Parallel RHSEG for Hyperspectral Image Analysis Using GPUs

Mahmoud Ahmed Hossam, Hala Mousher Ebied,
Mohamed Hasan Abdel-Aziz and Mohamed Fahmy Tolba
Faculty of Computers and Information Science, Ain-Shams University, Cairo, Egypt
Mahmoud.hossam@ gmail.com, hala_mousher@hotmail.com

Abstract

Power consumption and weight are important factors to be considered when building onboard hyperspectral image analysis systems. While power consumption and weight of computer clusters are hardly suitable to onboard processing systems, graphics processing units (GPUs) provide the solution. GPUs are parallel systems which can provide a high computational performance at a lower cost. In this paper, one investigates a preliminary parallel GPU-based implementation of recursive hierarchical segmentation algorithm (RHSEG), which is one of the latest image analysis techniques used by National Aeronautics and Space Administration (NASA). The key aspect of RHSEG is that it combines region growing segmentation, which produces spatially connected region objects, with region object classification, which groups sets of region objects together into region classes. The proposed parallel RHSEG algorithm has been implemented using NVidia's compute device unified architecture (CUDA). This paper shows that implementing the RHSEG algorithm on GPUs achieved an average processing speedup of 1.6 times over the sequential CPU implementation.

Keywords: *Hyperspectral Analysis, clustering, GPU, RHSEG algorithm*

1. Introduction

Hyperspectral imaging is a technique that has gained popularity in remote sensing research, satellite Imaging and aerial reconnaissance. Most applications of this emerging technology require timely responses for swift decisions which depend upon high computing performance. Examples of these applications include target detection of military and defense/security deployment, urban planning and management, risk/hazard prevention and response, including wild-land fire tracking, biological threat detection, monitoring of oil spills and other types of chemical contamination.

Hyperspectral images contain a large number of measured wavelength bands, so they provide plenty of spectral information to identify spectrally unique materials. Hyperspectral image analysis algorithms naturally integrate the wealth of spatial and spectral information contained in the data by treating the input volume as an image cube made up of spatially arranged pixel vectors [1].

The literature contains a lot of research on hyperspectral analysis techniques. However, they can be grouped under these approaches [3]: per-pixel analysis, mixed-pixel analysis and object-based image analysis. Of these major approaches, there exist many supervised and unsupervised classification / segmentation algorithms [4] and spectral mixture analysis algorithms [5]. The choice of what techniques to follow and use depends on what is the objectives of the analysis for this paper, one is interested in object based image analysis (OBIA) techniques, such as RHSEG [6]. RHSEG technique tends to produce [7] [3] better results on high spatial resolution images and more useful information about objects and regions in the image. The technique is currently implemented sequentially for CPUs and in parallel for cluster of CPU processors. However, in this paper, one introduces a GPU implementation of the technique.

Hyperspectral algorithms are naturally suitable for parallelization: either across pixel vectors, or across tasks. Commodity CPU systems (e.g. computer clusters) can be used for the parallelization of these algorithms. However, these systems are generally expensive and difficult to adapt to onboard remote sensing data processing scenarios, in which low-weight and low-power integrated components are needed to reduce mission payload [1]. Since the emergence of programmable graphics processing units (GPUs) as a low-power low-cost platform for high performance computing, GPUs gained the attention of many researches in increasing the number of research fields [8].

Hyperspectral imaging algorithms can benefit from GPU hardware, thus taking advantage of the compact size and relatively low cost of these units. Low-weight integrated components such a GPUs are desirable to reduce payload and data transmission overheads in onboard processing, and to satisfy the high computational requirements needed in many Earth-observing missions.

In this paper, one provides a preliminary GPU-based implementation of recursive hierarchical segmentation (RHSEG), which is a well-known hyperspectral image analysis algorithm used by NASA. The key aspect of RHSEG is that it combines region growing segmentation, which produces spatially connected region objects, with region object classification, which groups sets of region objects together into region classes. The proposed parallel RHSEG algorithm has been implemented, using NVidia's compute device unified architecture (CUDA).

The remainder of the paper is organized as follows. Section 2 gives a survey of the literature and related work regarding one's research. Section 3 explains the idea of RHSEG algorithm. Section 4 describes in detail the parallel implementation of RHSEG algorithm, using GPUs. Section 5 discusses the results provided by both serial and parallel RHSEG implementation from the viewpoint of parallel performance on selected hardware system. Finally, Section 6 concludes with some remarks and provides hints at future work.

2. Related Work

A lot of research work has been done before regarding hyperspectral image analysis. Digital analysis techniques can be divided into two approaches: supervised classification, and clustering or unsupervised classification. Supervised classifiers faces limitations, such as the limited availability of training samples [4] for hyperspectral data; therefore one decided to focus one's work on clustering and unsupervised algorithms.

Unsupervised techniques fall under two main categories: per-pixel analysis and mixed-pixel analysis [3] (or spectral unmixing). Per-pixel analysis is concerned with relating each pixel of the image to only one class or segment. On the other hand, mixed-pixel analysis assumes that each pixel vector is related to multiple underlying materials, so it is concerned with relating each pixel of the image to multiple classes not only one class. Either for per-pixel or mixed-pixel analysis, in many cases, the use of contextual or spatial classifiers provide better classification accuracies [3] [9], such as Markov Random Field-based (MRF) contextual classifier and automated morphological end-member extraction (AMEE).

Recently, with the increase of spatial resolutions of new sensors, object-based image analysis [3] (OBIA) has emerged as a new approach to image analysis. In object-based image analysis, image segmentation is applied to merge pixels into objects and classification is conducted based on the objects, instead of an individual pixel. This new approach is becoming more popular compared to traditional pixel-based image analysis [7] because of its efficiency with high spatial resolution images. Recursive Hierarchical Segmentation RHSEG [6] is an (OBIA) technique used by NASA as a starting point to facilitate moving from pixel-based image analysis to OBIA. The main advantages of RHSEG are: (1) grouping of spatially connected region objects into region classes, and (2) production of a hierarchical set of image segmentations.

There have been many efforts in the literature for implementing analysis techniques for different parallel architecture like FPGAs [10] [11] and computer clusters [12] [13]. Besides, GPU implementations of several methods has been reported [14], such as GPU (AMEE), GPU Pixel Purity Index (PPI), GPU N-FINDR, GPU iterative error analysis (IEA), GPU orthogonal sub-space projection (OSP) and Matlab Hyperspectral Image Analysis Toolbox (HIAT) on GPU [15]. RHSEG has a parallel version for computer clusters [16]. However, this paper is considered the first step towards a GPU implementation of RHSEG.

3. RHSEG Method

RHSEG is a hierarchical segmentation and region growing method. It is defined as a set of several segmentations of the same image at different levels of detail in which the segmentations at coarser levels can be produced from simple merges of regions at finer levels. RHSEG is based on HSWO (Beaulieu, & Goldberg 1989) [17] which can be summarized in three steps:

1- Initialize the segmentation by assigning each image pixel a region label. If a pre-segmentation is provided, label each image pixel according to the pre-segmentation. Otherwise, label each image pixel as a separate region.
2- Calculate the dissimilarity value between all pairs of spatially adjacent regions, find the pair of spatially adjacent regions with the smallest dissimilarity value, and merge that pair of regions.
3- Stop if no more merges are required. Otherwise, return to step (2).

The hierarchical segmentation (HSEG) algorithm (Tilton, 1998) [18] adds a new feature to HSWO, which differs from the latter in one major aspect. As opposed to HSWO, the HSEG algorithm allows for the merging of non-adjacent regions controlled by the "spectral clustering weight" input parameter. Using this parameter, one is free to choose how HSEG merges regions, by only merging adjacent regions, by equally merging spatially adjacent and non-adjacent regions, or finally by mixing the merging of adjacent and non-adjacent regions. HSEG also provides the selection of several dissimilarity functions like Euclidean distance (2-Norm), spectral information divergence (SID), spectral angle mapper (SAM), and band summed mean squared error (BSMSE).

Because HSEG is computationally intensive, the recursive approximation RHSEG has been developed. Figure (1) shows a flowchart for RHSEG method. To completely understand RHSEG, it is also essential to understand the idea of HSEG algorithm. HSEG is a region growing algorithm. Figure (2) shows an outline of its main procedures. HSEG is an iterative region merging process, initialized with every pixel as a region. In each step, dissimilarity value is calculated for each pair of spatially adjacent regions. The pair of regions with smallest dissimilarity value is chosen for merging, and then the new merged region replaces them. This process continues until the desired number of regions (or segments or classes) is reached.

4. Parallel GPU Implementation

For one's preliminary parallel RHSEG implementation, one has some restrictions applied to one's implementation, which are planned to be removed in future work. For this paper, the intention was focused on proving the concept first. These restrictions are (1) we assume that always the "spectral clustering weight" parameter is always 0, so that only spatially adjacent regions are merged in the same class, and (2) we assume that any image is a square and its width or height is an even number or a power of 4 (i.e. 128, 256, 420), so that image padding is not needed. Hyperspectral image is transferred to GPU memory, using CUDA memory copy APIs. Moreover, data structures that hold information for all regions information and corresponding spectral statistics are initialized and then reside in GPU memory till the termination of the GPU execution. In this paper, square root of band sum mean squared error (square root of BSMSE) is used for dissimilarity calculation. This is given between any two regions $\boldsymbol{i}$ and $\boldsymbol{j}$ in an image of $\boldsymbol{B}$ bands by:

$$Square\ root\ of\ BSMSE(i,j) = \sqrt{\frac{n_i n_j}{(n_i + n_j)} \sum_{b=1}^{B} (\mu_{ib} - \mu_{jb})^2} \quad (1)$$

where μ_{ib} and μ_{jb} are the mean values for regions ***i*** and ***j*** in spectral band b, respectively. n_i and n_j are the number of pixels in regions ***i*** and ***j*** respectively. The researcher's GPU implementation is based on distributing the process of dissimilarity calculation on GPU threads. Figure (3) shows the Parallel implementation of RHSEG on GPU.

Each GPU thread is responsible for a corresponding region from the image. For example, if image has 1024 regions, 1024 GPU threads will be initialized, each thread will calculate the dissimilarly value for every adjacent region, and then terminates. After the calculation of all dissimilarly values and all threads terminate, a parallel reduction step is used to find he minimum value. RHEG then continues execution normal. Figure (4) shows the steps of copying image data from CPU to GPU memory. At the deepest level of recursion, the current image is divided into 4 quarters, and then HSEG is executed for each quarter after transferring it to GPU memory, sequentially.

NVIDIA CUDA [19] technology is used for the implementation of GPU RHSEG. Compute device unified architecture (CUDA) is a unified GPU computing platform for any CUDA enabled device. It enables the developer to easily write parallel applications following stream computing fashion, in which single routine called kernel, is executed over multiple data streams in parallel. Scalability issue is maintained automatically by CUDA thread scheduler, which detects the number of available GPU multiprocessors and schedules the desired number of threads to run in parallel over these multiprocessors.

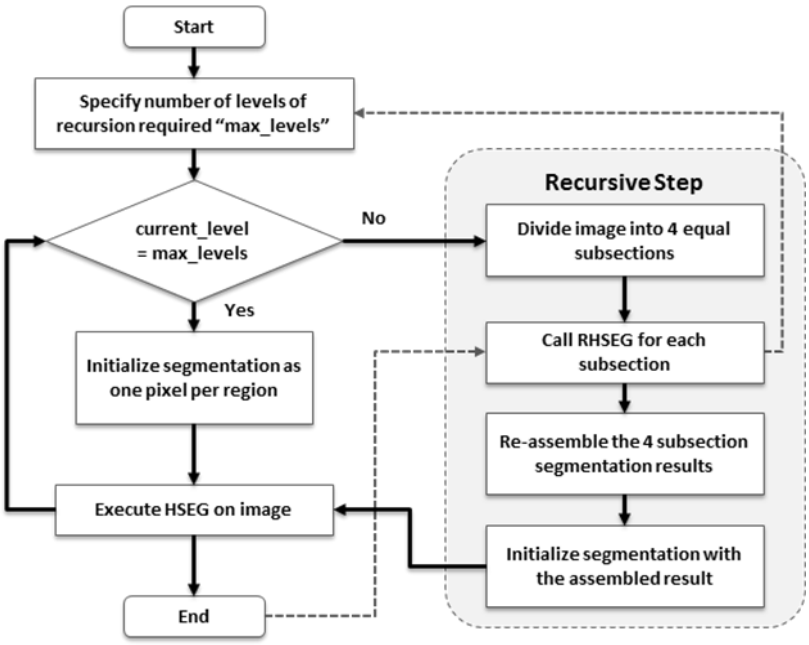

Figure 1: Flowchart of RHSEG method

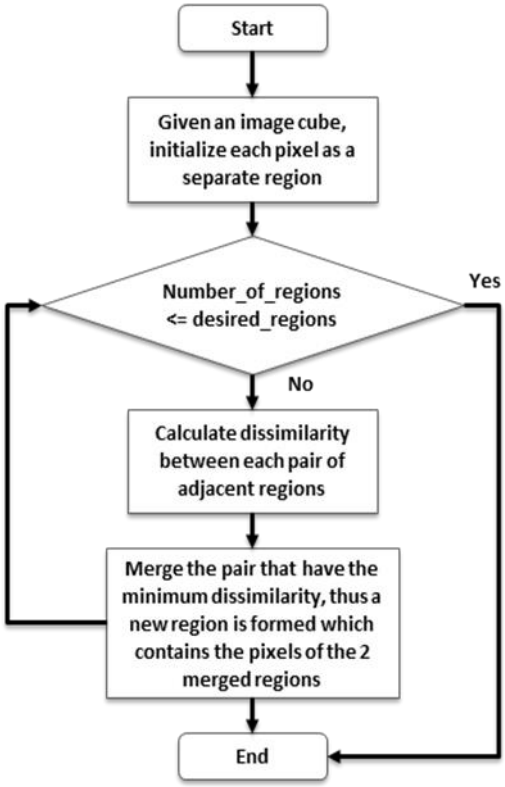

Figure 2: Outline of HSEG method

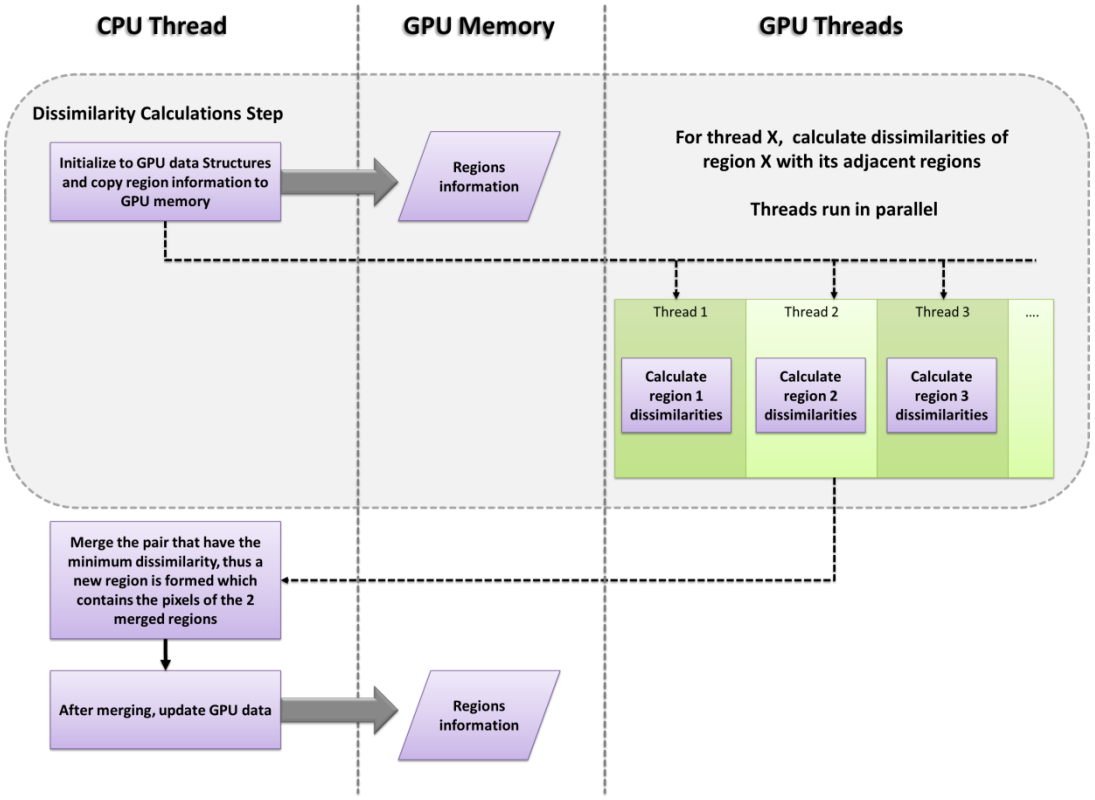

Figure 3: Parallel RHSEG implementation

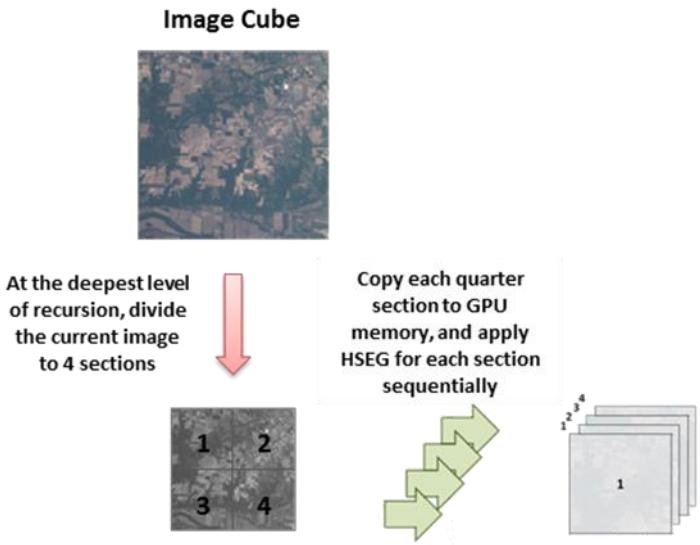

Figure 4: steps of copying image data from CPU to GPU memory.

Figures (5) and (6) show the CPU and GPU code responsible for the calculation of dissimilarity values. Figure (6) shows the CUDA kernel that performs dissimilarly values calculation. In figure (5), method **doStep()** contains the 2 nested loops that measures the dissimilarity for each region/adjacent-region pair. The first loop traverses all regions, the second loop traverses each neighbor region for the current region. For every pair of region-neighbor, dissimilarity is computed by calling **measureDissimilarity()** method.

```
void HSWO::doStep()
{
    int minDissimRegion1Label, minDissimRegion2Label;
    float minDissim = MY_FLT_MAX;
    Region *ptrR1=nullptr, *ptrR2=nullptr, *ptrTempR2 = nullptr;

    for each(auto currentRegion in m_regions)
    {
        for each(int neighborLabel in currentRegion.second->adjacentRegions)
        {
            ptrTempR2 = m_regions[neighborLabel];
            float dissim = measureDissimilarity(currentRegion.second ,ptrTempR2);
            if(dissim < minDissim)
            {
                minDissim = dissim;
                minDissimRegion1Label = currentRegion.first;  minDissimRegion2Label =
                neighborLabel ;
                ptrR1 = currentRegion.second; ptrR2 = ptrTempR2;
            }
        }
    }
}

inline float HSWO::measureDissimilarity(Region* region1, Region* region2)
{
    float temp_sum1 = 0;

    float r1nPixles = region1->pixels_IDs.size();   float r2nPixles = region2->pixels_IDs.
    size();
    float* r1Sums = region1->sumOfPixels.get();     float* r2Sums = region2->sumOfPixels.get
    ();

    for (int band = 0; band < m_nBands; band++)
    {
        m_tempPixelVector3[band] = r1Sums[band]/r1nPixles - r2Sums[band]/r2nPixles;
        temp_sum1 += m_tempPixelVector3[band] * m_tempPixelVector3[band];
    }

    float dissim = sqrt(temp_sum1*((r1nPixles*r2nPixles)/(r1nPixles+r2nPixles)));
    return dissim;
}
```

Figure 5: CPU dissimilarity calculation

In figure (6), method **DeviceCalcAllDissims()** contains GPU kernel launch instruction, that launches parallel kernel method **kernel_compute_dissim** that calculates dissimilarity values for all regions in **m_dev_RegionKeys** and their neighbors in **m_dev_pRegionAdjancencies**. All computed dissimilarities are stored in **m_dev_pDissims** matrix which is later used in the parallel reduction that finds the minimum dissimilarity value.

It is required, after each region merge execution, to update GPU memory by copying data of the new merged region to GPU memory. Although HSEG is executed at all recursion levels of RHSEG, one executes HSEG on GPU only at the deepest level of recursion that is

when the image is no longer divided into 4 sections. Any execution of HSEG at higher recursion level is executed on CPU thread not the GPU, as one found from experiments that at higher recursion levels and with reduced number of regions, CPU implementation matches or surpasses the speed of the GPU implementation.

GPU has several kinds of memory: global memory, shared memory, constant memory, and cache memory in some of the high end models. Shared and constant memory is faster than global memory. These memory types vary in capacity and bandwidth, and can be used in more optimization of applications. However, in this paper, GPU global memory is the only one used.

```
int DeviceCalcAllDissims(int numberOfRegions, unsigned int threadsPerBlock)
{
        int nBlocks = numberOfRegions/threadsPerBlock+1;
        dim3 numberOfBlocks(nBlocks,1); dim3 threadsPerBlockStruct(threadsPerBlock, 1);

        kernel_compute_dissim<<<numberOfBlocks, threadsPerBlockStruct>>>(m_deviceData.
        m_dev_regions_pixel_count
                                                        ,m_deviceData.m_dev_RegionKeys
                                                        ,m_deviceData.
                                                        m_dev_pRegionAdjancencies
                                                        ,numberOfRegions
                                                        ,m_nBands
                                                        ,m_deviceData.m_dev_pRegionMeans
                                                        ,m_deviceData.m_dev_pDissims);
        cudaThreadSynchronize();
        ....
}

__global__ void kernel_compute_dissim(int* regionsPixelCount, int* keysFilter ,int* adj, int
nRegions
, const int nBands, float* means, float* output)
{
    int index = blockIdx.x * blockDim.x + threadIdx.x;
    if(index < nRegions)
    {
        int regionID = keysFilter[index]-1;
        if (regionID == -2) return;

        int offset_regionID = regionID*MAX_REGION_ADJ;
        float r1nPixles = regionsPixelCount[regionID];

        for(int a=0; a<MAX_REGION_ADJ; a++)
        {
            int regionIDadj = adj[offset_regionID+a]-1;
            if (regionIDadj <= -1)  {   break;  }

            float sum_eculidean = 0.0f; float temp;
            float r2nPixles = regionsPixelCount[regionIDadj];

            int band = 0;
            for (; band < nBands; band++)
            {
                temp = means[regionID*nBands+band] - means[regionIDadj*nBands+band];
                sum_eculidean += temp * temp;
            }
            output[offset_regionID+a] = sqrtf(sum_eculidean * ((r1nPixles*r2nPixles)/(
            r1nPixles+r2nPixles))) ;
        }
    }
}
```

Figure 6: GPU dissimilarity calculation

5. Experiments and Results

For the data set, one used the Indian Pines AVIRIS hyperspectral data set that is available for request from NASA JPL website (www.jpl.nasa.gov). Figure (7) shows Indian Pines AVIRIS hyperspectral image. GPU RHSEG is experimented using ddifferent sizes 128x128, 256x256 and 420x420 pixels of the original Indian Pines image.

For any image size, the number of bands is 220 bands. For each image size, data was cropped from the large image, not scaled.

Figure 7: Indian Pines Data Set, the image consists of 220 spectral bands.

The parallel execution of RHSEG algorithm has been tested on NVIDIA GeForce 8800 GT, which consist of 112 processing cores each operating at 1500 MHz, with 512 MB GDDR3 256-bit memory interface and which operates at 900 MHz, that is capable of 57.6 GB/sec memory bandwidth. The sequential execution was tested on Intel Core2 CPU with 2400 MHz, 256 K.B. L1 and 8 M.B. L2 cache memory. Figure (8) shows both classification result and ground truth images.

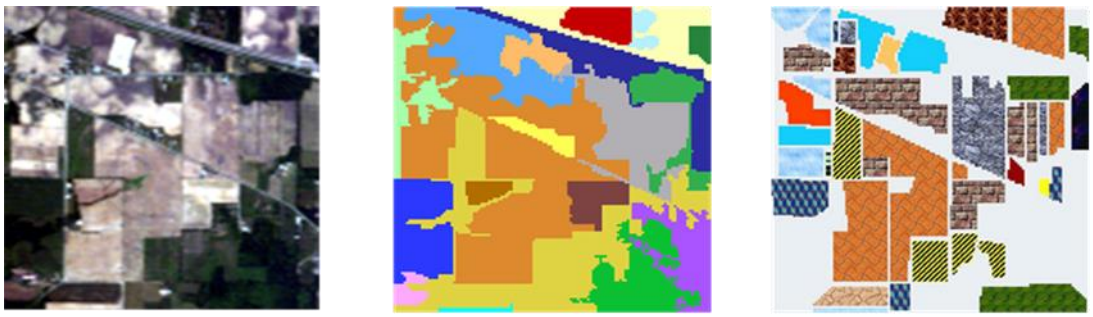

Figure 8: a) Indian Pines Data Set RGB image of size 128x128 pixels, b) the classification result image consists of 20 classes and c) the ground corresponding ground truth image of 16 classes

Table 1 summarizes speedup results. The experiment results show the speedups of RHSEG GPU implementation over CPU implementation. The speedup is computed by dividing CPU running time by GPU running time. GPU running time includes the memory copy time between main memory and GPU memory. For example, for 420x420 image size, the RHSEG CPU execution time was 57750 seconds, while the RHSEG GPU execution time was 36465 seconds. An average speed up of 1.6 times on GPU implementation over the sequential CPU implementation is achieved.

Table 1: Summery of experiments results

Image Size *(size in M.B.)*	**Average GPU Speedup**
128x128 (13 MB)	1.3
256x256 (55 MB)	1.65
420x420 (148 MB)	1.58

6. Conclusions and Future Work

This paper introduced the first step to parallelize RHSEG technique which is a well-known hyperspectral object-based image analysis (OBIA) technique. In this paper, one used the GPU parallel hardware for the parallel implementation of RHSEG technique. GPU has lower weight and power consumption compared with computer clusters. The GPU implementation was tested on NVIDIA 8800 GT board with 112 processing cores using CUDA platform. The main idea presented in this paper is the parallelization of the dissimilarity calculation step in RHSEG algorithm because of the suitability of parallel execution of these calculations. Other parts of the algorithm are executed on CPU thread. An average speed up of 1.6 times over the sequential CPU implementation is achieved.

Speedup rates reported in this paper still do not meet the expectations. Many improvements are needed to enhance the performance of GPU RHSEG in terms of running time, such as using GPU shared and constant memory for faster memory access rates, minimizing memory copies from CPU to GPU at each step when updating changed data structures, or the execution of HSEG on divided image sections in parallel, thus not only parallelizing dissimilarity calculations for each image section one by one, but also executing several image sections at once in parallel on the GPU.

In the future, one intends to experiment a hybrid multithreaded CPU/GPU and GPU cluster implementations for even more performance optimization of RHSEG technique.

References

[1] Setoain, J., Prieto, M., Tenllado, C. and F. Tirado, GPU for parallel on-board Hyperspectral image Processing. International Journal of High Performance Computing Applications, Vol. 22, no. 4, pp. 424-437, 2008.

[2] Jong, S. D. and Meer, F. V. D., Imaging spectrometry: Basic Principles and Prospective Applications, Kluwer Academic. 2002.

[3] Lu, D., and Weng, Q., A Survey of Image Classification Methods and Techniques For Improving Classification Performance. International Journal of Remote Sensing, Vol. 28, no. 5, pp.823-870, 2007.

[4] Benediktsson, J. A., Boardman, J. W., Brazile, J., Bruzzone, L., Camps-Valls, G., Chanussot, J., Fauvel, M., Gamba, P., Gualtieri, A., Marconcini, M., Tilton, J. C. and Trianni, G., Recent advances in techniques for hyperspectral image processing, Remote Sensing of Environment, Vol. 113, Supplement 1, pp. S110–S122, September 2009.

[5] Keshava N., A Survey Of Spectral Unmixing Algorithms. Lincoln Laboratory Journal, Vol. 14, no. 1, pp. 55-78, 2003.

[6] Tilton, J. C, Method for Recursive Hierarchical Segmentation By Region Growing And Spectral Clustering With A Natural Convergence Criterion. Disclosure of Invention and New Technology: NASA Case No. GSC 14,328-1, 2000.

[7] Blaschke, T., Object Based Image Analysis for Remote Sensing. Journal of Photogrammetry and Remote Sensing (ISPRS), Vol. 65, no. 1, pp. 2-16. 2010.

[8] Owens, J. D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A. E., and Purcell, T. J., A Survey of General- Purpose Computation on Graphics Hardware. Computer Graphics Forum, Vol. 26, no. 1, pp. 80-113. 2007.

[9] Plaza, A., Martinez, P., Perez, R. and Plaza, J., Spatial/Spectral Endmember Extraction by Multidimensional Morphological Operations. IEEE Transactions on Geoscience and Remote Sensing, Vol. 40, no. 9, pp. 2025-2041, 2002.

[10] Plaza, A. and Chang, C.-I., Clusters versus FPGA for Parallel Processing of Hyperspectral Imagery. International Journal of High Performance Computing Applications, Vol. 22, no. 4, pp. 366-385, November 2008.

[11] Guilhermino, A., Filho, S., Frery, A. C., Araújo, C. C. D., Alice, H., Cerqueira, J., Loureiro, J. A., de Lima, M. E., Oliveira, M. S., Horta, M. M., Hyperspectral Images Clustering on Reconfigurable Hardware using The K-Means Algorithm K-Means Clustering. 16th Symposium on Integrated Circuits and Systems Design (SBCCI), pp.99, 2003.

[12] Plaza, A., Valencia, D., Plaza, J. and Martinez, P., Commodity Cluster-Based Parallel Processing of Hyperspectral Imagery. Journal of Parallel and Distributed Computing, Elsevier, Vol. 66, no. 3, pp.345-358. 2006.

[13] Plaza A., Chang, C.-I, Plaza, J. and Valencia, D., Commodity Cluster and Hardware-Based Massively Parallel Implementations of Hyperspectral Image Analysis Algorithms. Journal of Parallel and Distributed Computing, Vol. 66, pp. 345−358. 2006.

[14] Sanchez, S. and Plaza, A., A Comparative Analysis of GPU Implementations of Spectral Unmixing Algorithms. High-Performance Computing in Remote Sensing, Proceedings of the SPIE, Vol. 8183, pp. 81830E-81830E-10, 2011.

[15] Rosario-Torres, S. and Velez-Reyes, M., Speeding up the MATLAB™ Hyperspectral Image Analysis Toolbox using GPUs and the Jacket Toolbox. First Workshop on Hyperspectral Image and Signal Processing Evolution in Remote Sensing, pp.1-4, 2009.

[16] Parallel RHSEG performance results, retrieved from NASA featured technologies - recursive hierarchical segmentation on Data Analysis website: http://ipp.gsfc.nasa.gov/ft_tech_rhseg.shtm.

[17] Beaulieu, J. M. and Goldberg, M., Hierarchy in Picture Segmentation: A Stepwise Optimization Approach. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 11, no. 2, pp. 150-163. 1989.

[18] Tilton, J., Image Segmentation by Region Growing and Spectral Clustering With a Natural Convergence Criterion. Geoscience and Remote Sensing Symposium Proceedings (IGARSS 98), Vol. 4, pp.1766 – 1768, 1998.

[19] Kirk, D., NVIDIA CUDA Software and GPU Parallel Computing Architecture. Proceedings of the 6th International Symposium on Memory Management (ISMM 07), pp.103-104, 2007.