J Real-Time Image Proc (2018) 14:413–432
https://doi.org/10.1007/s11554-014-0464-4

ORIGINAL RESEARCH PAPER

Accelerated hyperspectral image recursive hierarchical segmentation using GPUs, multicore CPUs, and hybrid CPU/GPU cluster

M. A. Hossam · H. M. Ebied · M. H. Abdel-Aziz · M. F. Tolba

Received: 6 March 2014 / Accepted: 14 October 2014 / Published online: 2 November 2014

Abstract Rescue missions, military target detection, hazard prevention, and other time-critical remote-sensing applications require real-time or autonomous decision making and onboard processing capabilities. Thus, light-weight, small size, and low-power-consumption hardware is essential for onboard real-time processing systems. With the increasing need for dimensionality, size, and resolution of hyperspectral sensors, additional challenges are posed upon remote-sensing processing systems, and more capable computing architectures are needed. Graphical processing units (GPUs) emerged as promising architecture for light-weight high-performance computing. In this paper, we propose accelerated parallel solutions for the well-known recursive hierarchical segmentation (RHSEG) analysis algorithm, using a GPU, hybrid multicore CPU with GPU and hybrid multicore CPU/GPU clusters. RHSEG is a method developed by the National Aeronautics and Space Administration, which is designed to provide more useful classification information with related objects and regions across the hierarchy of output levels. The proposed solutions are built using the NVidia's compute unified device architecture and Microsoft's C++ Accelerated Massive Parallelism (C++ AMP) and are tested using NVidia GeForce hardware and Amazon Elastic Compute Cluster (EC2). The achieved speedups by parallel solutions compared with CPU sequential implementations are 21× for parallel single GPU and 240× for hybrid multinode computer clusters with 16 computing nodes. The energy consumption is reduced to 74 % when using a single GPU, compared to that for the equivalent parallel CPU cluster.

Keywords Multicore · GPU · RHSEG algorithm · Clustering · Onboard processing

1 Introduction

Remote-sensing applications whether airborne or space borne provide huge benefits for important missions in a wide spectrum of fields ranging from scientific research, security and defense, agriculture, civil services, environmental studies, and exploration. Some of these applications are time critical and require real-time or autonomous decision making, such as search and rescue missions, target detection of military and defense deployment, risk or hazard prevention, wild land fire tracking, biological threat detection, and the monitoring of chemical contamination such as oil spills.

Therefore, onboard processing is needed such that a significant portion of remote-sensing data analysis is carried out on the vehicle, allowing for optional autonomous actions before sending data and feedback to the ground control station. The goal of the remote-sensing mission is always toward achieving smaller size, lower cost, flexible, and high computational power onboard processing. Instead

M. A. Hossam · M. H. Abdel-Aziz
Basic Sciences Department, Faculty of Computer and Information Sciences, Ain Shams University, Abbasia, Cairo 11566, Egypt
e-mail: mahmoud.hossam@gmail.com

M. H. Abdel-Aziz
e-mail: mhaziz67@gmail.com

H. M. Ebied (✉) · M. F. Tolba
Scientific Computing Department, Faculty of Computer and Information Sciences, Ain Shams University, Abbasia, Cairo 11566, Egypt
e-mail: hala_mousher@hotmail.com; halam@fcis.asu.edu.eg

M. F. Tolba
e-mail: fahmytolba@cis.asu.edu.eg

of storing and forwarding all the captured images from onboard sensors, data processing can be performed onboard prior to downlink, resulting in the reduction of a communication bandwidth with simpler subsequent computations to be performed at the ground stations [1].

A recent development in remote sensing is the introduction of hyperspectral imaging [2], in which images contain a large number, usually hundreds of measured wavelength bands, so that they provide plenty of spectral information to identify spectrally unique materials. Thus, image analysis algorithms can benefit from the wealth of spatial and spectral information leading to more accurate analysis of remote-sensing images. In turn, this wealth of data posed new challenges of high-dimensional data and intensive time-consuming computations. These high computational requirements, plus the fact that these systems will continue toward increasing their spatial and spectral resolutions, compelled researchers to investigate powerful computing platforms, which can efficiently handle high computational demands.

High-performance computing (HPC) can be achieved using high clock speed sequential processors or by using parallel computing platforms. However, the manufacturers of CPU chips are now faced by the clock wall of processing units, and the researchers found that the future of HPC depends on parallel computing rather than increasing the clock speed of single processing units [3].

Graphical processing units (GPUs) have emerged recently as a promising platform for HPC and captured the attention of researchers in a lot of research areas [4]; the computational power in (GFLOPS) of GPUs has grown much faster than the CPUs power over the last decade. The important benefits besides the computational power of GPUs are the small size, lightweight, and low-power consumption. This makes GPUs a highly desired platform for remote-sensing applications like satellite imaging and aerial reconnaissance [5]. Many hyperspectral analysis techniques have been implemented on parallel platforms, either on computer clusters or GPUs as we will discuss later.

Remote-sensing digital image analysis is a rich and vast research field that contains many kinds of pattern recognitions and statistical analysis methods. The choice of a suitable analysis method for a certain task is largely dependent on the nature of the desired scenario and domain. Hyperspectral analysis methods can use the spectral information only or both the spatial and the spectral information of the image. The spectral methods treat pixel values as individual unrelated sets of spectral intensities with no particular spatial arrangement. The spatial–spectral methods take into account the pixel arrangement and the contextual entities in the image. The research of hyperspectral image analysis is increasingly moving toward spatial–spectral methods because of the importance of incorporating spatial and spectral aspects of data simultaneously, which has been recognized by many researchers in the field [6–8].

Hyperspectral analysis methods can be grouped under three main approaches [6]: per-pixel analysis, mixed-pixel analysis, and object-based image analysis (OBIA) [9]. Of these major approaches, there exist many classification/segmentation algorithms [10, 11] including spectral mixture analysis algorithms [12]. With the increase of spatial resolutions of new sensors, OBIA has emerged as a promising approach to image analysis due to its efficiency with high spatial resolution images and the production of useful information about image classes and objects.

In the light of the above mentioned findings, it is logical to focus on spatial–spectral and OBIA methods for better and more useful analysis and classification results. In addition, unsupervised analysis is encouraged because of the limited training samples, the difficulty of obtaining ground truth data in remote sensing, and the need for automated responses for onboard processing [11]. Therefore, this work is concerned with unsupervised classification and clustering/segmentation approaches with spatial–spectral and object-based analysis. Recursive hierarchical segmentation (RHSEG) [13] is a well-known hyperspectral spatial–spectral OBIA method developed and used by the National Aeronautics and Space Administration (NASA). Hierarchial segmentation (HSEG) [13] has two main advantages: (1) it provides more accurate region boundaries; and (2) it produces a hierarchical set of image segmentations with different detail levels, i.e., from fine to coarse grain. However, hierarchical clustering methods are computationally intensive, especially when used with high-dimensional data. In order to meet these computational challenges and provide suitable solutions for onboard processing, suitable parallel solutions are needed.

In [14], a parallel/distributed implementation of the RHSEG is presented using GPUs, multicore CPUs, and network clusters; thus, shared memory architecture and distributed memory architectures are combined cooperatively and seamlessly. The RHSEG algorithm is implemented on both GPU clusters and hybrid CPU/GPU clusters. The speedup results were compared with the execution of an RHSEG algorithm on a single node hybrid CPU/GPU and on a parallel GPU. The fundamental idea of parallelizing and accelerating the RHSEG is to distribute the most intensive dissimilarity calculation part among GPU threads and partitioning the input image into sections, sending each section into multicore CPU threads and cluster computing nodes. The GPU platform used for the proposed solution was NVidia's Compute Device Unified Architecture (CUDA) [15]. In this paper, a new two-dimensional GPU kernel is introduced, which achieves

higher speedups over [14], and a new GPU platform implementation using Microsoft C++ Accelerated Massive Parallelism (C++ AMP) [16] is introduced. Also in this paper, power and energy consumptions for the proposed solutions are investigated. The software platforms that are used for multicore CPUs and distributed clusters are the QtConcurrent and the QtNetwork libraries by Digia [17], and the proposed cluster solution is executed by means of Amazon Elastic Compute Cloud cluster [18].

The remainder of the paper is organized as follows: Sect. 2 surveys the research work of related hyperspectral analysis methods and approaches. Section 3 explains the RHSEG algorithm. The hybrid GPU/CPU implementation of the RHSEG algorithm is described for the single computing node in Sects. 4 and 5 and for cluster implementations in Sects. 6. Section 7 shows experimental results and discussions. Finally, Sect. 8 concludes the results of this paper and mentions suggested future work.

2 Related work

There are many kinds of methods in the literature for unsupervised hyperspectral image analysis based on clustering or segmentation. These methods can be categorized under several main approaches [19]: partitional clustering, Watershed transformation for segmentation, Graph methods for segmentation, and hierarchical clustering. Partitional clustering is a classical approach which is based on dividing the input image to arbitrary clusters and iteratively assigning the data points to these clusters using an error criterion measurement like the squared error. The Watershed transformation for segmentation [20] uses the watershed contours that are generated from the input image as a boundary map for the segmentation process. The input image is considered as a topographic height map of pixels of intensity values, and the output watershed image represents the high boundaries around low points (or local minima areas) of the height map. The Watershed algorithm is originally calculated for gray scale single band images, but in [21, 22] it was adapted for multichannel images. Graph clustering methods [23] represent the image as a weighted undirected graph, where the pixels or the groups of pixels are the graph nodes and the weighted edges are the dissimilarity between adjacent pixels. Afterward, the graph is partitioned into smaller sub-graphs or trees that represent separate clusters. The partitioning process is carried out based on different criteria, like deleting edges with the largest dissimilarity. Finally, the hierarchical clustering method starts by assigning clusters to individual pixels, and then merges these pixels iteratively based on similarity measures until the desired number of clusters are reached. This approach generates multiple levels of classifications from fine grain close to individual pixels, to coarse grain at the final clusters.

Many methods exist in the literature for each of these four classification approaches. The segmentation techniques can be grouped into three classes working in a spatial domain, spectral domain, or combination of spatial–spectral domains [24]. A well-known example of partitional clustering is the Iterative Self-Organizing Data Analysis Algorithm (ISODATA) [25]. ISODATA is a spectral clustering method that does not incorporate spatial information. The advantage of ISODATA is the low computational complexity; however, these methods are sensitive to initial cluster-generation methods. The Watershed algorithm was used as a pre-segmentation step for enhancing the classification output [25]. In [26], Plaza developed unsupervised image classification/segmentation methodology by extending the watershed transformation to hyperspectral image processing. He compared this technique to a standard hyperspectral unsupervised classification algorithm; the ISODATA algorithm. Watershed transformation has low computation complexity compared to other segmentation techniques. However, Watershed transformation is known for over-segmentation of output regions and sensitivity to image noise [27]. Beucher [28] introduced a new algorithm called the waterfall algorithm to overcome the over-segmentation problem that usually comes with the watershed transformation.

An example of the graph clustering method is the minimum spanning forest (MSF) [29]. In this method, a graph G representing the initial classification of the image is generated by assigning each pixel to a graph vertex, and each edge connects couple of vertices and a given weight. This weight indicates the dissimilarity between these two vertices. A MSF is then calculated using the graph, and the resulting subtrees are used as regions for the final classification map. This method in combination with appropriate segmentation algorithms produces highly accurate results. However, depending on the initial classification method, if some regions are missed due to inappropriate classification parameters, these regions will be lost in the final classification map. The hierarchical segmentation (HSEG) [13] is a hierarchical clustering method in which each pixel is considered as a separate region, and iteratively HSEG merges these regions until the desired number of clusters is reached. Each HSEG contains two possible merges; adjacent and non-adjacent regions merge. HSEG produces accurate region boundaries and high classification accuracy but has high complexity and memory requirements. To address these challenges, a recursive approximation called the Recursive HSEG (RHSEG) [13] was developed. Plaza et al. [11] used the RHSEG clustering for unsupervised classification, which produced highly accurate classification results.

Several comparative studies have been conducted to compare the analysis methods and techniques to each other in the literature. For instance, Fauvel et al. [24] studied and compared the Watershed, HSEG, and MSF classifications for different hyperspectral datasets. HSEG-based classification produced better overall classification results compared to the Watershed-based method in all datasets—the best overall accuracy in one of the datasets compared to the MSF-based method and then the second best in the other dataset. A multiple classifier incorporating all three methods achieved the best overall accuracy for all datasets.

Hyperspectral clustering methods are computationally intensive due to the high data dimensionality. Therefore, suitable parallel solutions are needed to overcome computational and memory requirement challenges. Hyperspectral image analysis algorithms have been implemented on various and different parallel architectures, parallel multiprocessors, heterogeneous and homogeneous networks of distributed computers, and specialized hardware such as field-programmable gate arrays (FPGAs) and GPU hardware architecture. For example, ISODATA was parallelized using a Thunderhead CPU cluster [30] with 9× speedup using 16 processing nodes and also parallelized using hybrid CPUs and GPUs [31] using a Kenneland supercomputer [32] with hybrid nodes of Intel Xeon E5 8-core CPUs and NVidia M2090 GPUs. They achieved a speedup of 2.3× for distributed parallel GPUs versus distributed parallel CPUs using 36 nodes. ISODATA was also parallelized on a single NVidia Kepler K20 GPU in [33] achieving 45× versus sequential CPU implementation for 50 clusters of an output image. The Watershed-based classification [26] was parallelized using a Thunderhead cluster achieving 13× speedup using 16 nodes and 170× speedup using 256 nodes.

The RHSEG was parallelized using cluster CPUs and GPUs in [11] and [14], respectively. In [11] a homogenous Thunderhead Beowulf cluster at NASA's Goddard Space Flight Center is used to accelerate the RSHEG. The Beowulf cluster [34] is composed of dual 2.4-GHz Intel Pentium 4 Xeon nodes, 256-GB DDR memory (with 1.0 GB of main memory available per CPU) and connected with 2.2-GB/s fiber interconnection system. The speedups achieved for these algorithms were 13× using 16 CPU nodes and 82× using 256 CPU nodes. In [14], the GPU/RHSEG is implemented by means of a 1-D dissimilarity calculation kernel that processes every region's dissimilarity calculation with all other regions within a single thread per region. In addition, a hybrid multicore CPU/GPU cluster was used for cooperative processing between CPU cores and the GPU for different image sections. Using a single NVidia GeForce 550 Ti board, an average speedup of 3.5× was achieved over sequential Intel Core i5 CPU implementations. With the use of a hybrid 8-core Intel Xeon X5570 operator and a NVidia Tesla M2050 GPU, additional average speedup of up to 6× was achieved, using multicores cooperatively besides the GPU. Finally, using 16 node hybrid CPU/GPU clusters, each having a single GPU, resulted in a total of 112× speedup.

3 RHSEG method

Hierarchical image segmentation is 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 the finer levels [35]. Tilton [13] has developed a hierarchical segmentation and region growing (HSEG) method that is a combination of region growing and spectral clustering. The HSEG algorithm adds a new feature to the hierarchical step-wise optimal segmentation algorithm [36] that involves the addition of a spectral clustering step, which allows for the merger of non-adjacent regions controlled by the "spectral clustering weight" input parameter. HSEG can be summarized in four steps:

1. Initialize the segmentation by assigning each image pixel to a region label. If a pre-segmentation is provided, then 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 least dissimilarity value, and merge those pairs of regions.
3. Calculate the dissimilarity value between all pairs of spatially non-adjacent regions and find the pair with the least dissimilarity value, that is less than the minimum dissimilarity value found in (2). If found so, then merge that pair of regions. If not, just go to step (4).
4. Stop if no more merges are required (once the minimum number of regions is reached); otherwise, return to step (2).

The HSEG is an iterative region-merging process, initialized with every pixel as a region. Figure 1 shows an outline of its main procedures. At each step, the dissimilarity value is calculated for each pair of spatially adjacent regions. The pair of regions with the least dissimilarity value is chosen for merging, and then the new merged region replaces them. The same step is repeated for non-adjacent regions. This process continues until the desired numbers of regions (segments or classes) are reached. HSEG is very computationally intensive, because it

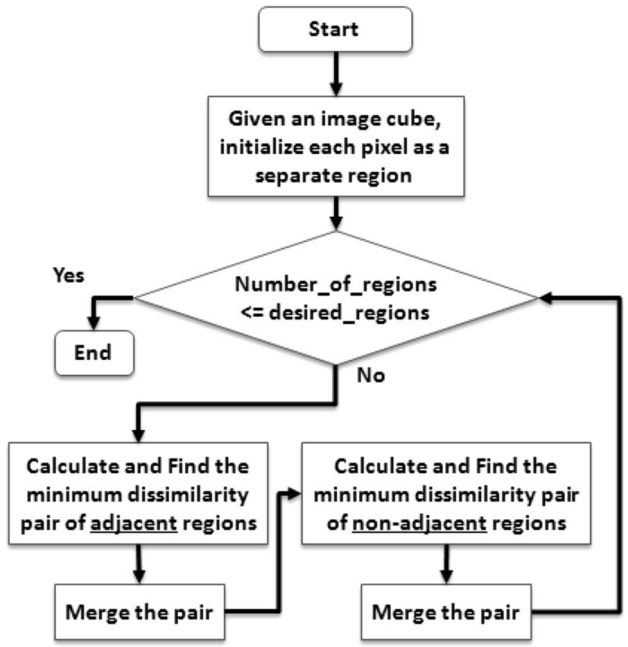

Fig. 1 Outline of HSEG method

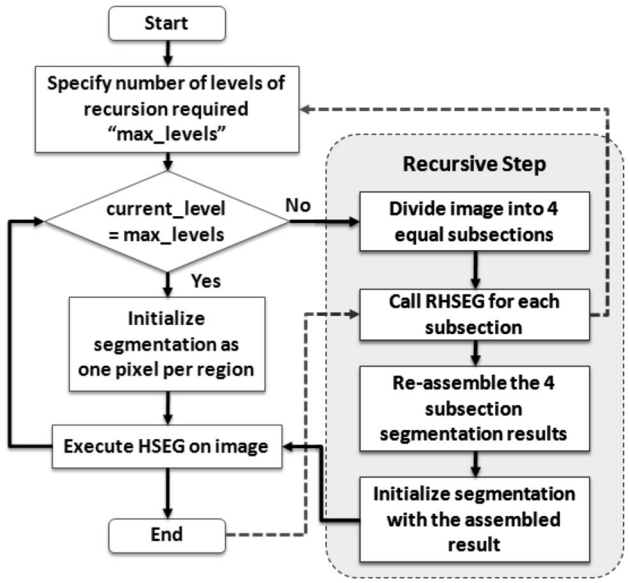

Fig. 2 Flowchart of RHSEG method

requires the calculation of the dissimilarity criterion value between each region and every other region in the image. Tilton [13] described a recursive implementation of this segmentation approach on a cluster called the RHSEG. Figure 2 shows a flowchart of divide-and-conquer, a recursive approach for implementing the HSEG method.

The square root of the band sum means squared error (square root of BSMSE) and is used for the dissimilarity measurement which is given between any two regions i and j in an image of B bands by

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

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 numbers of pixels in regions i and j, respectively.

4 Different approaches to parallelize RHSEG using GPUs

This section presents a parallelization technique proposed for a RHSEG algorithm using GPUs. The main idea of parallelizing the RHSEG algorithm is to distribute the computation of a pair of regions for a dissimilarity measurement to the massive number of GPU threads in parallel. This is the most computationally intensive task of the whole algorithm, and it takes over 95 % of the whole execution time. We propose two different approaches to distribute the dissimilarity measurement between regions among GPU threads: the first approach is to make each GPU thread responsible for all dissimilarity calculations of a single region toward all its spatially adjacent regions or all non-spatially adjacent regions in the image. The second approach is to make each GPU thread responsible for the calculation of dissimilarity between only two regions, either spatially or non-spatially adjacent. The first approach takes a sequential behavior for the calculation of all dissimilarities for a specific region to its adjacent and non-adjacent regions, while other regions calculations are done in parallel. Thus, the first approach does not take full advantage of the parallel GPU threads. However, the second approach results in a much broader parallelism because it allows all dissimilarities of any region's pairs to be computed in parallel, and at the same time, making use of the complete independence of region-pair measurements, and no sequential calculation is needed. Figures 3 and 4 show the difference between the two approaches.

For GPU implementations, many development platforms were considered such as OpenCL [37], NVidia Compute Unified Device Architecture (CUDA) [15], and Microsoft C++ Accelerated Massive Parallelism (C++ AMP) [16]. The main factors that were given the highest priority for selecting the desired platforms were platform maturity and performance. Thus, two of these platforms were selected: NVidia CUDA and Microsoft C++ AMP, which were the two most mature and advanced GPU platforms that also provided top computation performance. GPU/RHSEG is implemented using these two platforms for both the approaches 1 and 2.

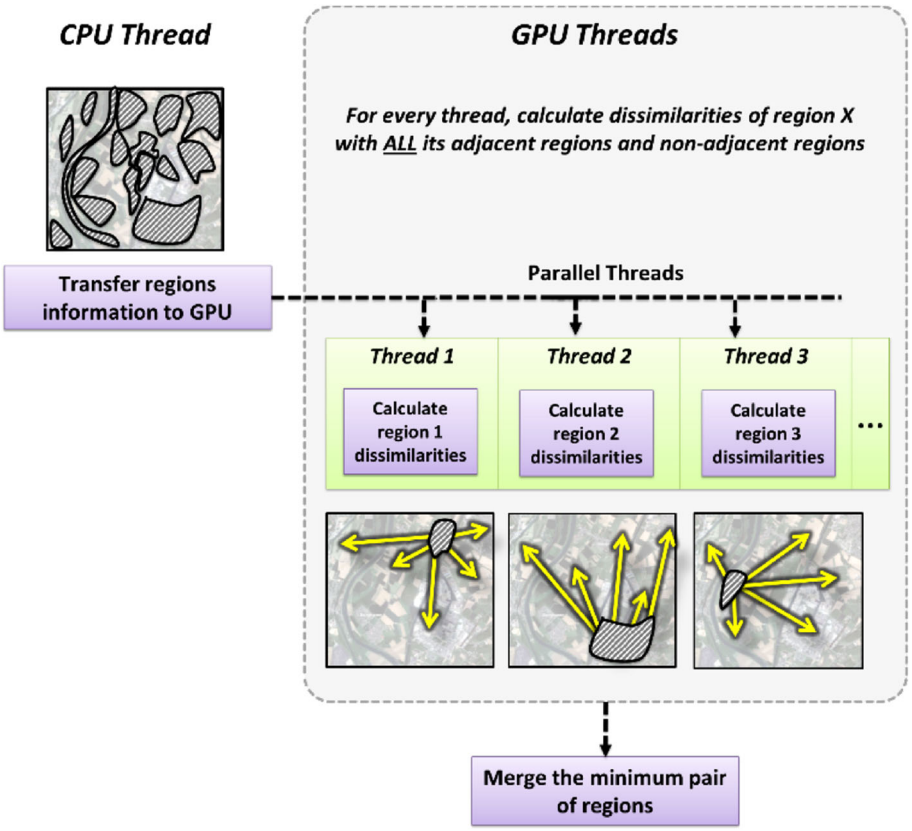

Fig. 3 GPU Approach 1 (first GPU parallelization approach). Each GPU thread is responsible for calculations of all dissimilarities for certain regions

The RHSEG dissimilarity calculation is carried out in two stages in each iteration step. The first stage is the dissimilarity between every region and their spatially adjacent regions; this stage is called the spatial stage. Then, the second stage is the dissimilarity measure between every region and all other non-adjacent regions, which is called the spectral stage. Figure 1 shows both stages in RHSEG flow chart. In both GPU approaches, each stage has a separate kernel—the spatial kernel and the spectral kernel. The spectral stage is the most computationally demanding task contributing to more than 95 % of total running time. To give a comprehensive overview of the GPU kernel implementation details, Fig. 5 illustrates in detail how the GPU approach 2 spectral kernel works and how regions are represented in the GPU memory.

In Fig. 5, a sample image of size 6×6 pixels is passed to the spectral kernel. Before the kernel starts working, every pixel is considered a separate region, which gives $6 \times 6 = 36$ regions (this is only done once at the start of RHSEG; the next iteration uses the produced regions instead of image pixels). Then, every region gets a unique ID from 1 to 36, and information of all the regions(adjacent regions, spectral values of bands and number of pixels) is transferred to the GPU. The spectral kernel uses three arrays. First, the "Adjacencies" 2-D array that is (number of regions) $\times$ (max_adjacencies) matrix is of type integer. It stores the adjacent region IDs of all regions and allows each region to know its adjacent regions by their regional ID. Second, the "Pixels_Count" is an array of the number of regions of type integer. And it stores the number of pixels for every region. Finally, the "Bands_Sum" is a matrix of (number of regions) $\times$ (bands) and stores the sum of the region's pixel values at every band for all regions. The first two arrays reside in the GPU global memory, and the last one resides completely in the global memory and partially in the shared memory (for faster memory access). Finally, a fourth array is needed and is called "Best_Dissim." It stores the best dissimilarity value found for every region against all other regions.

For optimizing memory access bandwidths, a GPU on-chip shared memory is used. A small part of the "Bands_Sums" array is stored in every block's shared memory, and the rest are accessed from the global memory. With the increase of GPU numbers of streaming multiprocessors and shared memory size, more speedup can be achieved by means of more shared memory. Figure 6 shows the detailed GPU code for approach 2 spectral kernel as illustrated in Fig. 5 and is called the "kernel_compute_spectral_dissims." The details about shared memory size, kernel registers size, and the achieved threads' occupancy are reported in Sect. 7.

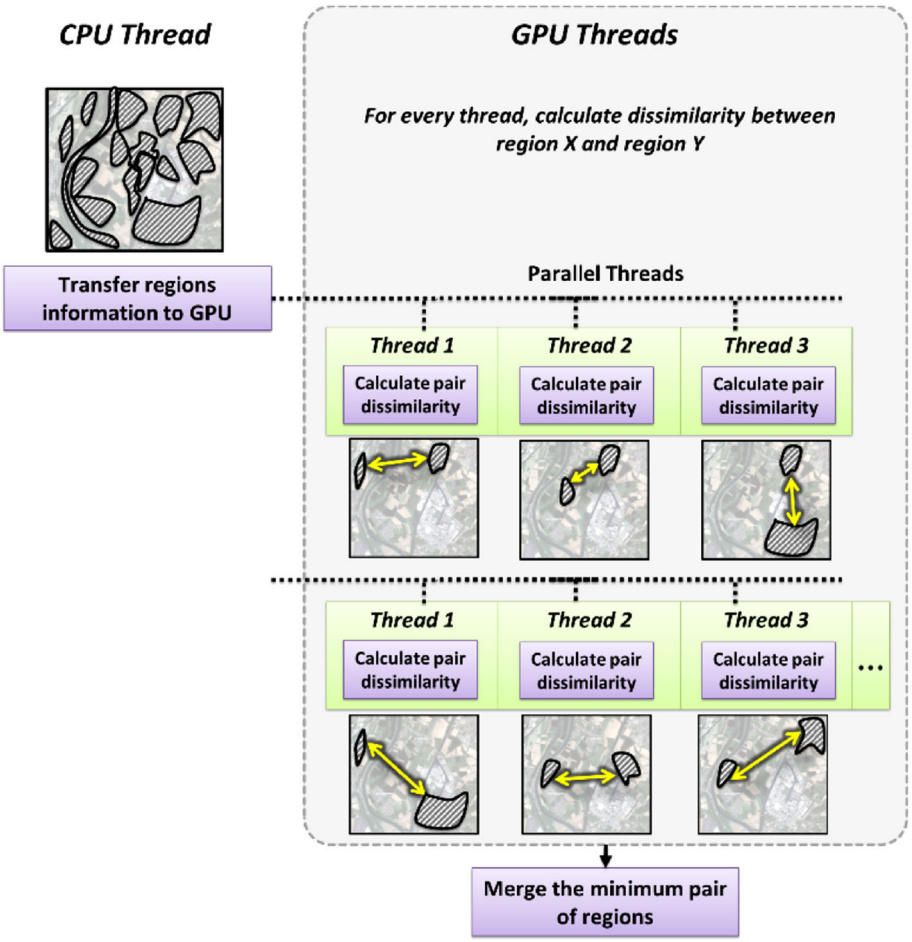

Fig. 4 GPU Approach 2 (second GPU parallelization approach). Each GPU thread is responsible for the calculation of dissimilarity of only one pair of regions

In GPU, each block is composed of a group of threads. The spectral kernel starts traversing all N × N regions using blocks of K × K threads in parallel; therefore, the total number of blocks = N/K × N/K. In each block, dissimilarity between all regions inside the block is calculated as shown in Fig. 5. The spectral kernel checks for every regional pair (Ri, Rj) if they are not-adjacent; if true, it calculates the band sum mean square error value (BSMSE) over all the bands of the two regions, and then the final dissimilarity is the square root of BSMSE. After calculating the dissimilarity, the kernel needs to update the "Best_Dissim" array if it finds that the calculated dissimilarity is the smallest one so far for region Ri. Updating the "Best_dissim" array needs to be done "atomically," using a spin lock critical section to be carried out correctly. After the kernel is finished with all dissimilarity calculations for all regions in the input image, a GPU reduction step over the "Best_Dissim" array is executed to find the pair or regions with minimum dissimilarity to be merged into one region. The two kernels (spatial then spectral) are then launched again after the merge is done to find new regional pairs to merge. The process continues further until the number of regions reaches the desired number of classes for the input image.

Several optimization techniques are taken into account in the design of either the sequential CPU or parallel GPU implementation. All proposed implementations are memory access optimized to improve the data locality and the cache memory hits. For example, all arrays are accessed in row-major order, which is the sequential order of the byte arrangement in the CPU and GPU memory, and all arrays that reside in the GPU global memory have coalescent memory access. In addition, all the proposed implementations are accessed in blocks of K × K elements to improve the data locality. The proposed parallelized parts of RHSEG, which are calculating dissimilarities for each step and choosing the minimum pair to merge, contribute to more than 95 % of the total execution time for both sequential and parallels of RHSEG. Other parts of the algorithm represent less than 5 % of the execution time and are not suitable for parallel implementation, like merging a pair of regions after each step and stitching image sections for every recursive level.

In the following two sections, the proposed solutions for executing RHSEG algorithms using both multicore CPUs

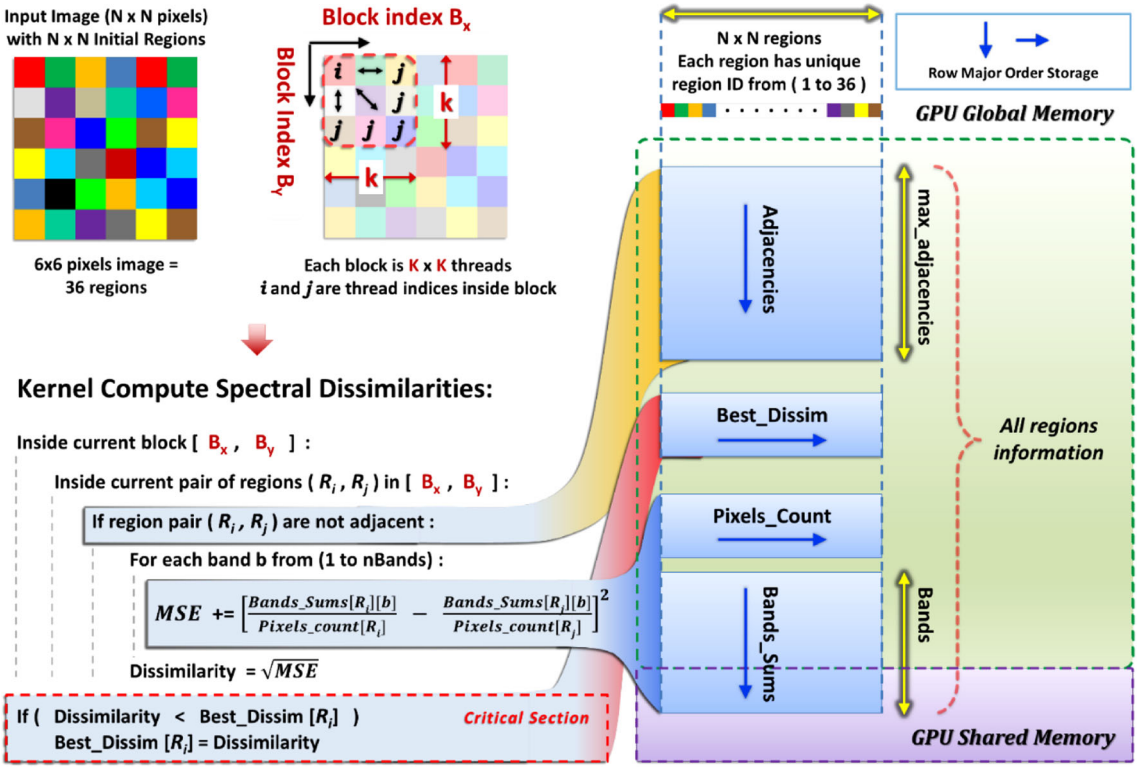

Fig. 5 Example of a spectral stage dissimilarities calculation for Approach 2 using GPU. The spectral kernel operates on N × N image using blocks of size K × K. GPU arrays that hold the required information for all regions. Dissimilarity equals square root of Band Sum Mean Square Error (MSE)

and GPUs cooperatively is presented and is called Hybrid RHSEG. Section 5 describes the implementation of RHSEG using a single multicore CPU and a single GPU, while Sect. 6 describes the Hybrid RHSEG execution on a multinode computer cluster of multicore CPUs and GPUs.

For a multinode hybrid cluster RHSEG algorithm, an Amazon Elastic Compute Cloud (EC2) service is used [18]. However, C++ AMP is not currently capable of running on an Amazon network cluster because EC2 compute instances do not support running DirectX. Thus, the implementation of both parallel approaches of RHSEG algorithms on network clusters is implemented using a CUDA platform.

5 Hybrid CPU/GPU parallel implementation using a single computing node

In the RHSEG, for each recursive level, the image is partitioned into four sections, and this partitioning is repeated again for each quarter recursively till the deepest recursive level is reached. This means that for a 3-level RHSEG, there will be $4^2 = 16$ image sections. The hybrid CPU/GPU implementation of RHSEG is based on distributing different image sections being processed at any level to the GPU and CPU cores. Therefore, different image section computations are executed in parallel on either a GPU or a CPU core. Besides, as the algorithm is designed to work cooperatively; a CPU core can pass its image section to a GPU if it is free, and thus GPU can help in finishing the computation faster and achieving the best utilization. Figure 7 shows the parallel execution of RHSEG on a hybrid CPU/GPU with a 4-core CPU and single GPU.

From Fig. 7, the execution starts with the deepest level of recursion, where we have four indivisible image sections ready for HSEG computation. The four image sections are distributed to GPU and CPU as follows: image section one goes to the GPU and one CPU core (thread); image sections 2, 3, and 4 go to the other three CPU cores (as threads). In this way, the computation of the four sections is executed in parallel. The GPU thread is already faster than any CPU thread; therefore, when the GPU has finished its own image section, it is considered free to conduct future computations of any other image sections. This allows RHSEG to assign a computation of any other image sections to the GPU. Therefore, the GPU picks up any remaining image section that has not been processed. If all image sections are being processed, then it picks up an image section from any running CPU thread to finish it faster. On the other hand, if the GPU finishes the current

```
__global__ void kernel_compute_spectral_dissims ( const int* Adjacencies, const float* Bands_Sums
    , const int* Pixels_Count, float* Best_Dissim, int* Best_Dissim_Labels, int nRegions, const int nBands)
{
    int R_i_index = blockIdx.y * blockDim.y + threadIdx.y;
    int R_j_index = blockIdx.x * blockDim.x + threadIdx.x;

    __shared__ float shared_R_i_sums[BLOCK_WIDTH][BLOCK_BANDS];
    __shared__ float shared_R_j_sums[BLOCK_WIDTH][BLOCK_BANDS];

    //Load BLOCK_BANDS band sums from global "Bands_Sums" matrix
    // to shared memory
    For ( int b = 0; b < BLOCK_BANDS; ++b ) {
        shared_R_i_sums[threadIdx.y][b] =
                        Bands_Sums[R_i_index*nBands+b];
    }
    //...
    For ( int b = 0; b < BLOCK_BANDS; ++b ) {
        shared_R_j_sums[threadIdx.x][b] =
                        Bands_Sums[R_j_index*nBands+b];
    }
    __syncthreads();

    //Check if R_i and R_j are not adjacent
    int foundAdjacent = 0;
    for(int a = 0; a < MAX_ADJACENCIES; ++a) {
        if(Adjacencies[R_i_index*MAX_ADJACENCIES+a]
                                            == R_j_index) {
            foundAdjacent = 1;
            break;
        }
    }
    __syncthreads();

    float MSE = 0.0f;     float Dissimilarity = MAX_FLOAT;

    if(foundAdjacent == 0) {
        float R_i_nPixels = Pixels_Count[R_i_index];
        float R_j_nPixels = Pixels_Count[R_j_index];

        //Calculate MSE using Regions means, first using those in
        //shared memory
        int band = 0;
        for ( band = 0; band < BLOCK_BANDS; ++band ) {
            float temp = shared_R_i_sums[threadIdx.y][band] /
                                                    R_i_nPixels
                       - shared_R_j_sums[threadIdx.x][band] /
                                                    R_j_nPixels;

            MSE += temp * temp
        }

        //and finally, using those in global "Bands_Sums"
        for ( band = band; band < nBands; ++band ) {
            float temp = Bands_Sums[R_i_index*nBands+band] /
                                                    R_i_nPixels
                   - Bands_Sums[R_j_index*nBands+band] /
                                                    R_j_nPixels;

            MSE += temp * temp;
        }

        Dissimilarity = sqrtf(MSE * (R_i_nPixels*R_j_nPixels) /
                                    (R_i_nPixels+R_j_nPixels) );
    }

    // Critical Section ===================================
    If ( Dissimilarity < Best_Dissim[R_i_index] ) {
        Best_Dissim[R_i_index] = Dissimilarity;
        Best_Dissim_Labels[R_i_index] = R_j_index;
    }
    // End of Critical Section ============================
}
```

Fig. 6 RHSEG GPU Approach 2 spectral kernel

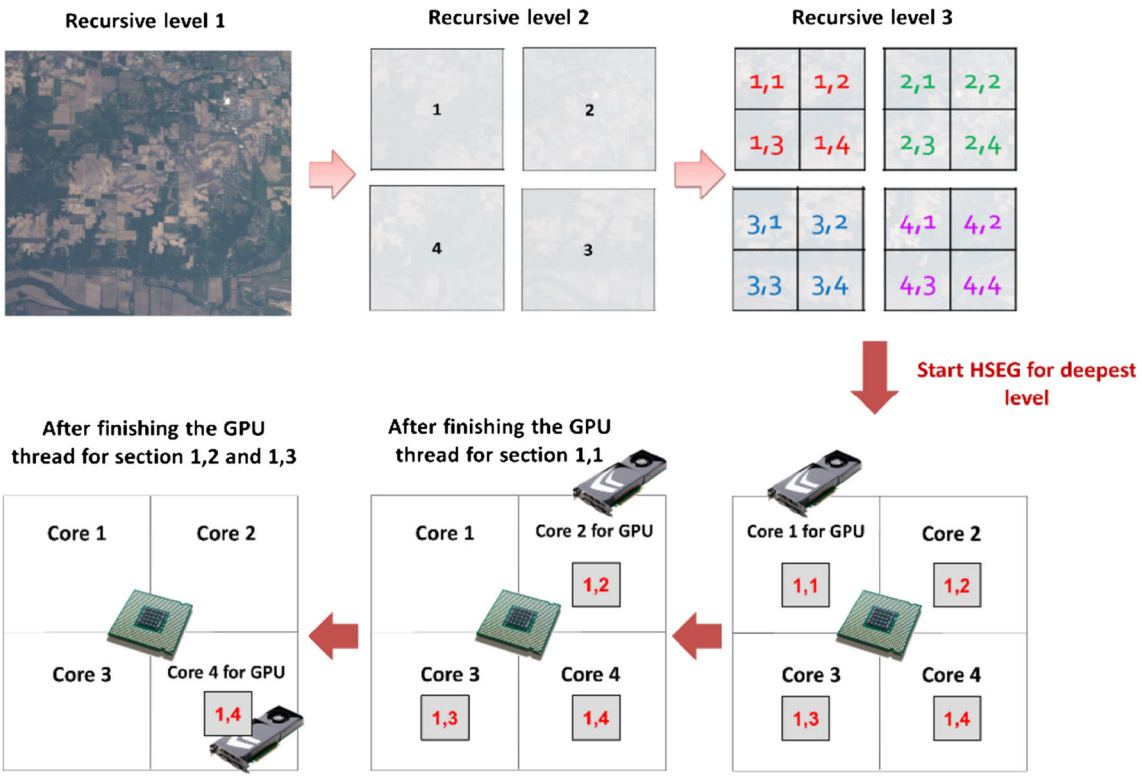

Fig. 7 Step by step Hybrid CPU/GPU RHSEG with 3 recursive levels using 4 cores CPU; computation starts at the deepest third level

image section, then it repeats the same technique by finding another image section to compute, until all images sections are finished. A control thread always looks for every four image sections finished in a certain level, then it combines their results, and the algorithm terminates when the control thread combines the results for the first level (level 1). Algorithm 1 illustrates a Hybrid CPU/GPU RHSEG implementation.

The implementation of RHSEG on GPU is based on distributing the process of a dissimilarity calculation among the GPU threads as described previously, either using the GPU Approach 1 or 2. For GPU Approach 1, each GPU thread is responsible for a corresponding region from the image section. For example, if the image section has 1,024 regions, then 1,024 GPU threads will be initialized, and each thread will calculate the dissimilarity values between its own region and all other adjacent regions. However, for GPU Approach 2, each GPU thread is responsible only for two regions (single region pair), and so if the image section has 1,024 regions, there will be 1,024 × 1,024 GPU threads, where each one of them calculates dissimilarity for the assigned region pair and terminates. After the dissimilarity calculation step is finished (either using Approach 1 or 2), a parallel reduction step is used to find the minimum regional pair and merges them. Then, RHSEG calculates the dissimilarity values for all non-adjacent regions using the GPU again. Next, the reduction step is used again to determine the minimum non-adjacent pair and merges them if found. Finally, RHSEG terminates.

To guarantee the scalability by increasing the number of CPU cores, the algorithm is designed to dynamically use any free available cores for the requisite image section computations. For example, if an 8-core CPU is used, then each core of the eight cores receives an image section from the control thread. Then, the computation is carried out for each section. After that, the results return to the control

Algorithm 1: Hybrid CPU/GPU RHSEG

Input: (f, L) where f is image with N x N pixels, L is the number of desired RHSEG recursion levels

1: Intialize *GPU_Ready* = true, *nSections* = $4^{(L-1)}$, array *Migrated_to_GPU* [*nSections*] = [false]

2: **procedure** Hybrid_RHSEG_Thread (S_i) {S_i is an image section of input image f}
3: *Migrated_to_GPU* [S_i] = false
4: **while** desired number of clusters not reached **do**
5: **if** *Migrated_to_GPU* [S_i] = true **then**
6: *GPU_Ready* = false
7: **GPU_HSEG** (S_i)
8: **else**
9: **CPU_HSEG** (S_i)
10: **end if**
11: **end while**
12: **if** *Migrated_to_GPU* [S_i] = true **then** *GPU_Ready* = true
13: **end procedure**

14: **procedure** Control_Thread (f, L)
15: partition f to equal $4^{(L-1)}$ image sections [S_1 – $S_{4^{(L-1)}}$]
16: Q = [S_1 – $S_{4^{(L-1)}}$] { put all image sections in queue Q for processing }
17: **while** Q not empty **do**
18: S_i = pop image section from Q
19: Hybrid_RHSEG_Thread(S_i) {Create new hybrid thread and send S_i as input}
20: **end while**
21: **while** not all threads are finished **do**
22: **if** *GPU_Ready* = true **then**
23: **for every** thread t in all running threads **do**
24: **if** S_i used in t is processed on CPU core **then**
25: *Migrated_to_GPU* [S_i] = true
26: *GPU_Ready* = false
27: **end if**
28: **end for**
29: **end if**
30: **end while**
31: **end procedure**

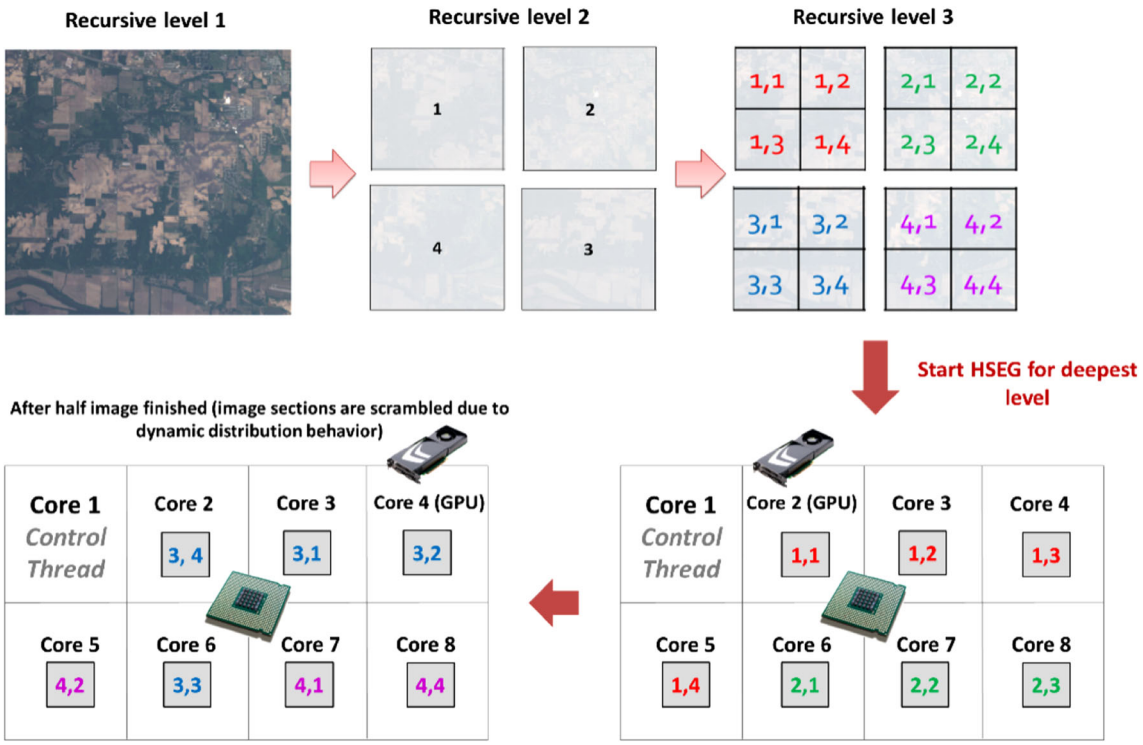

Fig. 8 Hybrid RHSEG using 8 CPU cores and one GPU

thread, and the eight cores will be free to process any other image sections. The control thread is responsible for dispatching image sections to threads and receiving results from different threads for combining at different levels. Figure 8 illustrates the execution process of RHSEG on eight CPU cores and one GPU.

6 Cluster parallel implementation

This section proposes multinode cluster parallel implementations of the RHSEG algorithm; Hybrid CPU/GPU cluster, GPU cluster, Multicore CPU cluster, and CPU cluster. The distributed cluster technique of the hybrid RHSEG is similar to the technique described earlier for multicore machines but uses network nodes as the distributed computing element. Image sections are distributed to network nodes instead of CPU cores (threads), and the control thread of the master node receives section results and stitches them for any recursive level. The master node itself is also used as a computing node. For example, in Fig. 9, four cluster nodes are used; each node in the cluster has eight CPU cores and one GPU. The eight CPU cores in each node are used for the computation of the dedicated image sections sent to this node.

The GPU cluster implementation of RHSEG is similar to the technique described earlier for hybrid CPU/GPU clusters but without the cooperation of multicore CPUs. The control thread allows the GPU only to process the images sections from the queue. Therefore, at each node, the GPU alone is working, and no CPU core is used for computation.

Similarly, the Multicore CPU Cluster implementation of RHSEG works just as the hybrid CPU/GPU cluster technique but the GPUs are not allowed to work or process any image sections. Finally, the CPU cluster implementation works just as the hybrid cluster but with only a single CPU core and a single GPU in each network node allowed to process image sections.

7 Experimental results

To study the different proposed parallelized versions of the RHSEG algorithm, three categories of experiments were carried out; accuracy assessment, execution time, and energy-consumption experiments. First, the accuracy assessment shows the classification accuracy of selected data set against the ground truth information for both parallel and sequential CPU and GPU solutions. Second, several execution time experiments are conducted to study the speedup of the proposed GPU solutions compared to sequential CPU solutions under different parameters and data configurations that affect the execution time such as

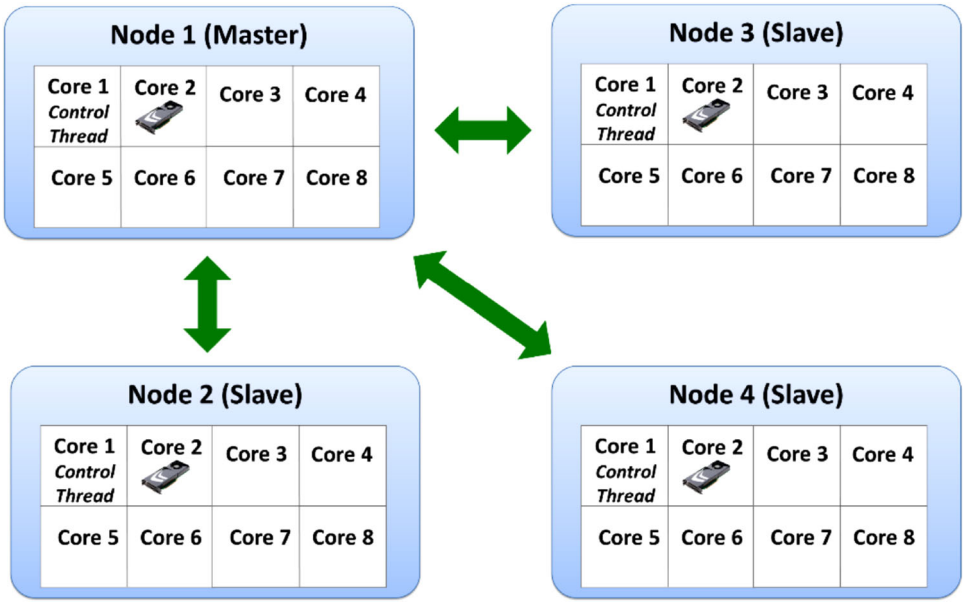

Fig. 9 Example of cluster Hybrid RHSEG, 4 cluster nodes (each one consists of 8 CPU cores and single GPU)

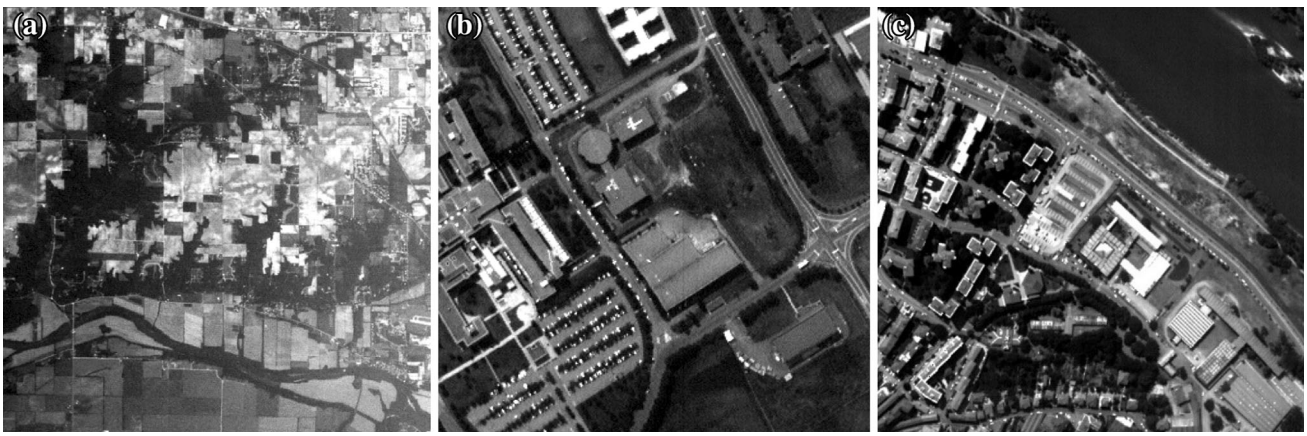

Fig. 10 **a** Indian Pines data set, **b** Pavia Center data set and **c** Pavia university data set

image size, image depth (number of bands), image details, and number and the dimensions of GPU threads. Finally, energy consumption experiments show the power/energy-consumption rates of GPU solutions compared to both sequential and parallel CPU solutions.

For execution time experiments, three sets of experiments were conducted. First, parallel RHSEG on a single GPU without a multicore CPU is carried out using both CUDA and C++ AMP technologies. Second, parallel RHSEG using a Hybrid CPU/GPU single computing node is carried out. Finally, parallel RHSEG using different multinode clusters are carried out: GPU cluster, hybrid CPU/GPU cluster, CPU cluster, and Multicore CPU cluster. The performance of the parallel implementation is measured by calculating the speedup, which is the number of times a parallel implementation is faster than the sequential one on a single CPU core.

7.1 The dataset

The experiments are performed using five different images: three real hyperspectral images, and two manually synthetic images. The three hyperspectral images are the Indian Pines AVIRIS hyperspectral data,[1] the Pavia Center data, and Pavia University data. Figure 10 shows portions of these hyperspectral images. The Indian Pines scene was gathered by the AVIRIS instrument; it consists of 16 ground truth classes. It was acquired over a mixed agricultural/forested region in NW Indiana. Four noisy bands were removed, and the rest of the 220 spectral bands are used. Pavia data in Italy was collected by the ROSIS [38] sensor. The first image was collected over Pavia city center, Italy. It contains 102 spectral channels and nine ground

[1] NASA JPL (http://www.jpl.nasa.gov).

truth classes. The second image was collected over the University of Pavia with nine ground-truth classes and 103 spectral bands.

Experiments are carried out using different hyperspectral image sizes of 128 × 128, 256 × 256, and 512 × 512 pixels. For each image size, data were cropped from the large image, not scaled. The number of bands for Indian Pines is 220, and for Pavia Center and the University they are 102 and 103, respectively. The spectral clustering weight parameter used in all the experiments is 0.25; thus, for every four adjacent region merges, one non-adjacent merge is attempted by RHSEG, and this is an acceptable value that produces clear shaped classification results, while not losing the recognition of the non-adjacent regions of image classes.

7.2 Hardware architectures

The execution of the RHSEG algorithm on a single GPU is tested using NVidia GeForce 550 Ti and NVidia Tesla M2050 devices. GeForce 550 Ti consists of 192 processing cores, each operating at 1,940 MHz, with 1,024 MB GDDR5 192-bit memory interface, which operates at 2,050 MHz that is capable of 98.4-GB/s memory bandwidth. Tesla M2050 contains 448 processing cores each operating at 1,147 MHz, with 384-bit memory operating at 1,546-MHz clock The CPU used is Intel Core i5 with 3,100 MHz, 256 KB L1, 1 MB L2, and 6 MB L3 cache memories.

For the multinode hybrid cluster RHSEG algorithm, an Amazon Elastic Compute Cloud (EC2) [18] service is used. Each EC2 node used dual quad core (total 8 cores) Intel Xeon X5570 operating at 2.93 GHz, and the GPU used the NVidia Tesla M2050 device. EC2 Cluster nodes are connected to each other by a 10 Gb/s Ethernet network. All implemented codes were compiled using Microsoft Visual C++ 2012 with compiler flag/O2 for speed optimization. To ensure the consistency of the square root floating point calculation across all different parallel and sequential architectures, appropriate compiler flags were used in all implementations to force accurate calculations based on the IEEE 754 floating point standard; For Visual C++ 2012 CPU sequential code, the flag/fp:precise was used; for all CUDA parallel implementations, the flags –prec-sqrt = true, –gpu-architecture = compute_20 and –gpu-code = sm_21 are used; finally for C++ AMP parallel implementation, the precise math library namespace "Concurrency::precise_math" was used.

7.3 Results

The classification accuracy assessment of proposed parallelized versions of the RHSEG algorithm is carried out using a Pavia Center dataset. The image was cropped to size 490 × 490 pixels with 97 bands after removing first five noisy bands. Figure 11 shows a section of the Pavia center image and the corresponding ground truth image of nine classes. The classification result is compared with the provided ground truth information. Classification is carried out using GPU, hybrid CPU/GPU, and sequential CPU solutions. In all three cases, the classification results were identical. Figure 12 shows the RHSEG segmentation result using the square root of BSMSE dissimilarity criterion with four levels of recursion, and the spectral clustering weight equals 0.15. The coarsest segmentation result is selected, which separates most of the nine classes. Each segmentation class was assigned to a specific ground truth class that covered the plurality of their pixels. Accuracy scores for all nine materials are shown in Table 1 with an overall accuracy of 76 %.

For execution time and speedup, three sets of experiments are carried out. The first set is carried out to study the parallelized implementation of RHSEG using a single a GPU. First, the next experiment was performed to determine the impact of varying the image sizes on the execution time of the RHSEG with Approaches 1 and 2 using a single GPU. The results are compared with sequential CPU execution. Figure 13 shows the execution times (in seconds) of the RHSEG algorithm using different image sizes for both approaches implemented by CUDA and C++ AMP. For the 128 × 128 image size, the RHSEG CPU sequential execution time is around 7,920 s, while the CUDA GPU Approach 1 execution time is around 2,486 s, C++ AMP Approach 1 is 2,180 s, CUDA Approach 2 is 640 s, and finally C++ AMP Approach 2 is 930 s. One can see from Fig. 13 that the proposed approaches to implement RHSEG using a single GPU have far less execution time than sequential implementations on a CPU.

The GPU running time includes the memory copy time between the main memory and the GPU memory. Table 2 shows the speedups of RHSEG parallel Approaches 1 and 2 on a single node GPU with respect to sequential implementation on a CPU using CUDA and C++ AMP platforms. A 3.1× and 3.5× average speedup is achieved for Approach 1 for CUDA and C++ AMP, respectively; and 12×, 8× and 21× average speedup for CUDA and C++ AMP Approach 2 over the sequential CPU implementation.

In this experiment, Approach 2 kernels were launched using a block size of 16 × 16 threads, and the maximum size of on-chip shared memory used is 2 KB per block. The spectral kernel in Fig. 6 uses 24 registers in each thread, so that for a 16 × 16 block, a total of 6,144 registers are used. The current implementation for Approach 2 using the described algorithm in CUDA results in 78 % occupancy, which means that every streaming multiprocessor in the

GPU runs 1,200 threads at a time, out of the maximum available 1,536 per streaming multiprocessor (with GeForce 550 Ti device).

The next experiment was performed to study the impact of changing the image details on the execution time of the RHSEG using a single GPU. Figure 14 shows three images that differ in spectral details. Figure 14a, b are synthetic images generated manually for the sake of the experiment. Figure 14c is a portion of the Indian Pines image. The images are differing in the number of classes/regions. Each image size is 50 × 50 pixels ×220 bands. Table 3 shows the speedup of RHSEG on single GPU using different images with different details. One can see from Table 3 that the speedup is almost not affected by the increasing number of region/classes. Then, changing the complexity and details of the image also does not affect the speedup significantly.

The next experiment was performed to study the impact of changing the image depth (number of bands) on the execution time of the RHSEG using a single GPU. For an image size of 32 × 32 pixels, the experiments are carried out using 3, 10, 50, 100, 150, and 220 bands. Table 4 shows the performance of the GPU implementation for the different numbers of bands. For the GPU Approach 1, the speedup increases slightly by increasing the number of bands. On the other hand, with GPU Approach 2, the speedup increases significantly with increase in the number of bands. GPU Approach 2 with three bands achieves 2× speedup, while using 220 bands achieves 12× speedup with respect to sequential CPU. Hence, it is clear that both the CPU approaches are significantly sensitive to the changes in the number of bands.

The next experiment was performed to study the effect of changing the number of threads per block for a single GPU for both GPU Approaches 1 and 2. For an image size of 32 × 32 pixels ×220 bands, the experiments are carried out using 4 × 4, 8 × 8, and 16 × 16 threads per block for Approach 2 (CUDA and C++ AMP). Table 5 shows the performance of the GPU implementation for the different numbers of threads per block. It is noticeable that changing the block size affects the speedups. For example, speedups increased significantly by increasing the block size from 4 × 4 to 16 × 16. The optimal block size for given inputs was 16 × 16 threads per block.

The second set of execution time experiments were performed for parallelized implementation of the RHSEG Approach 2 on a single-node hybrid CPU/GPU using CUDA. For 64 × 64 × 220 image size, the RHSEG CPU sequential execution time is around 2,033 s, while the RHSEG GPU execution time is around 94 s, and the hybrid parallel execution time was about 89 s. Table 6 shows speedup results for a GPU node and a single hybrid CPU/GPU node against sequential implementation on a CPU. 21.6 and 22.8× average speedups are achieved for a single GPU and hybrid CPU/GPU implementation, respectively, versus the sequential CPU implementation.

The third set of experiments was performed for a parallelized RHSEG on different multinode cluster types: GPU cluster, hybrid CPU/GPU multinode cluster, CPU cluster, and multicore CPU cluster. Execution times are recorded and compared to the CPU sequential execution time. Also the execution time is compared with the single GPU implementation. In this experiment, NVidia Tesla M2050 is used for both single and multinode GPU clusters and hybrid clusters. For the Indian Pines image, the experiments are carried out using 256 × 256 × 220 and 512 × 512 × 220 pixels × bands. Table 7 shows the results for 4, 8, and 16 cluster nodes. Figure 15 shows the speedup expressed as a function of the number of nodes for the Indian Pines image

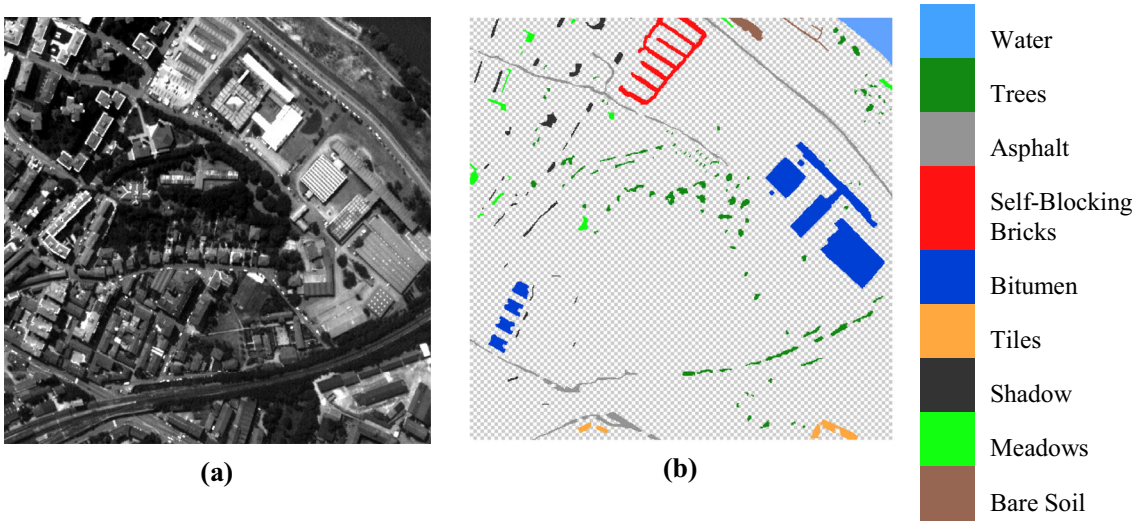

Fig. 11 **a** Pavia Center image section of 490 × 490 pixels containing all nine classes provided with the dataset, and **b** Pavia Center ground truth classes with a color key for each class

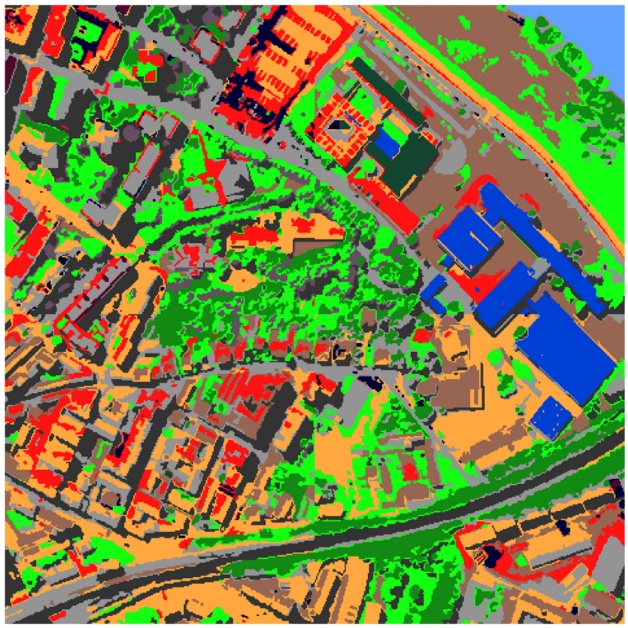

Fig. 12 Classification map for Pavia Center image section showing all nine ground truth classes

Table 1 Classification accuracy for each ground truth class of the Pavia Center dataset

Class	Accuracy (%)
Water	100
Trees	62.7
Asphalt	59.9
Self-blocking bricks	68.2
Bitumen	84.3
Tiles	56.1
Shadow	99.7
Meadows	61.4
Bare soil	92.3
Overall	76

of size 512 × 512 pixels. One can observe from Fig. 15 that the speedup increases by increasing the number of nodes. Furthermore, one can observe from Table 7 that speedups of 15, 55, 249, and 259 times on a CPU cluster, multicore CPU cluster, GPU cluster, and hybrid CPU/GPU multinode cluster, respectively, are achieved versus the sequential CPU implementation.

Finally, the last experiment was performed to study the power/energy consumption of the proposed parallel GPU/CPU solutions against the sequential CPU solution. A power meter device is used to read the watts consumed by the CPU unit from the wall socket; thus, the samples from the power meter are collected externally and separately from the experiment system, in order to prevent the measurements from affecting the accuracy of the experiments results. Power and energy consumed by the system in an idle state (i.e., disks, fans, and idle CPU/GPU processing) are measured separately and subtracted from the computation measurement results. During experiment execution, power readings decrease over time, and so the power readings from the meter are collected over the executions duration, and the average power and energy values are calculated and used for the comparison.

The power and total energy consumed are measured during the computing period of both CUDA and C++ AMP Approach 2 computations on 128 × 128 × 220 image size. The average power and energy consumption values are calculated by taking the mathematical average of five repetitive power and energy measurements for every experiment. Table 8 shows the power and energy consumption measurements for both CUDA and C++ AMP on a single NVidia GeForce 550 Ti GPU; columns four and six show the relative energy consumption ratios of the different parallel GPU platforms to both serial and parallel CPU.

From Table 8, it is noticeable that Approach 2 clearly achieves less energy consumption than the sequential CPU solution. It is more useful to also compare the

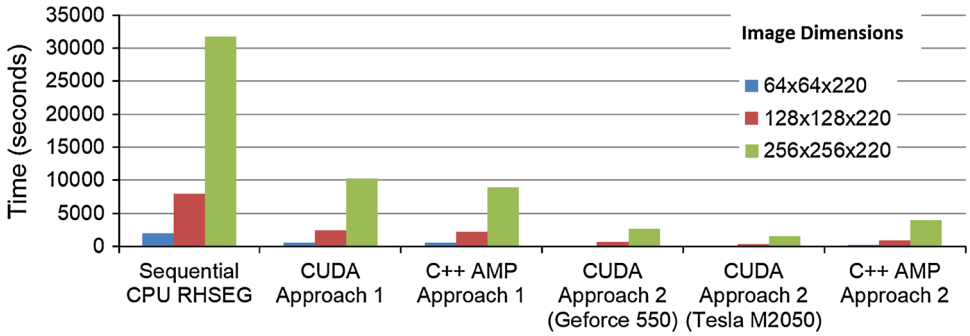

Fig. 13 Execution times (in seconds) of RHSEG parellel Approaches 1 and 2 using CUDA and C++ AMP on a single GPU, for different image sizes

Table 2 Speedups of RHSEG parallel Approaches 1 and 2 on a single node GPU with respect to sequential implementation on CPU

Image dimensions	CUDA GPU Approach 1 speedup	C++ AMP Approach 1 speedup	CUDA GPU Approach 2 speedup (GeForce 550 Ti)	CUDA GPU Approach 2 speedup (Tesla M2050)	C++ AMP Approach 2 speedup
64 × 64	3.2×	3.7×	12.6×	21.8×	8.9×
128 × 128	3.1×	3.6×	12.3×	21.7×	8.5×
256 × 256	3.1×	3.5×	12.1×	21.6×	8.0×
512 × 512	3.0×	3.4×	11.8×	21.5×	7.6×

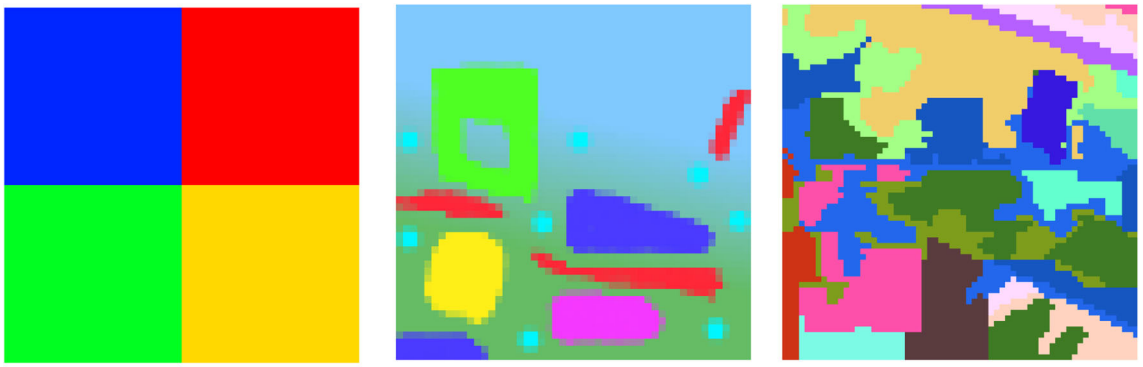

Fig. 14 **a** Detail image 1: synthetic image with 4 classes/4 regions, **b** detail image 2: synthetic image with 8 classes/12 regions and **c** detail image 3: portion Indian Pines image with 16 classes/25 regions

Table 3 Speedups of RHSEG on a single GPU (CUDA and AMP for Approaches 1 and 2, respectivley) using different image details with respect to sequential implementation on CPU

Image details	Single GPU speedup (CUDA/C++ AMP Approach 1)	Single GPU speedup (CUDA/C++ AMP Approach 2)
Image a (4 classes/4 regions)	3.1×/3.8×	12.7×/9.5×
Image b (8 classes/12 regions)	3.1×/3.8×	12.7×/9.5×
Image c (16 classes/25 regions)	3.3×/3.9×	12.8×/9.6×

Table 4 Speedups of RHSEG on a single GPU (CUDA and C++ AMP for Approaches 1 and 2, respectivley) using different image depths with respect to sequential implementation on CPU

Image depth (# of bands)	Single GPU speedup (CUDA/C++ AMP) Approach 1		Single GPU speedup (CUDA/C++ AMP) Approach 2	
3	1.3×	0.1×	2×	0.09×
10	2.8×	0.4×	6.5×	0.3×
50	3.0×	2.2×	11.4×	1.5×
100	3.3×	3.0×	12.5×	2.8×
150	3.3×	3.5×	13×	7.3×
220	3.3×	3.9×	12.8×	9.6×

energy consumptions of the proposed parallel GPU solutions against the parallel CPU solution and not only the sequential one, so that we can decide if it is beneficial in terms of energy consumption to use the GPU parallel system instead of the parallel CPU system. The last column in Table 8 shows the ratio of energy consumption of the GPU platforms to the equivalent parallel CPU clusters. The "Equivalent parallel CPU cluster" means that for a certain GPU platform speedup, a parallel CPU cluster is configured to achieve the same speedup, and then the power consumption of the two systems is compared. For example, for both CUDA and C++ AMP, a parallel CPU clusters of 4 and 3 computing nodes, each with four CPU cores achieving up to 12.8× and 9.6× speedups, respectively, are used and their energy consumptions are calculated (excluding the idle power). Then the ratio of Approach 2 (CUDA/C++ AMP) to the CPU cluster energy consumption is calculated. It is found that Approach 2 CUDA and C++ AMP energy consumption is lower than the equivalent parallel CPU cluster by 12 and 26 %, respectively, a reduction from 100 to 88 % and 74 %, respectively.

Table 5 Speedups of RHSEG on a single GPU (CUDA and C++ AMP for Approaches 1 and 2, respectivley) using different threads per block sizes with respect to sequential implementation on CPU

GPU threads per block	Single GPU speedup (CUDA/C++ AMP)
4 × 4 Threads	NA/5.3×
8 × 8 Threads	8.4×/8.9×
16 × 16 Threads	12.8/9.6×

Table 6 Speedups of RHSEG algorithm on a single node using GPU or Hybrid CPU/GPU with respect to sequential implementation on CPU

Image dimensions	GPU	Hybrid CPU/GPU (8 CPU cores)
64 × 64	21.8×	22.8×
128 × 128	21.7×	22.9×
256 × 256	21.6×	22.8×
512 × 512	21.5×	22.7×

8 Conclusions and future work

This paper proposed a parallelized RHSEG algorithm using GPUs with the cooperation of multicore CPUs and computer clusters for onboard processing scenarios. RHSEG is a well- known OBIA technique that was developed by NASA for effectively analyzing hyperspectral images with high spatial resolutions. The proposed parallel implementations are focused toward onboard processing by both accelerating execution times and reducing the power consumption by using GPUs that are lightweight computation devices with a low-power consumption potential for certain tasks.

Three parallel solutions are proposed: parallel RHSEG using a single GPU without a multicore CPU and implemented using both CUDA and C++ AMP technologies; parallel RHSEG using a Hybrid multicore CPU/GPU single computing node; and parallel RHSEG using multinode clusters—a GPU cluster, hybrid CPU/GPU clusters, CPU clusters, and a Multicore CPU cluster. The fundamental idea of the solution is the parallelization of the dissimilarity calculation step in the RHSEG algorithm because of the natural suitability of parallelization in these calculations. Other parts of the algorithm are executed on the main CPU thread.

The single GPU implementation was tested using the NVIDIA GeForce board. In the hybrid parallel CPU/GPU RHSEG, multicore CPUs were used in cooperation with GPU hardware for the parallel implementation of the RHSEG algorithm. Hybrid RHSEG works by distributing the workload of partitioned quad image sections among different CPU cores which run in parallel and cooperatively with the GPU. For the execution of the RHSEG algorithm on a single GPU and CPU/GPU (8 CPU cores) using a CUDA platform speedups of 21.6 and 22.8 times sequential CPU are achieved, respectively. The achieved speedups by parallel GPU solutions compared with a CPU sequential implementation using CUDA and C++ AMP platforms are 21.6× and 9.6×, respectively.

For cluster implementation of the RHSEG algorithm, multinodes of both GPU and hybrid CPU/GPU clusters are used. The network cluster is implemented using the Amazon Elastic Compute Cloud (EC2), with a number of computing nodes that range from 4 to 16. Cluster RHSEG distributes the partitioned image sections to computing nodes to process them in parallel and collects the results returning them to the main node. For a single node hybrid multicore CPU/GPU and multinode computer cluster with

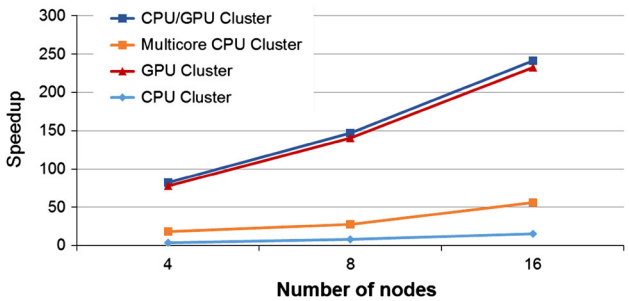

Fig. 15 Hybrid CPU/GPU RHSEG cluster speedups of different cluster sizes: 4, 8, and 16 nodes

Table 7 Speedups of RHSEG on a multinode Hybrid CPU/GPU cluster with respect to sequential implementation on CPU, CPU clusters, and multicore CPU clusters

Image size	No. of nodes	CPU cluster	Multicore CPU cluster (8 cores)	GPU cluster (NVidia tesla M2050)	Hybrid CPU/GPU cluster	Single GPU (GeForce 550 Ti)
256 × 256	4	3.9×	29×	80×	84×	21.6×
	8	7.8×	55×	146×	153×	
	16	15.4×	55×	249×	259×	
512 × 512	4	3.9×	30×	78×	82×	21.5×
	8	7.7×	57×	140×	146×	
	16	15.1×	106×	232×	241×	

Table 8 Single GPU energy consumption for CUDA and C++ AMP Approach 2 compared to sequential and parallel CPU energy consumptions

	Image size (width × height × bands)	Average power consumption (W)	Average energy consumption (J) (power × time)	GPU energy consumption compared to sequential CPU (%)	Equivalent parallel CPU energy consumption (J) (same GPU speedup)	GPU energy consumption compared to equivalent parallel CPU cluster (%)
CPU sequential RHSEG	128 × 128 × 220	15	117,600	NA	NA	NA
Approach 2 (CUDA)	128 × 128 × 220	115	69,920	59	80,260	88
Approach 2 (C++ AMP)	128 × 128 × 220	75	60,000	52	81,600	74

16 nodes for 256x256 image, speedups of 22 and 259 times the sequential CPU are achieved, respectively. Power consumption is reduced to 74 % using a single GPU C++ AMP solution. The achievements reported in this work represent a step forward for faster, efficient time-critical processing for onboard remote sensing.

In future, a number of optimizations are planned to achieve higher speedups. These optimizations include significantly optimizing RHSEG using dynamic programming by eliminating the re-computation of unchanged regions in each step, changing the RHSEG to make multiple region merges per step instead of a single merge per step to reduce the execution time for the first step, and reducing the number of steps needed for merging identical regions. CUDA and C++ AMP GPU implementations can be further tweaked by using loop-unrolling techniques and using global constant memory for parts of a regions data that are constant during the computation. For CPUs with a high number of cores, more than eight, parallel platforms like OpenMP or OpenACC can be introduced and compared with existing GPU and Hybrid CPU/GPU parallel implementations. Also for each core, instruction vectorization can be utilized using an enhanced instruction set as in streaming SIMD extensions to further achieve higher CPU resource usage for parallel CPU solutions.

References

1. Plaza, A.J., Chang, C.: High performance computing in remote sensing. Taylor & Francis Group, BocaRaton (FL) (2007)
2. Shippert, P.: Introduction to hyperspectral image analysis. Online J. Space Commun. 3 (2003)
3. Asanovic, K., Bodik, R., Catanzaro, B.C., Gebis, J.J., Husbands, P., Keutzer, K., Patterson, D.A., Plishker, W.L., Shalf, J., Williams S.W., Yelick, K.A.: The landscape of parallel computing research: a view from Berkeley. In: Technical report No. UCB/EECS-2006-183, EECS Department. University of California, Berkeley (2006)
4. Owens, J.D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A.E., Purcell, T.J.: A Survey of general-purpose computation on graphics hardware. Comput. Graph. Forum **26**(1), 80–113 (2007)
5. Setoain, J., Prieto, M., Tenllado, C., Tirado, F.: GPU for parallel on-board hyperspectral image processing. Int. J. High Perform. Comput. Appl. **22**(4), 424–437 (2008)
6. Lu, D., Weng, Q.: A survey of image classification methods and techniques for improving classification performance. Int. J. of Remote Sens. **28**(5), 823–870 (2007)
7. Li, J., Bioucas-Dias, J.M., Plaza, A.: Spectral-spatial classification of hyperspectral data using loopy belief propagation and active learning. IEEE Trans. Geosci. Remote Sens. **51**(2), 844–856 (2013)
8. Xu, L., Li, J.: Bayesian classification of hyperspectral imagery based on probabilistic sparse representation and Markov Random Field. IEEE Geosci. Remote Sens. Lett. **11**(4), 823–827 (2014)
9. Blaschke, T.: Object based image analysis for remote sensing. ISPRS J. Photogramm. Remote Sens. **65**(1), 2–16 (2010)
10. Camps-Valls, G., Tuia, D., Bruzzone, L., Atli Benediktsson, J.: Advances in hyperspectral image classification: earth monitoring with statistical learning methods. IEEE Signal Process. Mag. **31**(1), 45–54 (2014)
11. Plaza, A., 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., Trianni, G.: Recent advances in techniques for hyperspectral image processing. Remote Sens. Environ. **113**, 110–122 (2009)
12. Bioucas-Dias, J.M., Plaza, A., Dobigeon, N., Parente, M., Du, Q., Gader, P., Chanussot, J.: Hyperspectral unmixing overview: geometrical, statistical, and sparse regression-based approaches. IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens. **5**(2), 354–379 (2012)
13. Tilton, J.C.: Method for recursive hierarchical segmentation by region growing and spectral clustering with a natural convergence criterion. Discl. Invent. New Technol. NASA Case No. GSC **14**, 328–331 (2000)
14. Hossam, M. A., Ebied, H. M., Abdel-Aziz, M. H.: Hybrid cluster of multicore CPUs and GPUs for accelerating hyperspectral image hierarchical segmentation. In: 8th International conference on computer engineering and systems (ICCES'13), 262–267 (2013)
15. Kirk, D.: NVIDIA CUDA software and GPU parallel computing architecture. In: 6th International symposium on memory management (ISMM '07), 103–104 (2007)
16. Gregory, K., Miller, A.: C++ Amp: accelerated massive parallelism with Microsoft Visual C++. Microsoft Press Series, Microsoft GmbH (2012)

17. Qt Project Documentation. http://www.qt-project.org/doc/qt-4.8/
18. Amazon Elastic Compute cloud EC2. http://www.aws.amazon.com/ec2/
19. Jain, A.K., Murty, M.N., Flynn, P.J.: Data clustering: a review. ACM Comput. Surv. (CSUR) J. **31**(3), 264–323 (1999)
20. Beucher, S., Lantuéjoul, C.: Use of watersheds in contour detection. International Workshop on Image Processing, Real-Time edge and motion detection/estimation, CCETT/INSA/IRISA, IRISA Report no. 132, 2.1–2.12 (1979)
21. Tarabalka, Y., Chanussot, J., Benediktsson, J.A.: Segmentation and Classification of hyperspectral images using watershed transformation. Pattern Recognit. J. **43**(7), 2367–2379 (2010)
22. Moreno, R., Graña, M.: Segmentation of hyperspectral images by tuned chromatic watershed, recent advances in knowledge-based paradigms and applications. Springer Int. Publ. **234**, 103–113 (2014)
23. Felzenszwalb, P.F., Huttenlocher, D.P.: Efficient graph-based image segmentation. Int. J. Comput. Vision **59**(2), 167–181 (2004)
24. Fauvel, M., Tarabalka, Y., Benediktsson, J.A., Chanussot, J., Tilton, J.C.: Advances in spectral-spatial classification of hyperspectral images. Proc. IEEE **101**(3), 652–675 (2013)
25. Ball, G.H., Hall, D.J.: ISODATA: a novel method of data analysis and classification, Tech. Rep. Stanford University, Stanford CA (1965)
26. Plaza, A.J.: Parallel spatial-spectral processing of hyperspectral images. Comput. Intel. Remote Sens. SCI **133**, 163–192 (2008)
27. Deb, S., Sinha, S.: Comparative improvement of image segmentation performance with graph based method over watershed transform image segmentation, distributed computing and internet technology. Springer Int. Publ. **8337**, 322–332 (2014)
28. Beucher, S.: Watershed, hierarchical segmentation and waterfall algorithm. The second international conference on Mathematical Morphology and its Applications to Image Processing, 69–76 (1994)
29. Tarabalka, Y., Chanussot, J., Benediktsson, J.A.: Segmentation and classification of hyperspectral images using minimum spanning forest grown from automatically selected markers. IEEE Trans. Syst. Man Cybern. B Cybern. **40**(5), 1267–1279 (2010)
30. Plaza, A., Valencia, D., Plaza, J., Martinez, P.: Commodity cluster-based parallel processing of hyperspectral imagery, plaza, valencia. J. Parallel Distrib. Comput. **66**, 345–358 (2006)
31. Lai, C., Huang, M., Shi, X., You H.: Accelerating geospatial applications on hybrid architectures. In: Proceedings of 15th IEEE International Conference on High Performance Computing and Communications (HPCC 2013), 1545–1552 (2013)
32. Kenneland supercomputer. http://www.keeneland.gatech.edu/
33. Yang, S., Dong J., Yuan, B.: An efficient parallel ISODATA algorithm based on Kepler GPUs. In: International Joint Conference on Neural Networks (2014)
34. Becker, D.J., Sterling, T., Savarese, D., Dorband, J.E., Ranawake, U.A., Packer, C.V.: BEOWULF: a parallel workstation for scientific computation. In: Proceedings of International Conference on Parallel Processing (ICPP) (1995)
35. Tilton, J.C.: Image segmentation by region growing and spectral clustering with a natural convergence criterion. In: International Geoscience and Remote Sensing Symposium (IGARSS 98), pp 1766–1768. Seattle, WA (1998)
36. Beaulieu, J.M., Goldberg, M.: Hierarchy in picture segmentation: a stepwise optimization approach. IEEE Trans. Pattern Anal. Mach. Intell. **11**(2), 150–163 (1989)
37. Khronos Group OpenCL. Available: http://www.khronos.org/opencl/
38. Reflective Optics System Imaging Spectrometer, ROSIS. http://www.opairs.aero/rosis_en.html

Mahmoud A. Hossam is a research assistant at the University of Ain Shams, Egypt, where he currently pursues his M.Sc. degree in Computer Science. His research interests are high-performance computing, parallel architectures, and operating systems. His current work focus is on building parallel algorithms for hyperspectral data analysis using commodity graphics processing units.

Hala M. Ebied is an Associate Professor at the Faculty of Computer and Information Sciences (FCIS), Ain Shams University, Egypt since 2014. From 2006 to 2008, she was a Ph.D. student at Heinz Nixdorf Institute at the University of Paderborn, Germany. She received her M.Sc. and Ph.D. degrees in Computer and Information Sciences from Ain Shams University, Egypt in 2002 and 2009, respectively. She received her B.Sc. degree in Pure Math. and Computer Science from the Faculty of Science, Ain Shams University. Her research interests include Image Processing, Computer vision, Computational Intelligence, and Robotics.

M. H. Abdel-Aziz from 2007 till date is working as a tenure Assistant Professor at the Basic Science department in the Faculty of Computer and Information Sciences, Ain Shams University, Cairo-Egypt. In 2005, he got his Ph.D. from Wayne State University, MI, USA in High-energy Theoretical Nuclear Physics. In 2006, he got a Post Doc. He is a Visiting Scientist at the Institute for Theoretical Physics, Goethe University, Frankfurt, Germany. His main research interests are in the field of image processing, stochastic modeling, high-performance computing, and high-energy nuclear physics phenomenology. He coauthored about 39 research papers in refereed journals and international conferences. Since 2013, he is working as the Director of Scientific Computing Center at Ain Shams University. He served as a coordinator of the bioinformatics program in the Faculty of Computer and Information Sciences, Ain Shams University, from 2007 to 2013.

Mohamed F. Tolba is a Professor of Scientific Computing (from 1984 till date). He was the Vice President of Ain Shams University (2002–2006) and the Dean of the Faculty of Computers and Information Sciences (1996–2002). Dr. Tolba has more than 150 publications in the fields of AI, Image Processing, Pattern Recognition, OCR, Scientific Computing, Simulation, and Modeling. Also Dr. Tolba has supervised more than 50 M.Sc. and 25 Ph.D. degrees in Ain Shams University and other Egyptian Universities. He is a consultant to different local and international organizations for IT. Dr. Tolba is the Senior Member of the Institute of Electrical and Electronics Engineers (IEEE), USA. He is also the Chairman of the IT sector committee of the Supreme Council of Egyptian Universities, and the Chairman of the Scientific Committee of the Supreme Council for Professorship Promotion in the field of Information Sciences.