HIGH PERFORMANCE COMPUTING FOR HYPERSPECTRAL IMAGE ANALYSIS: PERSPECTIVE AND STATE-OF-THE-ART

Antonio Plaza[1], *Qian Du*[2], *Yang-Lang Chang*[3]

[1]Department of Technology of Computers and Communications, University of Extremadura, Spain
[2]Department of Electrical and Computer Engineering, Mississippi State University, USA
[3]Department of Electrical Engineering, National Taipei University of Technology, Taiwan

ABSTRACT

The main purpose of this paper is to describe available (HPC)-based implementations of remotely sensed hyperspectral image processing algorithms on multi-computer clusters, heterogeneous networks of computers, and specialized hardware architectures such as field programmable gate arrays (FPGAs) and graphic processing units (GPUs). Combined, the revision of existing techniques conducted in this paper, along with the description of performance results for a parallel hyperspectral processing chain on different architectures, delivers an excellent snapshot of the state-of-the-art in the area of HPC-based hyperspectral image processing and a thoughtful perspective of the potential and emerging challenges of applying HPC paradigms to hyperspectral imaging problems.

Index Terms— Hyperspectral imaging, high performance computing, cluster computing, distributed computing, hardware implementations, multi-cores, GPUs, FPGAs.

1. INTRODUCTION

Hyperspectral imaging is concerned with the measurement, analysis, and interpretation of spectra acquired from a given scene (or specific object) at a short, medium or long distance by an airborne or satellite sensor [1]. The wealth of spectral information available from latest-generation remotely sensed hyperspectral instruments [2] (hundreds of spectral bands in nearly-continual wavelength channels) has quickly introduced new challenges in the analysis and interpretation of hyperspectral data sets. It is expected that, in future years, hyperspectral sensors will substantially increase their spatial and spectral resolution (images with thousands of spectral bands are currently under development). Such wealth of spectral information has opened groundbreaking perspectives in many applications, including environmental modeling and assessment for Earth-based and atmospheric studies, risk/hazard prevention and response including wild land fire tracking, biological threat detection, monitoring of oil spills and other types of chemical contamination, target detection for military and defense/security purposes, urban planning and management studies, etc. The incorporation of hyperspectral imaging instruments on airborne and satellite platforms is currently producing a nearly continual stream of high-dimensional data, and this explosion in the amount of collected information has created new processing challenges in several remote sensing problems [3].

In this paper, we describe the state-of-the-art and future perspectives in the area of high performance computing (HPC) applied to hyperspectral imaging studies. The utilization of HPC systems in hyperspectral imaging applications has become more and more widespread in recent years. It should be noted that HPC implementations do not exclusively rely on computer architecture-based advances. Quite opposite, important developments in algorithm optimization for low-complexity implementations have been successfully developed in order to complement hardware-based improvements when designing computationally effective hyperspectral imaging techniques, including data compression approaches [4]. These approaches are covered in the form of a literature review and description of recent trends in subsequent sections. Then, we provide an HPC-based implementation case study using a processing chain for spectral unmixing as a representative example. The paper concludes with some remarks and open questions.

2. CLUSTER-BASED IMPLEMENTATIONS

Focusing on hardware-based improvements, one of the most widely used approaches is based on using COTS (commercial off-the-shelf) computer equipment, clustered together to work as a computational "team." This strategy, often referred to as Beowulf-class cluster computing, has already offered accesses to greatly increased computational power, but at a low cost (commensurate with falling commercial PC costs) in several hyperspectral imaging problems [5, 6]. In theory, the combination of commercial forces driving down cost and positive hardware trends (e.g., CPU peak power doubling every 18-24 months, storage capacity doubling every 12-18 months, and networking bandwidth doubling every 9-12 months) offers supercomputing performance that can now be applied a much wider range of remote sensing problems [3].

3. DISTRIBUTED IMPLEMENTATIONS

Although most parallel techniques and systems for image information processing employed by NASA and other institutions during the last decade have chiefly been homogeneous in nature (i.e., they are made up of identical processing units, thus simplifying the design of parallel solutions adapted to those systems), a recent trend in the design of HPC systems for data-intensive problems is to utilize highly heterogeneous computing resources [7]. In this regard, networks of heterogeneous COTS resources can realize a very high level of aggregate performance in hyperspectral imaging applications.

4. SPECIALIZED HARDWARE IMPLEMENTATIONS

Despite hypespectral imaging algorithms generally map quite nicely to parallel systems made up of commodity CPUs, 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 essential to reduce mission payload and obtain analysis results in real time, i.e., at the same time as the data are collected by the sensors. In this regard, an exciting new development in the field of commodity computing is the emergence of programmable hardware devices such as field programmable gate arrays (FPGAs) [8], which can bridge the gap towards onboard and real-time analysis of remote sensing data. FPGAs are now fully reconfigurable, which allows one to adaptively select a data processing algorithm (out of a pool of available ones) to be applied onboard the sensor from a control station on Earth.

On the other hand, the emergence of commodity graphics processing units GPUs [9] (driven by the ever-growing demands of the video-game industry) has allowed these systems to evolve from expensive application-specific units into highly parallel and programmable commodity components. Current GPUs can deliver a peak performance of 1 Teraflop (e.g., the NVidia Tesla C1060[1] GPU) and up to 4 Teraflops (e.g., the NVidia Tesla S1070[2] GPU) which is several times the performance of the fastest dual-core processor available. The ever-growing computational demands of hyperspectral imaging applications can fully benefit from compact hardware components and take advantage of the small size and relatively low cost of these units as compared to clusters or networks of computers.

5. IMPLEMENTATION CASE STUDY

This section provides an implementation example based on spectral unmixing. It should be noted that many other relevant implementation cases (e.g., detection, classification, band selection, compression) can be used to illustrate the advantages of parallel implementations. Due to space limitations, in this study we will only focus on the unmixing case study, leaving other examples for future extensions of this work.

5.1. Processing chain

A standard approach to decompose mixed pixels in hyperspectral images is linear spectral unmixing, which comprises the following stages. Firstly, a set of pure spectral signatures (endmembers) are extracted from the input data set. Then, the fractional abundances of such endmembers in each pixel of the scene are obtained using an inversion process. The chain can be summarized by the following steps [7]:

1. *Skewer generation*. Produce a set of *K* randomly generated unit vectors, denoted by $\{\mathbf{skewer}_j\}_{j=1}^{K}$.

2. *Extreme projections*. For each $\mathbf{skewer}_j$, all sample pixel vectors $\mathbf{f}_i$ in the original data set $\mathbf{F}$ are projected onto $\mathbf{skewer}_j$ via dot products of $|\mathbf{f}_i \cdot \mathbf{skewer}_j|$ to find sample vectors at its extreme (maximum and minimum) projections, forming an extrema set for $\mathbf{skewer}_j$ which is denoted by $S_{extrema}(\mathbf{skewer}_j)$.

3. *Calculation of pixel purity scores*. Define an indicator function of a set *S*, denoted by $I_S(\mathbf{f}_i)$, to denote membership of an element $\mathbf{f}_i$ to that particular set as $I_S(\mathbf{f}_i) = 1$ if $\mathbf{f}_i \in S$. Using the function above, calculate the number of times that given pixel has been selected as extreme using the following equation:

$$N_{times}(\mathbf{f}_i) = \sum_{j=1}^{K} I_{S_{extrema}(\mathbf{skewer}_j)}(\mathbf{f}_i) \qquad (1)$$

4. *Endmember selection*. Find the pixels with value of $N_{times}(\mathbf{f}_i)$ above v_c and form a unique set of E endmembers $\{\mathbf{e}_e\}_{e=1}^{E}$ by calculating the spectral angle distance (SAD) for all possible endmember pairs and discarding those which result in an angle value below v_a.

5. *Spectral unmixing*. For each sample pixel vector $\mathbf{f}_i$ in $\mathbf{F}$, a set of abundance fractions specified by $\{a_{\mathbf{e}_1}, a_{\mathbf{e}_2}, \cdots, a_{\mathbf{e}_E}\}$ is obtained using the set of endmembers $\{\mathbf{e}_e\}_{e=1}^{E}$ by minimizing the noise term $\mathbf{n}$ in the following expression: $\mathbf{f}_i = \mathbf{e}_1 \cdot a_{\mathbf{e}_1} + \mathbf{e}_2 \cdot a_{\mathbf{e}_2} + \cdots + \mathbf{e}_E \cdot a_{\mathbf{e}_E} + \mathbf{n}$.

5.2. Cluster-based implementation

To reduce code redundancy and enhance reusability, our goal was to reuse much of the code for the sequential algorithm in the different parallel implementations. For that purpose, we adopted a spatial-domain decomposition approach [5] that subdivides the image cube into multiple blocks made up of entire pixel vectors, and assigns one or more blocks to

[1] http://www.nvidia.com/object/product_tesla_c1060_us.html
[2] http://www.nvidia.com/object/product_tesla_s1070_us.html

each processing element in master-slave fashion. It should be noted that most hyperspectral processing algorithms are based on calculations in which pixels are always treated as entire spectral signatures. Therefore, a spectral-domain partitioning scheme (which subdivides the whole multi-band data into blocks made up of contiguous spectral bands or sub-volumes, and assigns one or more sub-volumes to each processing element) is not appropriate in our application because, in this case, the calculations made for each hyperspectral pixel need to originate from several processing elements, thus requiring intensive inter-processor communication.

5.3. Distributed implementation

In order to balance the load of the processors in a heterogeneous parallel environment, each processor should execute an amount of work that is proportional to its speed. Therefore, two major goals of data partitioning in heterogeneous networks should be: to obtain an appropriate set of workload fractions that best fit the heterogeneous environment, and to translate the chosen set of values into a suitable decomposition of the input data, taking into account the properties of the heterogeneous system. In order to accomplish the above goals, we have developed a workload estimation algorithm for heterogeneous networks that assumes that the workload of each processor must be directly proportional to its local memory and inversely proportional to its speed [7]. The parallel algorithm has been implemented using the C++ programming language with calls to standard *message passing interface* (MPI) functions (also used in our cluster-based implementation).

5.4. FPGA implementation

Our hardware-based parallel strategy for implementation of the hyperspectral data processing chain is aimed at enhancing replicability and reusability of slices in FPGA devices through the utilization of systolic array design [8]. Fig. 1 describes the systolic array design adopted in previous work. Here, local results remain static at each processing element, while pixel vectors are input to the systolic array from top to bottom and skewer vectors are fed to the systolic array from left to right. In Fig. 1, asterisks represent delays. The processing nodes labeled as *dot* in Fig. 1 perform the individual products for the skewer projections. On the other hand, the nodes labeled as *max* and *min* respectively compute the maxima and minima projections after the dot product calculations have been completed. In fact, the *max* and *min* nodes can be respectively seen as part of a 1-D systolic array which avoids broadcasting the pixel while simplifying the collection of the results. The design was ported to hardware using Handel-C, a prototyping language.

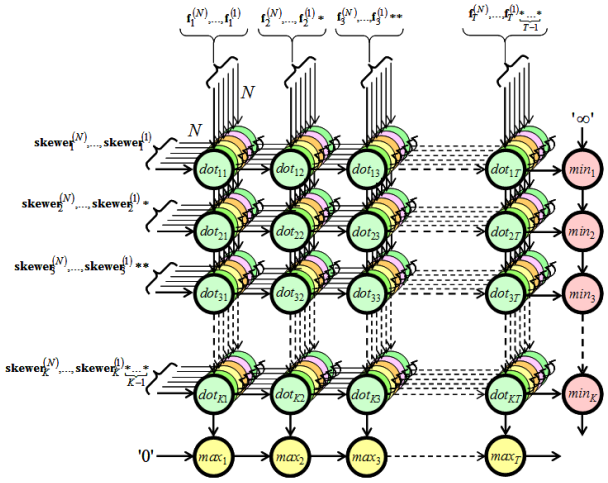

Fig. 1. Systolic array design for the FPGA implementation.

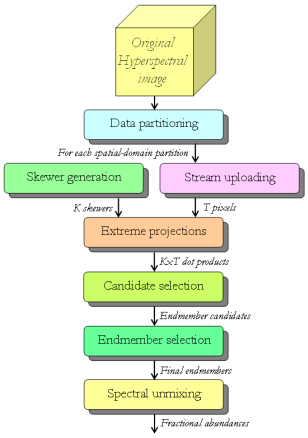

Fig. 2. GPU implementation.

5.5. GPU implementation

In order to accommodate the spatial-domain partitions onto the memory of an NVidia GPU, each partition is further subdivided into 4-band tiles (called spatial-domain tiles), which are arranged in different areas of a 2-D texture [9]. Such partitioning allows us to map four consecutive spectral bands onto the RGBA color channels of a texture (memory) element. Apart from the tiles, we also allocate additional memory to hold other information, such as the skewers and intermediate dot products, norms, and SAD-based distances. Figure 2 shows a flowchart describing our GPU-based implementation, which has been developed using the compute unified device architecture (CUDA). The *data partitioning* stage performs the spatial-domain decomposition of the original hyperspectral image. In the *stream uploading* stage, the spatial-domain partitions are uploaded as a set of tiles onto the GPU memory. The remaining kernels implement the different stages of the considered processing chain on the GPU.

Table 1. Performance on Thunderhead (4 to 256 CPUs) and the heterogeneous network (16 CPUs).

	Thunderhead Beowulf cluster								Heterogeneous network
# CPUs	**4**	**16**	**36**	**64**	**100**	**144**	**196**	**256**	**16**
Sequential computations	1.63	1.26	1.12	1.19	1.06	0.84	0.91	0.58	1.69
Parallel computations	292.09	73.24	30.46	15.44	8.76	5.08	3.18	1.91	79.56
Communications	2.20	2.41	2.39	2.21	2.46	2.65	2.32	2.49	3.56
Total time	295.92	76.91	33.97	18.84	12.38	8.57	6.41	4.98	83.05
Speedup	3.93	15.12	34.23	61.73	93.89	135.67	181.34	233.45	13.23

6. EXPERIMENTAL RESULTS

Four HPC platforms have been used in experiments: 1) Thunderhead Beowulf cluster at NASA's Goddard Space Flight Center in Maryland, composed of 256 dual 2.4 GHz Intel Xeon nodes, each with 1 GB of memory and 80 GB of main memory, interconnected with 2 Ghz optical fibre Myrinet; 2) A fully heterogeneous network which consists of 16 different workstations and four different communication segments; 3) A Xilinx Virtex-II XC2V6000-6 FPGA of Celoxica's ADMXRC2 board; 4) An NVidia GeForce 8800 GTX GPU with 16 multiprocessors. Both the GPU and the FPGA were connected to a 2006-model HP xw8400 workstation with dual Quad-Core Intel Xeon processor E5345 running at 2.33 GHz with 1.333 MHz bus speed and 3 GB RAM.

Table 1 shows the total time spent by the parallel implementation of the hyperspectral processing chain when unmixing the well-known AVIRIS Cuprite reflectance data set[3], with 614×512 pixels and 224 spectral bands, on the Thunderhead Beowulf cluster and on the heterogeneous network. It can be seen from Table 1 that the times for sequential computations were always very low when compared to the time for parallel computations, which anticipates high parallel efficiency on both the cluster and the heterogeneous network, even for a high number of processors. On the other hand, the impact of communications was not particularly significant.

On the other hand, our experiments revealed that the GPU can process a standard AVIRIS scene in 18.93 seconds. This response is not strictly in real-time since the cross-track scan line in AVIRIS, a pushbroom instrument [2], is quite fast (8.3 msec). This introduces the need to process a full image cube (614×512) pixels with 224 bands in no more than 5 seconds to achieve real-time performance. The speedup achieved by the GPU implementation over the multi-core CPU one was about 26, which doubles the speedup reported for the heterogeneous network in Table 1 at much lower cost. Finally, Table 2 shows a summary of resource utilization by the FPGA implementation. Here, the speedup was around 24, with processing time of about 20 seconds. Although better speedups can be obtained by increasing hardware utilization, it is very important to leave room in the FPGA so that dynamic algorithm selection and full reconfigurability can be achieved (as shown by Table 2, when 400 processors are used, 36% of available resources in the FPGA are already consumed).

[3]Available online: http://aviris.jpl.nasa.gov/html/aviris.freedata.html

Table 2. Resource utilization for the FPGA implementation.

Number of processors	Total gates	Total slices	% of total	Frequency (MHz)	Time (seconds)
100	97,443	1,185	3%	29,257	69.91
200	212,412	3,587	10%	21,782	35.36
400	526,944	12,418	36%	18,032	20.48

7. CONCLUSIONS

In this paper, we have discussed several techniques for parallel processing of hyperspectral images on commodity clusters, heterogeneous networks, FPGAs and GPUs. An example case study, focused on efficient implementation of a standard hyperspectral unmixing chain, has also been presented and discussed. Our study reveals that the computational power offered by commodity clusters and heterogeneous platforms needs to be evaluated in terms of the time-critical constraints introduced by many remote sensing applications, in which FPGA and GPU-based implementations may offer advantages such as reduced cost and less restrictions in terms of power consumption, size and weight, which are important when defining mission payload.

8. REFERENCES

[1] A. F. H. Goetz, G. Vane, J. E. Solomon, B. N. Rock, "Imaging spectrometry for Earth remote sensing," *Science*, vol. 228, pp. 1147-1153, 1985.

[2] R. O. Green et al., "Imaging spectroscopy and the airborne visible/infrared imaging spectrometer," *Remote Sensing of Environment*, vol. 65, pp. 227-248, 1998.

[3] A. Plaza and C.-I Chang, *High Performance Computing in Remote Sensing*, CRC Press, Boca Raton: FL, 2007.

[4] Q. Du and J. Fowler, "Low-complexity principal component analysis for hyperspectral image compression," *International Journal of High Performance Computing Applications*, vol. 22, no. 4, pp. 438-448, 2008.

[5] A. Plaza, D. Valencia, J. Plaza, and P. Martinez, "Commodity cluster-based parallel processing of hyperspectral imagery," *Journal of Parallel and Distributed Computing*, vol. 66, no. 3, pp. 345-358, 2006.

[6] Y.-L. Chang, J.-P. Fang, H. Ren, L. Chang, W.-Y. Liang, and K.-S. Chen, "Parallel simulated annealing approach to band selection for hyperspectral imagery," *Proceedings of IGARSS*, Boston, MA, pp. 994-997, 2008.

[7] A. Plaza, D. Valencia, and J. Plaza, "An experimental comparison of parallel algorithms for hyperspectral analysis using homogeneous and heterogeneous networks of workstations," *Parallel Computing*, vol. 34, no. 2, pp. 92-114, 2008.

[8] A. Plaza and C.-I Chang, "Clusters versus FPGA for parallel processing of hyperspectral imagery," *International Journal of High Performance Computing Applications*, vol. 22, no. 4, pp. 366-385, 2008.

[9] J. Setoain, M. Prieto, C. Tenllado, 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.