GPU Acceleration of the Simplex Volume Algorithm for Hyperspectral Endmember Extraction

Haicheng Qu[a,b*], Junping Zhang[a], Zhouhan Lin[a], Hao Chen[a], and Bormin Huang[c]

[a] Dept. of Information Engineering, Harbin Institute of Technology, Harbin 150001, China;
[b] Dept. of Software Engineering, Software College of Liaoning Technical University, Huludao,125105,China
[c]Space Science and Engineering Center, University of Wisconsin-Madison, Madison, WI 54706, USA

ABSTRACT

The simplex volume algorithm (SVA)[1] is an endmember extraction algorithm based on the geometrical properties of a simplex in the feature space of hyperspectral image. By utilizing the relation between a simplex volume and its corresponding parallelohedron volume in the high-dimensional space, the algorithm extracts endmembers from the initial hyperspectral image directly without the need of dimension reduction. It thus avoids the drawback of the N-FINDER algorithm, which requires the dimension of the data to be reduced to one less than the number of the endmembers. In this paper, we take advantage of the large-scale parallelism of CUDA (Compute Unified Device Architecture) to accelerate the computation of SVA on the NVidia GeForce 560 GPU. The time for computing a simplex volume increases with the number of endmembers. Experimental results show that the proposed GPU-based SVA achieves a significant 112.56x speedup for extracting 16 endmembers, as compared to its CPU-based single-threaded counterpart.

Keywords: Simplex volume algorithm, endmember extraction, GPU, CUDA

1. INTRODUCTION

Endmember extraction is the process of selecting a collection of pure signature spectra of the materials present in a remotely sensed hyperspectral scene which is the prerequisite for interpretation and future analysis of hyperspectral image data[2]. How to extract endmembers from hyperspectral image data rapidly and accurately is a hot topic in the research field of spectral unmixing. To solve the above problems, many classical algorithms which can be categorized as either the statistical methods based on Bayesian framework or the geometrical methods based on the theory of convex geometry have been proposed. The statistical methods generally require prior knowledge or appropriate prior distributions for the unknown signal parameters such as the abundance coefficients and the noise variance. The geometrical methods which are under the linear mixing model and the non-negativity and sum-to one constrains are further classified into two classes: one approach is based on the pure-pixel assumption such as N-FINDER, pure pixel index (PPI), vertex component analysis (VCA), and simplex growing algorithm (SGA), the other is based on non-pure-pixel assumption such as nonnegative matrix factorization, minimum volume enclosing simplex, iterated constrained endmembers[2,6], etc.

By comparison, the N-FINDER algorithm is a time-consuming approach to find the true endmember simplex subject to the number of samples and the number of endmembers. The approach requires dimension reduction as preprocessing which may lose the information of small targets. The simplex volume algorithm[2] (SVA) based on a special geometrical structure called convex simplex polytope was proposed to improve the algorithm without dimension reduction preprocessing[1]. However, the SVA requires the determinant computation which is still time-consuming, particularly when the number of endmembers is large. Fortunately, the computational complexity depends on the number of endmembers and there exists no dependency between pixels, so this algorithm is suitable for parallel implementation. The GPU computing[3-10] is a promising technique for hyperspectral image processing[3] due to its superior computing capabilities and higher data bandwidth, as compared to its CPU counterpart. In this paper we propose to develop a highly parallel GPU version of SVA. The rest of this paper will be arranged as follows. Section 2 describes the Simplex volume

* haichengqu@gmail.com; phone: 86-135-91995699

High-Performance Computing in Remote Sensing II, edited by Bormin Huang, Antonio J. Plaza,
Proc. of SPIE Vol. 8539, 85390B · © 2012 SPIE · CCC code: 0277-786/12/$18
doi: 10.1117/12.977956

algorithm. Section 3 explains its GPU implementation. Section 4 shows the experimental results and Section 5 makes a summary.

2. METHOD

In this section we briefly describe the simplex volume algorithm[1]. More details can be found in the original literature[1].

2.1 Simplex volume algorithm description

The volume formula for the n-dimensional parallelohedron can be defined as follows:

$$\bar{V}_m = \bar{V}(v_1, v_2, ..., v_m) = \sqrt{\det(A_m{}^T A)} = \sqrt{G(v_1, v_2, ..., v_m)} \tag{1}$$

where $v_i\ (i = 1, 2, ..., m)$ is a column vector in the *n*-dimensions space; $A_m = [v_1, v_2, ..., v_m]$; and $G(v_1, v_2, \cdots, v_m)$ is described by the Cramer determinant formula:

$$G(v_1, v_2, \cdots, v_m) = \begin{vmatrix} v_1 v_1{}^T & v_1 v_2{}^T & \cdots & v_1 v_m{}^T \\ v_2 v_1{}^T & v_2 v_2{}^T & \cdots & v_2 v_m{}^T \\ \cdots & \cdots & \cdots & \cdots \\ v_m v_1{}^T & v_m v_2{}^T & \cdots & v_m v_m{}^T \end{vmatrix} . \tag{2}$$

For example,

$$m = 1,\ \bar{V}(v_1) = \| v_1 \| = \sqrt{v_1 \cdot v_1} \tag{3}$$

$$m = 2,\ \bar{V}(v_1, v_2) = \sqrt{G(v_1, v_2)} = \sqrt{\det\left(\begin{pmatrix} v_1 \\ v_2 \end{pmatrix} (v_1, v_2)\right)} \tag{4}$$

$$m = 3,\ \bar{V}(v_1, v_2, v_3) = \sqrt{G(v_1, v_2, v_3)} = \sqrt{\det\left(\begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} (v_1, v_2, v_3)\right)} \tag{5}$$

The relation between an *m*-dimensional simplex volume V_m and its corresponding *m*-dimensional parallelohedron volume $\overline{V_m}$ is described in Eq. (6):

$$V_m = \frac{1}{m!} \overline{V_m} . \tag{6}$$

The SVA is a spectral unmixing method based on the geometrical structure of a simplex to extract endmembers. Suppose that $p_1, p_2 \ldots p_n$ are pixels (or spectral vectors) in a hyperspectral image and $A_{n-1} = (p_2 - p_1, p_3 - p_1, ..., p_{n-1} - p_1)$, then *n* pixels form an *n*-1 dimensional simplex whose volume is written as:

$$V_n = V(p_2 - p_1, p_3 - p_1, ..., p_{n-1} - p_1) = \frac{1}{(n-1)!} \sqrt{\left| A_{n-1}^T A_{n-1} \right|} \tag{7}$$

where $|\cdot|$ stands for the determinant operation.

2.2 SVA procedure

The SVA procedure takes the following steps:

1) Find the two points which forms the largest distance among all the samples. These two points are the simplex's vertices (or endmembers);

2) Find the third point (endmember) which forms a triangle with the previous two points and has the maximum volume.

3) Find out the fourth point (endmember) which forms a triangular pyramid with the previous 3 points and has the maximum volume.

4) Repeat the above steps until all the endmembers (n>4) are found.

3. GPU IMPLEMENTATION AND RESULT

Our implementation and testing are based on an environment with a dual-core 1.62GHz Intel Pentium E5300 CPU and an NVidia GTX 560 1.62 GHz GPU with 336 CUDA cores. The operating system is Linux Ubuntu v10.10. In our experiments we have used a real hyperspectral scene collected by the AVIRIS instrument over San Diego, California. The scene comprises 224 spectral bands between 0.4 and 2.5 μm. The portion used in our experiments corresponds to a 400×400-pixel subset. The image size is 400 x 400 x 224.

3.1 Initial implementation

Our initial GPU implementation of the SVA algorithm is depicted in Fig.1. The major part of this parallel implementation is an iteration operation to computing each endmember in a sequential manner. This iteration itself is divided into 4 stages. Except for the third stage which uses 2 GPU kernels, each stage uses one GPU kernel.

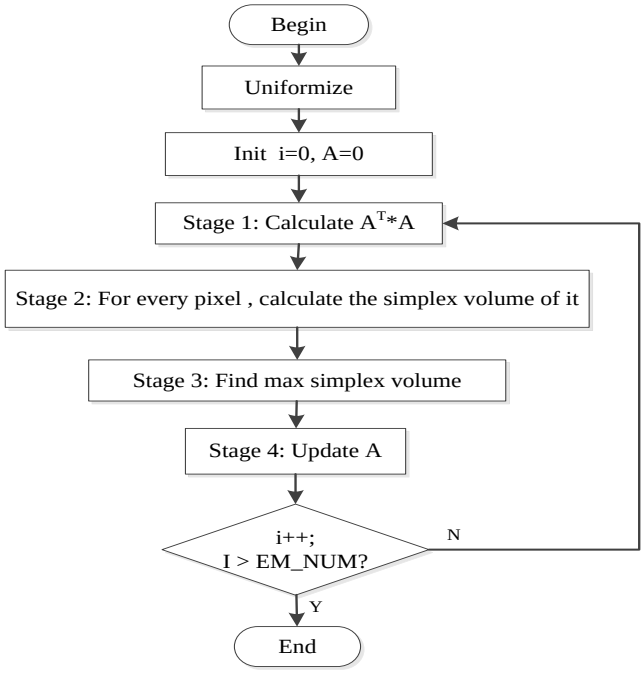

Fig.1. Flow chart of the GPU-based SVA implementation

To elaborate the implementation details, the allocation of the global memory is shown in Fig. 2, where EM_NUM is the number of endmembers to extract; DIM is the number of bands in the image; and SAMPLE_NUM equals the total number of pixels in the image. The first stage of the iteration can be deemed as a matrix multiplication operation. This kernel multiplies matrix *A* by its matrix transpose. It was done on the GPU by transferring each row in matrix *A* into the shared memories, synchronizing the threads, and then multiplying all the corresponding elements. The results of the multiplication are stored linearly in the last row of matrix dev_aTxa, which is allocated in the global memory. We store

the results in this sophisticated form because it brings convenience for the second step: calculation of the simplex volume.

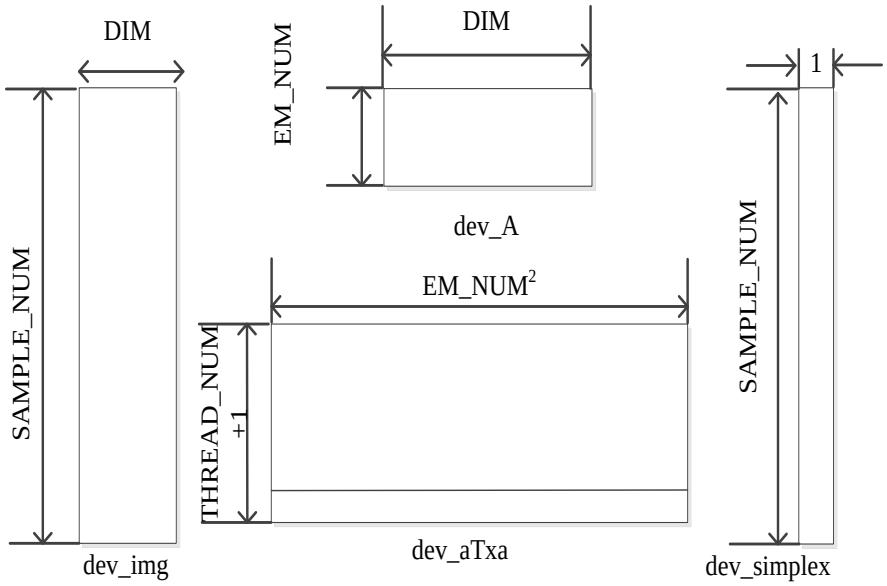

Fig. 2. Allocation of major variables in the global memory

In the simplex volume calculation stage, we need to calculate a simplex volume for each pixel respectively in a parallel manner. So a large number of matrix multiplications may be invoked. For instance, to construct the matrix in Eq. (7), we need to attach each spectrum of the hyperspectral image to the last row of matrix A, and then multiply A by its matrix transpose A^T, as depicted in Fig.3. These myriads of multiplications would consume a lot of computing time and curtail the total speedup of the algorithm if they are not optimized. From Eqs. (3)-(5), we see that the matrix G_m is actually an m-order principal minor of G_{m+1}. So we only need to calculate the (m+1)-th row and (m+1)-th column when we calculate G_{m+1}, as illustrated in the shaded parts of Fig. 3.

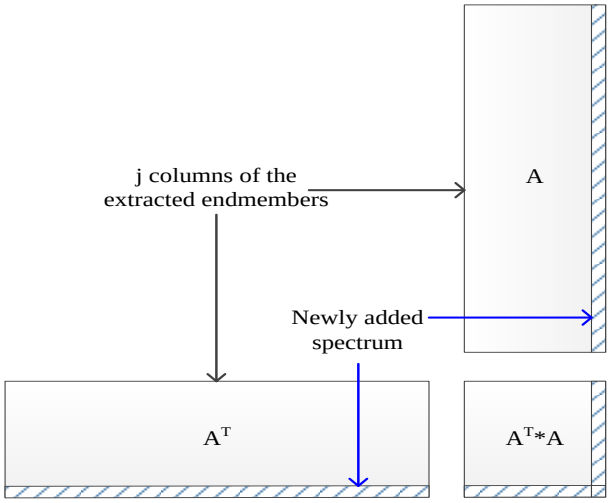

Fig.3. The newly added spectrum is attached to the last row of matrix A, and then multiply A by its matrix transpose A^T.

Knowing that the shaded part will differ with a new pixel, a delicately designed data structure to store the G matrixes can help reduce the computing time significantly, as depicted in Fig. 4.

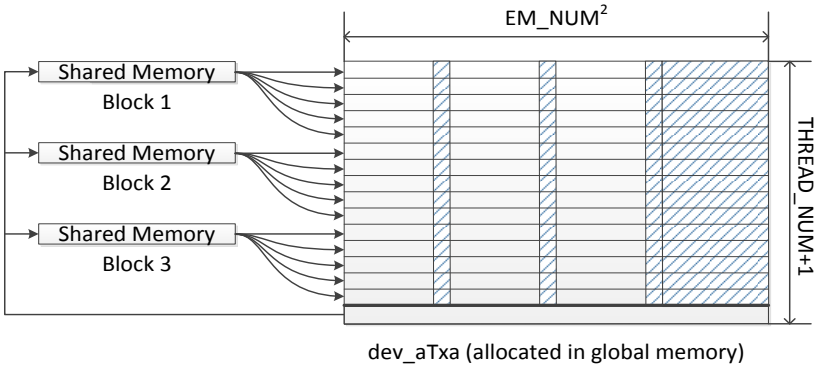

Fig.4. Tranfer of dev_aTxa from the shared memory to the global memory.

The solution is as follows. We allocate in the global memory a large matrix called dev_aTxa, whose width is the square of the number of endmembers expected to be extracted; and whose height equals the total number of threads to be invoked. We then attach an extra row as the last line of the matrix, so that this matrix has THREAD_NUM+1 rows. As we've shown in the first stage, a square matrix *G* is stored row by row in the last line of dev_aTxa. What the kernel in the first step does is to copy the last line into the shared memories of each block. Then, it establishes THREAD_NUM threads, flushing the data in shared memories into the corresponding rows in the matrix dev_aTxa. By utilizing the shared memory, it avoids memory access to the slower global memory (Fig.4). The following process is a circulation, during which all the threads calculate the matrix *G* in batches for all pixels. At this moment, instead of calculating all the elements of the matrix *G*, each thread only calculates the last line and the last row of the matrix *G*. Elements in *G* are directly modified in the corresponding rows of dev_aTxa located in the shaded part in Fig. 4. Then each thread follows Eq. (7) to compute the simplex volume for each pixel. To mitigate the computation loads and accelerate the algorithm, we omitted the computation of factorial and it will not affect the endmember selection. The numerical algorithm chosen in calculating the determinant is based on the Complete Gaussian Pivoting Elimination, and is implemented as a device function in the parallel algorithm. After the calculation, each thread transfers the simplex volume value of the corresponding pixel to dev_simplex, a 1-dimentional vector in the global memory.

The 3rd stage is to find the maximum volume among all the simplex volumes. Fig. 5 depicts how the two kernels in the 3rd stage collaborate to find out the largest value in the dataset. By parallel reduction operations conducted by threads in each block, the first kernel extracts the block-level largest value and the corresponding index, and then writes them back to the header bytes in dev_simplex. Then the second kernel further processes these block-level largest values by another parallel reduction operation. Eventually, the largest values along with its index are extracted from the whole dataset. The correlating spectrum is then deemed as the endmember. At the last stage, this spectrum is written to dev_A. Table 1 shows the CPU and GPU times for the initial SVA implementation. The data transfer time between CPU host and GPU device are included. As seen, the speedup increases with the number of endmembers involved.

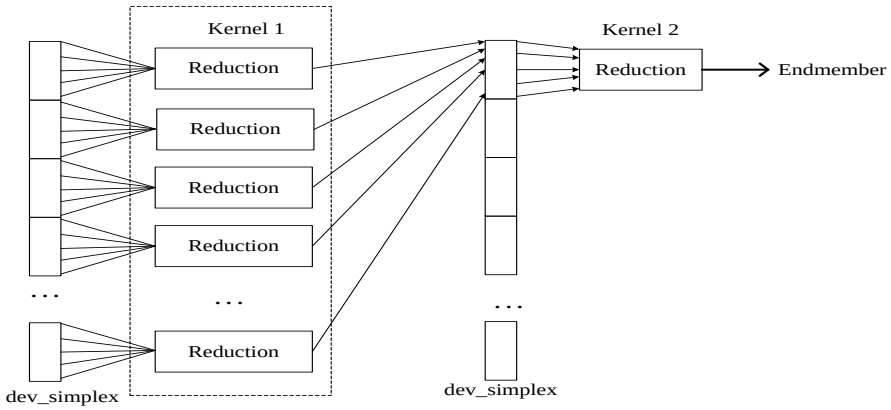

Fig.5. Two-stage parallel reduction

Table 1. Processing time (seconds) and speedups measured for the CPU and GPU implementations of the SVA algorithm

Endmembers	CPU	GPU	Speedup
1	0.41	0.22	1.86x
3	4.45	0.38	11.71x
5	16.97	0.67	25.33x
10	114.8	2.33	49.27x
16	446.85	7.19	62.15x

3.2 Improved implementation

In order to improve the speedup, the program needs to make more use of registers and the shared memory. The 5-kernel structure inhibits utilization of registers and the shared memory. To overcome this issue, we merged the 5 kernels into one single kernel. As a result, the matrix dev_aTxa and dev_simplex are rearranged to store in the shared memory and registers of each block.

The reference matrix, *i.e.* the last line of dev_aTxa, is now stored in the shared memory of each block. Each thread in a block keeps its matrix G in the corresponding registers. To calculate the n-th dimension simplex volume values, each thread first copies the elements of matrix G_{n-1}, which remains the same in the matrix G_n for every pixel, from the shared memory to registers. Instead of access to the slower global memory, we can now calculate determinants and simplex value of each block on the faster memories in a single block. To store the results generated by each thread, we need to allocate two one-dimensional matrixes in the shared memory to storing the simplex volume values and the indexes of the corresponding pixel, respectively. The index indicates the number of the pixel in the hyperspectral image, by which we can access the selected pixel in the image and extract it as an endmember.

In the third stage, a parallel reduction operation is carried in each block to find the largest simplex value and its corresponding index which are then transferred to the global memory where another subsequent parallel reduction is performed to find the largest simplex value, which is the endmember to extract. After the fourth stage that updates the reference matrix in the shared memory of each block, the simplex volume value of the endmember, together with its index, are transferred back to the host memory. Table 2 shows the CPU and GPU times for the improved SVA implementation. The data transfer time between CPU host and GPU device are included. As seen, the speedup increases with the number of endmembers involved. The speedups from the improved implementation in Table 2 are significantly better than the counterparts from the initial implementation in Table 1. This shows the advantage of using more registers and the shared memory over the global memory. It is important to note that the GPU versions of SVA provide exactly the same results as its CPU version.

Table 2. CPU and GPU processing time (seconds) and speedups for the improved implementations of the SVA algorithm.

Endmembers	CPU	GPU	Speedup
1	0.41	0.096	4.27x
3	4.45	0.234	19.02x
5	16.97	0.482	35.21x
10	114.8	1.606	69.16x
16	446.85	3.970	112.56x

4. CONCLUSIONS

In this paper, a GPU-based simplex volume algorithm for hyperspectral endmember extraction was implemented in CUDA. We take advantage of the large-scale parallelism of GPU to significantly accelerate the computation of SVA on the NVidia GeForce 560 GPU. In our experiments we have used a real hyperspectral image collected by the AVIRIS instrument over San Diego, California. The scene comprises 224 spectral bands between 0.4 and 2.5 μm. The portion used in our experiments corresponds to a 400×400-pixel subset. The time for computing a simplex volume increases with the number of endmembers. We show the advantage of using more registers and the shared memory over the global memory. The GPU versions of SVA provide exactly the same results as its CPU version. Experimental results show that the proposed GPU-based SVA achieves a significant 112.56x speedup for extracting 16 endmembers, as compared to its CPU-based single-threaded counterpart.

REFERENCES

[1] X. Geng, Y. Zhao, F. Wang, and P. Gong, "A new volume formula for a simplex and its application to endmember extraction for hyperspectral image analysis." International Journal of Remote Sensing, Vol. 31, No. 4, pp. 1027-1035, 2010.

[2] A. Plaza, D. Valencia, J. Plaza, and C.-I Chang, "Parallel implementation of endmember extraction algorithms from hyperspectral data," IEEE Geoscience and Remote Sensing Letters, Vol. 3, pp. 334-338, 2006.

[3] I. K. Park, N. Singhal, M. H. Lee, S. Cho, and C. W. Kim, "Design and performance evaluation of image processing algorithms on GPUs," IEEE Transactions on Parallel and Distributed Systems, Vol. 22, pp. 91-104, 2011.

[4] S. Hong and H. Kim, "An integrated GPU power and performance model," Proceedings of the 37th Annual International Symposium on Computer architecture, pp. 280-289, 2010.

[5] P. Harish and P. Narayanan, "Accelerating large graph algorithms on the GPU using CUDA," High Performance Computing–HiPC 2007, pp. 197-208, 2007.

[6] S. Sánchez, A. Paz, G. Martin, and A. Plaza, "Parallel unmixing of remotely sensed hyperspectral images on commodity graphics processing units," Concurrency and Computation: Practice and Experience, vol. 23, pp.1538-1557, 2011.

[7] S. Ryoo, C. I. Rodrigues, S. S. Baghsorkhi, S. S. Stone, D. B. Kirk, and W.-m. W. Hwu, "Optimization principles and application performance evaluation of a multithreaded GPU using CUDA," Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp. 73-82, 2008.

[8] F. Lu, J. Song. "Survey of CPU / GPU Synergetic Parallel Computing". *Computer Science*, Vo1.38, No.3, pp.5-9, Mar 2011.

[9] A. Plaza, Q. Du, Y.-L. Chang. "High Performance Computing for Hyperspectral Remote Sensing". IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, Vol. 4, No. 3, pp.529-543, Sep 2011.

[10] C. A. Lee, S. D. Gasster, A. Plaza, C.-I Chang, and B. Huang, "Recent developments in high performance computing for remote sensing: A review," IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, Vol. 4, No.3, pp. 508-527, Sep 2011.