10^{17}	49[3]/6.8	85[2]/12	119[2]/17	166[2]/27
10^{18}	44[3]/6.2	83[3]/11	117[3]/17	145[3]/23
10^{19}	37[4]/5.4	60[3]/8.5	104[3]/15	151[3]/25
10^{20}	–	52[4]/7.5	76[3]/11	113[3]/18

Table 8 shows the effect of the variation in the doping on a three-level preconditioner with coarser levels generated by taking 150 nodes per aggregate. One multigrid V(1,1)-cycle is used with ILU(2) as the smoother on the fine and medium levels and KLU on the coarsest level. The first column is the peak value of the Gaussian. A weak scaling study was performed with the value of the doping varied from 10^{16} to 10^{20} on a sequence of uniformly refined meshes from 1024×128 (397,000 unknowns) to 8192×1024 (25.2 million unknowns) with square mesh elements. The steady-state drift-diffusion solution was calculated for the zero bias case. Calculations were performed on Red Storm using both cores on a compute node. The linear solver tolerance was 10^{-6}. The values for each entry are: average Krylov iterations per Newton step[total number of Newton steps]/average linear solve time (including time to construct the preconditioner but not to construct the Jacobian) per Newton step.

If the mesh size is fixed, as the value of the doping is increased, the linear solve iterations decrease. (For the coarsest mesh at 10^{20} maximum doping there is insufficient mesh resolution with the uniform grid, and oscillations in the solution appear near the junction. For this reason we do not include these results in this preconditioner study.) There is a mild dependence of iteration count on maximum doping. This seems to be counterintuitive, because one would expect that the problem will become more difficult with larger doping because of the larger gradients. However as the doping is increased, the gradients become sharper and a smaller region is affected by the strong gradients. Although the number of linear solve iterations is decreasing, the number of nonlinear solver steps is increasing, which indicates that the problem is becoming more nonlinear with increased doping. Also, as the doping is increased, a finer mesh is needed to prevent oscillations so the problem becomes more expensive to solve. Over the four orders of magnitude increase on maximum doping there is only a factor of about two variation in the number of Krylov iterations to solution. This indicates a moderate to mild sensitivity and leads to a fairly robust preconditioner performance as a function of doping level.

6.3.2. *Two-dimensional BJT*

The second test case will examine the effect of the variation in maximum doping for the 2×1.5 μm NPN BJT test case presented earlier. In contrast to the diode just considered, where the location of the junction remained fixed as the doping was varied, for the BJT the location of the junctions can change as the maximum doping is varied. Table 9 shows the effect of the variation in the doping on a three-level preconditioner with coarser levels generated by taking 125 nodes per aggregate. One multigrid V(1,1)-cycle is used with ILU(2) as the smoother on the fine and medium levels and KLU solver on the coarsest level. The first column is the peak value of the Gaussian under the emitter. The peak value for the Gaussians for the base and collector are also multiplied by the same factor as the Gaussian under the emitter, i.e. when going from the 10^{16} to 10^{17} case, the peak value for all three Gaussians were multiplied by a factor of 10. The results for a 3520×2640 element mesh is presented (27.9M unknowns). 1024 Red Storm cores were used for these runs (both cores on a compute node). With this mesh resolution the highest magnitude doping case has no oscillations present. The values for each table entry are: average Krylov iterations per Newton step[total number of Newton steps]/average linear solve time (including the time to construct the preconditioner) per Newton step. Times are reported in seconds.

The trend for this scaling study as maximum doping is increased is more complex than for the 1D diode. Initially as doping increases the number of linear solve iterations increases until the maximum doping of 10^{19} case is reached. Then further

Table 9
Effect of doping on the three-level preconditioner for the 2D BJT; each entry: average iterations [total Newton step]/(average linear solve time); run on Red Storm machine.

Maximum Doping	27.9 million unknowns 1024 cores
10^{16}	151[3]/19
10^{17}	162[2]/21
10^{18}	233[2]/33
10^{19}	356[2]/59
10^{20}	293[3]/45
10^{21}	219[3]/32
10^{22}	136[4]/20

increase in the maximum doping leads to a decrease in the number of linear solve iterations. The number of Newton steps seems to increase as the problem is becoming more nonlinear.

6.4. Algorithmic scaling study comparing one-level and three-level Schwarz preconditioner for a two-dimensional NPN BJT: Red Storm vs. Purple

Finally, to demonstrate that the results presented previously are not particularly sensitive to the architecture of the computer, comparisons will be performed between the Red Storm and Purple platforms. Purple has many architectural differences compared with Red Storm, for example: eight-socket compute nodes vs. single-socket compute nodes, switch vs. 3D mesh interconnect topology and full Unix kernel vs. lightweight kernel on compute nodes.

Table 10 shows a comparison of results between the Red Storm and Purple machines for both a one-level DD ILU preconditioner and a three-level V(1,1) preconditioner for the steady-state drift-diffusion solution of the 2D 2×1.5 µm NPN BJT with 0.3V bias. For the three-level preconditioner, the coarser levels were generated by taking 85 nodes per aggregate, and ILU(2) with one-level of overlap was the smoother on the fine and medium levels with KLU direct solver on the coarsest level. For the 4096 processor case, the fine, medium, and coarse levels had 112 million, 1.31 million and 15,400 unknowns, respectively. “Avg iter/N” is the average number of Krylov iterations per Newton step and “time” is the average linear solve time plus Jacobian construction time per Newton step in seconds. The average time to construct the Jacobian for the calculations in this table was about 7 s for Red Storm and 23–26 s for Purple (about 27,000 unknowns per processor). The comparison is made with an equal number of cores per socket and therefore for Red Storm one core per socket (or compute node) is used and for Purple one core per socket is also used (the Purple sockets are actually dual core but only one core is active to prevent contention for the L2 and L3 cache). Fig. 7 shows the comparison for a one-level DD ILU preconditioner and the three-level preconditioner. From the results given in Table 10 for the one-level preconditioner, Red Storm is faster for

Table 10
Weak scaling study comparing one-level and three-level preconditioners for the 2D NPN BJT on the Red Storm and Purple machines. “Time/N” is the average linear solve time per Newton step.

Proc	Fine grid unk	1-level DD ILU				3-level V(1,1) agg85			
		Red Storm		Purple		Red Storm		Purple	
		Avg iter/N	Time/N (s)	Avg iter/N	Time/N (s)	Avg iter/N	Time/N (s)	Avg iter/N	Time/N (s)
4	110K	68	9.98	68	26.6	38	10.9	39	26.0
16	438K	142	14.4	143	29.8	75	13.3	74	28.3
64	1.75M	287	28.5	288	40.1	127	17.8	128	33.5
256	6.98M	571	80.1	574	82.3	188	24.7	190	40.1
1024	27.9M	1145	286	1156	265	271	38.4	273	52.9
4096	112M	2264	1081	2263	882	381	71.0	380	86.6

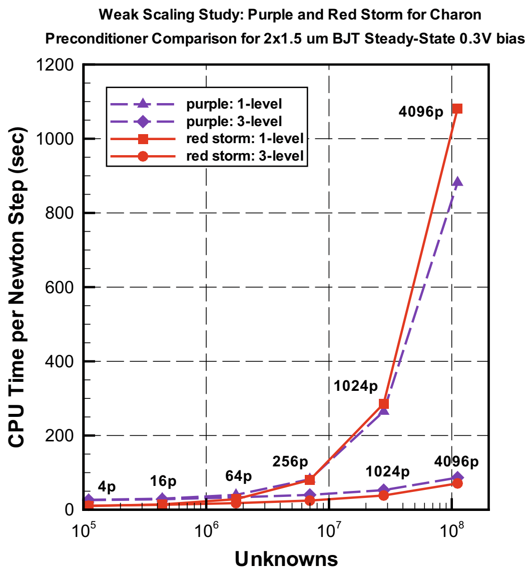


Fig. 7. Weak scaling study comparing CPU time between Red Storm and Purple for the one-level and three-level preconditioner for the 2D NPN BJT.

256 or fewer processors. For the 1024 and 4096 processor cases, Purple is faster. The existence of these two regimes is due to the slower Jacobian construction time on Purple that is important for smaller problem sizes. For the three-level preconditioner, Red Storm is faster for all the different numbers of processors. If one considers only the time to perform the linear solve for the one-level preconditioner, Purple is faster than Red Storm (20% faster for the 112 million unknown case). For the three-level preconditioner, the difference between the two platforms is small, with Purple having the slight edge on the linear solver time (i.e. not including Jacobian construction time). However when one is comparing total run time on the two different platforms, the time to construct the Jacobian needs to be considered.

7. Conclusions

This study compared the performance of a one-level and a three-level multilevel preconditioner for accelerating the linear solution for a Newton–Krylov solution method for the stabilized finite element discretization of the drift-diffusion equations for modeling semiconductor devices. The results demonstrated that the multilevel preconditioner provides a significant reduction both in the number of linear solve iterations and in solution time compared with the one-level method. For the largest problem considered the multilevel method was as much as twenty times faster than the one-level method. Although the three-level multilevel preconditioner was not strictly scalable, this reduction in CPU time obtained from an established existing multilevel preconditioning strategy is significant. Studies of the effect in variation of the doping on the performance of the multilevel preconditioner demonstrated the preconditioner to be reasonably robust to variation in maximum doping. While the current results for the parallel solution of large-scale semiconductor simulations by this multilevel preconditioner are encouraging, further investigation to improve the scalability of the method is in progress.

Acknowledgments

The authors would like to thank Richard Drake, Russell Hooper, Jonathan Hu and Eric Phipps for their contributions to this work. The authors would also like to thank Roger Pawlowski, Joseph Castro, Deborah Fixel, and Eric Keiter for their collaborative effort in developing the Charon code. Finally, the first author is very grateful to the SNL Red Storm team for all its help, especially Suzanne Kelly, Robert Ballance and Michael Davis, and the LLNL Purple team.

References

[1] Kevin M. Kramer, W. Nicholas, G. Hitchon, Semiconductor Devices, A Simulation Approach, Prentice-Hall PTR, 1997.
[2] S.M. Sze, Physics of Semiconductor Devices, second ed., John Wiley & Sons, 1981.
[3] D.L. Scharfetter, H.K. Gummel, Large-signal analysis of a silicon read diode oscillator, IEEE Trans. Electron Dev. 16 (1) (1969) 64–77.
[4] H.K. Gummel, A self-consistent iterative scheme for one-dimensional steady state transistor calculations, IEEE Trans. Electron Dev. ED-11 (1964) 455–465.
[5] Medici two-dimensional device simulation program user manual, Technical Report, Synopsys, February 2003.
[6] Davinci three-dimensional device simulation program manual type, Technical Report, Synopsys, February 2003.
[7] J.C. Meza, R.S. Tuminaro, A multigrid preconditioner for the semiconductor equations, SIAM J. Sci. Comput. 17 (1) (1996) 118–132.
[8] T. Clees, AMG strategies for PDE Systems with applications in industrial semiconductor simulation, Ph.D. Thesis, Universität zu Köln, 2005.
[9] J. Molenaar, P.W. Hemker, A multigrid approach for the solution of the 2D semiconductor equations, Impact Comput. Sci. Eng. 2 (1990) 219–243.
[10] D.A. Knoll, D.E. Keyes, Jacobian-free Newton–Krylov methods: a survey of approaches and applications, J. Comput. Phys. 193 (2004) 357–397.
[11] P.T. Lin, M. Sala, J.N. Shadid, R.S. Tuminaro, Performance of fully-coupled algebraic multilevel domain decomposition preconditioners for incompressible flow and transport, Int. J. Numer. Meth. Eng. 67 (2) (2006) 208–225.
[12] M. Sala, J.N. Shadid, R.S. Tuminaro, An improved convergence bound for aggregation-based domain decomposition preconditioners, SIAM J. Matrix Anal. 27 (3) (2006) 744–756.
[13] J.N. Shadid, R.S. Tuminaro, K.D. Devine, G.L. Hennigan, P.T. Lin, Performance of fully-coupled domain decomposition preconditioners for finite element transport/reaction simulations, J. Comput. Phys. 205 (1) (2005) 24–47.
[14] M. Sala, P. Lin, R. Tuminaro, J. Shadid, Algebraic multilevel preconditioners for nonsymmetric PDEs on stretched grids, in: O. Widlund, D. Keyes (Eds.), Domain Decomposition Methods in Science and Engineering XVI, Lecture Notes in Computational Science and Engineering, vol. 55, Springer-Verlag, 2007, pp. 741–748.
[15] P. Vaněk, J. Mandel, M. Brezina, Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems, Computing 56 (1996) 179–196.
[16] P. Vaněk, M. Brezina, J. Mandel, Convergence of algebraic multigrid based on smoothed aggregation, Numer. Math. 88 (2001) 559–579.
[17] G. Karypis, V. Kumar, Multilevel k-way partitioning scheme for irregular graphs, Technical Report 95-064, Department of Computer Science, University of Minnesota, 1995.
[18] G. Karypis, V. Kumar, ParMETIS: parallel graph partitioning and sparse matrix ordering library, Technical Report 97-060, Department of Computer Science, University of Minnesota, 1997.
[19] P.M. De Zeeuw, Nonlinear multigrid applied to a one-dimensional stationary semiconductor model, SIAM J. Sci. Stat. Comput. 13 (2) (1992) 512–530.
[20] R.E. Bank, H.D. Mittelmann, Continuation and multi-grid for nonlinear elliptic systems, in: W. Hackbusch, U. Trottenberg (Eds.), Multigrid Methods II Proceedings of the 2nd European Conference on Multigrid Methods, Lecture Notes in Mathematics, vol. 2, Springer-Verlag, 1986, pp. 23–37.
[21] T.J.R. Hughes, A. Brooks, A theoretical framework for Petrov–Galerkin methods with discontinuous weighting functions: application to the streamline-upwind procedure, in: R.H. Gallagher et al. (Eds.), Finite Elements in Fluids, vol. 4, J. Willey & Sons, 1982. pp. 47–65.
[22] T.J.R. Hughes, M. Mallet, A. Mizukami, A new finite element formulation for computational fluid dynamics: II. Beyond SUPG, Comput. Meth. Appl. Mech. Eng. 54 (1986) 341–355.
[23] F. Shakib, Finite element analysis of the compressible Euler and Navier–Stokes equations, Ph.D. Thesis, Division of Applied Mathematics, Stanford University, 1989.
[24] M. Sharma, G.F. Carey, Semiconductor device simulation using adaptive refinement and flux upwinding, IEEE Trans. Computer-Aided Des. 8 (6) (1989) 590–598.
[25] G.F. Carey, A.L. Pardhanani, S.W. Bova, Advanced numerical methods and software approaches for semiconductor device simulation, Technical Report SAND2000-0763J, Sandia National Laboratories, 2000.

[26] R. Codina, Comparison of some finite element methods for solving the diffusion–convection-reaction equations, Comput. Meth. Appl. Mech. Eng. 156 (1998) 185–210.
[27] J. Donea, A. Huerta, Finite Element Methods for Flow Problems, John Wiley & Sons, Ltd., 2003.
[28] J.N. Shadid, G.L. Hennigan, P.T. Lin, R.J. Hoekstra, Performance of stabilized finite element methods for solution of the drift-diffusion equations of semiconductor modeling, in preparation.
[29] T.J.R. Hughes, M. Mallet, A new finite element formulation for computational fluid dynamics: III.The generalized streamline operator for multidimensional advective–diffusive systems, Comput. Meth. Appl. Mech. Eng. 58 (1986) 305–328.
[30] P.N. Brown, Y. Saad, Hybrid Krylov methods for nonlinear systems of equations, SIAM J. Sci. Stat. Comput. 11 (3) (1990) 450–481.
[31] J.N. Shadid, A fully-coupled Newton–Krylov solution method for parallel unstructured finite element fluid flow, heat and mass transfer simulations, Int. J. CFD 12 (1999) 199–211.
[32] Y. Saad, Iterative Methods for Sparse Linear Systems, SIAM, 2003.
[33] A. Greenbaum, Iterative Methods for Solving Linear Systems, SIAM, Philadelphia, PA, USA, 1987.
[34] S.C. Eisenstat, H.F. Walker, Globally convergent inexact Newton methods, SIAM J. Optim. 4 (1994) 393–422.
[35] B. Smith, P. Bjorstad, W. Gropp, Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations, Cambridge University Press, 1996.
[36] A. Quarteroni, A. Valli, Decomposition Methods for Partial Differential Equations, Oxford University Press, 1999.
[37] X.-C. Cai, M. Sarkis, A restricted additive Schwarz preconditioner for general sparse linear systems, SIAM J. Sci. Comput. 21 (1999) 792–797.
[38] R.S. Tuminaro, M. Heroux, S.A. Hutchinson, J.N. Shadid, Aztec user's guide – version 2.1, Technical Report SAND99-8801J, Sandia National Laboratories, Albuquerque NM, 87185, Nov. 1999.
[39] M. Dryja, O.B. Widlund, Towards a unified theory of domain decomposition algorithms for elliptic problems, in: T.F. Chan, R. Glowinski, J.Périaux, O. Widlund (Eds.), Third International Symposium on Domain Decomposition Methods for Partial Differential Equations, SIAM, Philadelphia, PA, 1990, pp. 3–21.
[40] J. Xu, Iterative methods by space decomposition and subspace correction: a unifying approach, SIAM Rev. 34 (4) (1992) 581–613.
[41] J. Dendy, Black box multigrid for nonsymmetric problems, Appl. Math. Comput. 13 (1983) 261–283.
[42] M. Sala, R.S. Tuminaro, A new Petrov–Galerkin smoothed aggregation preconditioner for nonsymmetric linear systems, SIAM J. Sci. Stat. 31 (2008) 143–166.
[43] R. Tuminaro, C. Tong, Parallel smoothed aggregation multigrid: aggregation strategies on massively parallel machines, in: J. Donnelley (Ed.), SuperComputing 2000 Proceedings, 2000.
[44] H.H. Kim, J.C. Xu, L. Zikatanov, A multigrid method based on graph matching for convection–diffusion equations, Numer. Linear Algebra Appl. 10 (1–2) (2003) 181–195.
[45] O. Axelsson, Iterative Solution Methods, Cambridge University Press, Cambridge, 1994.
[46] R.S. Varga, Matrix Iterative Analysis, Prentice-Hall, Englewood Hills, NJ, 1962.
[47] B. Hendrickson, R. Leland, The Chaco user's guide-version 1.0, Technical Report SAND93-2339, Sandia National Laboratories, Albuquerque NM, 87185, 1993.
[48] M. Sala, M. Heroux, Robust algebraic preconditioners with IFPACK 3.0, Technical Report SAND2005-0662, Sandia National Laboratories, 2005.
[49] M. Heroux, AztecOO user guide, Technical Report SAND2007-3796, Sandia National Laboratories, 2007.
[50] T.A. Davis, Direct Methods for Sparse Linear Systems, SIAM, 2006.
[51] M.W. Gee, C.M. Siefert, J.J. Hu, R.S. Tuminaro, M.G. Sala, ML 5.0 smoothed aggregation user's guide, Technical Report SAND2006-2649, Sandia National Laboratories, 2006.
[52] M. Heroux, R. Bartlett, V. Howle, R. Hoekstra, J. Hu, T. Kolda, R. Lehoucq, K. Long, R. Pawlowski, E. Phipps, A. Salinger, H. Thornquist, R. Tuminaro, J. Willenbring, A. Williams, An overview of trilinos, Technical Report SAND2003-2927, Sandia National Laboratories, 2003.
[53] G.L. Hennigan, R.J. Hoekstra, J.P. Castro, D.A. Fixel, J.N. Shadid, Simulation of neutron radiation damage in silicon semiconductor devices, Technical Report SAND2007-7157, Sandia National Laboratories, 2007.
[54] I.D. Mayergoyz, Solution of the nonlinear Poisson equation of semiconductor device theory, J. Appl. Phys 59 (1) (1986) 195–199.