Iterative Krylov solution methods for geophysical electromagnetic simulations on throughput-oriented processing units

Michael Commer[1]

Filipe R. N. C. Maia[2]

Gregory A. Newman[1]

[1] Earth Sciences Division, Lawrence Berkeley National Laboratory, Berkeley, California, 94720, USA

[2] National Energy Research Scientific Computing Center, Lawrence Berkeley National Laboratory, Berkeley, California, 94720, USA

Abstract

Many geoscientific applications involve boundary value problems arising in simulating electrostatic and electromagnetic fields for geophysical prospecting and subsurface imaging of electrical resistivity. Modeling complex geological media with three-dimensional finite difference grids gives rise to large sparse linear systems of equations. For such systems, we have implemented three common iterative Krylov solution methods on graphics processing units and compare their performance with parallel host-based versions. The benchmarks show that the device efficiency improves with increasing grid sizes. Limitations are currently given by the device memory resources.

Keywords

iterative Krylov methods, electromagnetic modeling, finite-difference methods, parallel solutions, GPU, NVIDIA CUDA

1 Introduction

Modern graphics processing units (GPUs) are designed for efficiently manipulating computer graphics. Their highly parallel architecture makes them also suitable for compute-intensive scientific applications. To provide access to the multithreaded computational resources and associated memory bandwidth of GPUs, graphics hardware manufacturers have introduced new application programming interfaces enabling numerical calculations in a fashion similar to parallel computing paradigms.

A large class of geo-scientific applications involves boundary value problems arising in simulating electromagnetic (EM) and magnetotelluric (MT) fields for geophysical prospecting and subsurface imaging of electrical resistivity. Often the need is to simulate such fields in complex three-dimensional (3D) geological media. Finite difference techniques have been our methods of choice for complex simulation problems of this sort (Commer and Newman, 2008), and give rise to large sparse linear systems of equations of the form

$$\mathbf{A}_{N\times N}\mathbf{x} = \mathbf{b}\,. \tag{1}$$

The non-singular matrix $\mathbf{A}$ is either real-symmetric, $\mathbf{A}\in\mathbb{R}^{N\times N}$, or complex-symmetric, $\mathbf{A}\in\mathbb{C}^{N\times N}$, and the solution and right-hand-side vectors are $\mathbf{x},\mathbf{b}\in\mathbb{R}^{N}\,/\,\mathbb{C}^{N}$, respectively. The size of such systems arising from 3D EM simulations prohibits usage of direct solvers. Thus, iterative Krylov subspace techniques are commonly used.

1.1 Iterative Krylov subspace methods

Krylov subspace methods are defined as projection (Galerkin) or generalized projection (Petrov-Galerkin) methods for the solution of the linear system (1). The solution involves constructing the Krylov subspace K_m,

$$K_m = K_m\left(\mathbf{A}, \mathbf{r}_0\right) = span\left\{\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \ldots, \mathbf{A}^{m-1}\mathbf{r}_0\right\} \tag{2}$$

Starting with the residual vector, $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$, Krylov methods compute the optimal approximation $\mathbf{x}_m \in \mathbf{x}_0 + K_m$ to the solution of (1) in an iterative manner, where at each iteration the dimension m of K is updated. Krylov methods are named after the Russian applied mathematician and naval engineer Alexei Krylov, who published a paper on this topic in 1931 and formed the basis from which all Krylov methods later developed.

1.2 Krylov solvers used in geophysical resistivity prospecting

A common near-surface application is the DC resistivity method, where a DC (or a very low-frequency) current is introduced as a means of studying earth electrical resistivity, for example in groundwater mapping. Time-harmonic EM prospecting methods with larger penetration depths use transmitter frequencies below roughly 100 kHz. The Krylov solvers of interest for these applications are designed to handle linear systems where **A** is real/complex symmetric. For the real symmetric case, the conjugate gradient (CG) method of Hestenes and Stiefel (1952) is the method of choice. While in practice rounding errors introduce a loss of orthogonality in the conjugate directions, CG performs excellently when solving sparse linear systems that are reasonable well conditioned. CG can also be applied to any Hermitian (complex) linear system that is symmetric positive definite. For the complex symmetric case, where the matrix is non-Hermitian, the bi-conjugate gradient (BiCG) method first proposed by Lanczos (1952) and quasi

minimum residual (QMR), more recently proposed by Freund and Nachtigal (1991) and Freund (1992), are effective and cost efficient solvers.

The main computational burden of Krylov methods lies in a sparse matrix – vector (SpMV) product that occurs hundreds to thousands of times and is needed to iteratively construct the Krylov subspace. This subspace forms the basis from which the solution to equation (1) is constructed. The remaining operations with a significant computing percentage are dense linear algebra operations on vectors. To exploit the GPU throughput, a key point is parallelizing these operations in a way that fits the highly parallel device hardware architecture.

For the present performance study, we have developed three iterative solvers relevant for solving electrostatic, frequency-domain EM, and MT modeling problems on GPUs. In Section 2, we briefly introduce the relevant underlying equations, and describe their solution with a finite-difference (FD) technique. In Section 3, the GPU-specific methodology is introduced, before comparing the performance of our GPU Krylov solvers against different parallel CPU counterparts in Section 4.

2 Method

The BiCG and QMR methods are widely applied for modeling time-harmonic eddy current problems (e.g., Sarkar, 1987; Smith et al., 1990; Wang and Jin, 1998; Gersem et al., 1999). In contrast to typical engineering applications, geophysical EM responses involve the solution of Maxwell's equation in the quasi-static limit, where EM diffusion dominates (Alumbaugh et al., 1996). Simulations with both an AC and DC transmitter type can involve a real or a complex tensor for describing the earth's electrical resistivity. Therefore, we also consider complex arithmetic for the electrostatic problem. We choose the FD method over the integral equation method because of the ability to employ fast iterative solvers. Finite elements (FE) are superior

for simulating topographically complex structures, an aspect which becomes more important with higher excitation frequencies (as in radar methods). While this may be open to debate, considering the diffusive domain of our applications, we have concluded that the FD method offers more advantages over FE. Using our solvers in computationally expensive imaging methods, efficient grid separation strategies play a key role in keeping computing times at bay, where the regularity of 3D Cartesian FD meshes allows for a straightforward implementation of an elaborate material averaging scheme (Commer et al., 2008). In our case, the complex conductivity (reciprocal of resistivity) tensor for a 3D Cartesian FD grid cell is the diagonal tensor

$$\boldsymbol{\sigma} = \begin{pmatrix} \sigma_{xx} & 0 & 0 \\ 0 & \sigma_{yy} & 0 \\ 0 & 0 & \sigma_{zz} \end{pmatrix}, \tag{3}$$

where σ_{xx}, σ_{yy}, and σ_{zz} denote the directional (real or complex) conductivities sampled on a FD grid cell's *x*-, *y*-, and *z*- edges.

2.1 Finite-difference solution of wideband electromagnetic problems

The time-harmonic EM field simulation leads to the vector Helmholtz equation, where the unknowns are given by the components of the vector electric field, **E**,

$$\nabla \times \nabla \times \mathbf{E} + i\omega\mu_0 \boldsymbol{\sigma} \mathbf{E} = -i\omega\mu_0 \mathbf{J} \tag{4}$$

In this formulation, μ_0 and $\omega=2\pi f$ denote the free-space magnetic permeability and angular frequency, where f is measured in Hz. The FD discretization of (4) is illustrated in Figure 1, where each node has three designated field components E_x, E_y, and E_z, as depicted for the center node (i,j,k). Hence, given a mesh size (in nodes) of $N_x \times N_y \times N_z$, the size of $\mathbf{A}_{N \times N}$ resulting from (4)

is N=3× N_x×N_y×N_z. The FD discretization of the curl-curl operator results in a maximum number of 13 nonzeros per row. For example, all black arrows in Figure 1 are the field components involved in forming Equation (4) for the component E_x at the center node (i,j,k). Similarly, the red and blue arrows pertain to E_y(i,j,k) and E_z(i,j,k), respectively. This scheme leads to a sparse pattern as exemplified in Figure 2a.

Higher order difference schemes produce similarly sparse systems that can also be effectively solved using Krylov methods, but will come at added expense because of increased matrix bandwidths. For EM type problems, the need to go to higher order schemes is mitigated because calculations are performed out to several wavelengths, hence grid dispersion problems, which can be treated with higher order schemes, do not typically arise.

2.2 Finite-difference solution of electrostatic problems

The electrostatic problem gives rise to the Poisson equation, a partial differential equation of elliptic type,

$$\nabla \cdot \left(\boldsymbol{\sigma} \nabla \boldsymbol{\varphi} \right) = \nabla \cdot \mathbf{J}, \tag{5}$$

where **J** represents the DC source current distribution impressed by a galvanic source type. The numerical solution of (5) on a 3D FD grid is also illustrated in Figure 1. Here, the unknowns are given by the potential field vector $\boldsymbol{\varphi}$ and are node-based, thus leading to a matrix size of N=N_x×N_y×N_z. Exemplified by the green nodes, updating $\boldsymbol{\varphi}$ at the central node (i,j,k) involves a 7-point stencil, with a maximum of 7 nonzeros per row. The sparse structure resulting from Equation (5) is shown in Figure 2b.

3 Parallel iterative Krylov solvers

For low-level GPU access, we employ CUDA from NVIDIA, a parallel computing architecture that provides low-level access to GPUs through a C/C++-type programming language with NVIDIA extensions. All CUDA implementations of the three employed Krylov methods, namely CG, BiCG, and QMR, were reprogrammed from our original parallel FORTRAN90 EM modeling algorithms. The reader is referred to the works of Alumbaugh et al. (1996) and Commer et al. (2008) for details related to the massively parallel aspect of these simulators. Our FORTRAN90 versions, also referred to as CPU solvers in the following, do not rely on any external libraries. We found that the flexibility of general purpose library packages often comes at the expense of not achieving the optimal performance for specific problems. Nevertheless, we also provide timing results from external (CPU) solvers in order to relate our benchmarks to commonly applied library packages. Specifically, we employ the CG solver contained in the parallel solver library Aztec, version 2.1, (Tuminaro, 1999). For the BiCG and QMR methods, we employ the Portable Extensible Toolkit for Scientific Computation (PETSc, version 3.1) (Balay et al., 2010). Note that we did not make use of any external GPU library for SpMV.

We use Jacobi scaling as a default preconditioner for all Krylov solvers employed here. Being called only once before the call of the Krylov solver, it has proved good efficiency over a number of other preconditioners investigated for wideband EM problems (Alumbaugh et al., 1996). Even with Jacobi scaling, solving equation (4) in the low-frequency range - f<0.1 Hz is common in MT applications - is hindered by a badly conditioned system due to the null space of the curl-curl operator. Removing this null space can be accomplished by a preconditioning step which decomposes the electric field into curl-free and divergence-free projections using the Helmholtz theorem. However, owing to additional memory overhead, this low-induction-number preconditioner (Newman and Alumbaugh, 2002) is not part of our current GPU implementations.

3.1 Sparse matrix-vector multiplication on GPU

Owing to the relatively low operation count versus memory access count in SpMV involved in the solution of (4) and (5), memory bandwidth is a major limiting factor in the iterative Krylov solver performance. Poor cache utilization and extra load operations due to non-optimal matrix storage can decrease the performance of SpMV, because with modern hardware, cache miss latencies dominate latencies due to operations. While with irregular sparsity patterns, the number of cache misses can increase significantly, the matrix types of our applications are characterized by a regular sparse structure (Figure 2). For such types, where the number of non-zeros per row, K, is almost constant, the sparse ELLPACK (or ELL) storage format is particularly suited (Bell and Garland, 2009). The ELLPACK format stores the nonzero values in column-major order, filling a dense N-by-K data array, where rows with less than K non-zeros, pertaining to mesh boundaries, are zero-padded. The corresponding column indices are stored in another column-major ordered integer array, with the padding entries pointing to zeros.

Each thread of the GPU carries out the multiplication of one matrix row. Because the overwhelming majority of the rows have the same number of non-zero elements, almost no threads sit idle waiting for the rest of the warp to complete. By using a column-major ordering of the data, contiguous threads in a warp access contiguous global memory banks, thus maximizing memory throughput and avoiding memory bank conflicts. Other multiplication strategies were tested, including the schemes outlined by Bell and Garland (2009), where the one described above was the most efficient. We found no performance advantage when assigning multiple rows to one thread.

3.2 Other optimizations for iterative solvers

The Krylov solvers involve other dense linear algebra operations such as vector sum, vector scaling, and dot products. Table 1 lists the number of operation counts for each solver. To further reduce the memory bandwidth, several BLAS calls were fused into a single kernel whenever possible, such as vector scaling followed by a dot product. This cuts the number of memory accesses roughly in half. Speeding up these vector operations, becomes beneficial for more complex algorithms such as QMR, where the computing effort spent outside the matrix vector multiplication is significant. The kernels were optimized for the NVIDIA Tesla C2050 (Fermi); in particular no attempt was made to load the input vector to shared memory, instead relying on the L1 cache to limit the number of accesses to global memory. The kernels were configured to prefer L1 cache over shared memory. This provides 48KB of L1 cache per Streaming Multiprocessor instead of the default 16KB, but the performance improvement obtained was marginal.

While not pursued further in this work, cache utilization can be further optimized by matrix bandwidth reduction techniques and alternative data structures (Pınar and Heath, 1999; Toledo, 1997; Xu et al., 2010). To estimate the potential of reducing the bandwidth of **A** for our implementations, we run the QMR solver for a number of grid sizes ranging from $N=3.8\cdot10^5$ to $N=6.6\cdot10^6$. The matrix column indices were arbitrarily reordered to reach the theoretically lowest bandwidth of 13. While obviously not leading to solution convergence, we are only interested in the computing speedup compared to the actual matrix profile. Figure 3 indicates that the speedup becomes more significant with increasing N. However, it remains relatively modest given that the shown computing times pertain to the lowest bandwidth possible. For our CPU QMR solver (a), we obtain an average speedup of 3.4%. For the GPU QMR solver (b), the average is 10.8% which is in accordance with similar findings by Xu et al. (2010).

4 Krylov solver performance comparisons

All following CPU benchmarks were performed on a general purpose GPU testbed with 8 cores per compute node and a node configuration as follows: 2 Intel 5530 processors, 2.4 GHz, 8MB cache, 5.86GT/sec QPI Quad core Nehalem, where QPI stands for QuickPath Interconnect by Intel for high interconnect performance. Table 2 summarizes hardware and software specifications. The employed CPU solvers use 8 cores (one node) of the parallel cluster. Note that a higher memory bandwidth and thus faster solution can be achieved by distributing the problem across the same number of nodes, using one core per node. However, we restrict ourselves to the case that would be more realistic in a multi-user environment. The employed compute node is connected to an NVIDIA Tesla C2050 (Fermi) GPU with 3 GB of memory and 448 parallel CUDA processor cores.

It is our experience that, depending on the spectral norm of the underlying matrix, the three employed Krylov methods require several hundreds to thousands of iterations for achieving acceptable solution accuracies for typical EM modelling problems. Therefore, we use a fixed number of 1000 solver iterations for each timing data point and report the actual solution times in seconds. For our applications, this measure is preferable, owing to more practicality for geophysical imaging applications, where one imaging experiment requires a large number of solutions of equation (1) (Commer et al., 2008).

4.1 CG solver for electrostatic simulations

We first compare our GPU implementation of a CG solver with benchmarks obtained from two different CPU implementations. In addition to our original FORTRAN90 version of the CG method, we also compare our GPU implementation against the CG solver provided by the Aztec

library (Tuminaro et al., 1999). Because ELLPACK is not supported by Aztec, the chosen matrix storage format is the distributed modified sparse row format, which is a generalization of the modified sparse row format.

Figure 4a shows that the performance difference between CPU and GPU solvers becomes more prominent with increasing matrix size, *N*, shown on the abscissa in units of millions. Here, the matrix results from a sample modeling problem that involves solution of Equation (4). The largest matrix size shown here would represent a 3D FD grid of node size $250 \times 250 \times 250$. The GPU solver achieves a 2.8 fold speedup compared to its CPU equivalent. Furthermore, compared to the library version (named Aztec-CPU in Figure 4a), the speedup factor is 5. With 8 CPU cores, these factors would amount to 22.4 and 40, respectively, when expressed in terms of a single core. Also shown is the memory consumption by the square symbols. At the same time, these symbols denote sample points where actual calculations for the timing data were carried out. The parallel efficiency can be assessed in Figure 4b by the achieved memory throughput (black symbols). Relating the bandwidth to its peak, one observes that for the CG solver, because of L1 caching, we exceed the theoretical bandwidth for matrix sizes with more than 10^6 matrix rows.

4.2 BiCG and QMR solver for frequency-domain EM simulations

The BiCG solution times are presented in Figure 4c. The sample modeling problems considered here were taken from a CSEM imaging experiment involving data measured over the Troll gas reservoir in the North Sea (Commer and Newman, 2008). The biggest sample problem fitting into the memory of the employed GPU device is exemplified by a mesh size of $132 \times 132 \times 132$. For the marine CSEM data simulation of the Troll survey, the GPU's resources are absolutely

sufficient. Figure 4c exhibits a performance trend similar to the CG solution times for the three different BiCG solvers. Again, the results indicate an increasing performance difference with larger meshes. Note that the PETSc benchmark was obtained from the performance-optimized (non-debugging) build for complex double precision. Because of a higher operation count, compared to CG, the peak bandwidth is not exceeded, however does get close to the theoretical maximum (Figure 4d).

At last, we summarize QMR performance measurements in Figure 4e. The QMR solver is favorable because it combines the advantage of BiCG, namely the relatively low memory and computational overhead, with smoother convergence properties. The convergence behavior of QMR follows from a least-square solution of the reduced tridiagonal system produced by the Lanczos process; this is similar to the approach followed in the generalized minimal residual method (GMRES). Since the constructed basis for the Krylov subspace is biorthogonal, rather than orthogonal as in GMRES, the obtained solution is viewed as a quasi-minimal residual solution, which explains the name. QMR also exploits look-ahead techniques to avoid breakdowns in the underlying Lanczos process. This makes it more robust than the BiCG method at the expense of a slightly higher computational overhead. Like CG, both BiCG and QMR solvers use short recurrences, as opposed to long ones in GMRES. To avoid the loss of orthogonality, longer term recurrence relations, at higher computational expense, could be used as in GMRES, which renormalizes the conjugate directions against a subset of pre-determined ones through a Gram-Schmidt orthogonalization process.

The GPU QMR solver achieves an acceleration factor of 2.4 and 4, compared to the CPU counterpart and the solver provided by PETSc, respectively (again, using 8 CPU cores). PETSc provides two types of QMR solvers, where we used the transpose-free QMR method, being the

most comparable to our implementation. Compared to CG and BiCG, QMR has the highest operation count, which reflects in a lower memory throughput (Figure 4f).

5 Conclusions

We have implemented efficient Krylov subspace methods for the iterative solution of linear systems with a large sparse matrix on GPUs. Our solvers are suitable for the simulation of electrical and electromagnetic simulation problems that arise in geophysical resistivity prospecting. All shown timing comparisons clearly indicate the increasing efficiency of the GPU solvers for increasingly larger matrix sizes. For the largest problems exemplified, the GPU performance is equivalent to almost 23 (CG) and 19 (BiCG, QMR) CPU cores, when comparing to the faster CPU solvers that do not use external libraries.

As outlined in detail in a previous work (Commer and Newman, 2008), our EM field simulation code achieves computational efficiency by a FD grid optimization scheme for the underlying forward modeling operator. This scheme optimizes grid spatial extension and sampling, taking the survey characteristics of a given modeling scenario into consideration. However, to address both computing and memory needs of systems arising from typical large-scale exploration applications, we still need to distribute the solution of (1) across the order of 100 CPU cores or more.

We are exploring a corresponding GPU approach, i.e. solving one system on multiple GPUs. CUDA's direct GPU-to-GPU communication is one approach. However, it is limited to the number of GPUs connected to one host node, as communication is not possible across the network. Therefore, a non-blocking MPI device-host-device communication will also be investigated.

Acknowledgments

We greatly acknowledge the Chevron Energy Technology Corporation and the Petascale Initiative in Computational Science at the National Energy Research Scientific Computing Center (NERSC) for providing base funding for this work. We also thank the NERSC staff for support and computing time on a GPU cluster. This work was also supported by the Director, Office of Science, Advanced Scientific Computing Research, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

Alumbaugh, D. L., G.A. Newman, L. Prevost, and J.N. Shadid, 1996. Three-dimensional wideband electromagnetic modeling on massively parallel computers, Radio Science, 31, 1-23.

Balay, S., J. Brown, K. Buschelman, V. Eijkhout, W.D. Gropp, D. Kaushik, M.G. Knepley, L.C. McInnes, B.F. Smith, and H. Zhang, 2010. PETSc Users Manual, Tech. Rep. Number ANL-95/11 - Revision 3.1, Argonne National Laboratory.

Bell, N., and M. Garland, 2009. Implementing sparse matrix-vector multiplication on throughput-oriented processors, in "Proc. Supercomputing '09", November 2009.

Commer, M., and G.A. Newman, 2008. New advances in three-dimensional controlled-source electromagnetic inversion, Geophysical Journal International, 172, 513-535.

Commer, M., G.A. Newman, J.J. Carazzone, T.A. Dickens, K.E. Green, L.A. Wahrmund, D.E. Willen, and J. Shiu, 2008. Massively-parallel electrical-conductivity imaging of hydrocarbons using the Blue Gene/L supercomputer, IBM Journal of Research and Development, 52-1/2, 93-103.

Freund, R., 1992. Conjugate gradient type methods for linear systems with complex symmetric coefficient matrices, SIAM J. Sci. Stat. Comput., 13, 425-448.

Hestenes, M.R., and E. Stiefel, 1952. Methods of conjugate directions for solving linear systems, J. Res. Natl. Bur. Stand., 49, 409-435.

Lanczos, C., 1952. Solution of systems of linear equations by minimized iterations, J. Res. Natl. Bur. Stand., 49, 33-53.

Pınar A, and Heath MT, 1999, Improving performance of sparse matrix-vector multiplication, Proceedings of the 1999 ACM/IEEE conference on Supercomputing.

Smith CF, Peterson AF, and Mittra R, 1990, The biconjugate gradient method for electromagnetic scattering, IEEE Trans. Antennas Propagat., vol. 38, pp. 938–940.

Tuminaro, R.S., M. Heroux, S.A. Hutchinson, and J.N. Shadid, 1999. Official Aztec User's Guide: Version 2.1, Sand Report SAND99-8801J, December 1999, Sandia National Laboratories.

Wang, C.F., and Jin, J.M., 1998, Simple and efficient computation of electromagnetic fields in arbitrarily shaped inhomogeneous dielectric bodies using transpose-free QMR and FFT, IEEE Transactions on Microwave Theory and Techniques,vol.46, no.5, pp.553-558.

De Gersem H., Lahaye D, Vandewalle S, and Hameyer K, 1999, Comparison of quasi minimal residual and bi-conjugate gradient iterative methods to solve complex symmetric systems arising from time-harmonic simulations, COMPEL, 18, 3, 298-310.

Freund RW, and Nachtigal NM, QMR: A quasiminimal residual method for non-Hermitian linear systems, Numer. Math., vol. 60, no. 3, pp. 315–339, 1991.

Sarkar TK, On the application of the generalized biconjugate gradient method, Journal of Electromagnetic Waves and Applications, 1, 223-242, 1987.

Toledo S, 1997, Improving memory system performance of sparse matrix-vector multiplication, in Proc. of 8th SIAM Conf. on Parallel Processing for Scientific Computing, March 1997.

Xu S, Lin HX, and Xue W, 2010, Sparse matrix-vector multiplication optimizations based on matrix bandwidth reduction using NVIDIA CUDA, Ninth International Symposium on Distributed Computing and Applications to Business, Engineering and Science, Hong Kong.

Tables

Operation	CG	BiCG	QMR
Matrix-Vector multiply	1	1	1
Vector dot product	3	3	4
Vector addition/subtraction	3	3	6
Vector constant multiply	3	3	9

Table 1: Operation counts per iteration for the three Krylov solvers.

	CPU	GPU
Node configuration	2 Intel 5530 2.4 GHz, 8MB cache, 5.86GT/sec QPI Quad core Nehalem, 8 cores per node	NVIDIA Tesla C2050 (code named Fermi), 448 parallel CUDA processor cores
Memory	24GB DDR3-1066 Reg ECC	3GB
Compiler	pgf90 –O2	nvcc –O2 –arch sm_20
Programming language	FORTRAN90 with MPI library OpenMPI 1.4.2	CUDA 3.2 with CUBLAS library

Table 2: Hardware and software specifications of the GPU testbed used for the Krylov solver performance tests.

Figures

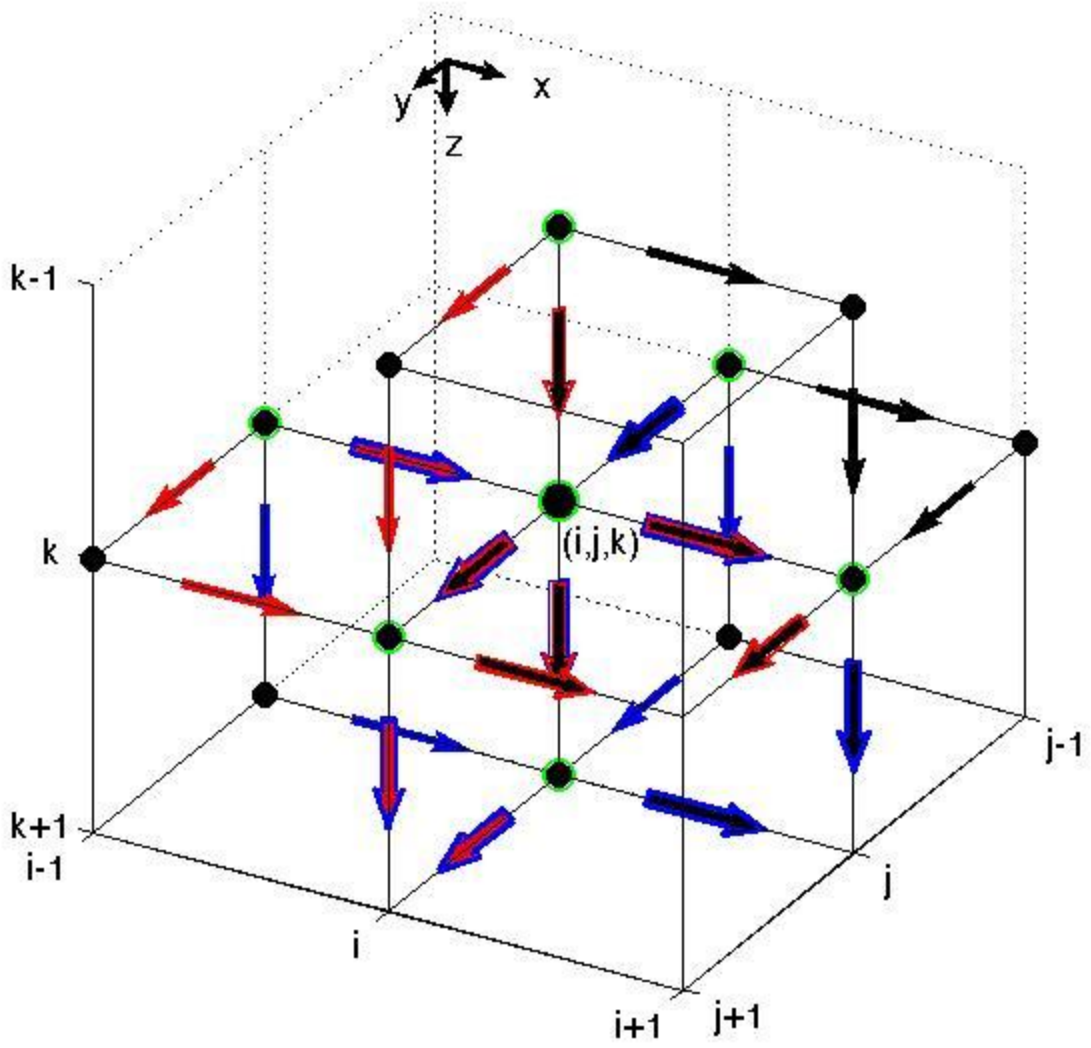

Figure 1: The staggered finite-difference grid for discretization of the 3D Helmholtz and Poisson equations. Electric field components are assigned to each grid node and are shown by arrows. The center node (i,j,k) has the three designated components E_x, E_y, and E_z shown by three-colored arrows. All black, red, and blue arrows mark the field components which go into constructing the matrix rows corresponding to E_x, E_y, and E_z, respectively, at (i,j,k). The seven-point stencil for the discrete Poisson equation is shown by the green-bordered circles. Electrical conductivities are assigned to individual grid nodes. The conductivity tensor (3) is then derived through a proper material averaging scheme (Commer and Newman, 2008).

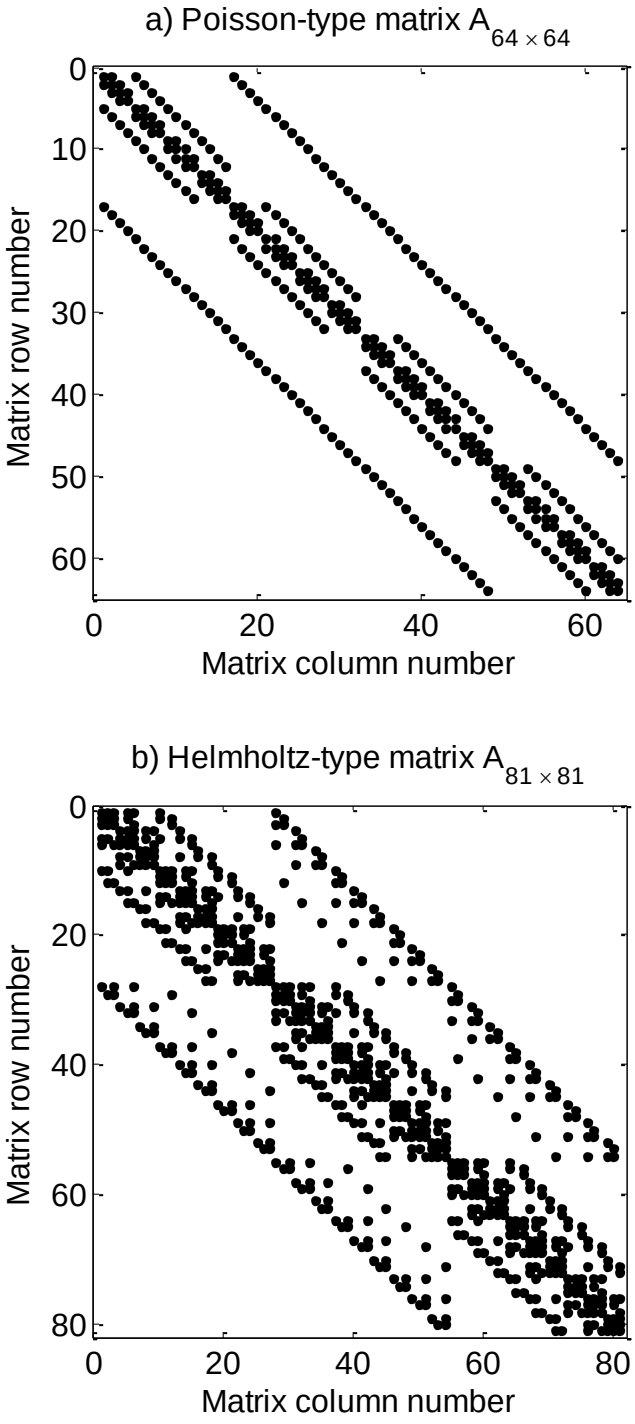

Figure 2: Sparse structure of matrix **A** in Equation (1) resulting from Helmholtz (a) and Poisson equation (b).

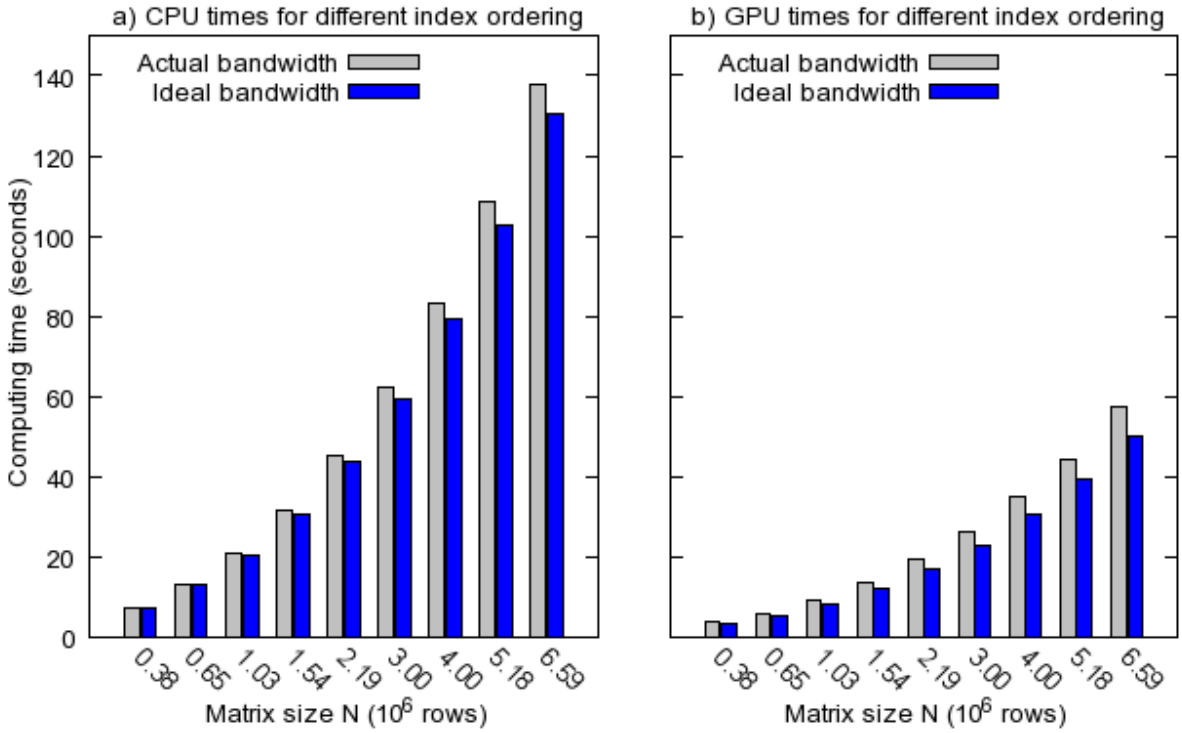

Figure 3: Performance improvement for different matrix sizes, where the sparse matrix resulting from Equation (4) has the ideal bandwidth of 13 (blue), compared against the actual bandwidth (grey). CPU (a) and GPU (b) computing times are for 1000 QMR iterations.

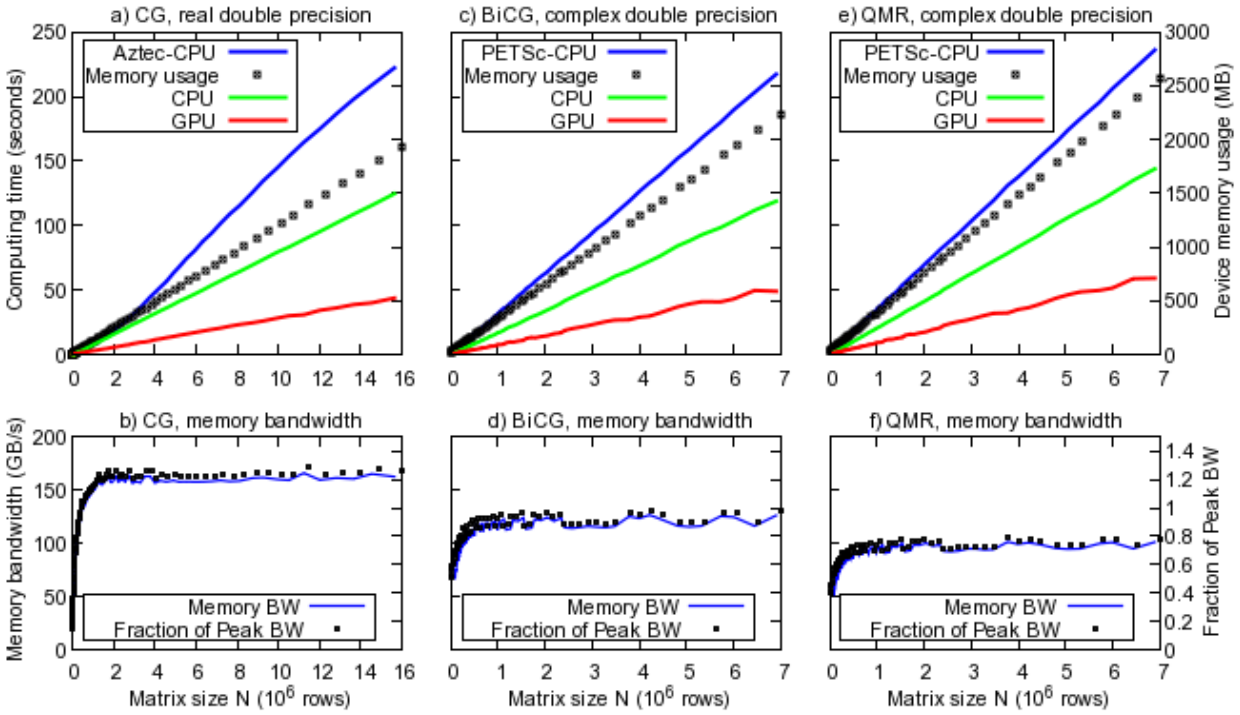

Figure 4: Solution times for the three different iterative Krylov solvers, CG (a), BiCG (c), and QMR (e). The solid lines correspond to the left y-axis and show the computing times in seconds required for 1000 Krylov iterations. The symbols pertain to the right y-axis and denote the requirements for storing all Krylov solver data structures in memory. Each GPU implementation is compared against its original parallel FORTRAN90 version, as well as a parallel solver provided by an external library. All CPU solvers were run on 8 processor cores. The corresponding memory throughput for each GPU solver is shown in the lower row (b,d,f).