KAPITEL 5 / *CHAPTER 5* [5]
GPU STREAM PROCESSING

DOI: 10.30890/2709-2313.2024-35-00-037

The development of graphics processing units (GPUs) is a great example of the rapid technological progress in the field of computer technology. Initially, GPUs were simple devices for rasterizing images, capable only of applying basic textures and performing the simplest graphic operations. As of today, they have turned into super-powerful computing systems, the performance of which exceeds the capabilities of central processors by tens of times.

To realize the uniqueness of GPUs, it is worth considering the fundamental differences between CPUs and GPUs. CPUs are built on a sequential computing model. In this model, data processing occurs sequentially, element by element, which creates a strong dependence on memory access speed. To optimize this architecture, CPU manufacturers use large amounts of cache memory and complex instruction prediction systems. However, even with these improvements, the level of parallelism in CPUs remains quite limited.

Graphics processors, on the other hand, were designed from the very beginning with a parallel data processing in mind. Their architecture is optimized for simultaneous executions of multiple operations of the same type on different data. GPUs use smaller but more efficient caches and have specialized blocks for processing graphics data. This fundamental difference of information processing determines the main advantages and features of using GPUs.

Threaded computing model. GPUs implement parallelism at various levels, allowing them to perform many operations simultaneously, increasing performance.

Modern GPUs are based on the threading model of computing or thread level [1] (Thread-Level Parallelism, TLP). In this model, data is organized as a stream is a sequence of the elements of the same type. Stream can be an array of vertex coordinates

[5]***Authors:*** *Romanyuk Oleksandr Nykyforovych, Shenshin Oleksandr Oleksandrovych, Bobko Oleksii Leonidovych, Titova Nataliia Volodymyrivna, Romanyuk Serhii Oleksandrovych*
Number of characters: 16260
Author's sheets: *0,41*

of three-dimensional objects, a set of texture coordinates, lighting parameters or material attributes. Each thread is responsible for executing one instruction on a separate data element, for example, on an image pixel. This allows thousands or even millions of pixels to be processed simultaneously. Stream processing is done by special computing modules known as cores.

Processing cores are one of the elements of a streaming architecture. They take one or more data streams as input, perform programmed operations on them, and generate output streams. An important feature is that the processing of each element of the stream occurs independently, which allows to establish parallelization of calculations with ease.

Core level [2] (Instruction-Level Parallelism, ILP). The level of instructions where the core can execute multiple instructions simultaneously, in parallel, on different data sets. Within a single core, a single instruction can process different data elements (such as pixels in an image), allowing many calculations to be performed quickly in a single cycle.

Data-Level Parallelism (DLP) [3]: At the vector level, the same instruction is executed on groups of data at the same time. For example, the GPU can calculate the color values of multiple pixels in a texture at the same time.

The streaming model has significant advantages in the context of modern trends in computer technology. First, it allows for efficient use of the growing memory bandwidth, which increases by approximately 25% annually. Second, the architecture scales well with an increase in the number of computing cores, which allows for increased system performance.

GPU threading. There are several levels of hierarchy that allow you to scale computations to solve problems of different scales.

Blocks and Grids – all threads are organized into blocks that form a grid. Each block can contain hundreds of threads, and a grid can contain thousands of blocks. This structure allows you to process huge amounts of data, for example, large 3D models.

Warps are groups of threads in a block that execute one instruction at a time. For example, on NVIDIA, a warp typically contains 32 threads. It is important to optimize

the code so that all threads in a warp run simultaneously, otherwise performance drops due to thread divergence.

Features of working with memory in GPUs. Working with memory in GPUs has its own specifics, which are manifested through two key operations: gather and scatter. The gather operation, which involves reading data at a computed address, is implemented through the texture sampling mechanism and is available for both fragment and vertex shaders. This operation is especially effective when working with random access to data.

The scatter operation, which involves writing to a computed address, has limited support on GPUs. This creates some programming difficulties and often requires algorithms to be reformulated to use gather operations instead of scatter. This limitation is a consequence of the parallel nature of GPUs and the need to avoid conflicts when writing data simultaneously.

Synchronization technologies. Synchronization of threads in GPUs is crucial to avoid conflicts when accessing shared resources and ensuring correct computations.

Synchronization barriers are used to stop thread execution until all threads in a block or warp have completed a certain stage of work. This is necessary to ensure the correct order of memory accesses.

Atomic operations allow multiple threads to simultaneously read and write data to shared memory without conflict. They are executed as non-divisible operations, which guarantees correctness even in the event of concurrent access.

To increase the speed of data access, GPUs use local memory, which is accessible only to threads of a single block. This minimizes latency and reduces the load on global memory.

Implementation of conditional structures in GPU. Unlike CPU, where conditional transitions are commonplace, in graphics processors implementation of conditional structures is done somewhat differently, in GPU they are implemented in a specific way due to the features of parallel architecture. There are several approaches to implementation of conditional structures and the simplest of them is the use of conditional codes, when according to the results of checking the condition, a special

flag is set that affects the execution of the following instructions. In this case, both branches of the conditional structure are executed, but the result is stored only for the branch whose condition is met.

A more advanced approach is SIMD [4] (Single Instruction Multiple Data), which allows the same instruction to be executed simultaneously on different data sets. This method is particularly effective when a condition is met or not met simultaneously for a large group of elements being processed in parallel.

SIMT [5] (Single Instruction, Multiple Threads) is an extension of SIMD used in GPUs. In SIMT, threads are grouped into warps (typically 32 threads) that execute one instruction per thread at a time. However, if an instruction requires a conditional branch, this can lead to thread divergence, where different threads in the same warp execute different branches of code, which slows down the computation.

The most flexible, but also the most difficult to implement, approach is MIMD [6] (Multiple Instruction Multiple Data), which allows different processing units to execute different instructions independently of each other. This approach is used in the most modern GPU architectures.

Current trends and the future of GPUs. Modern GPUs use a combination of different approaches to improve performance. For example, SIMD methods are ideal for graphics processing, while MIMD provides efficiency when performing complex artificial intelligence tasks. Such hybrid solutions allow you to adapt GPUs to a wide variety of application scenarios.

Optimizing GPU computing is very important for efficient resource utilization. For example, the Coalesced Memory Access technique [7] allows threads to access adjacent memory addresses simultaneously. This minimizes the number of the accesses to the global memory and improves performance.

Thread divergence [8] is designed to avoid slowdowns due to thread divergence in the warp, developers optimize the code so that all threads execute the same instructions. For example, they rewrite conditional constructs, dividing the computation into separate stages.

The current stage of GPU development is characterized by the emergence of

specialized computational units optimized for specific tasks. These include ray tracing cores, tensor processors for matrix operations, accelerators for artificial intelligence tasks, and specialized geometry processing units.

Significant innovations are also occurring in GPU architecture. For example, the introduction of chiplet design, which involves breaking a large chip into several smaller, independent chips called chiplets, which are then connected via high-speed channels or interfaces, ultimately helping to overcome the limitations associated with manufacturing large chips and providing flexibility during design.

The use of three-dimensional crystal layout [9] (3D Integrated Circuits, 3D ICs), which allows the creation of crystals with multiple layers (vertically connected) instead of traditional single-layer chips, which has allowed to increase the density of components per unit area, reduce the distance between components (which reduces latency and improves performance), and improve energy efficiency, as data can be transferred between layers with less energy consumption. 3D chips allow to place different parts of the system (for example, computing units, memory, input/output) on different layers to optimize the performance of tasks.

Development of new technologies for interconnects, which are used to connect different parts of a chip or between chips in a system. Interconnects use a variety of connection types, from traditional electrical copper-based ones to modern optical waveguides and hybrid solutions, which have enabled data rates ranging from several Gigabits per second to tens of Terabits per second, depending on the application. Advances in interconnect technologies, including the transition to optical channels and new materials, are enabling architectures capable of transmitting data at speeds exceeding 100 Gbits per second, providing 30% greater energy efficiency and increased scalability for complex multi-component systems capable of processing petabytes of data per second with latency of less than 10 nanoseconds. The use of low-voltage interconnects and energy-efficient data transfer protocols allows for a 15-20% reduction in power consumption compared to traditional copper connections at the same transfer rate and provides a bandwidth of up to 1 Tbit/s between chip components such as CPU, GPU and memory, as well as between different chips in the system during

processing stages such as data acquisition, training and output.

In the software domain, GPU development is accompanied by the emergence of new programming interfaces and frameworks, such as DirectX 12 Ultimate, Vulkan, CUDA, and OpenCL [10-13], which give developers more and more opportunities to effectively use GPU hardware resources. Algorithmic optimizations, including Variable Rate Shading, mesh shaders, and machine learning-based optimizations, also play an important role.

Thread usage limitations in rendering in graphics and 3D modeling can be caused by several factors that affect the efficiency and speed of processing. One such limitation is hardware limitations, particularly the number of processor cores. If the hardware does not have enough cores to run multiple threads efficiently, this can lead to overloading and reduced performance. Memory capacity is also important, as using many threads can require large amounts of memory, and if there is not enough RAM, performance can be slowed down due to frequent accesses to the hard disk. Another limitation is the bandwidth of the memory bus. High bus load can limit the efficiency of parallel processing, as threads must often exchange data. Additionally, thread synchronization is an important aspect, as rendering often requires synchronization between threads to ensure correct execution, for example, when calculating shaders or processing frames. Suboptimal synchronization can create delays and lead to unnecessary resource overhead. Additionally, some rendering steps may require processing in a specific order, which limits the ability to execute in parallel. For example, after calculating shadows, color or reflection adjustments may need to be made, which requires sequential processing. Another limitation is the threading limit in render engines, as some may not be able to efficiently use many threads due to a lack of optimization for parallel execution. This can be especially true for renderings that rely on complex computations, such as ray tracing. Software limitations, such as using older versions of OpenGL or DirectX, can also reduce the efficiency of threading. A final limitation is algorithm-level parallelism. Some rendering algorithms require many sequential computations, and they cannot be easily adapted to efficiently use multiple threads. Thus, while using threads can significantly speed up the rendering

process, there are limitations that depend on hardware and software resources, the structure of the algorithms, and the type of data being processed.

The development of streaming information processing in rendering is an important issue for ensuring high efficiency, scalability and realism in modern computer graphics and 3D modeling systems. One of the main directions is the use of parallel computing on graphics processing units (GPUs), which allows you to efficiently process many pixels or vertices simultaneously, providing significant acceleration of rendering. The use of technologies such as CUDA and OpenCL allow you to increase the efficiency of parallel processing and accelerate complex scenes. Another important direction is ray tracing technology, which allows you to achieve realistic reflections, light and shadows, which was previously a resource-intensive process. Streaming ray tracing allows for real-time rendering, which opens new possibilities for interactive applications. Real-time rendering is also becoming possible thanks to streaming algorithms that allow processing large amounts of data, for example in games or simulations, using distributed rendering, where information processing can be performed on multiple devices simultaneously. Streaming texture and geometry processing is also an important direction, especially for optimizing the rendering of complex scenes with large amounts of data. Techniques such as mipmap and level of detail (LOD) help reduce the load on resources by optimizing textures depending on the distance to the camera. In addition, machine learning and artificial intelligence are actively used to improve the quality of rendering, to remove noise during ray tracing or automatically optimize rendering processes to achieve better results. Distributed computing and cloud technologies also play an important role, as they allow rendering to be performed using scalable computing resources, which makes it possible to process scenes in real time or at post-processing stages without loading local devices. These areas allow you to increase the performance, quality, and realism of rendering in modern computer systems.

Conclusion.

In summary, the evolution of GPUs from simple rasterizers to powerful parallel computing machines is an impressive example of technological progress. Unlike the

sequential architecture of CPUs, GPUs are designed to process large amounts of data simultaneously, using a streaming model of computation at the thread, core, and vector levels. This parallel architecture, complemented by specialized computational units and memory optimizations, allows GPUs to achieve significant performance in graphics computing and a wider range of tasks, including artificial intelligence. Innovations in architecture, the development of software interfaces, and algorithmic optimizations provide developers with powerful tools for efficient use of GPU resources. Thanks to these achievements, GPUs remain a key element of modern computing, determining the future direction of technology development.