A MULTILEVEL PARALLEL AND SCALABLE SINGLE-HOST GPU CLUSTER FRAMEWORK FOR LARGE-SCALE GEOSPATIAL DATA PROCESSING

Grant J. Scott and Kirk Backus

University of Missouri
Center for Geospatial Intelligence
Columbia, Missouri, USA

Derek T. Anderson

Mississippi State University
Department of Electrical and Computer Engineering
Mississippi State, MS, USA

ABSTRACT

Geospatial data exists in a variety of formats, including rasters, vector data, and large-scale geospatial databases. There exists an ever-growing number of sensors that are collecting this data, resulting in the explosive growth and scale of high-resolution remote sensing geospatial data collections. A particularly challenging domain of geospatial data processing involves mining information from high resolution remote sensing imagery. The prevalence of high-resolution raster geospatial data collections represents a significant data challenge, as a single remote sensing image is composed of hundreds of millions of pixels. We have developed a robust application framework which exploits graphics processing unit (GPU) clusters to perform high-throughput geospatial data processing. We process geospatial raster data concurrently across tiles of large geospatial data rasters, utilizing GPU co-processors driven by CPU threads to extract refined geospatial information. The framework can produce output rasters or perform image information mining to write data into a geospatial database.

Index Terms— high performance computing, GPU clusters, geospatial data processing, image information mining

1. INTRODUCTION

We continue to see an explosion in the growth of high-resolution remote sensing geospatial data collections, such as EO, SAR, LIDAR, and hyperspectral rasters. This geospatial raster data has a variety of uses including mapping (e.g., vector data extraction), climate research, monitoring activities, etc. The prevalence of geospatial data collections represents a significant *big data* challenge, as a single remote sensing image is composed of hundreds of millions of pixels. Geospatial rasters vary in the type of content and amount of information for each pixel. Modern commercial EO remote sensing imagery may be panchromatic (e.g., single band, Digital Globe's Worldview-1) or multispectral (e.g., up to eight bands, Digital Globe's Worldview-2). Hyperspectral data may have over a hundred raster bands per pixel. Other geospatial raster data types exist as well, such as radar and digital elevation models. The variety and scale of geospatial data necessitates the development of general purpose high-performance processing frameworks in order to apply methods of ever increasing computational complexity to these massive data sets.

Large-scale, automated geospatial processing is a significant challenge; which is often overlooked or circumvented during the development or adaption of state-of-the-art techniques. Many non-trivial (in theory and implementation) state-of-the-art image processing and computer vision algorithms do not scale well to the size of full remote sensing imagery scenes. This has inspired an ever-increasing amount of research into parallel architectures as computing solutions for geospatial data. There are a variety of approaches to handle large geospatial rasters, such as partitioning data within a distributed system. However, many approaches involve hardware solutions which may be cost-prohibitive for some researchers. A promising, cost-effective solution to geospatial raster data processing is the use of modern GPU hardware and *general purpose GPU* (GPGPU) programming.

Various research has exploited GPU hardware in remote sensing applications, as well as numerous other fields. In [1], the authors utilize CG and OpenGL to implement minute change detection of tightly controlled image pairs. The authors acknowledge the limitations of the raster sizes that can be computed on the GPU and offer that partitioning large satellite images can circumvent this limitation. However, the authors do not provide any strategies regarding how to coordinate the partitioning, processing, and re-assembly of the large satellite rasters. Instead they conduct their experiments on a small 1150 x 1300 sub-image. [2] provides an in-depth treatment of implementing the hyperspectral remote sensing data *Lossy Compression for Exomars* algorithm on the GPU using the CUDA API. However, their approach requires the entire raster to be copied to the GPU global memory prior to launching the kernel.

The use of GPUs for solving large data problems continues to grow. In [3], the authors present a document clustering algorithm and discuss the challenges and limitations utilizing GPU hardware. [4] presents a graph-clustering algorithm

which employs the GPU to produce *k-nearest neighbor* networks, then again to reduce to clusters (i.e., sub-graphs) via a *minimum spanning tree* algorithm on the GPU. In [5], a hyperspectral imagery parallel target detection algorithm is implemented on GPU and performs statistically equivalent to a 32-CPU cluster. Many of these approaches report novel use of GPU hardware to accelerate geospatial data processing; however they often limit their experimentations to substantially smaller cropped sub-scenes.

To mitigate the limits of a single hardware element, GPUs are being integrated into traditional computing clusters. In these environments, *divide and conquer* data partitioning is standard practice. Numerous frameworks exist to facilitate distributed computation, include message passing and distributed shared memory frameworks. As GPU co-processors become more prevalent, we see an increased integration of GPGPU programming and distributed processing frameworks. In [6], the authors detail the use of *message passing interface* (MPI) in conjunction with CUDA for processing large Landsat 7 images. This represents one style of GPU cluster, whereby each node in a traditional cluster has a GPU co-processor, and existing parallel distributed processing techniques are leveraged to partition the large imagery to nodes. In both [7] and [8], distributed shared memory systems coupled with GPU-enabled cluster nodes are utilized to solve large data problems through partitioning schemes.

In the context of this paper, we refer to computing equipment with more than one GPU co-processor as a *single-host GPU cluster*. From an infrastructure perspective, one machine with N GPUs is more cost effective and easily utilized for high-performance geospatial data processing than N network-linked machines each with single a GPU. Additionally, the data communication latencies of networks is significantly more than that of the PCI express bus of modern computers.

We have developed a robust framework which exploits single-host GPU clusters to perform high-performance geospatial raster data processing. The framework is a hierarchy of parallelism. The first level of parallelism (host-level) steps through large sub-rasters (i.e., image tiles) of geospatial raster data in parallel. These sub-rasters are concurrently pushed onto GPU devices, where device-level parallelism for pixel processing is a fundamental tenant of GPGPU programming.

We have applied this framework to two initial processing algorithms; one performing EO imagery processing and another performing DEM information mining. In each case, we measure the computational time of CPU and GPU algorithm implementations within the framework, where the desired module for processing the sub-raster is selected. Our results show significant acceleration of the processing algorithms in our multilevel parallel computing framework, compared to performing the same processing using only CPU-thread parallelism.

The remainder of this paper is organized as follows. Section 2 provides the details of our geospatial data processing application framework which supports processing large remote sensing imagery. The use of GPU clusters are exploited within our framework is detailed in Section 3. Section 4 provide an example case-study and some emperical evaluations. Concluding remarks and future work are presented Section 5.

2. MULTILEVEL PARALLEL GEOSPATIAL DATA PROCESSING FRAMEWORK

Geospatial data processing is inherently a *big data* problem. For instance, full satellite imagery often exceeds hundred of millions of pixels. We have developed a robust geospatial data processing application framework that simultaneously exploits multi-CPU/core and multi-GPU computing to facilitate high-performance geospatial data processing. Figure 1(a) is an overview of the data flow of the application framework, how a geospatial data set moves through the framework, and how multiple GPUs are concurrently leveraged for processing. The processing can be broken down into five general stages: 1) parameterization and loading, 2) preprocessing and preparation of data, 3) multithreaded CPU driving of GPU cluster, 4) multiple GPU kernel execution, 5) sinking results as output rasters or database content. Figure 1(b) illustrates the relationship between the two levels of parallelism in the framework. A large geospatial data raster is divided into sub-rasters, which are parallel processed using a *threadpool pattern*. Each GPU represents a secondary level of parallelism in the context of the *grid of blocks*, which further logically partition each sub-raster for parallel processing on the GPU.

The application collects parameters for processing from either the command line and/or a database. Common parameters necessary for this framework include maximum number of threads for processing, minimum GPU compute capability, and the specifications of the input and output geospatial data. Based on the parameters, the large geospatial data is logically partitioned into tiles, which may include overlapping buffers if necessary for the particular processing algorithms. Each logical tile of the geospatial data to be processed will be loaded, either from a network or local storage, into memory as needed for the processing by a CPU core. The size of the tiles is a parameter driven by the goal of achieving optimal utilization of the GPU co-processors. Each CPU thread will load a tile into memory, then await access to a GPU. Once a GPU is acquired by a thread, the tile is pushed onto the GPU, the GPU kernel is launched, and finally the processing results are copied back to CPU thread memory. The thread will release the GPU, and sink the data either into a raster or database. Each CPU thread will continue this process until there are no more tiles to process.

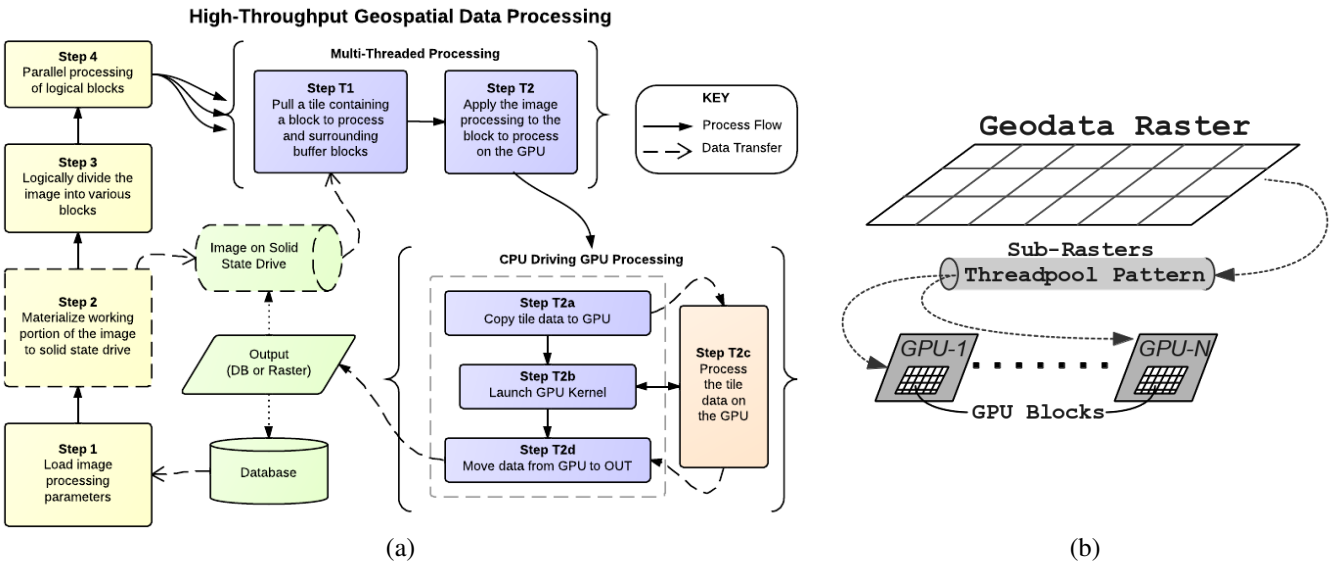

Fig. 1. (a) Framework for utilizing single-host GPU clusters to process large amounts of geospatial data, concurrently processing sub-rasters of the large raster. (b) Logical view of multilevel parallelism, where the CPU threads concurrently process sub-rasters and the GPUs further parallelize the sub-raster processing.

3. GPU KERNEL DESIGN CONSIDERATIONS

GPU kernels are executed under the *single instruction, multiple thread* (SIMT) paradigm; whereby a single kernel executes on the stream of *blocks*. Each block is processed in parallel by the number of threads as designed by the block dimension. Blocks are processed on the GPU *streaming multiprocessors* (SM), effectively processing all threads in the block in parallel. The set of blocks is generated from the logical configuration of the *grid*, and the grid is typically used to model the 1-, 2-, or 3-D input and/or output data. The most natural example of this is the sub-division of the *raster tile* into 2-dimensional blocks; and the grid as a 2-dimensional layout of blocks. In many algorithms, a single logical thread maps one input pixel to one output pixel. However, as is often the case in more advanced geospatial (and image) processing, a pixel must be considered in spatial context of neighboring pixels. Examples include texture analysis, morphological operations, and filtering; just to list a few. For example, the two fundamental morphological operations, erode and dilate, are simply reductions to a minimum and maximum value, respectively, using a spatial local neighborhood (i.e., structuring element). In these geospatial data processing scenarios, one can use a block to represent one output, and all the threads of the block read a spatial set of input values. This in commonly referred to as the reduction paradigm in GPGPU programming.

When a block is used to represent one output element, a single pixel in geospatial the raster case, the GPU hardware cannot be expected to fit the entire input raster into the logical grid structure. This highlights another benefit of our multilevel parallelism, as it allows the sub-raster tiles to be logically coordinated with the GPU algorithm and logical structures. Additionally, our multilevel parallel logical organization allows the design of blocks that are capable of effectively exploiting GPU shared memory. The GPU shared memory, which is localized to each GPU SM, is a limited and valuable resource for GPU algorithm optimization [9].

4. FRAMEWORK APPLICATIONS AND PERFORMANCE

Herein, we report two instances of processing large-scale geospatial raster data for different purposes using the proposed framework. In each case, the data set processed exceeds the capacity of the system GPU resources; requiring the use of the framework to seamlessly process the data. The implementations for both CPU and GPU are optimized, and the same sub-raster tile sizes are used when collecting timing data. Measurements regarding GPU timing include the time to perform data copies between host and device. The test system includes twelve CPU cores (Intel Xeon X5670) and four GPUs (Nvidia Tesla C2075).

EO raster processing: We have applied this framework to perform mean-shift clustering of EO satellite imagery. Our testing image is a raster of size 18944x14848, i.e., 281.3 million pixels. Mean-shift clustering is useful for segmenting imagery, which is a critical initial step in many advanced image processing algorithms such as object-based image analysis. The image is partitioned into tiles of 512x512 pixels. The data processed for each tile includes a 7 pixel buffer, i.e., 526x526. The grid is setup as 512x512 blocks, and each block uses a set of spatially organized threads to compute the mean

shift of the pixel driven by spectral similarity within its spatial neighborhood. The output of this process is a final spatial position of each input pixel, which can be post processed to generate segments. The per-pixel processing rate is 1.0 ms and 0.0036 ms per pixel for CPU and GPU, respectively. This is an effective processing throughput of 11 689.9 pixels/s for the twelve CPU cores versus 1 095 234.1 pixels/s for the four GPU; i.e., the framework achieves 93x pixel-throughput acceleration on our hardware.

DEM information mining: We utilized this framework to perform information mining from digital elevation model (DEM) data, such as [10]. The portion of the worldwide DEM that is being processed is materialized from network storage to local storage. We implemented an algorithm to extract terrain-sky silhouettes from DEM data, which are then stored into a geospatially enabled database. The processing area is partitioned into tiles of 128x128 pixels. The data processed for each tile includes a 2048 pixel buffer, resulting in processing sub-rasters of 4224x4224 pixels. For each of the 128x128 center pixels, the algorithm marches approximately 61 kilometers (38 miles) in 360 directions to capture the terrain silhouette. The output per pixel is a 360 element array representing the terrain silhouette. This output is stored in a database along with the geospatial coordinates. The per-pixel processing rate is 74.9 ms and 0.3 ms per pixel for CPU and GPU, respectively. This is an effective processing throughput of 160.3 pixels/s for the twelve CPU cores versus 13 133.5 pixels/s for the four GPU, i.e., an 81x acceleration.

5. CONCLUSION

We have presented a multilevel parallel framework to exploit single-host GPU clusters for high-throughput geospatial processing. This framework supports parallel sub-raster tile processing at the CPU/core level, as well as parallelization of algorithms to process each tile on the GPU as a stream of blocks. Our framework allows the processing of arbitrarily large rasters, exploiting GPGPU algorithm acceleration. It provides flexibility to optimize how each level of the framework is parallelized, i.e., the sub-raster tiles, GPU grids, and GPGPU algorithms. We have provided illustrative examples of two geospatial data processing tasks, using moderately complex algorithms and non-trivial input data sets. In both cases, significant acceleration of the algorithm is achieved by utilizing the second level of parallelism via GPGPU programming. The framework, on our testing hardware, reveals an 93x and 81x acceleration in the pixels/sec processing rate of the EO and DEM algorithms, respectively, by leveraging the GPU.

In future work we will continue to apply our GPU-cluster powered geospatial data processing framework to other challenging remote sensing tasks. We will incorporate algorithms for information mining to extract vector data from rasters, as well as additional imagery content data. We intend to develop an open source repository for this framework with numerous geospatial data processing algorithms to facilitate wider use within, and contributions from, the geosciences community.

6. REFERENCES

[1] S. Kockara, T. Halic, and C. Bayrak, "Real-time minute change detection on gpu for cellular and remote sensor imaging," in *Advanced Information Networking and Applications Workshops, 2009. WAINA '09. International Conference on*, 2009, pp. 13–18.

[2] L. Santos, E. Magli, R. Vitulli, J. Lopez, and R. Sarmiento, "Highly-parallel gpu architecture for lossy hyperspectral image compression," *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of*, vol. PP, no. 99, pp. 1–12, 2013.

[3] X. Cui, J. St.Charles, and T. Potok, "The gpu enhanced parallel computing for large scale data clustering," in *Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2011 International Conference on*, 2011, pp. 220–225.

[4] A. Arefin, C. Riveros, R. Berretta, and P. Moscato, "knn-mst-agglomerative: A fast and scalable graph-based data clustering approach on gpu," in *Computer Science Education (ICCSE), 2012 7th International Conference on*, 2012, pp. 585–590.

[5] A. Paz and A. Plaza, "Cluster versus gpu implementation of an orthogonal target detection algorithm for remotely sensed hyperspectral images," in *Cluster Computing (CLUSTER), 2010 IEEE International Conference on*, 2010, pp. 227–234.

[6] M. Rumanek, T. Danek, and A. Lesniak, "High performance image processing of satellite images using graphics processing units," in *Geoscience and Remote Sensing Symposium (IGARSS), 2011 IEEE International*, 2011, pp. 559–561.

[7] E. Kijsipongse and S. U-ruekolan, "Dynamic load balancing on gpu clusters for large-scale k-means clustering," in *Computer Science and Software Engineering (JCSSE), 2012 International Joint Conference on*, 2012, pp. 346–350.

[8] A. E. K. Zhe Fan, Feng Qiu, "Zippy: A framework for computation and visualization on a gpu cluster," *Computer Graphics Forum*, vol. 27, no. 2, pp. 341–350, 2008.

[9] NVIDIA Corporation, *NVIDIA CUDA C Programming Guide*, June 2011.

[10] http://asterweb.jpl.nasa.gov/gdem.asp.