CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2013; **00**:1–14
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe

Accurate Cross-Architecture Performance Modeling for Sparse Matrix-Vector Multiplication (SpMV) on GPUs

Ping Guo * and Liqiang Wang

Department of Computer Science, University of Wyoming, Laramie, WY 82071, USA

SUMMARY

This paper presents an integrated analytical and profile-based cross-architecture performance modeling tool to specifically provide inter-architecture performance prediction for Sparse Matrix-Vector Multiplication (SpMV) on NVIDIA GPU architectures. To design and construct the tool, we investigate the inter-architecture relative performance for multiple SpMV kernels. For a sparse matrix, based on its SpMV kernel performance measured on a reference architecture, our cross-architecture performance modeling tool can accurately predict its SpMV kernel performance on a target architecture. The prediction results can effectively assist researchers in making choice of an appropriate architecture that best fits their needs from a wide range of available computing architectures. We evaluate our tool with 14 widely-used sparse matrices on four GPU architectures: NVIDIA Tesla C2050, Tesla M2090, Tesla K20m, and GeForce GTX 295. In our experiments, Tesla C2050 works as the reference architecture, the other three are used as the target architectures. For Tesla M2090, the average performance differences between the predicted and measured SpMV kernel execution times for CSR, ELL, COO, and HYB SpMV kernels are 3.1%, 5.1%, 1.6%, and 5.6%, respectively. For Tesla K20m, they are 6.9%, 5.9%, 4.0%, and 6.6% on the average, respectively. For GeForce GTX 295, they are 5.9%, 5.8%, 3.8%, and 5.9% on the average, respectively.

Received . . .

KEY WORDS: Performance modeling; Sparse Matrix-Vector Multiplication; GPU; CUDA

1. INTRODUCTION

Sparse matrix-vector multiplication (SpMV) is an essential operation in solving linear systems and partial differential equations. In our previous research work [1], we proposed an integrated analytical and profile-based performance modeling tool to accurately predict SpMV kernel execution time on a Graphics Processing Unit (GPU) architecture. However, for users who have options to use different GPU architectures, if they intend to achieve SpMV kernel performance on multiple different GPU architectures in a low-cost way, or further, if they intend to choose an appropriate one from a wide range of available GPU architectures, they are facing a challenging issue to do large amount of profiling work by following our traditional modeling approach [1]. This paper addresses this problem by proposing an enhanced analytical and profile-based performance modeling tool to predict cross-architecture SpMV kernel performance with much less user's effort. According to the known performance measured on a reference GPU architecture, the proposed approach can accurately predict SpMV kernel performance on a target architecture based on the performance relationships between different GPU architectures. Such a tool can effectively assist researchers

*Correspondence to: Ping Guo, Department of Computer Science, University of Wyoming, Laramie, WY 82071, USA.
E-mail: pguo@uwyo.edu

Contract/grant sponsor: National Science Foundation; contract/grant number: 1118059 and 0941735.

in making choice of an appropriate architecture that best fits their needs from different available computing architectures. However, this prediction goal cannot be reached via a naive approach trying to multiply the known reference performance by theoretical peak performance ratio of these two architectures because the real performance ratios are significantly deviant from the peak performance ratios for different applications performed with different SpMV kernels.

Our performance modeling approach combines both profile-based technique and analytical technique. It follows the profile-based technique to divide the modeling into two phases: instrumenting and modeling. In the phase of instrumenting, some benchmark matrices are generated and their corresponding SpMV computations are conducted. The properties of the benchmark matrices and their SpMV kernel execution times are recorded as the input in the phase of modeling. In the phase of modeling, SpMV kernel execution time for a target sparse matrix is estimated. In our approach, the analytical technique is adopted for instructing the generation of benchmark matrices and constructing parameterized performance models.

We evaluate our tool with 14 matrices on four GPU architectures: NVIDIA Tesla C2050, Tesla M2090, Tesla K20m, and GeForce GTX 295. The SpMV kernels we used in our tool, which are Compressed Sparse Row (CSR), ELLPACK/ITPACK (ELL), Coordinate (COO), and Hybrid (HYB), are based on Bell and Garland's [2] work. In our experiments, NVIDIA Tesla C2050 is used as the reference architecture, the other three are used as the target architectures. For NVIDIA Tesla M2090, the average performance differences between the predicted and measured SpMV kernel execution times for CSR, ELL, COO, and HYB SpMV kernels are 3.1%, 5.1%, 1.6%, and 5.6%, respectively. For Tesla K20m, they are 6.9%, 5.9%, 4.0%, and 6.6% on the average, respectively. For GeForce GTX 295, they are 5.9%, 5.8%, 3.8%, and 5.9% on the average, respectively.

The rest of this paper is organized as follows: Section 2 surveys the related work about SpMV and the recent modeling techniques. Section 3 briefly introduces the major contents of our baseline tool. Section 4 presents the details of our cross-architecture SpMV performance modeling tool. Section 5 evaluates the accuracy of our performance modeling tool and gives the analysis of our experimental results. Section 6 gives the conclusions.

2. RELATED WORK

This paper extends our previous work [1] to support cross-architecture SpMV performance prediction. Specifically, our previous modeling approach proposed in [1] can accurately predict SpMV performance on a GPU architecture. However, it needs much effort when applying it to predict performance on multiple different GPU architectures due to a series of benchmark matrices required to be generated on each GPU architecture to instantiate the models. The modeling approach proposed in this paper addresses the challenging issue. It can significantly reduce the amount of profiling work while keeping the accuracy of the performance modeling tool.

Bolz *et al.* [3] first implemented SpMV computing on GPUs. Bell and Garland [2] implemented Compute Unified Device Architecture (CUDA)-based SpMV kernels for some well-known sparse matrix formats, *i.e.*, CSR, ELL, COO, and HYB. Our modeling approach utilizes their kernels.

The research work on SpMV optimization and tuning includes [4, 5, 6, 7, 8, 9, 10, 11]. Vazquez *et al.* [4] proposed a new format, called ELLR-T, which is a variant of ELL, to achieve high performance on GPUs. Monakov *et al.* [5] proposed a sliced ELL (SELL) format and used auto-tuning to find the optimal configuration, *e.g.*, the number of rows in a slice, to improve SpMV performance on GPUs. Dang and Schmidt [6] presented a new format called sliced COO (SCOO) and an efficient CUDA implementation to optimize SpMV on the GPU. Sun *et al.* [7] proposed a new storage format for diagonal sparse matrices, defined as Compressed Row Segment with Diagonal-pattern (CRSD), to speed up SpMV performance on GPUs. We [8] proposed an auto-tuning framework that can automatically compute and select the parameters of SpMV kernels to obtain the optimal performance on specific GPUs. Kubota and Takahashi [9] proposed an SpMV auto-selection algorithm on GPUs to automatically select the optimal storage scheme for the given sparse matrices. Baskaran and Bordawekar [10] proposed optimizations including: synchronization-free parallelism, optimized thread mapping, optimized off-chip memory access, and data reuse, to

DOI: 10.1002/cpe

speed up SpMV kernel. Pichel *et al.* [11] explored the SpMV performance optimization on GPUs using reordering techniques.

Some performance models focusing on optimizing performance have been proposed. Sim *et al.* [12] presented a performance analysis framework to precisely predict the performance and predict the potential performance benefits. Choi *et al.* [13] designed a blocked ELLPACK (BELLPACK) format and proposed a performance model to predict matrix-dependent tuning parameters. Karakasis *et al.* [14] presented a performance model that can accurately select the most suitable blocking sparse matrix storage format and its proper configuration. Zhang and Owens [15] 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. We [16] presented a modeling and auto-tuning integrated framework to speed up SpMV on GPUs. Xu *et al.* [17] proposed the optimized SpMV based on ELL format and a performance model. Resios [18] proposed a GPU parametrized analytical model to estimate execution time and identify potential bottlenecks in their programs. Besides prediction, their tool can also suggest values for parameters that influence performance. Jia *et al.* [19] presented GPURoofline, an empirical model for guiding optimizations on GPUs. Cui *et al.* [20] presented a performance model to optimize control flow divergence, further, to improve application performance.

Also, there are some performance models focusing on predicting execution times. Dinkins [21] proposed a model for predicting SpMV performance using memory bandwidth requirements and data locality. Baghsorkhi *et al.* presented a string-based [22] analytical model and a work flow graph (WFG)-based [23] analytical model to predict the performance of GPU applications. Kothapalli *et al.* [24] presented a performance model by combining several known models of parallel computation: BSP, PRAM, and QRQW. Yang *et al.* [25] presented a cross-platform performance prediction approach for parallel application based on inter-platform relative performance and partial execution. Marin and Mellor-Crummey [26] proposed a modeling tool for semi-automatically measuring and modeling static and dynamic characteristics of applications in an architecture-neutral fashion. Hong and Kim [27] proposed a simple analytical GPU model to estimate the execution time of massively parallel programs. Their model estimated the number of parallel memory requests by taking into account the number of running threads and memory bandwidth. We [28] proposed a performance modeling and optimization analysis tool to predict and optimize SpMV performance on GPUs. Schaa *et al.* [29] designed a methodology to accurately predict the execution time for GPU applications while varying the number and configuration of the GPUs, and the size of the input data set. Liu *et al.* [30] identified several key factors that determine the performance of GPU-based applications and proposed performance models for them.

3. BASELINE SPMV PERFORMANCE MODELING TOOL

Our cross-architecture performance modeling tool is based on our previous modeling tool [1], which is referred as the baseline tool in this paper. This section briefly introduces some important contents of the baseline tool, which are related to our cross-architecture modeling tool. More detailed descriptions can be found in [1].

(1) The size of matrix strip: A strip is a maximum submatrix that can be handled by a GPU with a full load of thread blocks within one iteration [13]. 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, denoted by S, is represented as follows:

$$S = \begin{cases} S_{CSR}, & \text{for CSR kernel} \\ S_{ELL}, & \text{for ELL kernel} \\ S_{COO}, & \text{for COO kernel} \end{cases}$$

S is determined by the physical limitations of a GPU and the specific SpMV kernels [1]. For CSR and ELL SpMV kernels, S_{CSR} and S_{ELL} represent the number of rows in a matrix strip; For COO kernel, S_{COO} represents the total number of non-zero elements in a matrix strip. They are computed as follows:

DOI: 10.1002/cpe

- $S_{CSR} = N_{SM} \times Warps/Multiprocessor$
- $S_{ELL} = S_{COO} = N_{SM} \times Threads/Multiprocessor$
  where,
  - $\diamond$ N_{SM} represents the number of streaming multiprocessors (SMs).
  - $\diamond$ $Warps/Multiprocessor$ represents the number of warps per SM.
  - $\diamond$ $Threads/Multiprocessor$ represents the number of threads per SM.

(2) The benchmark matrices: To obtain accurate performance models, we generate a series of benchmark matrices according to the following criteria:

- The number of rows (R): $R = S \times I$ and $I \in \mathcal{N}$
  where,
  - $\diamond$ I represents the number of matrix strips.
  - $\diamond$ $\mathcal{N}$ represents the set of natural numbers.
- The number of non-zero elements per row (P_{NZ_BM}):
  - CSR: $P_{NZ\_BM} \in [1, \frac{G_M - sizeof(int) \times (R+1)}{(sizeof(float) + sizeof(int)) \times R})$
  - ELL: $P_{NZ\_BM} \in [1, \frac{G_M}{(sizeof(float) + sizeof(int)) \times R})$
  - COO: $P_{NZ\_BM} \in [1, \frac{G_M}{(sizeof(float) + 2 \times sizeof(int)) \times R})$

  where,
  - $\diamond$ G_M represents the size (bytes) of GPU global memory.
  - $\diamond$ The non-zero elements in the above equations are in single-precision (float). For double-precision, the equations are similar, just replacing $float$ with $double$.

  Assume that each row of a benchmark matrix has the same number of non-zero elements. The maximum P_{NZ_BM} is derived according to the maximum non-zero elements that can be stored in the GPU global memory in the corresponding sparse matrix format [2]. Specifically, for CSR, ELL, and COO, the maximum P_{NZ_BM} can be derived from the following equations, respectively:
  - CSR: $sizeof(ptr) + sizeof(indices) + sizeof(data) < G_M$
  - ELL: $sizeof(indices) + sizeof(data) < G_M$
  - COO: $sizeof(row) + sizeof(indices) + sizeof(data) < G_M$

  where,
  - $\diamond$ ptr represents an array storing row pointers.
  - $\diamond$ $indices$ represents an array storing column indices of non-zero elements.
  - $\diamond$ $data$ represents an array storing values of non-zero elements.
  - $\diamond$ row represents an array storing row indices of non-zero elements.
  - $\diamond$ $sizeof(ptr) = sizeof(int) \times (R+1)$
  - $\diamond$ $sizeof(indices) = sizeof(int) \times R \times P_{NZ\_BM}$
  - $\diamond$ $sizeof(data) = sizeof(float) \times R \times P_{NZ\_BM}$
  - $\diamond$ $sizeof(row) = sizeof(int) \times R \times P_{NZ\_BM}$
- The number of columns (C): $C > P_{NZ\_BM}$ is required. Since the sparse matrices are stored in compressed formats, the value of C does not affect the performance.
- The value of each non-zero element: random value.

(3) The target matrix: Let D be a set consisting of the number of non-zero elements in each row of a target matrix. The P_{NZ} value of the target matrix can be estimated as follows:

- CSR: P_{NZ_CSR} is set to be the mode (in statistics) of a set D.
- ELL: P_{NZ_ELL} is set to be the maximum value of a set D.

(4) The linear relationships: The baseline tool estimates the SpMV kernel execution time of a target matrix according to the relationships established between the number of strips, the number of non-zero elements per row, and the execution times of the benchmark matrices.

DOI: 10.1002/cpe

4. CROSS-ARCHITECTURE SPMV PERFORMANCE PREDICTION

We refer to a GPU architecture, on which we have more knowledge about SpMV performance, as the reference architecture. Correspondingly, we refer to a GPU architecture, on which we intend to predict SpMV performance, as the target architecture.

Assume that, T_{Ref}, the SpMV kernel execution time of a sparse matrix A measured on the reference architecture, is known. Our goal is to predict T_{Tar}, the SpMV kernel execution time of matrix A on the target architecture. For different SpMV kernels, *i.e.*, CSR, ELL, COO, and HYB kernels, T_{Ref} and T_{Tar} represent the execution times of the specific SpMV kernels.

The absolute performance on the target architecture is predicted by using the following equation:

$$T_{Tar} = T_{Init\_Tar} + P_{Tar\_Ref} * (T_{Ref} - T_{Init\_Ref})$$

where P_{Tar_Ref}, the relative performance between the target and the reference architectures, is represented as follows:

$$P_{Tar\_Ref} = \begin{cases} P_{CSR}, & \text{for CSR kernel} \\ P_{ELL}, & \text{for ELL kernel} \\ P_{COO}, & \text{for COO kernel} \\ P_{HYB}, & \text{for HYB kernel} \end{cases}$$

T_{Init_Ref} and T_{Init_Tar}, the initialization time of the reference and target architectures, respectively, are represented as follows:

$$T_{Init\_Ref} = \begin{cases} T_{Init\_Ref\_CSR}, & \text{for CSR kernel} \\ T_{Init\_Ref\_ELL}, & \text{for ELL kernel} \\ T_{Init\_Ref\_COO}, & \text{for COO kernel} \\ T_{Init\_Ref\_HYB}, & \text{for HYB kernel} \end{cases}$$

$$T_{Init\_Tar} = \begin{cases} T_{Init\_Tar\_CSR}, & \text{for CSR kernel} \\ T_{Init\_Tar\_ELL}, & \text{for ELL kernel} \\ T_{Init\_Tar\_COO}, & \text{for COO kernel} \\ T_{Init\_Tar\_HYB}, & \text{for HYB kernel} \end{cases}$$

4.1. CSR Kernel

4.1.1. Benchmark Matrices We generate nine benchmark matrices on the target architecture according to the criteria we proposed for the baseline tool. Among the nine benchmark matrices, five benchmark matrices are generated with the same number of matrix strips, denoted by I_0, but with different P_{NZ_BM} values. Let M_0 denote the benchmark matrix with the minimal P_{NZ_BM} value. Let X_{00} denote this minimal P_{NZ_BM} value and $T_{0\_Tar\_0}$ denote the execution time of matrix M_0. Additionally, two more benchmark matrices are generated with X_{00} as the P_{NZ_BM} value, but with different number of matrix strips, denoted by I_1 and I_2, respectively. The two matrices, together with matrix M_0, are used to estimate $C_{2\_Tar\_CSR\_LT}$, the coefficient of the linear relationship established between the number of matrix strips and the execution times of the three benchmark matrices. After comparing the values of I_0, I_1, and I_2, the benchmark matrix with the minimal number of strips is denoted by M_1. Let T_{10} denote the execution time of matrix M_1.

Among the five benchmark matrices, it is required that there exists a benchmark matrix N_0 with P_{NZ_BM} = TAR_CSR_THD and a benchmark matrix N_1 with P_{NZ_BM} = REF_CSR_THD, where TAR_CSR_THD and REF_CSR_THD denote CSR threshold [1] of the target and reference GPU architectures, respectively. The CSR threshold is the value of max threads per block, which is determined by the physical limitations of a GPU architecture. Corresponding to X_{00}, $T_{0\_Tar\_0}$, $C_{2\_Tar\_CSR\_LT}$, and T_{10}, for matrix N_0, their counterparts, denoted by X_{01}, $T_{0\_Tar\_1}$, $C_{2\_Tar\_CSR\_RT}$, and T_{11}, respectively, can be known in a similar way. Similarly, for matrix N_1, X_{02}, $T_{0\_Tar\_2}$, and T_{12} can also be achieved. The three benchmark matrices, whose P_{NZ_BM} values are less than or equal to TAR_CSR_THD, are used to estimate $C_{1\_Tar\_CSR\_LT}$. Corresponding, the

DOI: 10.1002/cpe

three benchmark matrices, whose P_{NZ_BM} values are greater than or equal to TAR_CSR_THD, are used to estimate $C_{1\_Tar\_CSR\_RT}$. Divided by TAR_CSR_THD, $C_{1\_Tar\_CSR\_LT}$ and $C_{1\_Tar\_CSR\_RT}$ represent two specific coefficients of two linear relationships both established between P_{NZ_BM} and the execution times of the three benchmark matrices. Although, theoretically, we can estimate $C_{1\_Tar\_CSR\_LT}$, $C_{1\_Tar\_CSR\_RT}$, $C_{2\_Tar\_CSR\_LT}$, and $C_{2\_Tar\_CSR\_RT}$ using two benchmark matrices, respectively. To enhance accuracy, we generate three benchmark matrices.

4.1.2. Relative Performance According to our modeling algorithm for CSR SpMV kernel in the baseline tool, the execution time of matrix A, represented by E, is computed by the equation: $E = E_0 \times E_1$, where $E_0 = \frac{T_{1\_Ref}}{T_{0\_Ref}}$ and $E_1 = E_{Ref\_CSR}(I_{Ref\_CSR})$. E_{Ref_CSR} represents the equation of the linear relationship established between the number of matrix strips and the execution times of the benchmark matrices generated on the reference architecture. I_{Ref_CSR}, the number of matrix strips of matrix A computed for the reference architecture, is calculated by the equation: $I_{Ref\_CSR} = \lceil \frac{N_R}{S_{Ref\_CSR}} \rceil$, where N_R represents the number of rows of matrix A and the notation $\lceil\ \rceil$ represents the rounding computation. Let P_1 and P_2 denote the relative performance for these two parts, respectively. Thus, the relative performance P_{CSR} is approximately estimated by:

$$P_{CSR} = P_1 \times P_2$$

where,

$$P_1 = \frac{C_{1\_Tar\_CSR}}{C_{1\_Ref\_CSR}} \times \frac{T_{0\_Ref}}{T_{0\_Tar}} \times (1 - \frac{T_{0\_Ref}}{T_{1\_Ref}}) + \frac{T_{0\_Ref}}{T_{1\_Ref}}$$

and

$$P_2 = \frac{C_{2\_Tar\_CSR}}{C_{2\_Ref\_CSR}} \times \frac{S_{Ref\_CSR}}{S_{Tar\_CSR}}$$

T_{0_Ref} and T_{1_Ref} are computed as follows:

$$T_{0\_Ref} = C_{1\_Ref\_CSR} \times X_0 + B_0$$

$$T_{1\_Ref} = C_{1\_Ref\_CSR} \times P_{NZ\_CSR} + B_0$$

Since there exists CSR threshold, which is the value of max threads per block, $C_{1\_Tar\_CSR}$, $C_{2\_Tar\_CSR}$, T_{0_Tar}, X_0, and B_0 are represented as follows:

$$C_{1\_Tar\_CSR} = \begin{cases} C_{1\_Tar\_CSR\_LT}, & \text{if } P_{NZ\_CSR} \leq \text{TAR\_CSR\_THD} \\ C_{1\_Tar\_CSR\_RT}, & \text{if } P_{NZ\_CSR} \geq \text{TAR\_CSR\_THD} \end{cases}$$

$$C_{2\_Tar\_CSR} = \begin{cases} C_{2\_Tar\_CSR\_LT}, & \text{if } P_{NZ\_CSR} \leq \text{TAR\_CSR\_THD} \\ C_{2\_Tar\_CSR\_RT}, & \text{if } P_{NZ\_CSR} \geq \text{TAR\_CSR\_THD} \end{cases}$$

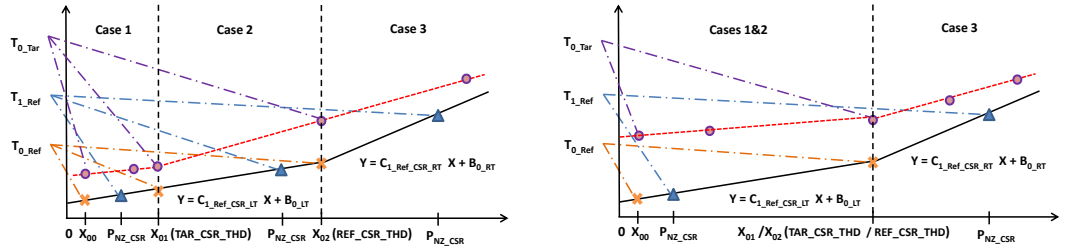
$$T_{0\_Tar} = \begin{cases} T_{0\_Tar\_0}, & \text{Case 1: if } P_{NZ\_CSR} \leq \text{TAR\_CSR\_THD} \\ T_{0\_Tar\_1}, & \text{Case 2: if } P_{NZ\_CSR} \in [\text{ TAR\_CSR\_THD , REF\_CSR\_THD }] \\ T_{0\_Tar\_2}, & \text{Case 3: if } P_{NZ\_CSR} \geq \text{REF\_CSR\_THD} \end{cases}$$

$$X_0 = \begin{cases} X_{00}, & \text{Case 1: if } P_{NZ\_CSR} \leq \text{TAR\_CSR\_THD} \\ X_{01}, & \text{Case 2: if } P_{NZ\_CSR} \in [\text{ TAR\_CSR\_THD , REF\_CSR\_THD }] \\ X_{02}, & \text{Case 3: if } P_{NZ\_CSR} \geq \text{REF\_CSR\_THD} \end{cases}$$

$$B_0 = \begin{cases} B_{0\_LT}, & \text{if } P_{NZ\_CSR} \leq \text{REF\_CSR\_THD} \\ B_{0\_RT}, & \text{if } P_{NZ\_CSR} \geq \text{REF\_CSR\_THD} \end{cases}$$

Some parameters and their corresponding relationships are illustrated in Figure 1. After considering the reference GPU architecture we used in the modeling, *i.e.*, NVIDIA Tesla C2050, and all existing NVIDIA GPU architectures so far, whose maximum value of compute capability is 3.5, we exclude the possibility of TAR_CSR_THD > REF_CSR_THD since it does not exist.

DOI: 10.1002/cpe



(a) TAR_CSR_THD < REF_CSR_THD (b) TAR_CSR_THD = REF_CSR_THD

Figure 1. Number of non-zero elements per row (X axis) *vs* execution time (Y axis). The black and red lines represent the established and assumed relationships for the reference and target architectures, respectively.

$C_{1\_Ref\_CSR}$ and $C_{2\_Ref\_CSR}$ are represented by the equations which are similar to that of $C_{1\_Tar\_CSR}$ and $C_{2\_Tar\_CSR}$, respectively, and their corresponding values can be known from the parameterized baseline tool, which is instantiated by the benchmark matrices generated on the reference architecture. S_{Tar_CSR} is computed according to the method we proposed for the baseline tool in Section 3. S_{Ref_CSR} can be known from the parameterized baseline tool, which is instantiated by the physical limitations of the reference architecture.

4.1.3. Initialization Time The initialization time of CSR kernel for the reference and the target architectures are computed as follows:

$$T_{Init\_Ref\_CSR} = \begin{cases} T_{00}, & \text{if } P_{NZ\_CSR} \leq \text{TAR\_CSR\_THD} \\ T_{01}, & \text{if } P_{NZ\_CSR} \in [\text{ TAR\_CSR\_THD , REF\_CSR\_THD }] \\ T_{02}, & \text{if } P_{NZ\_CSR} \geq \text{REF\_CSR\_THD} \end{cases}$$

$$T_{Init\_Tar\_CSR} = \begin{cases} T_{10}, & \text{if } P_{NZ\_CSR} \leq \text{TAR\_CSR\_THD} \\ T_{11}, & \text{if } P_{NZ\_CSR} \in [\text{ TAR\_CSR\_THD , REF\_CSR\_THD }] \\ T_{12}, & \text{if } P_{NZ\_CSR} \geq \text{REF\_CSR\_THD} \end{cases}$$

Similar to T_{10}, T_{11}, and T_{12}, T_{00}, T_{01}, and T_{02} represent the execution times of the corresponding benchmark matrices generated on the reference architecture, which are the counterparts of the benchmark matrices M_1, N_0, and N_1, respectively.

4.2. ELL Kernel

4.2.1. Benchmark Matrices We generate five benchmark matrices on the target architecture according to the criteria we proposed for the baseline tool. Among the five benchmark matrices, three benchmark matrices generated with the same number of matrix strips, denoted by I_0, but with different P_{NZ_BM} values, are used for estimating C_{Tar_ELL}, the coefficient of the linear relationship established between P_{NZ_BM} and the execution times of the three benchmark matrices. Let M_2 denote the benchmark matrix with the minimal P_{NZ_BM} value. Let X_1 denote this minimal P_{NZ_BM} value. Additionally, two more benchmark matrices are generated with the same P_{NZ_BM} value, *i.e.*, X_1, but with different number of matrix strips, denoted by I_1 and I_2, respectively. The two matrices, together with matrix M_2, are used for estimating E_{Tar_ELL}, the equation of the linear relationship established between the number of matrix strips and the execution times of the three benchmark matrices. Although, theoretically, we can estimate C_{Tar_ELL} and E_{Tar_ELL}, using two benchmark matrices, respectively. To enhance accuracy, we generate three benchmark matrices.

4.2.2. Relative Performance According to our modeling algorithm for ELL SpMV kernel in the baseline tool, knowing the number of matrix strips of matrix A, the coefficient of the linear relationship-1 established between P_{NZ_BM} and the execution time of the benchmark matrices can be estimated. Then, the intercept of the linear relationship-1 can be estimated by the linear relationship-2 established between the number of matrix strips and the execution time when the

DOI: 10.1002/cpe

unknown number of matrix strips is assigned as the specific value of matrix A. Finally, the execution time of matrix A can be estimated by replacing unknown P_{NZ_BM} in linear relationship-1 with the specific P_{NZ_ELL} value of matrix A. For cross-architecture prediction, assume that two linear relationships, *i.e.*, relationship-1, which are established for the benchmark matrices generated on the reference and target architecture, are represented by $Y = C_{Ref\_ELL} \times X + B_1$ and $Y = C_{Tar\_ELL} \times X + B_2$, respectively. Thus, the relative performance P_{ELL} is approximately estimated by:

$$P_{ELL} = \frac{C_{Tar\_ELL}}{C_{Ref\_ELL}} \times \frac{S_{Ref\_ELL}}{S_{Tar\_ELL}}$$

S_{Tar_ELL} is computed according to the method we proposed for the baseline tool in Section 3. S_{Ref_ELL} and C_{Ref_ELL} can be known from the parameterized baseline tool, which is instantiated by the physical limitations of the reference architecture and the benchmark matrices generated on the reference architecture, respectively.

4.2.3. Initialization Time The initialization time of ELL kernel for the reference and the target architectures are computed as follows:

$$T_{Init\_Ref\_ELL} = B_1$$
$$T_{Init\_Tar\_ELL} = B_2$$

B_1 and B_2 are the intercepts of the linear relationship-1 established for the benchmark matrices generated on the reference and target architecture, respectively. B_2 is computed as follows:

$$B_2 = E_{Tar\_ELL}(I_{Tar\_ELL}) - C_{Tar\_ELL} \times X_1 \times I_{Tar\_ELL}$$

I_{Tar_ELL} is the number of matrix strips of matrix A computed for the target architecture, which can be calculated by:

$$I_{Tar\_ELL} = \lceil \frac{N_R}{S_{Tar\_ELL}} \rceil$$

where N_R represents the number of rows of matrix A and the notation $\lceil\ \rceil$ represents the rounding computation.

The method to compute B_1 is similar to that of B_2, just replacing E_{Tar_ELL} and C_{Tar_ELL} with E_{Ref_ELL} and C_{Ref_ELL}, respectively, where E_{Ref_ELL} and C_{Ref_ELL} can be known from the parameterized baseline tool, which is instantiated by the benchmark matrices generated on the reference architecture.

4.3. COO Kernel

4.3.1. Benchmark Matrices We generate three benchmark matrices on the target architecture for estimating C_{Tar_COO}, the coefficient of the linear relationship established between the number of matrix strips and the execution times of the three benchmark matrices, according to the criteria we proposed for the baseline tool. Let T_1 denote the execution time of the benchmark matrix generated with the minimal number of matrix strips. Although, theoretically, we can estimate C_{Tar_COO}, using two benchmark matrices. To enhance accuracy, we generate three benchmark matrices.

4.3.2. Relative Performance According to our modeling algorithm for COO SpMV kernel in the baseline tool, knowing the number of matrix strips, the execution time of matrix A can be estimated by the linear relationship established between the number of matrix strips and the execution time of the benchmark matrices. For cross-architecture prediction, assume that two linear relationships, which are established for the benchmark matrices generated on the reference architecture and the target architecture, are represented by $Y = C_{Ref\_COO} \times X + B_{Ref\_COO}$ and $Y = C_{Tar\_COO} \times X + B_{Tar\_COO}$, respectively. Thus, the relative performance P_{COO} is approximately estimated by:

$$P_{COO} = \frac{C_{Tar\_COO}}{C_{Ref\_COO}} \times \frac{S_{Ref\_COO}}{S_{Tar\_COO}}$$

S_{Tar_COO} is computed according to the method we proposed for the baseline tool in Section 3. S_{Ref_COO} and C_{Ref_COO} can be known from the parameterized baseline tool, which is instantiated

DOI: 10.1002/cpe

by the physical limitations of the reference architecture and the benchmark matrices generated on the reference architecture, respectively.

4.3.3. Initialization Time The initialization time of COO kernel for the reference and the target architectures are computed as follows:

$$T_{Init\_Ref\_COO} = T_0$$
$$T_{Init\_Tar\_COO} = T_1$$

Similar to T_1, T_0 represents the execution time of the benchmark matrix generated on the reference architecture, whose number of matrix strips is minimal among three benchmark matrices.

4.4. HYB Kernel

4.4.1. Benchmark Matrices Since HYB kernel is the combination of ELL and COO kernels, the benchmarks matrices generated for modeling ELL and COO kernels can be reused. We do not need to generate any additional benchmark matrix for modeling HYB kernel.

4.4.2. Relative Performance According to our modeling algorithm for HYB SpMV kernel in the baseline tool, the execution time of matrix A equals to the sum of the execution time of ELL part of matrix A and that of COO part of matrix A, where ELL part and COO part of matrix A are divided by HYB threshold [2], denoted by HYB_THD. Thus, the relative performance P_{HYB} is approximately estimated by:

$$P_{HYB} = \begin{cases} P_{ELL}, & \text{if HYB\_THD = } P_{NZ\_ELL} \\ P_{COO}, & \text{if HYB\_THD = 0} \\ P_{ELL}, & \text{if HYB\_THD} \in (0,\ P_{NZ\_ELL}) \end{cases}$$

According to the above equation, P_{HYB} is estimated according to three cases. Case 1: no part of matrix A is in COO part, thus, $P_{HYB} = P_{ELL}$. Case 2: no part of matrix A is in ELL part, thus, $P_{HYB} = P_{COO}$. Case 3: matrix A is in both ELL and COO parts, thus, $P_{HYB} = P_{ELL}$.

4.4.3. Initialization Time The initialization time of HYB kernel for the reference and the target architectures are computed as follows:

$$T_{Init\_Ref\_HYB} = \begin{cases} T_{Init\_Ref\_ELL}, & \text{if HYB\_THD = } P_{NZ\_ELL} \\ T_{Init\_Ref\_COO}, & \text{if HYB\_THD = 0} \\ T_{Init\_Ref\_ELL} + \frac{P_{COO}}{P_{ELL}} \times T_{Init\_Ref\_COO}, & \text{if HYB\_THD} \in (0,\ P_{NZ\_ELL}) \end{cases}$$

$$T_{Init\_Tar\_HYB} = \begin{cases} T_{Init\_Tar\_ELL}, & \text{if HYB\_THD = } P_{NZ\_ELL} \\ T_{Init\_Tar\_COO}, & \text{if HYB\_THD = 0} \\ T_{Init\_Tar\_ELL} + T_{Init\_Tar\_COO}, & \text{if HYB\_THD} \in (0,\ P_{NZ\_ELL}) \end{cases}$$

5. EXPERIMENTAL EVALUATION

5.1. Experimental Architectures

Our experiments are performed on four architectures: NVIDIA Tesla C2050, Tesla M2090, Tesla K20m, and GeForce GTX 295, whose global memory are 3GB, 5GB, 5GB, and 1GB, respectively. In our experiments, NVIDIA Tesla C2050 is used as the reference architecture, the other three are used as the target architectures. The physical limitations of four architectures are summarized in Table I. The parameters and their specific values used in Section 4, derived from the parameterized baseline tool instantiated by the reference architecture, are listed in Table II, where $S'(M'(0.011, I'), 0.062)$ represents $0.011 \times I_{Ref\_ELL} + 0.062$ and $I_{Ref\_ELL} = \lceil \frac{N_R}{S_{Ref\_ELL}} \rceil$ (Refer to Section 4.2.3).

DOI: 10.1002/cpe

Table I. Experimental architectures used in our experimental evaluation.

Architectures	# SM	Warps / SM	Threads / SM	Max. Threads / Block	Capability
Tesla C2050	14	48	1536	1024	2.0
Tesla M2090	16	48	1536	1024	2.0
Tesla K20m	13	64	2048	1024	3.5
GeForce GTX 295	30	32	1024	512	1.3

Table II. Parameters and their specific values known from the baseline tool.

Parameters	Values	Parameters	Values	Parameters	Values
$C_{1\_Ref\_CSR\_LT}$	0.00027	$C_{1\_Ref\_CSR\_RT}$	0.00029	S_{Ref_CSR}	672
$C_{2\_Ref\_CSR\_LT}$	0.0061	$C_{2\_Ref\_CSR\_RT}$	0.06	S_{Ref_ELL}	21504
B_{0_LT}	0.078	B_{0_LT}	0.066	S_{Ref_COO}	21504
C_{Ref_ELL}	0.002	C_{Ref_COO}	0.0084	E_{Ref_ELL}	$S'(M'(0.011, I'), 0.062)$

Table III. Sparse matrices used in our experimental evaluation.

Matrix	Dimensions	NZs	Matrix	Dimensions	NZs
Dense	2K*2K	4.0 M	FEM/Harbor	47K*47K	2.37 M
Protein	36K*36K	4.3 M	QCD	49K*49K	1.90 M
FEM/Spheres	83K*83K	6.0 M	FEM/Ship	141K*141K	7.81 M
FEM/Cantilever	62K*62K	4.0 M	Epidemiology	526K*526K	2.1 M
Transport	1.6M*1.6M	23.5 M	CurlCurl_4	2.38M*2.38M	26.52 M
af_shell1	505K*505K	17.59 M	hood	221K*221K	10.77 M
BenElechi1	246K*246K	13.15 M	msdoor	416K*416K	20.24 M

5.2. *Experimental Sparse Matrices*

The sparse matrices used in our experiments are shown in Table III, where NZs represents the total number of non-zero elements. They are from widely-used sparse matrices collection [31, 32]. Our experimental evaluation does not adopt the rest 6 matrices ("Wind Tunnel", "Economics", "FEM/Accelerator", "Circuit", "Webbase", and "LP") in [32] since NVIDIA's SpMV implementations cannot execute ELL SpMV kernel on the reference architecture because of the limitation of "$num\_cols\_per\_row$".

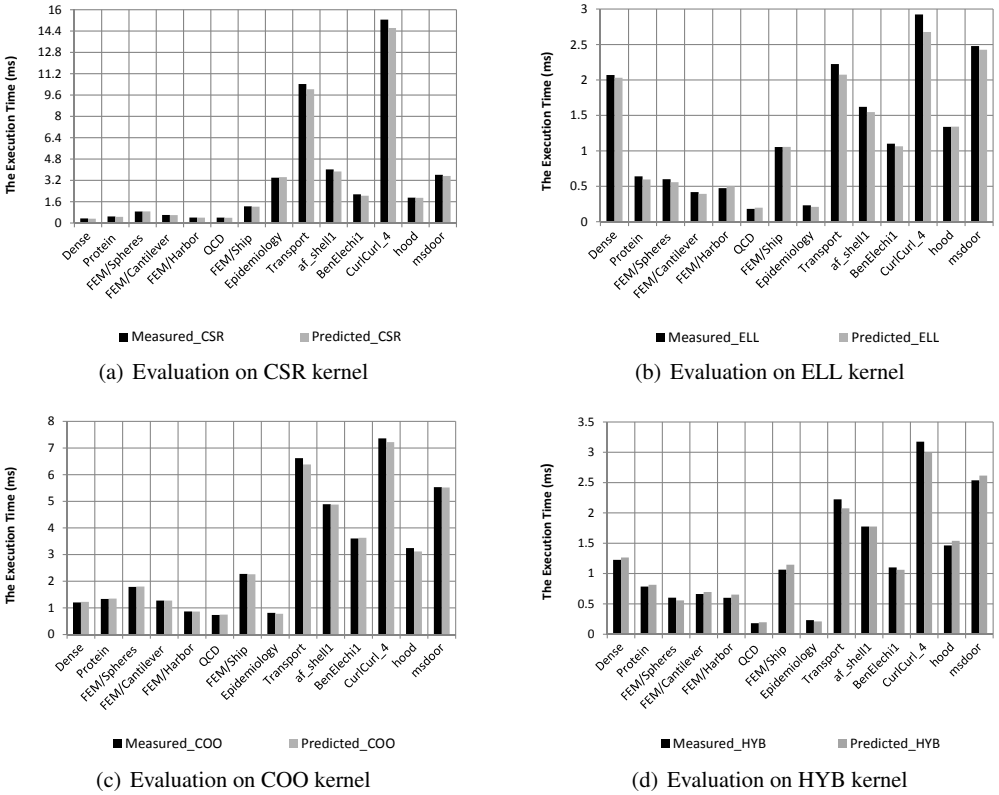
5.3. *Experimental Environment*

In our experiments, the version numbers of CUDA libraries installed on NVIDIA Tesla C2050, Tesla M2090, Tesla K20m, and GeForce GTX 295 are 4.1, 5.0, 5.0, and 4.1, respectively. Although, SpMV performance can be slightly affected by different CUDA versions, the accuracy of performance modeling tool is not affected. For a specific GPU architecture, we only evaluate whether the predicted and measured SpMV execution times on every sparse matrix can match well.

5.4. *Accuracy of Performance Modeling Tool*

For a sparse matrix, assume that its SpMV kernel execution times measured on a reference GPU architecture, NVIDIA Tesla C2050, are known. Our cross-architecture performance modeling tool can report its predicted SpMV kernel execution times on a target GPU architectures, *e.g.*, NVIDIA Tesla M2090, Tesla K20m, and GeForce GTX 295. The SpMV kernels used in our modeling tool for performance prediction include CSR, ELL, COO, and HYB. Figures 2, 3, and 4 compare the measured SpMV kernel execution times for CSR, ELL, COO, and HYB kernels, represented by black bars, and the predicted kernel execution times by our modeling tool, represented by gray bars, on the target GPU architectures NVIDIA Tesla M2090, Tesla K20m, and GeForce GTX 295, respectively. For NVIDIA Tesla M2090, the average performance differences between the

DOI: 10.1002/cpe



(a) Evaluation on CSR kernel

(b) Evaluation on ELL kernel

(c) Evaluation on COO kernel

(d) Evaluation on HYB kernel

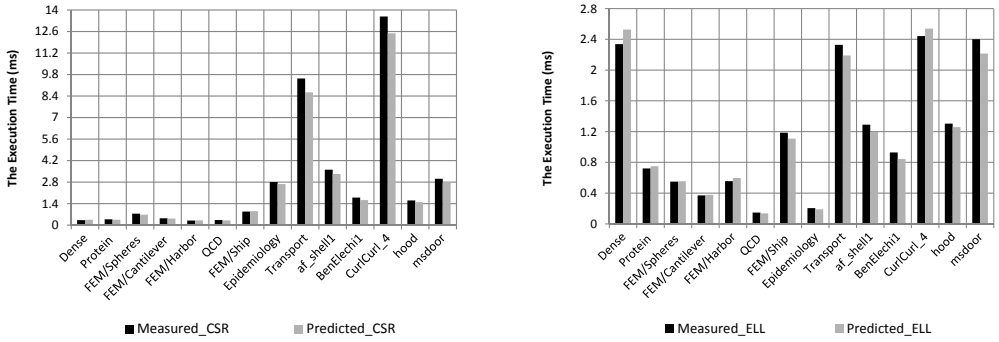
Figure 2. Performance modeling evaluation on NVIDIA Tesla M2090.

predicted and measured SpMV kernel execution times for CSR, ELL, COO, and HYB SpMV kernels are 3.1%, 5.1%, 1.6%, and 5.6%, respectively. For Tesla K20m, they are 6.9%, 5.9%, 4.0%, and 6.6% on the average, respectively. For GeForce GTX 295, they are 5.9%, 5.8%, 3.8%, and 5.9% on the average, respectively. In our experiments, the measured kernel execution times, obtained by averaging the total 500 times of executions of the SpMV kernel, are relative stable and accurate. Note that, the warm up times of the GPUs are excluded. For each SpMV kernel, we first compute the performance differences between predicted and measured kernel execution times for 14 experimental matrices. Then, for each kernel, we average the performance differences in percentage on 14 experimental matrices. Thus, it is reasonable to claim accuracy to 0.1%.

5.5. Analysis of Experimental Results

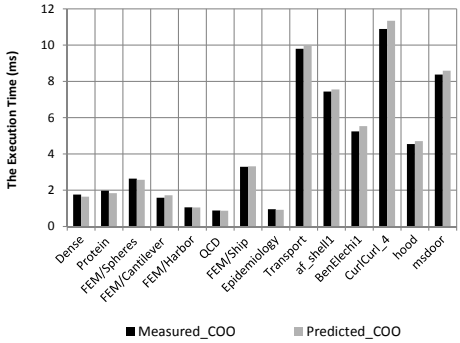
Given a sparse matrix and a GPU architecture, it is straightforward to select an optimal SpMV storage format and corresponding SpMV kernel by comparing the performance shown in Figures 2, 3, and 4. However, given a sparse matrix and an SpMV storage format, when choosing an appropriate GPU architecture, many important factors (*e.g.*, the performance, the cost, the memory size, and the power consumption) may be considered by researchers to make their choice. Our modeling tool can offer support for performance prediction in an accurate and easy way. For some applications with small size, their execution times on different GPUs are similar. However, for some others, the advantage of a new-generation GPU is more obvious. An older GPU with competitive lower price is good enough for researchers if it can satisfy the memory requirement of an application and offer similar performance as a high-end GPU. Otherwise, researchers may consider to purchase a new-generation GPU. For example, when performing ELL SpMV kernel computation with matrix "FEM/Spheres", after comparing the performance on all our experimental GPU architectures, we show that GeForce GTX 295 is good enough. However, Tesla K20m might be a better choice when performing the same kernel computation with matrix "CurlCurl_4".

DOI: 10.1002/cpe

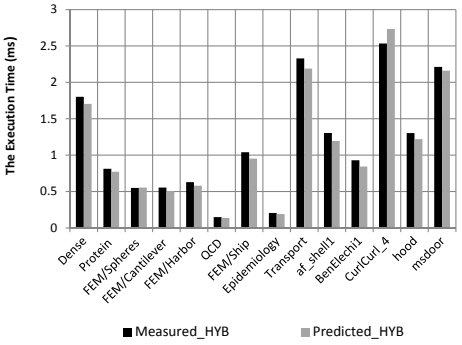


(a) Evaluation on CSR kernel

(b) Evaluation on ELL kernel



(c) Evaluation on COO kernel



(d) Evaluation on HYB kernel

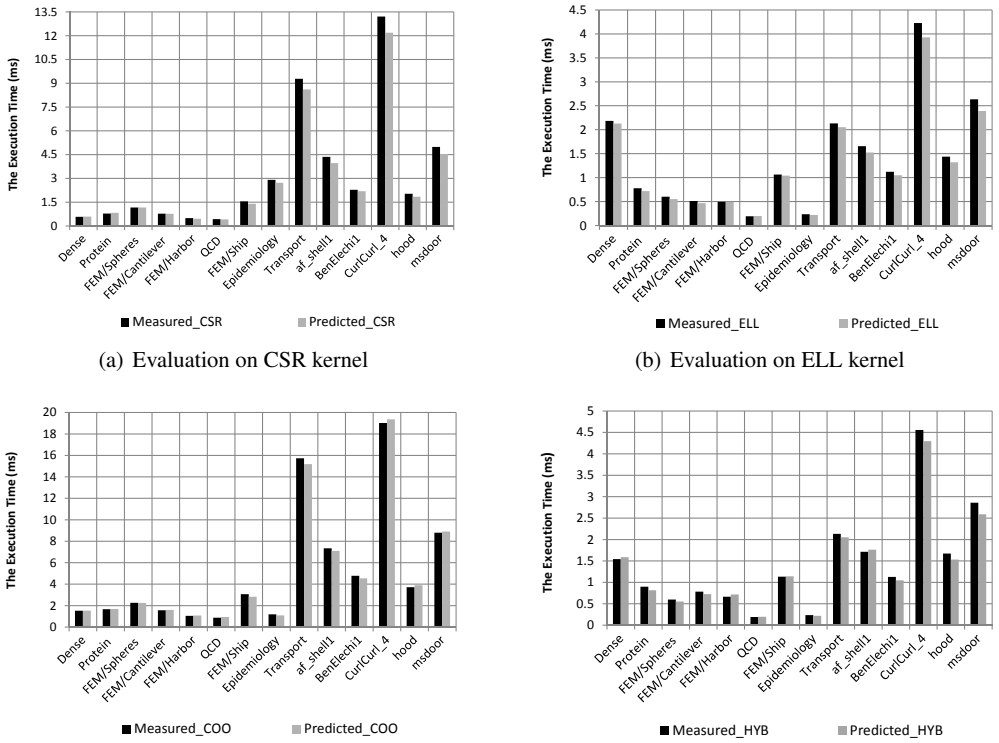
Figure 3. Performance modeling evaluation on NVIDIA Tesla K20m.

6. CONCLUSIONS

In this paper, we propose an integrated analytical and profile-based cross-architecture performance modeling tool to specifically provide inter-architecture SpMV performance prediction on NVIDIA GPU architectures. Our tool includes four performance models, *i.e.*, CSR, ELL, COO, and HYB models. For a sparse matrix, knowing its SpMV kernel performance measured on the reference GPU architecture, our tool can accurately predict its performance on the target GPU architectures. The prediction results can effectively assist researchers in making choice of an appropriate architecture that best fits their needs from a wide range of available computing architectures.

REFERENCES

1. Guo P, Wang L. Accurate CUDA performance modeling for sparse matrix-vector multiplication. *The 2012 International Conference on High Performance Computing and Simulation (HPCS)*, IEEE, 2012; 496 –502,doi: 10.1109/HPCSim.2012.6266964.
2. Bell N, Garland M. Implementing sparse matrix-vector multiplication on throughput-oriented processors. *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, ACM, 2009; 18:1–18:11, doi:10.1145/1654059.1654078.
3. Bolz J, Farmer I, Grinspun E, Schroder P. Sparse matrix solvers on the GPU: conjugate gradients and multigrid. *ACM Trans. Graph.* 2003; **22**(3):917–924, doi:10.1145/882262.882364.
4. Vazquez F, Ortega G, Fernandez JJ, Garzon EM. Improving the performance of the sparse matrix vector product with GPUs. *Proceedings of the 2010 10th IEEE International Conference on Computer and Information Technology*, CIT '10, IEEE Computer Society, 2010; 1146–1151, doi:10.1109/CIT.2010.208.
5. Monakov A, Lokhmotov A, Avetisyan A. Automatically tuning sparse matrix-vector multiplication for GPU architectures. *Proceedings of the 5th international conference on High Performance Embedded Architectures and Compilers*, HiPEAC'10, Springer-Verlag, 2010; 111–125, doi:10.1007/978-3-642-11515-8_10.
6. Dang HV, Schmidt B. The sliced coo format for sparse matrix-vector multiplication on CUDA-enabled GPUs. *Procedia Computer Science* 2012; **9**(0):57 – 66, doi:10.1016/j.procs.2012.04.007. Proceedings of the International Conference on Computational Science, ICCS 2012.



(a) Evaluation on CSR kernel

(b) Evaluation on ELL kernel

(c) Evaluation on COO kernel

(d) Evaluation on HYB kernel

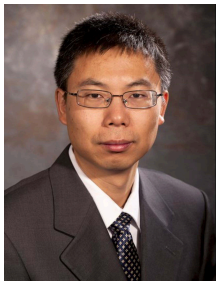
Figure 4. Performance modeling evaluation on NVIDIA GeForce GTX 295.

7. Sun X, Zhang Y, Wang T, Zhang X, Yuan L, Rao L. Optimizing spmv for diagonal sparse matrices on GPU. *2011 International Conference on Parallel Processing (ICPP)*, 2011; 492 –501, doi:10.1109/ICPP.2011.53.
8. Guo P, Wang L. Auto-Tuning CUDA parameters for sparse matrix-vector multiplication on GPUs. *Proceedings of the 2010 International Conference on Computational and Information Sciences*, ICCIS '10, IEEE Computer Society, 2010; 1154–1157, doi:10.1109/ICCIS.2010.285.
9. Kubota Y, Takahashi D. Optimization of sparse matrix-vector multiplication by auto selecting storage schemes on GPU. *Proceedings of the 2011 international conference on Computational science and its applications - Volume Part II*, ICCSA'11, Springer-Verlag, 2011; 547–561, doi:10.1007/978-3-642-21887-3_42.
10. Baskaran MM, Bordawekar R. Optimizing sparse matrix-vector multiplication on GPUs. *Technical Report*, Research Report RC24704, IBM TJ Watson Research Center december 2008.
11. Pichel JC, Rivera FF, Fernandez M, Rodriguez A. Optimization of sparse matrix-vector multiplication using reordering techniques on GPUs. *Microprocessors and Microsystems* 2012; **36**(2):65 – 77, doi:10.1016/j.micpro.2011.05.005.
12. Sim J, Dasgupta A, Kim H, Vuduc R. A performance analysis framework for identifying potential benefits in GPGPU applications. *Proceedings of the 17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming*, PPoPP '12, ACM, 2012; 11–22, doi:10.1145/2145816.2145819.
13. Choi JW, Singh A, Vuduc RW. Model-driven autotuning of sparse matrix-vector multiply on GPUs. *PPoPP '10: Proceedings of the 15th ACM SIGPLAN symposium on Principles and practice of parallel programming*, PPoPP '10, ACM, 2010; 115–126, doi:10.1145/1693453.1693471.
14. Karakasis V, Goumas G, Koziris N. Perfomance models for blocked sparse matrix-vector multiplication kernels. *Proceedings of the 2009 International Conference on Parallel Processing*, ICPP '09, IEEE Computer Society, 2009; 356–364, doi:10.1109/ICPP.2009.21.
15. Zhang Y, Owens J. A quantitative performance analysis model for GPU architectures. *Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture*, HPCA '11, 2011; 382–393, doi:10.1109/HPCA.2011.5749745.
16. Guo P, Huang H, Chen Q, Wang L, Lee EJ, Chen P. A model-driven partitioning and auto-tuning integrated framework for sparse matrix-vector multiplication on GPUs. *Proceedings of the 2011 TeraGrid Conference: Extreme Digital Discovery*, TG '11, ACM, 2011; 2:1–2:8, doi:10.1145/2016741.2016744.
17. Xu S, Xue W, Lin H. Performance modeling and optimization of sparse matrix-vector multiplication on NVIDIA CUDA platform. *The Journal of Supercomputing* 2013; **63**:710–721, doi:10.1007/s11227-011-0626-0.
18. Resios A. GPU performance prediction using parametrized models. Master's Thesis, Utrecht University 2011.
19. Jia H, Zhang Y, Long G, Xu J, Yan S, Li Y. Gpuroofline: A model for guiding performance optimizations on GPUs. *Euro-Par 2012 Parallel Processing*, *Lecture Notes in Computer Science*, vol. 7484, Kaklamanis C, Papatheodorou T, Spirakis P (eds.). Springer Berlin Heidelberg, 2012; 920–932, doi:10.1007/978-3-642-32820-6_90.

20. Cui Z, Liang Y, Rupnow K, Chen D. An accurate gpu performance model for effective control flow divergence optimization. *2012 IEEE 26th International Parallel Distributed Processing Symposium (IPDPS)*, 2012; 83 –94, doi:10.1109/IPDPS.2012.18.
21. Dinkins S. A model for predicting the performance of sparse matrix vector multiply (spmv) using memory bandwidth requirements and data locality. Master's Thesis, Colorado State University 2012.
22. Baghsorkhi SS, Delahaye M, Gropp WD, Wen-mei WH. Analytical performance prediction for evaluation and tuning of GPGPU applications. *Workshop on Exploiting Parallelism using GPUs and other Hardware-Assisted Methods (EPHAM?9), In conjunction with The International Symposium on Code Generation and Optimization (CGO) 2009*, 2009.
23. Baghsorkhi SS, Delahaye M, Patel SJ, Gropp WD, Hwu WmW. An adaptive performance modeling tool for GPU architectures. *Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '10, ACM, 2010; 105–114, doi:10.1145/1693453.1693470.
24. Kothapalli K, Mukherjee R, Rehman M, Patidar S, Narayanan P, Srinathan K. A performance prediction model for the CUDA GPGPU platform. *2009 International Conference on High Performance Computing (HiPC)*, 2009; 463–472, doi:10.1109/HIPC.2009.5433179.
25. Yang LT, Ma X, Mueller F. Cross-platform performance prediction of parallel applications using partial execution. *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, SC '05, IEEE Computer Society, 2005; 40–, doi:10.1109/SC.2005.20.
26. Marin G, Mellor-Crummey J. Cross-architecture performance predictions for scientific applications using parameterized models. *Proceedings of the joint international conference on Measurement and modeling of computer systems*, SIGMETRICS '04/Performance '04, ACM, 2004; 2–13, doi:10.1145/1005686.1005691.
27. Hong S, Kim H. An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness. *Proceedings of the 36th annual international symposium on Computer architecture*, ISCA '09, ACM, 2009; 152–163, doi:10.1145/1555754.1555775.
28. Guo P, Wang L, Chen P. A performance modeling and optimization analysis tool for sparse matrix-vector multiplication on GPUs. *IEEE Transactions on Parallel and Distributed Systems* 2013; **99**(PrePrints):1, doi: http://doi.ieeecomputersociety.org/10.1109/TPDS.2013.123.
29. Schaa D, Kaeli D. Exploring the multiple-GPU design space. *Proceedings of the 2009 IEEE International Parallel & Distributed Processing Symposium*, IPDPS '09, IEEE Computer Society, 2009; 1–12, doi:10.1109/IPDPS.2009.5161068.
30. Liu W, Muller-Wittig, Schmidt B. Performance predictions for general-purpose computation on GPUs. *2007 International Conference on Parallel Processing (ICPP)*, 2007; 50, doi:10.1109/ICPP.2007.67.
31. Davis TA, Hu Y. The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software* 2011; **38**(1):1:1–1:25, doi:10.1145/2049662.2049663.
32. Williams S, Oliker L, Vuduc R, Shalf J, Yelick K, Demmel J. Optimization of sparse matrix-vector multiplication on emerging multicore platforms. *Proc. 2007 ACM/IEEE Conference on Supercomputing*, SC '07, ACM, 2007; 38:1–38:12, doi:10.1145/1362622.1362674.

AUTHORS' BIOGRAPHIES

Ping Guo is a Ph.D. candidate in the Department of Computer Science at the University of Wyoming. Her research focuses on designing High Performance Computing (HPC) system on Graphics Processing Unit (GPU) platform. She received the BS degree in computer science from Harbin University of Science and Technology, China, in 2005, and the MS degree in computer science from University of Kentucky, in 2008.



Liqiang Wang is an associate professor in the Department of Computer Science at the University of Wyoming. He received the BS degree in mathematics from Hebei Normal University, China, in 1995, the MS degree in computer science from Sichuan University, China, in 1998, and the PhD degree in computer science from Stony Brook University in 2006. His research interests include the design and analysis of data-intensive parallel computing systems, including GPU and Cloud Computing. He received a US National Science Foundation CAREER Award in 2011.