GPU Accelerated Randomized Singular Value Decomposition and Its Application in Image Compression

Hao Ji and Yaohang Li
Department of Computer Science
Old Dominion University
hji@cs.odu.edu, yaohang@cs.odu.edu

Abstract
In this paper, we present a GPU-accelerated implementation of randomized Singular Value Decomposition (SVD) algorithm on a large matrix to rapidly approximate the top-k dominating singular values and correspondent singular vectors. The fundamental idea of randomized SVD is to condense a large matrix into a small dense matrix by random sampling while keeping the important information. Then performing traditional deterministic SVD on this small dense matrix reveals the top- k dominating singular values/singular vectors approximation. The randomized SVD algorithm is suitable for the GPU architecture; however, our study finds that the key bottleneck lies on the SVD computation of the small matrix. Our solution is to modify the randomized SVD algorithm by applying SVD to a derived small square matrix instead as well as a hybrid GPU-CPU scheme. Our GPU-accelerated randomized SVD implementation is around 6~7 times faster than the corresponding CPU version. Our experimental results demonstrate that the GPU-accelerated randomized SVD implementation can be effectively used in image compression.

Keywords: Random Sampling, Singular Value Decomposition, Low-Rank Approximation, Image Compression, Graphics Processing Unit

1. Introduction

A factorization of a real matrix $A \in \mathbb{R}^{m*n}$ is singular value decomposition (SVD) if

$$A = U * \Sigma * V^T$$

where $U \in \mathbb{R}^{m*m}$ and $V \in \mathbb{R}^{n*n}$ are matrices with orthonormal columns, $\Sigma \in \mathbb{R}^{m*n}$ is a diagonal matrix whose elements, $\sigma_1, \sigma_2, \ldots, \sigma_n$, are nonnegative singular values in non-decreasing order. SVD plays an important role in a wide range of modeling and simulation applications, such as modeling of genome-wide expression data [1], large-scale atmosphere-ocean interaction analysis [2], data-unfolding in high energy physics [3], recommendation engine in social network modeling [4] and simulations with MRI data [5], etc. One primary advantage of using SVD is that the low rank approximation A_k to matrix A with rank k can be readily formed as

$$A_k = M * N$$

where M is an $m * k$ matrix and N is an $k * n$ matrix. Consequently, the factorized matrices containing most important characteristics of the original matrix can be used for efficient modeling and computing, while those small matrices are inexpensive to store and manipulate.

The traditional deterministic SVD algorithm [6] on a large matrix is computationally intensive, which has cubic-time complexity with respect to the size of the given matrix. For an $m * n$ matrix A, when both m and n are large, deterministic SVD also requires large memory space. Randomized SVD [7-10, 15], by contrast, offers efficient alternatives to approximate the dominant singular components. Williams and Seeger [7] proposed a Nyström method to accelerate decomposition in kernel machines. Frieze et al. [8] studied column-sampling method for finding low-rank approximations in constant time. Drineas [9] modified column-sampling method to make it fit the pass-efficient model of data-streaming computation. Holmes et al. [15] developed a cosine tree sampling method for fast approximation of the complete matrix SVD. Halko et al. [10] performed random sampling on the original matrix to construct a small condensed subspace. The dominant actions of the original matrix A could be quickly estimated from this small subspace with relatively low computation cost and high confidence. Matrix operations on the projected small subspace allow randomized SVD algorithms to take advantage of the emerging high performance computing platforms, for instance, distributed memory systems, multi-core processors, multi-general purpose graphics process units (GPGPU), and the Cloud computing infrastructure [11, 14].

Graphics Processing Unit (GPU) is a specialized single-chip processor to take advantage of parallelism to achieve high performance computing. Many high performance linear algebra libraries on GPU architectures are available. For instance, CUBLAS (CUDA Basic Linear Algebra Subroutines) [12] contains the GPU-accelerated functions of basic dense matrix operations. Complementary to CUBLAS, CULA [13] is an extended linear algebra library provides high-level equivalent routines of LAPACK over CUDA runtime. Based on these GPU-accelerated libraries, Foster et al. [16] designed a GPU-based cosine tree sampling algorithm for

column-sampling SVD and achieved speedup of 6~7 over CPU implementation in large matrices.

In this paper, we focus on the randomized SVD algorithm proposed by Halko et al. [10] and present a GPU-accelerated implementation to quickly obtain the approximate of dominant singular components of a given large matrix. We find that the main bottleneck in the GPU implementation is the deterministic SVD on GPU with "short-and-wide" matrix. Using SVD decomposition on a derived square matrix instead can significantly reduce the overall computational time. In addition, in the case of matrices with a small dominant rank *k* value, if a hybrid GPU-CPU scheme is carried out, the efficiency of our implementation can be further improved.

The rest of the paper is organized as follows. Section 2 describes the randomized SVD algorithm. In section 3, we analyze the GPU-accelerated implementation of randomized SVD algorithm. Section 4 shows experimental results on large matrices and a NASA Mars image. Section 5 summarizes the paper.

2. The Randomized SVD Algorithm

The Randomized SVD algorithm was introduced by N. Halko [10-11] to obtain a low-rank approximation of a large matrix. Instead of directly performing deterministic SVD on a large matrix, which is usually not only computationally costly but also memory intensive, the randomized SVD algorithm starts from a small random subspace and then projects the original matrix onto this subspace. The fundamental idea is, as the most important characteristics of the original matrix A are condensed into a small randomized subspace, this projected subspace becomes an amenable choice to approximate matrix decomposition but avoiding high computational cost. Algorithm 1 describes the Randomized SVD algorithm given an input matrix.

Algorithm 1 Randomized SVD

Input: $A \in \mathbb{R}^{m*n}$, $k \in \mathbb{N}$ and $p \in \mathbb{N}$ satisfying $k + p \leq min(m, n)$.

Output: *k*-rank randomized SVD components $U \in \mathbb{R}^{m*k}$, $\Sigma \in \mathbb{R}^{k*k}$ and $V \in \mathbb{R}^{k*n}$

1. Construct an $n * (k + p)$ random matrix Ω
2. $Y = A\Omega$
3. Compute an orthogonal basis $Q = qr(Y)$
4. $B = Q^T A$
5. $[U_B, \Sigma_B, V_B] = svd(B)$
6. Update $U_B = Q U_B$
7. $U = U_B(:, 1:k), \Sigma = \Sigma_B(1:k, 1:k)$ and $V = V_B(:, 1:k)$

To show how random sampling of A can extract information of the top k singular values/vectors, let $\omega_j \in \mathbb{R}^n$ and $y_j \in \mathbb{R}^n$ denote the jth column vector of random matrix Ω and the j th column vector of matrix Y, respectively. Since each element in Ω is chosen independently, ω_j can be represented as

$$\omega_j = c_{1j}v_1 + c_{2j}v_2 + \cdots + c_{nj}v_n, \quad j = 1, \ldots, k + p$$

where $v_i \in \mathbb{R}^n$ is ith right singular vector of matrix A and $c_{ij} \neq 0$ with probability 1.0. In Algorithm 1, after simply projecting A onto Ω, we could have

$$y_j = \sigma_1 c_{1j} v_1 + \sigma_2 c_{2j} v_2 + \cdots + \sigma_n c_{nj} v_n.$$

where σ_i is the ith singular value of A sorted by non-decreasing order such that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$ and $\sigma_i c_{ij}$ constitutes the weight of y_j on v_i. Consequently, random sampling ensures that all singular vectors are kept in the subspace but the singular vectors corresponding to bigger singular values likely yield bigger weights in y_j. Therefore, compared to ω_j, weights of dominating right singular vectors are amplified by the corresponding singular values. As a result, the space spanned by the columns of Y reflects dominating weights in high probability on the singular vectors corresponding to the top k singular values.

For stability consideration, Y is augmented to $m * (k + p)$ instead of $m * k$ to incorporate additional p dimensional subspace. Correspondingly, B is a $(k + p) * n$ matrix. When the SVD decomposition on B is carried out to approximate the top k singular values/vectors of A, this additional p −dimension space can serve as a noise-filter to get rid of unwanted subspace corresponding to small singular values. In practice, p is given with small value, such as 5 or 10, as suggested by Halko [10-11].

The main computational operations in the randomized SVD algorithm involve matrix-matrix multiplications, QR decompositions, and SVD on small matrices, which are naturally fit to parallel computing platforms, for instance, distributed memory, multi-core, multi-general purpose graphics process units (GPGPU) and the Cloud computing infrastructure[11, 14].

3. Implementation

3.1 GPU-accelerated Implementation

The randomized SVD involves the following five primary computational components described in the section 2:

(1) generation of random matrix Ω;
(2) matrix-matrix multiplication of $A\Omega$ to produce Y;
(3) QR decomposition on Y;
(4) matrix-matrix multiplication of $Q^T A$; and
(5) deterministic SVD decomposition on B.

Figure 1 shows a hypothetical description of randomized SVD in finding approximate right-hand-side top- k singular vectors. The overall performance of randomized SVD depends on the efficiency of matrix-matrix multiplication, QR factorization, and SVD on small

matrices. Fortunately, after random matrix sampling by Ω, the large matrix A is condensed into either "tall-and-skinny" or "short-and-wide" matrix, such as Y and Q are $m * (k + p)$ "Tall-and-skinny" matrices, B is an $(k + p) * n$ "short-and-wide" matrix where $k + p$ is much smaller than $min(m, n)$. These small and dense matrices are particularly suitable fit in GPU memory to take advantage of high-performance computation provided. We implemented randomized SVD on GPU using CUBLAS [12] and CULA [13], and its corresponding CPU version using the Intel multi-thread MKL (Math Kernel Library) for the sake of performance illustration.

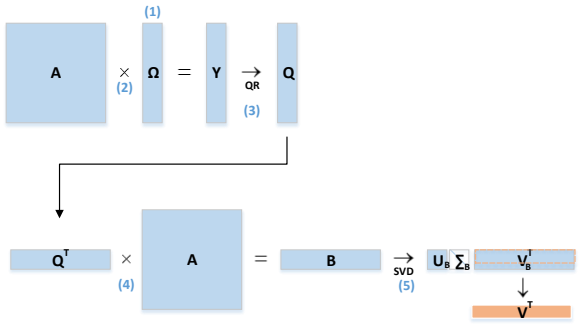

Figure 1. Procedure of Randomized SVD to Approximate Right-singular Vectors

The elapsed time spent on each primary computational component in randomized SVD is shown in Figure 2 for a 4,096 × 4,096 random matrix where k is 128 and p is 3. Multiplication between A and a "tall-and-skinny" or "short-and-wide" matrix can be efficiently carried out on the GPU's SIMT architecture and hence the computational time in generating matrix Ω and performing matrix-matrix multiplications shrinks to nearly negligible. Nevertheless, deterministic SVD, particularly when the target matrix is small, has difficulty in fully taking advantage of GPU architecture, due to a series of sequential Householder transformations need to be applied. As a result, deterministic SVD becomes the main bottleneck and thus this GPU implementation has only 1.65 over that of the CPU.

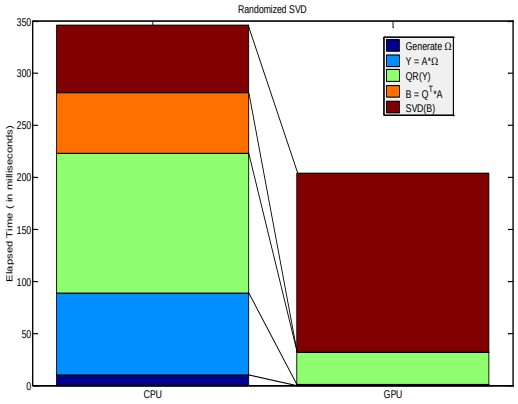

Figure 2. The Elapsed Computational Time Used in Randomized SVD on CPU-Only and GPU-Only

3.2 Approximate SVD decomposition of B

To reduce the computational cost of deterministic SVD in GPU randomized SVD implementation, we alternatively calculate the top-k singular vectors of BB^T instead of directly carrying out deterministic SVD on the "short-and-wide" matrix B. Figure 3 depicts the procedure of obtaining approximate SVD decomposition of B. Note that SVD decomposition of B is defined as

$$B = U_B \Sigma_B V_B^T.$$

Since BB^T is a small square matrix whose size is independent of the size of the original matrix A, and has SVD format as,

$$BB^T = U_B \Sigma_B U_B^T,$$

U_B could be very efficiently derived from BB^T rather than from B.

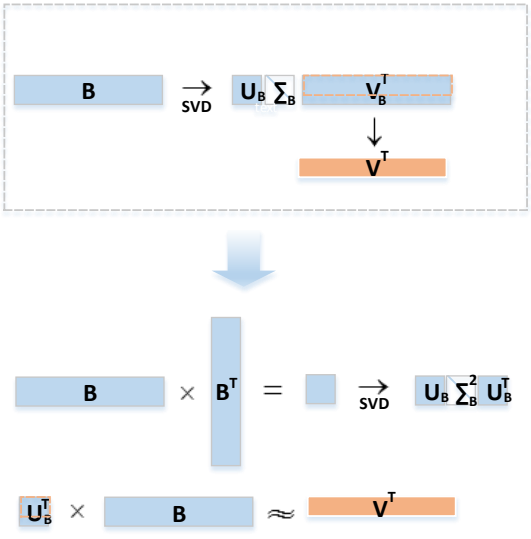

Figure 3. Procedure of Obtaining Approximate SVD Decomposition of B

Once the left singular vectors U_B become available, under the assumption that $U_B^T U_B \approx I$, where I is an identity matrix, the top k singular components could be approximated effectively through a single matrix-matrix operation

$$U_B^T B \approx \Sigma_B V_B^T.$$

Figure 4 shows the elapsed time of the improved implementation by using BB^T on the same 4,096 × 4,096 random matrix used in Figure 2. One can find that the portion of SVD computation time is significantly reduced on both CPU and GPU implementations. Consequently, the achieved speedup of GPU implementation grows up to 4.6.

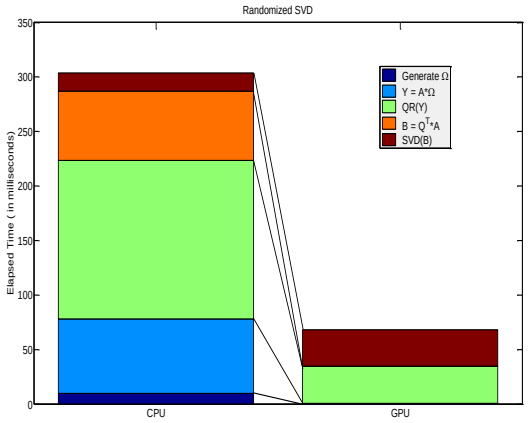

Figure 4. The Elapsed Time Used in Randomized SVD on CPU-Only and GPU-Only by Using BB^T

3.3 Hybrid GPU-CPU Scheme

As shown in figure 4, even though the alternative approach of approximating top-k singular values/singular vectors on BB^T is used, the computational time of deterministic SVD on GPU is still more than that of the CPU version due to hidden setup on GPU. To further understand the performance of deterministic SVD on GPU, we compute deterministic SVD to a set of square matrices varying in size. Figure 5 compares the computational time of deterministic SVD on CPU and GPU. One can find that the CPU implementation outperforms the GPU one on small matrices less than $2{,}500 * 2{,}500$. Therefore, using GPU to run SVD operations on small matrices is not appropriate, particularly for applications where the singular values decay very quickly and k is typically set with very small value.

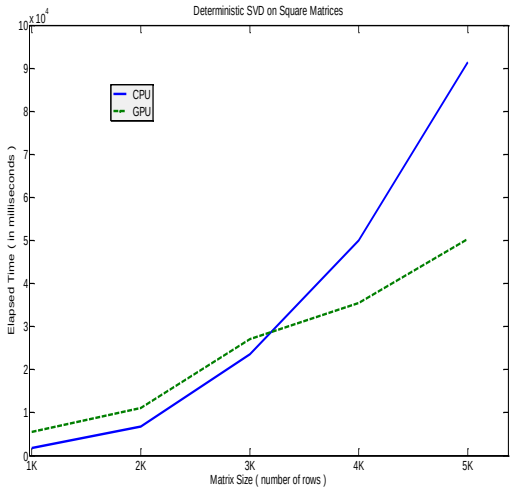

Figure 5. Comparison of Running Time for Performing Deterministic SVD on GPU and CPU

In our implementation, we develop a simple hybrid GPU-CPU scheme. If the $k * k$ square matrix is small, it will be transferred to the CPU to carry out deterministic SVD decomposition instead.

4. Results

In this section, we present the numerical results obtained with GPU-accelerated implementation on large random matrices and Mars image. The experiments are carried out on a Linux computer with an Intel Core i5-2500K CPU 3.30GHz CPU, 8GB of RAM and an NVIDIA GK110GL GPU.

4.1 Random Matrices

We generate a series of large random dense matrices of varying sizes to benchmark the performance achieved by using our GPU-accelerated randomized SVD algorithm. Figure 6 compares the computational time in logarithmic scale of performing complete SVD and randomized SVD on CPU as well as GPU-accelerated randomized SVD algorithm. The same k and p ($k = 256$ and $p = 3$) values are used. Compared to doing the complete SVD calculation on the matrix, randomized SVD has a clear computational advantage when only the top-k approximated singular components are needed. When the GPU architecture is taken advantage of, a more aggressive speedup is achieved.

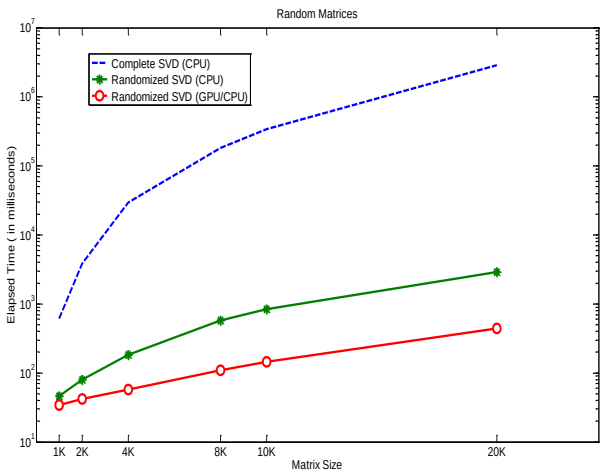

Figure 6. Comparison of Elapsed Time (logarithmic scale) of Deterministic SVD, CPU versions of Randomized SVD and GPU-accelerated Randomized SVD

Figure 7 illustrates the speedup factor for our GPU-accelerated implementation of randomized SVD over the corresponding CPU-based one. Similar to many other GPU-based algorithms, our GPU randomized SVD implementation favors larger matrices. For a $20{,}000 * 20{,}000$ matrix, the speedup can reach up to 6~7.

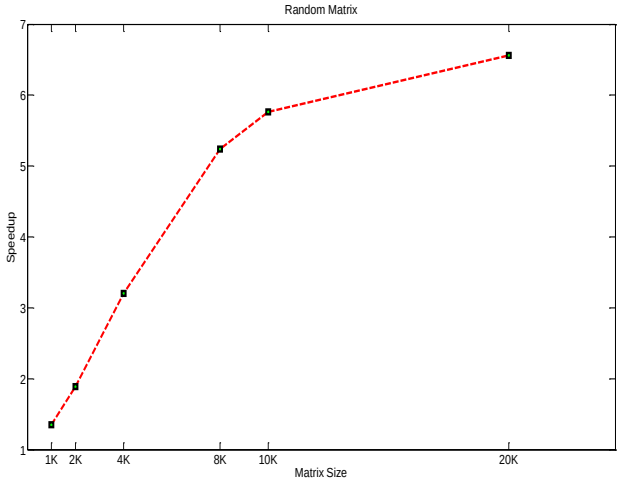

Figure 7. The Speedup of GPU-accelerated Implementation over the CPU-only Implementation.

4.2 Image Compression

We apply the randomized SVD algorithm for lossy data compression to a NASA synthesis image from the Mars Exploration Rover mission [17] shown in Figure 8. The image is an RGB $7671 * 7680 * 3$ matrix, which requires 176.74 million bytes for memory storage.

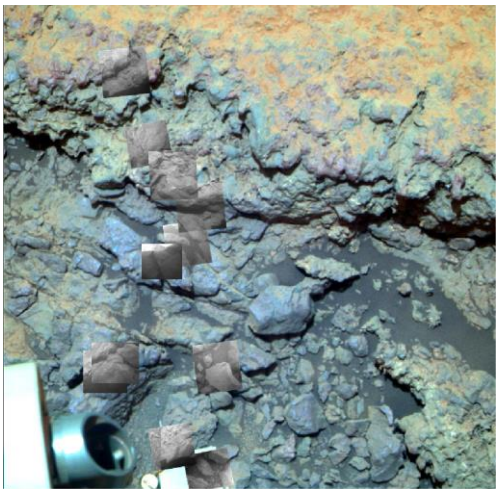

Figure 8. The Original Image

In order to compress the image, we use our GPU-accelerated implementation to obtain its low rank approximation A_k with rank 470,

$$A_k = M * N$$

where M is a 7671×470 matrix and N is a $470 * 7680$ matrix on each color channel (R,G,B). Figure 9 shows the reconstructed image, where M is computed by combining the 470 left singular vectors with the corresponding singular values while N is stored as the 470 right singular vectors as columns. To outline the effectiveness of our implementation of randomized SVD, Table 1 lists the elapsed computational time and error used in compression with Mars Image. As one can find, compared to deterministic SVD which consumes more than one thousand seconds to obtain the top 470 approximation, the GPU-accelerated randomized SVD only takes slightly more than one second. The overall storage of the decomposed image requires less than 1/8 of that of the original matrix with an acceptable 1.63% error.

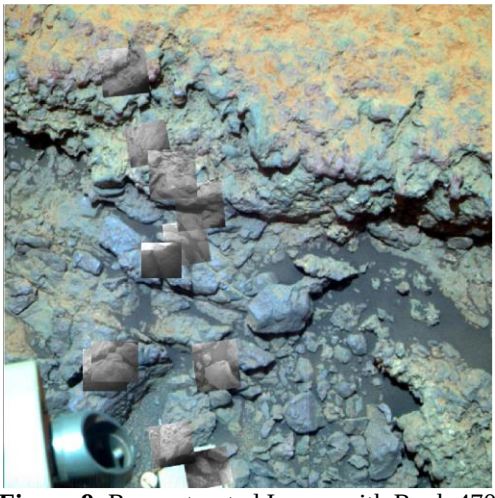

Figure 9. Reconstructed Image with Rank 470

	Elapsed Time (in seconds)	Error in Compression
Deterministic SVD	1144.71	1%
Randomized SVD	1.29	1.63%

Table 1. Elapsed Computational Time and Error in Compression with the Mars Image

5. Conclusions

In this paper, we present a GPU-accelerated implementation of randomized SVD to accelerate the process of approximating dominating singular components using both GPU and CPU. The efficiency is further improved by performing SVD decomposition on a small square matrix, which is the product of a "tall-and-skinny" matrix and its transpose. Our computational results on large random matrices and a NASA synthesis image show that the dominating singular components can be effectively obtained and the GPU-accelerated implementation outperforms the corresponding CPU version by around 6~7 times.

Acknowledgements

Yaohang Li acknowledges support from ODU 2013 Multidisciplinary Seed grant. Hao Ji acknowledges support from ODU Modeling and Simulation Fellowship.

References

[1] Alter, Orly, Patrick O. Brown, and David Botstein. 2000. "Singular value decomposition for genome-wide expression data processing and modeling." Proceedings of the National Academy of Sciences 97, no. 18: 10101-10106.

[2] Wallace, John M., Catherine Smith, and Christopher S. Bretherton. 1992. "Singular value decomposition of wintertime sea surface temperature and 500-mb height anomalies." Journal of climate 5, no. 6: 561-576.

[3] Hoecker, Andreas, and Vakhtang Kartvelishvili. 1996. "SVD approach to data unfolding." Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 372, no. 3: 469-481.

[4] Sarwar, Badrul, George Karypis, Joseph Konstan, and John Riedl. 2002. "Incremental singular value decomposition algorithms for highly scalable recommender systems." In Fifth International Conference on Computer and Information Science: 27-28.

[5] Calamante, Fernando, David G. Gadian, and Alan Connelly. 2000. "Delay and dispersion effects in dynamic susceptibility contrast MRI: simulations using singular value decomposition." Magnetic resonance in medicine 44, no. 3: 466-473.

[6] Golub, Gene H., and Charles F. 2012. Van Loan. Matrix computations. Vol. 3. JHU Press.

[7] Williams, Christopher, and Matthias Seeger. 2001. "Using the Nyström method to speed up kernel machines." In Advances in Neural Information Processing Systems 13.

[8] Frieze, Alan, Ravi Kannan, and Santosh Vempala. 2004. "Fast Monte-Carlo algorithms for finding low-rank approximations." Journal of the ACM (JACM) 51, no. 6: 1025-1041.

[9] Drineas, Petros, Ravi Kannan, and Michael W. Mahoney. 2006. "Fast Monte Carlo algorithms for matrices II: Computing a low-rank approximation to a matrix."SIAM Journal on Computing 36, no. 1: 158-183.

[10] Halko, Nathan, Per-Gunnar Martinsson, and Joel A. Tropp. 2011. "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions." SIAM review 53, no. 2: 217-288.

[11] Halko, Nathan P. 2012. "Randomized methods for computing low-rank approximations of matrices." PhD diss., University of Colorado.

[12] Nvidia, C. U. D. A. 2008. "Cublas library." NVIDIA Corporation, Santa Clara, California15.

[13] Humphrey, John R., Daniel K. Price, Kyle E. Spagnoli, Aaron L. Paolini, and Eric J. Kelmelis. 2010. "CULA: hybrid GPU accelerated linear algebra routines." InSPIE Defense, Security, and Sensing, pp. 770502-770502. International Society for Optics and Photonics.

[14] Mahoney, Michael W. 2011. "Randomized algorithms for matrices and data." arXiv preprint arXiv:1104.5557.

[15] Holmes, Michael P., Alexander G. Gray, and Charles Lee Isbell Jr. 2008. "QUIC-SVD: Fast SVD Using Cosine Trees." In NIPS, pp. 673-680.

[16] Foster, Blake, Sridhar Mahadevan, and Rui Wang. 2012. "A GPU-based approximate SVD algorithm." In Parallel Processing and Applied Mathematics, pp. 569-578. Springer Berlin Heidelberg.

[17] http://photojournal.jpl.nasa.gov/catalog/PIA14745.

Biographies

Hao Ji is a Ph.D. student in the Department of Computer Science at Old Dominion University. He received the B.S. degree in Applied Mathematics and M.S. degree in Computer Science from Hefei University of Technology in 2007 and 2010, respectively. His research interest is large-scale scientific computing.

Yaohang Li is an Associate Professor in Computer Science at Old Dominion University. He received his B.S. in Computer Science from South China University of Technology in 1997 and M.S. and Ph.D. degrees from the Department of Computer Science, Florida State University in 2000 and 2003, respectively. After graduation, he worked as a research associate in the Computer Science and Mathematics Division at Oak Ridge National Laboratory, TN. His research interest is in Computational Biology, Monte Carlo Methods, and High Performance Computing.