Accurate CUDA Performance Modeling for Sparse Matrix-Vector Multiplication

Ping Guo
Department of Computer Science
University of Wyoming, USA
pguo@uwyo.edu

Liqiang Wang
Department of Computer Science
University of Wyoming, USA
lwang7@uwyo.edu

***Abstract*—This paper presents an integrated analytical and profile-based CUDA performance modeling approach to accurately predict the kernel execution times of sparse matrix-vector multiplication for CSR, ELL, COO, and HYB SpMV CUDA kernels. Based on our experiments conducted on a collection of 8 widely-used testing matrices on NVIDIA Tesla C2050, the execution times predicted by our model match the measured execution times of NVIDIA's SpMV implementations very well. Specifically, for 29 out of 32 test cases, the performance differences are under or around 7%. For the rest 3 test cases, the differences are between 8% and 10%. For CSR, ELL, COO, and HYB SpMV kernels, the differences are 4.2%, 5.2%, 1.0%, and 5.7% on the average, respectively.**

***Keywords*—CUDA; GPU; Performance modeling; Sparse Matrix-Vector Multiplication**

I. Introduction

Sparse matrix-vector multiplication (SpMV) is an essential operation in solving linear systems and partial differential equations. For many scientific and engineering applications, the matrices can be very large and sparse. It is a challenging problem to accurately and effectively predict the execution time of SpMV CUDA kernel for a matrix with any scale and sparsity. The CUDA performance modeling approach proposed by this paper addresses this challenge.

In this paper, we propose an integrated analytical and profile-based performance modeling approach to accurately predict the CUDA kernel execution time of SpMV. Our modeling approach consists of two phases: instrumenting and modeling. In the phase of instrumenting, benchmark matrices are generated according to a GPU's architecture features, then SpMV computations with these benchmark matrices are conducted on the GPU to obtain the execution times. The properties and the execution times of benchmark matrices are recorded as the input in the phase of modeling. In the phase of modeling, we instantiate our parameterized performance models according to the experimental results of benchmark matrices. Finally, we utilize the instantiated models to estimate the CUDA kernel execution time of SpMV for any target sparse matrix. In this paper, we use SpMV CUDA kernels developed by NVIDIA [1] and NVIDIA Tesla C2050 for our performance modeling and experiments. However, the proposed approach is general and can be applied to any SpMV kernel and NVIDIA GPU architecture. For different SpMV kernels, only the execution times for benchmark matrices need to be retested; for different GPU architectures, the benchmark matrices also need to be regenerated. However, these changes only occur in the phase of instrumenting. Our parameterized performance models can be reused in the phase of modeling.

To predict the performance of SpMV for a target sparse matrix on a GPU, we partition the target matrix into strips. A strip is a maximum submatrix that can be handled by a GPU within one iteration. The size of matrix strip is determined by the physical limitations of a GPU and SpMV kernel granularity. The CUDA kernel execution time of SpMV for a target sparse matrix can be predicted by comparing the given target matrix and benchmark matrices.

Our innovative approach combines two major techniques used for performance modeling: profiling and analytics. In our approach, dividing modeling into two phases follows the profile-based technique; and it follows the analytical technique to generate benchmark matrices and performance models according to hardware properties. The integration of both analytical and profile-based modeling has the following advantages: (1) Compared to analytical models, our model is simple and easy to use. (2) Compared to traditional profile-based models, which are usually inaccurate for parallel architectures [2], our model can accurately and effectively capture the performance effects of GPUs.

We evaluated our performance modeling on 8 matrices using NVIDIA SpMV CUDA kernels [1]. For 29 out of 32 test cases, the performance differences are under or around 7%. For the rest 3 test cases, the performance differences are between 8% and 10%. Specifically, the differences are 4.2%, 5.2%, 1.0%, and 5.7% on the average for CSR, ELL, COO, and HYB SpMV kernels, respectively.

The rest of this paper is organized as follows: Section II surveys the related work about sparse matrix-vector multiplication (SpMV) and the recent performance modeling techniques. Section III presents the details of our CUDA performance modeling. Section IV evaluates the accuracy of our performance modeling by comparing the measured and predicted execution times, and the difference rates. Section V gives the conclusion and future work.

II. Related Work

Bolz *et al.* [3] proposed one of the first SpMV CUDA [4] kernel implementations. Bell and Garland [1] implemented SpMV CUDA kernels for some well-known sparse matrix formats, *i.e.*, DIA, CSR, ELL, COO, and HYB. Our modeling approach utilizes their implementations. The research work on SpMV optimization and tuning includes [5]–[11].

Choi *et al.* [12] designed a blocked ELLPACK format and proposed a CUDA performance model to predict matrix-dependent tuning parameters. Xu *et al.* [13] proposed the optimized SpMV based on ELL format and a SpMV CUDA performance model. Zhang and Owens [14] adopted a microbenchmark-based approach to develop a throughput model for three major components of GPU execution time: instruction pipeline, shared memory access, and global memory access. Their model focuses on identifying performance bottlenecks and guiding programmers for optimization; our model focuses on predicting the execution time, which is similar to [15]–[17]. Baghsorkhi *et al.* [15] presented a compiler-based GPU performance modeling approach with accurate prediction using program analysis and symbolic evaluation techniques. Hong and Kim [16] proposed a simple analytical GPU model to estimate the execution time of massively parallel programs. Their model estimates the number of parallel memory requests by taking into account the number of running threads and memory bandwidth. Kothapalli *et al.* [17] presented a performance model by combining several known models of parallel computation: BSP, PRAM, and QRQW. However, their proposed analytical models are based on the abstraction of GPU architecture. Unlike these analytical performance models, our model is based on both analytical and profile-based modeling techniques.

III. CUDA Performance Modeling for SpMV

A. The workflow of our modeling

The modeling workflows for CSR and ELL, COO SpMV CUDA kernels are shown in Figure 1 and Figure 2, respectively.

1) The phase of instrumenting:

- Compute the size of matrix strip (Section III-B).
- Generate the benchmark matrices (Section III-C).
- Test the execution times of the benchmark matrices (Section III-D).
- Compute the number of matrix strips and non-zero elements per row (it is not applicable to COO) for a target matrix (Section III-E).

2) The phase of modeling:

- Instantiate parameterized performance models according to the experimental results of benchmark matrices.
- Estimate the kernel execution time of SpMV for a target matrix using CUDA performance models (Section III-F).

The symbols used in our model are shown in Table I.

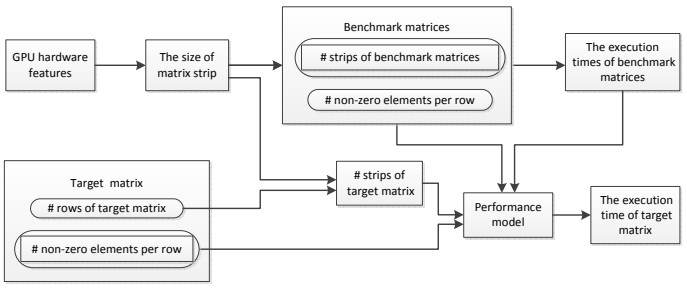

Figure 1. The modeling workflow for CSR and ELL SpMV CUDA kernels.

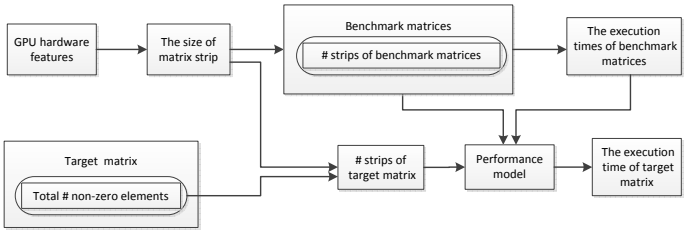

Figure 2. The modeling workflow for COO SpMV CUDA kernel.

TABLE I
SYMBOLS USED IN OUR PERFORMANCE MODELING

N_{SM}	The number of streaming multiprocessors (SMs), which is 14 in NVIDIA Tesla C2050.
R	The number of rows of a benchmark matrix.
S	The size of a matrix strip, which is the maximum number of rows (for CSR and ELL) or non-zero elements (for COO) that can be processed by a GPU within one iteration.
I	The number of strips of a benchmark or target sparse matrix.
$\mathcal{N}$	The set of natural numbers.
C	The number of columns of a benchmark matrix.
P_{NZ}	The number of non-zero elements per row of a benchmark or target sparse matrix.
G_M	The size (bytes) of GPU global memory.
$M_{R\times C}$	A benchmark matrix, where $R \times C$ indicates the dimension of the benchmark matrix.
V_C	A random vector, where C indicates the dimension of the random vector.
$\phi_{(M\times V)}$	The execution time of matrix-vector multiplication, where M indicates the benchmark matrix and V indicates the random vector.
α, β	Natural numbers, where $\alpha < \beta$ is required.
T	The execution time of a benchmark matrix.
N_R	The number of rows of a target sparse matrix.
N_{NZ}	The number of non-zero elements of a target sparse matrix.
D_S	A data set consisting of the number of non-zero elements in each row of a target matrix.

TABLE II
PHYSICAL LIMITATIONS OF GPUS WITH COMPUTE CAPABILITY 2.0

Threads / Warp	32
Warps / Multiprocessor	48
Threads / Multiprocessor	1536

TABLE III
SPMV KERNEL GRANULARITY

SpMV Kernel	Granularity
CSR	One warp per row
ELL	One thread per row
COO	One thread per non-zero element

B. The size of matrix strip (S)

- A strip is a maximum submatrix that can be handled by a GPU within one iteration. For a large matrix, it may need multiple iterations to handle the whole matrix. Thus, a matrix may contain multiple strips.
- The size of matrix strip is determined by the physical limitations of a GPU and SpMV kernel granularity, which are shown in Table II and III, respectively. The physical limitations of a GPU are determined by its compute capability. Our experiments are based on NVIDIA Tesla C2050, whose compute capability is 2.0.

1) CSR kernel:

$$S_{CSR} = N_{SM} \times Warps/Multiprocessor$$

2) ELL kernel:

$$S_{ELL} = N_{SM} \times Warps/Multiprocessor \times Threads/Warp$$

3) COO kernel:

$$S_{COO} = N_{SM} \times Threads/Multiprocessor$$

C. The benchmark matrices

1) The criteria for generating the benchmark matrices:

- The number of rows (R): $R = S \times I$
  - CSR: $S = S_{CSR}$, $I \in \mathcal{N}$
  - ELL: $S = S_{ELL}$, $I \in \mathcal{N}$
  - COO: $S = S_{COO}$, $I \in \mathcal{N}$
- The number of columns (C): $C > P_{NZ}$ is required
  - The value of C does not affect the performance since the sparse matrices are stored in compressed formats.
- The number of non-zero elements per row (P_{NZ}):
  - CSR: $P_{NZ} \in [1, \frac{G_M - sizeof(int) \times (R+1)}{(sizeof(float)+sizeof(int)) \times R})$
  - ELL: $P_{NZ} \in [1, \frac{G_M}{(sizeof(float)+sizeof(int)) \times R})$
  - COO: $P_{NZ} \in [1, \frac{G_M}{(sizeof(float)+2 \times sizeof(int)) \times R})$

  We assume that the non-zero elements are in single-precision ($float$) and each row has the same number of non-zero elements. The maximum P_{NZ} is derived according to the maximum non-zero elements that can be stored in the GPU global memory in the corresponding sparse matrix format. The values used in our experiments are introduced in Section III-C2.
- The value of the matrix entry: random value

2) The experimental setup: To obtain accurate performance models, we generate a series of benchmark matrices. A benchmark matrix is determined by R and P_{NZ}. Since $R = S \times I$, where S is fixed, we just enumerate values of I and P_{NZ} according to the above criteria to obtain combinations. Each combination indicates a benchmark matrix.

- The number of strips (I):
  - CSR and ELL: Let $I = 1, 2, 3...10$
    - In our experiment, the largest benchmark matrix contains 10 strips, which is accurate enough to measure the performance.
  - COO: Let $I = 1$
    - Since each non-zero element is handled by one thread, we just need to increase the number of non-zero elements per row for different benchmark matrices to duplicate strips instead of increasing the values of the number of strips.
- The number of non-zero elements per row (P_{NZ}):
  - CSR and ELL: Let $P_{NZ} = 4, 16...1024, 2048...$
  - COO: Let $P_{NZ} = 10, 20, 30...100$

D. The execution times of benchmark matrices (T)

- For each benchmark matrix, a random vector is generated for measuring the execution time of SpMV kernel.
- We remove the effect of long initialization delay and average the execution time of a benchmark matrix as follows:

$$T = \frac{\sum_{j=1}^{\beta} \phi_{((M_{R \times C}) \times V_C)} - \sum_{j=1}^{\alpha} \phi_{((M_{R \times C}) \times V_C)}}{\beta - \alpha}$$

E. The target matrix

1) The number of strips (I):

- Given a target matrix with N_R rows and N_{NZ} non-zero elements, the number of strips can be computed as follows:

$$I_{CSR} = \lceil \frac{N_R}{S_{CSR}} \rceil$$

$$I_{ELL} = \lceil \frac{N_R}{S_{ELL}} \rceil$$

$$I_{COO} = \lceil \frac{N_{NZ}}{S_{COO}} \rceil$$

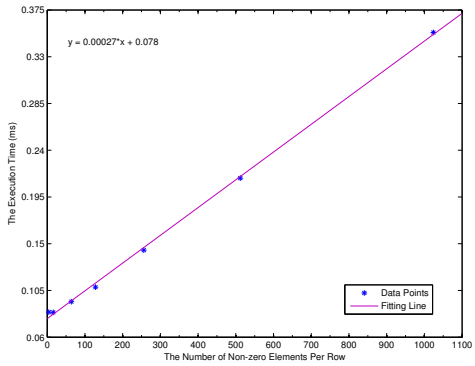

Figure 3. The number of non-zero elements per row vs the execution time when $P_{NZ} <= threshold$ and the number of strips is fixed (CSR).

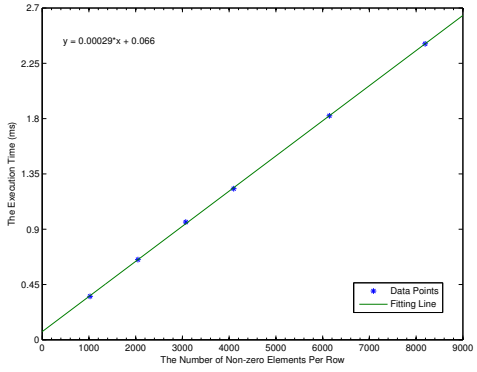

Figure 4. The number of non-zero elements per row vs the execution time when $P_{NZ} >= threshold$ and the number of strips is fixed (CSR).

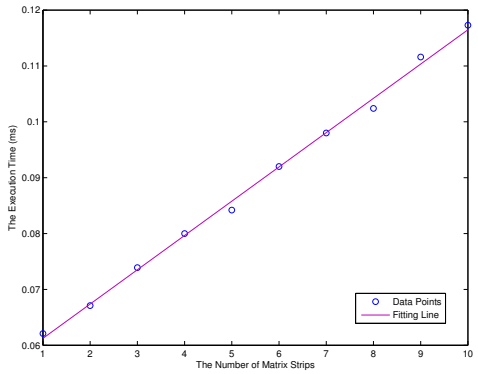

Figure 5. The number of strips vs the execution time when $P_{NZ} <= threshold$ and P_{NZ} is fixed (CSR).

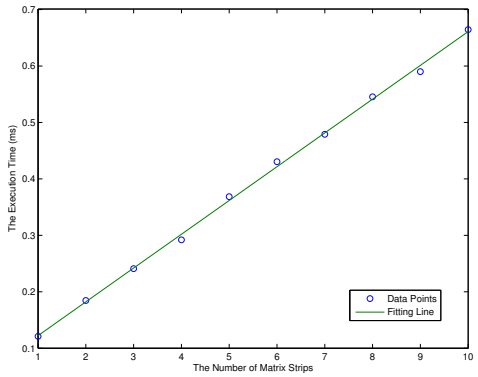

Figure 6. The number of strips vs the execution time when $P_{NZ} >= threshold$ and P_{NZ} is fixed (CSR).

2) The number of non-zero elements per row (P_{NZ}):

- CSR: P_{NZ} is set to be mode (statistics) of a data set D_S.
- ELL: P_{NZ} is set to be the Max. value of a data set D_S.

F. Performance Modeling and Estimating

There exists relationships between the number of strips, the number of non-zero elements per row, and the execution times of the benchmark matrices. Hence, we can estimate the CUDA kernel execution time of SpMV for a target matrix according to these relationships.

1) CSR kernel: Our method contains the following steps:

Step 1: Establish the following relationships:

- Relationship-1 ($T = A * x + B$): For a set of benchmark matrices with the same number of strips (it can be any arbitrary value within the range defined in Section III-C), we establish the relationship between the number of non-zero elements per row (x) and the execution time of the benchmark matrices (T).
- Relationship-2 ($T' = C * y + D$): For a set of benchmark matrices with the same number of non-zero elements per row, we establish the relationship between the number of strips (y) and the execution time of the benchmark matrices (T').

By studying the physical limitations of NVIDIA Tesla C2050, we were surprised to discover that its number of max threads per block, *i.e.*1024, is exactly a threshold: when the number of non-zero elements per row is smaller or larger than it, there are two different linear relationships. The two linear equations in Relationship-1 are shown in Figure 3 and Figure 4. The two linear equations in Relationship-2 are shown in Figure 5 and Figure 6.

Step 2: Estimate the execution time of a target matrix:

- According to the number of non-zero elements per row of the target matrix (denoted by x_0), find $t_0 = A * x_0 + B$ from Relationship-1, and the execution time t_1 of any previously tested benchmark matrix M. Here, we assume that Z is the number of non-zero elements per row of matrix M.
- According to the number of strips of the target matrix (denoted by y_0), find $t_2 = C * y_0 + D$ from a corresponding linear equation in Relationship-2. Note that, the number of non-zero elements per row of matrix M (denoted by Z) is set to be the number of non-zero elements per row in Relationship-2.
- Estimate the execution time of the target matrix by $(t_0/t_1) * t_2$.

2) ELL kernel: Our method works as follows:

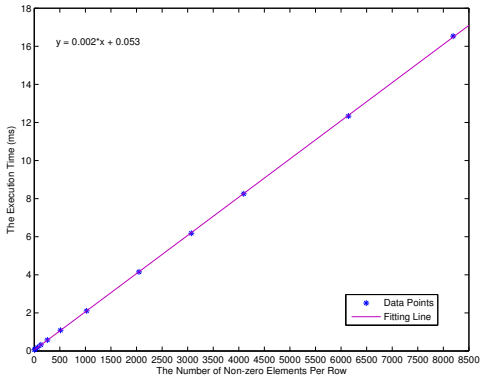

Figure 7. The number of non-zero elements per row vs the execution time when the number of strips is fixed (ELL).

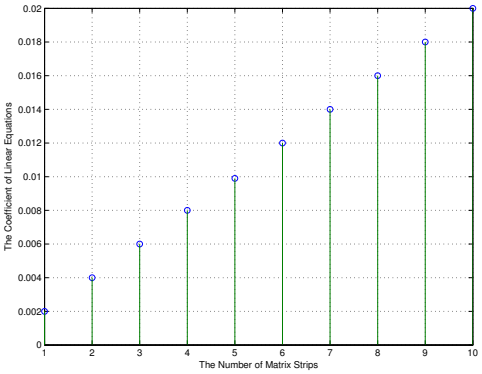

Figure 8. The number of strips vs the coefficient of linear equations (ELL).

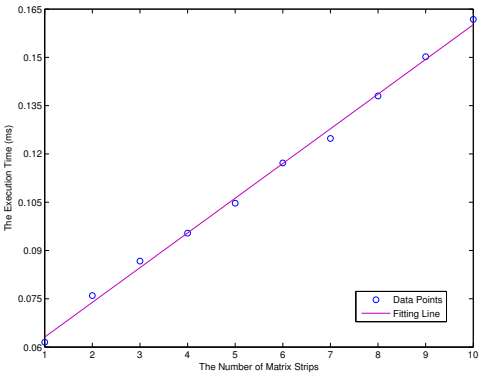

Figure 9. The number of strips vs the execution time when P_{NZ} is fixed (ELL).

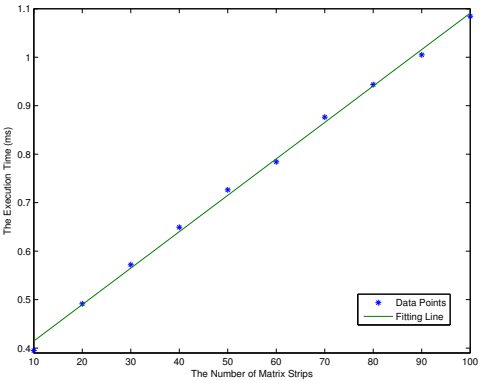

Figure 10. The number of strips vs the execution time (COO).

Step 1: Establish the following relationships:

- Relationship-1 ($T = f(y_1) * x + g(y_1)$): For a set of benchmark matrices with the same number of strips (it can be any arbitrary value within the range defined in Section III-C and denoted by y_1), we establish the relationship between the number of non-zero elements per row (x) and the execution time of the benchmark matrices (T), as shown in Figure 7.
- Relationship-2 ($f(y)$): For sets of benchmark matrices with different number of strips, we establish the relationship between the number of strips of the benchmark matrices (y) and the corresponding coefficient of the linear equations (f) in Relationship-1, as shown in Figure 8.
- Relationship-3 ($e(y) = f(y) * x_1 + g(y)$): For a set of benchmark matrices with the same number of non-zero elements per row (it can be any arbitrary value within the range defined in Section III-C and denoted by x_1), we establish the relationship between the number of strips (y) and the execution time of the benchmark matrices (e), as shown in Figure 9. Thus, $g(y) = e(y) - f(y) * x_1$.

Step 2: Estimate the execution time of a target matrix:

- Given a target matrix, in order to estimate its execution time, we need to obtain the coefficient $f(Y)$ and the intercept $g(Y)$ of the linear equation, where Y is the number of strips of the target matrix. This can be done as follows:
  - According to the number of strips (Y) of the target matrix, obtain the coefficient of the linear equation from Relationship-2, *i.e.*, $f(Y)$.
  - To obtain the intercept of the linear equation of the target matrix (*i.e.*, $g(Y)$), we find the execution time (e) from Relationship-3 according to Y. Thus, $g(Y) = e(Y) - f(Y) * Y$.
- Estimate the execution time of the target matrix by $f(Y) * X + g(Y)$, where X and Y are the number of non-zero elements per row and the number of strips of the target matrix, respectively.

3) COO kernel: Our method contains the following steps:

Step 1: Establish the following relationships:

- Relationship-1: We establish the relationship between the number of strips and the execution time of the benchmark matrices, as shown in Figure 10.

Step 2: Estimate the execution time of a target matrix:

- Count the total number of non-zero elements of the target matrix, then calculate the number of strips according to Section III-E1.

- Estimate the execution time of the target matrix using Relationship-1.

4) HYB kernel: Our method works as follows:

Step 1: Establish the following relationships:

- Since HYB kernel is the combination of ELL and COO kernels, we can reuse the relationships in Sections III-F2 and III-F3.

Step 2: Estimate the execution time of a target matrix:

- Compute HYB threshold [1] to divide the target matrix into two parts: ELL and COO.
- Calculate the number of strips of ELL and COO parts of the target matrix, respectively.
- Use HYB threshold as the number of non-zero elements per row of ELL part of the target matrix.
- Count the total number of non-zero elements of COO part of the target matrix.
- Estimate the execution times of ELL and COO parts of target matrix, respectively.
- Sum above two parts of execution times.

IV. Experimental Evaluation

Our experiments are performed on NVIDIA Tesla C2050 with 3GB global memory. We evaluated our performance models on the 14 widely-used testing matrices [18]. However, NVIDIA's SpMV implementations [1] cannot execute ELL SpMV kernel on 6 sparse matrices on NVIDIA Tesla C2050 ("Wind Tunnel", "Economics", "FEM/Accelerator", "Circuit", "Webbase", and "LP") because of the limitation of "$num\_cols\_per\_row$". Hence, we conducted experiments on the rest 8 unstructured sparse matrices [18], as shown in Table IV.

To show the prediction accuracy, we compare our four performance prediction models with NVIDIA's SpMV implementations by two aspects: the execution times and the performance difference rates.

Figure 11, 12, 13, and 14 show the comparisons between the measured execution times of NVIDIA's CSR, ELL, COO, and HYB SpMV implementations and the execution times predicted by our CSR, ELL, COO, and HYB performance prediction models, respectively.

Figure 15 shows the performance difference rates between the measured and predicted execution times. The experiments evaluate CSR, ELL, COO, and HYB SpMV CUDA kernels on a collection of 8 unstructured sparse matrices. Hence, there are 32 test cases in total. The execution times of SpMV CUDA kernels predicted by our model match the measured execution times of NVIDIA's SpMV implementations very well. Specifically, for 29 out of 32 test cases, the performance differences are under or around 7%. For the rest 3 test cases, the differences are between 8% and 10%. For CSR, ELL, COO, and HYB SpMV kernels, the differences are 4.2%, 5.2%, 1.0%, and 5.7% on the average, respectively.

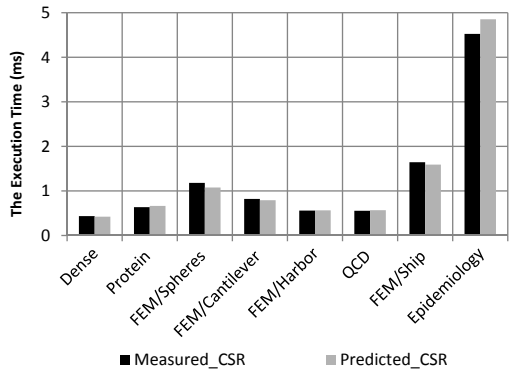

Figure 11. Performance modeling evaluation on CSR kernel.

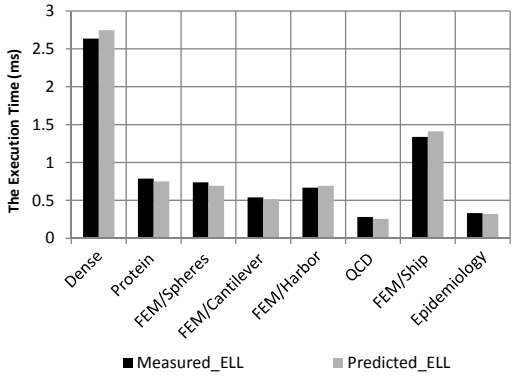

Figure 12. Performance modeling evaluation on ELL kernel.

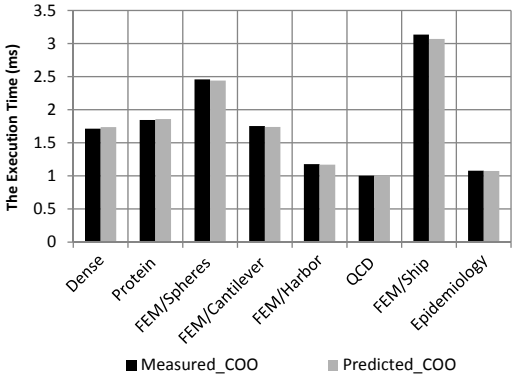

Figure 13. Performance modeling evaluation on COO kernel.

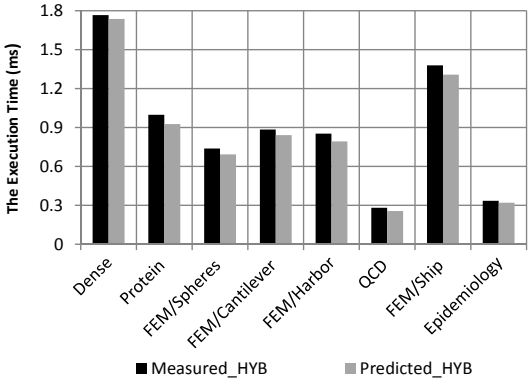

Figure 14. Performance modeling evaluation on HYB kernel.

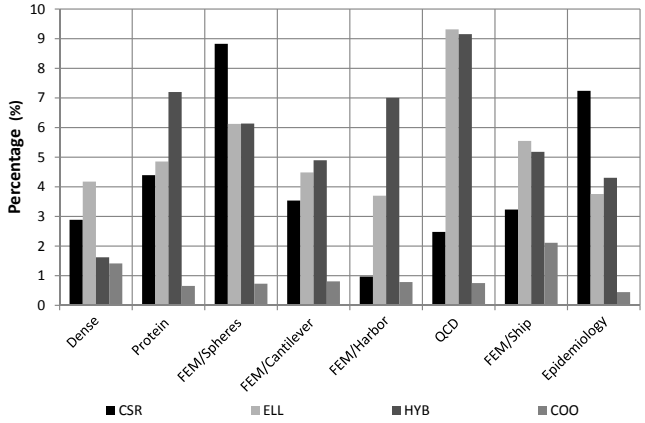

Figure 15. Performance modeling evaluation on CSR, ELL, COO, and HYB kernels.

TABLE IV
SPARSE MATRICES USED IN OUR EXPERIMENTAL EVALUATION

Matrix	Dimensions	Nonzeros / Row	Nonzeros
Dense	2K*2K	2000.0	4.0 M
Protein	36K*36K	119.3	4.3 M
FEM/Spheres	83K*83K	72.1	6.0 M
FEM/Cantilever	62K*62K	64.1	4.0 M
FEM/Harbor	47K*47K	50.6	2.37 M
QCD	49K*49K	39.0	1.90 M
FEM/Ship	141K*141K	55.4	7.81 M
Epidemiology	526K*526K	3.9	2.1 M

V. CONCLUSION AND FUTURE WORK

In this paper, we proposed four CUDA kernel performance prediction models: CSR, ELL, COO, and HYB SpMV performance prediction models, which are based on CSR, ELL, COO, and HYB SpMV CUDA kernels, respectively, to accurately predict the kernel execution time of sparse matrix-vector multiplication for a target sparse matrix. Our proposed models utilize an integrated analytical and profile-based CUDA performance modeling approach. Compared to analytical models, our model is simple and easy to use; compared to traditional profile-based models, our model can effectively capture the performance effects of GPUs. The experimental results show that the execution times of SpMV kernels predicted by our models match the measured execution times of NVIDIA's SpMV implementations very well.

In the future, we will extend our CUDA SpMV performance modeling to predict the execution times of other SpMV CUDA kernels (*e.g.* DIA). In addition, we will also propose and design a performance modeling to predict the execution time of dense matrix-vector multiplication.

ACKNOWLEDGMENT

The work was supported in part by NSF under Grants 0941735, CAREER-1054834, and by the Graduate Assistantship of the School of Energy Resources at the University of Wyoming.

REFERENCES

[1] N. Bell and M. Garland, "Implementing sparse matrix-vector multiplication on throughput-oriented processors," in *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, New York, NY, USA, 2009, pp. 1–11.

[2] A. Resios and V. Holdermans, "GPU performance prediction using parametrized models," Master's thesis, Utrecht University, 2011.

[3] J. Bolz, I. Farmer, E. Grinspun, and P. Schroder, "Sparse matrix solvers on the GPU: conjugate gradients and multigrid," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 917–924, 2003.

[4] *NVIDIA CUDA C Programming Guide, Version 4.0*, May 2011.

[5] J. Kurzak, W. Alvaro, and J. Dongarra, "Optimizing matrix multiplication for a short-vector simd architecture-cell processor," *Parallel Comput.*, vol. 35, no. 3, pp. 138–150, 2009.

[6] 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," *Proceeding of IEEE*, vol. 93, no. 2, pp. 293–312, 2005.

[7] E.-J. Im, K. Yelick, and R. Vuduc, "Sparsity: Optimization framework for sparse matrix kernels," *Int. J. High Perform. Comput. Appl.*, vol. 18, no. 1, pp. 135–158, 2004.

[8] M. M. Baskaran and R. Bordawekar, "Optimizing sparse matrix-vector multiplication on GPUs using compile-time and run-time strategies," Research Report RC24704, IBM TJ Watson Research Center, Tech. Rep., december 2008.

[9] P. Guo and L. Wang, "Auto-tuning CUDA parameters for sparse matrix-vector multiplication on GPUs," in *Proceedings of the 2010 International Conference on Computational and Information Sciences*, ser. ICCIS '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1154–1157.

[10] A. Nukada and S. Matsuoka, "Auto-tuning 3-D FFT library for CUDA GPUS," in *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, New York, NY, USA, 2009, pp. 1–10.

[11] F. Vazquez, G. Ortega, J. J. Fernandez, and E. M. Garzon, "Improving the performance of the sparse matrix vector product with gpus," in *Proceedings of the 2010 10th IEEE International Conference on Computer and Information Technology*, ser. CIT '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1146–1151.

[12] J. W. Choi, A. Singh, and R. W. Vuduc, "Model-driven autotuning of sparse matrix-vector multiply on GPUs," in *PPoPP '10: Proceedings of the 15th ACM SIGPLAN symposium on Principles and practice of parallel programming*, New York, NY, USA, 2010, pp. 115–126.

[13] S. Xu, W. Xue, and H. Lin, "Performance modeling and optimization of sparse matrix-vector multiplication on NVIDIA CUDA platform," *The Journal of Supercomputing*, pp. 1–12, 2011.

[14] Y. Zhang and J. Owens, "A quantitative performance analysis model for GPU architectures," in *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*, feb. 2011, pp. 382 –393.

[15] S. S. Baghsorkhi, M. Delahaye, S. J. Patel, W. D. Gropp, and W.-m. W. Hwu, "An adaptive performance modeling tool for GPU architectures," in *Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '10. New York, NY, USA: ACM, 2010, pp. 105–114.

[16] S. Hong and H. Kim, "An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness," in *Proceedings of the 36th annual international symposium on Computer architecture*, ser. ISCA '09. New York, NY, USA: ACM, 2009, pp. 152–163.

[17] K. Kothapalli, R. Mukherjee, M. Rehman, S. Patidar, P. Narayanan, and K. Srinathan, "A performance prediction model for the CUDA GPGPU platform," in *High Performance Computing (HiPC), 2009 International Conference on*, dec. 2009, pp. 463 –472.

[18] S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, and J. Demmel, "Optimization of sparse matrix-vector multiplication on emerging multicore platforms," in *Proc. 2007 ACM/IEEE Conference on Supercomputing*, 2007.