Towards Personal High-Performance Geospatial Computing (HPC-G): Perspectives and a Case Study

Jianting Zhang
Department of Computer Science
City College of New York
New York City, NY, 10031
jzhang@cs.ccny.cuny.edu

ABSTRACT

Cluster computing, Cloud computing and GPU computing play overlapping and complementary roles in parallel processing of geospatial data within the general HPC framework. The fast increasing hardware capacities of modern personal computers equipped with chip multiprocessor CPUs and massively parallel GPUs have made high performance computing of large-scale geospatial data in a personal computing environment possible. We discuss the framework of Personal HPC-G and compare it with traditional Cluster computing and the newly emerging Cloud computing. We consider Personal HPC-G possesses many favorable features: low initial and operational costs, good support for data management and excellent support for both numeric modeling and interactive visualization. A case study on developing a parallel spatial statistics module for visual explorations on top of Personal HPC-G is subsequently presented.

1. INTRODUCTION

High-Performance Computing (HPC) is an important component of Geospatial Cyberinfrastructure (GCI) and is critical to large-scale geospatial data processing and problem solving [1][2]. While traditionally HPC mostly refers to parallel numerical calculations on supercomputers equipped with specially designed hardware and software (such as vector processing, optimized memory hierarchy and parallel file system), modern HPC architectures increasingly rely on commodity hardware and software to achieve a high cost-effectiveness ratio. As a result, more and more cluster computers are appearing in the Top500's list of fastest computers [3]. Many institutions and departments now own cluster computers of different sizes. Accessibility of HPC resources to general researchers has been significantly increased over the past few years. The recently emerging Cloud Computing [4] makes HPC resources "rentable" which opens tremendous opportunities for general users to speed up their computing intensive tasks and enable solving large-scale problems, including geospatial data processing.

While simultaneously accessing multiple Web-GIS services can be viewed as a single-step parallel processing, speeding up geospatial data processing on HPC facilities based on fine-grained parallel computing paradigms is still a significant research challenge. As discussed in [5], research on parallel processing of geospatial data has very little impact on mainstream geospatial data processing applications prior to 2003 despite archived research efforts in designated journal special issues and books [6][7]. Creating parallel GIS operations is non-trivial and there is a lack of parallel GIS algorithms, application libraries and toolkit [5]. Fortunately, in recent years, the increasingly available computer clusters and Cloud computing resources have spawned quite a few pioneering studies on HPC-G, such as spatial statistics [8][9], hyperspectral image processing [10], LIDAR data processing[11], extracting drainage networks [12] and agent-based modeling [13][14], in addition to experimenting with parallelized environmental models [15].

In parallel with the development of large-scale cluster-based computing technologies (or Cluster computing for short), Graphics Processing Unit (GPU) computing represents a quite different type of parallel computing paradigm. Originally designed as an accelerator to CPU, a modern GPU device has multiple identical processors to speed up 2D and 3D graphics rendering by adopting the Single-Instruction Multiple-Data (SIMD) architecture and rendering screen pixels independently. The concept of General Purpose GPU (GPGPU) turns the massive floating-point computational power of a modern graphics accelerator's graphics-specific pipeline into general-purpose computing power [16]. GPU devices are becoming more powerful yet affordable due to massive demands from game and entertainment industries. As an example, an Nvidia Fermi-based GeForce GTX480 GPU device has 480 cores with a peak performance of 1.35T flops [17] and is now available from market for $500. As many reasonably current desktop computers have already equipped with GPGPU enabled graphics cards, GPGPU based processing of geospatial data can improve system performance significantly without additional costs. GPGPU have gained considerable interests in many scientific research areas in the past few years [18][19].

We believe that Cluster computing, Cloud computing and GPU computing play overlapping and complementary roles in parallel processing of geospatial data within the general HPC framework. Different from traditional HPC applications that mostly run in a batch mode on shared resources, GPU computing provides a personal computing environment on desktop computers which is attractive to certain types of applications and groups of users. Parallel computing of geospatial data on GPGPU devices is ideal for visual and interactive applications, in a way similar to computer graphics applications. While Cloud computing is closely related to Cluster computing and other traditional HPC from a technological perspective, GPU computing is less familiar to the geospatial community. In this paper, we provide an introduction to GPU computing after discussing a few issues related to HPC on geospatial data. We outline a few perspectives on integrating GPGPU technologies with modern multiprocessor technologies for parallel geospatial data processing in a personal computing environment and term the new framework as Personal HPC-G. We also present a case study on parallelizing Geographically Weighted Regression (GWR) on GPUs at the conceptual level to exemplify how the proposed Personal HPC-G framework can be realized.

2. GEOSPATIAL DATA, GIS, SPATIAL DATABASES AND HPC

2.1 Geospatial data: what's special?

The differences between geospatial data and relational data from data modeling perspective are well understood. Our focus in this subsection is to address how geospatial data partition affects the throughputs of parallel geospatial data processing. It is well known that the slowest processing unit determines the overall performance in parallel computing and it is best to evenly distribute workloads to multiple units as much as possible. Unfortunately, real world data very often are skewed. As an example, 70% of the Earth is covered by oceans and they are often excluded from terrestrial pattern analysis and modeling. In many geospatial computing, e.g., window query and spatial interpolation, the number of data items within the neighborhood of a focal data item can grow with the degree of data skewness superlinearly. As such, the computing intensity can grow at least quadratically with the skewness in the densest area when an even space decomposition approach is adopted. Subsequently the overall parallel performance degrades at least quadratically with the skewness.

Techniques to handle skewness can generally be classified into two categories: data decomposition and task scheduling. Data decomposition techniques partition the data items into groups so that the computing workloads among all the groups are balanced. For example, dividing a set of points or raster cells into equal-sized groups will achieve workload balancing easily when only local operations are involved, e.g., increasing the Z value of the points by a certain amount. However, such simple decomposition may not work for focal, zonal or global operations as discussed above and a more sophisticated decomposition approach is needed to achieve workload balancing. Task scheduling techniques find an optimal task execution schedule to maximize parallel throughputs among a set of submitted tasks. As an example, while the simple equal-sized decomposition approach discussed above alone will not achieve workload balancing, when the computing workloads for the groups are submitted as individual tasks, it is possible to combine long-run tasks with short-run tasks and achieve workload balancing. In the simplest form, the computing task associated with each data item can be treated as a task and let the scheduling middleware (e.g., Condor [20]) to achieve workload balancing. While task scheduling techniques seem attractive as finding a good data decomposition scheme is usually difficult, we note that the complexities of task scheduling grow fast with the number of tasks and generic scheduling heuristics may not always produce good results [21]. The tasks, when running in parallel but poorly coordinated, can potentially compete intensively for shared resources, such as network bandwidth, disk I/Os, memory accesses, cache lines and registers, which can potentially decreases the parallelism among the tasks. As a result, normalized speedups usually decrease as the number of computing nodes increases in many Cluster computing applications, as reported in [8][13][14][15] among others.

2.2 GIS: impacts of hardware architectures

Traditionally geospatial data are managed and processed by Geographical Information Systems (GIS). GIS have been evolving along with mainstream information technologies. Examples are major platform shift from Unix workstations to Windows PCs in the early 1990s and the marriage with Web technologies to create Web-GIS in the late 1990s. Both desktop GIS and Web-GIS are now mature technologies and a large number of commercial and open source GIS exist and are ready to be applied to a variety of applications [22]. However, most of them are based on uniprocessor computer architecture and their performance is limited by the processing power of a uniprocessor (although multithreading technologies may have been incorporated in a few advanced products). While enterprise GIS products support multi-tasking and load balancing at the process level to a certain extent, usually they do not support fine-grained parallel computing that can be scaled up to a large number of processors for large-scale data and computing. As observed in [5], lacking software and tool support in GIS may have played a key role in discouraging geospatial applications to embrace HPC technologies. As the computing power increase for uniprocessors is coming to an end and modern computing architectures are shifting towards multiprocessors [23], it becomes urgent for GIS to integrate HPC technologies in order to efficiently process increasingly large volumes of geospatial data.

We consider GIS have three major roles, i.e., data management, information visualization and modeling support. Among these roles, GIS-based spatial modeling, such as agent based modeling, is naturally suitable for HPC for a couple of reasons. First, spatial models usually are computational intensive and require HPC resources to speed up computing. Second, many spatially explicit models and remote sensing data processing algorithms adopt a raster tessellation and mostly involve local operations and/or focal operations with small constant numbers of neighbors. These models are very parallelization-friendly or even "Embarrassingly parallel" [24]. Third, spatial models are mostly run in an offline mode and do not require significant user interactions. The outputs can be downloaded to local storage and be visualized in a desktop GIS environment. Loose integration of HPC and GIS do not require fundamental architecture changes of current GIS in this case. While outsourcing local and some focal GIS operations to HPC is certainly valuable, we consider it technically more challenging to parallelize complex raster and vector operations in GIS.

2.3 Spatial Database: parallel DB or MapReduce?

The data management functionality of GIS is increasingly relying on spatial databases backend. For example, the concept of Geodatabases plays a central role in ESRI's ArcGIS product line. PostGIS is essentially a plug-in module of PostgreSQL database. In addition, some Web-GIS (such as MapServer) connect with databases directly and use spatial databases to perform certain spatial operations without requiring a traditional GIS server. In the database community, while the importance of managing geographical and other spatial data has been recognized, it was not until the object-relational data model was widely adopted in the mid to late 1990s that major database systems began to support spatial data. Major commercial databases, such as Oracle, DB2 and SQL Server, now have modules to support spatial data. Indeed, Oracle claimed itself as "a GIS without GUI" [25]. Unfortunately, compared with physics-based numeric modeling communities, the database industry seems to be slow in adopting parallel computing. Despite the existence of commercially available parallel databases (see [26] for a brief overview), they are highly-priced and are largely inaccessible to general users. Furthermore, these parallel database systems/extensions are mostly designed for relational data and may not be applicable to geospatial data. Although some databases have both spatial and parallel modules, it is unlikely that they can be integrated to process large-scale geospatial data easily.

Parallel processing of geospatial data to achieve high performance is not a completely new concept and quite a few early works on parallel spatial data structures [27][28], spatial join [29][30], poylgonalization [31][32] and spatial clustering [33] have been reported. Unfortunately, very few of these early works have been incorporated into commercial or open source spatial databases. On the other hand, the increasing availabilities of Cluster and Cloud computing facilities and the MapReduce parallel computing framework [34] have motivated considerable research interests in applying MapReduce style computing abstractions to large-scale data processing. Cary et al [35] reported their experiences on processing spatial data with MapReduce using Google and IBM cluster and Hadoop [36]. Their experiments include R-Tree construction on point data and image tile quality computation. Another related work on developing a parallel algorithm to identify clusters from spatial data on shared-nothing Cloud computing resources is reported in [37] and a related work on handling skewness is reported in [38]. Comparisons between parallel database based and MapReduce based approaches to large scale data analysis have been reported [26][39]. Hybrid approaches have also been proposed [40][41]. Given that neither parallel databases nor MapReduce has been extensively applied to practical large-scale geospatial data management, the community is free to adopt either approach. We thus call for pilot studies in experimenting the two approaches to provide insights for future synthesis.

2.4 HPC: many options

Compared with software, hardware changes are more prominent with respect to high-performance computing in the past few years. While the combination of architectural and organizational enhancements lead to 16 years of sustained growth in performance at an annual rate of 50% from 1986 to 2002, due to the combined power, memory and instruction-level parallelism problem, the growth rate has dropped to about 20% per year from 2002 to 2006 ([23], pp.3). Indeed, in 2004, Intel cancelled its high-performance uniprocessor projects and joined IBM and Sun to declare that the road to higher performance would be via multiple processors per chip (or Chip Multiprocessors, CMP) rather than via faster uniprocessors. Another prominent hardware breakthrough is the proliferation of GPU devices and the emergence of GPGPU computing [16].

While more details will be provided in the next section, many GPU devices now have floating computing power ranges from a few tens of GFlops to more than a TFlops, a level that only reached by supercomputers not many years ago. The GPGPU computing technologies allows utilizing GPU devices for general computing purposes although neither translating serial code or CPU-based parallel code nor developing GPGPU code from scratch is trivial. The new CMP and GPU accelerator architectures have significant impacts on Cluster computing (and possibly Cloud computing in the near future) as well. Although traditional computing nodes in a cluster computer are based on uniprocessor architecture, it is now necessary to fully utilize the multiple-cores on multiple CPUs available at individual computing nodes to improve Cluster computing performance. Also, as more and more GPU devices are attached to computing nodes, there are significant technical challenges to coordinate CPU and GPU cores to achieve maximum performance [43].

We next briefly compare four major existing parallel computing frameworks: CPU multi-cores (CPU-MC), GPU many-cores (GPU-MC), multiple CPU computing nodes (CPU-MN) and multiple CPU+GPU computing nodes (CPU+GPU-MN). While existing APIs and middleware packages, e.g., pthread/OpenMP for CPU-MC and MPI/Condor for CPU-MN, have been extensively tested, GPU-MC and CPU+GPU-MN based parallel computing are still young. Despite the differences in mapping analytical tasks and data blocks to computing units for parallel processing, workload balancing plays a key role in achieving high-throughputs. Compared with Cluster computing that usually has dedicated high-speed interconnections (e.g., Infiniband) among tightly coupled computing nodes (e.g., within a private local area network), Cloud computing usually utilizes public Internet to communicate among distributed nodes in different data centers which can be significant slower. Also the Cloud computing nodes may be less powerful than these of Cluster computing with respect to processor clock rate, memory capacity and disk I/O speed. According to [4], the bandwidth among Amazon EC2 computing nodes is 50±2MB/s, which is significantly lower than similar experiment results in a Cluster computing environment [42]. It is interesting to observe that many applications running on Cloud computing facilities adopt the coarse-grained MapReduce parallel computing framework (e.g., Hadoop) while a large portion of applications running on Cluster computing facilities adopt fine-grained message-passing parallel computing frameworks (e.g., MPI), although technically it is possible to run both Hadoop and MPI-based applications on Cluster and Cloud computing facilities. In addition to historical reasons, it is possible that the bursty nature of the Internet based communications may make precise synchronizations that are often required by MPI-based applications difficult and less efficient in practice. While using distributed file system, such as HDFS in Hadoop, provides excellent fault tolerance and improves scalability, making intermediate results persistent on distributed hard disks can be costly due to the expensive disk I/Os (typically only 100-200MB/s). On the other hand, as users have full control of rented Could computing facilities, it is much easier for them to build specific software stacks for their applications which has certain advantages over Cluster computing facilities that usually have strict security and access control policies.

Since the current generation GPUs have hundreds of cores and the numbers are comparable to the numbers of computing nodes/processors used in Cluster/Cloud computing applications, it is also interesting to compare Cluster/Cloud computing with GPU computing. As a marketing strategy, Nvidia calls a personal computer equipped with one or more of its high-end GPGPU cards as a personal supercomputer. Nvidia claimed that when compared to the latest quad-core CPU, Tesla 20-series GPU computing processors deliver equivalent performance at 1/20th of power consumption and 1/10th of cost. While the comparison made by Nvidia focused on floating point computing, for data intensive applications, GPU computing also has some attractive features. The most prominent one might be the high bandwidth (in the order of tens to hundreds of GB/s) between GPU device memory and processing cores when compared with the speeds of networks connecting shared-nothing computing nodes in a Cluster/Cloud computing environment (in the order of tens to hundreds of MB/s as discussed above). Although the data transfer rate between CPU and many GPU devices is currently limited by PCI-Express (8GB/s), when fully utilized, GPU computing can achieve high performance for data intensive applications.

3 PERSONAL HPC-G: A NEW FRAMEWORK

While some geospatial data processing tasks are computationally intensive, many more are data intensive in nature. In addition, some tasks may not have sufficient degrees of parallelism that can be deployed into a large number of processors. As discussed above, visualization and interaction are indispensable in geospatial data processing which renders a pure Cluster or Cloud computing approach inadequate in many cases, in addition to ownership costs (Cluster computing) or operational costs (Cloud computing) considerations. Given the fast increasing computing power provided by multi-core CPUs and many-core GPUs, we believe that the parallel computing power provided by the combined CPU and GPU hardware capabilities and facilitated by novel data structures and parallel algorithms have the potential to improve computing performance by one or two orders or even more, when compared to current GIS running on uniprocessors. The improved performance will not only solve old problems faster but also allow many traditionally offline data processing tasks run online in an interactive manner. The uninterrupted exploration processes are likely to facilitate novel scientific discoveries more effectively.

3.1 Why Personal HPC for geospatial data?

Table 1 lists high-level comparisons among Cluster, Cloud and Personal high-performance computing. Unlike Cluster/Cloud computing that can use any number of processors in theory, the maximum number of CPU cores that can be attached to a personal computer and the number of GPU cores per device/host are limited by manufacture product lines. This is the primary reason that we give “high” for scalability to Cluster/Cloud computing while give “medium” to Personal HPC from a theoretical perspective. However, we note that the numbers of CPU and GPU cores are growing fast every year [23][44]. On the other hand, very few Cluster/Cloud computing applications use more than a thousand computing nodes based on published sources while GPU devices already have 500+ cores. In Cluster/Cloud computing, normalized speedups can decrease significantly when the numbers of processors go beyond a threshold [8][13][14]15], possibly due to the combined factors of communication/scheduling overheads, workload imbalances and contention of shared resources. From a practical perspective, Personal HPC can have better scalability with the help from convenient debug and performance tuning tools. From an end-user perspective, we rate the required new software development efforts for Cloud computing as “Low” due to the wide availability of commercial software supports while Cluster computing and Personal HPC largely reply on individual and community development efforts at present. Compared to Cluster computing, personal HPC is fairly new and requires significant software development efforts especially when considering that existing CPU serial/parallel codebases need to be refactored for GPUs.

The combined advantages of low costs (initial+operational), good support for data management and excellent support for both numeric modeling and interactive visualization make Personal HPC an ideal candidate for HPC-G for a variety types of problems. As modern PCs have adopted Chip Multiprocessor (CMP) architectures, there are increasing supports from both industry and open source community. In addition to pthread, OpenMP is also supported by Intel and Microsoft compilers. Compared to parallel programming on CPUs (e.g. pthread, OpenMP and MPI), GPGPU computing technologies are less known to the geospatial community. As such, we next provide a brief introduction to GPGPU computing.

Table 1 High-Level Comparisons among Cluster Computing, Cloud Computing and Personal HPC

	Cluster Computing	Cloud Computing	Personal HPC
Initial cost	High	Low	Low
Operational cost	High	Medium	Low
End user control	Low	High	High
Theoretical scalability	High	High	Medium
User code development	Medium	Low	High
Data management	Low	Medium	Medium
Numeric modeling	High	Medium	High
Interaction & visualization	Low	Low	High

3.2 GPGPU Computing: a brief introduction

As many reasonably current desktop computers have already equipped with GPGPU enabled graphics cards, GPGPU based processing of geospatial data can improve system performance significantly without additional costs. Despite the differences among the GPGPU enabled devices and development platforms, a GPGPU device can be viewed as a parallel Single Instruction Multiple Data (SIMD) machine [16]. Although we will be mainly focusing on Nvidia’s Compute Unified Device Architecture (CUDA, [16]) enabled GPUs due to its popularity, we argue that the data structures and parallel algorithms developed for CUDA-enabled GPUs can be adapted to other types of GPUs, such as AMD/ATI Stream programming enabled ones.

While different models of Nvidia GPU cards have different architectures, CUDA-enabled GPU devices are organized into a set of Stream Multiprocessors (SMs). Each SM has a certain number of computing cores. All the cores in a SM share a certain amount of fast memory called shared memory and all the SMs have access to a large pool of global memory on the device. According to CUDA, developers write a special C-like code segments called kernels. The kernels are invoked by the companioning CPU code to run on GPU devices. The kernel code does not allow dynamic memory allocation and recursion which imposes significant technical challenges for many geospatial data processing applications that rely on these techniques, including tree indices constructions. CUDA based GPGPU programming makes it easier for task and data decomposition and subsequent parallel computing. Basically a developer specifies the sizes of the layout of the data to be processed in the units of data blocks and the number of threads to be launched inside a computing block. The GPU device is responsible for mapping the data blocks to the computing blocks through space and time multiplexing which is transparent to developers/users. Since each SM has limited hardware resources, such as the number of registers, shared memory and thread scheduling slots, a SM can accommodate only a certain number of computing blocks subjected to the combination of the constraints. Carefully selecting block sizes allows a SM to accommodate more blocks simultaneously and improve parallel throughputs. The memory hierarchy and processor layout is shown in the top-left part of Fig. 1.

3.3 Pipelining CPU and GPU workloads for performance

Compared with Cluster and Cloud computing facilities, the hardware capabilities of a personal computer are limited. To achieve the desired high-performance, it is crucial to fully utilize hardware capabilities of both CPU and GPU. We note that multiple GPU devices can be attached to a computer through the PCI-E interface to further enhance the theoretical peak computing power when necessary. To facilitate discussions, the following simplifications are assumed (1) CPU cores on different chipsets are flatted and are treated as symmetric units. For example, a dual quad-core machine will be considered as having eight identical CPU cores. (2) Only a single Nvidia Fermi based GPU device is attached to the machine.

Given a geospatial dataset D and a processing task T, the following high-level steps are suggested to coordinate CPU and GPU to achieve high performance. First, analyze the dataset and the nature of the processing task. Second, decompose the task into subtasks and identify data subsets that are associated with the subtasks. A m:n relationship should be assumed between the subtasks and sub datasets for generality. The subtasks can be classified into each of the three categories: data management, numeric modeling and interactive visualization as discussed previously. Third, for each subtask, determines whether they are suitable to be carried out on CPU, GPU or their combinations. An important factor to consider is the parallelizability of relevant data structures and algorithms. For those involve dynamic memory allocations, recursive function calls or are mostly bit/integer operations, they should be left for CPU executions. Similarly, for those that mostly involve floating point computations, they are ideal for GPU executions. For subtasks that do not exhibit sufficient parallelism, they should be left for CPU executions in principle.

It is beyond the scope of this paper to provide detailed scheduling algorithms to assign tasks to CPU and GPU and coordinate their executions. There are significant technical challenges in the respective areas. Instead, we provide an example to show the basic idea of pipelining CPUs and GPUs in Fig. 1. Assuming the task is to build a quadtree for a global raster dataset. As the 70% of the Earth surface is covered by ocean, many top level quadrants will have small percentages of cells with valid data. By assigning quadrants with dense valid cells to GPU and quadrants with sparse valid cells to CPUs, a reasonable load balancing may be achieved. While GPU cores and some of CPU cores are generating statistics necessary for quadrants in constructing the quadtree, some CPU cores can combine the derived subtrees into the final quadtree. The subtree generation and combination processes can be pipelined naturally.

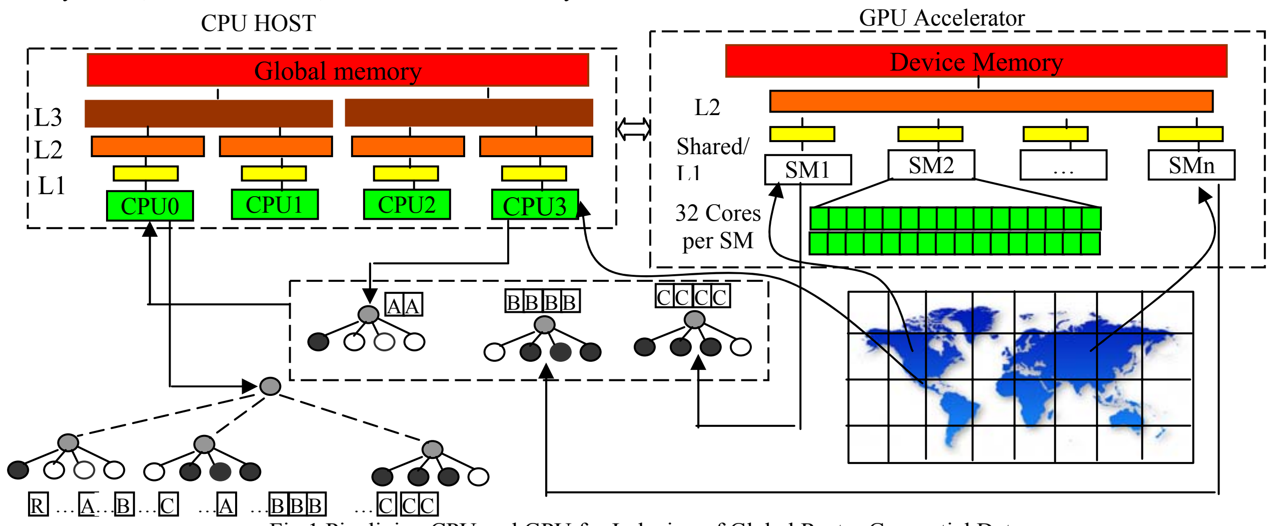

Fig.1 Pipelining CPU and GPU for Indexing of Global Raster Geospatial Data

3.4 Parallel GIS prototype development strategies

Most existing HPC-G applications on Cluster computing facilities targeted at specific applications. We are not aware of the existence of a comprehensive parallel GIS that is functionally comparable with current GIS running on uniprocessors. We envision that Personal HPC-G provides an opportunity to evolve traditional GIS to parallel GIS gradually. Community research and development efforts are needed to speed up the evolution. We first propose to learn from existing parallel geospatial data processing algorithms and adapt them to CMP CPU and GPU architectures. Second, we suggest study existing GIS modules (e.g., ArcGIS geoprocessing tools) carefully, identify most frequently used ones and develop parallel code for multicore CPUs and many-core GPUs, ideally in an interoperable way. Open source GIS software [22] can play an important role in the process. Third, while exiting database research on CMP CPU [45][46] and GPU [47][48] architectures are still relatively limited, they can be the starting point to investigate how geospatial data management (e.g., indexing and query processing) can be realized on the new architectures and their hybridization. Finally, we note that the Computer Graphics community has developed considerable spatial data structures (e.g., kd-tree and octree) and algorithms (e.g., iso-surface generation and ray-tracing) based on GPGPU computing technologies and they can be incorporated into parallel GIS. Similarly, progresses in image/signal processing and computer vision communities can also be incorporated. We note that, as CMP CPU and GPGPU based computing are fairly new, many source codes associated with research publications are available from the research communities, e.g., hundreds of modules published in Nvidia

CUDA website [49]. They can be valuable for parallel GIS development. We also note that a few companies, such as Manifold [50] and Azavea [51], are exploring GPGPU technologies for their products. Although they tend to target at the parallelizing local and some focal operations that are relatively easy to parallelize as discussed before, their experiences in providing commercial products and services can be valuable in eventually building a highly functional parallel GIS.

4 A CASE STUDY: GEOGRAPHICALLY WEIGHTED REGRESSION

To demonstrate how Personal HPC-G can be utilized to realize parallel GIS modules, we have chosen to develop a popular GIS module called Geographically Weighted Regression (GWR) on Nvidia CUDA enabled GPU device and use it as a case study. We note that the techniques to be introduced in this case study, such as group-based data decomposition and partial computation, can be applied to many other geospatial data processing tasks, including spatial window queries and a variety types of local spatial statistics.

GWR extends the traditional regression framework by allowing local parameters to be estimated [52]. Given a neighborhood definition (or Bandwidth) of a data item, a traditional regression can be applied to data items that fall into the neighborhood or region. In the simplest form, when only one independent variable is involved, each data item will obtain a correlation coefficient between the dependent and independent variables. The correlation coefficients for all the geo-referenced data items (raster cells or points) form a scalar field that can be visualized and interactively explored [53][54][55]. By interactively changing some GWR parameters (e.g., bandwidth) and visual exploring the changes of the corresponding scalar fields, users can have better understanding of the distributions of GWR statistics and the original dataset. Unfortunately, GWR is usually very computationally intensive and is not suitable for interactive visual explorations of large datasets on uniprocessors.

The inherent parallelism of GWR has motivated research in parallel implementations. The work reported in [9] provides an implementation in a shared-nothing grid/cluster computing environment by splitting the computing task to multiple processors with each processor handles a single data item at a time. One disadvantage of the approach is that each processor requires a copy of the whole dataset to compute the distances between the focal data item assigned to a processor and all the other data items. While the approach can tolerate heterogeneity of the computing processor configurations, it does not utilize search window effectively in limiting unnecessary computation and achieving high throughputs. We consider GWR suitable for massively parallel GPU computing due to GPU floating point computing power and the high memory bandwidth on GPU devices compared with network bandwidths among cluster computing nodes. Furthermore, Nvidia GPU devices have per-SM fast memory that can be shard among all the threads in a thread block. The shared per-SM fast memory and global memory can significant improve throughputs by hiding memory access latency while keeping floating point processing units fully utilized. While no direct comparison results will be reported in this study due to the early stage of the work and the focus of the paper, we next explain how skewed /clustered geospatial data can be decomposed to achieve workload balancing and compute GWR statistics efficiently on GPUs. The approach has the following steps: (1) Indexing geographical data using a proper quadtree data structure. (2) For each data item, compute the data items that fall within a search window definition (or bandwidth). (3) For a selected independent and dependent variable pairs, compute the desired statistic indicator (z) for each data item at (x, y). We have designed and implemented a GPGUP-based quadtree construction algorithm [56] and we will be focusing on Step 2 and Step 3 next.

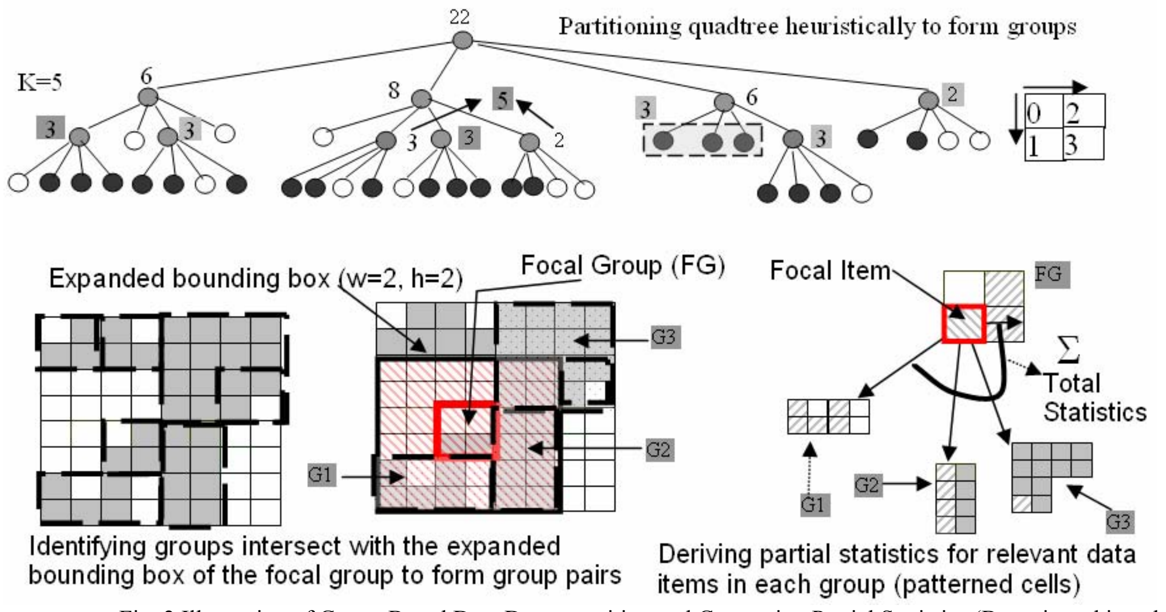
22
Partitioning quadtree heuristically to form groups
K=5
6
8
6
2
3
3
3
3
5
2
3
3
0
2
1
3
Expanded bounding box (w=2, h=2)
Focal Group (FG)
Focal Item
FG
G3
G2
G1
Σ
Total Statistics
G1
G2
G3
Identifying groups intersect with the expanded bounding box of the focal group to form group pairs
Deriving partial statistics for relevant data items in each group (patterned cells)

Fig. 2 Illustration of Group-Based Data Decomposition and Computing Partial Statistics (Best viewed in color)

As shown in Fig. 2 (top), the number of descendents of all the quadtree nodes can be computed by post-order traversal of the tree. Assuming that each processing unit can accommodate K data items, by recursively pre-order traversal of the tree, the quadtree nodes that satisfy the following two conditions can be identified: (1) the number of descendents of its parent node is greater than K (2) the number of descendents of at least one of its child nodes is less than K. For each of the identified node during the traversal, determine an assignment scheme to divide its child nodes whose numbers of descendents are less than K into groups where each group has less than K descendents. The proposed tree partition heuristic has a low computing overhead: only one full traversal to compute number of decedents and a partial traversal to locate suitable nodes for decomposition are required. For each located node, there are only 15 possible combinations for four quadrants. Some may not be desirable and can be discarded. The low computing overhead makes it suitable for online data decomposition.

Assuming the search window in GWR has a size of (w,h), then each data item at location (x,y) needs to find data items that fall within a window of (x-w,y-h,x+w,x+h) to compute a GWR coefficient. An extended tile-based approach using the data decomposition scheme discussed above is proposed to fully utilize GPU computing power. The solution is to use the data item groups identified above as the following (bottom-right part of Fig. 2 – best viewed in color). For each group, its bounding box (x1, y1, x2, y2) can be easily computed. The groups whose bounding boxes intersect with rectangle (x1-w, y1-h, x2+w, y2+h) can then be retrieved by querying the quadtree. A pair list to record the intersection spatial relationship among the groups will be subsequently built. In the example shown in Fig. 2, the 22 data items (cells) are decomposed into 7 groups. Among them, 6 are sub-trees and one is a combination of two sub-trees. The focal group (FG) (highlighted by a red square in the bottom-middle of Fig. 2) has three data items. Its bounding box intersects with three groups, G1, G2 and G3, as shown in the bottom-right part of Fig. 2. By the definition of the bounding box, we know that any data items that are within the w*h neighborhood of an item in the focal group must belong to one or more of the intersected groups, in addition to the focal group itself. Four group pairs for the example, i.e., (FG, G1), (FG, G2), (FG, G3), can be derived. During GPU parallel execution, the four pairs can be loaded to different computing blocks and each computing block will calculate partial statistics for all the data items in the focal group. The partial statistics can then be added to the total statistics of respective data items as detailed below.

Assuming the focal item (in the focal group) is D_k and the total statistics of D_k can be computed from its neighboring data items D_1^k, D_2^k, …D_n^k as $S(D_k)=f(D_1^k, D_2^k, \ldots D_n^k)$. To compute correlation coefficient for D_k, pairs of dependent and independent variables for D_1^k, D_2^k, …D_n^k are required. Assuming they are (x_1,y_1), (x_2,y_2)…(x_n,y_n), respectively, f can be defined as the following based on [57]. Note that we have removed superscript k for notation convenience as all the discussions below are limited to the focal data item D_k.

$$f = r_{xy} = \frac{\sum_{i=1}^{n}(x-\bar{x})(y-\bar{y})}{(n-1)s_x s_y} = \frac{\sum_{i=1}^{n} xy - n\bar{x}\bar{y}}{(n-1)s_x s_y}$$

$$= \frac{n\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{\sqrt{n\sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2}\sqrt{n\sum_{i=1}^{n} y_i^2 - (\sum_{i=1}^{n} y_i)^2}}$$

Let $S_1=n\Sigma x_i y_i$, $S_2=\Sigma x_i$, $S_3= \Sigma y_i$, $S_4=n\Sigma x_i^2$, $S_5=n\Sigma y_i^2$, f can be computed from n and S_1 through S_5. Assuming that data items D_1, D_2, …D_n are divided into m groups and each group has computed their partial statistics s_1, s_2, s_3, s_4, s_5, then f can easily be computed from n_j, S_{1j}, S_{2j}, S_{3j}, S_{4j} and S_{5j} as the follows (j=1,m): $n= \Sigma n_j$, $S_1=n\Sigma (S_{1j}/n_j)$, $S_2=\Sigma S_{2j}$, $S_3=\Sigma S_{3j}$, $S_4=n\Sigma (S_{4j}/n_j)$, $S_5=n\Sigma (S_{5j}/n_j)$.

In this particular example, there are three data items in the focal group. While changing focal items in the focal group will certainly change the data items that fall within the respective focal item's bandwidth, all these data items are completely included in the three groups as well as the focal group itself. As such, after loading (FG, G1), (FG, G2), (FG, G3) pairs into shared memory from the GPU device global memory, the correlation coefficients for the three data items in the focal group can be computed simultaneously. In general, when N data items are grouped into O(N/K) groups, assuming each group is paired with m_i groups, then the total number of pairs is $M=\Sigma m_i$. Depending on the data distributions and search window sizes, M can vary from a constant to $O(N^2)$. Our group based data decomposition approach amortizes M to B computing blocks to achieve workload balance and have a parallel computing time of O(M/B). Note that computation within a block is in the order of K*K which is a constant. The effective parallel throughput does not depend on N directly. On the other hand, if we assign each data item to each GPU core in a way similar to the grid/cluster computing approach proposed in [9], the parallel computing time is in the order of $\Sigma_j(\mathrm{Max}(T_{ij}))$ where j vary from 1 to N/B and T_{ij} is the computing time for data item i in round j. The problem is that even if there are only a few data items in a dense area and have large numbers of data items fall in their search windows, the parallel computing time become worse, possibly in the order of $O(N^2)$ when each N/B round has at least one bad case.

5 SUMMARY AND CONCLUSIONS

In this study, we aimed at introducing a new HPC framework for processing geospatial data in a personal computing environment, i.e., Personal HPC-G. Towards this end, we first discussed different aspects of HPC that are related to geospatial data processing, including data decomposition for workload balancing, different roles of GIS and their respective HPC requirements, recent progresses of processing large-scale data in the context of database research, and, the characteristics of different HPC frameworks and their implications to large-scale data processing. We argue that the fast increasing hardware capacities of modern personal computers equipped with chip multiprocessor CPUs and massively parallel GPU devices have low initial and operational costs, good support for data management and excellent support for both numeric modeling and interactive visualization. These desired features make Personal HPC-G an attractive alternative to traditional Cluster computing and newly emerging Cloud computing for geospatial data processing. We used a parallel design of GWR on Nvidia CUDA enabled GPU device as an example to discuss how Personal HPC-G can be utilized to realize parallel GIS modules by synergistic software and hardware co-programming.

REFERENCES

1. S. W. Wang and Y.Liu. TeraGrid GIScience Gateway: Bridging cyberinfrastructure and GIScience. IJGIS 23(5) 631-656.
2. C. W. Yang, R. Raskin, and M. A. Goodchild. Geospatial cyberinfrastructure: Past, present and future. Computers, Environment and Urban Systems, 34(4):264–277, 2010.
3. TOP500 Supercomputing Sites. http://www.top500.org/
4. M. Armbrust, A. Fox, and R. A. Griffith. A view of cloud computing. CACM, 53(4):50–58, 2010.
5. A. Clematis, M. Mineter, and R. Marciano. High performance computing with geographical data. Parallel Computing, 29(10):1275–1279, 2003.
6. R. Healey, S. Dowers et al. Parallel Processing Algorithms for GIS. CRC, 1997.

7. R. G. Healey. Special issue on parallel processing in GIS. IJGIS, 10(6):667–668, 1996.
8. S. W. Wang, M. K. Cowles and M. P. Armstrong. Grid computing of spatial statistics: using the TeraGrid for g(i)*(d) analysis. CC&PE, 20(14):1697–1720, 2008.
9. R. Harris, A. Singleton, et al. Grid-enabling geographically weighted regression: A case study of participation in higher education in England. Transactions in GIS, 14(1):43–61, 2010.
10. A. Plaza, D. Valencia et al. Commodity cluster-based parallel processing of hyperspectral imagery. Journal of Parallel and Distributed Computing, 66(3):345–358, 2006.
11. S. H. Han, J. Heo et al. Parallel processing method for airborne laser scanning data using a pc cluster and a virtual grid. Sensors, 9(4):2555–2573, 2009.
12. J. Y. Gong and J. Xie. Extraction of drainage networks from large terrain datasets using high throughput computing. Computers Geosciences, 35(2):337–346, 2009.
13. Q. F. Guan and K. C. Clarke. A general-purpose parallel raster processing programming library test application using a geographic cellular automata model. IJGIS, 24(5):695–722, 2010.
14. X. Li, X. H. Zhang et al. Parallel cellular automata for large-scale urban simulation using load-balancing techniques. IJGIS, 24(6):803–820, 2010.
15. J. B. Xie, C. W. Yang et al. High-performance computing for the simulation of dust storms. Computers Environment and Urban Systems, 34(4):278–290, 2010.
16. D. B. Kirk and W. Hwu. Programming Massively Parallel Processors: A Hands-on Approach. Morgan Kaufmann, 2010.
17. Wikipedia. Nvidia GeForce 400 series specification. http://en.wikipedia.org/wiki/GeForce_400_Series.
18. J. D. Owens, M. Houston et al. GPU computing. Proceedings of the IEEE, 96(5):879–899, 2008.
19. J. D. Owens, D. Luebke et al. A survey of general-purpose computation on graphics hardware. Computer Graphics Forum, 26(1):80–113, 2007.
20. Condor project. http://www.cs.wisc.edu/condor/
21. M. J. Fischer, X. Su, and Y. Yin. Assigning tasks for efficiency in Hadoop. ACM SPAA ’10. 30–39, 2010.
22. B. Hall and M. G. Leahy. Open Source Approaches in Spatial Data Handling. Springer, 2008.
23. J. L. Hennessy and D. A. Patterson. Computer Architecture: A Quantitative Approach, 4th Edition. Morgan Kaufmann, 2006.
24. Wikipedia, Embarrassingly parallel. http://en.wikipedia.org/wiki/Embarrassingly_parallel
25. R. Kothuri, A. Godfrind, and E. Beinat. Pro Oracle Spatial. Apress, 2004.
26. A.Pavlo, E. Paulson et al. A comparison of approaches to large-scale data analysis. SIGMOD ’09, 165–178, 2009.
27. I.Kamel and C. Faloutsos. Parallel R-trees. SIGMOD’92, 195–204, 1992.
28. M. H. Ali, A. A. Saad, and M. A. Ismail. The PN-tree: A parallel and distributed multidimensional index. Distributed and Parallel Databases, 17(2):111–133, 2005.
29. X. Zhou, D. J. Abel, and D. Truffet. Data partitioning for parallel spatial join processing. GeoInformatica, 2(2):175–204, 1998.
30. J. M. Patel and D. J. DeWitt. Clone join and shadow join: two parallel spatial join algorithms. In ACM-GIS ’00, 54–61, 2000.
31. E. G. Hoel and H. Samet. Data-parallel polygonization. Parallel Computing, 29(10):1381–1401, 2003.
32. M. J. Mineter. A software framework to create vector-topology in parallel GIS operations. IJGIS, 17(3):203–222, 2003.
33. X. W. Xu, J. Jager, and H. P. Kriegel. A fast parallel clustering algorithm for large spatial databases. Data Mining and Knowledge Discovery, 3(3):263–290, 1999.
34. J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. OSDI’04: 10–10, 2004.
35. A.Cary, Z. Sun et al. Experiences on processing spatial data with MapReduce. SSDBM’09, 302–319, 2009.
36. Apache Hadoop http://hadoop.apache.org/.
37. Y. Kwon, D. Nunley et al. Scalable Clustering Algorithm for N-Body Simulations in a Shared-Nothing Cluster. SSDBM’10: 132-150, 2010.
38. Y. Kwon, M. Balazinska et al. Skew-resistant parallel processing of feature-extracting scientific user-defined functions.SoCC’10, 75–86, 2010.
39. M. Stonebraker, D. Abadi et al. MapReduce and parallel DBMSs: friends or foes? CACM, 53(1):64–71, 2010.
40. A.Abouzied, K. Bajda et al. HadoopDB in action: building real world applications. SIGMOD ’10, 1111–1114, 2010.
41. Y. Xu, P. Kostamaa, and L. Gao. Integrating Hadoop and parallel DBMS. SIGMOD ’10: 969–974, 2010.
42. J. Duato, A.J. Pena et al. Modeling the CUDA Remoting Virtualization Behaviour in High Performance Networks. Workshop on Language, Compiler, and Architecture Support for GPGPU, Bangalore, Jan. 2010.
43. L. Chen, O. Villa et al. Dynamic load balancing on single- and multi-GPU systems. IEEE IPDPS’10, 2010.
44. B.Dally. The future of GPU computing. http://www.nvidia.com/content/GTC/documents/SC09_Dally.pdf.
45. A.Ailamaki, D. J. DeWitt et al, 1999. DBMSs on a Modern Processor: Where Does Time Go? VLDB Conference’99, 266--277
46. S. Manegold, M. L. Kersten, and P. Boncz. Database architecture evolution: mammals flourished long before dinosaurs became extinct. Proc. VLDB Endow 2(2):1648–1653, 2009.
47. B. S. He, M. Lu et al. Relational query coprocessing on graphics processors. ACM TODS 34(4), 2009.
48. P. Bakkum and K. Skadron. Accelerating sql database operations on a GPU with CUDA. GPGPU 2010, 94-103.
49. CUDA Community Showcase http://www.nvidia.com/object/cuda_apps_flash_new.html
50. Manifold GIS. http://www.manifold.net
51. Azavea Labs. http://www.azavea.com/blogs/labs/
52. A.S. Fotheringham, M. E. Charlton, and C. Brunsdon. Geographically weighted regression: a natural evolution of the expansion method for spatial data analysis. Environment and Planning A, 30(11):1905–1927, 1998.
53. J. Mennis. Mapping the results of geographically weighted regression. Cartographic Journal, 43(2):171–179, 2006.
54. U. Demsar, A. S. Fotheringham, and M. Charlton. Combining geovisual analytics with spatial statistics: the example of geographically weighted regression. Cartographic Journal, 45(3):182–192, 2008.
55. U. Demsar, A. S. Fotheringham, and M. Charlton. Exploring the spatio-temporal dynamics of geographical processes with geographically weighted regression and geovisual analytics. Information Visualization, 7(3-4):181–197, 2008.
56. J. Zhang, S. You and G. Gruenwald. Indexing Large-Scale Raster Geospatial Data Using Massively Parallel GPGPU Computing. To appear in ACMGIS’10.
57. Wikipedia, Correlation and dependence. http://en.wikipedia.org/wiki/Correlation_and_dependence