Journal of Computer and Communications, 2017, 5, 65-83
http://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

Performance Prediction Based on Statistics of Sparse Matrix-Vector Multiplication on GPUs*

Ruixing Wang[1], Tongxiang Gu[2#], Ming Li[3]

[1]Country Graduate School of Chinese Academy of Engineering Physics, Beijing, China
[2]Laboratory of Computational Physics, Institute of Applied Physics and Computational Mathematics, Beijing, China
[3]Country Xi'an Aeronautics Computing Technique Research Institute, AVIC, Xi'an, China
Email: bitwangruixing@163.com, #txgu@iapcm.ac.cn, hitliming@163.com

How to cite this paper: Wang, R.X., Gu, T.X. and Li, M. (2017) Performance Prediction Based on Statistics of Sparse Matrix-Vector Multiplication on GPUs. *Journal of Computer and Communications*, **5**, 65-83.
https://doi.org/10.4236/jcc.2017.56005

Received: March 17, 2017
Accepted: April 25, 2017
Published: April 28, 2017

Abstract

As one of the most essential and important operations in linear algebra, the performance prediction of sparse matrix-vector multiplication (SpMV) on GPUs has got more and more attention in recent years. In 2012, Guo and Wang put forward a new idea to predict the performance of SpMV on GPUs. However, they didn't consider the matrix structure completely, so the execution time predicted by their model tends to be inaccurate for general sparse matrix. To address this problem, we proposed two new similar models, which take into account the structure of the matrices and make the performance prediction model more accurate. In addition, we predict the execution time of SpMV for CSR-V, CSR-S, ELL and JAD sparse matrix storage formats by the new models on the CUDA platform. Our experimental results show that the accuracy of prediction by our models is 1.69 times better than Guo and Wang's model on average for most general matrices.

Keywords

Sparse Matrix-Vector Multiplication, Performance Prediction, GPU, Normal Distribution, Uniform Distribution

1. Introduction

Sparse matrix-vector multiplication (SpMV) is an essential operation in solving linear systems and eigenvalue problems. For many iterative methods, the fraction of the execution time of SpMV may be more than 80% in the total time, so the study of its performance has attracted a lot of attention. Right now, the GPU has been from a graphics accelerator to a computing device with a broad spectrum of purposes, due to the characteristics of the multi-thread, high memory

*The project is partly supported by the NSF of China (61472462, 11671049) and foundation of Key Laboratory of Computational Physics.

bandwidth. It can solve massively parallel problems and obtain very high performance. However, how to predict the execution time of SpMV on GPUs accurately is still a big challenge.

In 2003, Bolz *et al.* [1] first implemented conjugate gradient (CG) method on GPU which contains SpMV. Bell and Garland [2] proposed SpMV CUDA kernels for some well-known sparse matrix storage formats, such as compressed sparse row (CSR), Ellpack-Itpack (ELL), Coordinate (COO), Diagonal (DIA) and Hybrid (HYB). The jagged diagonal format (JAD) was used to implement the SpMV kernel in [3], and they also realized the GPU-accelerated preconditioned CG and GMRES methods. In this paper, we utilize the SpMV kernels in [3]: CSR-V, CSR-S, ELL and JAD.

In order to improve the performance of SpMV on GPUs, Vazquez *et al.* [4] presented a new sparse storage format, termed ELLR-T, which is improved from ELL format, to achieve higher performance. Monakov *et al.* [5] put forward a sliced ELL format and used auto-tuning to find the optimal configuration for batter performance. Zheng and Gu [6] proposed bisection ELL (BiELL) and bisection JAD (BiJAD) format based on ELL and JAD format for optimizing SpMV on GPUs. Especially for irregular matrices, BiELL and BiJAD format can get greater load balance and higher performance by adjusting the elements storage location of matrix and make the zero-padding less. Meanwhile, they realized CG and GMRES method with BiELL and BiJAD format on GPU [7]. Choi *et al.* [8] designed blocked ELL format and select proper parameters for better performance. Guo *et al.* [9] proposed an auto-tuning framework that can select the parameters of SpMV kernels to obtain the optimal performance on GPUs.

Besides studying how to improve the performance of SpMV on the GPUs, there are also many performance models focusing on performance prediction. Resios [10] proposed a parameterized analytical model to estimate execution time and identify potential bottlenecks in programs. Dinkins [11] put forward a model for predicting SpMV performance using memory bandwidth requirements and data locality. Werkhoven *et al.* [12] gave an analytical performance model that includes PCIe transfers and overlapping computation and communication to predict the execution time for CPU-GPU data transfers. Baghsorkhi *et al.* [13] presented a model to predict the performance of GPU applications based on the string and work flow graph. Guo *et al.* [9] showed a performance modeling and optimizing analysis to predict and optimize SpMV performance on GPUs. A simple analytical GPU model to predict the execution time of massively parallel programs was given by Hong *et al.* [14]. Schaa *et al.* [15] presented a model to accurately estimate the execution time of GPU applications by varying the configurations. In a word, most of the performance prediction researches analyze and predict the execution time from the perspective of the machine itself, which focus on the physical properties and parameters of GPUs. There has few performance prediction models were established from the view of mathematics.

In this paper, we present two new improved models based on [16] and get better prediction accuracy. Our models mainly consist of two phases: generating parameters and fitting for prediction. In the first phase, some benchmark matrices will be generated according to a GPU's architecture features and four different sparse matrix storage formats, then SpMV with these benchmark matrices are implemented on the GPU to obtain the execution times. We will establish two parametric models according to the results of the benchmark matrices and predict the execution time of the SpMV kernels with a given target matrix on the GPU by our models in second phase.

The performance prediction models are essentially at the statistical point of view to predict the execution time of different SpMV kernels on GPUs. Firstly, the execution time of the benchmark matrices with different parameters is required, and then fitting the prediction functions according to the execution time of the benchmark matrices and two parameters. Finally, the estimated execution time of a target matrix will be got after putting two parameters into the prediction functions.

Compared with [16]'s model, the important difference of this paper is generating benchmark matrices. In [16], the number of non-zero elements per row (P_{NZ}) of the benchmark matrices is set to a fixed value. While in many practical problems, such as in computer science or mathematics, there are many matrices are in irregular structure, which P_{NZ} is different largely. So if P_{NZ} of benchmark matrices generated is fixed, the performance prediction model will go awry to some extent and then the accuracy of prediction will be decreased. Keeping this in mind that we generated two kinds benchmark matrices whose P_{NZ} in the uniform distribution or normal distribution, respectively, and then establish its own performance prediction model. In the numerical experiments, we used our model to estimate the execution time of different SpMV kernels with 30 matrices, these matrices are from the University of Florida Sparse matrix collection [17]. The experimental results show that the average prediction error of our models are two to three times lower than [16] at least, some matrix even tens of times lower.

The remainder of this paper is organized as follows: Section 2 gives some preliminaries and Section 3 shows the details of the performance prediction model. Experimental results and analyses are reported in Section 4. Finally, some conclusions and future works are stated in Section 5.

2. Preliminaries

Firstly, we state in brief the GPU architecture and CUDA (Compute Unified Device Architecture) programming model. Traditionally, GPUs have been especially designed to handle the computation for computer graphics in real-time. Today, they are increasingly being exploited as general-purpose attached processor to speed-up computations in image processing, physical simulations, data mining, linear algebra, etc. [3]. It is suitable for processing massively parallel tasks, which have high density and simple branching logic. CUDA introduced by NVIDIA is similar in style to a single program multiple data (SIMD) software model [18]. CUDA programs on the host (CPU) invoke a kernel which runs on the device (GPU). All threads within a block are executed concurrently on a architecture [14]. In addition, when a multiprocessor is given one or more thread blocks to execute, it partitions them into groups of 32 parallel threads termed *warp*.

Four sparse matrix storage formats used in our model are described below. The CSR is probably the most popular format for storing general sparse matrices [19]. To parallelize the SpMV for a matrix in the CSR format, a simple scheme called CSR-S (CSR *scalar*) kernel [2] is to assign each thread by one row. The main drawback of this scheme is that the pattern of memory access is un-coalesced, so it shows not very efficient. A better approach, termed CSR-V (CSR *vector*) is proposed in [2] and modified in [3] to realize the memory access contiguously. [2] assign a warp (32 threads) to each row, while [3] assign each row a half-warp (16 threads). In this approach, since all threads within a warp or a half-warp access non-zero elements of one row, these accesses are more likely to belong to the same memory segment, so the chance of coalescing should be higher. In addition, CSR-V kernel in [3] will be used in our model. ELL format is efficient if the maximum number of non-zeros per row is not substantially different from the average. However, when the number of non-zeros varies largely between rows, excess padded zeros decrease the performance of SpMV. The JAD can be viewed as a generalization of ELL format which removes the assumption on the fixed-length rows [3]. Compared to the ELL format, there are fewer zero-padding in the JAD format, so the performance of JAD will be better than ELL when implement SpMV on GPUs. Other details of these sparse matrix storage formats can be consulted in [19].

3. The Performance Prediction Model for SpMV

The work-flow of our model is similar to [16], which contains two phases. The main difference is the criteria for generating the benchmark matrices, which we described in follows.

3.1. Phase One: The Establishment of Fitting Function for Performance Prediction

Firstly, we give the definition of matrix strip. The strip of a matrix is a maximum sub-matrix that can be handled by a GPU with a full load of thread blocks within one iteration. Let N_{SM} be the number of streaming multiprocessors for a NVIDIA GPU, N_{HW} be the number of half-warps per multiprocessor and N_T denote the number of threads per multiprocessor. Then, *the size of strip* for CSR-V, CSR-S, ELL and JAD format can be computed as follows:

$$S_{\text{CSR-V}} = N_{SM} \times N_{HW} \tag{1}$$

$$S_{\text{CSR-S}} = N_{SM} \times N_{T} \tag{2}$$

$$S_{\text{ELL}} = S_{\text{JAD}} = S_{\text{CSR-S}} \tag{3}$$

Secondly, we state the criteria for generating benchmark matrices.

- The number of rows (R):

$$R = S \times I \tag{4}$$

where I is a positive integer, S is the size of strip defined for four formats (in different sub-index) as above.

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

In [16]'s model, each row has the same P_{NZ} for the benchmark matrices. However, P_{NZ} will not be same exactly for a target matrix in practical situations. So we combine with the characteristics of the matrix structure and let P_{NZ} to meet a certain distribution. Two distributions will be adopted: normal and uniform. In addition, the mean of the distribution will be treated as P_{NZ}. The benchmark matrices meet two kinds of distribution can be generated by Matlab. In addition, we assume that then on-zero elements are in single precision (*float*).

- The number of columns (C):

For the sake of simplicity, the benchmark matrices generated in our numerical experiments will be square. Obviously, it should be assumed that $C > P_{NZ}$.

Thirdly, we set parameters of benchmark matrices.

In order to get more accurate fitting functions in our models, a series of benchmark matrices will be generated according to the above criteria. A benchmark matrix is only determined by R and P_{NZ}. Since $R = S \times I$, where S is fixed for a certain sparse matrix format, we just need change the value of I to get different benchmark matrices. Due to P_{NZ} in the benchmark matrices follows two kinds of distributions, so it is determined by the mean of each distribution based on the distribution density P. Then combine the value of I and P, we can obtain a benchmark matrix.

- The number of strips (I):

➢ CSR-V: Let $I = 1,2,3,\cdots,9,10,15,20,25,\cdots,45,50$

In our experimental platform, the size of matrix strip for CSR-V format is fewer than other formats, in order to predict the performance accurately, we increase the value of I to 50.

➢ CSR-S, ELL, JAD: Let $I = 1,2,3,\cdots,9,10$

- The distribution density (P):

➢ CSR-V: Let $P = \frac{4}{R}, \frac{8}{R}, \frac{16}{R}, \cdots, \frac{512}{R}, \frac{1024}{R}, \frac{1536}{R}, \frac{2048}{R}, \frac{2560}{R}, \frac{3072}{R}$

Because of *out of memory* occurred in Matlab when P is too large, so we make $3072/R$ as the maximum value, but it does not affect the accuracy of the performance prediction.

➢ CSR-S, ELL, JAD: Let $P = \frac{4}{R}, \frac{8}{R}, \frac{16}{R}, \cdots, \frac{512}{R}, \frac{1024}{R}$

Finally, the formula for calculating average the execution time of benchmark matrices T_B is same as that of [16].

$$T_B = \frac{\sum_{j=1}^{\beta} \phi\left(\left(M_{R\times C}\right)\times V_C\right) - \sum_{j=1}^{\alpha} \phi\left(\left(M_{R\times C}\right)\times V_C\right)}{\beta - \alpha} \tag{5}$$

where $M_{R\times C}$ denotes a benchmark matrix of dimension $R \times C$; V_C is a random vector of length C; α and β are the number of executions and $\alpha < \beta$. ϕ is the execution time for each time the benchmark matrix be executed. For a target matrix with N_R rows and N_{NZ} non-zero elements, the number of strips I and the number of non-zero elements per row P_{NZ} with four formats can be compute as follows:

$$I_{\text{CSR-V}} = \left\lceil \frac{N_R}{S_{\text{CSR-V}}} \right\rceil,\ I_{\text{CSR-S}} = \left\lceil \frac{N_R}{S_{\text{CSR-S}}} \right\rceil,\ I_{\text{ELL}} = \left\lceil \frac{N_R}{S_{\text{ELL}}} \right\rceil,\ I_{\text{JAD}} = \left\lceil \frac{N_R}{S_{\text{JAD}}} \right\rceil \tag{6}$$

Let D be the set consisting of the number of non-zero elements in each row of the target matrix. Then P_{NZ} is set to be mode of D for CSR-V matrix, while it is the maximum value of D for CSR-S, ELL, JAD matrices.

3.2. Phase Two: Prediction Based on the Performance Model

According to the statistics methods, we fit the performance function of SpMV for different storage formats, which based on three parameters of benchmark matrices: I, P_{NZ} and T_B. After the performance function obtained, we can estimate the execution time of SpMV for a target matrix T_T by substituting two parameters I and P_{NZ} of the target matrix into it.

3.2.1. CSR-V *Format*

After a large amount of experiments and fitting, we found that for CSR-V matrices, the relationship between T_B and P_{NZ} is different when P_{NZ} is smaller or larger than the number of maximum threads per block (1024 for GeForce GTX 540 M). Therefore, the performance fitting function is obtained by the following method.

- Establish the function $T(P_{NZ})$

For the benchmark matrices with the same number of strips, we establish the relationship between P_{NZ} and the execution time T_B for SpMV. The fitting functions of two distributions for the number of strips 40 are shown in Figure 1. As can be seen, the relationship between P_{NZ} and T_B is approximately linear, so we make $T(P_{NZ}) = m \times P_{NZ} + n$.

- Establish the function $E(I)$

For the benchmark matrices with same P_{NZ}, we establish the relationship between I and the execution time E (*i.e.* T_B in the above) of the benchmark matrices for SpMV. The fitting functions of two distributions for $P_{NZ} = 64$ and 2048 are shown in Figure 2. The relation between I and E is also approximately linear, so we make $E(I) = p \times I + q$.

- Estimate the execution time of a target matrix

For a target matrix, we need to calculate two parameters according to the Equation (6) and D: the number of non-zero elements per row P_0 and the number of strips I_0, then derive $T(P_0)$ and $E(I_0)$ from above functions, respectively. In order to combat the effects of the difference about functions when the number of non-zero elements per row is smaller or larger than 1024, another execution time t_0 of any previously tested benchmark matrix whose P_{NZ} is set to be the number of non-zero elements per row in $E(I)$. At this point, estimated execution time of the target matrix in CSR-V format is

$$T_0 = \frac{T(P_0)}{t_0} \times E(I_0).$$

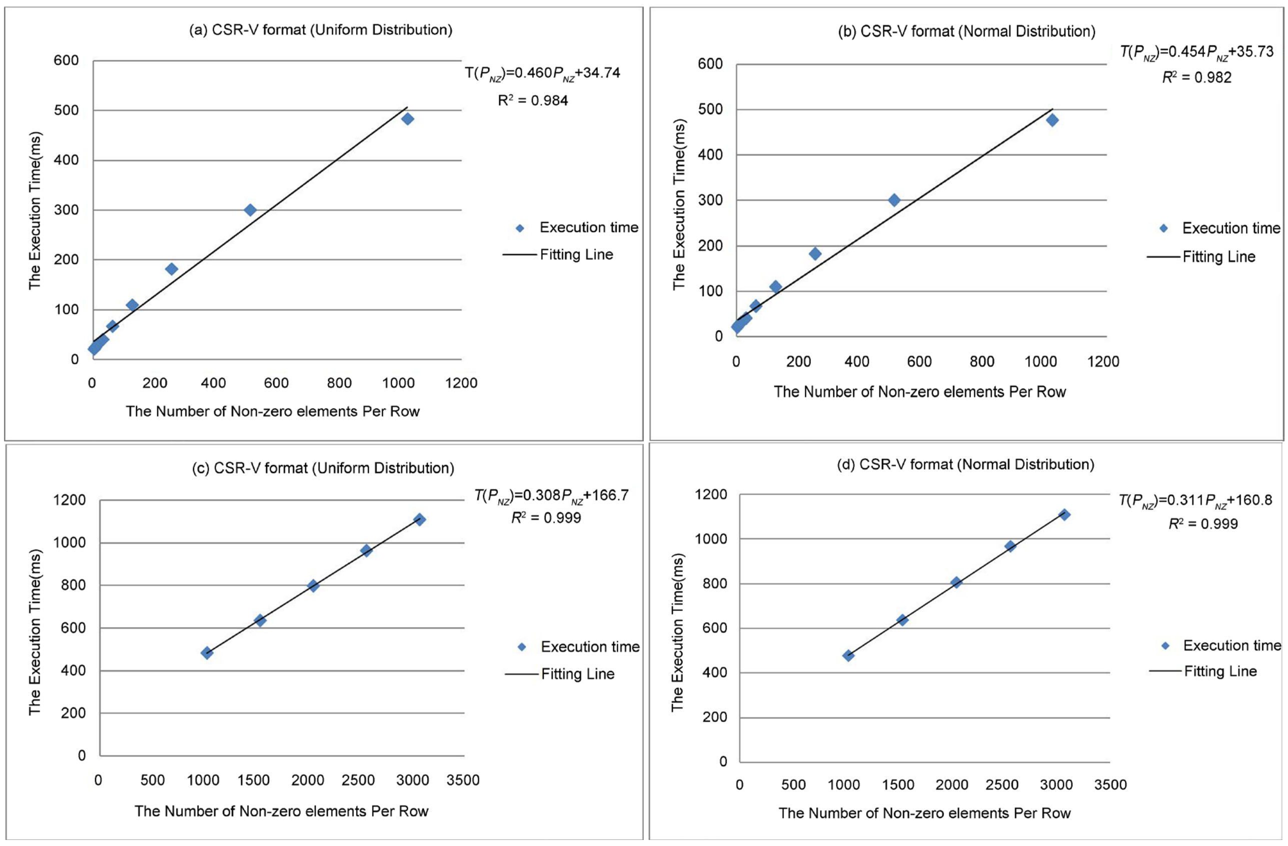


Figure 1. The fitting functions of two distributions for $I = 40$ ((a) and (b): $P_{NZ} < 1024$, (c) and (d): $P_{NZ} > 1024$).

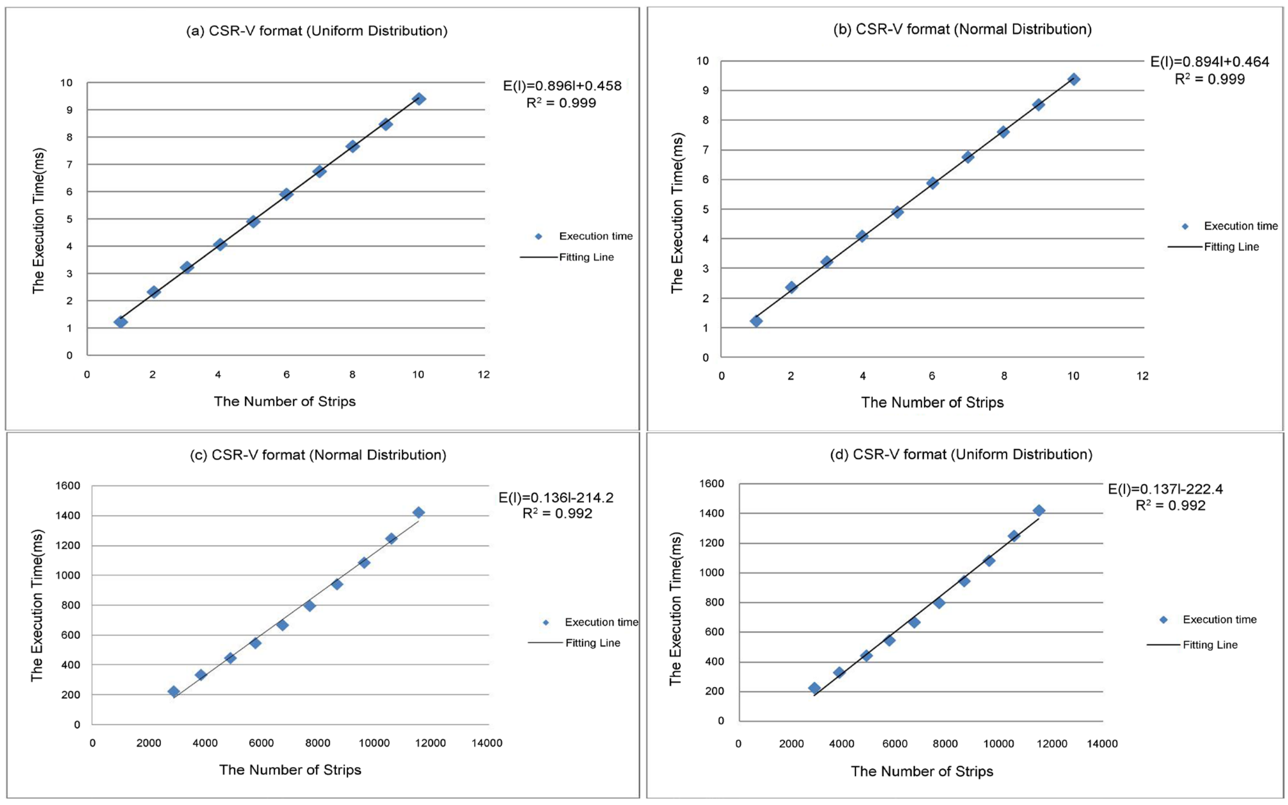


Figure 2. The fitting functions of two distributions for $P_{NZ} = 64$ and 2048 ((a) and (b): $P_{NZ} = 64$, (c) and (d): $P_{NZ} = 2048$).

3.2.2. CSR-S *Format*

- Establish the function $T(P_{NZ})$

For the benchmark matrices with the same I, we establish the relationship between P_{NZ} and T_B for SpMV. The fitting functions of two distributions for $I=5$ are shown in Figure 3. As showed in the Figure, the relationship between P_{NZ} and T_B is approximately linear, so we make $T(P_{NZ})=f(I_1)\times P_{NZ}+g(I_1)$ (I_1 can be any arbitrary value within the range of I).

- Establish the function $f(I)$

For sets of benchmark matrices with different number of strips, we establish the relationship between the number of strips I and the coefficient of the linear functions $f(I)$ in $T(P_{NZ})$. The fitting functions of two distribution are shown in Figure 4. In Figure 4, it is approximately linear between I and $f(I)$, so we make $f(I)=A\times I+B$.

- Establish the function $E(I)=f(I)\times P_1+g(I)$

For Like the fitting function $T(P_{NZ})$, we establish the relationship between the number of strips I and the execution time E (*i.e.* T_B in the above) of the benchmark matrices with the same number of non-zero elements per row

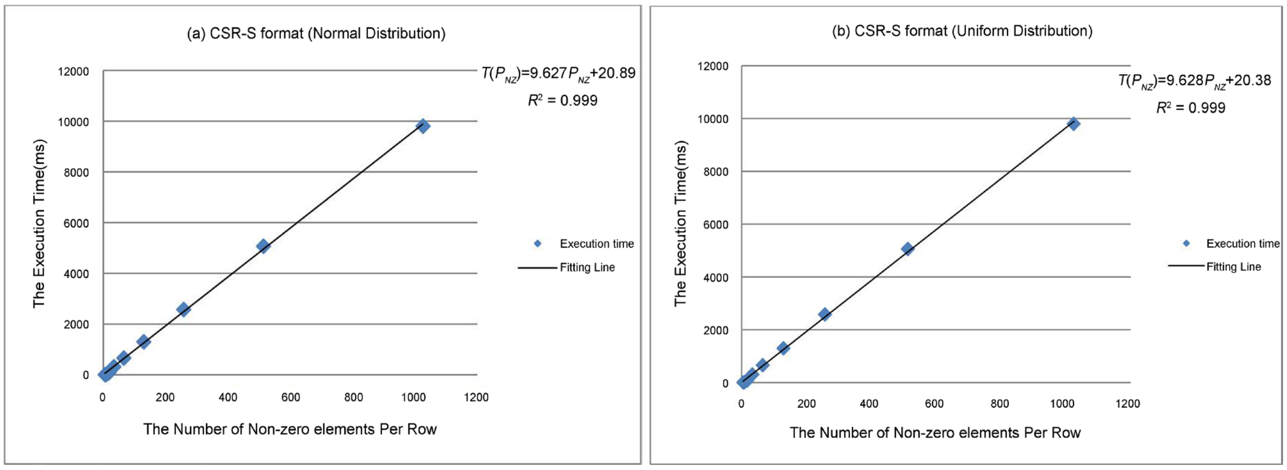


Figure 3. The fitting functions of two distributions for $I=5$.

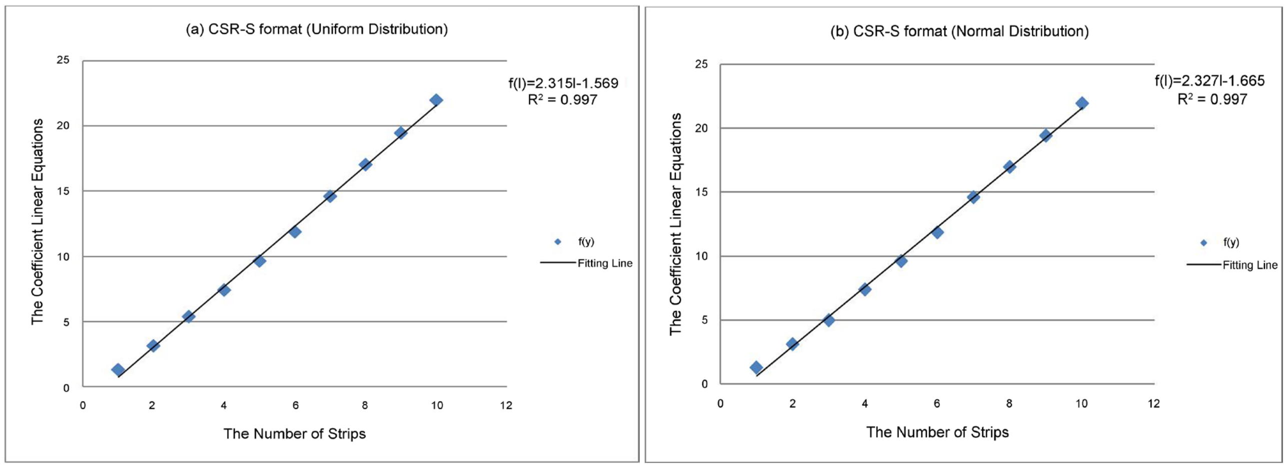


Figure 4. The fitting functions for I versus $f(I)$.

P_1, which can be any arbitrary value within the range defined P_{NZ}. The fitting functions of two distributions for $P_{NZ}=32$ are shown in Figure 5. We can see that the relationship between I and E is approximately linear, so we make $E(I)=f(I)\times P_1+g(I)$, and the intercept function $g(I)=E(I)-f(I)\times P_1$.

- Estimate the execution time of a target matrix

Given a target matrix, we need to calculate two parameters according to the Equation (6) and D: the number of non-zero elements per row P_0 and the number of strips I_0, then derive $f(I_0)$ and $g(I_0)$ from above functions, respectively. At this moment, estimated execution time of the target matrix in CSR-S format is $T(P_0)=f(I_0)\times P_0+g(I_0)$.

After getting the performance function of CSR-S format, we find that the relationship between dependent variables T_B and two variables P_{NZ}, I is saddle surface in the functional image, that is to say, when I is fixed, the relationship between T_B and P_{NZ} is linearity and vice versa, which coincided with the 3D fitting image we get in Matlab, as shown in Figure 6, which the darker of the colors means the smaller of the values.

3.2.3. ELL and JAD *Formats*

Note that, the granularity of ELL and JAD format is the same as CSR-S format, which assigns one thread to each row to implement SpMV on GPUs. Therefore, fitting the performance function of ELL and JAD format is done in a similar way with CSR-S format, except the functional expressions. In addition, the 3D images obtained by the Matlab can be fitted with the performance functions and need not be repeated here.

4. Experimental Results and Analyses

The experiments are performed on NVIDIA GeForce GTX 540 M with 1 GB global memory, the operating system is a 64-bit Linux with CUDA 6.5 driver. We evaluated our performance prediction model on 30 matrices with each sparse matrix storage format, respectively. These matrices are square real matrices from the University of Florida Sparse matrix collection [17]. Moreover,

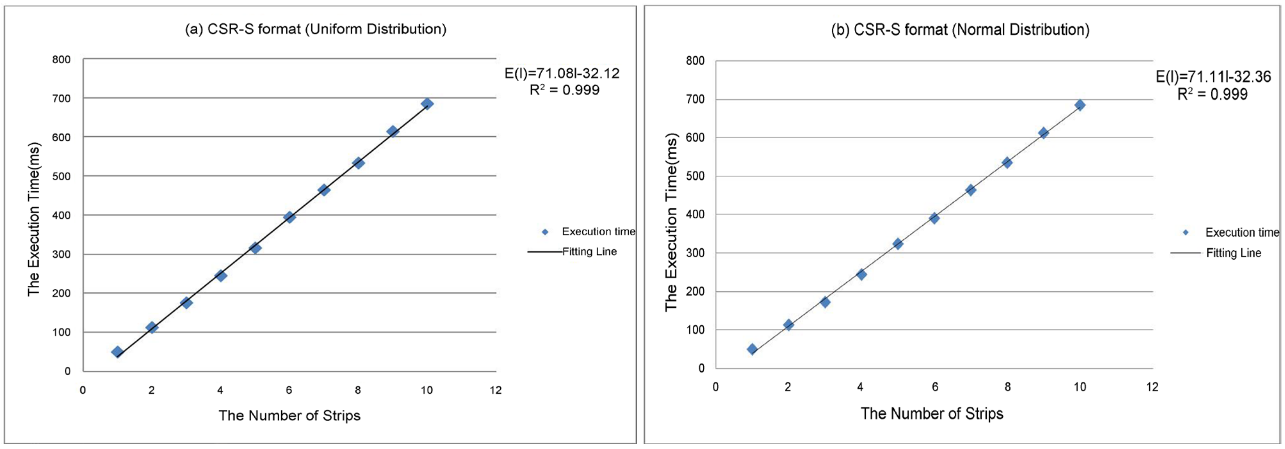
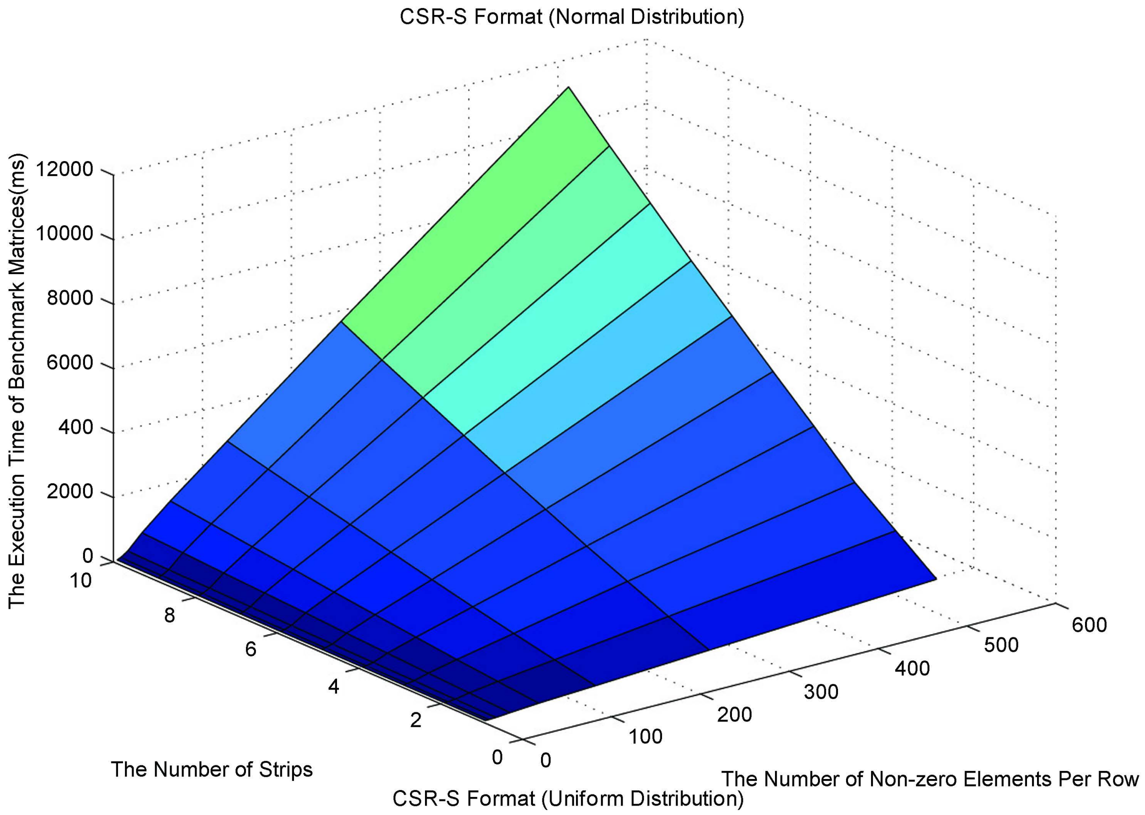


Figure 5. The fitting functions of two distributions for $P_{NZ}=32$.



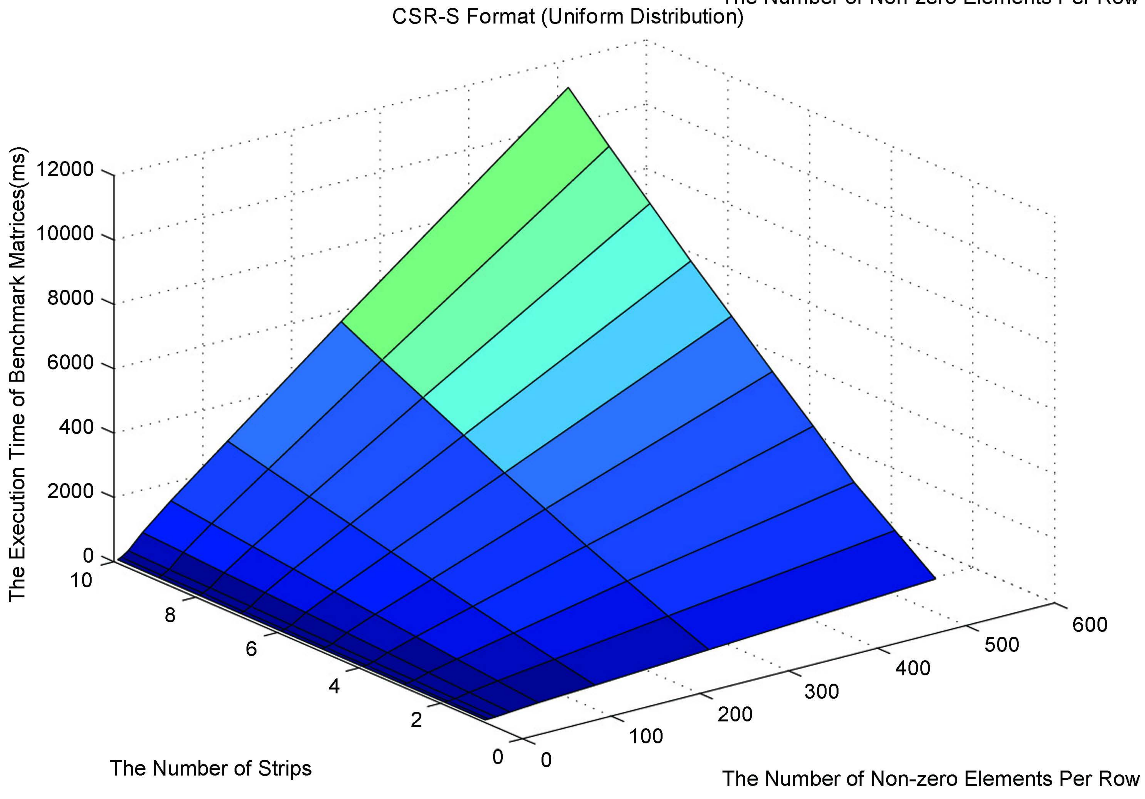


Figure 6. The 3D fitting images of two distributions for T_B.

in order to compare with [16]'s model, we also implement the model in [16] whose P_{NZ} is fixed in benchmark matrices. The estimated time and the performance difference rate of the 30 target matrices in four sparse matrix storage formats with three models are given, which is convenient to compare with each other.

We define the performance difference rate for different model as

$$D_r = \frac{|\text{estimated time} - \text{mesured time}|}{\text{mesured time}} \tag{7}$$

For CSR-V format, the performance difference rate of SpMV in three models of 30 matrices is shown in **Figure 7**. We can see that D_r in [16]'s model is 5.05% on average, while it is 2.42% and 2.56% for our normal and uniform model, respectively. Compared with the [16]'s model, the prediction accuracy of the normal model and uniform model are improved by 2.08 times and 1.97 times on average, respectively.

Furthermore, there are three cases for the prediction accuracy of [16]'s model is higher than that of uniform model (such **cavity05**), and that of normal model (such as **bcsstk04**). The prediction accuracy of normal distribution model is higher than uniform distribution model for **bp_1600** and other 17 matrices.

When implement SpMV on GPU with CSR-S format, the performance difference rate in three models of 30 matrices is shown in **Figure 8**. The average D_r in [16]'s model, our uniform and normal model is 5.44%, 3.21% and 3.52%, respectively. The average factor for the prediction accuracy of uniform model higher than that of [16]'s model is 1.69, while for normal model, it is 1.54.

In addition, the prediction accuracy of [16]'s model is higher than that of normal model on 15 matrices (such as **bcsstk16**), while for uniform model the better number is 12 (such as **bips98_1142**). The better number for normal model vs. uniform model is 13 (such as **bayer09**) in 30 cases.

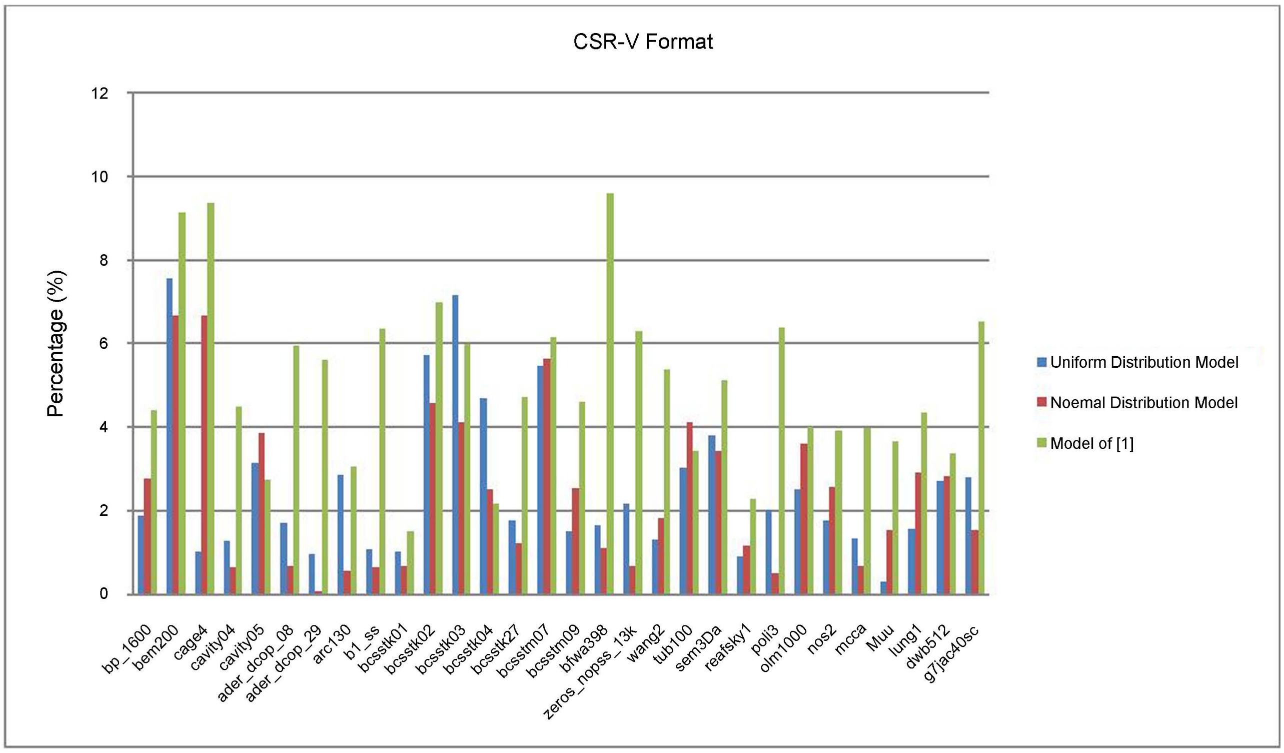


Figure 7. The performance difference rate of SpMV on CSR-V matrices.

The similar results for ELL matrices are given in Figure 9. It says the average D_r of three models are 5.79%, 3.53% and 3.26%, respectively. The average

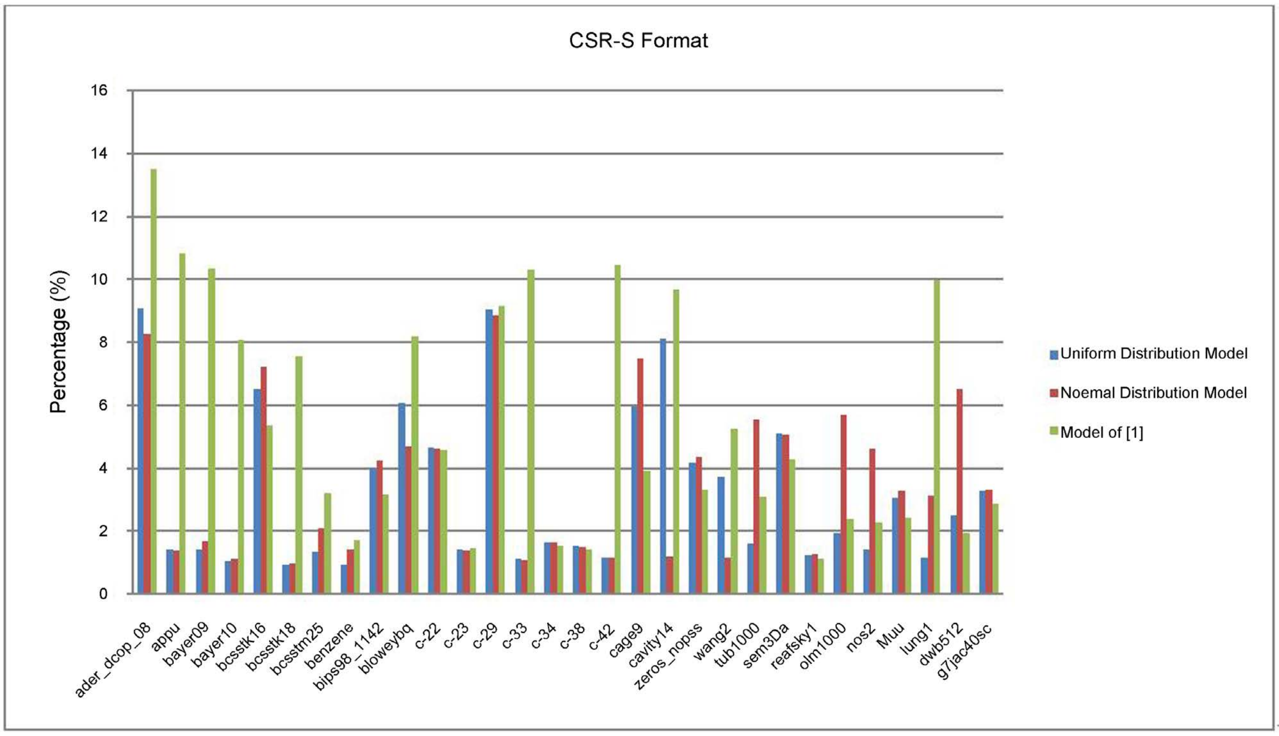


Figure 8. The performance difference rate of SpMV on CSR-S matrices.

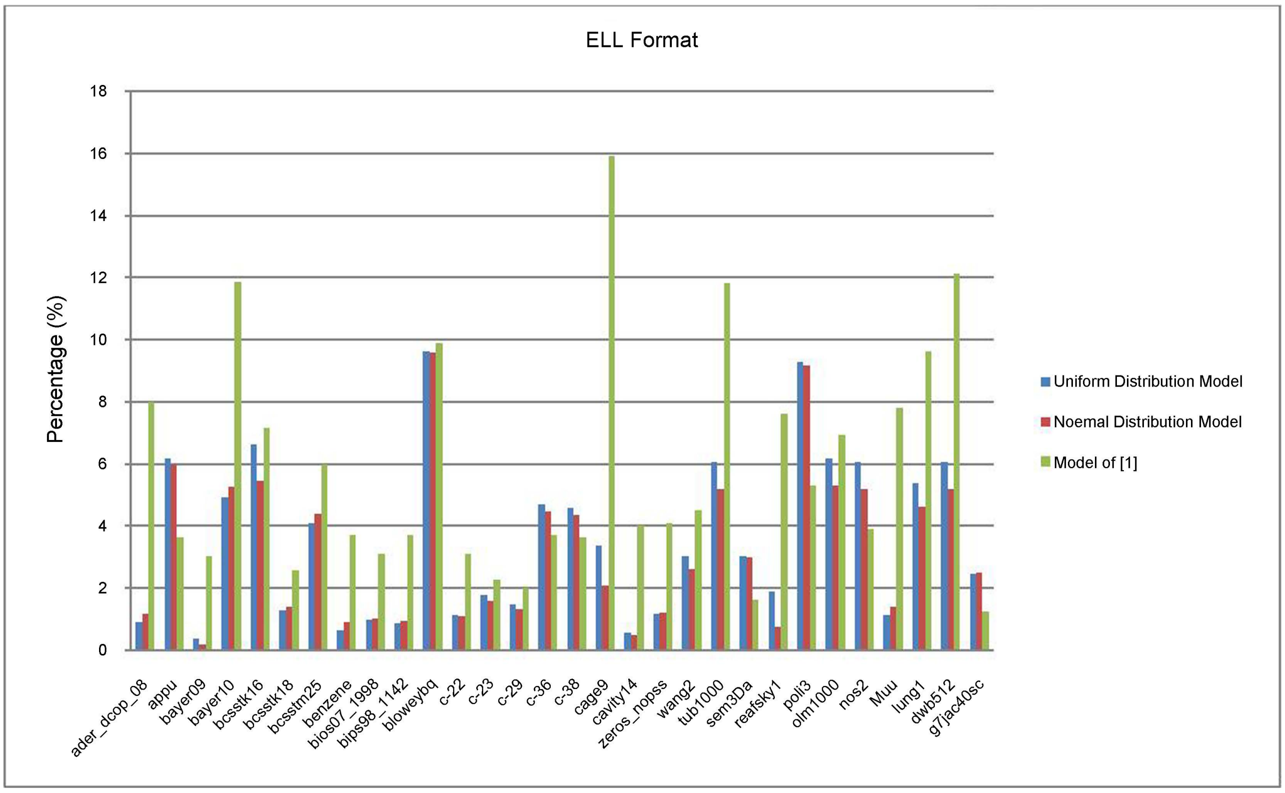


Figure 9. The performance difference rate of SpMV on ELL matrices.

better number for the factor of normal and uniform model vs. [16]'s model are 1.77, 1.64 respectively. The better number for [16]'s model vs. normal model and uniform model is the same 7:23, while for normal model vs. uniform model it is 20:10.

Figure 10 give the results for JAD matrices. D_r of three models are 5.97%, 3.94% and 4.28% on average, respectively. The average improved factor for normal and uniform model over [16]'s model are 1.46 and 1.35. The better number for [16]'s model vs. normal model is 9:21, while for uniform model it is 11:19, while for normal model vs. uniform model it is 16:14.

The execution time of four SpMV kernels in three model on 30 matrices is shown in **Figures 11-14**. There is large difference in the execution time for all matrices in different storage formats. So we put the execution time into two figures: the shorter and the longer in (a) and (b), respectively. Almost all of the estimated time of the matrices in four different storage formats is greater than the actual measured time. The possible reasons are that we take the number of strips I by rounding up to an integer.

5. Conclusion and Future Work

Aiming at the better performance model of SpMV on GPU based on statistics and [16], we have presented two new models, which consider the structure of matrices. We predict four SpMV CUDA kernels: CSR-V, CSR-S, ELL and JAD. The numerical result shows that the prediction accuracy of our models is higher than that of [16]'s model.

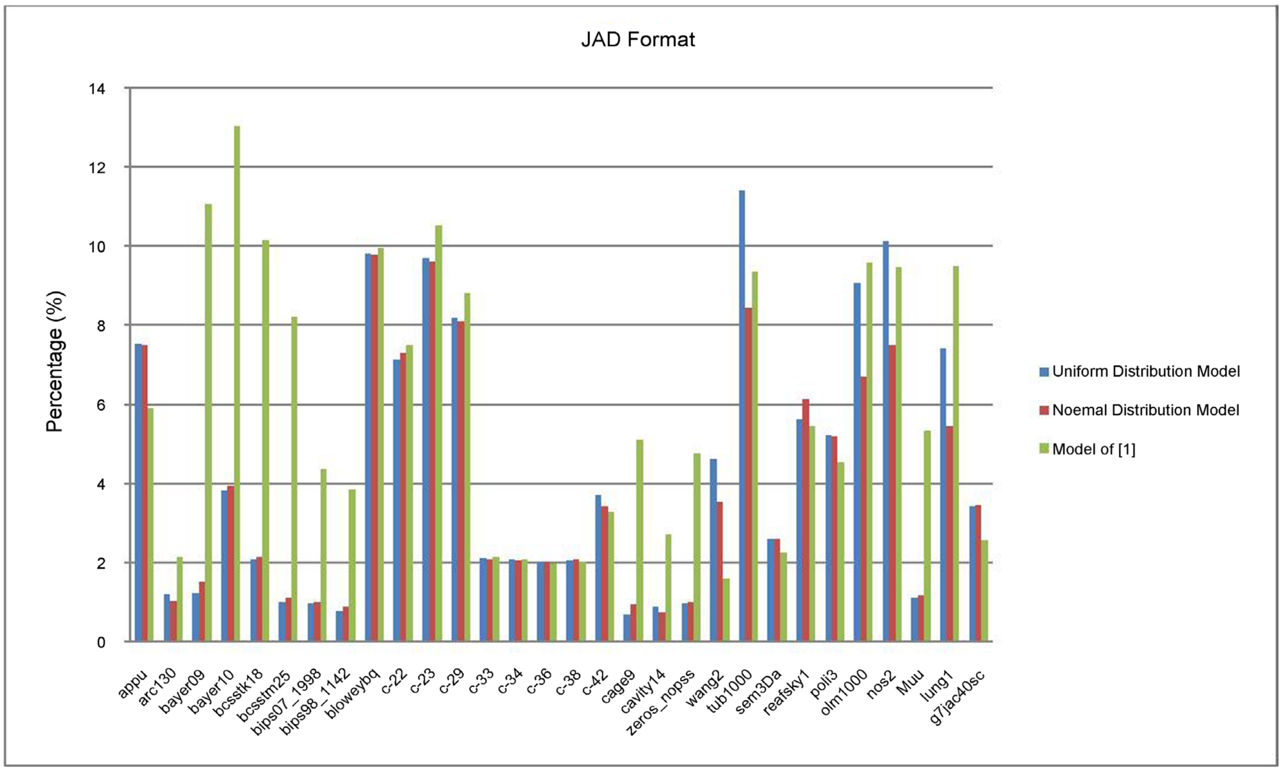
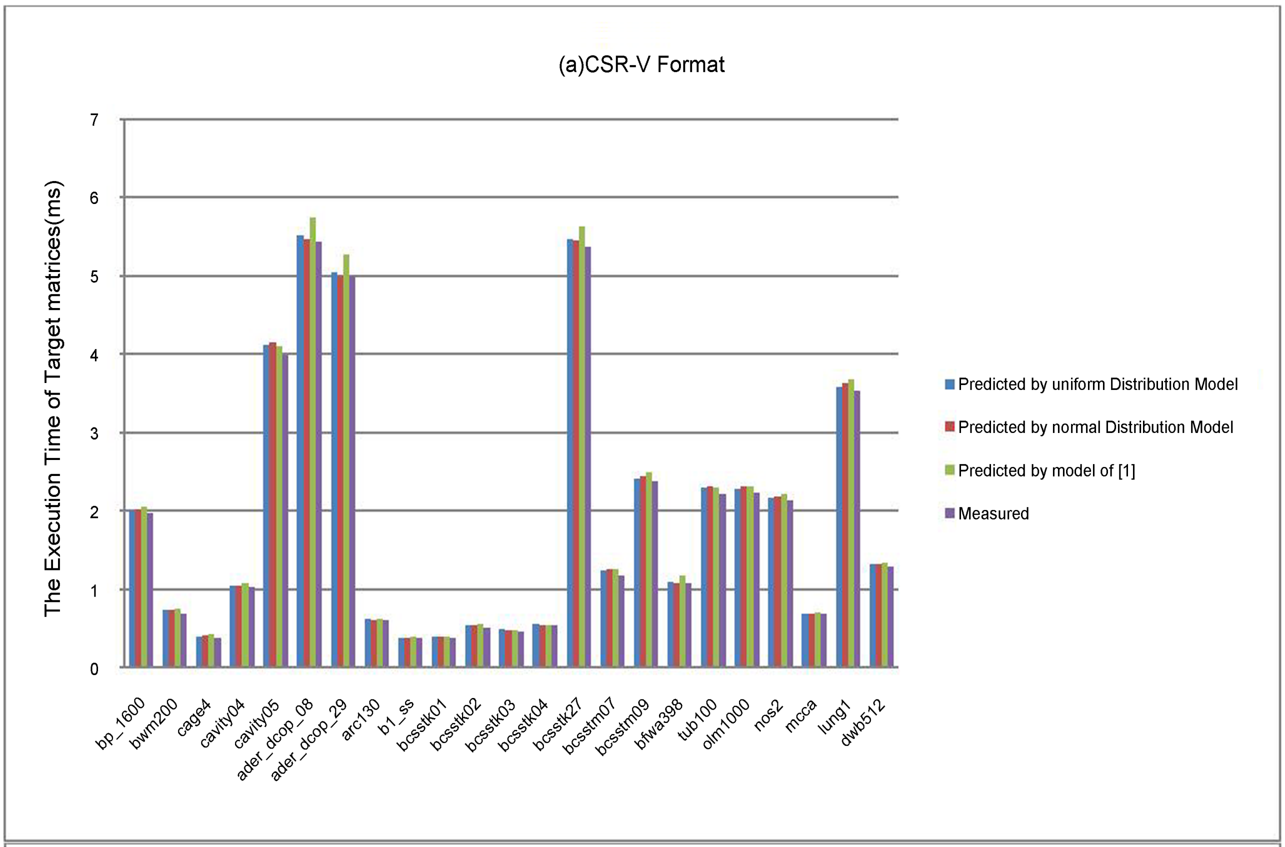


Figure 10. The performance difference rate of SpMV on JAD matrices.



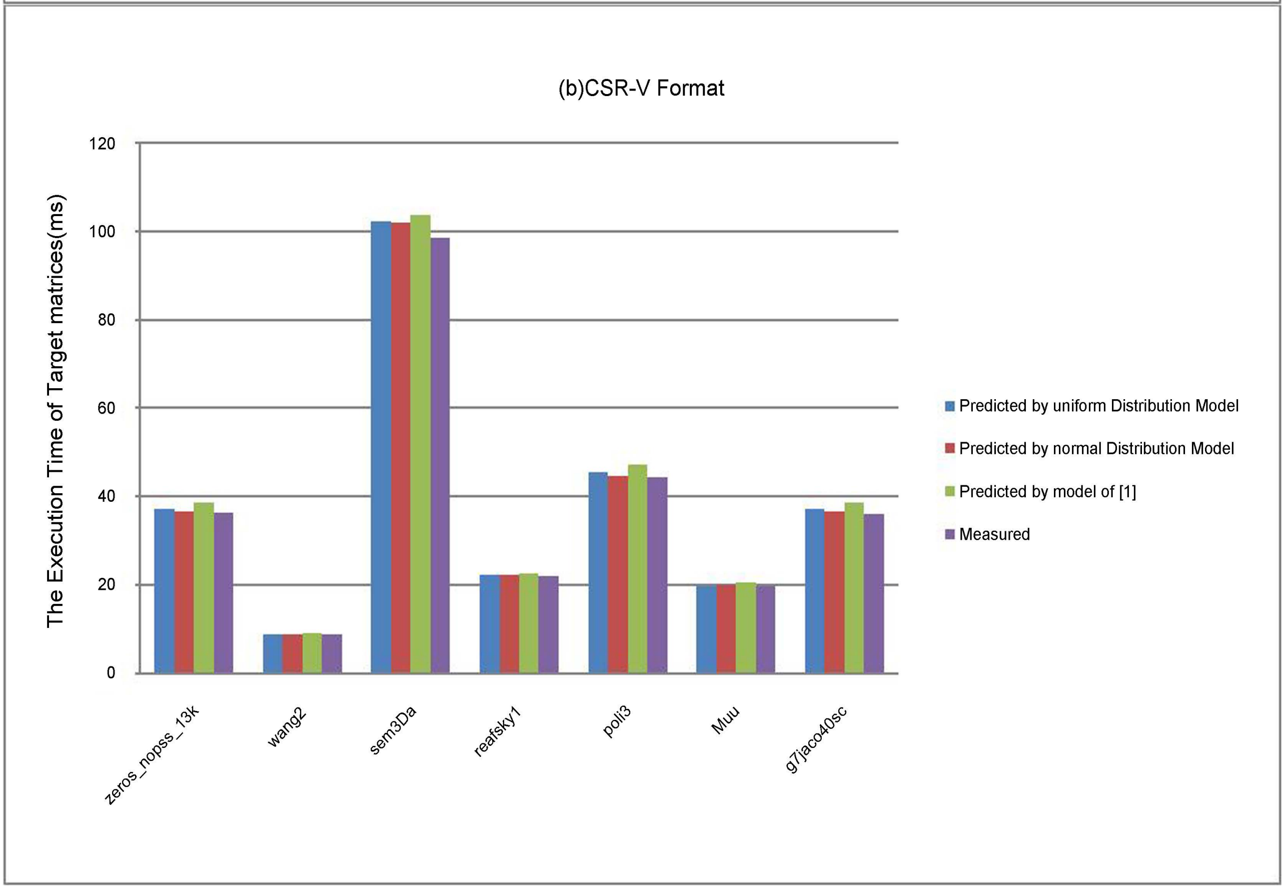


Figure 11. The comparison of estimated and measured time on CSR-V matrices.

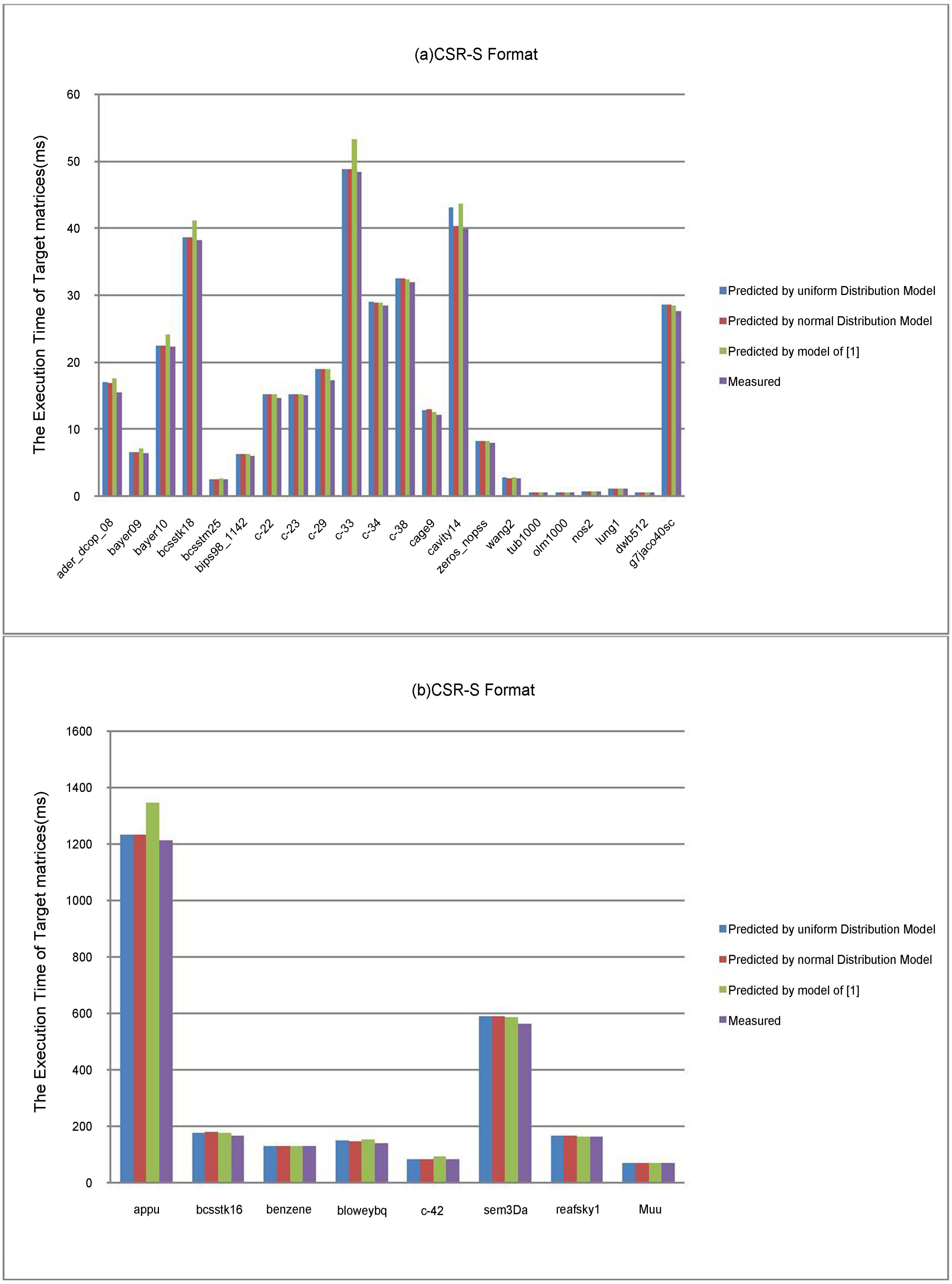
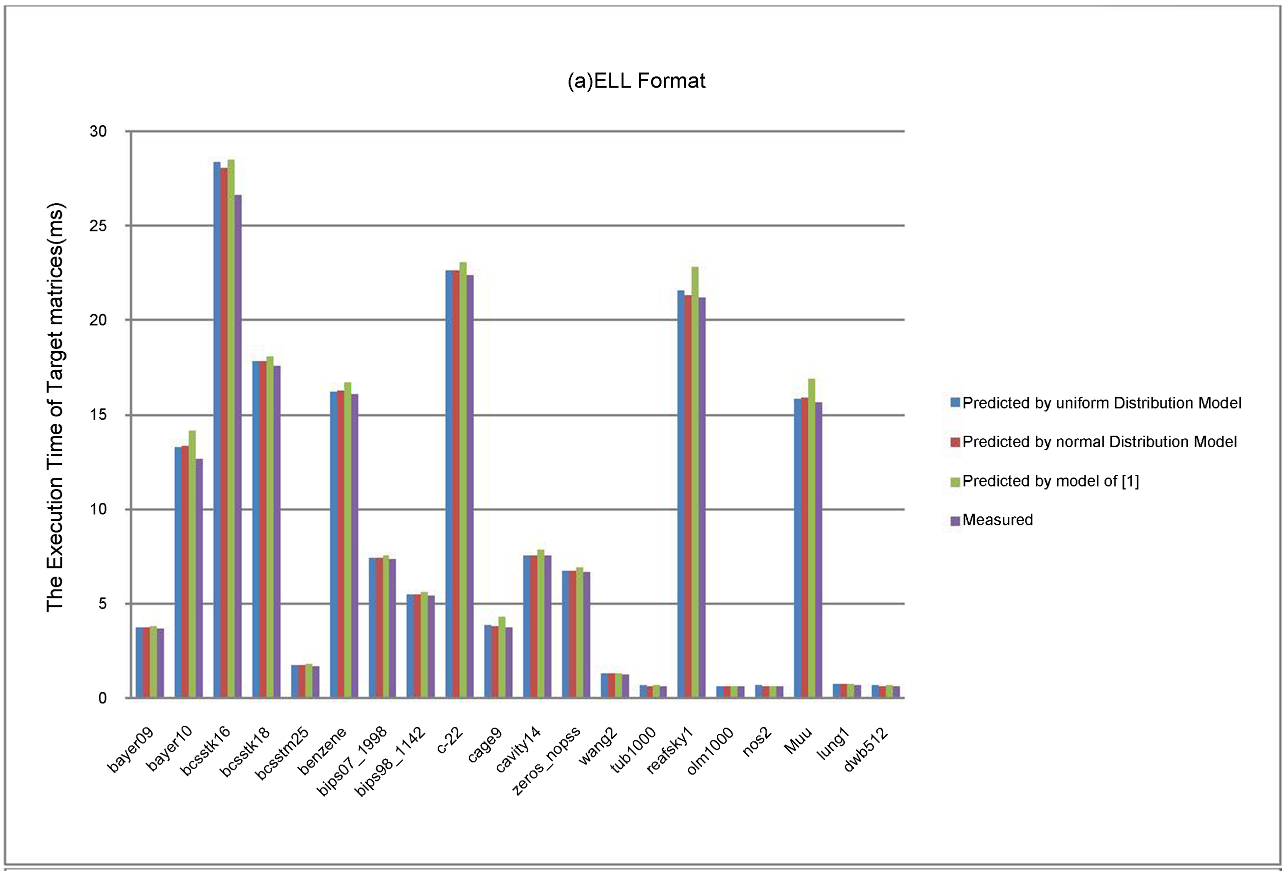


Figure 12. The comparison of estimated and measured time on CSR-S matrices.



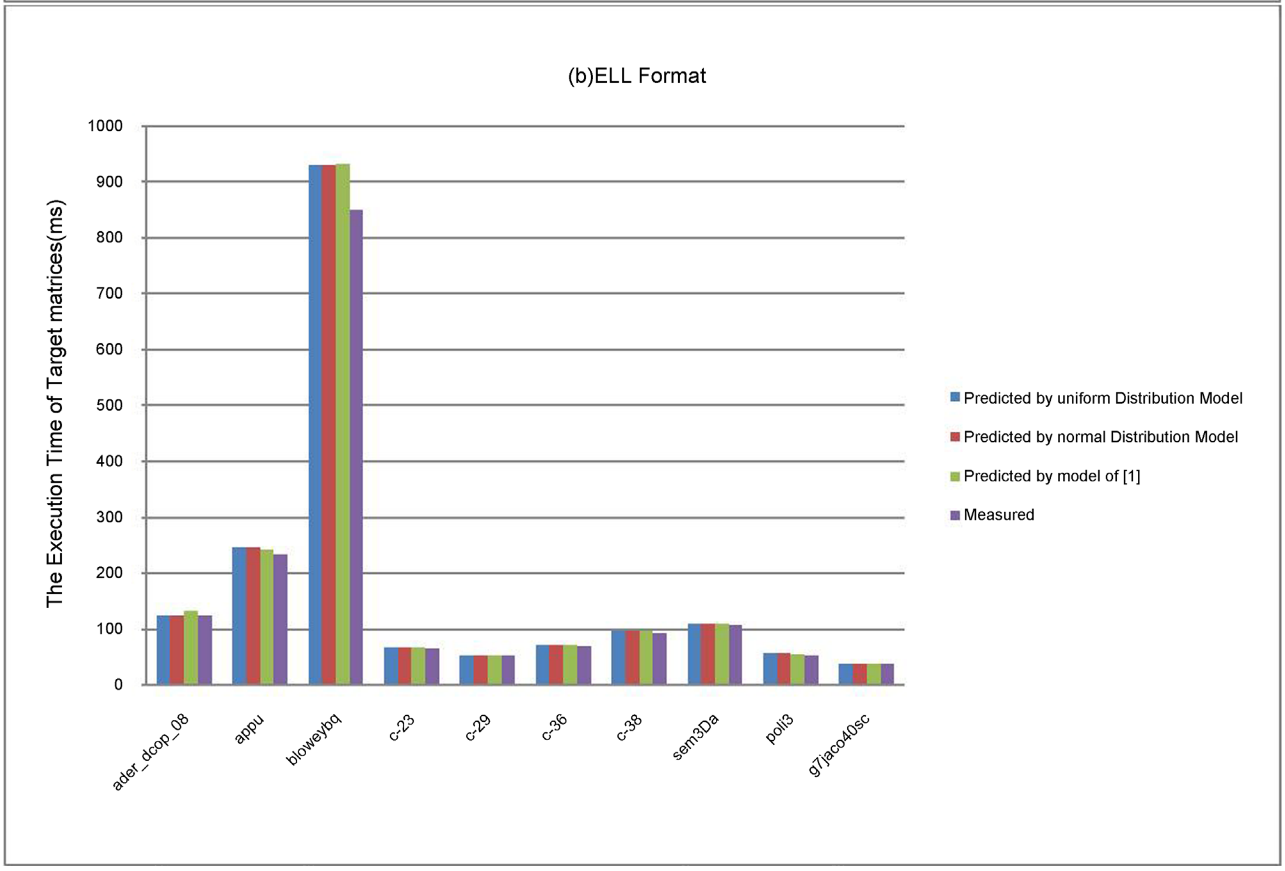
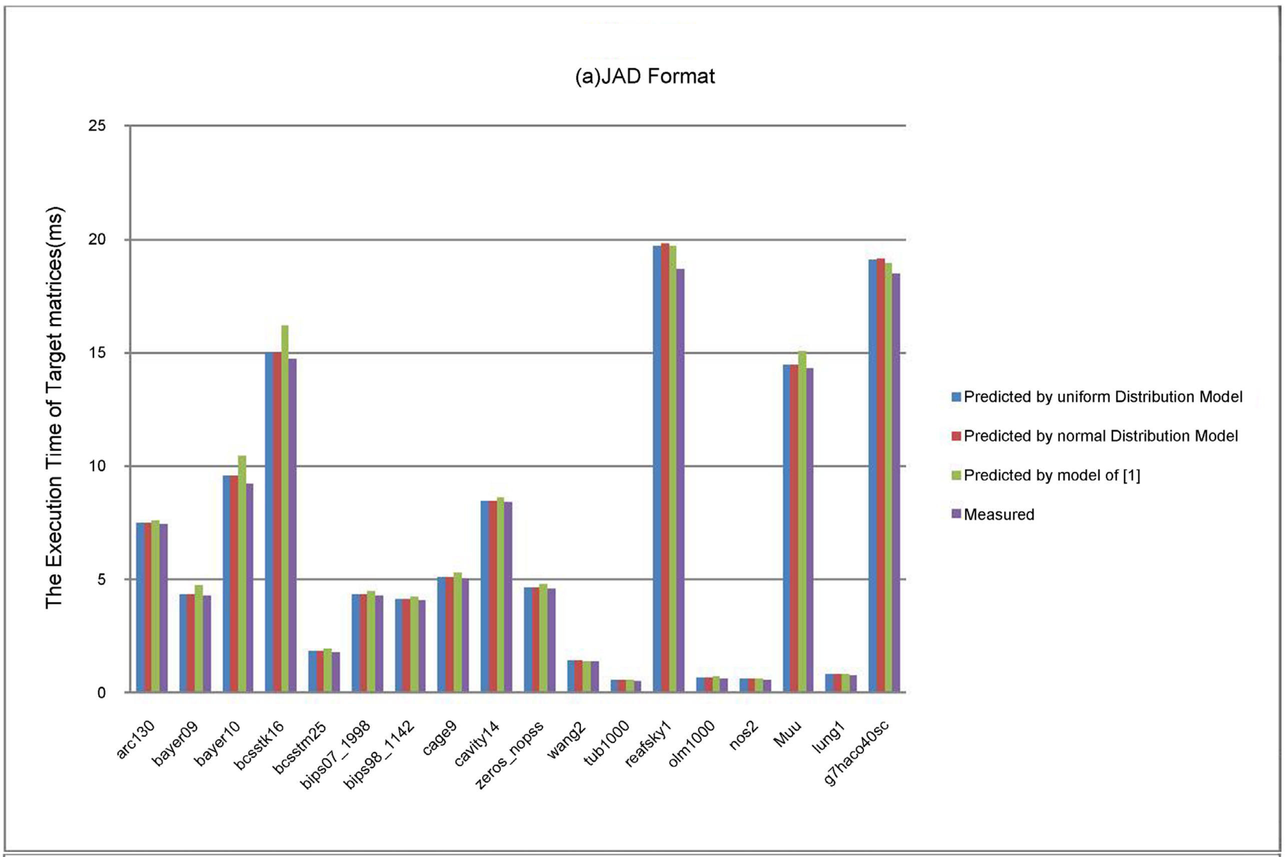


Figure 13. The comparison of estimated and measured time on ELL matrices.



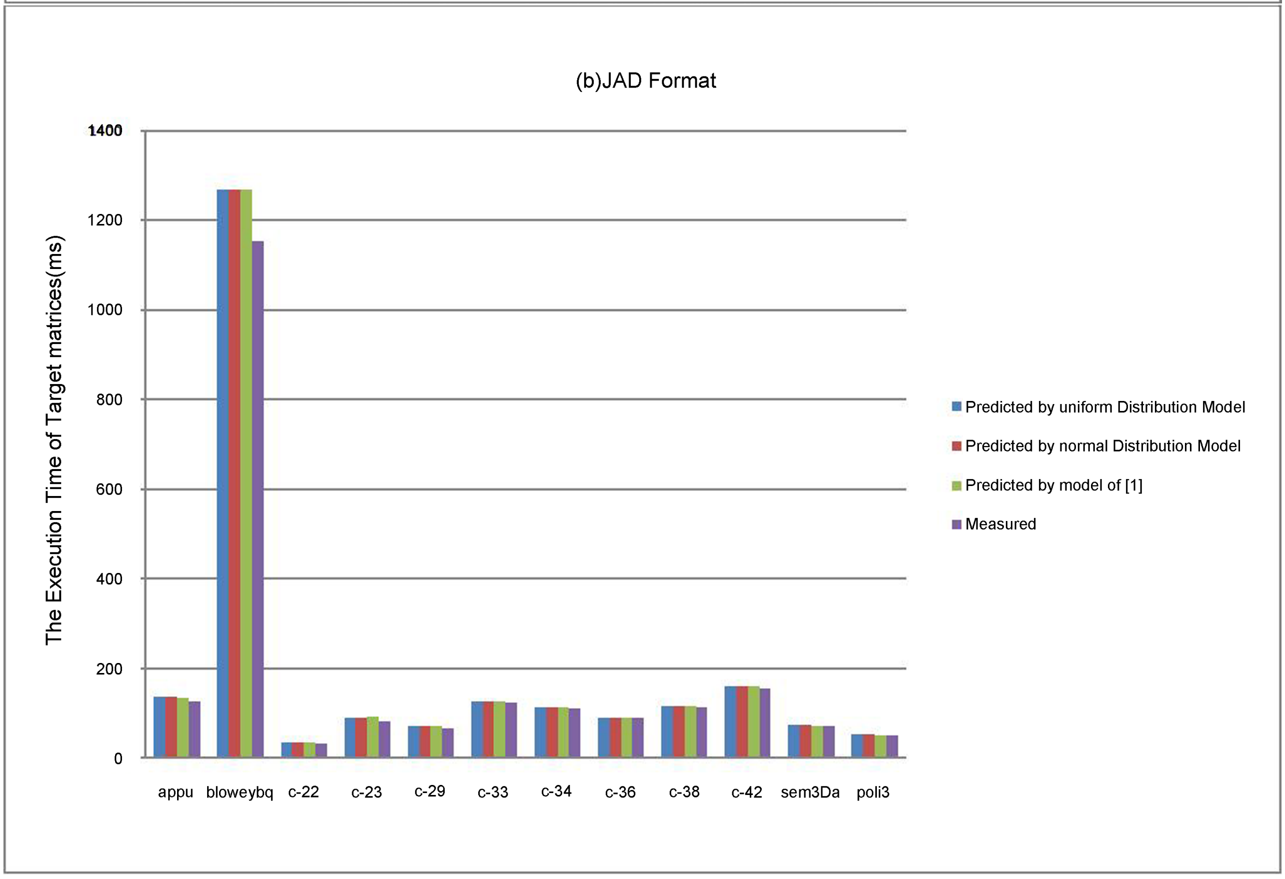


Figure 14. The comparison of estimated and measured time on JAD matrices.

In the future, we will extend our performance prediction model to other SpMV with different storage formats on different kinds of GPUs. In addition, we will propose a new performance model to predict the execution time of a class of iterative methods on heterogeneous parallel machines.

References

[1] Bolz, J., Farmer, I., Grinspun, E. and Schroder, P. (2005) Sparse Matrix Solvers on the GPU: Conjugate Gradients and Multigrid. *ACM Transactions on Graphics*, **22**, 917-924.

[2] Bell, N. and Garland, M. (2009) Implementing Sparse Matrix-Vector Multiplication on Throughput-Oriented Processors. *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, Portland, 14-20 November 2009, Article No. 18. https://doi.org/10.1145/1654059.1654078

[3] Li, R.-P. and Saad, Y. (2013) GPU-Accelerated Preconditioned Iterative Linear Solvers. *The Journal of Supercomputing*, **63**, 443-466. https://doi.org/10.1007/s11227-012-0825-3

[4] Vazquez, F., Ortega, G., Fernandez, J.J. and Garzon, E.M. (2010) Improving the Performance of the Sparse Matrix Vector Product with GPUs. *Proceedings of the 10th IEEE International Conference on Computer and Information Technology*, IEEE Computer Society, Bradford, 29 June-1 July 2010, 1146-1151.

[5] Monakov, A., Lokhmotov, A. and Avetisyan, A. (2010) Automatically Tuning Sparse Matrix-Vector Multiplication for GPU Architectures. In: Patt, Y.N., Foglia, P., Duesterwald, E., Faraboschi, P. and Martorell, X., Eds., *High Performance Embedded Architectures and Compilers. HiPEAC* 2010. *Lecture Notes in Computer Science*, Vol. 5952. Springer, Berlin, Heidelberg, 111-125. https://doi.org/10.1007/978-3-642-11515-8_10

[6] Zheng, C., Gu, S., Gu, T.-X., Yang, B. and Liu, X.-P. (2014) BiELL: A Bisection ELLPACK Based Storage Format for Optimizing SpMV on GPUs. *Journal of Parallel and Distributed Computing*, **74**, 2639-2647. https://doi.org/10.1016/j.jpdc.2014.03.002

[7] Gu, T.-X., Zheng, C., Gu, S. and Liu, X.-P. (2014) Solving Sparse Linear Systems on GPUs Based on the BiELL Storage Format. *Proceedings of International Conference on Parallel, Distributed Systems and Software Engineering*, Singapore, 96-107.

[8] Choi, J.W., Singh, A. and Vuduc, R.W. (2015) Model-Driven Autotuning of Sparse Matrix-Vector Multiply on GPUs. *ACM SIGPLAN Notices*, **45**, 115-126. https://doi.org/10.1145/1837853.1693471

[9] Guo, P., Wang, L.-Q. and Chen, P. (2014) A Performance Modeling and Optimization Analysis Tool for Sparse Matrix Vector Multiplication on GPUs. *IEEE Transactions on Parallel and Distributed Systems*, **25**, 1112-1123. https://doi.org/10.1109/TPDS.2013.123

[10] Resios, A. (2011) GPU Performance Prediction Using Parametrized Models. Master's Thesis, Utrecht University, Utrecht.

[11] Dinkins, S. (2012) 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, Fort Collins.

[12] van Werkhoven, B., Maassen, J., Seinstra, F.J. and Bal, H.E. (2014) Performance Model for CPU-GPU Data Transfers. 14*th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, Chicago, 26-29 May 2014, 11-20.

[13] Baghsorkhi, S.S., Delahaye, M., Gropp, W.D. and Hwu, W.-M.W. (2012) Analytical Performance Prediction for Evaluation and Tuning of GPGPU Applications. *Workshop on Exploiting Parallelism Using GPUs and Other Hardware-Assisted Methods* (*EPHAM*'09), *in conjunction with the* 2009 *International Symposium on Code Generation and Optimization* (*CGO*), Seattle, Washington DC.

[14] Hong, S. and Kim, H. (2009) An Analytical Model for a GPU Architecture with Memory-Level and Thread-Level Parallelism Awareness. *Proceedings of the* 36*th Annual International Symposium on Computer Architecture*, Austin, 20-24 June 2009, 152-163. https://doi.org/10.1145/1555754.1555775

[15] Schaa, D. and Kaeli, D. (2009) Exploring the Multiple-GPU Design Space. *Proceedings of the* 2009 *IEEE International Parallel & Distributed Processing Symposium*, *Rome,* 23-29 May 2009, 1-12. https://doi.org/10.1109/IPDPS.2009.5161068

[16] Guo, P. and Wang, L.-Q. (2012) Accurate CUDA Performance Modeling for Sparse Matrix-Vector Multiplication. *Proceeding of the* 2012 *International Conference on High Performance Computing and Simulation* (*HPCS*), Madrid, 2-6 July 2012, 496-502. https://doi.org/10.1109/HPCSim.2012.6266964

[17] Davis, T. and Hu, Y. (2016) The University of Florida Sparse Matrix Collection. http://www.cise.ufl.edu/research/sparse/matrices

[18] Nvidia Corporation (2011) NVIDIA CUDA Programming Guide. Santa Clara, Nvidia Corporation, USA.

[19] Saad, Y. (2003) Iterative Methods for Sparse Linear Systems. Society for Industrial Applied Mathematics, Philadelphia. https://doi.org/10.1137/1.9780898718003