Accelerating Nano-scale Transistor Innovation with NEMO5 on Blue Waters

Harshad Sahasrabudhe*[1], Jim Fonseca†[3], and Gerhard Klimeck‡[2,3]

[1]Department of Physics and Astronomy, Purdue University
[2]School of Electrical and Computer Engineering, Purdue University
[3]Network for Computational Nanotechnology, Purdue University

March 31 2014

Abstract

This article details the extension of the NEMO5 nanoelectronics modeling software to take advantage of the unique characteristics of the Blue Waters system. The simulation capability of the NEMO family of software has evolved dramatically over the last two decades to include full device simulations, quantum transport and 3D atomistic simulations of electronic structure in nanostructures. With those capabilities come increased computational requirements. The NEMO5 software has been designed to run on large-scale CPU machines and has already been successful in this regard. This work extends NEMO5's flexibility to include execution on GPUs and heterogeneous CPU/GPU systems.

1 Introduction

Relentless downscaling of transistor size has continued according to Moore's law for the past 40 years. Transistor size will continue to decrease in the next ten years, but foundational issues with currently unknown technology approaches must be pursued [1]. This downscaling has reached the range where the number of atoms in critical dimensions is countable, geometries are formed in three dimensions and new materials are being investigated. As semiconductor devices scale to new dimensions, the materials and designs become more dependent on atomic details. Under these conditions we argue that the overall geometry constitutes a new material that cannot be found as such in nature [2]. Quantum effects such as tunneling, state quantization, and atomistic disorder dominate the characteristics of these nano-scale devices. NEMO5 is a nanoelectronics modeling package designed for comprehending the critical multi-scale, multi-physics phenomena through efficient computational approaches and quantitatively modeling new generations of nanoelectronic devices as well as predicting novel device architectures and phenomena [3].

2 Approach to NEGF

At the heart of NEMO5's quantum transport approach is the non equilibrium Green's function (NEGF) method which is a computational approach to handling quantum transport in nanoelectronic devices [4]. NEGF has been well established to provide a fundamental connection between the quantum mechanical (Schrödinger Equation) treatment of electrons and their non-equilibrium interactions with contact reservoirs and incoherent scattering mechanisms. NEGF is numerically expensive when applied on atomistic tight binding representations. The empirical tight binding approach models each atom individually as well as its interactions with nearest neighbors. Because

*hsahasra@purdue.edu
†jfonseca@purdue.edu
‡gekco@purdue.edu

of this, NEGF requires storage, inversion and multiplication of matrices on the order of the number of electronic degrees of freedom. Parametrization of atomic orbitals is by way of first principle theories, namely Density Functional Theory.

A well known method to ease the numerical burden is the recursive Green's function method (RGF) that allows for limiting the calculation and storage of the retarded Green's function to specific matrix blocks (such as only block diagonals and a single block column). Even with RGF, the computational time scales with the cross sectional area (to the direction of electron flow) cubed, and linearly with the length of the device. For example, a toy calculation of a 50 nm long wire with a 3 nm diameter requires for a single energy around 1 TFLOP. Resolution of a device's characteristics requires about 1,000 energy points, and this calculation must be repeated perhaps a dozen times for a full current-voltage sweep. The treatment of a technically relevant finFET device would require an atomistic resolution of a device with a cross section around $20 \times 40nm^2$, which includes the core semiconductor and the surrounding gate material. Therefore, it is apparent that efficient use of large heterogeneous machines is needed to accurately simulate devices of interest to the community.

2.1 RGF

The Recursive Green's Function algorithm (RGF)[5] is a scheme for solving the Non-Equilibrium Green's Functions (NEGF) for devices. The RGF method has already been implemented in NEMO5, however, this implementation relies on the Transfer Matrix method, which in turn is heavily dependent on solving the generalized eigenvalue problem. The Transfer Matrix method is one approach to solving RGF and was initially used for calculating self energy of the source and drain leads for the NEGF approach. The self energies describe the effect of the leads on the central device region, and were then used for calculating Hamiltonian of the device. The use of self energies of the leads makes the Hamiltonian non-Hermitian. The eigenvalues and eigenvectors of the device Hamiltonian need to be solved to obtain the electron wavefunctions and energies. However, the eigenvalue calculation (LAPACK's ZGEEV) of a general matrix does not scale well on multi-threaded/distributed systems–in our case GPUs. Thus, the GPU-focused work required a shift to a different implementation of RGF which utilized only multiplication and inversion of matrices. The Sancho Rubio[6, 7] and General Leads methods can be used to calculate the self energy of the leads instead of Transfer Matrix method. These methods primarily use matrix multiplication and inversion and are therefore good substitutes for Transfer Matrix method when scalability on heterogeneous systems is desired.

2.2 Sancho Rubio method

Sancho Rubio is an iterative, computationally intensive method with low memory usage which is prime for GPUs. Self energy of the leads is required for calculating the transmission and charge density in the device using NEGF. Self energy gives rise to open boundary conditions and is calculated by assuming semi-infinite leads. Transfer Matrix and General Leads algorithms recursively solve NEGF equations for adjacent slabs of atoms in the leads to obtain self energy. After a reasonable number of iterations, the self energy resembles that of a semi-infinite lead. The Sancho Rubio method [6][7] uses renormalization on NEGF equations and thus requires much fewer iterations to get a self energy to the same degree of accuracy.

Forward iteration in the NEGF equations or "Forward RGF" involves sparse-dense matrix multiplication and matrix inversion. Similarly, Sancho Rubio method requires only these functions. It is thus inherently compute intensive and well-suited for calculation on GPUs. It also requires storage of only 13 dense matrices making possible the calculation of self energy for devices of large cross-sections on GPUs.

3 Interfaces to External Libraries

3.1 Numerical Algorithms

Allowing NEMO5 to use the resources on Blue Waters efficiently requires enabling NEMO5 to use GPUs on the Cray XK7 nodes. It is known that dense matrix multiplication and matrix inversion get respectable FLOPs on GPU[8, 9]. Proper usage of GPUs can be made by scalable algorithms which primarily use dense/sparse-dense matrix-matrix multiplication and matrix inversion. Thus, the RGF algorithm discussed in section 2.1 was employed. NEMO5 is already heavily leveraging large-scale community efforts such as PETSc, SLEPc, Python, and libMesh. These libraries allowed a focus on code generality (more physics) and portability. Numerical solvers are isolated from the physical models through object-oriented software development principles.

An initial approach to add GPU capability on a distributed heterogeneous system was to implement interfaces to libraries such as MAGMA, CUSP, PARSEC and cuSPARSE. Since NEMO5 relied completely on the library PETSc for handling matrices, the plan was to add these interfaces in PETSc and an interface to the MAGMA library in PETSc was built. The interface only handled LU factorization of dense matrices. Work is ongoing to expand that interface to include more MAGMA functions such as linear system solution, matrix multiplication etc.

To enable GPU support in PETSc, it needed to be compiled with CUDA and CUSP libraries. To use GPU capabilities, special matrix formats have been developed by the PETSc development team. Also, the usual complex datatype

```
std::complex<double>
```

is replaced by

```
cusp::complex<double>
```

in PETSc compiled with CUSP. This created problems because of datatype mismatch errors while compiling NEMO5. Reinterpret cast was used as a workaround.

```
std::complex<double> std_val;
cusp::complex<double> cusp_val =
      reinterpret_cast<cusp::complex<double> >(std_val);
```

This approach solved the compilation errors. NEMO5 was compiled and linked with two instances of PETSc (compiled with CUDA+CUSP), one each for real and complex values, a unique usage of PETSc. While trying to initialize 2 such PETSc instances, a run time error was thrown because of multiple initializations of the GPU from a single process. A solution for this error requires development by the PETSc team and should be available in the forthcoming PETSc version 3.5. This drawback meant that the initial approach of using PETSc to handle matrices on GPUs was not going to work until PETSc 3.5. MAGMA was interfaced directly with NEMO5 to solve these issues. Figure 1 describes this approach.

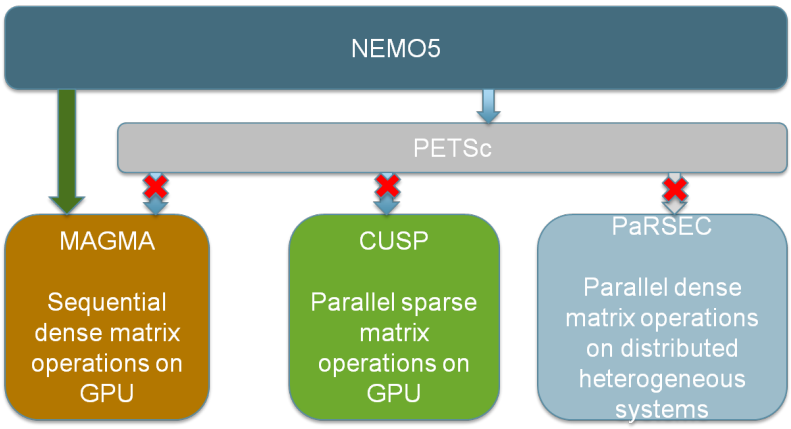
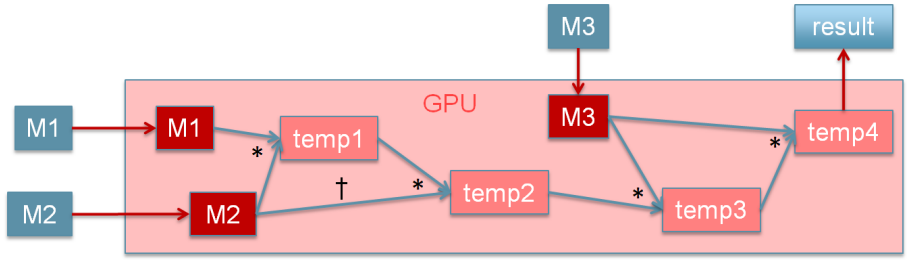


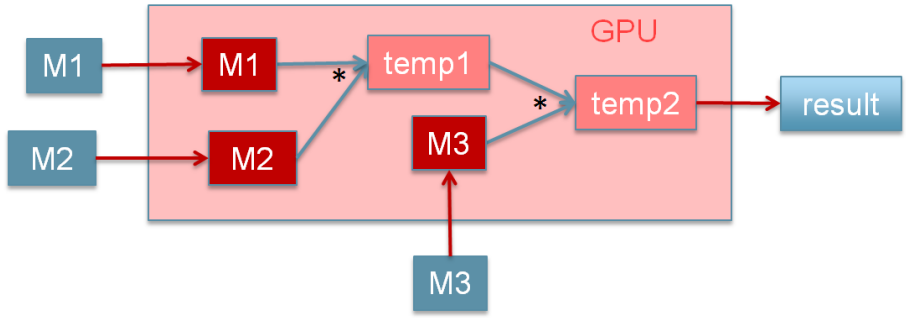
Figure 1: Interfacing external libraries with NEMO5 for GPU support.

3.2 MAGMA Interface

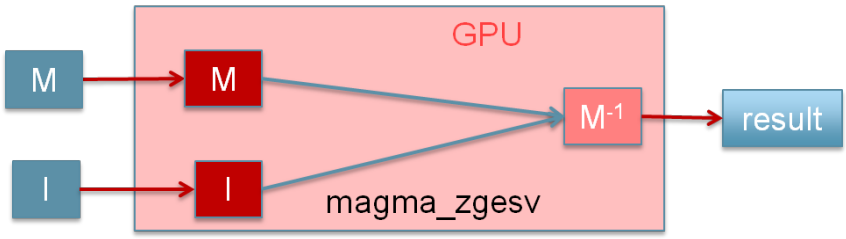
The RGF algorithm heavily relies on matrix-matrix multiplication and matrix inversion. Typically on a single CPU running the RGF algorithm, 37% of the total time is taken by matrix inversion and 26% is taken by matrix multiplication, so these algorithms were chosen as the first candidates for GPU offload. Thus, the MAGMA [8] library was interfaced directly in NEMO5. MAGMA is a library of next generation linear algebra GPU accelerated libraries for heterogeneous GPU-based architectures, and it supports interfaces with current linear algebra packages and standards such as LAPACK and BLAS. MAGMA allows applications to fully exploit heterogeneous CPU/GPU systems. A multiplication scheme allowed for the storage of matrices on the GPU so that data transfer to and from the GPU is minimized. Figure 2 describes how multiplications and inversions are offloaded, computed and the results downloaded. The resulting speedup can be seen in Figure 3.



(a) $result = M_3 * M_2^{\dagger} * M_1 * M_2 * M_3$



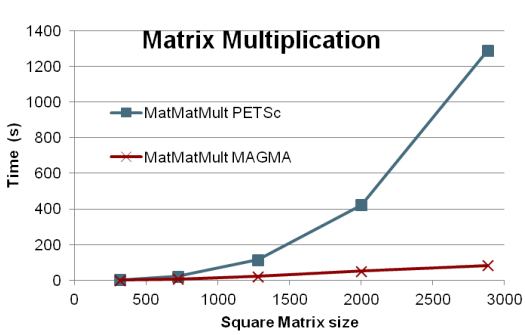
(b) $result = M_1 * M_2 * M_3$



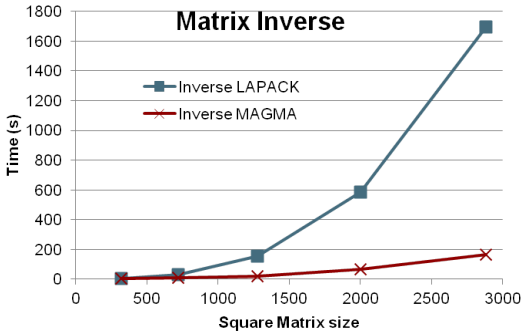
(c) $result = M^{-1}$

Figure 2: Matrix multiplications and inversion using MAGMA. Blue blocks are dense matrices in RAM and red blocks are dense matrices in GPU memory.

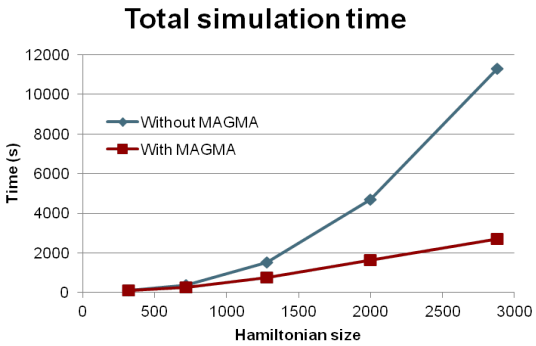
A temporary task manager was developed, which would assign certain processes for offload to 1 GPU each. With this, and the already set up MPI parallelization scheme, the algorithm was scaled to multiple nodes. Scaling results showed that the run time remained almost constant on increasing problem size and number of cores. The GPUs were idle for about 70% of the total run time and were not utilized efficiently. A stand alone library is currently under development for running the RGF algorithm on GPUs. This library would handle asynchronous data transfer as well as simultaneous computation over CPU and GPU. Overlapping these tasks would provide more efficient resource utilization.



(a) Multiplication scaling



(b) Inversion scaling



(c) Total simulation scaling

Figure 3: Scaling for matrix multiplications and inversion using MAGMA. The blue line shows single CPU time, red line shows 1 CPU + 1 GPU time. 3a shows scaling of dense matrix multiplication vs problem size. 3b shows scaling of inversions of corresponding problem sizes. 3c shows how the total start to end simulation time scales. Realistic problems have matrices of size about 3000×3000.

3.3 MAGMA interface in PETSc

An interface to the MAGMA library has been built in PETSc. The MAGMA library provides some LAPACK functions on GPUs as discussed in the MAGMA section. The plan was to use this interface in NEMO5 for offloading LU factorization, MatMatMult and Linear equation solving to the Kepler GPU's on BlueWaters XK nodes. After finishing the implementation and while it was undergoing testing, it was determined with PETSc developers that both the double and complex builds of PETSc attempt to initialize the GPU and there was no current work around. NEMO5 is unique in that it links to two PETSc builds and so this was unknown before our attempt. NEMO5 could use the PETSc-MAGMA interface but for only double or complex PETSc builds. Therefore, MAGMA was used directly from NEMO5. The PETSc-MAGMA will be part of the future PETSc releases and will benefit other PETSc users. Additionally, PETSc developers may be able to fix this issue of dual-builds to allow support for sparse matrices via the CUSP interface (see below).

Calling MAGMA functions directly from NEMO5 was actually tested early on the MAGMA/PETSc/NEMO5 development with an eigenvalue solver, but the PETSc interface solution would have been easier, more elegant and more robust—NEMO5 should not particularly care how the linear algebra problem is being solved—PETSc is primarily handling matrix management and manipulations. Currently the routines for LU factorization and linear solve (which relies on factorization) have been completed. Performance improvements are sizeable (shown below). Implementation of other routines is straightforward. NEMO5 calculates sequential dense matrix operations on GPU directly via MAGMA.

The Following MAGMA functions are implemented in NEMO5

Linear solves using LU factorization

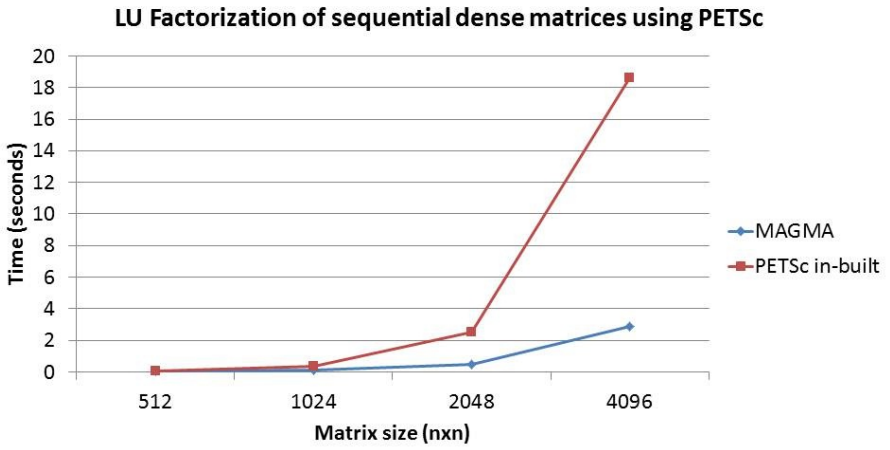


Figure 4: Speedup using GPU for LU factorization. Block size in compression algorithm 2kx2k for 12x12 nm sample. Developed an interface to the LU and Cholesky factorization in MAGMA library in PETSc.

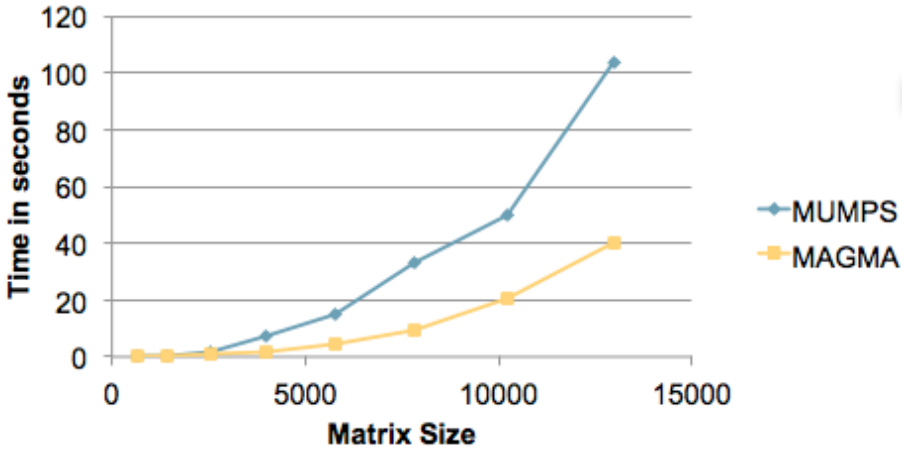


Figure 5: GPU speedup for the linear solve. MAGMA solve time includes time for offloading and retrieving matrices. Range of wire cross-sections: 2x2 – 9x9 10x10 cross-section wire is too large for GPU memory (16k x 16k matrix)

- magma_zgetrf_gpu
- magma_dgetrf_gpu
- magma_zgetrs_gpu
- magma_dgetrs_gpu

Sequential dense MatMat mult using GPUs

- magmablas_zgemm
- magmablas_dgemm

Matrix inversion using LU/Cholesky factorization

- magma_zgetri_gpu
- magma_zpotri_gpu

3.4 PETSc-CUSP Interface

MAGMA is designed for dense matrices. The existing PETSc code provides an interface to CUSP which is a library of algorithms for sparse linear systems[10]. These algorithms for sparse matrices

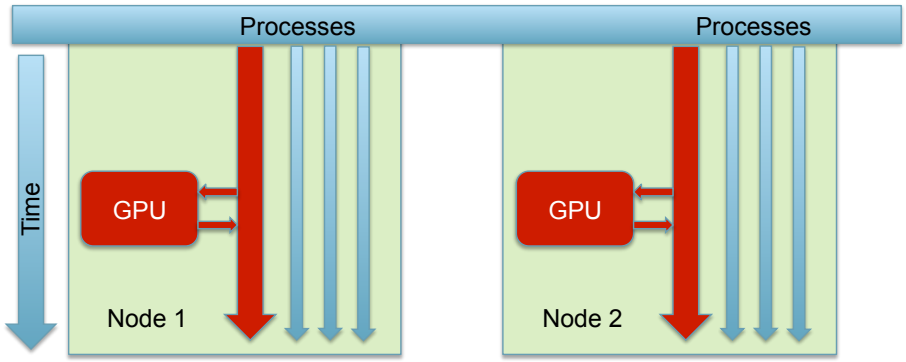


Figure 6: Offloading processes take on more work. 18 energies on a Bluewaters XK node, 14 on CPUs and 4 on CPU+GPU.

will primarily help with the self-energy calculations. However, a usable CUSP implementation will not be available until PETSc 3.5, which is roughly scheduled for a quarter 2 2014 release. Linking NEMO5 to PETSc built with CUSP currently produces runtime errors which we are investigating, however, these errors occur even without using any CUSP functionality. MAGMA can theoretically act on sparse matrices, but with extreme performance degradation. In the meantime, some of these CUSP functions may be called directly from NEMO5, after the NEMO5-MAGMA call implementations have been finished.

4 Status

Asynchronous data transfer has been implemented in RGF algorithm. The following API is used for computations in RGF.

Sparse-dense matrix multiplication

`cusparseZcsrmm and cusparseZcsrmm2`

Dense-dense matrix multiplication

`cublasZgemm`

Matrix inversion

`magma_zgetrf_gpu and magma_zgetri_gpu`

Miscellaneous algebra: custom CUDA kernels for sparse to dense conversion with dense shift, cublasZaxpy, etc.

A formal task distribution system is currently under development, which will be able to discover and automatically assign resources to different tasks. After work on the task distribution is complete, testing of load balancing and scalability of the RGF algorithm on BlueWaters can continue.

5 Future Work

After a recent discussion with NVIDIA engineers, it was found that sparse-dense matrix multiplication is not fast on GPUs for matrix sizes smaller than about 4500. The current problem size involves matrices of size 3000x3000. A strategy needs to be developed for resource management for larger problem sizes if a good performance is to be expected from the GPU. Work is ongoing to build a more general framework for GPUs similar to the one current approach using PETSc matrices. This involves developing wrapper classes for Sparse CSR, BSR and Dense matrices which are GPU aware. Other work involves refining a task manager for balancing the heterogeneous load and optimizing performance for improved scaling results for larger simulations.

6 Summary

The approach described here allowed NEMO5 to take advantage of a variety of CUDA-enabled libraries. A custom code gives more flexibility in overlapping computation with data transfer to and from GPU. These developments have cleared the way for NEMO5 to make use of the Blue Waters machine and achieve a high level of performance and scalability. GPU capability in NEMO5 has been achieved through the use of the MAGMA, cuBLAS, and cuSPARSE libraries. A PETSc-MAGMA interface has been developed in conjunction with PETSc developers and will be part of the future PETSc releases to benefit other users. A more general framework for computation on GPU can now be developed which uses CUDA capable libraries and custom CUDA kernels.

References

[1] International technology roadmap for semiconductors, 2012.

[2] Martin Fuechsle, Jill A. Miwa, Suddhasatta Mahapatra, Hoon Ryu, Sunhee Lee, Oliver Warschkow, Lloyd C. L. Hollenberg, Gerhard Klimeck, and Michelle Y. Simmons. A single-atom transistor. *Nat Nano*, 7(4):242–246, 2012. 10.1038/nnano.2012.21.

[3] Jim Fonseca, Tillmann C Kubis, Michael Povolotskyi, B. Novakovic, A. Ajoy, Ganesh Hegde, Hesameddin Ilatikhameneh, Zhengping Jiang, Parijat Sengupta, Yaohua Tan, and Gerhard Klimeck. Efficient and realistic device modeling from atomic detail to the nanoscale. *Journal of Computational Electronics*, 12(4):1–9, 2013.

[4] Supriyo Datta. *Electronic transport in mesoscopic systems*. Cambridge studies in semiconductor physics and microelectronic engineering. Cambridge University Press, Cambridge ; New York, 1995.

[5] Roger Lake, Gerhard Klimeck, R. Chris Bowen, and Dejan Jovanovic. Single and multiband modeling of quantum electron transport through layered semiconductor devices. *Journal of Applied Physics*, 81(12):7845, 1997.

[6] M P Lopez Sancho, J M Lopez Sancho, and J Rubio. Highly convergent schemes for the calculation of bulk and surface Green functions. *Journal of Physics F: Metal Physics*, 15:851–858, 1985.

[7] M P Lopez Sancho, J M Lopez Sancho, and J Rubio. Quick iterative scheme for the calculation of transfer matrices: application to Mo (100). *Journal of Physics F: Metal Physics*, 14(5):1205–1215, May 1984.

[8] R. Nath, S. Tomov, and J. Dongarra. An Improved Magma Gemm For Fermi Graphics Processing Units. *International Journal of High Performance Computing Applications*, 24(4):511–515, November 2010.

[9] P. Ezzatti, E. S. Quintana-Ortí, and a. Remón. Using graphics processors to accelerate the computation of the matrix inverse. *The Journal of Supercomputing*, 58(3):429–437, April 2011.

[10] V. Minden, B.F. Smith, and M.G. Knepley. Preliminary implementation of petsc using gpus. *Proceedings of the 2010 International Workshop of GPU Solutions to Multiscale Problems in Science and Engineering*, 2010.