A GPU-Accelerated Parallel Preconditioner for the Solution of the Boltzmann Transport Equation for Semiconductors

Karl Rupp[1,2], Ansgar Jüngel[1], and Tibor Grasser[2]

[1] Institute for Analysis and Scientific Computing, TU Wien
Wiedner Hauptstraße 8–10, A-1040 Wien, Austria
[2] Institute for Microelectronics, TU Wien
Gußhausstraße 27-29, A-1040 Wien, Austria
rupp@iue.tuwien.ac.at

Abstract. The solution of large systems of linear equations is typically achieved by iterative methods. The rate of convergence of these methods can be substantially improved by the use of preconditioners, which can be either applied in a black-box fashion to the linear system, or exploit properties specific to the underlying problem for maximum efficiency. However, with the shift towards multi- and many-core computing architectures, the design of sufficiently parallel preconditioners is increasingly challenging.

This work presents a parallel preconditioning scheme for a state-of-the-art semiconductor device simulator and allows for the acceleration of the iterative solution process of the resulting system of linear equations. The method is based on physical properties of the underlying system of partial differential equations and results in a block preconditioner scheme, where each block can be computed in parallel by established serial preconditioners. The efficiency of the proposed scheme is confirmed by numerical experiments using a serial incomplete LU factorization preconditioner, which is accelerated by one order of magnitude on both multi-core central processing units and graphics processing units with the proposed scheme.

1 Introduction

With the introduction of multi-core central processing units (CPUs) in average desktop computers as well as the use of graphics processing units (GPUs) for general purpose computations, established serial algorithms need to be adjusted or even replaced by parallel variants. In particular, it can be very challenging to use the massively parallel architecture of GPUs with hundreds of threads even for standard algorithms like matrix-matrix multiplications efficiently [13].

While impressive performance for linear algebra operations can be obtained on GPUs, there are concerns about the use of GPUs from a productivity point of view [2]. While in some cases OpenMP [15] allows for a parallelization of existing code by adding a few lines of code only, GPU programming using OpenCL [11]

R. Keller et al. (Eds.): Facing Multicore-Challenge II 2011, LNCS 7174, pp. 147–157, 2012.

or CUDA [14] requires a deep understanding of the underlying GPU computing architecture and often a complete redesign of existing CPU-based code. Consequently, the shorter execution times may not balance the increased development effort.

While existing GPU libraries tend to provide only basic LAPACK-style functionality, our C++ library ViennaCL [19] provides high-level access to the vast computing resources of multi-core CPUs and GPUs using OpenCL. The application programming interface is compatible with uBLAS from the peer-reviewed Boost libraries [1] and thus hides the details of the GPU computing hardware from the user, while providing convenient use and high performance computations. Like other GPU libraries such as CUBLAS [14] and MAGMA [12], ViennaCL provides BLAS level 1, 2 and 3 routines for dense linear algebra operations. However, the focus is on sparse matrices and iterative solvers as well as high usability, which is also the case for the CUDA-based Cusp library [3]. ViennaCL targets shared memory systems and can be run on multiple GPUs. An investigation of dense matrix-matrix multiplications on heterogeneous distributed memory architectures using MPI has already been carried out [21] and shown that a distribution of the problem at hand should be accomplished on a higher level in order to keep communication overhead under control.

For the solution of partial differential equations, discretization schemes like the finite element, the finite difference or the finite volume method lead to large systems of linear equations, for which iterative solvers are typically employed [17]. The efficiency of such iterative solution schemes depends on the condition number of the underlying system matrix. Therefore, preconditioners often need to be employed in order to obtain a good convergence rate. However, the design of good parallel preconditioners can be very challenging and problem-specific [18]. Parallel black-box preconditioners for ViennaCL are in preparation, but they typically come at the expense of spectral efficiency compared to – possibly serial – problem-specific techniques. The successful implementation of rather complex preconditioners for GPUs is continuously reported by a number groups in various fields, e.g. [7] for algebraic multigrid methods, [22] for a factored sparse approximate inverse technique or [8] for results on a multi-colored incomplete LU (ILU) factorization.

In this work we present a parallel preconditioner for our state-of-the-art semiconductor device simulator. We demonstrate that only a single additional compute kernel added to the functionality already provided in ViennaCL allows to reduce execution times by about one order of magnitude compared to a single-threaded execution. This readily shows that a library-centric design of GPU-based algorithms allows for short code development times, while leveraging the full power of multi-core CPUs and GPUs.

We give an overview of the simulator in Sec. 2. The block preconditioning scheme we have recently derived from the underlying problem formulation is motivated, detailed and discussed in Sec. 3, 4 and 5, which is the key for the parallelization of the iterative solver. Results are discussed in Sec. 6 and a conclusion is drawn in Sec. 7.

2 A Deterministic Solution Approach for the Boltzmann Transport Equation for Semiconductors

The Boltzmann transport equation (BTE) for semiconductors is given by

$$\frac{\partial f}{\partial t} + \boldsymbol{v} \cdot \nabla_{\boldsymbol{x}} f + \boldsymbol{F} \cdot \nabla_{\boldsymbol{p}} f = Q\{f\} \tag{1}$$

and commonly considered to be the best semi-classical description of carrier transport in semiconductors. Here, $f(\boldsymbol{x}, \boldsymbol{p}, t)$ denotes the distribution function of carriers in the device with respect to the spatial location $\boldsymbol{x}$, momentum $\boldsymbol{p}$ and time t. The velocity $\boldsymbol{v}$ is given by the energy band structure of the material, the force $\boldsymbol{F}$ is obtained from the electrostatic potential, and the scattering operator $Q\{f\}$ is given in low-density approximation as

$$Q\{f\} = \int_{\mathcal{B}} S(\boldsymbol{p}', \boldsymbol{p}) f(\boldsymbol{p}') - S(\boldsymbol{p}, \boldsymbol{p}') f(\boldsymbol{p}) \, \mathrm{d}\boldsymbol{p}' \ , \tag{2}$$

where $\mathcal{B}$ denotes the Brillouin-zone of the material and $\boldsymbol{S}(\cdot, \cdot)$ denotes the scattering rate from one state to another.

The high dimensionality of the problem as well as the integro-differential nature make the solution of the BTE very challenging. While the stochastic Monte Carlo method has been the method of choice for a long time, the spherical harmonics expansion (SHE) method has become an increasingly attractive alternative. Here, the momentum-part of the distribution function is expanded into spherical harmonics $Y_{l,m}$ as

$$f(\boldsymbol{x}, \boldsymbol{p}, t) \cong \sum_{l=0}^{L} \sum_{m=-l}^{l} f_{l,m}(\boldsymbol{x}, H, t) Y_{l,m}(\theta, \varphi) \ , \tag{3}$$

where H denotes total energy. The series is truncated at a finite expansion order L, typically $L \in \{1, 3, 5\}$. In the following, only the steady-state is considered.

While the application of the SHE method has long been restricted to one-dimensional device simulations due to high memory requirements, enough memory is available on modern computers to allow for two-dimensional device simulations [9]. While fully parallel implementations of the Monte Carlo method have already been reported [23], an artificial restriction of the SHE method to a single CPU core would be detrimental to the attractiveness of the method.

The SHE method ultimately leads to the solution of large systems of linear equations for the expansion coefficients $f_{l,m}$ in the $(\boldsymbol{x}, H)$ space. A nonlinear iteration scheme is typically employed to ensure self-consistency of the BTE with the Poisson equation describing the electrostatic potential. As discussed by Jungemann *et al.* [10], the indefinite system of linear equations resulting from the SHE equations requires a good preconditioner in order to obtain convergence of iterative solvers. This is in contrast to many other problem classes, where e.g. the positive definiteness of the system matrix typically ensures convergence, even if the number of iterations required might be large. In recent publications on the

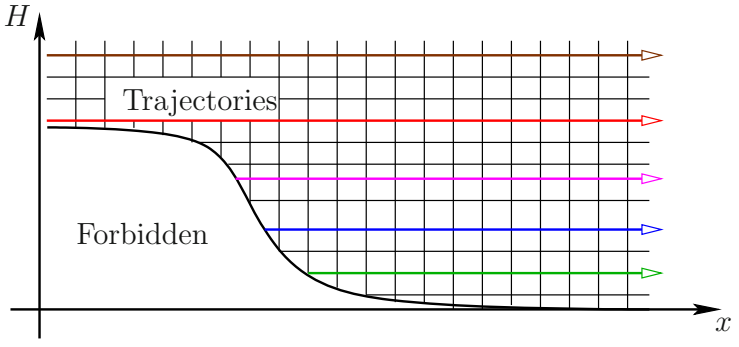

Fig. 1. Trajectories of carriers in free flight within the device are given by constant total energy H. Carriers can change energy only through inelastic scatting, which is instant in time and localized space.

SHE method [9,10], an ILU preconditioner was used for that purpose. ILU is a widely accepted black-box preconditioner [17], but in its pure form restricted to single-threaded execution. Even though parallel block-variants of ILU as well other parallel preconditioning techniques such as sparse approximate inverses [6] or polynomial preconditioners have been developed, their convergence enhancement can be typically considerably lower than for single-threaded variants [17,18].

3 Physics-Based Block-Preconditioning

To obtain a set of equations for the unknown expansion coefficients $f_{l,m}$ in (3), the BTE is formally projected onto the individual spherical harmonics $Y_{l,m}$. In operator form, the SHE equations in steady state can then be written as

$$L_{l,m}\{f\} = Q_{l,m}\{f\} \ , \quad l = 0, \ldots, L, \quad m = -l, \ldots, l \ ,$$

where $L_{l,m}$ and $Q_{l,m}$ denote the projections of the streaming operator and the scattering operator onto the spherical harmonics $Y_{l,m}$ respectively. Employing the H-transform [9,5], carrier trajectories in free flight are given by hyperplanes of constant total energy H in the simulation domain $(\boldsymbol{x}, H)$, cf. Fig. 1. This is reflected in the model by the fact that $L_{l,m}$ does not couple any of the, say, N_{H} different energy levels in the simulation domain.

Carriers within the device can change their total energy only by inelastic scattering events, thus the scattering operator $Q_{l,m}\{f\}$ is responsible for coupling different energy levels. However, if only elastic scattering processes are considered, the total energy of the involved particles remains unchanged and the different energy levels do not couple. Therefore, in a SHE simulation using only elastic scattering and N_{H} different energy levels, the resulting system of linear equations is consequently decoupled into N_{H} independent problems. Such

a decomposition has been observed already in early publications on SHE [4], but it has been of no practical relevance since inelastic scattering processes are essential for predictive device simulation.

Inelastic scattering processes like optical phonon scattering couple different energy levels. As devices are scaled down, the average number of scattering events of a carrier while moving through the device decreases and weakens the coupling between different energy levels. At the algebraic level this can be reasoned as follows: Using a box integration scheme as proposed by Hong *et al.* [9], the volume integral over the free streaming operator $L_{l,m}$ is transformed to a surface integral due to the divergence operator with respect to the spatial variable $\boldsymbol{x}$. Therefore, if the typical device length d is scaled to $d' := \alpha d$ with $0 < \alpha < 1$, the contributions from the free streaming operator scale as α^{n-1}, where n denotes the spatial dimension considered in the simulation. However, the scattering terms are obtained by an integration over the control volume, which scales as α^n. Therefore, in the limit of extremely scaled devices, the coupling between different energy levels is negligible.

While the preconditioner is motivated by physical arguments at the continuous level, a discretization still has to be employed. Following the discretization proposed by Hong *et al.* [9], the even order expansion coefficients are associated with grid nodes and represent densities, while odd order expansion coefficients are associated with edges and represent fluxes. Odd order expansion coefficients can be condensed [10,16], such that one finally obtains a linear system of equations for the even order expansion coefficients at discrete locations in the $(\boldsymbol{x}, H)$ domain. Let $\boldsymbol{S}_{\text{full}}$ denote this condensed system matrix and $\boldsymbol{S}_{\text{elastic}}$ the system matrix of the decoupled problem obtained in the same way by ignoring any contributions from inelastic scattering processes. Then we propose to construct the preconditioner $\boldsymbol{P}_{\text{full}}$ for $\boldsymbol{S}_{\text{full}}$ as

$$\boldsymbol{P}_{\text{full}} \approx (\boldsymbol{S}_{\text{full}})^{-1} \approx (\boldsymbol{S}_{\text{elastic}})^{-1} \approx \boldsymbol{P}_{\text{elastic}} \ . \tag{4}$$

Since the elastic problem is decoupled into N_{H} subproblems, $\boldsymbol{S}^{\text{elastic}}$ decomposes into N_{H} independent blocks. For each of these blocks, a (possibly serial) preconditioner can be efficiently set up as well as applied to the residual vector in parallel. Moreover, due to the decoupling of the system matrix into independent blocks, the proposed scheme is also perfectly suitable for distributed memory architectures.

4 Symmetrization of the Scattering Processes

Due to the exponential decay of the distribution function with respect to carrier energy, the scattering rate from higher energy to lower energy is much higher than vice versa. This asymmetry of inelastic scattering processes for energies H_i and H_j, $i < j$, with respect to energy manifests itself in the system matrix in the form of large values in the block with energy index (H_i, H_j), and small entries in the block (H_j, H_i), cf. Fig. 2. Therefore, the upper triangular part of the system matrix is populated with much larger values than the lower triangular part. It

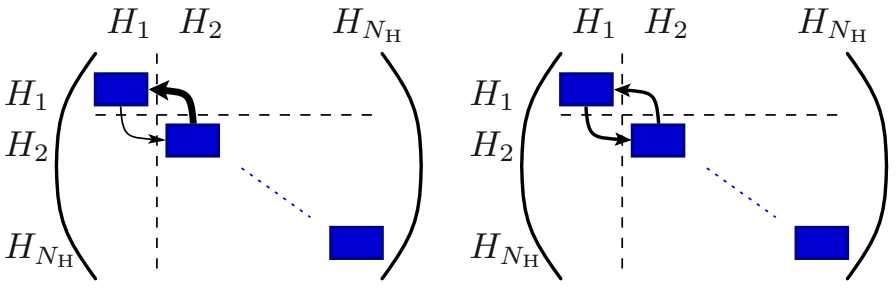

Fig. 2. Structure of the system matrix for total energy levels $H_1 < H_2 < \ldots < H_{N_H}$ before (left) and after (right) symmetrization. Unknowns at the same total energy H_i are enumerated consecutively, inducing a block-structure of the system matrix. For simplicity, scattering is depicted between energy levels H_1 and H_2 only, using arrows with thickness proportional to the magnitude of the entries. As devices are scaled down, the entries in off-diagonal blocks become small compared to the entries in the diagonal blocks.

should be noted that this asymmetry ensures that the equilibrium solution is a Maxwell (or more generally, a Fermi-Dirac) distribution.

The large values in the upper triangular part of the matrix are a hindrance for the construction of the preconditioner by neglecting off-diagonal blocks. We reduce this asymmetry by rescaling the unknowns of the discrete system according to the expected exponential decay. The new discrete unknowns $f'_{l,m}(\boldsymbol{x}_i, H_i, t)$ are obtained from the old discrete unknowns $f_{l,m}(\boldsymbol{x}_i, H_i, t)$ by

$$f'_{l,m}(\boldsymbol{x}_i, H_i, t) := \exp\left(\frac{\varepsilon_i}{k_\mathrm{B}T}\right) f_{l,m}(\boldsymbol{x}_i, H_i, t) \ , \tag{5}$$

where ε_i denotes the kinetic energy at point $(\boldsymbol{x}_i, H_i)$, k_B is the Boltzmann constant and T denotes a scaling temperature which is either set to room temperature, lattice temperature or can be seen as a numerical parameter. The benefit of this rescaling is that in equilibrium the primed unknowns are then of similar order and show little to no exponential behavior. It should be noted that the proposed rescaling can be written in matrix form equivalently as

$$\boldsymbol{S}\boldsymbol{f} = \boldsymbol{b} \qquad \Leftrightarrow \qquad \boldsymbol{S}\boldsymbol{D}\boldsymbol{f}' = \boldsymbol{b} \ ,$$

where $\boldsymbol{D}$ is a diagonal matrix with the diagonal terms given by the reciprocals of the exponentials in (5). The matrix $\boldsymbol{S}' := \boldsymbol{S}\boldsymbol{D}$ represents the system matrix with rebalanced off-diagonal scattering blocks. Here, symmetrization refers to rescaling the unknowns such that the entries in the off-diagonal blocks (H_i, H_j) and (H_j, H_i) are of similar magnitude – it does not denote symmetry of the system matrix in the strict mathematical sense.

5 Practical Considerations

For the construction of the preconditioner it is not necessary to set up another system matrix $\boldsymbol{S}_{\text{elastic}}$ explicitly. Since the contribution of inelastic scattering operators to the diagonal blocks is positive, it is of advantage to use the block diagonal of $\boldsymbol{S}_{\text{full}}$ for setting up the preconditioner. Thereby, extra memory for a second system matrix is avoided.

It has been observed in numerical experiments that the rescaling of unknowns leads to better results if the temperature T in (5) is set above room temperature. The physical interpretation is that carriers are heated in areas of large electric fields, thus having a lower exponential decay rate, which relates to a higher temperature. Good results are obtained with $T = 400\text{K}$ and only a low sensitivity of the number of iterations on the parameter T is observed.

The rows of the system matrix $\boldsymbol{S}'$ can be normalized prior to the block-factorization. This leads to a matrix $\boldsymbol{S}''$ given as

$$\boldsymbol{S}'' = \boldsymbol{E}\boldsymbol{S}' = \boldsymbol{E}\boldsymbol{S}\boldsymbol{D} \ ,$$

where the diagonal matrix $\boldsymbol{E}$ consists of the inverses of the row norms. Thus, a two-sided diagonal preconditioner is applied to the initial system matrix $\boldsymbol{S}$ before launching the block-preconditioning scheme.

Within ViennaCL, the call to the iterative solver is given in the mnemonic form

```
x = solve(A, b, solver_tag, precond);
```

where `A` denotes the system matrix, `x` and `b` the result and the right hand side vector respectively, the `solver_tag` allows to select the respective solver and `precond` specifies the optional preconditioner. For the case of SHE using the BiCGStab [20] iterative solver and the custom parallel block preconditioner, the respective code lines for the user are

```
x = solve(A, b, bicgstab_tag(), she_block_precond);
```

where `she_block_precond` is a functor that applies the preconditioner to the residual in each iterative solver step. Therefore, the user can focus all development efforts on the preconditioner only, without having to deal with other details of the underlying iterative solver. In particular, comparisons with the built-in ILU factorization with threshold (ILUT) preconditioner in ViennaCL are carried out by

```
//MatrixType denotes the type of the matrix A
ilut_precond<MatrixType> ilut(A, ilut_tag());
x = solve(A, b, bicgstab_tag(), ilut);
```

thus allowing a simple means to switch between different iterative solvers as well as different preconditioners.

6 Results

As a benchmark for the proposed block preconditioning scheme, we consider the spatially two-dimensional simulation of an n^+nn^+ diode with different lengths of the intrinsic region. ILUT is used as a preconditioner for each block. As outlined in Sec. 5, the same preconditioner is used as a single-threaded preconditioner for the full system matrix, since ILU-type preconditioners have been employed in other recent works. It has to be emphasized that the preconditioner used for each block in our scheme can be chosen arbitrarily, thus we aim at confirming the applicability of the physically motivated scheme only, since it then enables the use of any possibly serial preconditioner in a highly parallel fashion. BiCGStab [20] is used as linear solver, since it provides a lower memory footprint than the GMRES method [17] used in [10].

Execution times of the iterative BiCGStab solver are compared for a single CPU core using ILUT for the full system matrix, and for the proposed parallel scheme using multiple CPU cores of a quad-core Intel Core i7 960 CPU with eight logical cores. In addition, comparisons for a NVIDIA Geforce GTX 580 GPU are found in Figs. 3. The parallelization on the CPU is achieved using the Boost.thread library [1], and the same development time was allotted for the OpenCL kernel on the GPU. This allows for a comparison of the results not only in terms of execution speed, but also in terms of productivity.

As can be seen in Figs. 3, the performance increase for each linear solver step is more than one order of magnitude compared to the single-core implementation. This super-linear scaling with respect to the number of cores on the CPU is due to the better caching possibilities obtained by the higher data locality within the block-preconditioner.

The required number of iterations using the block-preconditioner decreases with the device size. For a 25 nm intrinsic region, the number of iterations is only twice than that of an ILUT preconditioner for the full system. At an intrinsic region of 200 nm, four times the number of iterations are required. This is a very small price to pay for the excellent parallelization possibilities.

Overall, the multi-core implementation is by a factor of three to ten faster than the single core-implementation even though a slightly larger number of solver iterations is required. The purely GPU-based solver with hundreds of simultaneous lightweight threads is by up to one order of magnitude faster than the single-core CPU implementation.

The comparison in Fig. 3 further shows that the SHE order does not have a notable influence on the block-preconditioner efficiency compared to the full preconditioner. The slightly larger number of solver iterations for third order expansions is due to the higher number of unknowns in the linear system. The performance gain is almost uniform over the length of the intrinsic region and slightly favors shorter devices, thus making the scheme an ideal candidate for current and future scaled-down devices.

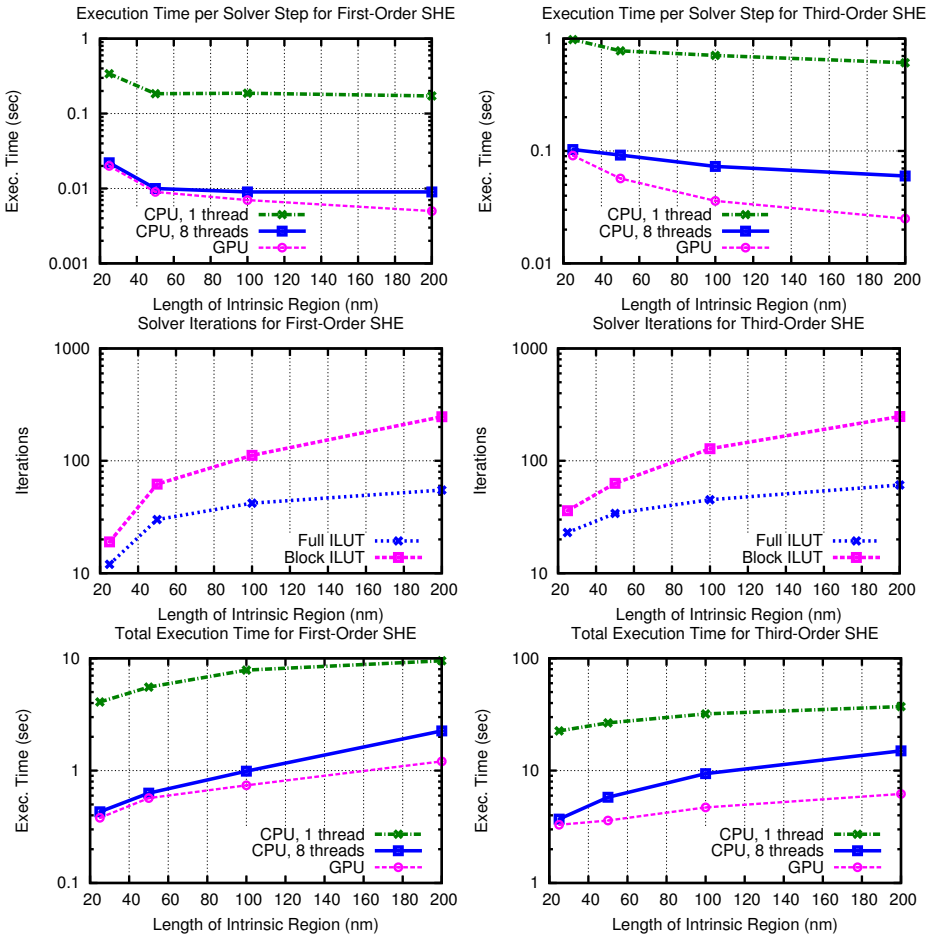

Fig. 3. Execution times per solver iteration, number of solver iterations and total solver execution time for a first-order (left) and a third-order (right) SHE simulation of n^+nn^+ diodes with different lengths of the intrinsic region. As expected from physical arguments, the parallel preconditioner performs the better the smaller the length of the intrinsic region gets. The GPU version performs particularly well for the computationally more challenging third-order SHE. A reduction of total execution times compared to a single-threaded implementation by one order of magnitude is obtained.

7 Conclusions

Our case-study of employing a problem-specific parallel preconditioner within ViennaCL for the acceleration of a semiconductor device simulator readily shows that library-centric design for algorithms on GPUs and multi-core CPUs based on OpenCL allows for high productivity. A development of the full GPU solver from scratch for the particular problem at hand would have resulted in devel-

opment effort that is at least an order of magnitude larger than a comparable implementation for multi-core CPUs in e.g. C++, while only a performance gain of about an additional factor of two would have been obtained.

The parallel block-preconditioning scheme is proposed and demonstrated to be very efficient especially for scaled-down devices. In contrast to black-box block preconditioners, the proposed scheme is based on a sound physical principle. The number of iterations compared to a single-threaded ILUT preconditioner for the full system matrix is two to four times as large, but this is only a minor price to pay for the huge degree of parallelism provided for the crucial preconditioning step. On the whole, an overall performance improvement of one order of magnitude is obtained for our test case.

Acknowledgment. K. Rupp and A. Jüngel acknowledge partial support from the Austrian Science Fund (FWF), grants P20214, P22108, and I395; the Austrian-Croatian Project HR 01/2010; the Austrian-French Project FR 07/2010; and the Austrian-Spanish Project ES 08/2010 of the Austrian Exchange Service (ÖAD). The authors gratefully acknowledge support by the Graduate School PDEtech at the TU Wien.

References

1. Boost C++ libraries, `http://www.boost.org/`
2. Bordawekar, R., Bondhugula, U., Rao, R.: Can CPUs Match GPUs on Performance with Productivity? Experiences with Optimizing a FLOP-intensive Application on CPUs and GPU. Technical report, IBM T. J. Watson Research Center (2010)
3. Cusp Library, `http://code.google.com/p/cusp-library/`
4. Gnudi, A., Ventura, D., Baccarani, G.: One-dimensional Simulation of a Bipolar Transistor by means of Spherical Harmonics Expansion of the Boltzmann Transport Equation. In: Proc. SISDEP, vol. 4, pp. 205–213 (1991)
5. Gnudi, A., Ventura, D., Baccarani, G., Odeh, F.: Two-dimensional MOSFET Simulation by Means of a Multidimensional Spherical Harmonics Expansion of the Boltzmann Transport Equation. Solid-State Electr. 36(4), 575–581 (1993)
6. Grote, M.J., Huckle, T.: Parallel Preconditioning with Sparse Approximate Inverses. SIAM J. Sci. Comput. 18, 838–853 (1997)
7. Haase, G., Liebmann, M., Douglas, C., Plank, G.: A Parallel Algebraic Multigrid Solver on Graphics Processing Units. In: Zhang, W., Chen, Z., Douglas, C.C., Tong, W. (eds.) HPCA 2009. LNCS, vol. 5938, pp. 38–47. Springer, Heidelberg (2010)
8. Heuveline, V., Lukarski, D., Weiss, J.P.: Enhanced Parallel ILU(p)-based Preconditioners for Multi-core CPUs and GPUs – The Power(q)-pattern Method. EMCL Preprint 2011-08, EMCL (2011)
9. Hong, S.M., Jungemann, C.: A Fully Coupled Scheme for a Boltzmann-Poisson Equation Solver based on a Spherical Harmonics Expansion. J. Comp. Electr. 8, 225–241 (2009)
10. Jungemann, C., Pham, A.T., Meinerzhagen, B., Ringhofer, C., Bollhöfer, M.: Stable Discretization of the Boltzmann Equation based on Spherical Harmonics, Box Integration, and a Maximum Entropy Dissipation Principle. J. Appl. Phys. 100(2), 024502–+ (2006)
11. Khronos Group. OpenCL, `http://www.khronos.org/opencl/`

12. MAGMA library, http://icl.cs.utk.edu/magma/
13. Nath, R., Tomov, S., Dongarra, J.: An Improved MAGMA GEMM For Fermi Graphics Processing Units. Intl. J. HPC Appl. 24(4), 511–515 (2010)
14. NVIDIA CUDA, http://www.nvidia.com/
15. OpenMP, http://openmp.org/
16. Rupp, K., Jüngel, A., Grasser, T.: Matrix Compression for Spherical Harmonics Expansions of the Boltzmann Transport Equation for Semiconductors. J. Comp. Phys. 229(23), 8750–8765 (2010)
17. Saad, Y.: Iterative Methods for Sparse Linear Systems, 2nd edn. Society for Industrial and Applied Mathematics (2003)
18. Vassilevski, P.S.: Multilevel Block Factorization Preconditioners. Springer, Heidelberg (2008)
19. ViennaCL, http://viennacl.sourceforge.net/
20. van der Vorst, H.A.: Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Non-Symmetric Linear Systems. SIAM J. Sci. and Stat. Comp. 12, 631–644 (1992)
21. Weinbub, J., Rupp, K., Selberherr, S.: Distributed Heterogenous High-Performance Computing with ViennaCL. In: Abstracts Intl. Conf. LSSC, pp. 88–90 (2011)
22. Xu, K., Ding, D.Z., Fan, Z.H., Chen, R.S.: FSAI Preconditioned CG Algorithm combined with GPU Technique for the Finite Element Analysis of Electromagnetic Scattering Problems. Finite Elem. Anal. Des. 47, 387–393 (2011)
23. Zang, W., Du, G., Li, Q., Zhang, A., Mo, Z., Liu, X., Zhang, P.: A 3D Parallel Monte Carlo Simulator for Semiconductor Devices. In: Proc. IWCE, pp. 1–4 (2009)