Universiteit Leiden

Master Computer Science

DAS-BR: A tool for heterogeneous and distributed Blender renders

Name: Brandon Kroes
Student ID: s3329879
Date: 22/07/2023

Specialisation: Computer Science and Education

1st supervisor: dr. A. Uta
2nd supervisor: dr. K.F.D. Rietveld

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

Rendering is the process of generating images out of a digital environment using a computer. Rendering is the foundation of all modern video games and blockbuster movies. As rendering has become more mainstream, so has the drive for ever more immersive, photorealistic, and advanced scenes, this has dramatically increased the complexity of the render process. As a single system is constrained in its rendering capabilities, it becomes imperative to utilize multiple systems.

A rendered scene can feature complex details in an environment, from lighting and textures to shaders and animations. Render scenes are created in tools such as Blender, a cross-platform and free 3D creation suite. Blender lacks the functionality to efficiently distribute renders on a small-scale computer cluster, even though distributing across multiple servers becomes a necessity at larger scale.

To solve Blender's limitation, this thesis introduces a software system called DAS-BR. DAS-BR has been designed to be scalable in both homogenous and heterogeneous environments. The performance of DAS-BR has been evaluated through the execution of experiments on GPU environments, the selection of the distribution algorithm, a thorough examination of the environment, and a comparison to a first-party tool for distributed rendering. The evaluation results demonstrate that DAS-BR is an efficient and capable system for distributing Blender renders across a small-scale computing cluster.

Contents

Preface

The first film I ever saw in a cinema was Toy Story. With its runtime of 77 minutes, it amazed me as a child. At the time, Toy Story was revolutionary as it was the first feature computer-generated animation film [1]. With the development of Toy Story came new computer challenges. The film required a tremendous amount of computer power to render all the frames. Toy Story required more than 800,000 computation hours to compute [2]. If the film were computed on a single computer instead of distributed across a cluster, I would not have had the chance to see it as a child, nor as a teen, not even when I was an adult. The film would have taken 91 years of rendering. Therefore, it was not merely a desire to distribute rendering, it was a necessity. The goal of this thesis is the research and development of a distributed and heterogeneous rendering system capable of rendering computer-generated content, which provides a low-cost and convenient way of scaling and speeding up renders.

Chapter 1

Introduction

Rendering is the process of generating images out of a digital environment using a computer. Pioneered in 1962 with the computer program Sketchpad [3], developed by Ivan Sutherland. In 1988, Dr. Sutherland received a Turing Award for this work. Since 1962, computer graphics and rendering by extension have been improved in terms of realism, diligence, and scale. Rendering has become a pillar of the entertainment [4], architectural [5] and automotive [6] industry. As rendering has become more mainstream, so has the drive for ever more immersive [7], photorealistic [8], and advanced scenes, this has dramatically increased the complexity of the render process.

(a) A dog in Toy Story (1995) **(b)** A cat in Toy Story 4 (2019)

Figure 1.1: Screenshots of the 1995 Toy Story film and the 2019 sequel, highlighting the difference in graphic fidelity throughout the years

Due to the complex details of the scene, such as lighting, textures, shaders and animations, the rendering process can take a long time. The complexity and size of the environment determine the amount of rendering that needs to be done [9]. The render time is based on the availability of computer hardware and the complexity of the rendered scene [9]. The result of a render is usually a single frame or, in the case of animation, a series of frames.

In 2023, graphic artists use tools like Blender to create render-able scenes. Blender is a cross-platform free and open-source 3D creation suite. Blender is used for modelling, animation, simulation, and texturing. Blender features the most demanding graphics features, such as global illumination and reflections.

To render complex scenes, the need for powerful hardware and the utilization of the hardware has become a pressing matter. As the computing power of a single machine is limited by Moore's law, it would be preferable to distribute the process of rendering to multiple machines. The Blender organization has developed a tool capable of distributing renders, named Flamenco. Flamenco was designed to be used in a local network of workstations, while this thesis investigates the distribution of render workloads for a small-scale cluster of computers. To this end, we will use the small-scale computer cluster DAS6 located at the Advanced School for Computing and Imaging as our target platform. Running Flamenco on DAS6 results in numerous challenges, ranging from the inability to launch multiple workers to the inability to access GPUs. To facilitate a small-scale computer cluster capable of distributed Blender rendering, a new system was required.

Developing a standalone system to distribute Blender renders has been done in the past [10, 11], however, most attempts are several years old, are closed source, only scale with homogenous hardware, lack acceleration by the graphics processing unit (GPU) or are unable to run on a small-scale computer cluster such as DAS6. The goal of this thesis is to develop a system that can alleviate all these points. To develop such a system, the following research questions have been asked and answered:

RQ1 How can Blender renders efficiently be scaled across a distributed computing cluster?

RQ2 What effect does GPU acceleration have on render times compared to CPU?

RQ3 How can Blender renders efficiently be scaled across heterogeneous hardware?

RQ4 What are the benefits of a distributed render compared to Blenders native rendering?

To answer all these questions, this thesis has developed a prototype system capable of GPU- and CPU-based rendering across a diverse and distributed computing cluster. The objective of this thesis is to provide a comprehensive description of this system and demonstrate its performance through a series of experiments.

Chapter 2

Background

2.1 Heterogeneous computing systems

Merriam-Webster defines homogenous as "of the same or a similar kind or nature." Within the scope of computing systems, this means that a homogenous cluster of computers contains the same hardware and software. Homogenous computing contrasts with heterogeneous computing. In heterogeneous computing, a system contains different hardware and software. Heterogeneity in computing can be found in many ways, from different server farms with their speciality for a single enterprise application to a CPU-die containing different cores to allow for better power efficiency[12]. The scope is broad, which is why we have outlined a specific focus for this thesis. Within the scope of this thesis, we refer to heterogeneous computing systems as servers within the same cluster that are equipped with different GPUs. The goal of using different GPU hardware is to accelerate the render speed and decrease the total render time by utilising the available GPU compute power. Using different GPUs allows us to leverage one of the big advantages of a heterogeneous system, which is scalability without needing homogenous hardware. The heterogeneous scalability allows adding more servers without the need for identical hardware. This means that a cluster can be modified without an entire overhaul to the cluster. This does not mean that heterogeneous computing does not come without its challenges. Scheduling, allocating resources and utilisation remain a constant challenge when it comes to heterogeneous computing [13], this is because the variety of hardware and software means that each computer needs tasks tailored to their requirements and capabilities for high utilisation.

2.2 DAS

The Distributed ASCI Supercomputer (DAS) is a Computer Science infrastructure designed by the Advanced School for Computing and Imaging (ASCI) for controlled experiments with parallel and distributed systems [14]. DAS is a collaborative effort between public universities and ASTRON, the Netherlands' Institute for Radioastronomy [15]. DAS has six clusters spread around the Netherlands, all connected via 1GB/s [16]. Multiple versions exist of DAS, all are named by the number of their sequence [17]. For example, DAS-5 is the fifth iteration of DAS. The most recent version is DAS-6. DAS-6 makes use of the Ada generation of NVIDIA GPUs and AMD Rome CPUs. Accessing DAS6 is facilitated by SLURM. SLURM is a cluster workload manager, allowing users to reserve and use the available hardware of DAS6 for research purposes.

Cluster site	CPU	GPU
VU	24-core AMD EPYC 7402P	A4000, A6000, A100, A2
TU Delft	2 × 16-core AMD EPYC 7282	A4000, A5000

Table 2.1: Example DAS hardware configuration

2.3 Hardware specifications

The render capacity of a component is based on its hardware [18]. Two specifications of hardware are of particular importance for this thesis, namely clock speeds and thermal design power.

2.3.1 Clock speed

The processor clock coordinates all the CPU and memory operations by periodically generating a time reference signal called a clock cycle or tick. Clock frequency is specified in megahertz (MHz), or millions of ticks per second, and determines how fast instructions execute[19]. This means that a higher clock speed is generally preferable over a low clock speed when talking about identical chips. The manufacturer will often set a base-clock speed that a component is expected to reach and a boost clock that a component can reach under high-load situations. The boost clock is not the highest achievable clock. Modern GPUs and CPUs attempt to boost to the highest stable clock speed. The mechanism for the boosting behaviour is referred to as dynamic frequency scaling or automatic overlocking (Auto-OC). Auto-OC can therefore go beyond the boost clock. The clock speed that Auto-OC reaches is not the same for processors of the same make and model. The clocks that a component can reach depend on silicon quality, available power, workload, thermal headroom and software configuration.

2.3.2 Thermal Design Power

Thermal design power (TDP) is the amount of heat generated by a computer component that needs to be dissipated for the manufacturer's expected performance [20]. TDP is not the maximum amount of heat a system can dissipate. Auto-OC can and will increase the power draw, and by extension the component heat, above manufacturer specification.

2.3.3 Thermal Throttling

Thermal performance is one of the determining factors of hardware performance. The temperature of a component plays a role in the clock speed, this in turn determines the computational power of a component. If a component is not adequately cooled, it will experience a phenomenon called thermal throttling, which means that the clock speed is lowered for stable and safe operations [21]. Thermal throttling is a limit usually built-in by the manufacturer or designer of the component [21]. The thermal throttle temperature is called the maximum junction temperature (TJMax). The reduction of a processor clock speed effectively reduces its computing power at that particular moment. A reduction of computing power might sound disadvantageous, but it serves an important purpose, namely protecting the hardware. Elevated temperatures risk damaging or reducing the lifespan of a component [22, 23].

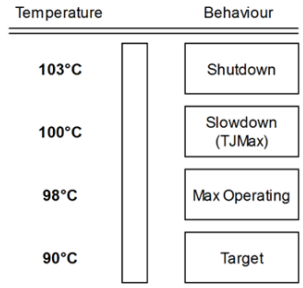


Figure 2.1: NVIDIA A4000 Temperature levels

When monitoring the performance of a component, there are four temperature levels of importance:

- Target, this is the temperature that Auto-OC takes into consideration when determining if it can boost higher and sees as the target of temperature.
- Max operating, This is the temperature level at which Auto-OC will stop boosting higher.
- Slowdown, Thermal throttling will initiate at this point, reducing the clock speed.
- Shutdown, The processor will perform a hardware shutdown to protect itself.

In the past (20 years ago) thermal throttling was less of an issue as components could be cooled by a small fan [24]. Today, the computational power of data centres is more limited by the cooling capacity than the actual hardware [25]. This is due to both the higher component heat and increased server density [26]. This has led to the rise of more complex cooling such as water loops and immersion cooling [27].

Thermal throttling is of particular importance for doing long and computationally intensive operations, such as rendering. Thermal throttling can take a while to develop, as an inadequate cooling solution can dissipate the initial burst of heat. A sustained workload on an inadequate cooling solution will eventually overwhelm the cooling solution, which results in thermal throttling. It only becomes evident after a prolonged experiment that a thermal solution is inadequate. For this reason, this thesis contains an environmental analysis in which we detail the hardware performance of DAS-6.

Chapter 3

Related Work

As mentioned in the introduction, creating a distributed Blender rendering system has been attempted before. In this chapter, we discuss the existing work and where this thesis intends to improve upon.

3.1 Existing work

One of the earliest attempts at such a system was detailed in M. Z. Patoli et al. [10]. The paper details a service that allows users to submit Blender projects for rendering and collaboration. After a Blender project has been completed, the user can submit the project for rendering. The most recent attempt was Ganesh V. et al. [11]. In the paper, a low-cost, distributed, and efficient rendering system is introduced. The author detailed how earlier research was focused on large-scale render farms inaccessible to small-scale users. Comparative testing with the earlier work was not possible. This was attributable to non-supported versions of Blender, the closed-source nature of the software, or a lack of functionalities. Furthermore, previous systems lack the three features that this thesis implements, which would make a comparison ineffectual. The three features are heterogeneous rendering, performance analysis and GPU-usage.

3.2 Flamenco

As previously mentioned, the Blender organisation has created the tool Flamenco to aid in distributed scaling for Blender. Flamenco is a lightweight, open-source, cross-platform framework to dispatch and schedule rendering jobs for smaller teams, or individuals [28]. There are three primary challenges when using Flamenco on DAS6, path related issues, GPUs incompatibility, and the dashboard access.

3.2.1 Path related issues

Flamenco credentials and configuration files are stored in the home directory. DAS6 synchronises the home directories across all workers. The synchronization stops the creation of multiple workers, as the newest worker overrides the previous workers' information, rendering the previous worker inoperable. Flamenco has no command-line interface functionality allowing the remapping of these paths. In the Flamenco experiments, the source code has been modified to permit local paths, thereby enabling the utilization of multiple workers on DAS6.

3.2.2 GPU compatibility

At the time of writing, it was not possible to use GPUs for Flamenco renders. The Flamenco system couldn't convey instructions for GPU rendering, manually forcing instructions resulted in worker instabilities. The underlying cause of this issue remains uncertain, and it is possible that it may stem from a driver issue, a permission issue, or a storage issue. The issues with GPU rendering seem to be specific to the combination of DAS6 and Flamenco, not necessarily to Flamenco or DAS6 themselves.

3.2.3 Dashboard

Users can interact with the Flamenco manager through the dashboard, a web-based interface. Using the web-based interface, users can connect to their local installation of Blender using an address and a port. When Blender is instructed to render their project, the local installation will contact the web server and instruct Flamenco to render the project. This creates technical hurdles to overcome when the Flamenco manager is located on a DAS6 node. DAS6 is inaccessible to the public; therefore, it necessitates SSH tunnelling from a pre-approved location. The Flamenco manager's accessibility is limited to a web-server, which means that a user would require a proxy to reach the server. Since DAS6 is not publicly available, this requires multiple proxy servers. Imagine a student enrolled at the Leiden Institute of Advanced Computer Science (LIACS) who required tunnelling of their local Blender installation to the Flamenco manager. The student would need to perform 3 tunnelling steps, as shown in Figure 3.1, to access the dashboard. This need for tunnelling complicates it for users who are less technically inclined and locks up ports on the shared servers.

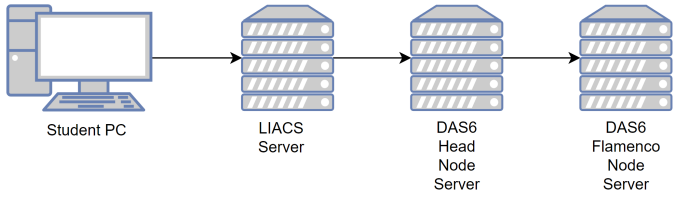


Figure 3.1: Tunnelling steps to access the Flamenco dashboard

3.2.4 Conclusion

There were additional concerns regarding Flamenco on DAS6, including the issues arising from the utilization of the Windows version of Blender for job submissions. However, we will refrain from addressing these issues as, based on the previous three obstacles, it can be inferred that Flamenco is not suitable to be used on a small-scale computer cluster. The three obstacles could be resolved by implementing further extensive modifications to the code base; however, this would transition Flamenco to DAS-Flamenco. In conclusion, Flamenco is not suitable to be used on a small-scale computing cluster such as DAS6 because it introduces user inconvenience for interacting with the dashboard by needing SSH tunnelling, it requires codebase alterations to solve path related issues, and lacks GPU acceleration.

3.3 Heterogeneous rendering

Despite the extensive research into the creation of a distributed Blender rendering system, there is a significant overlap in the content of the various papers. The idea of heterogeneous rendering is mentioned in the future work section of every paper, but no papers have developed such a system.

3.4 Performance analysis

In the background chapter, we discussed how render performance varies depending on the hardware and its operational environment. The effects of the environmental conditions can take some time to occur, so the goal of this thesis is to thoroughly study the environment where experiments are done and to analyse the impact on performance.

3.5 GPU usage

GPUs are capable of outperforming CPUs in rendering [29]. Previous attempts have not featured the utilization of GPUs. Flamenco can use GPU acceleration; however, this was not possible on DAS6. GPU acceleration merits further investigation, as low-cost GPUs possess the capability to surpass CPUs with a higher price tag in rendering tasks. This thesis aims to use the available GPUs in DAS6 to accelerate renderings and to demonstrate the impact that GPUs can have on the rendering duration.

Chapter 4

DAS-BR System design

In the previous chapters, we discussed the research questions, the relevant topics, and the challenges this thesis aims to solve. This chapter presents a prototype system that allows the challenges to be solved, and details the decisions that were made in its design.

4.1 Overview

DAS Blender Render (DAS-BR) is a system designed to distribute the rendering across a cluster of servers, leveraging the power of multiple machines to decrease the render time by using the available hardware. DAS-BR is built in Python 3.6 and requires only a single package that facilitates the import of YAML files. It uses Blender 3.3 LTS as the Blender version, but is compatible with older versions of Blender. The source code of DAS-BR is publicly available on GitHub[30]. The DAS-BR architecture is divided into three operator types:

- Master, responsible for managing and maintaining the render farm.
- Worker, responsible for performing the render.
- Client, responsible for submitting operations to the cluster.

Activities within DAS-BR are separated into three categories:

- Operations: An operation is a cluster-wide activity requiring the labour of multiple workers. An operation is responsible for dividing the work among the cluster, managing the progress, and dealing with arising issues such as node failure. An example is a Blender render operation.
- Jobs: A job is a division of the operation that an individual worker performs. The jobs are tailor-made based on the performance of the client. An example is a Blender render job; the Blender render operation divides the work into the jobs that each of the nodes needs to perform.

- Tasks: A Task is the activity that a worker does to finish a job. In the example of a Blender render job, it is rendering frames and informing the operator about the progress.

4.2 Operation execution

Initially, a client submits a request to the master. The request will contain the location of the Blender project and render parameters. Based on the request, the master will initiate a new operation. The operation will manage the progress of the render. The new operation will guarantee that all worker nodes receive their assigned tasks and that any arising issues will be addressed. Each worker receives their new job, which consists of instructions for their designated frames. Based on their job instructions, the workers will start their tasks. A task is a separated process on the node meant to be standalone from the DAS-BR worker node system. During the duration of the task process, the task periodically informs the worker node of its progress. The worker informs the operation of its progress, which in turn lets the user know the progress of the render. When the task is complete, the worker will inform the operation. If the operation has received a message indicating the completion of the task from all the workers, the operator will proceed to assemble all the frames into a video. The figure 4.1 provides a detailed description of this procedure in a diagram.

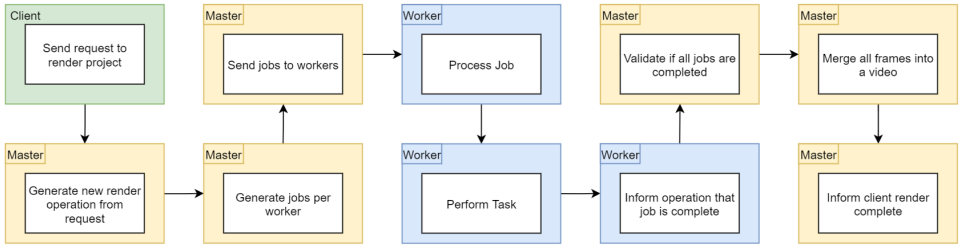


Figure 4.1: Flowchart detailing system activity flow

4.3 Features

This section highlights the capabilities of the system. We do this based on the two main criteria: scalability, and performance. The system contains other features, such as fault tolerance, that will not be discussed, as those features are not intrinsically linked to rendering.

4.3.1 Scalability

Scalability is at the core of DAS-BR. To utilize a variable amount of worker nodes, DAS-BR needs to be scalable. We have implemented the following features under the guise of scalability:

Node registrations

All that is required to add a node to the render farm is to start the worker daemon and give it the address and port of the master and the worker's address and port. The worker daemon will register with the master. The master will return a worker ID to differentiate between worker IDs.

Dynamic cluster size

As previously mentioned, node registrations also allow the cluster to freely scale, this means that adding or removing worker nodes can be done at any time with minimal consequences.

Operation queue

The master will automatically place all incoming messages in a queue. By using a queue, messages will not prevent the master node from performing important jobs, such as checking the cluster's status. The operation queue also allows the master to operate with multiple workers simultaneously because the master cannot become overwhelmed by nodes.

4.4 Performance considerations

As Blender renders are computationally intensive, performance has been an important consideration for DAS-BR. The following features have been implemented to improve the performance:

Dynamic performance-based scheduling

Each worker node has its hardware capabilities. A node may have an NVIDIA GPU, an AMD GPU, or no dedicated GPU at all. To accommodate for this variable, the workers will be benchmarked at initialization. The benchmark score will be used to measure the performance of the worker and how it compares to other workers in the cluster. The master will assign a set of frames to each worker based on the benchmark score when a new render operation starts. This means that which worker does which frame is established before any frame is rendered. For the remainder of this thesis, this approach of fixed batches of frames is referred to as the batched scheduling approach. There is not only one way to approach scheduling; an alternative scheduling approach is a task queue.

Task queue

The queued approach, instead of dividing frames in advance, as is the case with the batched approach, instructs workers to query the master and ask for a frame to be assigned. A worker would request frames to render, render the frames, and request more frames until the queue has been cleared. A task queue would spread out the workload efficiently because it allows workers to render as many frames as they can without needing to predict the performance. Flamenco uses a queued approach instead of a batched approach. Nonetheless, DAS-BR explicitly disapproves of this scheduling approach. In the early experimenting of Blender, we discovered that Blender spends a considerable amount of time initializing. The duration of initialization is influenced by the complexity of the Blender scene and the memory specifications (speed and capacity) of the graphics card. The Blender render engine necessitates knowledge of the frames to render before initialization, and such information cannot be altered once initialized. It is possible that initialization will take longer than rendering the frame. This makes it undesirable to have to restart the Blender render engine for every new frame. Based on the slowdown introduced by the initialisation delay, using the task queue could result in longer render times. Based on the conclusion that task queues would result in longer render times, we made the decision for a batched approach. The batched approach means that all frames are distributed at the time of initialization. As all the frames can be rendered without the need for repeated initialization, the initialization delay should apply only once. To assess the optimal scheduling approach, two experiments have been designated for benchmarking the scheduling approaches.

Multiprocessing

As the project was made in Python, we had to find a way around Python's Global Interpreter Lock(GIL). The DAS-BR was built with multiprocessing in mind from the very start. To ensure that no TCP/IP packets are dropped, Python Sockets are handled in a separate process. The rendering phase is handled separately to avoid any interference from the worker-daemon. The worker-daemon on the main process is only there to route packets, ensuring that all separate processes can execute their work. The multiprocessing strategy was devised to safeguard against a potential bottleneck that could negatively affect performance.

Dynamic Render Engine

To speed up renders, we are using hardware-based acceleration to utilize the hardware as much as we can. By doing this, we can decrease the render time by making use of NVIDIA OptiX™ or NVIDIA CUDA®. There can be a combination of GPU acceleration frameworks, so the entire cluster does not need to adhere to a single framework. By allowing a combination of GPU acceleration frameworks, we are ensuring that the heterogeneous nature of DAS-BR does not decrease performance. Furthermore, using hardware-based acceleration ensures that we take advantage of the hardware features available.

Chapter 5

Experiments and Results

The first section discusses the experimental setting, then the experiments, their results, and their analysis. All the experiments are centred around a rendering operation that aims to provide insight into the performance in a specific circumstance. For the rendering project, two projects were selected.

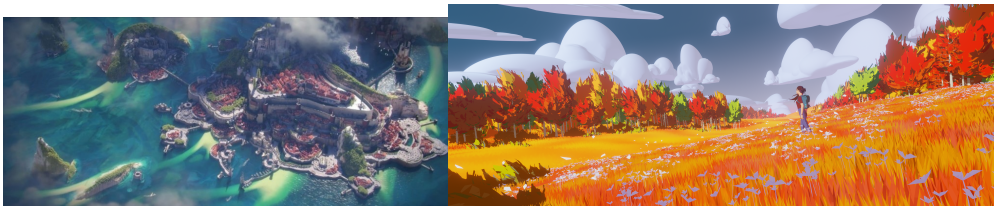
5.1 Projects

5.1.1 Scans Island

Scans Island is a project by art director and concept artist Piotr Krynski. The project is used within Blender 3.4 as a start-up splash image and has been made publicly available. This project has been altered to include a camera that flies around the island. This camera was needed to turn the splash image into an animation video. Scans Island consists of 187867 objects resulting in a vertices count of 2763097.

5.1.2 Red Autumn Forest

Red Autumn Forest is a project by Senior Concept Artist Robin Tran. The project was the Blender start-up splash screen for version 2.91 released in November 2020 and has been made publicly available. Red Autumn Forest has 2903587 vertices spread out among 1291 objects. No modifications were made for this project.



(a) Scans Island **(b)** Red Autumn Forest

Figure 5.1: Blender projects used for experiments

These two projects were chosen for two reasons:

- Professional Blender users made them. Creating complex and large projects requires skilled users.
- They are graphically demanding enough to put a strain on the most powerful hardware currently available.

5.2 Experiment context

5.2.1 Experiment procedure

To analyse DAS-BR, five experiments were conducted, and each experiment was selected to evaluate a distinct aspect of DAS-BR. To prevent mistakes, outliers, and flukes, every experiment was run for five times on the same hardware without other experiments running at the same time. Experiment 5 was performed on the LIACS cluster, and all other experiments were performed on the VU cluster.

5.2.2 Experiment metric

The outcome of an experiment is expressed in total seconds. The seconds signify the amount of time the render took from the point of distribution to the time at which all frames were rendered. This means that less time rendered is better. The duration is shown on the y-axis of the graphs. Even though DAS-BR is capable of assembling the frames into a single video, this post-rendering procedure was disabled for all experiments. The method of assembling the videos is the same, even when it comes to Flamenco, as it involves merely contacting the multimedia library FFmpeg to accomplish this task. The video assembly has therefore been omitted from the results. Each experiment was done on multiple servers. For instance, a one node test means, one worker and one master and one client. A four node test means four workers, one master and one client. The number of workers is shown on the x-axis of the graphs.

5.2.3 Experiment renders resolutions

For the experiments, three resolutions were chosen:

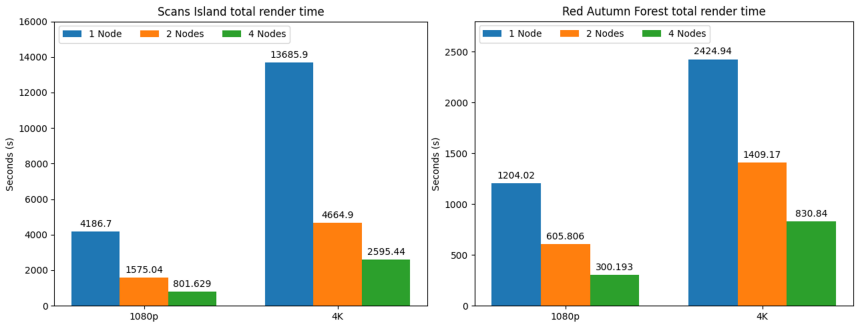
- 1280 by 720 pixels (HD)
- 1920 by 1080 pixels (Full HD)
- 3260 by 2160 pixels (4K)

These resolutions were chosen as these are common and standard ((F)HD [31] and UHD TV [32]) resolutions for TV on which animations are watched. All experiments consisted of rendering 120 frames of their designated project at 24 frames per second.

5.3 Experiment 1: Scaling

The scaling experiment is designed to analyse how the system performs when we add more nodes and what the impact is of adding more nodes to the rendering cluster. Scaling will be evaluated by running both projects (Scans Island and Red Autumn Forrest) with one-worker, two-worker, and four-worker modes.

5.3.1 Results



(a) Scans Island total render time **(b)** Red Autumn Forest total render time

Figure 5.2: Scaling test results

5.3.2 Analysis

As can be seen in Figure 5.2, scaling to more nodes reduces the overall render time. Red Autumn Forest exhibits a significantly lower render time in comparison to the Scans Island project, due to the lower object count. The scaling in both projects has a variance and does not scale perfectly. This disparity can be attributed to the difference in rendering difficult per frame. To illustrate the disparity, the time to render a frame was recorded. Figure 5.3 shows the result of the render disparity. With this disparity in mind, a batched approach will prevent a perfectly equal division of labour, as it assumes the workload is equal. This is less noticeable at lower resolutions, such as 720p, but it starts to become noticeable at 1080p and 4K.

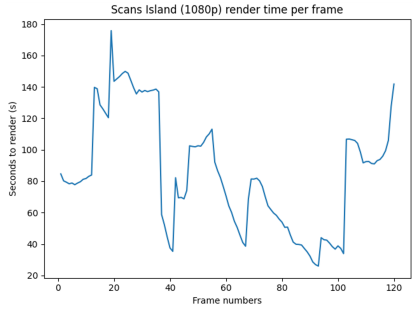


Figure 5.3: Render time variance per frame

5.4 Experiment 2: Heterogeneous render

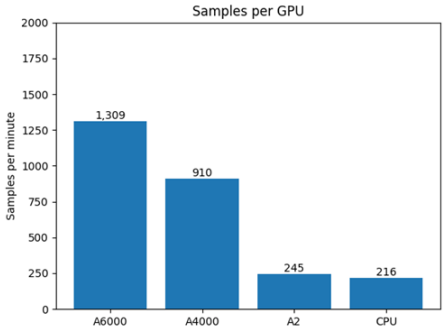
In the heterogeneous experiment, we will mix hardware configuration to highlight how the system performs in a heterogeneous configuration. To evaluate this, we will use four nodes with varying GPUs and render the Scans Island project in 4K. The goal of this experiment is to answer research question two: How can Blender renders efficiently be scaled across heterogeneous hardware? To divide the work across the nodes, DAS-BR uses the results of a Blender benchmark to calculate the number of samples a GPU can process. Samples are an indicative measure established by Blender to quantify the amount of computation the hardware is capable of in a limited time. The GPUs used for each node configuration can be found in Table 5.1.

Node configuration	Hardware
1 Node	1x A6000
2 Nodes	1x A6000, 1x A4000
4 Nodes	1x A6000, 1x A4000, 1x A2, 1x AMD EPYC 7402P

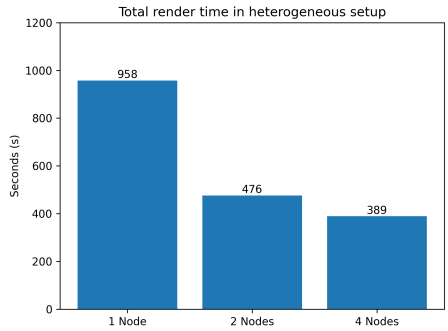
Table 5.1: Hardware used per node for distribution experiment

5.4.1 Results

The experiment results consists of three graphs detailing how the workload has been divided. Figure 5.4a shows the number of samples each GPU can perform in a minute. Figure 5.4b details the comparative render times, and Figure 5.5 details how the work had been divided for each experiment.



(a) Samples per GPU per minute



(b) Total render time in the heterogenous experiment setups

Figure 5.4: Samples per GPU and the distribution of algorithm

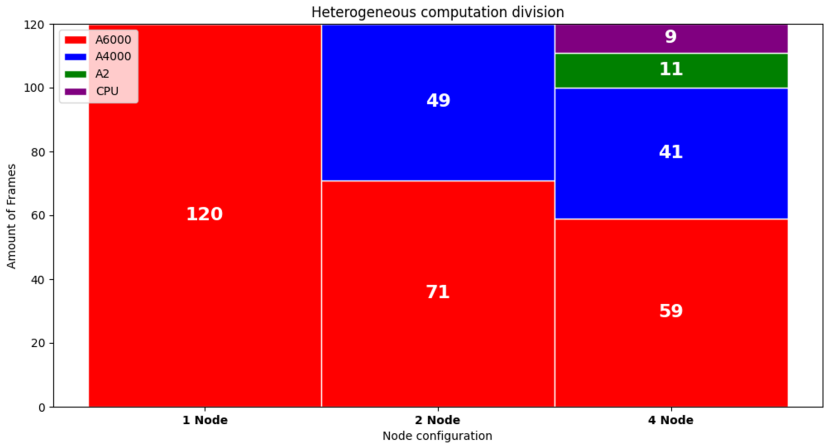


Figure 5.5: Distribution algorithm

5.4.2 Analysis

The distribution algorithm utilizes the available resources based on the samples per minute score, thereby enabling heterogeneous hardware to contribute to the rendering process. The distribution algorithm considers the render capacity of a node, compares it to the total render capacity, and then determines the share of the render operation a node will need to perform. The findings demonstrate that DAS-BR possesses the capability to adapt to a heterogeneous cluster and effectively utilize hardware resources to reduce render time.

5.5 Experiment 3: GPU vs CPU

The objective of experiment three is to determine the impact of different component-based rendering. We searched for a paper detailing the performance difference of the Blender render engine using NVIDIA enterprise hardware to establish a baseline of comparative performance. Unfortunately, we were unable to find any reference to this. By executing this experiment, we hope to answer research question 3, the impact of GPU acceleration on the render time. The experiment was performed on nodes containing an NVIDIA A4000 GPU and an AMD EPYC 7402P CPU whilst rendering the Red Autumn Forest scene at 720P.

5.5.1 Results

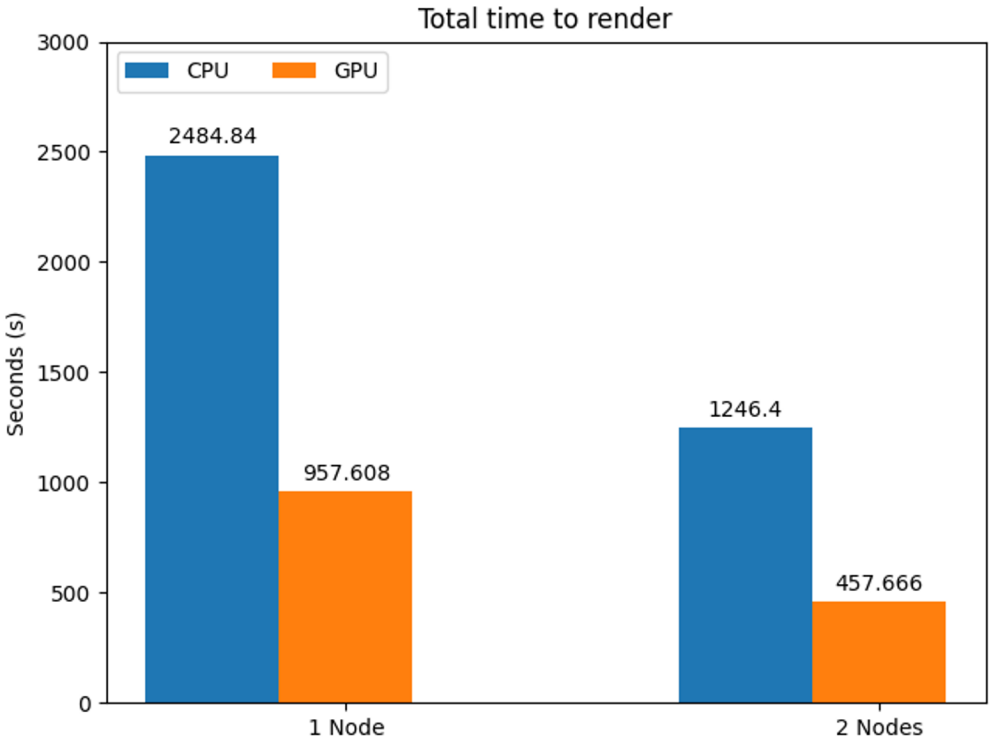


Figure 5.6: Total render time GPU vs CPU

5.5.2 Analysis

The findings clearly demonstrate that the utilization of a GPU reduces the rendering time. The scaling of the CPU is more linear than the GPU. Repeated experimenting showed the same pattern. The exact cause for this remains difficult to determine. The CPUs may experience a higher degree of bottle necking in memory access, resulting in a more linear scaling performance. Establishing the exact cause requires a more in-depth investigation.

5.6 Experiment 4: Batched vs Queued scheduling

As previously mentioned, DAS-BR uses a batched approach, where batches of frames are distributed among the nodes. The assumption was made that rendering would benefit from rendering a single continuous sequence of frames, thus preventing the initial Blender render engine loading from increasing the render time. In this experiment, we will apply the queued scheduling, where each node will ask for an individual frame, to evaluate whether the previous assumption is justified. Together with experiment two, this experiment aims to address research question 1 concerning how effectively and efficiently renderings can be scaled. The experiment was done on nodes with an NVIDIA A4000 GPU and an AMD EPYC 7402P CPU and running the Red Autumn Forest scene at 720P.

5.6.1 Results

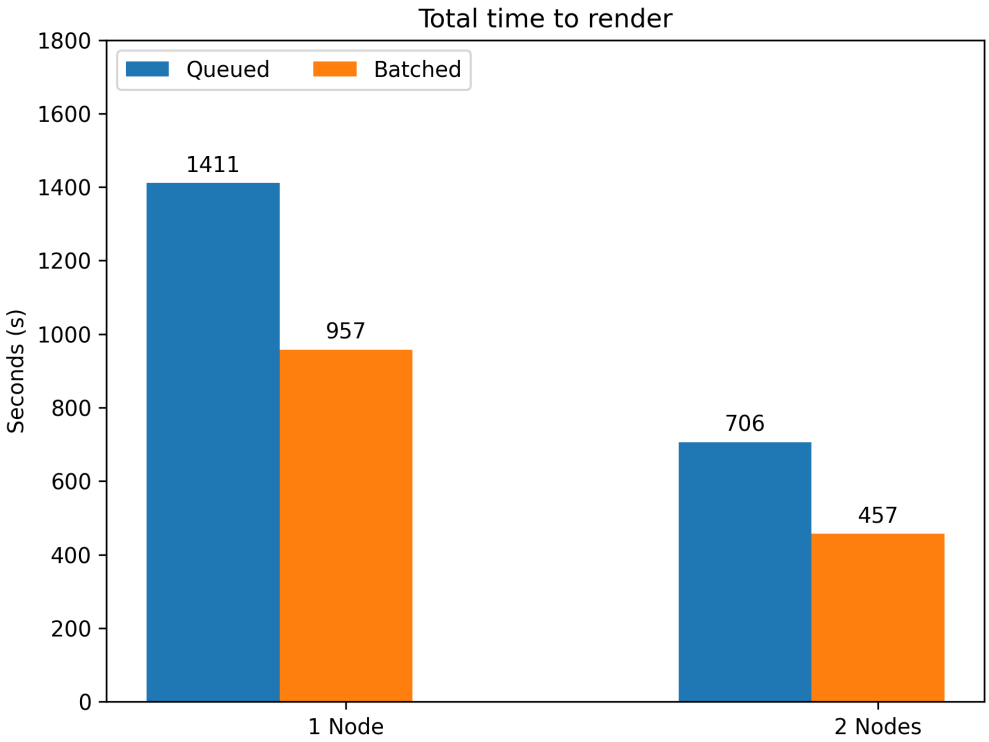


Figure 5.7: Total render time batched vs queued scheduling

5.6.2 Analysis

As demonstrated, there exists a distinct advantage to utilizing the batched approach. Compared to the batched approach for a single node, the queueing is 47% slower, the slowdown is even larger when it comes to the two-node setup where the slowdown is 54%. Based on these results, we believe that using a batched approach

is a more efficient way to schedule render jobs in Blender compared to the queued approach.

During the experiment, we observed a benefit of the queuing approach. It appears that reinitializing the same project after a render was completed is faster than the first initialization. This is likely due to caching on the GPU side. This means that the original assumption that repeatedly restarting the render pipeline was less severe than expected.

Another observation of the queued scheduler was that the utilization of the cluster was higher. Previous experiments have shown that at the end of the render, a situation occurs in which only a single node is actively rendering while the rest of the cluster is idle. This did not occur during the queuing approach. However, this situation only presents itself because the nodes spend a considerable time performing computations that were not necessary for the batching approach.

In the future, repeating the experiment with more nodes might be worth it. Future GPUs might have faster memory, which reduces or eliminates the start-up time. The queued approach allows for a more natural distribution based on actual performance rather than hypothetical performance expectations of the batched approach.

5.7 Experiment 5: Flamenco VS DAS-BR

To compare Flamenco with DAS-BR, an experiment was created that looks at their performance on identical hardware.As mentioned in Chapter 3, Flamenco was unable to run with GPUs; therefore the experiment was performed with only CPU nodes that contained the AMD EPYC 7402P CPU. As render project, the Red Autumn Forest scene at 720P was picked. In Chapter 4 we mentioned that DAS-BR uses a batched approach while Flamenco uses a queued approach. In this experiment, we compare the queued and batched scheduler of DAS-BR to Flamenco's queued scheduler.

5.7.1 Results

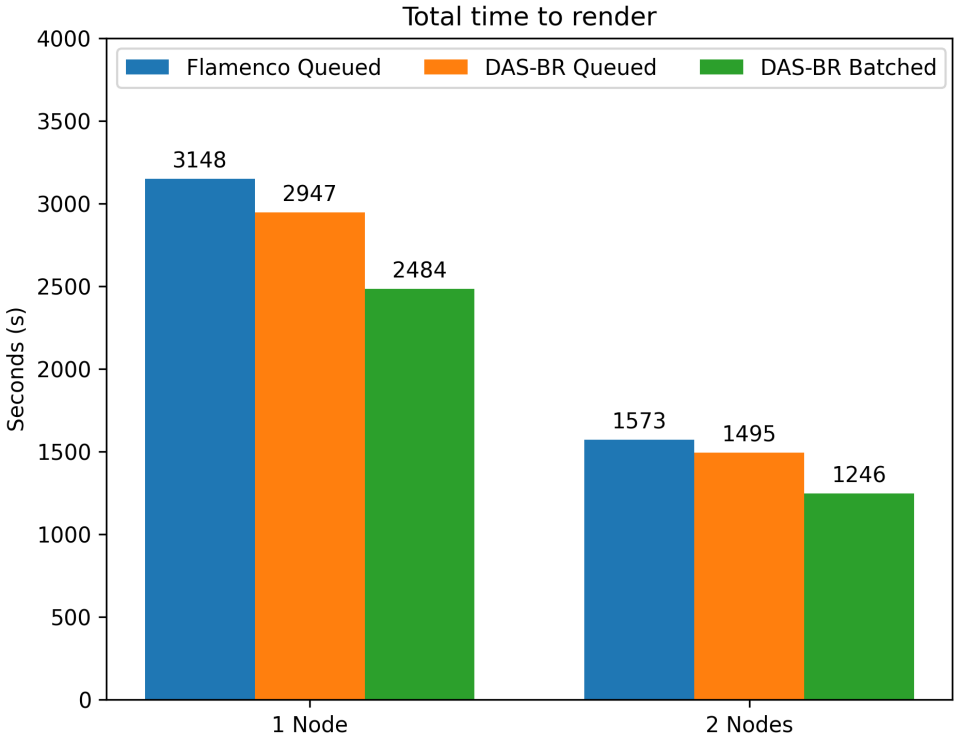


Figure 5.8: Total Flamenco vs DAS-BR render times

5.7.2 Analysis

As Figure 5.8 shows, there is a clear advantage to using DAS over Flamenco when looking purely for the shortest render time. Even though Flamenco is slower than DAS-BR, this might be a trade-off that would be worth it for people running Flamenco outside DAS6. The overhead of Flamenco is contributed to the other functionalities of Flamenco. The added functionality of the storage manager and post-processing nodes allows users to move, edit and post-process (for example, adding audio) while the remainder of the render is still being done or the dashboard providing a clearer

view of the running job. These are features that DAS-BR does not contain.

Compared to the GPU-based scheduling experiment detailed in Figure 5.7, the difference in render time between scheduling algorithms is comparatively smaller when performed using CPUs. This can be contributed to the smaller impact of the initial loading time. The initial loading time represents a larger part of the GPU render cycle, since GPU's render frames faster, whereas CPUs are comparatively slower, which means that the initial loading time represents a smaller part of the render cycle.

A future experiment could prototype a batched approach, or a queued-batched approach, into Flamenco to see how it impacts the overall render time and what bottlenecks might be introduced regarding the functionalities that provided the overhead in the experiment. Another idea for a future experiment would be to use GPU's. As previously mentioned, GPU's were unable to work on DAS6 at the time of running, but this might not be the case in the future.

Chapter 6

Discussion

6.1 Environment analysis

As mentioned in the background chapter, the environment of the system can play a significant role in the computing power of the system. To determine the quality of the results, a considerable amount of time has been spent on analysing the environmental performance. During an earlier evaluation of DAS-BR, DAS system performance was evaluated on the cluster of the technical university of Delft (TUD). In the earlier evaluation, the nodes were evaluated by using the Blender benchmark in a sequential, meaning node after node, and a parallel, meaning all nodes at once, order. Upon analysis of the results, the presence of thermal throttling was apparent. There was a noticeable decrease in performance when running all the systems at the same time. The results of this evaluation can be seen in Figure 6.1.

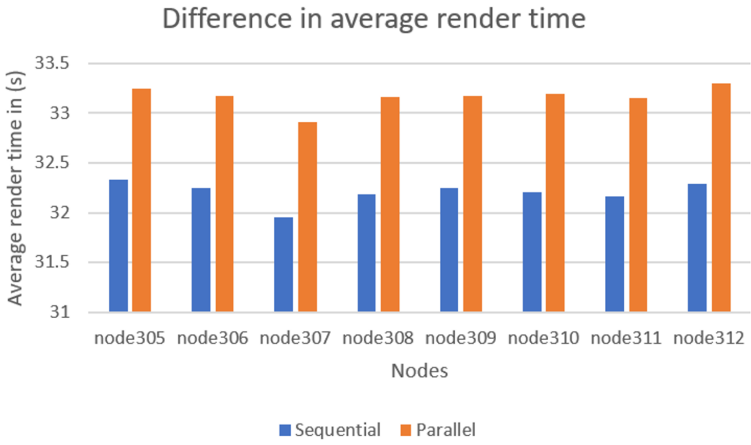


Figure 6.1: Difference in sequential versus parallel render times on TUD

For this thesis, we used the DAS cluster of the VU. TUD was occupied by other researchers and lacked different GPUs for conducting the heterogeneous experiments. As the TUD cluster had thermal throttling issues, we requested access to the NVIDIA System Management Interface (NVSMI). NVSMI is a command line utility intended to aid in the management and monitoring of NVIDIA GPU devices [33]. With NVSMI we can observe the clock speeds, temperature, utilization and throttling issues. The goal of this analysis was to detail the performance of the environment in a separate research sub-question and analyse the impact it had on the results.

After performing the same experiments as done during the earlier evaluation, we discovered throttling. In contrast to TUD, the throttling on VU was not thermal throttling, but power throttling. DAS6 VU prevents GPUs from going above the TDP. This meant that the GPUs could not draw more power and were unable to boost higher. The advantage of limiting the available power is lower power usage and better thermals. As previously mentioned, running hardware at a lower temperature will increase the lifespan. This also meant that the node-to-node performance was more similar in comparison to the earlier evaluation.

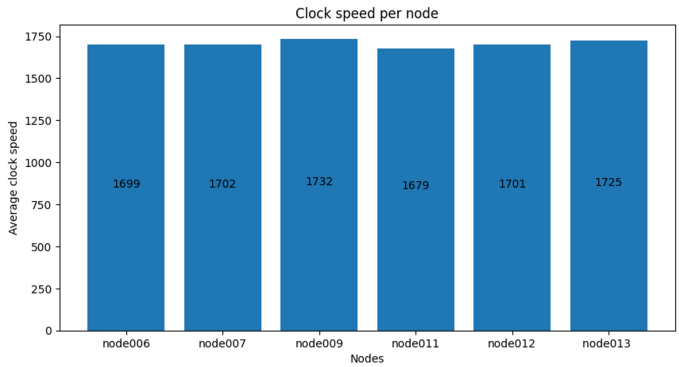


Figure 6.2: Difference in clock speeds

Figure 6.2 shows that the largest difference in average clock speed is between node009 and node011. The difference is 3.1%. While this might sound like a notable difference, in terms of render performance, the sample difference is 983 for node009 compared to 982 for node011. This means that for rendering purposes, there is no practical difference. As a follow-up experiment, we investigated bypassing the TDP limit. At the time of writing, there are three ways of bypassing these limits.

1. Increasing the power limit in software. This is the easiest option, but it requires the user to manually set the frequencies of the clock. This effectively disables Auto-OC, as the clock is manually supplied instead of the card reaching the maximum clock dynamically. The challenge with this approach is that it allows the user to enter an unstable clock, which causes it to crash. Furthermore, it only allows clock speeds that NVIDIA designates as acceptable.

2. Shunt modding. The most difficult approach. The practice of de-soldering and removing resistors that prevent the power limit protection from triggering. This approach tricks the GPU into thinking it is drawing 140W while being able to draw all the wattage the system is capable of supplying. As the NVIDIA A4000 uses a 6-pin ATX cable, which can supply 75W, and uses power from the PCI-E slot, which is capable of 75W, the remaining 10W could potentially be made available. However, the GPU will likely draw more than 150W, causing the system to crash. The NVIDIA A6000 can draw more power and a higher available power budget, but these were usually occupied by other researchers. It is important to note that soldering the GPU will void the warranty.

3. BIOS-flashing. A BIOS is the Basic Input Output System (BIOS) of a graphics card or the integrated graphics controller in a computer [21]. There is no need to trick the GPU into using more power as with the previous approach, as the BIOS will allow the system to clock higher. There are challenges with this approach. A BIOS must originate from NVIDIA or its partners. They do not wish for people to perform these types of operations, which makes it unlikely to receive one. It appears the most efficient way of getting a BIOS is asking around on overclocking forums, and one might magically appear in your inbox.

As is hopefully clear, there is no easy and safe way of removing this limit. Furthermore, there is no practical use for it for the evaluation of DAS-BR at the current time because the improvements would be negligible, and we are more interested in the scaling of the system than the render speed of the experiment environment. In a follow-up study, in which we have access to more powerful hardware or a more strained cooling environment, this should certainly be investigated. The potential for varying clock frequencies and by extension the experiment reproducibility could differ. To conclude, the environment evaluation, with the low variance in clock speed and the intentional nature of the throttling, we feel confident that throttling had no meaningful impact on the experiment results.

6.2 8K experiments

We attempted to use higher resolutions, such as 7680 by 4320. The system functions as expected, but we were running into environmental limitations. During the preliminary experimenting, we encountered excessively long render times. The average render time for a frame at 8K was 39 minutes, this was merely the time to render a frame, not including initialisation. As DAS6 has limits on how long users can reserve slots, rendering the entire video would not have been possible. Furthermore, doing so would have been a waste of DAS-6 resources and quite frankly the electricity necessary to run this. We could have changed the experiment variables to make 8K doable within our constraints. This could have been accomplished by reducing the length of the video or the frame rate. We decided not to evaluate 8K because the main issue was not the DAS-BR, it was the limitations of DAS-6.

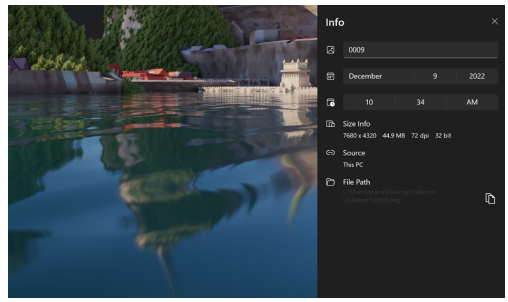


Figure 6.3: Screenshot of 8K render, frame number 9

6.3 Blender benchmark

At the start of the project, the Blender benchmark seemed as a reasonably accurate way to measure and compare the performance of each GPU. During the experimentation phase, a fundamental flaw was discovered with the benchmark. The benchmark ignores the set-up time of a benchmark. We discovered that the time to load the kernel was affected by the amount of video memory. More video random access memory (VRAM) meant increased loading times. With the NVIDIA A100, a GPU with 40 GB of video memory, this meant an excessively loading time. Curiously enough, this played no role in the resulting score. Where the CPU-based (AMD EPYC 7402 24-Core) rendering experiment received a lower score than the A100, it completed the experiment in less than a third of the time. This means that the resulting score of the A100 is not comparable to other GPUs. There is no easy solution to this problem. The benchmark could be updated to feature a score based on the set-up time, with the side effect of invalidating previous results of the benchmark database.

6.4 Project variety

Even though great care was taken in selecting projects, an unfortunate bias was detected during the analysis phase. Particularly when it comes to Nvidia Optix and Nvidia CUDA. Both are technologies are used for GPU-based acceleration. Optix is specifically a render-oriented GPU acceleration. Optix makes use of Nvidia's propriety ray tracing cores. Under normal circumstances, these cores should improve their performance when performing light-based operations such as reflections. Optix is intended to be significantly faster but takes a considerable amount of time to set up. When rendering a new or modified project, it loads a render kernel. In our experiments, we experienced upwards of 15 minutes for the Blender benchmark. The kernel loading duration is not affected by other factors, such as the number of frames that need to be rendered. The delay can cause short renders to increase dramatically in render time compared to CUDA. For a comparison in CUDA and Optix render time, we performed an experiment shown in Figure 6.4

A future study should feature a more diverse batch of renders. The initial aim was to get projects made by professionals who are skilled with Blender. A better alternative would be to gather a variety of projects as possible, made by a diverse group of authors. This approach would make the experiments more representative of the Blender user base and the (potential) DAS-BR user base. A future scheduler could also take these measures into account. If a user wants to render a few frames, it is not worth it to enable Optix due to the kernel loading delay.

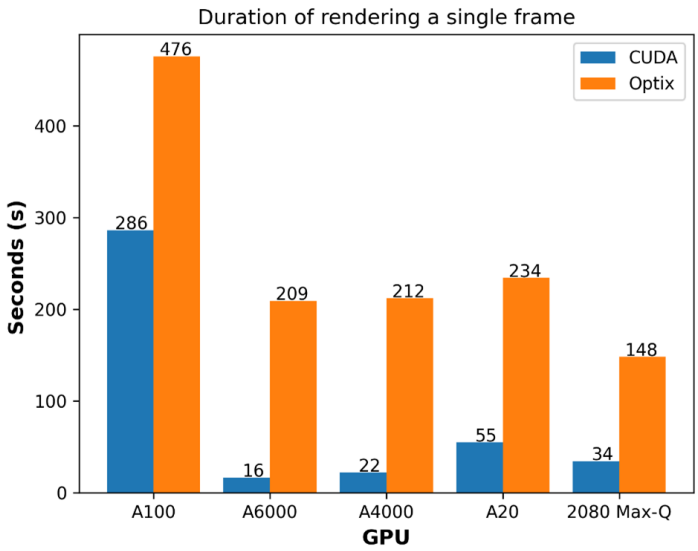


Figure 6.4: Render time per GPU

Chapter 7

Future work

In the previous two chapters, we discussed the experiments, the results and noteworthy findings. In this chapter, we propose a series of follow-up experiments and guiding principles. The goal of these propositions is to inspire follow-up research, address how we would have solved certain shortcomings, and give suggestions based on our experience.

7.1 Single machine multi-GPU compatibility

DAS-BR scales across servers with varying hardware to utilize the hardware available. One type of system that DAS-BR does not utilize is a single system with multiple GPUs. Evaluating the performance of a single machine with a multi-GPU configuration could pose some unique challenges. It could introduce bottlenecks such as IO limits of the storage cluster [34]. Other than a system with multiple GPUs, there are also systems with interconnected GPUs by technologies such as Nvidia's NVLINK/NVSwitch. It would be interesting to analyse the performance of these systems as well.

7.2 Batched-Queued scheduler

In experiment 1, 4 and 5 we mentioned the impact of the scheduler and highlighted the flaws present. In situations where the same number of frames must be rendered, the Blender render engine prefers a single extended render over several short renders. The single extended render prevents repeated initialisation, which causes a slowdown. For a distributed render, if the amount of computation per frame is not equal, the batched approach causes underutilisation. The underutilisation is caused by the imperfect division of the total computation power, as not all frames require the same amount of computation. The queued approach does not have this specific kind of underutilisation. A future study could therefore introduce a combination of both schedulers, a batched-queued scheduler. It would be interesting to see if it is possible to prevent the batched underutilisation, but also limit the queued initialisation time.

7.3 Power efficiency scheduler

The node with the highest power draw in our system has a 400W Nvidia A6000 GPU and a 225W Intel Xeon Gold 6342 CPU, which means the nodes can use over 625W on GPU and CPU. It would be interesting to develop a power efficiency scheduler that would schedule frames based on the power efficiency of each node. For instance, a user may specify the desired render outcomes, and an eco-friendly scheduler will allocate the compute task in a manner that prioritizes the power-efficient nodes for rendering.

7.4 Continuous node integration

DAS-BR does not redistribute a running render job when a new node is introduced into the network. This means that a new node is waiting until the current render job is finished. This causes an underutilisation of available hardware. This could be implemented in the future, but within the scope of the thesis it was of no importance; hence it was postponed.

7.5 Higher resolution renders

As previously mentioned, higher resolution renders such as 8K resolution were not conducted. The primary reason was the long render times. This does not mean this pain point cannot be alleviated. Using a less complex scene, adding more nodes to the cluster or reserving the hardware for a long time are all solutions to the problem. The main reason to do a higher resolution render is to see the impact of a GPU-based render where the VRAM is not sufficiently large. In such a situation, there will be an increased level of storage access, which will cause a slowdown across the cluster. It is currently unclear how realistic such a situation is, as modern GPUs can have 24 GB, 48 GB or even 80 GB of VRAM.

7.6 Security

DAS-BR was not made with security in mind. It is simple to spoof messages, perform man-in-the-middle attacks, or turn DAS-BR into a botnet. DAS6 is blocked off from the wider internet and requires a university gateway to access, this makes the network isolated. A future revision of DAS-BR meant for a public release should certainly contain more security measures such as encrypted traffic, credentials for nodes and tighter package limitations.

7.7 Predictive scheduling

Even though two ways of scheduling have been presented, neither is perfect. In an ideal world, DAS-BR would know how computationally intensive each frame is that needs to be rendered and allocate work based on this computational intensity. Unfortunately, Blender lacks a function that returns this information. Regardless of the feature set of Blender, there are ways we can improve the scheduling. A solution would be to analyse the system resources the render engine uses to estimate when a particular frame is complex or is becoming complex. By using the frame rate of the animation, DAS-BR could take an educated guess to predict which frames will be complex. This solution does have its weaknesses and its effectiveness is difficult to predict. Another solution would be to do a post-mortem on render operations. This

would aid in the improvement of subsequent operations of the same variety. Imagine a project in which renders are done frequently, and only minor changes are made to the project. In such a situation, DAS-BR could use a predictive frame distribution model to optimize render time. Once again, how applicable, and effective this would be in practice remains difficult to predict.

7.8 GPU-Hardware choices

DAS-6 was chosen out of necessity, it was the only freely available high-performance computing cluster. DAS-6 was built with Nvidia enterprise GPUs that are built for data centres. For example, the NVIDIA A100 was developed with data science and artificial intelligence applications in mind, and excels at large data set interaction and tensor-based computation, not rendering. The Nvidia A100 has a 10,000 EURO manufacturer-suggested retail price (MSRP) [35], but it is outperformed by consumer GPUs of less than 500 Euros in rendering tasks[1]. A future study might best be performed with consumer GPUs for a more affordable and cost-benefit-based cluster representative of a potential user.
Unfortunately, NVIDIA puts users in a bind when it comes to rendering. Using Consumer GPUs in data centres is prohibited by NVIDIA's end-user licence agreement [37]. A future study might prefer to use hardware from a different manufacturer or acquire many workstations with NVIDIA GPUs rather than a data centre setup. As mentioned in the discussion chapter, there are ways of improving the performance by using less orthodox methods that we were unable to perform. If a follow-up study does not use DAS, it might be interesting to attempt of these methods for optimising performance.

[1]For example, when performing the Blender benchmark in CUDA, it is outperformed by a 2060 SUPER by 21.8% [36]. The 2060 SUPER has an average score of 1374.82 while the A100 has a score of 1075.96 in CUDA. The 2060 SUPER is a budget GPU with an MSRP of 414 EURO.

Chapter 8

Conclusions

To render the evermore complex scenes, the need for powerful hardware and the utilization of the hardware has become a pressing matter. As a single computer can only become so powerful, there is a need for distributed rendering. There is currently no effective and convenient way of performing distributed Blender renders on small-scale computer clusters, this thesis set out to solve this problem by creating DAS-BR. The thesis started by introducing the research questions, followed by detailing key concepts for the thesis, such as component specifications and the DAS. The background was followed by a review of existing works in which we brought attention to what this thesis aims to present new work, particularly when it comes to heterogeneous rendering, GPU usage, and environmental awareness. After the existing works, we discussed the experiments, followed by showing the empirical results, discussing research findings and proposing follow-up research. Based on the experience and results, we can now answer the research questions.

> *RQ1: How can Blender renders efficiently be scaled across a distributed computing cluster?*

The architecture presented in DAS-BR has shown its efficacy. Render times have improved significantly, for every doubling of comparable hardware the render time halves. When looking at the results of experiment four, the overhead of DAS-BR is minimal.

> *RQ2: What effect does GPU acceleration have on render times compared to CPU?*

As shown in the results of experiment three, the difference is significant. GPUs are at a serious advantage when it comes to rendering times. GPUs spend less time computing light and waiting on memory operations. The impact is so noteworthy that modern GPUs regardless of make and model are likely to be advantageous and users are encouraged to use them if possible.

> *RQ3: How can Blender renders efficiently be scaled across heterogeneous hardware?*

Heterogeneous scaling based on a benchmark has shown its benefits. Utilizing hardware, regardless of power, has an impact on lowering the overall render duration. With those questions answered, we can now comfortable answer the question central to the usability and applicability of DAS-BR:

> *RQ4: What are the benefits of a distributed render compared to Blenders native rendering?*

DAS-BR is an effective way to utilize the available hardware (CPU and GPU) and decrease the total wait time for rendering by utilising both heterogenous and homogenous systems.

Bibliography

[1] M. Henne et al. "The making of Toy Story [computer animation]". In: *COMPCON '96. Technologies for the Information Superhighway Digest of Papers*. 1996, pp. 463–468. DOI: 10.1109/CMPCON.1996.501812.

[2] Muhammad Daim and Muhammad Talha. *How long did it take to render toy story? quick answer*. Jan. 2023. URL: https://www.technologitouch.com/tech-tips/how-long-did-it-take-to-render-toy-story/.

[3] D.T. Ross and Massachusetts Institute of Technology. Electronic Systems Laboratory. *Computer-aided Design: a Statement of Objectives*. Technical memorandum 8436. M.I.T. Electronic Systems Laboratory, 1960. URL: https://books.google.nl/books?id=4IquHAAACAAJ.

[4] Dispatch. *Understanding the magic of 3D rendering*. Jan. 2020. URL: https://wp.nyu.edu/dispatch/2019/08/16/understanding-the-magic-of-3d-rendering/.

[5] Sai Arun. *Why 3D rendering is important?* Sept. 2020. URL: https://timesofindia.indiatimes.com/readersblog/vidiyum-munn/why-3d-rendering-is-important-26148/.

[6] Travis Highlander. en. Jan. 2019. URL: https://www.motortrend.com/features/1901-rendering-of-your-car-important-part-build/.

[7] Carsten Dachsbacher and Marc Stamminger. "Reflective shadow maps". In: Apr. 2005. DOI: 10.1145/1053427.1053460.

[8] K. Fatahalian et al. *Evolving the Real-time Graphics Pipeline for Micropolygon Rendering*. Stanford University, 2010. URL: https://books.google.nl/books?id=vTF7nQAACAAJ.

[9] Blender Foundation. *Performance considerations*. URL: https://docs.blender.org/manual/en/2.79/render/blender_render/optimizations/performance.html.

[10] M. Z. Patoli et al. "An open source Grid based render farm for Blender 3D". In: *2009 IEEE/PES Power Systems Conference and Exposition*. 2009, pp. 1–6. DOI: 10.1109/PSCE.2009.4839978.

[11] Ganesh V. Patil and Santosh L. Deshpande. "Distributed rendering system for 3D animations with Blender". In: *2016 IEEE International Conference on Advances in Electronics, Communication and Computer Technology (ICAECCT)*. 2016, pp. 91–98. DOI: 10.1109/ICAECCT.2016.7942562.

[12] Ki-Seok Chung and TaeHoon Kim. "Benefits of the big . LITTLE Architecture". In: 2012.

[13] Florin Pop, Alexandru Iosup, and Radu Prodan. "HPS-HDS: High Performance Scheduling for Heterogeneous Distributed Systems". In: *Future Generation Computer Systems* 78 (2018), pp. 242–244. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2017.09.012. URL: https://www.sciencedirect.com/science/article/pii/S0167739X17319659.

[14] Henri Bal et al. "A Medium-Scale Distributed System for Computer Science Research: Infrastructure for the Long Term". English. In: *Computer (New York)* 49.5 (May 2016), pp. 54–63. ISSN: 0018-9162. DOI: 10.1109/mc.2016.127.

[15] URL: https://www.cs.vu.nl/das6/sponsors.shtml.

[16] URL: https://cs.vu.nl/pub/das6/network.shtml.

[17] URL: https://www.cs.vu.nl/das6/achievements.shtml.

[18] Blender Foundation. *Open data*. URL: https://opendata.blender.org/.

[19] R.B. Thompson and B.F. Thompson. *PC Hardware in a Nutshell: A Desktop Quick Reference*. In a Nutshell (O'Reilly). O'Reilly Media, 2003. ISBN: 9780596552343. URL: https://books.google.nl/books?id=kG8LcWfruOAC.

[20] Steve Burke and Patrick Lathan. *AMD Ryzen TDP explained: Deep-dive on TDP Definitions andamp; what cooler manufacturers think*. URL: https://www.gamersnexus.net/guides/3525-amd-ryzen-tdp-explained-deep-dive-cooler-manufacturer-opinions.

[21] Intel. *https://www.intel.com/content/www/us/en/support/articles/000005749/graphics.html*. Oct. 2020. URL: https://www.intel.com/content/www/us/en/support/articles/000005749/graphics.html.

[22] Steve Coulton. *Increasing the lifespan and reliability of electrical components*. Dec. 2012. URL: https://www.automation.com/en-us/articles/2012-2/increasing-the-lifespan-and-reliability-of-electri.

[23] Brian Stone. *Why you should prolong GPU lifespan (and ways to do it right)*. May 2023. URL: https://www.computer.org/publications/tech-news/trends/why-you-should-prolong-gpu-lifespan.

[24] Howard. *What are server cooling technologies?: FS Community*. June 2022. URL: https://community.fs.com/blog/what-are-server-cooling-technologies.html.

[25] Luiz André Barroso, Jimmy Clidaras, and Urs Hölzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Second Edition*. 2013. URL: http://dx.doi.org/10.2200/S00516ED2V01Y201306CAC024.

[26] *Understanding the True Total Cost of Ownership of Water Cooling for Data Centers*. Vol. Volume 1: Thermal Management. International Electronic Packaging Technical Conference and Exhibition. V001T09A013. July 2015. DOI: 10.1115/IPACK2015-48152. eprint: https://asmedigitalcollection.asme.org/InterPACK/proceedings-pdf/InterPACK2015/56888/V001T09A013/2505200/v001t09a013-ipack2015-48152.pdf. URL: https://doi.org/10.1115/IPACK2015-48152.

[27] Nugroho Agung Pambudi et al. "The immersion cooling technology: Current and future development in energy saving". In: *Alexandria Engineering Journal* 61.12 (2022), pp. 9509–9527. ISSN: 1110-0168. DOI: https://doi.org/10.1016/j.aej.2022.02.059. URL: https://www.sciencedirect.com/science/article/pii/S1110016822001557.

[28] Blender Foundation. *About*. URL: https://flamenco.blender.org/about/.

[29] Tina Lee. *CPU vs. GPU Renderer: Which is Better?* Mar. 2023. URL: https://academyofanimatedart.com/cpu-vs-gpu-renderer-which-is-better/.

[30] *DAS-BR Github*. Brandon Kroes. URL: https://github.com/BrandonKroes/DAS-BR.

[31] European Union. "Implementing Directive 2005/32/EC of the European Parliament and of the Council with regard toandnbsp; ecodesign requirements for televisions". In: *Official Journalandnbsp; of the European Union* 51 (Dec. 2008).

[32] Frans De Jong. "New guidance on UHD/HDR service parameters for 2020+". In: *Testing times for the UHD transition* 0.41 (Aug. 2019), pp. 11–11.

[33] NVIDIA. July 2016. URL: https://developer.download.nvidia.com/compute/DCGM/docs/nvidia-smi-367.38.pdf.

[34] Brad Nemire. *https://developer.nvidia.com/blog/nvswitch-leveraging-nvlink-to-maximum-effect/*. Mar. 2018. URL: https://developer.nvidia.com/blog/nvswitch-leveraging-nvlink-to-maximum-effect/.

[35] Kif Leswing. *Meet the 10,000 Nvidia chip powering the race for A.I.* Feb. 2023. URL: https://www.cnbc.com/2023/02/23/nvidias-a100-is-the-10000-chip-powering-the-race-for-ai-.html.

[36] Julian Huijbregts. *Nvidia brengt GeForce RTX 2070 en 2060 Super op 9 juli uit*. July 2019. URL: https://tweakers.net/nieuws/154656/nvidia-brengt-geforce-rtx-2070-en-2060-super-op-9-juli-uit.html.

[37] *License For Customer Use of NVIDIA GeForce Software*. NVIDIA. URL: https://www.nvidia.com/en-us/drivers/geforce-license/.