IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 17, Issue 4, Ver. II (July – Aug. 2015), PP 36-43
www.iosrjournals.org

SpMV Profiling and Optimization Analysis

Aditi V. Kulkarni[1], Prof. C. R. Barde[2]
[1](*Dept. of Computer Engg., G.E.S's R.H. Sapat College of Engineering, Management Studies and Research, Affiliated to Savitribai Phule Pune University, Nasik, India.)*
[2](*Dept. of Computer Engg., G.E.S's R.H. Sapat College of Engineering, Management Studies and Research, Affiliated to Savitribai Phule Pune University, Nasik, India.)*

Abstract: *Sparse matrix-vector multiplication is an important operation when it comes to sparse matrix computations. Very large and sparse matrices are used in many engineering and scientific operations. Hence the matrix needs to be partitioned properly. Even though the matrix is partitioned and stored appropriately there still exists a possibility, the performance achieved is not significant. Thus, the need to address these issues. System proposes an integrated analytical and profile based performance modelling that accurately measures the kernel execution time of various SpMV CUDA kernels for 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. The system is evaluated on NVIDIA GeForce GTX 680 and NVIDIA Quadro 8000. The system is further extended to one more matrix storage format.*

Keywords: *SpMV, GPU, CUDA, performance modeling, optimization analysis.*

I. Introduction

Sparse matrix is a matrix that consists 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. Also the SpMV CUDA Kernel is an important kernel and is an integral part of various iterative solvers such as sparse matrix conjugate gradient solver and a regular-grid multigrid solver [5]. This kernel is computed iteratively and hence there is a need to accurately model the performance of the SpMV CUDA Kernel and optimize the same. Hence the need for storing and partitioning the sparse matrix arises. Again after storing and partitioning the matrix, the performance achieved is not significant. Hence the need to address these issues as well.

The system proposes and integrated performance modeling technique that predicts the kernel execution times of CSR, COO, ELL and HYB [2] kernels for a target sparse matrix. Based on this performance modeling technique the optimized solution i.e. execution time of target sparse matrix, and optimal storage format will be reported using a dynamic-programming based auto-selection algorithm [1].

The following sections describe the methodology in detail. Section 2 describes related work. Section 3 describes the system in detail. Section 4 describes the experimental setup, Section 5 results and Section 6 describes the conclusion, whilst acknowledgement and references follow it.

II. Related Work

SpMV CUDA kernels for various matrix storage formats viz. DIA, COO, CSR, ELL and HYB have been proposed and implemented previously by Bell and Garland [2] and same are used in the system. Apart from these two basic computational kernels viz. a sparse matrix conjugate gradient solver and a regular-grid multigrid solver were proposed and implemented by J. Bolz et al. [5].

Various SpMV CUDA Kernel optimization techniques have been proposed. These include performance-model driven approach for partitioning sparse matrix into appropriate formats, and auto-tuning configurations of CUDA kernels [4]; optimization of single precision matrix multiplication kernels for the short vector SIMD architecture of the SPE of IBM's CELL BE processor [6]; optimization of two operations viz. a sparse matrix times a dense vector and a sparse matrix times a set of dense vectors [7].

Other optimization techniques proposed were: optimization of SpMV Kernel on CUDA GPUs with focus on exploitation of synchronization-free parallelism, optimization of thread mapping based on the affinity towards optimal memory access pattern, optimized off-chip memory access to tolerate the high access latency, and exploitation of data reuse [8]; optimization of ATLAS and BeBOP kernels using the AEOS (Automated Empirical Optimization of Software) approach for dense and sparse operations respectively [9]; auto-tuning framework for automatically computing and selecting CUDA parameters for SpMV [10]; new implementations of SpMV for GPUs called ELLR-T [11]; system-independent representation of sparse matrix formats [12];

optimized implementation of sparse matrix-vector multiplication on NVIDIA GPUs using CUDA programming model is presented with inclusion of optimized CSR storage format, optimized threads mapping, and avoiding divergence judgment [13]; non-parametric and self-tunable approach to data representation for computing SpMV [14]; and examination of sparse matrix-vector multiply (SpMV) across a broad spectrum of multicore designs [23].

Similarly various SpMV CUDA Kernel performance modelling techniques were also proposed. These include 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 (previous work) [3]; pruning optimization space to reduce tuning time for a program [15]; a performance model-driven framework for automated performance tuning (autotuning) of sparse matrix-vector multiply (SpMV) on systems accelerated by graphics processing units (GPU); a modelling framework that produces accurate estimates when moving single-GPU applications to a multiple-GPU platform [17]; and 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 [18].

Other performance modeling techniques include: a microbenchmark-based performance model for NVIDIA GeForce 200-series GPUs [19]; compiler-based approach to application performance modelling on GPU architectures [20].; a memory parallelism aware analytical model that estimates execution cycles for the GPU architecture S. Hong and H. Kim [21]; and a performance model that combines several known models of parallel computation viz. BSP, PRAM, and QRQW [22].

III. Proposed System

3.1 System Architecture

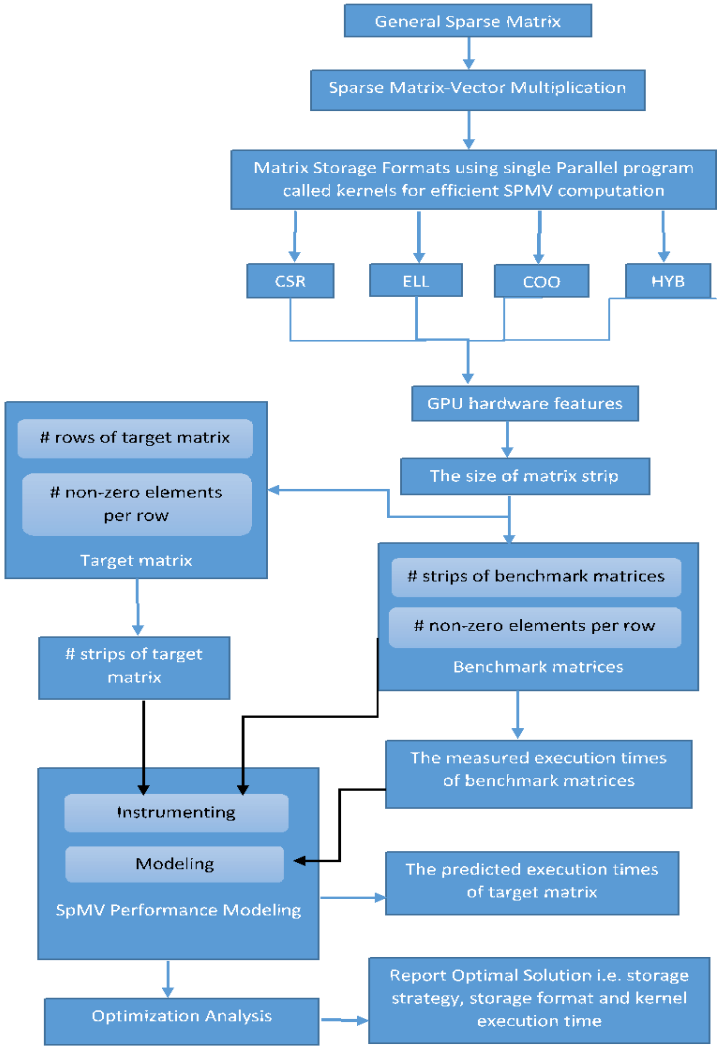

Fig. 1: System Architecture

The Fig 1 shows the system architecture. Initially the various matrix storage formats for a general sparse-matrix that is an integral part of the sparse matrix-vector multiplication computation is implemented using a single parallel program called kernel on GPU. First the performance of the SpMV CUDA kernels will be modelled based on the hardware features of the GPU and then based on the performance modelling the optimal solution will be reported using the dynamic-programming based SpMV optimal solution auto-selection algorithm [1]. The two phases viz. performance modeling and optimization are explained in brief in the following subsections.

3.2 Performance Modeling

The performance modeling comprises of two phases viz. instrumentation and modeling. In the instrumentation phase first the size of matrix strip is calculated then based on the GPU hardware features the benchmark matrices are generated. The properties and execution times are then recorded and are provided as input to the modeling phase. Except for the COO kernel the number of matrix strips and non-zero elements per row for a target matrix are calculated. In the modeling phase the parameterized models are instantiated as per the experimental results of the benchmark matrices. Finally the execution times of SpMV kernel for a target sparse matrix is estimated [1].

3.3 Optimization Analysis

Based on the performance modelling, the optimal solution will be reported using the dynamic-programming based SpMV optimal solution auto-selection algorithm [1]. For a given target sparse matrix, the algorithm will look for all possible storage strategies and checks if the performance can be improved further when the target matrix is partitioned into one or more matrix blocks and each one of them is stored in an appropriate storage format. If such a situation is found then, the optimal solution, including the storage strategy, the storage format for each matrix block, and the predicted overall execution time, will be reported by the algorithm; else, for the entire matrix, if a single storage format has the best SpMV performance, then the storage format, and its corresponding predicted overall execution time, will be reported as the optimal solution [1]. The algorithm is based on the following recursive equation which defines the value of an optimal solution in terms of optimal solution to sub problems [1]

$$T[1,j] = E[1,1] \quad if\ j = 1 \qquad (1)$$

$$T[1,j] = min\left\{ \min_{1 \le k < j} \{T[1,k] + E[k+1,j]\}, E[1,j] \right\}, \quad if\ 2 \le j \le N \ (2)$$

Algorithm: SpMV Optimal Solution Auto-Selection Algorithm [1]

Input: A target sparse matrix and SpMV CUDA Kernel matrix storage formats

Output: SpMV Kernel execution times and optimal solution for target matrix i.e. storage strategy, storage format and target matrix execution time.

Variables and Data Structures:

N is the number of matrix strips;

$E[i,j]$ stores the minimum predicted execution times of the matrix block $[i,j]$;

$F[i,j]$ stores optimal storage format (e.g., CSR, ELL, COO, or HYB) of the matrix block $[i,j]$;

$T[i,j]$ stores the minimum predicted execution time of $\#j$ matrix strips starting from $[1,1]$ to $[j,j]$;

$S[1,j]$ records the index k which splits $\#j$ matrix strips starting from $[1,1]$ to $[j,j]$ into two matrix blocks: $[1,k]$ and $[k+1,j]$. Note that if k equals to j, all strips from $[1,1]$ to $[j,j]$ are in the same matrix block.

Steps:

1: Begin
2: Initialize T [1, 1] ←E [1, 1]
3: Initialize S [1, 1] ←1
4: Initialize j←2
5: repeat
6: for each j in N do
7: Assign T [1, j] ←E [1, j]
8: Assign S [1, j] ←j
9: initialize k←1

10: repeat
11: for each k in j-1 do
12: compute q ← T [1, k] +E [k+1, j]
13: If q<T [1, j] then
14: Assign T [1, j] ←q
15: Assign S [1, j] ←k
16: end if
17: end for
18: end for
19: Print "Optimal Time: T [1, N]"
20: Call procedure Print Optimal Strategy

Procedure: Printing Optimal Strategy
Steps:
1: if j=1
2: Print "Optimal Strategy: [1, 1]"
3: Print "Optimal Format: F [1, 1]"
4: else if S [1, j] =1 then
5: Print "Optimal Strategy: [1, 1], [2, j]"S
6: Print "Optimal Format: F [1, 1], F [2, j]"
7: else if S [1, j] =j then
8: Print "Optimal Strategy: [1, j]"
9: Print "Optimal Format: F [1, j]"
10: else
11: PRINT_OPTIMAL_STRATEGY (S [1, j], S);
12: Print ", [S [1,j]+1, j], F [S [1, j]+1, j]"
13: endif
14: endif
15: Endif

3.4 Extension to other SpMV storage format

The system is further extended to diagonal (DIA) matrix storage format i.e. the execution time of DIA SpMV CUDA kernel for a given target matrix is calculated. From the execution times of the DIA kernel it can be observed that the DIA kernel although is one of the most appropriate matrix storage formats but takes slightly longer time for execution when compared to other SpMV CUDA kernels viz. CSR, COO, ELL and HYB. Thus, it can be loosely inferred that the DIA format is not one of the most preferred formats for the SpMV operation.

IV. Experimental setup

The system is implemented on the NVIDIA GeForce GTX 680 and NVIDIA Quadro 8000 graphics cards using CUDA toolkit 6.5/7.0 and NVIDIA Nsight Eclipse. The CUDA toolkit can be downloaded for free at www.nvidia.com/getcuda. The specifications for NVIDIA GeForce GTX 680 are as given in Table 1.

Table 1. Nvidia Geforce GTX 680 Specifications

CUDA Cores	1536
Base Clock (MHz)	1006
Boost Clock (MHz)	1058
Texture Fill Rate (Billion/Sec)	128.8
Memory speed	6.0 Gbps
Standard Memory Configuration	2048 MB
Memory Interface Width	256-bit GDDR5
Memory Bandwidth (GB/sec)	192.2
Bus Support	PCI Express 3.0
Certified for Windows 8	Yes

4.1 Data Sets

The data sets are the matrices used for study from various engineering and scientific applications together given in "The University of Florida Sparse-Matrix Collection" [24] available for download at http://www.cise.ufl.edu/ research/sparse/matrices/. Examples of some matrices are as given below in Table 2.

Table 2. Examples of Some Matrix Data Sets

Matrix Name	Description	#Rows	#Columns	#Nonzeros
linverse	Statistical problem	11999	11999	95977
1138_bus	power network problem	1138	1138	2596
Harvard500	directed graph	500	500	2636
conf5_0-4x4-10	theoretical/quantum chemistry problem	3072	3072	119808
conf5_0-4x4-14	theoretical/quantum chemistry problem	3072	3072	119808
conf5_0-4x4-18.mtx	theoretical/quantum chemistry problem	3072	3072	119808
conf5_0-4x4-22	theoretical/quantum chemistry problem	3072	3072	119808
cant	2D/3D problem	62451	62451	2034917

V. Results

The execution times obtained for the matrix storage formats CSR, COO, ELL, HYB and DIA for some sample matrices evaluated NVIDIA Quadro 8000 and NVIDIA GeForce GTX are given in the Table 3 and Table 4 respectively. The execution times for the same using graph are shown in Fig.2 and Fig.3 that follow. These execution times were obtained using CUDA toolkit 6.5/7.0. These are just some sample results obtained on a few matrices. Based on the execution times of the matrix storage formats the optimal storage format is predicted. As compared to the previous work [1] obtained results are better than the previous results.

Table 3. Kernel Execution Times for various matrix formats for a given matrix on NVIDIA Quadro 8000.

Matrix Name	Kernel Execution Time for Matrix Storage Format (ms)					Optimal Storage Format
	CSR	COO	HYB	ELL	DIA	
1138_bus	0.00044992	0.00068544	0.00066336	6.78E-05	0.000853824	CSR/ELL
Harvard500	0.000237536	0.000679648	0.00065408	0.000359872	0.00100627	CSR/ELL
conf5_0-4x4-10	0.0005736	0.000632	0.000762304	0.00011968	0.000332608	CSR/ELL
conf5_0-4x4-14	0.000543488	0.000645856	0.000629696	0.000119616	0.000334176	CSR/ELL
conf5_0-4x4-18.mtx	0.000543104	0.000633856	0.000616288	0.000119904	0.000333568	CSR/ELL
conf5_0-4x4-22	0.000555488	0.000635296	0.000624608	0.000118464	0.000332416	CSR/ELL
cant	0.00152851	0.00264931	0.00149952	0.00197549	0.00144506	HYB

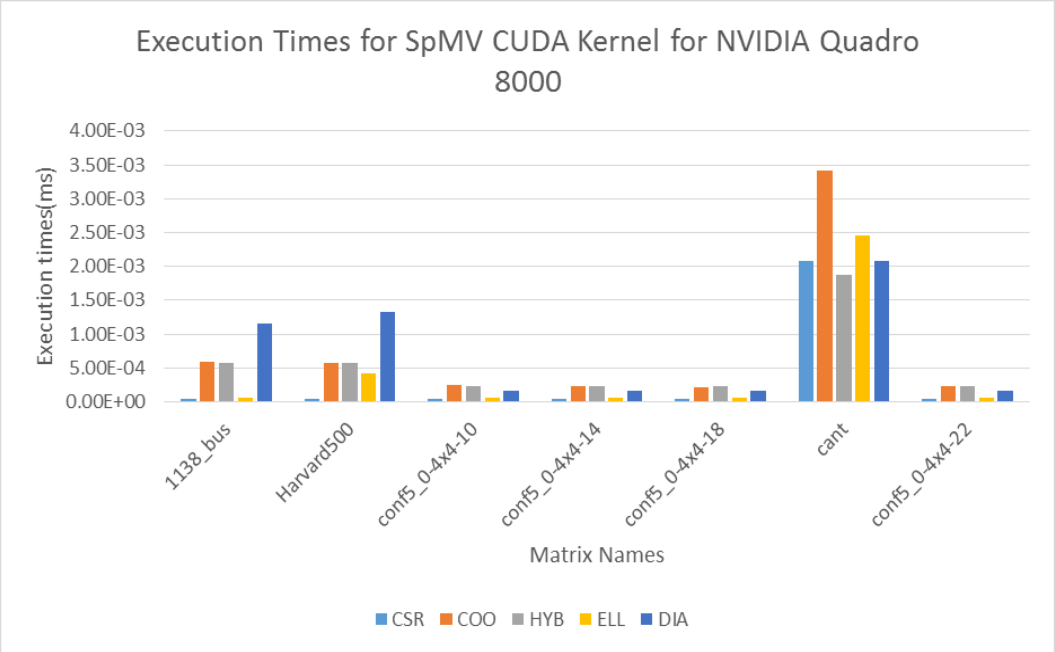

Fig. 2: Execution Times for Various Matrices

Table 3. Kernel Execution Times for various matrix formats for a given matrix on NVIDIA GeForce GTX 680.

Matrix Name	Kernel Execution Time for Matrix Storage Format (ms)					Optimal Storage Format
	CSR	COO	HYB	ELL	DIA	
1138_bus	4.144e-05	0.000588064	0.00058192	6.5664e-05	0.00115475	ELL
Harvard500	5.008e-05	0.000573472	0.000567648	0.000426752	0.00132701	CSR
conf5_0-4x4-10	4.208e-05	0.000242368	0.000230496	6.1856e-05	0.000161408	ELL
conf5_0-4x4-14	4.1984e-05	0.000231808	0.000231232	6.2048e-05	0.000160608	ELL
conf5_0-4x4-18.mtx	4.0096e-05	0.000206976	0.000231872	6.5856e-05	0.00016	ELL
conf5_0-4x4-22	3.98E-05	0.000239904	0.000228608	6.14E-05	0.000160576	ELL
cant	0.00207782	0.00340995	0.00188157	0.00245386	0.00208189	HYB

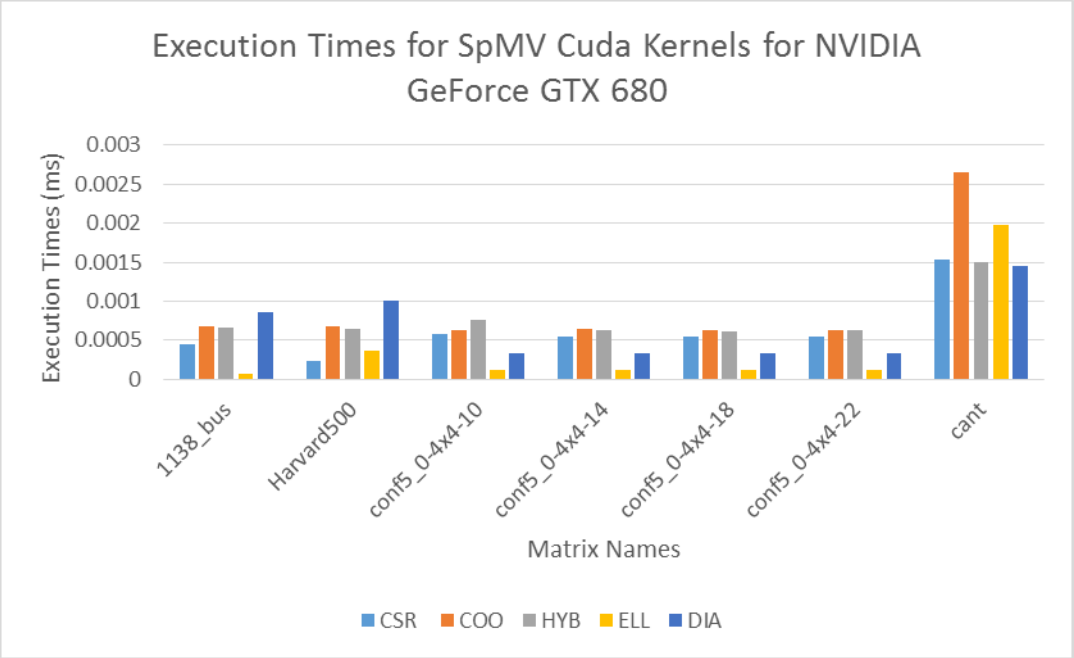

Fig. 3: Execution Times for Various Matrices (NVIDIA GeForce GTX 680)

VI. Conclusion

Sparse-matrix vector multiplication is tedious and when carried out repeatedly becomes more difficult. GPU makes this job much easier but require efficient storage strategies, performance modelling and optimization techniques. Hence, an integrated performance modelling and optimization analysis system for SpMV CUDA kernels is proposed. The system is evaluated using NVIDIA GeForce GTX 680 and NVIDIA Quadro 8000 using CUDA toolkit 6.5/7.0 and significant results have been obtained. The system is further extended to one more matrix storage format i.e. the DIA kernel. From the execution times of the DIA kernel one can loosely infer that the DIA kernel is not one of the most preferred formats when it comes to sparse matrix vector multiplication operation.

Acknowledgements

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'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's. R. H. Sapat College of Engg. Nashik.

I would also like to thank Prof. C.R. Barde my project guide for his invaluable guidance. We 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 modeling.

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. 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.

[4]. 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.

[5]. 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.

[6]. 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.

[7]. M.M. Baskaran and R. Bordawekar, "Optimizing Sparse Matrix- Vector Multiplication on GPUs," Research Report RC24704, IBM TJ Watson Research Center, Dec. 2008.
[8]. 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. 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.
[9]. 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.
[10]. 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.
[11]. 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.
[12]. 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.
[13]. 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.
[14]. 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.
[15]. 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.
[16]. 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.
[17]. 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.
[18]. 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.
[19]. 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.
[20]. 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.
[21]. 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. T.A. Davis and Y. Hu, "The University of Florida Sparse Matrix Collection," ACM Trans. Math. Software, vol. 38, no. 1, pp. 1:1-1