Geocomputation over the Emerging Heterogeneous Computing Infrastructure

Xuan Shi
Department of Geosciences | University of Arkansas | Fayetteville, AR 72701

Abstract

Spatiotemporal thinking and analysis has been a common interest in a growing research community aiming at understanding the spatial patterns and dynamics in the changing geographical phenomena. While there is an increasing awareness of the importance in the empirical analysis over varied space-time dimensions, the rich details of space-time complexity remain largely unexplored because of the constraint in geocomputation capacity in response to the challenge of data intensive computing.

While heterogeneous computer architecture is the emerging and future trend as the computing infrastructure and environment, new hardware systems leads to major changes and challenges in algorithm re-design and software re-engineering in order to accelerate scientific computation. In the geographic information science (GIScience) community, a generic research question has arisen: *How can existing geospatial data and computation be adapted into the architecture of Cyberinfrastructure (CI)*?

This paper reviews the evolutional trend in hardware advancement and discusses how to utilize heterogeneous computer architecture and system, which contains multiple modern Graphics Processing Units (GPUs) and Central Processing Units (CPUs), for geocomputation. Based on some exploratory research initiatives sponsored by the National Science Foundation (NSF), it can be concluded that renovating both of geospatial data structure and geocomputation algorithms will have equal significance in the transformative process in response to the above generic research question and challenge.

Modern Cyberinfrastructure is a multiprocessing environment for parallel computing that leads to significant acceleration in scientific computation high performance. Traditionally geospatial data structures, formats and algorithms have been designed for serial programs implemented on desktop computers. Consequently, existing geospatial data and computational solutions may not be appropriate in the parallel computing environment. Many conventional approaches may have been weak in practice partly due to the lack of scalability in handling large scale data defined in the common geospatial data formats. When much attention and effort were placed on algorithm optimization, the bottleneck originated in the data structure might be ignored.

In parallel computing environment, obviously any hidden or inexplicit information could be a barrier that prevents the efficient data and task partition. In practice, shapefile is a prevalent vector data format but not designed for parallel computing. Looping through a large number of features to retrieve the required but hidden information in a sequential process is not a good solution, especially for parallel computing. As a result, a few prior works exploited a blind process that manipulated the data without knowing the hidden information. In this case, the tiled approach is a typical solution. Without knowing the details about the input datasets, all data are split based on a uniform grid. Data within one grid cell can be processed by a computing node.

Since spatial features are distributed unevenly in the space, the QuadTree approach was adopted to improve the load balance issues that arose in the uniform grid approach.

In general, the tiled approach may significantly increase the overhead and difficulty, for example, in polygon overlay computation. For any one of the overlay computations using topological operators of intersect, difference, union, and XOR, the tiled approach will actually result in three operations. Splitting the features based on the grid cell boundary is the first step. Implementing the overlay operation in each grid cell is the second. Merging the features that are separated in different grid cells to generate the output product is the last step. The more levels of grid that are generated, the more overheads are expected. Without an appropriate data structure and format, optimizing algorithm for overlay computation may not lead to transformative solution to scale up the geocomputation to achieve high performance. Such a result may explain why polygon overlay computation was regarded as a killer application in parallel computing.

In geocomputation, raster data consists of rows and columns of cells and has been commonly utilized in spatial modeling and simulation. Although the divide and conquer strategy may work in the parallel computing environment by breaking down a matrix grid into multiple segments, significant re-design of both the data structure and algorithm may be required to enable parallelism and scalability. For example, as a common scenario, a spatial simulation is implemented by calculating the multi-dimensional matrix and assigning the accumulated results to another matrix with a different dimension. In serial programs, the value in each cell of the result matrix can be updated with the new value after each calculation. In parallel computing environment, however, at a given time-stamp, one cell in the result matrix cannot be assigned by multiple values. For this reason, such a computation may have to be re-designed by two separate operations, a combination of a parallel process and a serial process, or two parallel processes, in which a new data structure has to be designed to store the intermediate computing results.

Algorithm re-design and software re-engineering may be a significant challenge in order to efficiently deploy the high performance computing power over the heterogeneous computer architecture, in which GPUs are the accelerators. For example, Compute Unified Device Architecture (CUDA) is NVIDIA's parallel computing architecture for GPGPU application development. In this case, the CPU is referred to as a host, while an individual GPU is referred to as a device. The host program may have more than one sequential procedures running on the CPU or host. The kernel is the function that runs on the device and is executed by an array of threads, while all threads on the device or GPU can run the same code concurrently. CUDA has specific syntax to specify whether a function is executed on the host or on the device and whether it is callable from the host or from the device. A __host__ function can be executed on the host, and is callable only from the host. A __device__ function can be executed on the device, and is callable only from the device. A __global__ function can be executed on the device, but is callable only from the host. Although CUDA is an extension to the C programming language, significant re-engineering is thus inevitable along with algorithm re-design when transforming the serial programs into parallel programs with CUDA specific syntax, which may result in 10 times of workload in code writing. When Message Passing Interface (MPI) is used to control and coordinate the works done over multiple GPUs, another level of complexity and workload can be expected, especially when data communication is heavily involved. Several exploratory works will be introduced in this paper.