Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 30 (2014) 66 – 74

Procedia Computer Science

1st International Conference on Data Science, ICDS 2014

A CUDA-enabled Parallel Implementation of Collaborative Filtering

Zhongya Wang[a], Ying Liu[a,b,*], and Pengshan Ma[a]

[a]*School of Computer and Control, University of Chinese Academy of Sciences, Beijing 100190, China*
[b]*Fictitious Economy and Data Science Research Center, Chinese Academy of Sciences, Beijing 100190, China*

Abstract

Collaborative filtering (CF) is one of the essential algorithms in recommendation system. Based on the performance analysis, two computational kernels are identified. In order to accelerate CF on large-scale data, a CUDA-enabled parallel CF approach is proposed where an efficient data partition scheme is proposed as well. Various optimization techniques are also applied to maximize the performance of the GPU. The experimental results demonstrate up to 48× speedup on a single Tesla C2070 graphics card.

Keywords: collaborative filtering; CUDA; recommendation system; parallel computing;

1. Introduction

Collaborative filtering (CF) is a widely used recommendation algorithm in recommendation systems. It looks for users who share the same rating patterns with the active user (the user whom the prediction is for). Then, use the ratings from those like-minded users to calculate a prediction for the active user. In practice, since the number of users and items are huge, the serial method of collaborative filtering algorithms has encountered inevitable performance bottleneck, which makes parallel collaborative filtering in demand. Systems using distributed computing paradigms are deployed to solve the scalability problem. Although they are able to process large scale

* Corresponding author. Tel.: 86-10-82683605; fax: 86-10-82680698.
E-mail address: yingliu@ucas.ac.cn

doi:10.1016/j.procs.2014.05.382

data sets and relatively easy to deploy, such systems are demonstrated to be energy inefficient and introduce significant accidental complexity. By using OpenMP, MPI and Hadoop parallel programming models, CF algorithms on parallel computing platforms or distributed platforms have achieved some performance enhancement [1-5].

GPU (Graphics Processing Unit) is driven by the insatiable market demand for real-time, high-definition 3D graphics, and has evolved into a highly parallel, multithreaded, many-core processor with tremendous computational horsepower and very high memory bandwidth. Compute Unified Device Architecture (CUDA) is a general purpose parallel computing architecture, which is more and more popular as it makes full use of the computing power of the GPUs to solve complex computational problems efficiently. Heterogeneous computing systems integrated with CPUs and GPUs have offered a new solution for parallel acceleration and gradually become a new hotspot.

By analyzing the bottleneck of the serial collaborative filtering method, in this paper, we propose a novel CUDA-enabled collaborative filtering algorithm, which uses an efficient data partition scheme to maximize the parallelism in the algorithm and make full use of the computational capability of the GPU. Various CUDA optimization techniques are also applied. Experiments were performed on a NVIDIA Tesla C2070 graphics card and Intel Core i3 dual core CPU. By comparing with an efficient implementation of the serial collaborative filtering method, the proposed CUDA-based algorithm shows up to 48.3x speedup on a real-world data set.

The rest of this paper is organized as follows. Section 2 presents the background knowledge of collaborative filtering, GPU architecture and CUDA programming model. Section 3 overviews the existing parallel implementations of collaborative filtering on various parallel processing platforms and their strength and weakness as well. Section 4 describes the algorithm detail of collaborative filtering. In Section 5, we present our CUDA-based algorithm. Experimental results are presented in Section 6. In Section 7, we conclude this paper.

2. Background knowledge

2.1. Collaborative Filtering

As the number of users and digital resources on the Web increase dramatically, recommendation system [6] is experiencing a fast growth and is changing the way for people to receive messages. Collaborative filtering (CF) is one of the most successful technologies in recommendation systems, which has been developed and improved over the past decade to the point where a variety of algorithms have been proposed to generate recommendations. In order to establish a profile of interests, each user in a CF system rates the items that they have experienced. Then, the CF system matches the user with people who share similar interests or tastes with him/her. Ratings from those likely-minded people are used to generate recommendations for the given user.

CF systems have been successful in research, such as GroupLens [7], Ringo [8], Video Recommender [9], and MovieLens [10], etc. In business domain, e-business like Amazon.com, MovieFinder.com, Jd.com, Douban.com have used CF technologies successfully.

2.2. CUDA programming model

Graphic Processor Unit (GPU) has evolved from a specialized processor into a highly parallel, multithreaded, many-core processor with tremendous computational horsepower and very high memory bandwidth. Compared to the CPU, more transistors on the GPU are devoted to computing, so the peak floating-point capability of the GPU is an order of magnitude higher than that of the CPU, as well as the memory bandwidth due to NVIDIA's efforts on optimization.

At the hardware level, CUDA-enabled GPU is a set of SIMD stream multiprocessors (SMs) with 32 stream processors (SPs) each. Tesla C2070 has 448 SPs. Each SM contains a fast shared memory, which is shared by all of its SPs as shown in Fig. 1. It also has a read-only constant cache and texture cache which is shared by all the SPs on the GPU. A set of local 32-bit registers is available for each SP. The SMs communicate through the global/device memory. The global memory can be read or written by the host, and is persistent across kernel launches by the same application. Shared memory is managed explicitly by the programmers.

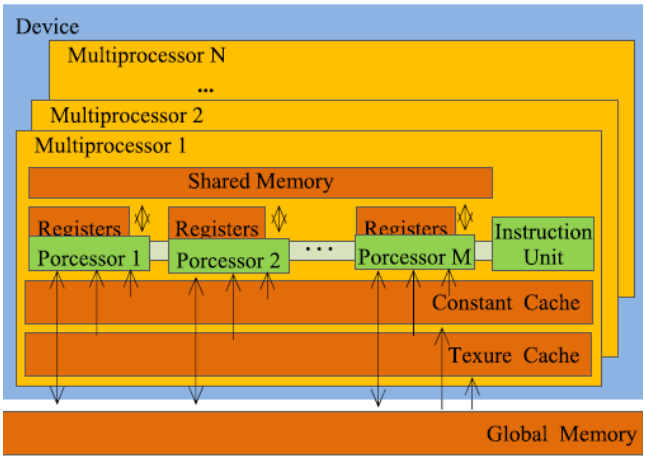

Fig. 1. A set of SIMD stream multiprocessors with memory hierarchy

At the software level, Compute Unified Device Architecture (CUDA) [11] is a general purpose parallel computing architecture with a new parallel programming model. CUDA parallel programming model overcomes the difficulty of programming parallel program on GPU by providing adequate C language-like APIs. The CUDA model is a collection of threads running in parallel. The unit of work issued by the host computer to the GPU is called a kernel. CUDA program is running in a thread-parallel fashion. Computation is organized as a grid of thread blocks which consists of a set of threads as shown in Fig. 1. At instruction level, 32 consecutive threads in a thread block make up of a minimum unit of execution, which is called a thread warp. Each SM executes one or more thread blocks concurrently. A block is a batch of SIMD-parallel threads that runs on the same SM at a given moment. For a given thread, its index determines the portion of data to be processed. Threads in a single block communicate through the shared memory.

As illustrated by Figure 2, the CUDA programming model assumes that the CUDA threads execute on physically separate device that operates as a coprocessor to the host running the C program. This is the case, for example, when the kernels execute on a GPU and the rest of the C program executes on a CPU. The CUDA programming model also assumes that both the host and the device maintain their own separate memory spaces in DRAM, referred to as host memory and device memory, respectively. Therefore, a program manages the global, constant, and texture memory spaces visible to kernels through calls to the CUDA runtime.

3. Related work

As the number of existing users grows tremendously as well as the number of items, traditional CF algorithms suffer from scalability problem, with computational resources going beyond practical or acceptable levels. On the other hand, many systems have to react immediately to online requirements and make recommendations to all users, where a high scalability is in demand.

In order to meet the demands from large-scale systems, distributed computing platforms are typically adopted in data centers. A. Das [1] implemented CF algorithm in MapReduce programming model. Zhao [2] implemented CF algorithm by Hadoop. Although they are able to process large-scale data sets and relatively easy to deploy, they have recently been proved to be energy inefficient [3] and introduce significant accidental complexity [4]. Karydi [5] presented a parallel implementation of slope one algorithm for collaborative filtering with the use of OpenMP and MPI, which is 9.5x faster than the serial algorithm.

As far as our knowledge goes, there is little work focusing on accelerating recommendation systems on GPUs. Kato and Hosino [12] proposed a CUDA implementation of user-user k-nearest neighbor search for user-based CF.

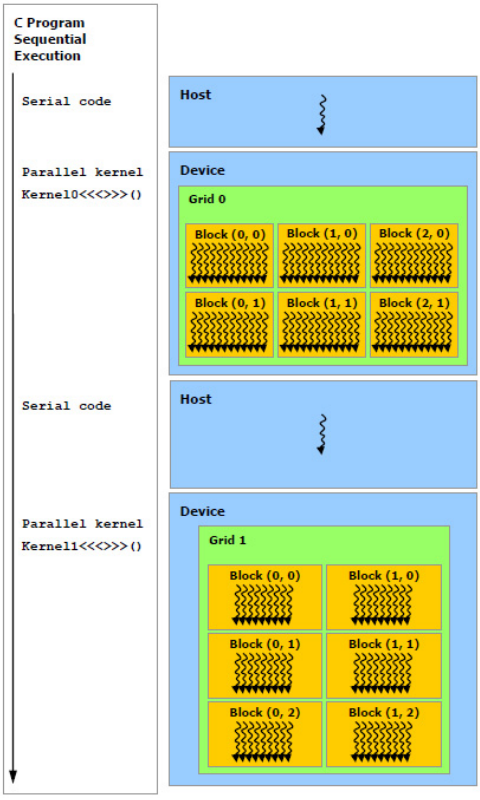

Fig. 2. Serial execution on the host and parallel execution on the device

4. Collaborative filtering algorithm

The computation in collaborative filtering algorithm can be summarized into four steps [13]: similarity matrix calculation, nearest neighbors finding, rating value prediction, and item recommendation.

Let R denote a two-dimensional original rating matrix, with ratings from m users on n items, and let $v(u, i)$ denote the rating value for item i by user u. In addition, we define *USER_K* as the number of neighbors required to calculate the predictive ratings, and *ITEM_K* as the number of items to be recommended. The flow of collaborative filtering algorithm is shown in Figure 3.

Step 1: Calculate the similarity matrix $S(m*m)$ using rating matrix R. Pearson Correlation is adopted as the similarity measure. Each element (u_1, u_2) in S denotes the Pearson Correlation between user u_1 and user u_2.
Step 2: Find *USER_K* nearest neighbors for user u. The larger the correlation, the more similar the two users are.
Step 3: Predict the rating values for items unrated yet by user u. The predictive rating values are obtained by the weighted sum of the items from *u's USER_K* nearest neighbors.
Step 4: Recommend top *ITEM_K* items to user u by the predictive values.

Fig. 3. Flow of collaborative filtering algorithm

The complexity of each step is summarized in Table 1. Obviously, step 1 dominates the execution as it incurs various operations on a large-scale matrix. In real systems, step 1 could be performed offline as a preparation while the other steps have to be calculated online. Fortunately, CUDA has been proved to offer an outstanding performance in matrix operations.

Table 1. Complexity of each step in collaborative filtering algorithm

Step	Complexity
1	O(*m**m*n*)
2	O(*USER_K**m*)
3	O(*USER_K**n*)
4	O(*ITEM_K**n*)

.

5. Design and implementation of CUDA-based collaborative filtering algorithm

5.1. Parallelism in similarity matrix calculation

The purpose of step 1 is to calculate the similarity matrix from a given rating matrix *R* (*m***n*). Each value in this matrix, denoting the similarity between two users, is obtained by Pearson Correlation as in Equation 1:

$$sim(i,j)=\frac{\sum_{c=1}^{m}(R_{i,c}-\bar{R}_i)*(R_{j,c}-\bar{R}_j)}{\sqrt{\sum_{c=1}^{n}(R_{i,c}-\bar{R}_i)^2}*\sqrt{\sum_{c=1}^{n}(R_{j,c}-\bar{R}_j)^2}} \quad (1)$$

Here $R_{i,c}$ and $R_{j,c}$ denotes the rating on item *c* by user *i* and user *j,* $\bar{R}_i$is the average rating of user *i* and *user j,* where *i=1,…m, j=1,…,m*.

Following Equation 1, the program for Pearson Correlation calculation can be summarized as follows:

(1) Calculate $\bar{R}_i$: calculate the average of each row in *R*, and store them in an array;

(2) Calculate $(R_{i,c}-\bar{R}_i)$: transform matrix *R* to *R'* by subtracting the average value of each row;

(3) Calculate $\sum_{c=1}^{n}(R_{i,c}-\bar{R}_i)*(R_{j,c}-\bar{R}_j)$: calculate the dot product of each two rows in *R'*;

(4) Calculate $\sqrt{\sum_{c=1}^{n}(R_{i,c}-\bar{R}_i)^2}$: calculate norms of each row in *R'*;

(5) Calculate the similarity: divide the dot product (step 3) by the norms of two corresponding rows (step 4).

Evidently, the computational core of collaborative filtering is step 3, where massive parallelism exists in pairwise dot product between two vectors (step 3). Actually, all steps except step 1 share the same basic element $(R_{i,c}-\bar{R}_i)$ and do not require independent data traverses. Thus, we parallelize these four steps together in a single CUDA kernel, *similarity matrix calculation*, on CUDA-enabled GPU. In addition, average calculation (step 1) will be parallelized on CUDA independently from other steps.

5.2. CUDA-based collaborative filtering algorithm

Two kernels are identified and parallelized for collaborative filtering. In this section, we present the details of the kernel, *similarity matrix calculation*, as it dominates the total execution time.

Let *TILE_WIDTH* be the width of a CUDA thread block, and *m* be the number of rows in the rating matrix *R*. The similarity matrix calculation is configured in a two-dimensional *m*/*TileWidth* * *m*/*TileWidth* grid, where each block consists of *TileWidth* * *TileWidth* threads. There are *m***n* values in the rating matrix *R* and the size of the similarity matrix *S* is *m***m*.

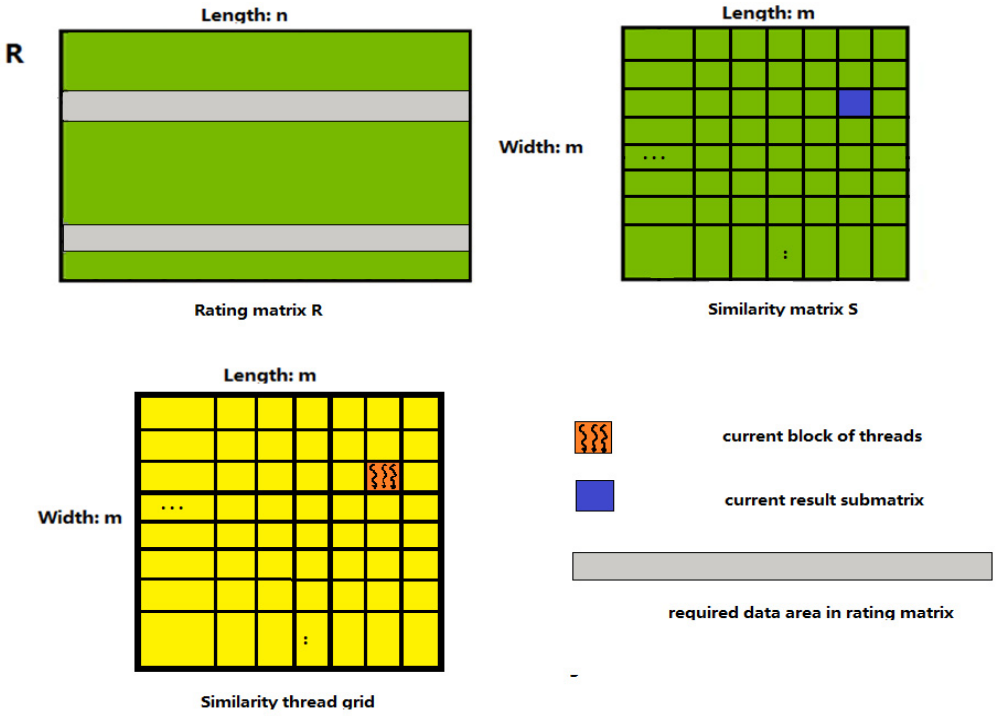

Fig. 4. Data partition and CUDA kernel configuration

The CUDA kernel for similarity matrix calculation is illustrated in Figure 4. Each thread takes care of a single element in the similarity matrix *S*, and each block takes care of a *TileWidth* * *TileWidth* tile of *S*. *S* is obtained by pairwise dot product between any two rows in *R*. Thus, each tile of *S* requires to load two corresponding *TileWidth* * *n* rectangles in *R* as the input, which are in grey in Figure 4. In this CUDA kernel, firstly, threads in each block simultaneously load two tiles of the corresponding rectangles in *R*, respectively; secondly, each thread calculates the partial Pearson Correlation independently using the loaded data and then, stores it in shared memory; next, each thread block proceeds to the next two tiles and the partial correlation is accumulated. It repeats until all the elements in the two rectangles are loaded.

Let *R* be the rating matrix with *wR*TileWidth* columns and *hR*TileWidth* rows. *S* stores the similarity matrix. *IndexA* and *indexB* are the offsets. The pseudo code the similarity matrix calculation CUDA kernel is presented in Figure 5. Since the *S* is a symmetric matrix, we skip the calculation of results below the diagonal. Thus, whenever *by* > *bx*, the kernel quits immediately.

- Memory coalescing

In this kernel, two global memory accesses are incurred. One is to load the average value of each row to shared memory (line 9 and 12), the other is to load the tiles in *R* into shared memory (line 17 and 18). Since the type of the data in *R* is float (4 bytes), and the memory access of a half warp lies in the same 128-byte segment, the global memory access is managed in a coalesced manner.

```
(1)  __global__ void calcSimMatrix(float *R, float *S, float *aveArray, int wA, int hA){
(2)     If (by > bx) return;
(3)     __shared__ float ATile[TILE_WIDTH][TILE_WIDTH];
(4)     __shared__ float BTile[TILE_WIDTH][TILE_WIDTH+1];
(5)     __shared__ float AAverage[TILE_WIDTH];
(6)     __shared__ float BAverage[TILE_WIDTH];
(7)
(8)    if(ty == 0){
(9)        load the average values of ATile into AAverage array;
(10)   }
(11)   if(ty == 1){
(12)       load the average values of BTile into BAverage array;
(13)   }
(14)   synchronize threads;
(15)
(16)   for(int i=0; i<wR; i++){
(17)                ATile[ty][tx]=R[indexA] - AAverage[ty];
(18)                BTile[ty][tx]=R[indexB] - BAverage[ty];
(19)                indexA+=TILE_WIDTH;
(20)                indexB+=TILE_WIDTH;
(21)                synchronize threads;
(22)
(23)       dotProduct + =∑_{k=0}^{TILE_WIDTH}(ATile[ty][k] * BTile[tx][k]);
(24)       sigmaA + =∑_{k=0}^{TILE_WIDTH}(ATile[ty][k] * ATile[ty][k]);
(25)       sigmaB + =∑_{k=0}^{TILE_WIDTH}(BTile[tx][k] * BTile[tx][k]);
(26)       synchronize threads;
(27)       }
(28)       pearson_correlation = dotProduct / sqrt(sigmaA*sigmaB);
(29)       store pearson correlation in global memory;
```

Fig. 5. Pseudo code of the CUDA kernel for similarity matrix calculation

- Bank conflict

When threads in a half warp access *AAverage* (line 17) and *BAverage* (line 18), they actually acquire exactly the same address, that is, the data is broadcast. When threads in a half warp access *ATile* (line 17), no bank conflict is incurred. As different addresses in the same bank are requested by a half warp, bank conflict is incurred when accessing *BTile* (line 18). In order to circumvent the problem, we allocate an extra column as in line 4. Thus, when reading *BTile*, all the threads in a half warp read from consecutive banks.

6. Experiment results

6.1. Experimental setup

The device we used in our experiment is a NVIDIA's Tesla C2070 graphics card, which is a dedicated general-purpose computing GPU with 448 1.15GHz SPs and 6 GB global/device memory. All the experiments were performed on a workstation with a dual-core 3.3 GHz Intel Core i3-3220 CPU and 8 GB main memory.

6.2. Data set

We evaluated our proposed CUDA-based collaborative filtering algorithm on a real-world data set downloaded from MovieLens.com, which contains 72000 distinct users and their ratings for 10000 distinct items.

6.3. Experimental results

In order to evaluate the scalability, we varied the number of users and the number of items, respectively. The time cost in data transfer between the CPU and GPU was not included as it is trivial in the experiments. The execution time of the traditional CPU-based collaborative filtering algorithm and our CUDA-based algorithm are measured and presented in Table 2 and Table 3. The corresponding speedups are presented as well.

Table 2. Execution time and speedup when varying the number of users (number of users m=4800)

number of items n	Execution time on CPU (ms)	Execution time on GPU (ms)	Speedup
n=3200	113175.0	2439.3	46.4
n=1600	57049.0	1222.5	46.7
n=800	28308.0	614.4	46.1
n=400	14300.0	310.2	46.1

Table 3. Execution time and speedup when varying the number of users (number of items n=4800)

number of users m	Execution time on CPU (ms)	Execution time on GPU (ms)	Speedup
m=3200	75067.0	1628.6	46.1
m=1600	18717.0	410.8	45.6
m=800	4695.0	105.6	44.5
m=400	1186.0	27.4	43.3

The minimum speedup, 43.3X, is observed at the case with only 400 users and 4800 items. It indicates that when the number of users is small, the computing power of the GPU is not fully used. When the number of users increases, better speedups are observed. It indicates that as the parallelism increases, the GPU is utilized more and more efficiently.

7. Conclusion

In order to overcome the scalability problem in collaborative filtering algorithm, in this paper, we proposed a CUDA-based collaborative filtering algorithm. Computational cores and bottlenecks were identified after careful complexity analysis: similarity matrix calculation and average calculation. Thus, we proposed and implemented the two kernels on CUDA-enabled GPU. Since similarity matrix calculation dominates the execution time, we presented the detail of the kernel as well as the optimization techniques applied on it. Experiments were performed on a real-world data set. The results demonstrated that our proposed CUDA-based collaborative filtering algorithm is efficient and scalable. The observations also indicate that the more computation on the GPU, the higher the utilization of the computing horsepower.

Acknowledgements

This project is partially supported by grants from Natural Science Foundation of China #61202312/70621001/70921061. It is partially supported by NVIDIA as we were awarded as CUDA Teaching Center in 2011 and CUDA Research Center in 2013.

References

[1] Das A. S., Datar M, Garg A and RajaramS. Google news personanlization: scalable online collaborative filtering. WWW`07: Proceedings of the 16th international conference on World Wide Web, 2007 Banff, Alberta, Canada, pp. 271-280.
[2] Z.-D. Zhao and M. sheng Shang. User-based collaborative-filtering recommendation algorithms on hadoop. International Workshop on Knowledge Discovery and Data Mining, 0:478-481, 2010.
[3] Leverich, J., and Kozyrakis, C., "On the energy (in)efficiency of Hadoop clusters", SIGOPS Oper. Syst. Rev., 2010, 44, (1), pp. 61-65.
[4] Borkar, V., Carey, M., et al., "Hyracks: A flexible and extensible foundation for data-intensive computing". Proc. IEEE 27th International Conference on Data Engineering (ICDE) Hanover, Germany, 2011 pp. 1151-1162.
[5] Karydi E, Margaritis K G. Parallel Implementation of the Slope One Algorithm for Collaborative Filtering[C], 2012 16th Panhellenic Conference on. IEEE, 2012: 174-179.
[6] N.J.Belkin, W.B.Croft. Information filtering and information retrieval: two sides of the same coin, Community. ACM, vol.35 issue 12, pp.29-38, 1992.
[7] Konstan, J.A., Miller, B.N., Maltz, D., Herlocker, J.L., Gordon, L.R., Riedl, J., 1997. GroupLens:Applying collaborative filtering to Usenet news. Communications of the ACM 40(3), 77-87.
[8] Shardanand, U., Maes, P., 1995. Social Information Filtering: Algorithms for Automation “World of Mouth”. Proceedings of ACM CHI`95. Denver, CO., pp. 210-217.
[9] Hill, W., Stead, L., Rosenstein, M., Furnas, G.W., 1995. Recommending and Evaluating Choices in a Virtual Community of Use. Proceeding of ACM CHI`95 Conference on human factors in computing systems. Denver, CO., pp, 194-201.
[10] Dahlem, B.J., Lonstan, J.A., Herlocker, J.L., Good, N., Borchers, A., Riedl, J., 1998. Jump-starting movielens: User benefits of starting a collaborative filtering system with “dead data”. University of Minnesota, TR 98-017.
[11] CUDA C Programming Guide version 4.1, NVIDIA Corp, 2012.
[12] K. Kato and T. Hosino, “Solving k-Nearest Neighbor Problem on Multiple Graphics Processors,” IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, 769-773, 2010.
[13] J Herlocker, J A Konstan, A Borchers, et al., An algorithmic framework for performing collaborative filtering. Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieve. New York: ACM Press, 1999, pp. 230-237.