ISSN:0975-9646

A Survey on Performance Modelling and Optimization Techniques for SpMV on GPUs

Ms. Aditi V. Kulkarni[#1], Prof. C. R. Barde[*2]

[#]*Student, Department of Computer Engineering, G.E.S's. R.H. Sapat College of Engineering Nashik, Affiliated to University of Pune, India.*

[*]*Assistant Professor, Department of Computer Engineering, G.E.S's. R.H. Sapat College of Engineering Nashik, Affiliated to University of Pune, India.*

***Abstract*— Sparse Matrix is a matrix consisting of very few non-zero entries. Large sparse matrices are often used in engineering and scientific operations. Especially sparse-matrix vector multiplication is an important operation for solving linear system and partial differential equations. However, there is a possibility that even though the matrix is partitioned and stored appropriately, the performance achieved is not significant. Hence a need arises to address these issues. System proposes an integrated analytical and profile based performance modelling that accurately predicts the kernel execution time of various SpMV CUDA kernels and also that of a given target sparse-matrix. Based on this the designed optimal solution auto-selection algorithm automatically reports the SpMV optimal solution for a target sparse-matrix. System was evaluated on NVIDIA Tesla C2050 and significant results were obtained. Proposed system would like enhance the existing system by trying the same on different SpMV CUDA kernels as well look for optimization. Proposed system would also like to try and execute the same on multi-GPU kernels. Proposed system would also like to evaluate the existing system on other NVIDIA GPU such as the NVIDIA GeForce GT 750M card. This paper presents a survey of various performance modelling and optimization techniques for SpMV CUDA kernels on GPUs. It also presents a survey of the various SpMV CUDA kernel implementation techniques**.

***Keywords*—SpMV, GPU, CUDA, performance modelling, optimization.**

I. INTRODUCTION

Sparse Matrix is a matrix consisting of very-few non-zero elements. Large sparse-matrices are used in various engineering and scientific applications. Sparse matrix-vector multiplication is a very important operation when it comes to solving linear system and partial differential equations. When solving matrix-vector multiplication operations the term Ax of the equation Ax=y needs to be computed iteratively, which is tedious when it comes to large sparse-matrices. Hence a need for storing and partitioning the matrix arises. Again after storing and partitioning the matrix, the performance achieved is not significant. Hence the need to address these issues as well. Various SpMV CUDA kernel implementations, performance modelling and optimization techniques have been proposed.

In this paper, the following sections present an extensive literature survey of the various performance modelling and optimization techniques for SpMV CUDA kernels on GPUs. It also presents a survey of various SpMV CUDA kernel implementations techniques on GPUs. A kernel in CUDA is a simple single program implemented and executed using parallel thread. But a prior to it a brief information on sparse-matrix, matrix storage and sparse-matrix vector multiplication is presented.

A. Sparse Matrix

A sparse matrix is a matrix in which very few elements are non-zero as shown in figure 1. On the contrary, if very few elements are zero, then the matrix is a dense matrix. The fraction of zero elements in a matrix is called the sparsity while the fraction of non-zero elements is called dense. Sparsity is used in combinatorics and applications like network theory. Large sparse matrices are often used in engineering and scientific operations. Special algorithms and data structures are required to store and manipulate sparse-matrices on a computer. This is because standard dense-matrix structures and algorithms are slow and inefficient. Sparse data is more easily compressed and thus require significantly less storage. [24].

B. How is a sparse-matrix stored?

A matrix is stored as a two-dimensional array. Each entry in the array represents an element $a(i, j)$ of the matrix and is accessed by the two indices i and j where, i is the row index, numbered from top to bottom, and j is the column index, numbered from left to right. For a $m \times n$ matrix, the amount of memory required to store the matrix in this format is proportional to $m \times n$. [24].

C. Sparse Matrix Vector Multiplication

Sparse matrix-vector multiplication (SpMV) of the form $y = Ax$ is a widely used computational kernel existing in many scientific applications. The input matrix A is sparse. The input vector x and the output vector y are dense. In case of repeated $y = Ax$ operation involving the same input matrix A but possibly changing numerical values of its elements, A can be preprocessed to reduce both the parallel and sequential run time of the SpMV kernel. [25].

II. RELATED WORK

A. SpMV CUDA Kernel Implementations

J. Bolz et al. [5]. This work proposes and implements two basic SpMV CUDA computational kernels viz. a sparse matrix conjugate gradient solver and a regular-grid multigrid solver. Through this work it was observed that real-time applications ranging from mesh smoothing and parameterization to fluid solvers and solid mechanics can

greatly benefit from these. As a matter of proof the implementation was used in the example applications of geometric flow and fluid simulation running on NVIDIA's GeForce FX. [5].

$$A = \begin{bmatrix} 1 & 7 & 0 & 0 \\ 0 & 2 & 8 & 0 \\ 5 & 0 & 3 & 9 \\ 0 & 6 & 0 & 4 \end{bmatrix}$$

DIA format:

$$data = \begin{bmatrix} * & 1 & 7 \\ * & 2 & 8 \\ 5 & 3 & 9 \\ 6 & 4 & * \end{bmatrix} \qquad offsets = [-2 \;\; 0 \;\; 1]$$

ELL format:

$$data = \begin{bmatrix} 1 & 7 & * \\ 2 & 8 & * \\ 5 & 3 & 9 \\ 6 & 4 & * \end{bmatrix} \qquad indices = \begin{bmatrix} 0 & 1 & * \\ 1 & 2 & * \\ 0 & 2 & 3 \\ 1 & 3 & * \end{bmatrix}$$

CSR format:

$$ptr = [0 \;\; 2 \;\; 4 \;\; 7 \;\; 9]$$
$$indices = [0 \;\; 1 \;\; 1 \;\; 2 \;\; 0 \;\; 2 \;\; 3 \;\; 1 \;\; 3]$$
$$data = [1 \;\; 7 \;\; 2 \;\; 8 \;\; 5 \;\; 3 \;\; 9 \;\; 6 \;\; 4]$$

COO format:

$$row = [0 \;\; 0 \;\; 1 \;\; 1 \;\; 2 \;\; 2 \;\; 2 \;\; 3 \;\; 3]$$
$$indices = [0 \;\; 1 \;\; 1 \;\; 2 \;\; 0 \;\; 2 \;\; 3 \;\; 1 \;\; 3]$$
$$data = [1 \;\; 7 \;\; 2 \;\; 8 \;\; 5 \;\; 3 \;\; 9 \;\; 6 \;\; 4]$$

Figure.1. Sparse matrix representations for a simple example matrix A. Padding entries (marked _) are set to zero. Courtesy: Bell and Garland [2].

N. Bell and M. Garland [2]. This work proposed and implemented SpMV CUDA kernels for some well-known storage formats such as DIA, CSR, ELL, COO and HYB. [2]. They are as follows:

1) Diagonal Format

The diagonal format is formed by two arrays: data and offsets. The data array stores the nonzero values, and the offsets array, stores the offset of each diagonal from the main diagonal. Diagonals above the main diagonal have positive offsets and below the main diagonal have negative offsets, accordingly. [2]

2) ELL Format

The ELL format is more general than DIA as the non-zero columns do not follow any particular pattern. It is in particular well-suited for vector architectures. An $M \times N$ sparse matrix with at the most K nonzeros per row is stored as a dense $M \times K$ array data of nonzeros and array indices of column indices. All rows are zero-padded to length K. [2]

3) Compressed Sparse Row Format

The compressed sparse row (CSR) format explicitly stores column indices and nonzero values in arrays indices and data. A third array of row pointers, ptr, allows the CSR format to represent rows of varying length. [2]

4) Coordinate Format

The coordinate (COO) format is a storage scheme. The arrays: row, indices, and data store the row indices, column indices, and values, respectively, of the nonzero entries. It is assumed that entries with the same row index are stored contiguously. [2]

5) Hybrid Format

The ELLPACK format is well-suited to vector and SIMD architectures, but its efficiency rapidly degrades when the number of nonzeros per matrix row varies. On the contrary, the storage efficiency of the COO format is invariant to the distribution of nonzeros per row, and the use of segmented reduction makes its performance largely invariant as well. To gain the advantages of both, it is combined into a hybrid ELL/COO format. The purpose of the hybrid (HYB) format is to store the typical number of nonzeros per row in the ELL data structure and the remaining entries of exceptional rows in the COO format. The typical number of nonzeros per row is often known a priori, as in the case of manifold meshes, and the ELL portion of the matrix is readily extracted. However, in the general case this number must be determined directly from the input matrix. [2]

B. SpMV Optimization Techniques

The following is a survey of various SpMV Optimization techniques.

P. Guo et al. [4]. In this work, an innovative performance-model driven approach for partitioning sparse matrix into appropriate formats, and auto-tuning configurations of CUDA kernels to improve the performance of SpMV on GPUs is presented. The following are the features of the system: (a) It proposes an empirical CUDA performance model to predict the execution time of SpMV CUDA kernels. (b) It designs and implements a model-driven partitioning framework to predict how to partition the target sparse matrix into one or more partitions and transform each partition into appropriate storage format, which is based on the fact that the different storage formats of sparse matrix can significantly affect the performance of SpMV, and (c) It integrates the model-driven partitioning with the existing auto-tuning framework [3] to automatically adjust CUDA-specific parameters to optimize performance on specific GPUs. The approach was evaluated on 14 matrices using NVIDIA's GeForce GTX 295 and it was observed that compared to the NVIDIA's existing implementations, the approach showed a substantial performance improvement. It had 222%, 197%, and 33% performance improvement on the average for CSR vector kernel, ELL kernel and HYB kernel, respectively. [4]

J. Kurzak, W. Alvaro, and J. Dongarra [6].In this work, optimized single precision matrix multiplication kernels are presented for the short vector Single Instruction Multiple Data architecture of the Synergistic Processing Element of IBM's CELL BE processor. The operations $C = C - A \times B^T$ and $C = C - A \times B$ are implemented for matrices of size 64×64 elements. In the former case a performance of 24.09 Gflop/s which is 94% of peak performance is reported whereas in the latter case the performance of 25.55 Gflop/s is reported, or 99.80% of the peak, using as little as 5.9 kB of storage for code and auxiliary data structures. [6].

E. J. Im, K. Yelick, and R. Vuduc [7]. In this work, the optimization of two operations viz. a sparse matrix times a dense vector and a sparse matrix times a set of dense vectors are discussed. It was found that register level

optimizations are effective for matrices arising in certain scientific simulations, especially finite-element problems. The cache level optimizations find importance when the vector used in multiplication is larger than the cache size, especially for matrices in which the nonzero structure is random. For applications involving multiple vectors, reorganizing the computation to perform the entire set of multiplications as a single operation produces significant speedups. Also the different optimizations and parameter selection techniques are described. These techniques are evaluated on several machines using over 40 matrices taken from broad set of application domains. The results showed speedups of up to 4x for the single vector case and up to 10x for the multiple vector case. [7]

M.M. Baskaran and R. Bordawekar [8]. In this work, the various challenges in developing a high-performance SpMV Kernel on CUDA GPUs are evaluated using the CUDA programming model and also optimizations for the same are proposed. The optimizations included: (a) exploitation of synchronization-free parallelism, (b) optimization of thread mapping based on the affinity towards optimal memory access pattern, (c) optimized off-chip memory access to tolerate the high access latency, (d) exploitation of data reuse. The system was evaluated on two classes of NVIDIA GPUs viz. GeForce 8800 GTX and GeForce GTX 280. The system performance was compared with that of existing parallel SpMV implementations viz. (a) one from NVIDIA's SpMV Library, (b) one from NVIDIA's CUDPP library, and (c) one implemented using optimal segmented scan primitive. It was found that the system outperformed the CUDPP and segmented scan implementations by a factor of 2 to 8. It was also found that the system achieved 15% improvement over NVIDIA's SpMV Library. [8].

J. Demmel et al. [9]. In this work, two software systems viz. ATLAS (Automatically Tuned Linear Algebra Software) and BeBOP (formerly known as SPARSITY version 2) are presented for dense and sparse linear algebra kernels respectively. These softwares' use heuristic search strategies for exploring the architecture parameter space. For optimization of these kernels the AEOS (Automated Empirical Optimization of Software) is presented for both the softwares ATLAS and BeBOP, for dense and sparse operations respectively. It was found that through a combination of automatic code generation and hand-coded optimizations these packages deliver several factors improvement over what even the best of compilers can achieve on reference implementations. The SALSA package which uses statistical data modelling as a tool for automatic algorithm choice is also presented. The results obtained showed great promise for the future of portable adaptive high-performance libraries. [9].

P. Guo and L. Wang [10]. In this work, an auto-tuning framework that can automatically compute and select CUDA parameters for SpMV to obtain the optimal performance on specific GPUs is presented. The framework was evaluated on two NVIDIA GPU platforms viz. GeForce 9500 GTX and GeForce GTX 295. It was found that, for GeForce 9500 GTX, the auto-tuning framework had 237% performance improvement on the average, and the median improvement was 278% and for GeForce GTX 295, the auto-tuning framework had 33% performance improvement on the average, and the median improvement was 25.6%, as compared to NVIDIA's implementation, for both cases. [10].

F. Vazquez et al. [11]. In this work, new implementations of SpMV for GPUs called ELLR-T is proposed and evaluated. They are based on the format ELLPACK-R, which allows storage of the sparse matrix in a regular manner. The comparative evaluation with other systems showed that the performance achieved by ELLR-T was the best after an extensive study on a set of representative test matrices. A comparison of ELLR-T on a GeForce GTX 285 had revealed that acceleration factors of up to 30x can be achieved in comparison to optimized implementations of SpMV which exploit state-of-the-art superscalar processors. [11].

D. Grewe and A. Lokhmotov [12]. In this work, a system-independent representation of sparse matrix formats is presented. It allowed a compiler to generate efficient, system-specific code for sparse matrix operations. To show the viability of such a representation a compiler that generates and tunes code for sparse matrix-vector multiplication (SpMV) on GPUs was developed. Additionally, the format description also can be used to automatically generate vectorized code to fully exploit the capabilities of vector-architectures. The framework was evaluated on six state-of-the-art matrix formats and it was found that the generated code performed similar to or better than hand-optimized code. It was observed that for every single matrix the vector version clearly outperformed the scalar version, with speedups of up to a factor of 4. The average performance gain of vectorizing the code were 1.6x (geometric mean). [12].

Z. Wang et al. [13]. In this work, optimized implementation of sparse matrix-vector multiplication on NVIDIA GPUs using CUDA programming model is presented. Three optimizations including: optimized CSR storage format, optimized threads mapping, and avoid divergence judgment are outlined to improve the performance of SpMV kernels. The optimizations were evaluated on GeForce 9600 GTX, connected to Windows XP 64-bit system. In comparison with NVIDIA's SpMV library and NVIDIA's CUDDPA library, it was observed that the results showed that optimizing sparse matrix-vector multiplication on CUDA achieved better performance than other SpMV implementations. [13].

X. Yang et al. [14]. In this work, a novel non-parametric and self-tunable approach to data representation for computing SpMV, particularly targeting sparse matrices representing power-law graphs is presented. It was observed that by using the real web graph data, the representation scheme, coupled with a novel tiling algorithm, yielded significant benefits over the state of the art GPU efforts on a number of core data mining algorithms such as PageRank, HITS and Random Walk with Restart. On these algorithms, it was observed that the best kernel was 1.8 to 2.1 times faster than an industrial strength GPU competitor and from 18 to 32 times faster than a similar CPU implementation. [14].

S. Williams et al. [23]. In this work, sparse matrix-vector multiply (SpMV) is examined across a broad spectrum of multicore designs. The experiments were evaluated on the homogeneous AMD dual-core and Intel quad-core designs, the heterogeneous STI Cell, as well as the first scientific study of the highly multithreaded Sun Niagara2 platforms. Several optimization strategies effective for the multicore environment are presented, and significant performance improvements compared to existing state-of-the-art serial and parallel SpMV implementations are demonstrated. [23].

C. SpMV Performance Modelling Techniques

There is extensive work on performance models.

P. Guo and L. Wang [3]. This work presents an integrated analytical and profile-based CUDA performance modelling approach that accurately predicts the kernel execution times of sparse matrix-vector multiplication for CSR, ELL, COO, and HYB SpMV CUDA kernels. The experiments were conducted on a collection of 8 widely-used testing matrices on NVIDIA Tesla C2050 and the execution times predicted by the model matched the measured execution times of NVIDIA's SpMV implementations. For 29 out of 32 test cases, the performance differences were observed under or around 7%. For the rest 3 test cases, the differences were observed between 8% and 10%. For CSR, ELL, COO, and HYB SpMV kernels, the differences were observed as 4.2%, 5.2%, 1.0%, and 5.7% on the average, respectively. [3].

S. Ryoo et al. [15]. In this work, two metrics i.e., efficiency and utilization are introduced to reduce optimization space. The model focuses on pruning optimization space to reduce tuning time for a program. An approach for attacking the complexity of optimizing code for the NVIDIA GeForce 8 Series is proposed. Because predicting the performance effects of program optimizations is difficult, developers or compilers may need to experiment to find the configuration with the best performance. By plotting the configurations and examining only those configurations on a Pareto-optimal curve, the search space was ably reduced by up to 98% without missing the configuration with the highest performance. [15].

J.W. Choi, A. Singh, and R.W. Vuduc [16]. In this work, a performance model-driven framework for automated performance tuning (autotuning) of sparse matrix-vector multiply (SpMV) on systems accelerated by graphics processing units (GPU) is presented. The study consists of two parts. First, several carefully hand-tuned SpMV implementations for GPUs is described, identifying key GPU-specific performance limitations, enhancements, and tuning opportunities. It was observed that these implementations, that included variants on classical blocked compressed sparse row (BCSR) and blocked ELLPACK (BELLPACK) storage formats, matched or exceeded state-of-the-art implementations. It was observed that the best BELLPACK implementation achieved up to 29.0 Gflop/s in single-precision and 15.7 Gflop/s in double precision on the NVIDIA T10P multiprocessor (C1060), enhancing prior state-of-the-art unblocked implementations by up to 1.8x and 1.5x for single- and double precision respectively. However, achieving this level of performance required input matrix-dependent parameter tuning. Thus, in the second part of the study, a performance model that guided tuning was developed. This model required offline measurements and run-time estimation, but more directly modelled the structure of multithreaded vector processors like GPUs. It was observed that the model could identify the implementations that achieved within 15% of those found through exhaustive search. [16].

D. Schaa and D. Kaeli [17]. In this work, a modelling framework is designed that produces accurate estimates when moving single-GPU applications to a multiple-GPU platform. The approach develops a set of performance equations that capture many of the latencies and dependencies introduced in a multiple-GPU environment. The system was tested on six applications and the execution time across multiple GPU applications with an average difference of 11% was estimated when compared to actual execution times. The validation study included applications that have a wide range of execution durations. [17].

S. Xu, W. Xue, and H. Lin [18]. In this work, optimization of SpMV based on ELLPACK from two aspects: (a) enhanced performance for the dense vector by reducing cache misses, and (b) reduce accessed matrix data by index reduction, is proposed. With matrix bandwidth reduction techniques, both cache usage enhancement and index compression can be enabled. For GPU with better cache support, differentiated memory access scheme to avoid contamination of caches by matrix data is proposed. Performance evaluation showed that the combined speedups of proposed optimizations for GT-200 are 16% (single-precision) and 12.6% (double-precision) for GT-200 GPU, and 19% (single-precision) and 15% (double-precision) for GF-100 GPU. [18].

Y. Zhang and J. D. Owens [19]. In this work, a microbenchmark-based performance model for NVIDIA GeForce 200-series GPUs is developed. The model identifies GPU program bottlenecks and quantitatively analyses performance, and thus allows programmers and architects to predict the benefits of potential program optimizations and architectural improvements. The microbenchmark-based approach is used to develop a throughput model for three major components of GPU execution time: the instruction pipeline, shared memory access, and global memory access. Because the model is based on the GPU's native instruction set, the performance can be predicted with a 5–15% error. To demonstrate the usefulness of the model, three representative real-world and already highly-optimized programs: dense matrix multiply, tridiagonal systems solver, and sparse matrix vector multiply were analysed. The model provided detailed quantitative analysis on performance, which enabled understanding of the configuration of the fastest dense matrix multiply implementation and to optimize the tridiagonal solver and sparse matrix vector multiply by 60% and 18% respectively. [19].

S.S. Baghsorkhi et al. [20]. In this work, a compiler-based approach to application performance modelling on GPU architectures is presented. The model is equipped with an efficient symbolic evaluation module to determine the effects of the structural conditions and complex memory

access expressions on the performance of a GPU kernel. This approach combines the effects of different performance factors into a coherent framework. In cases where it cannot statically determine performance information, a parametric latency is derived which can be customized later, according to the kernel inputs. In the case of data dependent conditions or access patterns, it employs a light-weight dynamic instrumentation approach to specialize the parametric latency. This model allows a compiler to determine the relative merits of parallel kernel configurations without running all the variations. Also, the model identifies the bottlenecks and can guide the compiler through the optimization process. The performance model was validated on the NVIDIA GPUs using CUDA for the matrix multiply, prefix sum scan, FFT, and sparse matrix-vector benchmarks. The evaluation showed that there was good agreement between predicted and observed performance rankings for the various tuning versions of these kernels and that the model captured the effect of all major performance factors for GPU architecture. [20].

S. Hong and H. Kim [21]. This work proposes and evaluates a memory parallelism aware analytical model that estimates execution cycles for the GPU architecture. The key idea of the analytical model is to find the maximum number of memory warps that can execute in parallel, a metric which is called MWP, to estimate the effective memory instruction cost. The model calculates the estimated CPI, that provides a simple performance estimation metric for programmers and compilers to decide whether they should perform certain optimizations or not. The evaluations show that the geometric mean of absolute error of the analytical model on microbenchmarks is 5.4% and on GPU computing applications is 13.3%. [21].

K. Kothapalli et al. [22]. This work presents a performance model that combines several known models of parallel computation viz. BSP, PRAM, and QRQW. The model encompasses the various facets of the GPU architecture like scheduling, memory hierarchy and pipelining among others. The usage of the model and its accuracy was illustrated with three case studies viz. Matrix Multiplication, List Ranking, and histogram generation. [22]

III. PROPOSED SYSTEM

System proposes an integrated analytical and profile based performance modelling that accurately predicts the kernel execution time of various SpMV CUDA kernels and also that of a given target sparse-matrix. Based on this the designed optimal solution auto-selection algorithm automatically reports the SpMV optimal solution for a target sparse-matrix. System was evaluated on NVIDIA Tesla C2050 and significant results were obtained [1].Proposed system would like enhance the existing system by trying the same on different SpMV CUDA kernels as well look for optimization. Proposed system would also like to try and execute the same on multi-GPU kernels. Proposed system would also like to evaluate the existing system on other NVIDIA GPU cards such as NVIDIA GeForce GT 750M Notebook GPU.

IV. CONCLUSIONS

Sparse matrix thus is a matrix which has very few nonzero elements. Sparse-matrix vector multiplication is a tedious operation and when carried out iteratively becomes more difficult. GPU eases this job however efficient storage strategies, performance modelling and optimization techniques are needed. Various SpMV CUDA kernels have been proposed in this regard however still significant achievement is not seen. System proposes and integrated performance modelling and optimization analysis system for SpMV CUDA kernels. Proposed system will enhance the existing system and evaluate it on different NVIDIA GPU card such as GeForce GT 750M and also try and extend it to multi-GPU kernels. A detailed survey of all possible SpMV performance modelling and optimization techniques is presented in this work. We assume that the above survey helps to better understand SpMV CUDA kernel and it implementation including its performance modelling and its optimization in a better way.

ACKNOWLEDGEMENT

We are glad to express our sentiments of gratitude to all who rendered their valuable guidance to us. We would like to express our appreciation and thanks to Prof. Dr. P. C. Kulkarni, Principal, G. E. S. R. H. Sapat College of Engg., Nashik. We are also thankful to Prof. N. V. Alone, Head of Department, Computer Engg., G. E. S. R. H. Sapat College of Engg., Nashik. We thank the anonymous reviewers for their comments.

We also acknowledge all the scientists, researchers, scholars and the SpMV CUDA kernel development community and fraternity who have taken efforts towards the research and development of the SpMV CUDA kernel, its implementation, optimization and performance modelling.

REFERENCES

[1] P. Guo, L. Wang and P. Chen, "A Performance Modeling and Optimization Analysis Tool for Sparse-Matrix Vector Multiplication on GPUs", *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 5, pp. 1112-1123, May 2014.

[2] N. Bell and M. Garland, "Implementing Sparse Matrix-Vector Multiplication on Throughput-Oriented Processors," *Proc. Conf. High Performance Computing Networking, Storage and Analysis* (SC'09), pp. 1-11, 2009.

[3] P. Guo and L. Wang, "Accurate CUDA Performance Modeling for Sparse Matrix-Vector Multiplication," *Proc. IEEE Int'l Conf. High Performance Computing and Simulation (HPCS '12)*, pp. 496-502, July 2012.

[4] P. Guo, H. Huang, Q. Chen, L. Wang, E.-J. Lee, and P. Chen, "A Model-Driven Partitioning and Auto-Tuning Integrated Framework for Sparse Matrix-Vector Multiplication on GPUs," *Proc. TeraGrid Conf. Extreme Digital Discovery (TG '11)*, pp. 2:1-2:8, 2011.

[5] J. Bolz, I. Farmer, E. Grinspun, and P. Schroder, "Sparse Matrix Solvers on the GPU: Conjugate Gradients and Multigrid," *ACM Trans. Graphics*, vol. 22, no. 3, pp. 917-924, 2003.

[6] J. Kurzak, W. Alvaro, and J. Dongarra, "Optimizing Matrix Multiplication for a Short-Vector Simd Architecture-Cell Processor," *J. Parallel Computing*, vol. 35, no. 3, pp. 138-150, 2009.

[7] E.-J. Im, K. Yelick, and R. Vuduc, "Sparsity: Optimization Framework for Sparse Matrix Kernels," *Int'l J. High Performance Computing Applications*, vol. 18, no. 1, pp. 135-158, 2004.

[8] M.M. Baskaran and R. Bordawekar, "Optimizing Sparse Matrix-Vector Multiplication on GPUs," Research Report RC24704, IBM TJ Watson Research Center, Dec. 2008.

[9] J. Demmel, J. Dongarra, V. Eijkhout, E. Fuentes, A. Petitet, R.C.W.R. Vuduc, and K. Yelick, "Self-Adapting Linear Algebra Algorithms and Software," *Proc. IEEE*, vol. 93, no. 2, pp. 293-312, Feb. 2005.

[10] P. Guo and L. Wang, "Auto-Tuning CUDA Parameters for Sparse Matrix-Vector Multiplication on GPUs," *Proc. Int'l Conf. Computational and Information Sciences (ICCIS '10)*, pp. 1154-1157, 2010.

[11] F. Vazquez, G. Ortega, J.J. Fernandez, and E.M. Garzon, "Improving the Performance of the Sparse Matrix Vector Product with GPUs," *Proc. 10th IEEE Int'l Conf. Computer and Information Technology (CIT '10)*, pp. 1146-1151, 2010.

[12] D. Grewe and A. Lokhmotov, "Automatically Generating and Tuning GPU Code for Sparse Matrix-Vector Multiplication from a High-Level Representation," *Proc. ACM Fourth Workshop General Purpose Processing on Graphics Processing Units (GPGPU-4)*, pp. 12:1-12:8, 2011.

[13] Z. Wang, X. Xu, W. Zhao, Y. Zhang, and S. He, "Optimizing Sparse Matrix-Vector Multiplication on CUDA," *Proc. Second Int'l Conf. Education Technology and Computer (ICETC '10)*, vol. 4, pp. V4-109-V4-113, June 2010.

[14] X. Yang, S. Parthasarathy, and P. Sadayappan, "Fast Sparse Matrix-Vector Multiplication on GPUs: Implications for Graph Mining," *Proc. VLDB Endowment*, vol. 4, no. 4, pp. 231-242, Jan. 2011.

[15] S. Ryoo, C.I. Rodrigues, S.S. Stone, S.S. Baghsorkhi, S.-Z. Ueng, J.A. Stratton, and W.-m.W. Hwu, "Program Optimization Space Pruning for a Multithreaded GPU," *Proc. ACM Sixth Ann. IEEE/ACM Int'l Symp. Code Generation and Optimization (CGO '08)*, pp. 195-204, 2008.

[16] J.W. Choi, A. Singh, and R.W. Vuduc, "Model-Driven Autotuning of Sparse Matrix-Vector Multiply on GPUs," *Proc. 15th ACM SIGPLAN Symp. Principles and Practice of Parallel Programming (PPoPP '10)*, pp. 115-126, 2010.

[17] D. Schaa and D. Kaeli, "Exploring the multiple-GPU Design Space," *Proc. IEEE Int'l Parallel & Distributed Processing Symp. (IPDPS '09)*, pp. 1-12, May 2009.

[18] S. Xu, W. Xue, and H. Lin, "Performance Modeling and Optimization of Sparse Matrix-Vector Multiplication on NVIDIA CUDA Platform," *J. Supercomputing*, vol. 63, pp. 710-721, 2013.

[19] Y. Zhang and J. Owens, "A Quantitative Performance Analysis Model for GPU Architectures," *Proc. IEEE 17th Int'l Symp. High Performance Computer Architecture (HPCA '11)*, pp. 382-393, Feb. 2011.

[20] S.S. Baghsorkhi, M. Delahaye, S.J. Patel, W.D. Gropp, and W. -M.W. Hwu, "An Adaptive Performance Modeling Tool for GPU Architectures," *Proc. 15th ACM SIGPLAN Symp. Principles and Practice of Parallel Programming (PPoPP '10)*, pp. 105-114, 2010.

[21] S. Hong and H. Kim, "An Analytical Model for a GPU Architecture with Memory-Level and Thread-Level Parallelism Awareness," *Proc. 36th ACM Ann. Int'l Symp. Computer Architecture (ISCA '09)*, pp. 152-163, 2009.

[22] K. Kothapalli, R. Mukherjee, M. Rehman, S. Patidar, P. Narayanan, and K. Srinathan, "A Performance Prediction Model for the CUDA GPGPU Platform," *Proc. Int'l Conf. High Performance Computing (HiPC '09)*, pp. 463-472, Dec. 2009.

[23] S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, and J. Demmel, "Optimization of Sparse Matrix-Vector Multiplication on Emerging Multicore Platforms," *Proc. ACM/IEEE Conf. Supercomputing*,2007.

[24] http://en.wikipedia.org/wiki/Sparse_matrix

[25] http://en.wikipedia.org/wiki/Sparse_matrix-vector_multiplication