[CP-01-006] Graphics Processing Units (GPUs)

Abstract

Graphics Processing Units (GPUs) represent a state-of-the-art acceleration technology for general-purpose computation. GPUs are based on many-core architecture that can deliver computing performance much higher than desktop computers based on Central Processing Units (CPUs). A typical GPU device may have hundreds or thousands of processing cores that work together for massively parallel computing. Basic hardware architecture and software standards that support the use of GPUs for general-purpose computation are illustrated by focusing on Nvidia GPUs and its software framework: CUDA. Many-core GPUs can be leveraged for the acceleration of spatial problem-solving.

Keywords: co-processor, general-purpose computatoin, many-core parallelism, massively parallel computing

Author & citation

Tang, W. (2017). Graphics Processing Units. The Geographic Information Science & Technology Body of Knowledge (2nd Quarter 2017 Edition), John P. Wilson (ed.). DOI: 10.22224/gistbok/2017.2.8

Explanation

1. Definitions
2. Graphics Processing Units
3. Compute Unified Development Architecture (CUDA)
4. GPUs and Geospatial Activities

1. **Definitions**

Graphics Processing Units: many-core computing hardware that functions as co-processor for the acceleration of general-purpose computation

CUDA: Compute Unified Development Architecture, which is a software framework developed by NVIDIA for the thread-based parallel programming of GPUs for general-purpose acceleration

massively parallel computing: parallel computing for which the number of processors or cores is large

2. **Graphics Processing Units**

Graphics Processing Units (GPUs) are many-core computing hardware that can be used to accelerate general-purpose computation, for example, GIS algorithms or spatial analysis and modeling. GPUs are often known as co-processors that work with Central Processing

Units (CPUs) for the so-called heterogeneous computing. The original purpose of GPUs is to speed up graphics operations of computers (i.e., functioning as video cards). The massively parallel computing power of GPUs that its many-core architecture and associated massively parallel processing capability bring has been extensively recognized and applied for general-purpose acceleration in alternative scientific domains. This is ascribed to one decade ago (in early 2007) when NVIDIA released its framework and standards—CUDA (Compute Unified Development Architecture)—for directly interfacing with GPUs for general-purpose computing instead of only supporting graphics acceleration. Nowadays, GPUs have been amply equipped with alternative computing environments (e.g., mobile devices, laptops, desktops, and high-performance computing clusters). A variety of GPU-enabled solutions have been developed to accelerate domain-specific applications, and such acceleration can be several orders of magnitude in terms of its computing performance.

GPUs are produced by different manufacturers, for example, Intel, AMD, and Nvidia. Among these, NVIDIA GPUs are the most popular in terms of support for general-purpose computation. NVIDIA GPUs are based on many-core architecture that has experienced several generations of architecture development (e.g., Tesla, Fermi, Kepler, Maxwell, and Pascal). A typical GPU device comprises a series of streaming multi-processors (SMs), each of which is configured with a set of streaming processors (SPs). For example, the latest NVIDIA Tesla P100 GPU based on the NVIDIA Pascal GP100 architecture has 60 streaming multiprocessors each having 64 streaming processors—in total, 3,840 cores (single precision; see NVIDIA (2016)) are available.

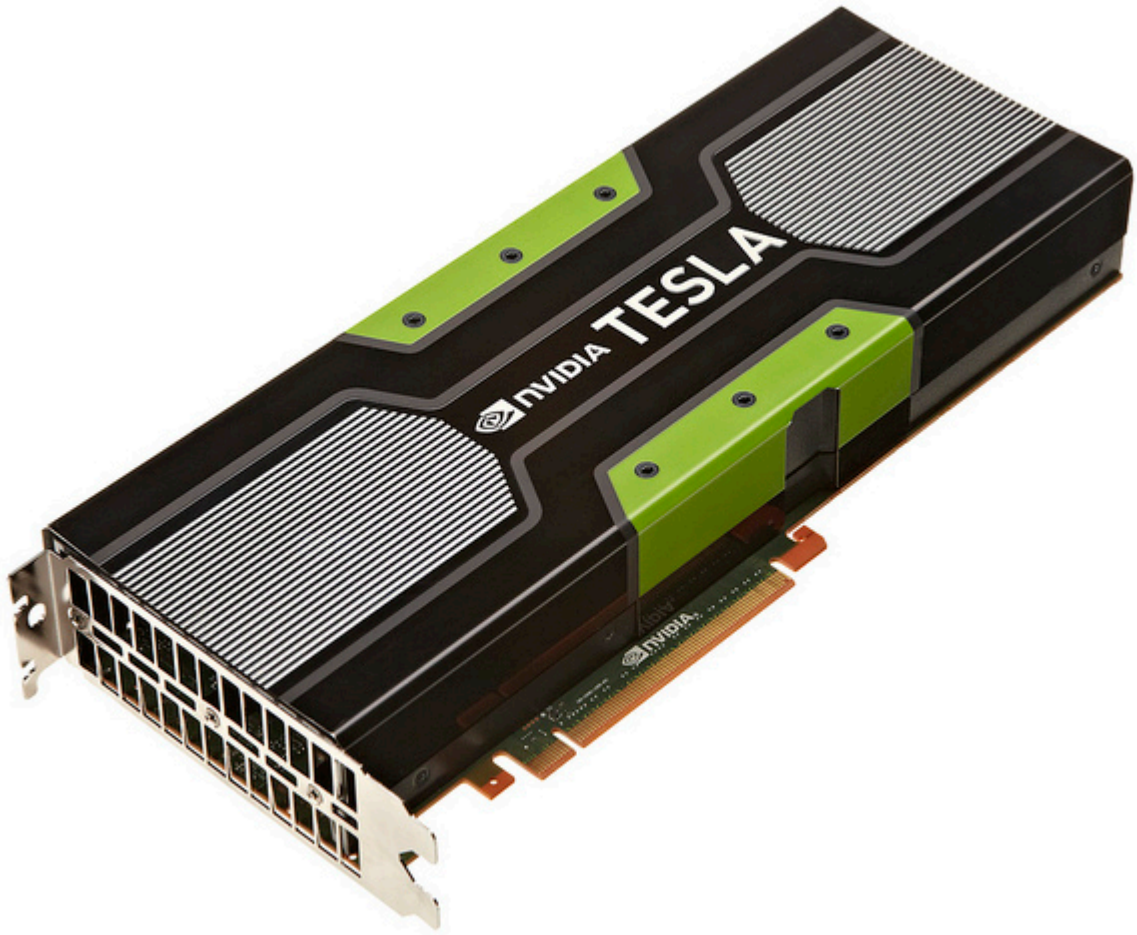


Figure 1. Hardware of a GPU device (GPU type: NVIDIA K20 GPU). Image from Flickr.

GPUs (often known as devices) function as coprocessors that are typically attached to the CPU host via PCI Express (PCIe) connection or other high-bandwidth interconnect such as NVLink (supported by NVIDIA, 5-12 times faster than PCIe-based connection; see NVIDIA 2016). Each CPU host can be connected with multiple GPUs. Further, these CPU hosts can form into a computing cluster—i.e., multi-GPU cluster. The many-core architecture within a GPU and multi-GPU cluster configuration makes GPUs well suited for data-intensive computation—that is, data-intensive tasks can be offloaded (deployed) to, and accelerated by, GPU devices.

Computer programs compiled and executed within CPU environments cannot be directly run on GPU devices. Therefore, original sequential algorithms (programs) have to be re-implemented for GPU-enabled acceleration. There are a suite of programming platforms and frameworks for GPU-based computation, including OpenACC, OpenCL, and CUDA. OpenACC is similar to OpenMP and is based on the use of computing directives to simplify the GPU programming, but at the expense of relatively low acceleration performance. OpenCL is an open framework that supports multiple types of GPUs, not just NVIDIA's. CUDA, specifically designed for NVIDIA's GPUs, is highly popular and technically mature, thus leading the use of GPU programming for general-purpose computation.

3. **Compute Unified Development Architecture (CUDA)**

CUDA (latest version: 8) provides interfaces for utilizing the many-core computing power within GPUs. While originally CUDA supports C and Fortran programming languages, other language bindings (e.g., Java, Python) are now available. CUDA relies on thread-based mechanism and shared-memory parallelism for massively parallel computing. Threads are the building blocks of computing in CUDA. These threads are light-weight and can be grouped into thread blocks in multiple dimensions (1D, 2D or 3D). A collection of thread blocks (also allowed in one-three dimensions) constitutes a grid that corresponds to a kernel function. All threads within a kernel function run the same program but on different data—the so-called Single Program Multiple Data (SPMD) manner. While a kernel is typically invoked by CPU host, recent generations of GPUs (e.g., Kepler, Maxwell, or Pascal) do allow a grid (parent) to invoke a set of new grids (children)—the so-called dynamic parallelism. This thread-block-grid hierarchy creates flexibility for decomposing a computing task into sub-tasks with alternative granularities. While each thread is light-weight in terms of computing, a large number of threads (e.g., millions, billions or even more instead of several for CPUs) are in favor of using CUDA to leverage many-core computing power in GPUs. In terms of execution on hardware, a thread grid is run in SMs on a block basis (blocks can be assigned to SMs and executed in parallel); threads within a block are grouped into warps (32 threads), which are scheduled to and executed by streaming processors.

Corresponding to thread-based hierarchy, there are three major types of memory supported by CUDA: register, shared, and global. Register memory is only dedicated to an individual thread (thus small but low access latency—i.e., fast). Shared memory enables communication for threads within a thread block. Further, if all threads within a thread grid are in need of communication, they have to rely on global memory (with relatively high latency). The use of global memory should be minimized as much as possible and a tiling strategy that partitions data in global memory to fit into shared memory was suggested (see Kirk and Hwu 2010). Global memory in GPUs can communicate with CPU memory. Early versions of CUDA only allow for allocating memory for data in CPU host and then copying the data to the global memory. However, the recent generations of CUDA (6 or higher versions) support unified memory that simplifies and automates the creation and replication of data between host memory and global memory (suitable for Kepler, Maxwell, and Pascal architectures). Nevertheless, data transfer between host and device represents a barrier for GPU-based general-purpose computation (the latest NVLink technology may be a partial solution to this barrier). In other words, users may need to minimize the transferring of data between CPU host and GPU device.

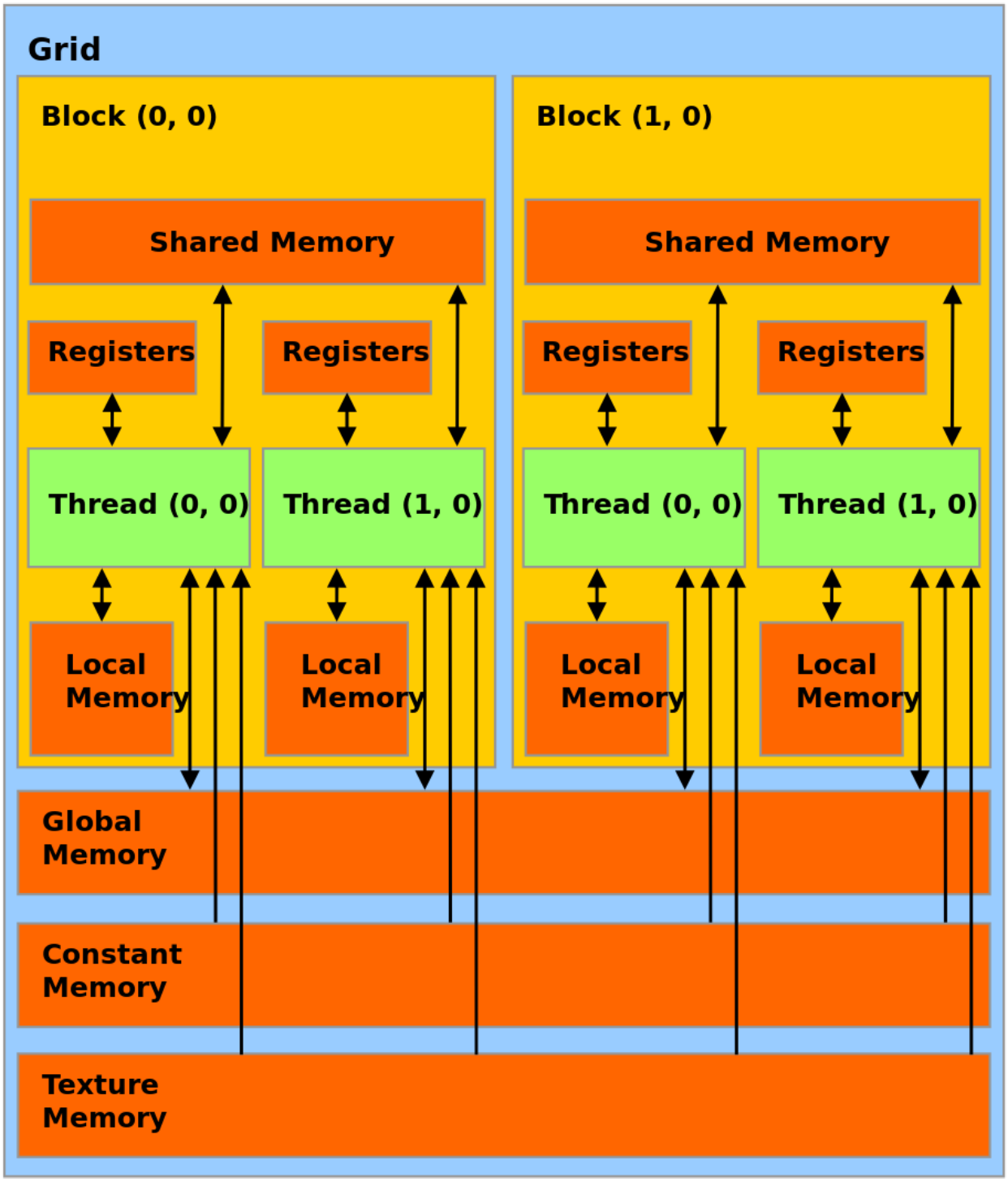


Figure 2. CUDA framework for the programming of GPUs. Image from Wikimedia.

The parallel programming of GPUs is not trivial due to the relatively complex nature of many-core architecture in modern GPUs. Kirk and Hwu (2010) suggested four parallel GPU programming steps (while generic for any parallel computing): problem decomposition, algorithm selection, implementation, and performance tuning. As Kirk and Hwu further advocated, the following computational thinking skills provide strong support for the use of GPUs for general-purpose computation acceleration: computer architecture, programming

models and compilers, algorithm techniques, and domain knowledge. Equipped with these skills, researchers can design and develop parallel strategies that allow for transforming a domain-specific problem of interest into a set of tasks to be tackled at the thread level in GPUs. These parallel strategies include, but are not limited to, domain decomposition, task scheduling, load balancing, and synchronization (see Wilkinson and Allen 2004). In particular, as more attention is being paid on multi-GPUs, these parallel strategies become more important for efficacious heterogeneous parallel computing. Further, for spatiotemporal problem-solving using GPUs, these parallel strategies may need to be adapted to take into account spatiotemporally explicit characteristics of geographic data.

The evaluation of computing performance of GPU-enabled parallel algorithms is of necessity. Basically, this performance evaluation is similar to that of CPU-based parallel algorithms: use of metrics based on computing time of parallel and sequential algorithms. The performance metric is typically the so-called acceleration factor (same as speed up, frequently used for CPU-based parallelism), which is the value of computing time from sequential algorithm (based on a single CPU) divided by that from GPU-enabled parallel algorithm (run on a GPU). A large value of acceleration factor means high acceleration performance from GPUs. Depending on the configuration of chosen CPUs, the acceleration factors for the parallel algorithm on the same GPU device can vary substantially. Thus, an advanced modern CPU configuration should be preferred for the performance evaluation of GPU algorithms. The performance of GPU algorithms depends on alternative facets, for example, characteristics of data (e.g., locality or spatial heterogeneity), implementation of domain algorithms per se, parallel strategies, and hardware configuration of CPUs and GPUs. Thus, performance evaluation is essential in the design and implementation of efficacious GPU-enabled parallel algorithms.

4. **GPUs and Geospatial Activities**

A suite of geospatial studies accelerated using GPUs have been reported in the literature (Shi, Kindratenko, & Yang, 2013; Tang, 2013; Tang, Feng, & Jia, 2015; Zhang, 2011; Zhao, Padmanabhan, & Wang, 2012). For example, Tang, Feng, & Jia (2015) proposed a GPU-enabled solution for spatial point pattern analysis based on Ripley's K function and the acceleration from 50 GPUs can reach about 1,500 times (127 seconds vs. 52.97 hours on a single CPU; i.e., three order of magnitude). The highly parallel structure of GPUs renders GPUs tailored to acceleration on the spatial problems with the following characteristics (Kirk and Hwu, 2010): fixed problem size with long computing time, fixed problem size in need of refined solutions, or large problem size. Spatial problems supported by alternative GIS-based algorithms (e.g., data processing and visualization) and models (e.g., statistics, simulation, and optimization) fall within these three types that can gain acceleration benefits from GPUs. Driven by the built-in data-parallel mechanism, GPUs are conducive to GIS-based processing and analysis of massive spatial data (implication of large problem size), for example, processing of remote sensing imagery at global level or terrain analysis of fine-resolution DEMs for the U.S. A large-sized spatial dataset (vector or raster) can be decomposed to a number of fine-grained sub-datasets that can be assigned to, and handled by, threads in GPUs.

Often, depending on the capacity of GPU memory, a large spatial dataset which size exceeds the GPU device memory may be further decomposed into coarse-grained sub-

datasets within CPU environments first. These coarse-grained sub-datasets can then be processed by a single GPU in a batch manner or by multiple GPUs in parallel (e.g., in combination with message-passing parallelism across GPUs). Second, GPUs are a suitable platform for accelerating spatial problems with datasets that may not be large, but generally have long computation time. Searching for optimal solutions (spatial optimization problems) or conducting Monte Carlo simulation runs (spatial simulation problems) are such examples. Third, calibration of spatial models (e.g., simulation models of land change in a specific study area) can receive benefits from GPU acceleration in terms of, for example, calibrating a model's parameters with higher model accuracy (fixed problem size for refined solutions).

Since 2006, GPUs have experienced several generations of development (including hardware and software). The number of processing cores available in a single GPU have increased from hundreds to thousands. The continually evolving many-core architecture in GPUs offers considerable computing performance. For example, the latest Tesla P100 (Pascal architecture) can provide 5.3 tera FLOPS (floating-point operations per second) of peak performance (double-precision; 10.6 TFLOPs for single-precision performance) due to its design that supports 15.3 billion transistors and 3,840 cores (see NVIDIA 2016). This massively parallel computing power makes GPUs a natural choice for big data analytics or other computationally intensive spatial analysis or modeling. Of course, to best utilize the many-core massively parallel computing power in GPUs for spatial problem-solving, researchers often need to develop skills that couples computational thinking with spatial thinking. Through this coupled thinking, spatial problems can be efficaciously transformed into computational problems for GPU-enabled acceleration.

References

Kirk, D. B., & Hwu, W. W. (2010). Programming Massively Parallel Processors: A Hands-On Approach, 1st Edition. San Francisco, CA: Morgan Kaufmann Publishers, Inc.

NVIDIA. (2016). NVIDIA Tesla P100: The Most Advanced Datacenter Accelerator Ever Built. NVIDIA Whitepaper.

Shi, X., Kindratenko, V., & Yang, C. (Eds.). (2013). Modern Accelerator Technologies for Geographic Information Science. New York: Springer.

Tang, W. (2013). Parallel construction of large circular cartograms using graphics processing units. International Journal of Geographical Information Science, 27(11), 2182-2206.

Tang, W., Feng, W., & Jia, M. (2015). Massively parallel spatial point pattern analysis: Ripley's K function accelerated using graphics processing units. International Journal of Geographical Information Science, 29(3), 412-439.

Wilkinson, B., & Allen, M. (2004). Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers, 2nd edition. New York: Pearson Publishers.

Zhang, J. (2011). Speeding up large-scale geospatial polygon rasterization on GPGPUs.

Paper presented at the Proceedings of the ACM SIGSPATIAL Second International Workshop on High Performance and Distributed Geographic Information Systems, Chicago, Illinois.

Zhao, Y., Padmanabhan, A., & Wang, S. (2012). A parallel computing approach to viewshed analysis of large terrain data using graphics processing units. International Journal of Geographical Information Science, 27(2), 363-384.