Adjustable GPU Acceleration for Hermitian Eigensystems

Michael T. Garba[1], Horacio González–Vélez[1] and Daniel Roach[2]

[1] IDEAS Research Institute, Robert Gordon University
Aberdeen AB25 1HG, United Kingdom
Email:{m.t.garba; h.gonzalez-velez}@rgu.ed.ac.uk
[2] Materials and Physics Research Centre, University of Salford
Salford M5 4WT, United Kingdom
d.roach@salford.ac.uk

Abstract. This paper explores the early implementation of high-performance routines for the solution of multiple large Hermitian eigenvector and eigenvalue systems on a Graphics Processing Unit (GPU). We report a performance increase of up to two orders of magnitude over the original EISPACK routines with a NVIDIA Tesla C2050 GPU, potentially allowing an order of magnitude increase in the complexity or resolution of a Inelastic Neutron Scattering (INS) modelling application.
...

Keywords: GPU, Eigensystems, CUDA, Parallel Computing, Computational Linear Algebra

1 Introduction

Eigenvector and eigenvalue determination are a recurrent problem in computational modelling applications for which a number of broadly accepted libraries exist. However, as increasingly elaborate models become of practical value to scientific and engineering applications, the demands of solving larger systems typically require high performance computing with large high-performance clusters or supercomputers.

As an emerging parallel architecture for high-performance computing, the Graphics Processing Unit (GPU) has the potential to enable a new generation of applications for desktop machines and small clusters. Originally intended for intensive high-end 3D graphics and gaming, GPUs have demonstrated cluster-level performance at a fraction of the cost and energy consumption of traditional CPUs for certain general purpose applications. GPU advances are expected to sustain the trend of Moore's law that conventional CPUs are straining to maintain.

Conversely, the shift towards GPU computing is a drastic architectural change that has left a void in the space of application software and support libraries that are able to leverage the full capabilities of the platform. While the solution to computational modelling problems—which were impractical on the desktop and uneconomical on the supercomputer—may very well become the dominant GPU applications of the future, effective GPU programming remains an open problem in computational science.

With NVIDIA's CUDA, AMD's Firestream, Microsoft's DirectCompute and the vendor-neutral OpenCL platform, significant resources are being directed towards developing a supporting ecosystem for GPU computing in the form of libraries, specifications, and tools that are at various levels of maturity.

Driven by our own immediate need for high performance solvers for modelling Inelastic Neutron Scattering (INS), this paper describes the early stages of our application of GPUs to the solution of Hermitian eigensystems, based on the EISPACK library and using the CUDA platform. Our work may arguably shed some light on a computational problem of even wider significance than the intended modelling application.

2 Background

EISPACK, and its successor LAPACK, provide extensive linear algebra routines in mathematical and scientific computing. Originally developed in the US in the seventies [17], the accuracy and numerical stability of EISPACK has been established through diverse application over the past 30+ years, leading to a number of developments in the field [3].

We have developed an initial high performance parallel version of SCATTER [16], a software component to calculate coherent and incoherent polycrystalline INS (poly-CINS) capabilities for lattice models in

the highly popular Molecular/Lattice Dynamics package, the General Utility Lattice Program [6], that has demonstrated linear scaling up to 1024 nodes on the Huygens prototype supercomputer in the SARA facilities in the Netherlands [8]. Central to this parallel version are phonon mode calculations carried out with the support of EISPACK. However, not every SCATTER deployment may have access to major supercomputing installations and it is clear that more affordable computational power is required on the lower end of the computing scale [1].

Furthermore, GPU modules are becoming a frequent presence in high performance computing platforms and the application of SCATTER to progressively more complex models on larger installations will require reasonable usage of these resources. As a result, an efficient GPU implementation of the most computationally intensive parts of the SCATTER routine will alleviate this imperative demand. Of particular interest are solvers for the class of Hermitian eigensystems that occur in INS modelling, arising from the quantum mechanical determination of phonon modes and their associated scattering contributions [15].

For numerically intensive tasks, GPUs have substantial computing potential [18]. However, complex control flow with conditional branching and thread divergence incur a noticeable performance penalty [14]. To achieve reasonable performance benefits, it is necessary to augment traditional development techniques with low-level knowledge of the underlying GPU architecture [10].

The Compute Unified Device Architecture (CUDA) is NVIDIA's platform for GPU computing, providing compilation tools, libraries and a runtime system. CUDA allows the execution of *kernels*, written in CUDA C, on the GPU device. A kernel executes as a configurable grid of independent thread blocks that may contain up to 1024 threads in second generation CUDA devices.

A *Single Instruction Multiple Thread (SIMT)* abstraction, where threads within a block execute identical instructions and may operate on different memory locations, allows fine-grained data parallelism within blocks and task parallelism at kernel level [13]. Thread blocks are divided into *warps* of 32 threads. For a given block, only one of these warps is scheduled to execute on the actual hardware at any given instant.

GPU memory is hierarchically organised and independent from host memory. Global Memory, high-latency and high-bandwidth DRAM, is the primary memory available on the device and is accessible by all executing kernels as well as for host to GPU data transfer. Limited high-speed Shared Memory, essentially a user-managed cache, exists locally on each streaming multiprocessor to allow the explicit avoidance of expensive off-chip global memory accesses. Also present are register, texture and constant memories with various performance characteristics.

Memory transfer contention and bandwidth represent the predominant bottlenecks to GPU performance. Different authors have suggested various approaches to managing memory for scientific applications in GPU architectures, such as the use of cache analysis techniques to improve tiling algorithms [9], the deployment of low-level compiler annotations within CUDA source files to steer the memory hierarchy traversing [20], and the automatic translation of OpenMP structures into CUDA primitives [11].

However, such techniques are both application and architecture dependant, and may rely on manual intervention for code analysis, annotation, or tool coupling. Ideally, we would like to have a seamless integration of the application and the architecture without the need of additional code or human intervention.

In fact, a critical performance consideration is that high cost global memory operations can be performed simultaneously or *coalesced* for a thread warp if certain access constraints are satisfied. In practice, significant efforts are usually dedicated to optimising memory access patterns of this kind by what is frequently a hit-or-miss approach involving conflicting trade-offs to maximise the *compute to global memory access (CGMA) ratio* [10]. The GPU architecture and best practices for achieving good performance are extensively documented in the CUDA platform, and extensive efforts are now geared towards the use of platform-agnostic GPU frameworks which can deal with standard unified language deployments such as CUDA and OpenCL [4].

2.1 Contribution

Despite a number of emerging GPU numerical libraries, no open library for eigensystem analysis is available to completely meet the application requirements. Therefore, a basic port of the required functional subset of EISPACK to the GPU has been undertaken.

Admittedly, the more modern LAPACK—which has largely superseded EISPACK—may have formed a functionally superior basis. However, the inherent architectural complexity and reliance on an efficient

BLAS implementation implies a long-term effort that the immediacy of our requirements does not allow. The MAGMA library is such an effort that is in the early stages of providing hybrid multicore-CPU/GPU implementations of LAPACK routines [19].

The challenges of achieving efficient performance on a GPU architecture may justify the extended effort of custom algorithms developed specifically for the strengths of the platform [21]. However, we maintain the original algorithms of the legacy EISPACK implementation for several reasons:

(a) This work is motivated by a very practical application for which the EISPACK eigensolver has proven adequate.
(b) As EISPACK has been in production use for nearly 40 years, the numerical characteristics and accuracy have been established by exhaustive application and testing.
(c) The problem of creating a data-parallel GPU version is conceptually similar to that of creating a vector-processor version of the EISPACK routines. A vector implementation was created for for the IBM 3090-VF by [2].
(d) While alternative algorithms used in LAPACK may possess superior cache usage characteristics and performance in modern processor configurations, they provide this at the expense of software complexity and reliance on an efficient BLAS implementation.

The EISPACK implementation provides the ch driver for double-precision Hermitian matrices and its three subroutine dependencies shown in Table 1. It extends our initial work [7] by introducing a coordinated approach to the development of GPU-based eigensystem solvers.

3 Methods

Let A be a square matrix. Thus, $\mathbf{v}$ is an eigenvector of A (a non-zero vector) if and only if there exists a scalar λ which satisfies expression (1).

$$A\mathbf{v} = \lambda\mathbf{v} \tag{1}$$

The generalised nonsymmetric eigenvalue problem is to find eigenvalues λ and eigenvectors $\mathbf{v}$ which satisfy expression (2).

$$A\mathbf{v} = \lambda B\mathbf{v} \tag{2}$$

The solution of the real generalised nonsymmetric eigensystem problem for a matrix pair (A, B) involves computing the generalised Schur decomposition

$$A = QSZ^T \tag{3}$$
$$B = QTZ^T \tag{4}$$

where Q and Z are orthogonal matrices of left and right Schur vectors respectively, and (S, T) is the generalised Schur form whose diagonal elements give the α and β values. The algorithm used is the QZ method due to Moler and Stewart [12].

4 Implementation

As EISPACK is implemented in Fortran, these routines require source-level translation with the f2c tool [5] into equivalent C sources for compatibility with the C-based CUDA SDK (Figure 1). Thence, the subroutines from Table 1 have served as the basis for the creation of three functionally equivalent GPU kernels.

Performance gains emerge as data-parallel intensive loops are distributed between cooperating threads in a block and synchronisation constructs inserted to avoid race conditions between thread warps. These loops are identified from source-level line-profiling on the original CPU version of EISPACK, the assumption being that CPU performance is strongly indicative of potential performance bottlenecks in the GPU kernels. This is a necessary workaround as CUDA profiling tools provide relatively basic functionality.

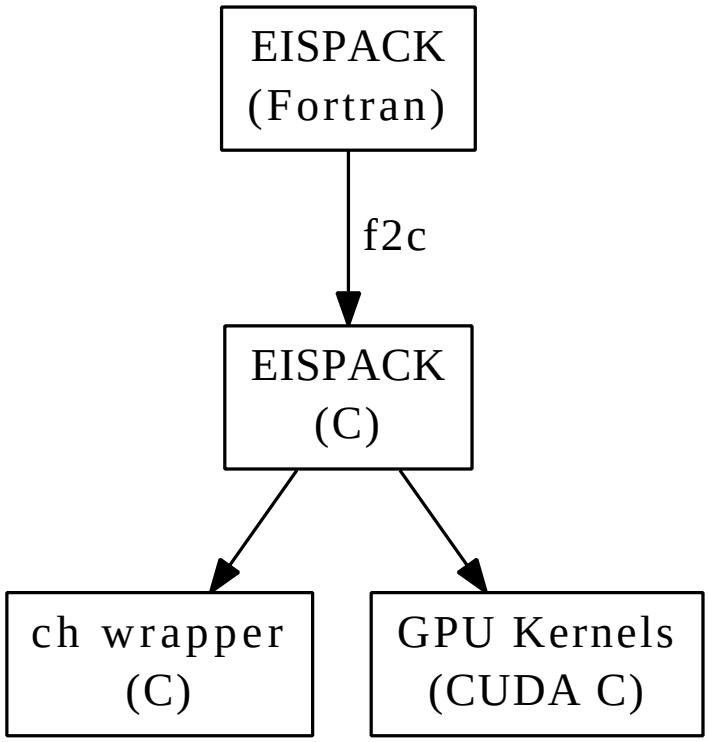

Fig. 1. The Fortran EISPACK source translated to C with f2c forms the basis of the CUDA Port

A number of thread blocks independently handle the solution of multiple eigensystems in parallel. In our implementation, a thread block or cooperative thread array (CTA) is mapped to an input problem set, allowing parallelism at both independent block and cooperative thread levels.

Some performance optimisations applied include:

(a) Asynchronous transfers to and from the Host over multiple streams allow concurrent kernel execution and overlapped I/O.
(b) Algorithm reorganisation for improved coalesced memory access. Transposed matrix layout in some subtasks is necessary to achieve higher memory transfer bandwidth.
(c) Improved register memory usage by the elimination (or reuse when appropriate) of extraneous register variables to improve GPU occupancy and facilitate latency hiding on the streaming multiprocessors.
(d) Use of explicit caching in shared memory to limit costly global memory accesses.
(e) Empirical determination of launch configuration by trial and error. While, the guidelines recommend that thread blocks sizes should be multiples of a warp to allow latency hiding for multiple warps, it is necessary to determine actual optimal block sizes by testing. The different kernels performed optimally at distinct block dimensions.

Routine	Description
htridi	Reduction of complex Hermitian matrix to real symmetric tridiagonal matrix via unitary similarity transformations.
tql2	Eigenvalues and eigenvectors of symmetric tridiagonal matrix by ql method.
htribk	Eigenvectors of complex Hermitian matrix by back transformation of corresponding real symmetric tridiagonal matrix.

Table 1. Relevant Hermitian Eigensystem routines in EISPACK

5 Results

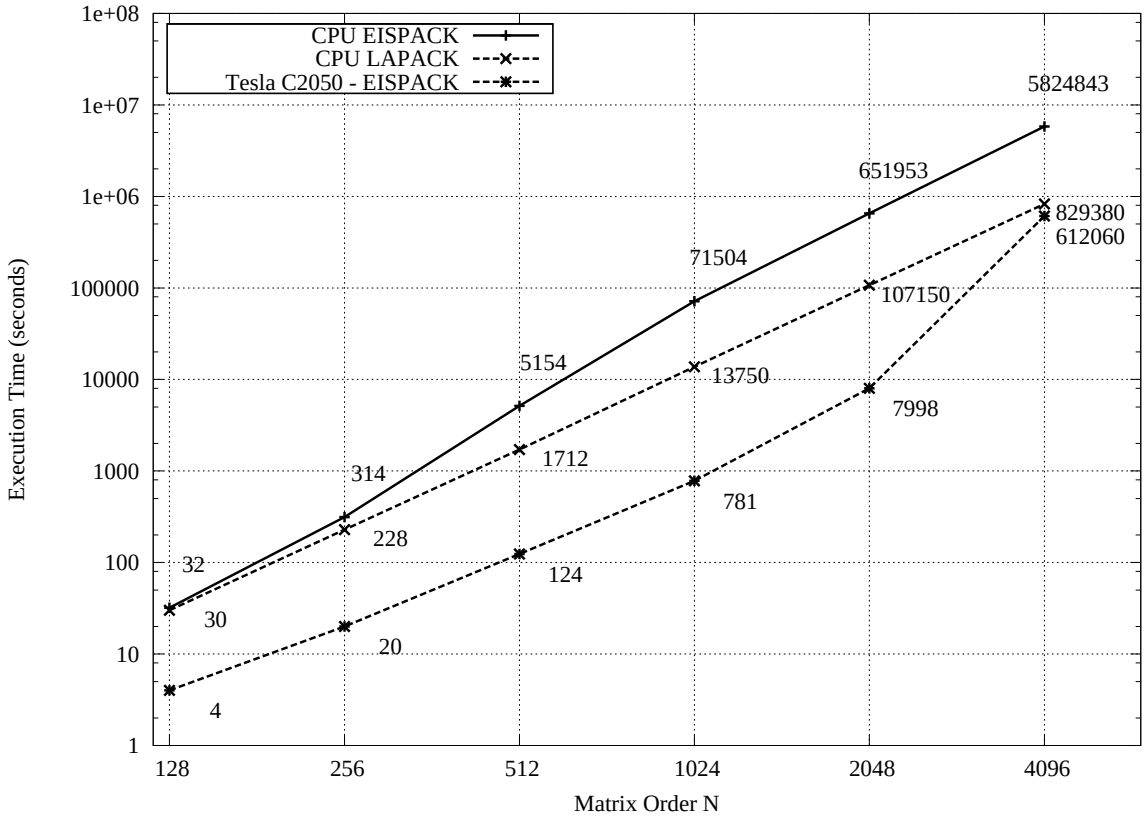

Fig. 2. Execution time for 1000 double precision Hermitian matrices of order N with (i) the current EISPACK CPU implementation, (ii) LAPACK on CPU and (iii) the test EISPACK implementation on a NVIDIA Tesla C2050 GPU.

Parameter	Value
Number of CUDA Cores	448
Frequency of CUDA Cores	1.15GHz
Double Precision floating point performance (peak)	515 Gflops
Single Precision floating point performance (peak)	1.03 Tflops
Total Dedicated Memory	3GB GDDR5

Table 2. NVIDIA Tesla C2050 GPU Specifications

Performance evaluations are carried out on a 64-bit Dell Precision T7500 Server with 4 Intel Xeon 2GHz CPU cores, 4GB RAM and a NVIDIA Tesla C2050 GPU with a PCI express interface running Version 3.2 of the CUDA SDK on 64-bit Ubuntu 10.04 Linux.

The second generation NVIDIA Tesla C2050 GPU is designed specifically for scientific and numerical computing applications. 14 streaming multiprocessors (SM), each providing 32 streaming processors (SP), offer 448 parallel cores in total. While many earlier GPUs completely lacked double precision support, the Tesla GPU provides improved double-precision floating point performance.

The execution times for 1000 N-order input matrices with EISPACK and LAPACK on a single CPU core and on the GPU are shown in Figure 2. GPU times are collected via the platform timers and are inclusive of memory transfer overhead.

Within a critical window ($N = 512 - 2048$), the current GPU routine yields performance increases of between $50 - 100\times$ over the previous EISPACK implementation, a result of performance gains at both thread and block levels. As the matrix order increases, the GPU memory is able to accommodate fewer matrices to provide any block-level performance advantage and execution resources begin to idle. Therefore, the scalability of the approach is restricted for higher values of N by the hard limit that memory places on GPU occupancy despite the still-observable benefits of thread-level parallelism.

The superior LAPACK cache behaviour delivers consistently higher performance over EISPACK for larger values of N. While equivalent routines in both LAPACK and EISPACK are of storage order $O(n^2)$, LAPACK reuses the same input matrix memory for output and is therefore more memory efficient.

6 Discussion

For the intended neutron scattering application, good performance within the critical window will be sufficient to allow an order of magnitude advance in the size, complexity or grid refinement of the INS models.

In the long-term, it is expected that subsequent GPU models will offer improved memory characteristics and deliver higher performance. Furthermore, the need for migration to LAPACK as larger systems are modeled is evident.

The intention of this work has been to create an efficient GPU port that meets the need created by SCATTER and that is based on the established numerical EISPACK code. We anticipate the emergence of standard numerical libraries for the GPU that are based on efficient algorithms oriented towards the particular strengths of the platform. A very recent release of the MAGMA library introduces a Hermitian eigensolver for hybrid multicore-CPU-GPU configurations that is based on an alternative divide-and-conquer algorithm. The suitability of this LAPACK-based version is being evaluated. However, our initial observations indicate that MAGMA performance is optimised for very large values of N, outside the critical window identified previously.

7 Conclusions

The particular applicability of INS to the study of nano-materials has led to increasing popularity for structural determination in the materials science community. Libraries of mathematical routines remain the foundation of these applications and it is important to establish and maintain efficient implementations. We have demonstrated the substantial performance potential of the GPU in INS modelling and similar applications that rely on significant numerical computation.

The current implementation remains in the tuning and testing stages and performance improvements are probable. Testing and deployment in a multi-GPU cluster configuration is intended during re-integration of the new eigenanalysis routines with GULP, the SCATTER host application [6], before simulations with actual INS models are evaluated.

Further work will investigate other computationally intensive aspects of INS modelling that will benefit from GPU acceleration. This includes derivation of the dynamical matrix and nearest-neighbour search.

The challenge of determining optimal parameters for launch configuration and performance tuning presents an opportunity to apply heuristic techniques. Ultimately, we seek to investigate deployment of the neutron scattering program for complex models in large dynamic GPU-accelerated heterogeneous environments and techniques for improving co-operative CPU-GPU throughput. This will combine CPU, GPU-EISPACK and, prospectively, MAGMA in an adaptive framework that optimises for performance by balancing between alternative execution paths. Multiple implementations of GPU kernels, optimised for various problem scales, become a means of achieving this performance balance.

Acknowledgements

The authors would like to thank the Partnership for Advanced Computing in Europe (PRACE) for their support and grant of computing time in the SARA supercomputing facilities. The PRACE project

receives funding from the EU's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. RI-211528. The Authors would also like to acknowledge the support of a collaboration travel grant awarded by the STFC Collaborative Computational Project 5 (CCP5). One of the authors (Roach) would like to acknowledge the support of EPSRC (EP/G049130) in the development of the SCATTER code.

The authors would like to express their gratitude to the NVIDIA Corporation for the donation through the Professor Partnership programme of the GPU Tesla equipment employed in this work. Our appreciation to Julian Gale of Curtin University (Australia) for making the GULP source code available.

References

1. Bethel, E., van Rosendale, J., Southard, D., Gaither, K., Childs, H., Brugger, E., Ahern, S.: Visualization at Supercomputing Centers: The Tale of Little Big Iron and the Three Skinny Guys. IEEE Computer Graphics and Applications 31(1), 90–95 (2011)
2. Cline, A.K., Meyering, J.: Converting eispack to run efficiently on a vector processor. Tech. rep., Pleasant Valley Software, Austin, Texas (1991)
3. Dongarra, J.J., Duff, I.S., Sorensen, D.C., van der Vorst, H.A.: Numerical linear algebra for high-performance computers. SIAM, 2nd edn. (1998)
4. Du, P., Weber, R., Luszczek, P., Tomov, S., Peterson, G., Dongarra, J.: From CUDA to OpenCL: Towards a performance-portable solution for multi-platform GPU programming. Parallel Computing (2011), in Press. doi:10.1016/j.parco.2011.10.002
5. Feldman, S.: A Fortran to C converter. In: ACM SIGPLAN Fortran Forum. vol. 9, pp. 21–22. ACM (1990)
6. Gale, J.D., Rohl, A.L.: The general utility lattice program (GULP). Molecular Simulation 29(5), 291–341 (2003)
7. Garba, M., González-Vélez, H.: Towards ad-hoc GPU acceleration of parallel eigensystem computations. In: ECMS 2011: 25th European Conference on Modelling and Simulation. ECMS, Krakow, Poland (Jun 2011)
8. Garba, M., González-Vélez, H., Roach, D.: Parallel computational modelling of inelastic neutron scattering in multi-node and multi-core architectures. In: 11th IEEE Int Conf on High Performance Computing and Communications. pp. 509–514. IEEE, Melbourne (Sep 2010)
9. Govindaraju, N.K., Larsen, S., Gray, J., Manocha, D.: A memory model for scientific algorithms on graphics processors. In: SC'06: ACM/IEEE Conf on Supercomputing. p. 6. IEEE, Tampa (Nov 2006)
10. Kirk, D., Wen-mei, W.: Programming massively parallel processors: A Hands-on approach. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA (2010)
11. Lee, S., Min, S.J., Eigenmann, R.: Openmp to gpgpu: a compiler framework for automatic translation and optimization. SIGPLAN Not. 44, 101–110 (February 2009)
12. Moler, C.B., Stewart, G.W.: An algorithm for generalized matrix eigenvalue problems. SIAM Journal on Numerical Analysis 10(2), 241–256 (1973)
13. Nickolls, J., Buck, I., Garland, M., Skadron, K.: Scalable parallel programming with CUDA. Queue 6(2), 40–53 (2008)
14. Nvidia Corporation: NVIDIA CUDA C Programming Best Practices Guide. Manual Version 2.3 (2009), available from: `http://developer.nvidia.com/` (Last Accessed: 1 Feb 2011)
15. Roach, D., Ross, K., Gale, J.D.: The application of coherent inelastic neutron scattering to the study of polycrystalline materials. Physical Review B (2010), submitted for publication
16. Roach, D.L., Gale, J., Ross, D.: Scatter: A New Inelastic Neutron Scattering Simulation Subroutine for GULP. Neutron News 18(3), 21–23 (2007)
17. Smith, B.T., Boyle, J.M., Dongarra, J., Garbow, B.S., Ikebe, Y., Klema, V.C., Moler, C.B.: Matrix Eigensystem Routines - EISPACK Guide, LNCS, vol. 6. Springer-Verlag (1976)
18. Tomov, S., Dongarra, J., Baboulin, M.: Towards dense linear algebra for hybrid GPU accelerated manycore systems. Parallel Computing 36(5-6), 232–240 (2010)
19. Tomov, S., Nath, R., Ltaief, H., Dongarra, J.: Dense linear algebra solvers for multicore with GPU accelerators. In: IPDPS 2010 Workshops. pp. 1–8. IEEE, Atlanta (Apr 2010)
20. Ueng, S.Z., Lathara, M., Baghsorkhi, S., Hwu, W.m.: CUDA-Lite: Reducing GPU programming complexity. In: Languages and Compilers for Parallel Computing (revised papers). Lecture Notes in Computer Science, vol. 5335, pp. 1–15. Springer-Verlag, Edmonton (Jul 2008)
21. Vázquez, F., Fernández, J.J., Garzón, E.M.: A new approach for sparse matrix vector product on NVIDIA GPUs. Concurrency and Computation: Practice and Experience 23(8), 815–826 (2011)