Improved Graphic Rendering Using Parallelization

Shane A. Smith, Joshua Gearhart

**Abstract**—Graphic rendering is a very important part of many industries and therefore must be focused on in order to be improved to reduce cost and increase performance. One of the primary methods that have been researched and implemented in graphical rendering has been the utilization of parallelization in both physical and virtual representation. More specifically, we will be focusing on the physical developments of parallelization of Graphic Processing Units (GPUs). Graphical rendering has primarily been done on GPUs since they have been specifically designed and calibrated to be fed information to do fast calculations in order to render an image. GPU’s have been built to include natural parallelization by using technology such as Nvidia’s CUDA cores and AMD’s Stream Processors as well as the software aspect utilizing the popular searching algorithms sort-first, sort-middle, and sort-last.

**Index Terms**—Parallel Algorithms, Parallel Architectures, Graphical Processors, Graphic Rendering

————————————————

1. J.S. Author is with Slippery Rock University, Slippery Rock, PA 16057. E-mail: jjg1018@sru.edu
2. S.S. Author is with Slippery Rock University, Slippery Rock, PA 16057. E-mail: sas1041@sru.edu

—————————— u ——————————

# 1 Introduction

In this paper, we will talk about the graphics card and the Graphic Processing Unit (GPU) advances that have made these technologies more dynamic, efficient, and more powerful. The primary goal of graphics cards is to do graphic rendering calculations which are, in short, converting descriptive information of a digital image such as geometry, viewpoint, texture, lighting, and shading, etc. into a multi-dimensional space [Parallel Rendering]. Primarily, these calculations are performed with the multi-GPU organization on graphics cards. This paper will cover the architecture of modern-day graphics cards, including the inherit parallel nature of them. Modern-day graphics cards have been made to incorporate parallel computation that programs must take advantage of. This paper will include the primary methods that software developers have been able to parallelize their graphic renderings software. This has been done with various sorting algorithms that these computers use to process image data.

# 2 Literature Review

**2.1 “Cuda Refresher: Reviewing the origins of GPU computing”**

A in-depth article about the history of CUDA and talks about the differences between CPU’s and GPU’s as well as their differences in growth and architecture as well as how they process differently.

**2.2 “Cuda Overview”**

This article is an informational article on CUDA and how it works. It starts with a brief history on what is CUDA, then the article talks about the architecture of CUDA as well as its memory hierarchy and shared memory, as well how it runs and works and computes as well as the API to work with CUDA.

**2.3 “Cuda Refresher: The Cuda Programming Model”**

The informational article was very interesting to read and learn about, it talks heavily about the use and API of coding with CUDA as well as shows an example, it talks about the memory hierarchy and coding with CUDA better than the previous article. As well as it reinforces the knowledge in the article well with figures.

**2.4 “Cuda Refresher: Getting Started with Cuda”**

The article was a start to learning about how to code with CUDA but what I found especially relevant was the talk about scalability and how scaling does not have to be set by the programmer as it is already planned in by the CUDA runtime.

**2.5** “[**Mixed-mode database miner classifier: Parallel computation of graphical processing unit mining**](https://www.researchgate.net/publication/349476061_Mixed-mode_database_miner_classifier_Parallel_computation_of_graphical_processing_unit_mining)”

This article seemed interesting and useful to our paper though no information was used, I believe the way they explained the process of flow of information in CUDA was especially useful for this reason this resource was used for their figure.

**2.6 “Parallel Rendering – Fast Graphics”**

This article primarily discussed the different types of parallelism and a few of the various algorithms used to render computer graphics. The three types of algorithms that are focused on are the Sort-first, Sort-middle, and Sort-last which are used depending on where the location of the algorithm is called in the rendering pipeline. The information reported in this article is critical to understanding the way algorithms can affect the performance of a graphical rendering.

**2.7 “An Introduction to parallel Rendering”**

Diagram

Description automatically generatedAlthough this article is somewhat dated for today’s standards it is able to provide valuable insight into basic ideas of what computer rendering is and the necessary requirement of using a parallel computer architecture to meet the high-power requirements for complex rendering. It is also useful for identifying the different ways in which parallelism is applied in the many different steps in the process of rendering graphics.

**2.8 “MPI-Hybrid Parallelism for Volume Rendering on Large, Multi-core Systems”**

This article contributes to the paper by providing information on the effect of hundreds of thousands of GPU cores interacting in a parallel architecture in a High-Performance Computing (HPC) environment. Specifically, this work focused on a large data set doing “raycasting volume rendering”. The researchers were able to test the scalability of the rendering utilizing a hybrid approach to parallel volume rendering. The importance of this topic is in the scalability of the system as we know that more parallelism causes more communication which can face demising returns. Information in this article is also significant to explain an implementation of both sorting level hybrid parallel algorithms as well as architecture level hybrid parallel systems.

**2.9 “Multi-GPU Rendering with Vulkan API”**

The importance of this article lies in the idea of implementation. The author of this paper was able to identify that the ability to control and execute commands on explicitly separate GPUs was invaluable when creating graphic rendering on an open API system. While the ability to explicitly render on specific GPUs is possible with extensions of other APIs it is not directly supported without packages provided by the specific graphics card manufacturers. This article provides an overview of the Vulkan API used to create efficient parallel systems with the direct ability to assign explicit tasks and processes to specific GPUs.

Line chart

Description automatically generated**2.10 “Hybrid Sort-first and Sort-last Parallel Rendering with a Cluster of PCs”**

In order to gain more information and examples on how to create hybrid mixes of algorithm’s this article has been included. The article covers two of the most utilized sorting algorithms in the graphical rendering pipeline, Sort-first and Sort-last in parallelization with a large cluster of PCs. The mix of these algorithms is applied to a graphical polygon rendering. This project had many goals including: finding lower costs in order to increase the price-to-performance ratio of graphical hardware, allowing consumers to increase performance and extend the lifetime of their hardware, create flexible network systems, and increase scalability in a linear fashion as the capacity of a PC cluster grows.

# 3 Methods

**3.1 Graphics Cards**

Originally graphics cards were created for the use of only graphical processing, as graphics cards needed to become stronger and faster to deal with the higher graphical demands of consumers. As graphics cards advanced it was realized that there was a strong processing power located in a computer that wasn’t being fully utilized. In fact, this advancement of graphics surpassed the rate of Moore’s law, this performance was noted by Professor John Poulton as we can see in figure one [1].

Fig.1. Graphics performance prediction measured in triangles/per second [1].

The need for power in graphics made an urge for advancements to graphics cards. These advancements have changed the overall use of graphics cards, making a graphics card a more diverse and useful product. This was done through the architecture of the graphics cards. There are two variants of this advancement, the first being AMD’s Stream Processors, and the other being NVIDIA’s developed CUDA.

**3.2 CUDA**

NVIDIA’s advancement of the CUDA which stands for a “Compute Unified Device Architecture” allows for the use of general-purpose computing on graphical processing units, this action has been called GPGPU’s. The importance of this advancement is that it now allows your everyday graphical processing unit to communicate with the CPU as well as run general computations on the graphics card rather than only being used for image processing. This fuller utilization of the graphics card allowed them to run large parallel algorithms for a faster speed-up than CPU’s. This is shown in figure 2.

Fig. 2. Theoretical Peak Performance of GPU’s vs CPU’s. [2]

**3.3 How CUDA Works**

Table

Description automatically generated with medium confidenceCUDA Programming starts out with code being run on the CPU. Then each function or problem to be solved known as a kernel gets sent to the GPU to be computed. This communication occurs through the PCIe bus [3]. CUDA programming works in a way that allows scaling when the number of processors is increased. This is because CUDA programming defines the program into a set number of blocks. Then through the CUDA runtime the blocks are scheduled to different streaming multiprocessors known as SM’s. These blocks can be worked on in any order. Figure 3. Shows this scalability on different graphical processors with different amounts of SM’s.

Fig. 3. Blocks assigned to different GPU’s [4].

Each of these blocks is a group of threads and each of these thread blocks becomes assigned to a grid of thread blocks that work on the CUDA kernel or the problem. It’s also important to note the memory of CUDA Graphics cards. A CUDA graphics card has a register which are only available to each thread in the graphics card. The registers are also unique to each thread and one thread cannot access another's register These registers are in the Stream processors (SM’s) and these registers communicate with an L1 cache which is the fastest cache available to the stream processes. These L1 caches are used as shared memory for all the threads in CUDA blocks assigned to the processor where it resides. On the same hierarchy as the L1 is also the read-only memory this memory is used for kernel code and for reading valuable information the coder might need. Next is the L2 Cache, which is out of the SM’s so every SM can access this memory, this allows for all the threads in CUDA blocks to access the memory in this cache, as well as it is a bigger cache than the L1 memory. After that, there is the global memory where memory and instruction from the host (the CPU) gets sent to the device (the GPU). The importance of this memory and knowledge of it, allows programmers to manipulate memory resources for optimization. Below in figure 4. Is a physical representation of the memory architecture of CUDA graphics cards [3].

Fig. 4. Memory layout and architecture of a CUDA graphics card [3].

To better describe the flow of information in a CUDA instructed program and how the host CPU talks to a device GPU see figure 5 below.

Fig. 5. Data flow of a host to a device.

**3.4 Types of Graphics Parallelism**

Throughout the process of creating a parallelized graphic rendering using GPUs, there are several types of parallelism that can occur based on the step that the graphic rendering

Fig. 6. A typical polygon rendering pipeline.

is in. The primary types of parallelism in graphic rendering are identified as Functional parallelism, Data parallelism, and Temporal parallelism.

The purpose of Functional parallelism is typically constructed a rendering pipeline decomposed into two parts: the geometry processing and Rasterization [6]. These different functions are applied to a series of different processes when they are fed through the pipeline, they are then worked on, completed, and call the next process to be fed into the pipeline.

Data parallelism can be seen as a contrast to Functional parallelism since instead of creating a singlular pipeline that assigns different functions it creates many data streams, operating on each of those streams simultaneously and replicating data on from other streams to complete a process. This method is favored by many software developers since it is a very scalable implantation. The two types of Data parallelism are Object parallelism and Image parallelism which are both part of the pipelines. Object parallelism occurs first in the pipeline which are performed independently on geometric primitives of the scene render and is part of the Transformation Phase while Image parallelism occurs in the later steps of the pipeline which compute individual pixel values and is part of the Rasterization Phase [7].

Diagram

Description automatically generatedThe last type of graphic rendering parallelism is Temporal parallelism which used in animation or video applications. This is because in these situations many frames must be rendered simultaneously in order to produce a playback. Parallelism is created in this situation by decomposing the playback into the individual frames, grouping them, and assigning them their own processor to render a frame, and then combining them at the end to produce the required playback [7]. Another aspect of parallelism which can be utilized is a Hybrid approach which can be summarized as using any of the other methods together to create a more parallelized architecture; we will discuss this method in the following section.

**3.5 Hybrid Parallelism**

There are two primary methods of creating a hybrid parallelism system. The two methods can be defined by looking at the scope of the system structure, whether the hybrid method is being applied in the sorting algorithm level or the architecture level. Typically, these two systems may be used in conjunction in order to further increase the parallelized speedup based on the task being processed. Looking at the algorithmic approach of hybrid parallelism, there are many examples that can be shown on how to implement useful processes' of combining Functional, Data, and Temporal parallelism. For example, a speed increase can be shown by combining a functional and data-parallel approach inside of a graphic rendering pipeline. This approach was used by Silicon Graphics’ Reality Engine to improve their rendering rates in a highly pipelined architecture [6].

Diagram

Description automatically generatedOther than the algorithmic scope level, hybrid parallelism can also be applied in the system architecture. This can be achieved in a MPI-hyrbid case that was constructed of both a shared-memory parallel application as well as a distributed-memory parallel application [8]. It is able to achieve this hybrid system by utilizing a distributed-memory application by creating many different *n* processes which perform computation and then exchange messages to create an image. It then uses a shared-memory system during the thread computations [8]. This MPI-hyrbid model can be seen below.

Fig. 7. MPI-hybrid block level view [8].

**3.6 Increasing Performance with Algorithms**

Another aspect to consider when trying to increase the performance and capabilities of graphic rendering, is the graphic rendering algorithms that are being run in order to do these intensive graphical computations. In order to effectively contribute to the scalability of rendering with hardware, the primary algorithms Sort-first, Sort-middle, and Sort-last. Each have their own advantages, disadvantages, and specific uses inside of the rendering pipelines. Within Sort-first, the aspects that need to be focused on to improve performance are that it is susceptible to load-imbalance and is poor for scalability on many processors. Its goal is to arbitrarily assign tasks to processors, perform an assessment, and then redistributes the tasks to processors with other tasks located near it [6].

Fig. 8. Sort-first algorithm example [6].

The second primary algorithm, Sort-middle, is used to redistribute objects in the middle of the rendering pipeline, implying its name. In application, this algorithm sorts geometry processing and rasterization processes onto separated processors which can either share memory or be created as separate independent threads. For every frame, the geometry calculation is done and labeled by their screen size and then distributed to the rasterization calculation to be rendered [6]. Like the Sort-first algorithm, it can be susceptible to load-imbalence when running in parallel if the objects that are being calculated on become undistributed evenly over the screen. This algorithm also requires high communication resources when performing a high tessellation ratio [6].

Diagram

Description automatically generatedFig. 9. Sort-middle algorithm example [6].

A picture containing diagram

Description automatically generatedFinally, the last algorithm adopted in many parallel rendering systems is Sort-last. This algorithm’s responsibility is to distribute all of the object calculations to each of the rendering processors. In this algorithm, every processor operates on the data completely independent until all calculations have been completed. Then the sorting is done by transmitting the pixels across a network of processors which receive images in order to complete the final image. The major flaw with this sorting algorithm is in the final re-composition step, because is very expensive due to the amount of pixel data that must be transmitted [6].

As previously mentioned though, it is possible to alleviate some of these flaws within the algorithms. This is done by creating a hybrid-sorting algorithm using each of the benefits of the algorithms at various stages in the graphical pipeline [10]. (IF MORE NEEDED TALK ABOUT [10])

# 4 Results

From the methods that we have examined, there are many conclusions to be made about how parallelization can improve graphical rendering. CUDA and the ability of GPGPU’s has greatly increased the effectiveness of graphics cards as well as making them a much more rounded unit in the computer, the enabling of GPGPU’s has made high performance computing stronger in this manner as well, the extra and stronger piece of equipment has made great lengths in all fields of scientific computing and will further increase the speed at which intense calculation-based algorithms are solved. Another way that we have seen graphical rendering with parallelization improved has been by looking at altering the known and standard parallel algorithms and architectures into more effective and efficient creations. This has been achieved in many ways and primarily depends on the scope at which the system is being altered at. Looking at the architecture on the processors doing computations, a performance increase has been shown by incorporating a MPI-hybrid system in a High Performance Computing Space doing Raytracing renderings. In order to measure this increase performance multiple different factors were considered such as Ghost Data, Raycasting, and Composting. In this experiment two layers of Ghost Data must be used, due to the nature of the hybrid approach it uses a lower amount of data blocks that needto be communicated and therefore saw a 40% decrease in all data exchange and data storage, as seen below.

Chart, line chart

Description automatically generatedFig. 10. Decrease in Ghost Data for MPI-hybrid [8].

Other than the architecture scope of increasing performance in parallel graphic rendering, there is also an increase in performance involving the sorting algorithms in the graphical rendering pipeline. Similar to the architecture level performance increase, creating a hybrid algorithm based off of the Sort-first and Sort-last created an efficiency of up to 70.5% better than the previously tested in a network of 64 PCs. This was done by creating a three-step process. The first phase utilizes a parititioning algorithm similar to the standard Sort-first algorithm which decomposes the polygon. Next, the second phase assigns a polygon to every server which then does computation and communicates with the other servers with its computed data. The last step combines the data from all servers utilizing an algorithm similar to the Sort-last standard without using a depth-frist comparison.

Fig. 11. Multi-phase Hybrid Sort-first/Sort-last Algorithm [10].

# 5 Discussion/Conclusion

From our research we have found that the creation of CUDA and other GPGPU enabling hardware has greatly increased not only the lengths and a drive for a stronger GPU but also used the strong processing power that was already located in the GPU. As GPU’s increase in performance this aspect of the CUDA core will also increase, and lead to stronger computations being done by GPU’s. From our research we also have found that mixing methodologies of sorting algorithms to create new hybrid sorting algorithms can greatly increase the performance done by them, with this research continuing we believe that greater performances will be achieved, and more research in this topic and perfecting them will see greater and greater performance metrics. We also would like to look into the future of the graphics card. Particulary, the future of CUDA and how it could impact ray-tracing technology, The possibilies of new graphical rendering sorting algorithms and how they could be improved. And how commercial vs industrial hardware parallelization can increase performance in the graphical rendering field.

# 6 References/Bibliography

1. P. Gupta, “Cuda Refresher: Reviewing the origins of GPU computing,” *NVIDIA Developer Blog*, 22-Jun-2021. [Online]. Available: https://developer.nvidia.com/blog/cuda-refresher-reviewing-the-origins-of-gpu-computing/. [Accessed: 28-Nov-2021].
2. M. Romero and R. Urra, “CUDA Overview,” *CUDA Programming*. <http://cuda.ce.rit.edu/cuda_overview/cuda_overview.htm>. [Accessed: 26-Nov-2021].
3. P. Gupta, “Cuda Refresher: The Cuda Programming Model,” *NVIDIA Developer Blog*, 25-Aug-2020. [Online]. Available: <https://developer.nvidia.com/blog/cuda-refresher-cuda-programming-model/>. [Accessed: 28-Nov-2021].
4. P. Gupta, “Cuda Refresher: Getting started with Cuda,” *NVIDIA Developer Blog*, 22-Jun-2021. [Online]. Available: https://developer.nvidia.com/blog/cuda-refresher-getting-started-with-cuda/. [Accessed: 28-Nov-2021].
5. S. R. Nayak, S. Sivakumar, A. K. Bhoi, P. K. Mallick, and G.-S. Chae, “mixed-mode database miner classifier: Parallel computation of Graphical Processing Unit Mining,” *ResearchGate*, Feb-2021. [Online]. Available: https://www.researchgate.net/publication/349476061\_Mixed-mode\_database\_miner\_classifier\_Parallel\_computation\_of\_graphical\_processing\_unit\_mining. [Accessed: 29-Nov-2021].
6. L. Sun, “Parallel Rendering – Fast Graphics”, https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.570.7478&rep=rep1&type=pdf [Accessed: 9-Dec-2021].
7. Thomas W. Crockett, “An introduction to parallel rendering”, https://www.sciencedirect.com/science/article/abs/pii/S0167819197000288 [Accessed: 9-Dec-2021].
8. M. Howison, E. W. Bethel, and H. Childs, “MPI-hybrid Parallelism for Volume Rendering on Large, Multi-core Systems”, <https://www.osti.gov/servlets/purl/983174> [Accessed: 9-Dec-2021].
9. Lars Olav Tolo, “Multi-GPU Rendering with Vulkan API”, https://bora.uib.no/bora-xmlui/bitstream/handle/1956/19628/report.pdf?sequence=1 [Accessed: 28-Nov-2021]
10. R. Samanta, T Funkhouser, J. P. Singh, “Hybrid Sort-First and Sort-Last Parallel Rendering with a Cluster of PCs”, https://gfx.cs.princeton.edu/pubs/Samanta\_2000\_HSA/gh2k.pdf , [Accessed: 9-Dec-2021]

**Shane A. Smith** Bachlors in Computing and Cybersecurity, Student and Web Assistant for Slippery Rock University.

**Joshua Gearhart** Bachelors in Computing and Cybersecurity, Student.

**Figures**

Diagram

Description automatically generated

Line chart

Description automatically generatedFig.1. Graphics performance prediction measured in triangles/per second [1].

Fig. 2. Theoretical Peak Performance of GPU’s vs CPU’s. [2]

Table

Description automatically generated with medium confidence

Diagram

Description automatically generatedFig. 3. Blocks assigned to different GPU’s [4].

Fig. 4. Memory layout and architecture of a CUDA graphics card [3].

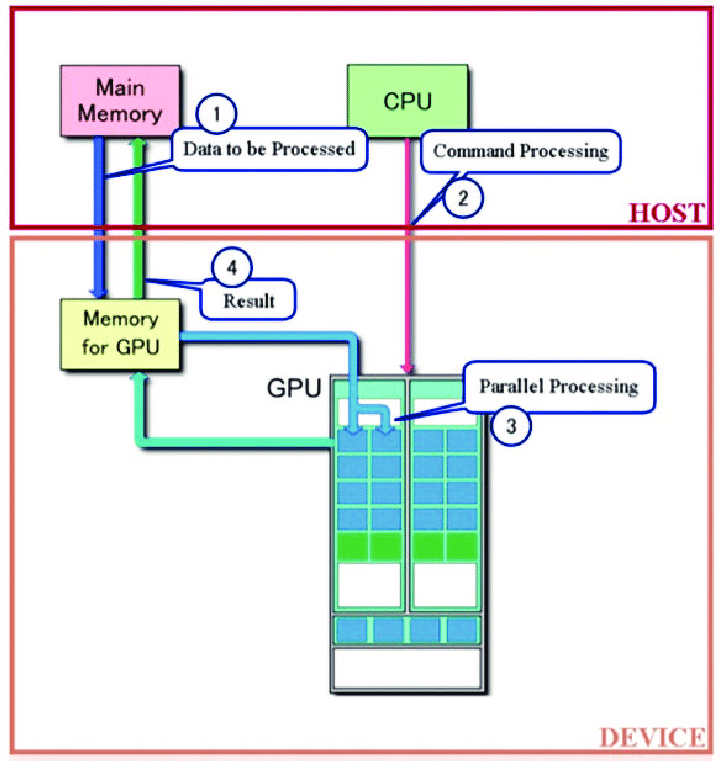
Fig. 5. Data flow of a host to a device.

Figure 7

Figure 8

Figure 9

Figure 10Diagram

Description automatically generated