Technical Report – Project 9

GPU-Based Parallel Computing

Yakshita B Rakholiya

CS_610 Introduction to Parallel Computing

Pace University – NYC

Instructor: Dr. Lixin Tao

Spring 2023

Contents

Introduction

The method of doing several computations or processes simultaneously is known as parallel computing. Large computational problems can be divided into smaller ones that are easier to handle and then tackled concurrently. Computers that employ parallel processing are called parallel computers.

There are several different forms of parallel computing, including bit-level, instruction-level, data, and task parallelism. The interest in parallelism, which has long been employed in high-performance computing, has intensified because of the physical restrictions that prevent frequency scaling.

However, explicitly parallel algorithms, especially those that use concurrency, are more tough to write than sequential algorithms because parallel algorithms might create several new classes of possible software bugs, out of those, race conditions are the most frequent. One of the biggest challenges to getting the best performance out of a parallel program is often communication and synchronization between the many subtasks.

GPU

A GPU, a specialized electrical circuit created to easily operate and update memory, is usually used to shorten the time to generate the pictures in a frame buffer which is intended for output to a display device. GPUs mostly are more efficient than regular central processing

units (CPUs) for algorithms that handle large blocks of data in parallel because of their highly parallel design.

These computations were carried out by the central processing unit in the early stages of computer development. However, as more graphics-intensive programs were designed and developed, the requirements of these apps put additional burden on the central processing unit, resulting in a reduced performance. GPUs were developed to relieve CPUs of the stress of accomplishing these tasks and to improve 3D graphics rendering. GPUs work by employing a technique known as parallel processing, in which many processors combined do a small fraction of task parallelly for whole operation.

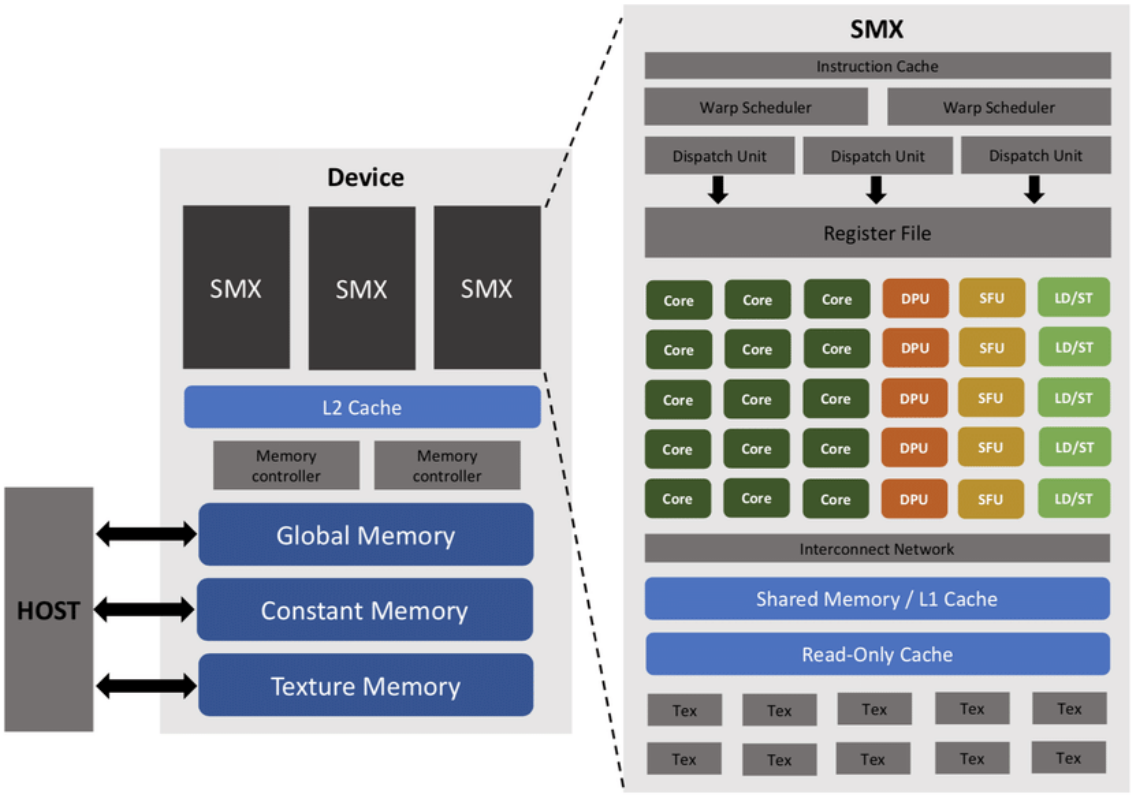
A GPU is a specialized co-processor that, like many other things, has advantages and disadvantages, excelling at some tasks while failing badly at others. It collaborates with a CPU to boost data operations and the number of concurrent calculations inside an application.

GPUs, or graphics processing units, are commonly utilized in PC gaming because they allow for smooth, high-quality graphics rendering. Furthermore, programmers began to use GPUs to accelerate workloads in fields such as artificial intelligence (Al) and Crypto mining.

Nvidia GPU

Framework for compilers that focuses on maximizing how much memory an application consumes. They used the GeForce GTX8800 and GTX280, two outdated NVIDIA GPUs, to test their framework. In conclusion, we continue the same line of inquiry but concentrate on the impact of high-level optimization discovered in the CUDA compiler on specific instructions running in the pipeline and on the access overhead of various memory present in contemporary GPUs.

The GPU architecture is based on a collection of Steaming Multiprocessors (SMX), each capable as a separate processor and can handle many parallel threads using the single instruction multiple threads (SIMT) method. Each SMX includes multiple CUDA cores which can execute a single 32-bit integer or floating-point operation per cycle and have fully prepared 32-bit ALUs and FPUs. Additionally, it has load and store units (LD/ST) for calculating source and destination memory addresses, special function units (SFU) for executing intrinsic instructions, and double-precision units (DPU) for 64-bit computations. Each SMX has computational resources plus some specific number of warp schedulers, instruction dispatch units, instruction buffers, texture units, and shared memory units.



System Architecture

GPU-based parallel computing is a type of parallel computing that takes advantage of the processing capability of a GPU to do activities that need a large amount of data and calculation. Originally, GPUs were designed to produce graphics and pictures for video games and computer simulations. As the development of General-Purpose Computing on Graphics Processing Units got better, a wide range of parallel computing applications can now be run on GPUs.

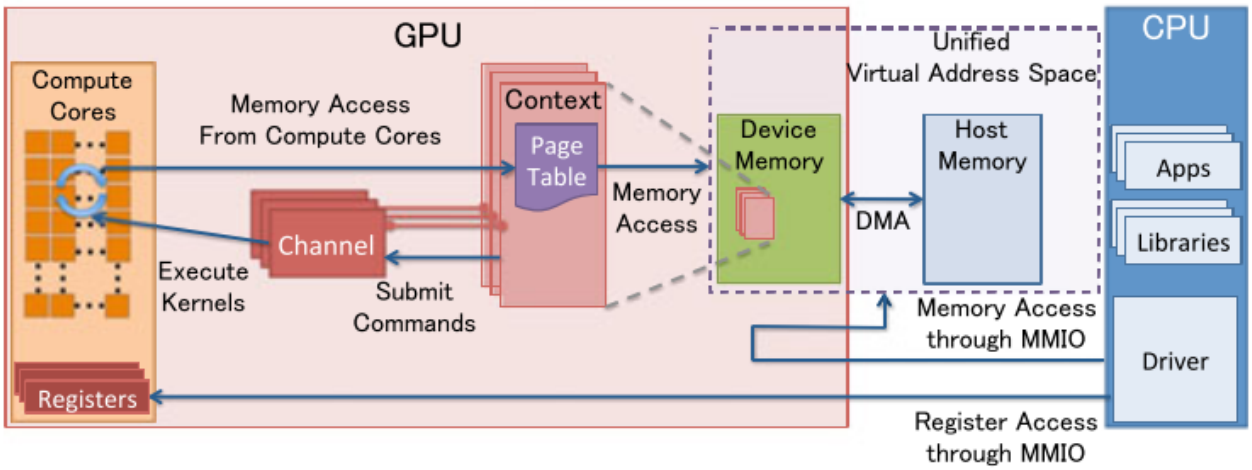


Fig. 1. GPU resource management model.

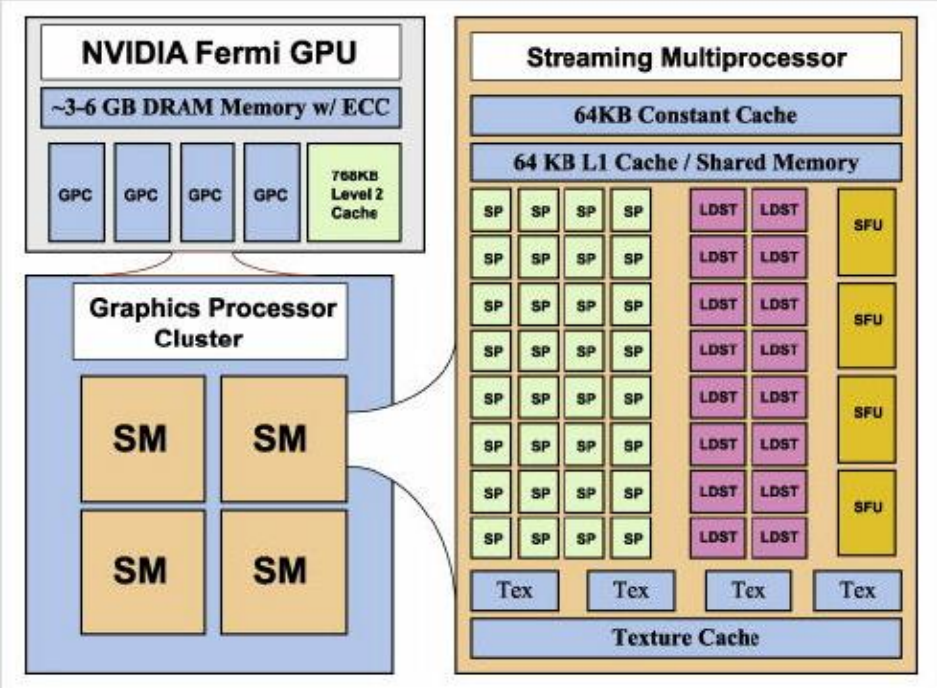
The justification of several significant designs used by modern GPUs. Through MMIO, the CPU and GPU communicate. Large-scale data transfers can be enabled by hardware engines for DMA. Her, commands need to be written using MMIO. Although they are rarely used, the I/O ports can be used to indirectly access the MMIO regions. Nouveau presently never utilizes an open-source device driver.

Any operation is driven by commands sent from the CPU, such as launching a kernel. The GPU channel is a piece of hardware that receives the command stream. One or more GPU channels may be present in each GPU context. GPU channel descriptors are found in each GPU context; each descriptor is produced as a memory object in the GPU memory. A page table is one of the options that are stored in each GPU channel descriptor.

Hardware Architecture:

A graphics processing unit, or GPU, is an integrated circuit that improves performance through data management and manipulation. Deep learning involves a huge number of mathematical calculations and other procedures, which may be considerably parallelized and hence expedited by using GPUs (Graphics Processing Units). A graphics processing unit (GPU) can contain hundreds of cores, but a central processing unit (CPU) may only have twelve. Long training times and restricted GPU memory, which has a significant impact on the amount of data that can be stored on the GPU, limit the practical usage of GPU. This chapter introduces readers to a variety of distributed massively parallel systems that reduce training time and increase memory economy.

These steps have been taken to address the previously mentioned issues. The most recent Tesla graphics processing unit (GPU) has only 1 Gigabits of RAM. Reason behind this of having smaller RAM is due to graphics processing units (GPUs), which is used for deep learning, have a lower memory capacity. As a result, GPU networks needs to design to fit within the memory that is available. It's probable that this is a barrier to progress and overcoming it would be extremely advantageous to the area of computational intelligence.

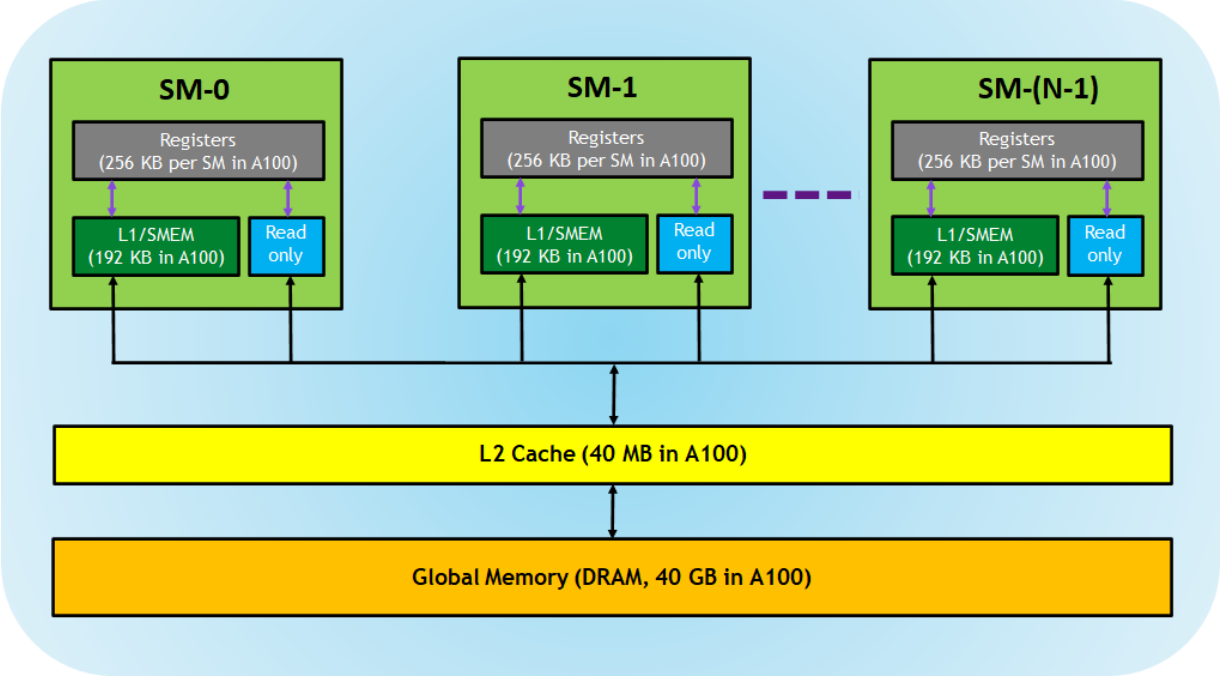


A cross-platform parallel programming model called OpenCL enables programmers to create applications for a variety of compute accelerators, such as GPUs, CPUs, and FPGAs. There are many programming languages are supported by OpenCL, which offers a consistent programming interface on many hardware platforms.

The interplay between hardware generalization and specialization is fascinating to investigate and comprehend. The graphics processing unit was originally designed for graphics processing, which is more parallelized than other types of computing. Designers later discovered that GPUs could be used for more general computing tasks, which led to the development of CUDA and OpenCL. GPUs have their own set of graphics-specific resources.

Memory hierarchy and data management:

The memory structure of the GPU architecture is complex, consisting of registers, shared memory, and global memory. Registers are the fastest memory, and it is used for storing temporary data. Shared memory type is normally shared by all threads in a block, which allows for better synchronization and communication. At last, Data that is shared by all threads is stored in global memory. In GPU-based parallel computing, robust data management is required for high-performance processing. GPUs handle data concurrently using method known as the Single Instruction Multiple Data (SIMD) technique.



Comparison with CPU architecture:

CPUs and GPUs have many similarities. Both are important computer engines which are both silicon-based microprocessors. And both handles data, but CPUs and GPUs have distinct architectures and are designed for distinct purposes. The CPU is well-suited to a wide range of tasks, particularly those requiring low latency or high per-core performance. The CPU is a strong execution engine that concentrates its lesser number of cores on tasks and completing them rapidly. This makes it ideally suited to tasks ranging from serial computing to database administration. Following diagram compares the architectures of a traditional CPU and a GPU:

The CPU is made up of billions of transistors that are linked together to form logic gates, which are subsequently linked together to form functional blocks. On a bigger level, the CPU is made up of three main components such as the Arithmetic and Logic Unit (ALU), cache, RAM, or peripherals. Cache contains interim values required for ALU computations or aids in the tracking of functions in the program being performed.

Programming Environment

Because of the expanding popularity of GPUs, high-level programming paradigms and tools are now available. GPU programming has always required the use of low-level languages and packages such as CUDA or OpenCL. An introduction to GPU

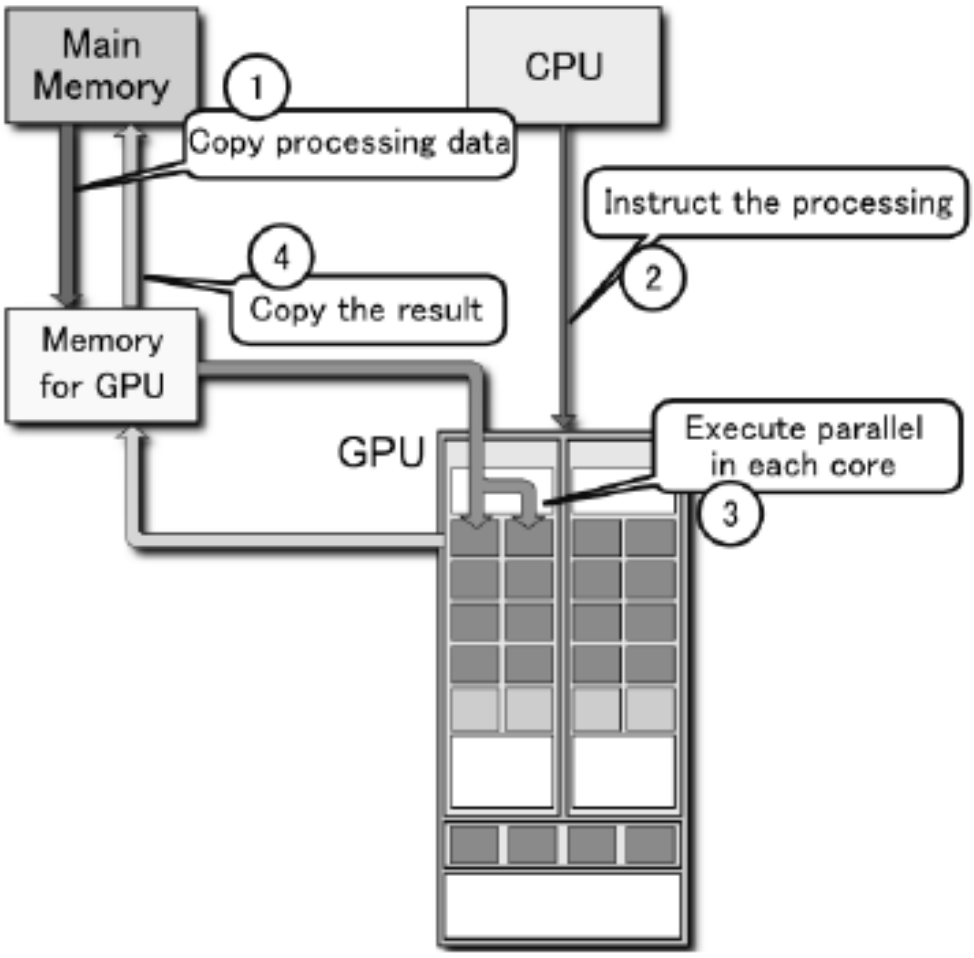
computing programming models: There is some program paradigms available for GPU computation.

CUDA model:

The CUDA programming approach is founded on the concept of a kernel, allows a function to be performed in parallel on the GPU. The global property specifies if the kernel should be run on the GPU. C or C++ language is normally use to write the kernel. The kernel is divided into threads, that can execute the same code on a different part of the data. The CUDA runtime system automatically connects the threads to the GPU's available processing units in order to perform concurrent execution.

A collection of libraries that support the CUDA programming model provide additional capabilities for GPU-based parallel programming. These include the CUDA BLAS Library, CUDA FFT Library, and CUDA Math Library, all of which provide optimized implementations of common mathematical operations. The CUDA FFT Library also includes efficient versions of the Fourier transform.

CUDA processing flow:



The CUDA programming style makes building parallel programs that operate on NVIDIA GPUs straightforward and effective. It provides a collection of tools and APIs that make it easier to write efficient and scalable parallel code, allowing programmers to take use of the GPU's high bandwidth and massive parallelism.

OpenCL model:

Although OpenCL programs can be compiled and linked into binary objects using traditional off-line compilation methods, OpenCL also supports run-time compilation, which allows OpenCL programs to run natively on target hardware, even on platforms that the original software developer did not create. Run-time compilation removes instruction set dependencies, allowing hardware makers to make large modifications to instruction sets, drivers, and supporting libraries from one hardware generation to the next. Applications that employ OpenCL's run-time compilation features will immediately take use of the newest hardware and software features of the target device, eliminating the requirement for the primary application to be recompiled.

The OpenCL programming model encapsulates CPUs, GPUs, and other accelerators as "devices" with one or more "compute units". Then this allows one or more SIMD "processing elements" to perform instructions in lockstep. OpenCL defines four types of memory systems that devices may include: global memory, small low-latency read-only constant memory, shared local memory accessible from multiple PEs within the Each PE has its own computing unit and "private" memory or device registers.

However, managing memory transfers between the host and device as well as writing effective parallel code that can make use of the available processing resources make programming for OpenCL challenging. The OpenCL programming model is still a valuable tool

for accelerating computationally demanding applications on a variety of hardware.

OpenACC model:

The NVIDIA HPC SDKTM with OpenACC provides scientists and researchers with a fast path to accelerated computing with minimal programming effort. By putting compiler "hints" or directives into your C11, C++17, or Fortran 2003 code, you may offload and run your code on the GPU and CPU using the NVIDIA OpenACC compiler.

In addition to the NVIDIA OpenACC compilers, the HPC SDK contains GPU-enabled libraries and developer tools to aid in your GPU acceleration efforts.

Applications

High-performance computing (HPC)

Cloud computing

Machine learning and deep learning

Cryptography and security

Financial risk administration

Climate Change

GPU uses

Graphics processing units were created to accelerate the process of graphic rendering. As a result, they are well suited for high-level graphics, where the rendering of lighting, textures, and forms must all take place concurrently to keep images moving quickly across the screen.

GPUs for Gaming:

Graphics processing units (GPUs) were created to accelerate the process of graphic rendering. As a result, they are well suited for high-level of graphics, where the rendering of lighting, textures, and forms must all occur concurrently to keep images moving quickly across the screen.

GPUs can handle the ever-increasing visual demands because games are getting more computationally demanding, with hyper-realistic images and massive, complex in-game worlds; and screens require higher resolution and quicker refresh rates; GPUs can meet these needs.

This means gamers now being able to play games with higher frame rates or both.

Graphics processing units have been incredibly expensive from the inception of gaming servers, but that value will only increase as virtual reality gaming gets more widespread.

GPUs for Video Editing and Content Creation:

Long rendering times have plagued video editors, graphic designers, and other creative professions for years, tying up computational resources and stifling creative flow.

Graphic designers, video editors, and other professionals involved in the creation of visual material had to wait painfully long periods of time for their videos to be rendered prior to the introduction of GPUs.

Because graphics processing units (GPUs) are capable of parallel computation, rendering an excellent-quality movie or image today takes significantly less time and resources than it did previously.

GPU for Machine Learning:

AI and machine learning are the most interesting use case for GPU technology. Because GPUs have a huge amount of computational capability, they can facilitate incredible acceleration in workloads that benefit from GPUs' highly parallel nature, such as image recognition. Today's deep learning technologies do not command on GPUs collaborating with CPUs.

GPUs are still utilized to drive and create gaming graphics, as well as to improve PC workstations; nevertheless, due to their broad variety of applications, GPUs have also become a significant component of modern supercomputing.

The high-performance computing capabilities make them excellent choice for the machine learning.

The goal of machine learning, also known as deep learning, is to educate computers to improve their knowledge and abilities on their own over time. To accomplish this goal, we must provide facts and knowledge to youngsters through observations and interactions with the actual world.

What I learned in this course

I realized that good GPU-based parallel programming is required for fast and big cloud computing and machine learning applications. By knowing the GPU hardware architecture, memory structure, and programming methods like CUDA, OpenCL, and OpenACC, programmers may design and create highly parallel programs that fully utilizes the GPU's capabilities.

Furthermore, by utilizing the tools and libraries available for GPU programming, programmers can speed up the development process. After that they achieve higher levels of optimization for their applications. Learning GPU-based parallel programming is becoming increasingly important for programmers as the need for data-intensive and computationally expensive applications in cloud computing and machine learning grows.

Conclusion:

Finally, GPU-based parallel computing has emerged as a powerful tool for large-scale, rapid calculations, mainly in fields like as machine learning, cryptography, financial modeling, and others. GPU hardware and programming frameworks like as CUDA, OpenCL, and OpenACC have enabled faster processing of complex data. However, several issues remain in the areas of memory management, scalability, portability, and performance optimization. Future GPU-based parallel computing advancements are expected to introduce advanced powerful and efficient hardware, improving programming environments and tools, and its unique algorithms and applications.

Overall, GPU-based parallel computing has accomplished record to be a useful tool for high-performance computing and will keep playing a big part in the development of science and innovation in the years to come.

Reference:

[1] Munshi Aaftab. OpenCL Specification Version 1.0. Dec, 2008

[2] Nickolls John, Buck Ian, Garland Michael, Skadron Kevin. Scalable parallel programming with CUDA. ACM Queue. 2008;6(2):40–53.

[3] Cao J, Yin B, Lu X, Kang Y, Chen X (2018) A modified artificial bee colony approach for the 0-1 knapsack problem. Appl Intell 48(6): I 582-1595

[4] Yang Z., Zhou Q., Lei L., Zheng K., Xiang W. (2016). An IoT-cloud based wearable ECG monitoring system for smart healthcare. J. Med. Syst. 40:286. 10.1007/s10916-016-0644-9

[5] Chitty D (2016) Improving the performance of GPU-based genetic programming through exploitation of on-chip memory. Soft Comput 20(2):661—680

[6] Cano A, Zafra A, Ventura S (2015) Speeding up multiple instance learning classification rules on GPUs. Knowl Inf Syst 44(1): 127- 145