Bachelor of Science in Computer Science
June 2020

DirectX12: A Resource Heap Type Copying Time Analysis

Simon Törnblom
Pontus Hellman

Faculty of Computing, Blekinge Institute of Technology, 371 79 Karlskrona, Sweden

This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfilment of the requirements for the degree of Bachelor of Science in Computer Science. The thesis is equivalent to 10 weeks of full time studies.

The authors declare that they are the sole authors of this thesis and that they have not used any sources other than those listed in the bibliography and identified as references. They further declare that they have not submitted this thesis at any other institution to obtain a degree.

Contact Information:
Author(s):
Simon Törnblom
E-mail: sito17@student.bth.se

Pontus Hellman
E-mail: pohe17@student.bth.se

University advisor:
Mats-Ola Landbris
DIDA

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

ABSTRACT

Background
The API DirectX 12 allows programmers to have more control over the GPUs memory management. This includes the ability to allocate resources on different types of memory heaps. But there is a lack of research on how these heap types affect the copying performance.
Objectives
The aim of this thesis is to benchmark the copying performance of the different heap types in DirectX 12 when increasing the data size. The heaps are tested with the three types of command queue that can be used to execute commands to the GPU.
Method
To answer our research question, a DirectX 12 prototype was implemented and used to copy increasing amount of data between different heap types. The copy operations were also combined with three different types of command queues to see if these have any impact on the performance. The tests ran on three different Nvidia graphic cards on the same computer setup, both to validate our results but also to spot any potential differences.
Results
The results from this study show that there is a difference in copying speed when copying data between resources that have been allocated on different heap types. The fastest to slowest were as follows: *Default to Default, Upload to Default / Default to Readback* and *Upload to Readback.* Using different types of command queues did not have an impact on performance with the exception of when data was copied from *Default to Default* on an RTX 2080. All of the tests that were carried out showed that the copying time scaled linearly with the data size.
Conclusion
This study shows the importance of allocating resources on the most suitable heap as there is a difference in copying time between them. In contrast, was the choice of command queue less important as this had no impact on performance in the majority of the tests. The results also show that the copying time scales linearly with the data size.

Keywords: DirectX 12, Memory management, Performance test

ACKNOWLEDGMENTS

We would like to thank our supervisor Mats-Ola Landbris that has given us a lot of feedback on our thesis and Stefan Peterson that has helped by lending us a RTX 2080 GPU and for giving feedback on our method. Lastly, would we like to thank our opponents Ala Jafar och Daniel Lagula who have given us a lot of feedback on our thesis, which has helped us improve our work.

CONTENTS

Chapter 1

INTRODUCTION

Graphics Processing Units (GPU) are used today in many different fields, where they are commonly used for their high computational power and image processing abilities. The main usage is for real time rendering applications as in games or simulations were images are refreshed and displayed on the screen in a high frequency. There is also the alternative to save the images, which opens up for more processing time for the image which in many cases can give a more graphical pleasant result for the viewer. This is commonly used when making or editing movies, both in the professional sector and for hobby enthusiasts. These are just some of the fields that use the GPU for its image processing power.

The second type of usage is comparable with the Central Processing Unit (CPU) where instructions on data are executed and stored on either system RAM (Random Access Memory) or VRAM (Video Random Access Memory). This is commonly used by a process that can be divided into many smaller tasks that can be executed in parallel. Many take advantage of the GPU to speed up the process when it would take a too long time for the CPU to process it. In this case, the GPU can be referenced as GP-GPU (General-purpose computing on graphics processing units).

Both types of usage usually store the processing data on the VRAM which can be referenced as resources. Resources are usually moved between system RAM and VRAM depending on the type of usage. Where the speed of the transfer is heavily influenced by the hardware setup. This study will go in more depth about how the resources should be managed when transferring them in memory and how it scales with data over different GPUs.

To manage resources on the GPU the communication goes through an API (Application programming interface). Some APIs are more focused on the graphical aspect like OpenGl, DirectX 11, Vulkan and DirectX 12 while others like CUDA focus on the computational parts of the GPU. CUDA is commonly used in the field of science and has some GPU research related topics for memory optimizations and management, which there is a lack of with the graphical APIs. It has not been possible on the same scale until recent years with the release of DirectX 12 and Vulkan as they allow more control over the GPU's resources. This study will use DirectX 12 to get a better understanding of resource management for the API and set some guidelines of the usage.

1.1 Aim and Objectives

The aim of this study is to benchmark the copying time of different types of heaps and command queues in DirectX 12 and to see how the performance scales when the data size is increased.

These are the following objectives that we need to accomplish to reach our aim:

1. Implement a model for testing the copying speed of different heap and queue types with DirectX 12 when increasing the data size.
2. Run the model on multiple GPUs and collect the data.
3. Analyze the data from our tests and from this draw a conclusion.

1.2 Research Question

The research question this study aims to answer is: "*How does data copy times differ between buffer heap types with increasing data sizes in DirectX 12?*"

1.3 Limitations

Because of the corona pandemic, there was no possibility of lending graphics cards from our college, which resulted that we could only get our hands on a few GPUs from Nvidia. Otherwise, we would have hoped that the tests could have been executed with cards from AMD as well.

1.4 The structure of the thesis

We will start by presenting some previous work related to memory copying and background for the DirectX 12 API in chapter 2. In chapter 3 will we describe the structure of our experiment and the scientific method that we used to answer our research question. The results will be shown in chapter 4, then analyzed and discussed in chapter 5. Lastly, in chapter 6 will we answer our research question and present future work that can be made from this study.

Chapter 2

RELATED WORK AND BACKGROUND

1.1 Background

When working with GPUs it is often required to upload resources to VRAM for lower reading latency when computing, this results in a need for moving data in between the RAM and VRAM on the PCI-E bus.

In the paper by Besedin K.Y. et al. [1] they compress data and transfer it from system memory to a processing device. In their study, it is mentioned that the time it takes to transfer the data on the PCI-E bus is considered the main bottleneck for their specific work.

Another worked that has been done with a similar approach is mention in the conference paper by S. Chien et al. [2]. They focus on improving the memory transfer model by compressing the data that is sent through the same bandwidth and gain a lower data transfer latency.

The latency in between the CPU and GPU is measured with the CUDA API in the study by Chaibou A.et al. [3]. Their work focused on when to use the GPU as a GP-GPU for processing and thereby copying the data over to the VRAM or if it is more beneficial to process the data directly in system RAM.

2.2 Background

There is much research when it comes to computing and memory management on the GPU with CUDA. This API is commonly used for computational tasks when it comes to larger data sets, but it is also used for applications to parallelize work with the CPU. However, this was not achievable with the graphical focused APIs like OpenGL or DirectX 11 because the programmer was not allowed to manage the memory in the same way. But with newer APIs like Vulkan or DirectX 12, it is possible to map and reserve memory for the GPU on both system RAM and VRAM.

The DirectX API is mainly used for developing games for both the consoles and PC market over the course of the last 25 years and a lot has changed with the API over that time, from new shader models to stages in the rendering pipeline. It was not until DirectX 12 where the memory resources could be managed by the programmer instead of the API.

With DirectX 12 is the communication to the GPU done with commands that get recorded to a command lists, which can get executed on three different commands queues that is provided by the API, which are Direct, Compute and Copy. The Copy queue can only take lists that has recorded copy commands which is used to tell the GPU to copy resources. While the Compute queue can execute copy commands as well as computational commands that is used for task that benefits from parallel computing. The Direct queue can do everything that the compute and copy can do and all the graphical commands that is used for 3D rendering. The different queues are enabling a parallelized workflow for both the CPU and GPU for optimization of the pipeline.

.

Chapter 3

METHOD

To be able to answer our research question, an experimental research method was used. A DirectX 12 prototype was implemented and used as our test case to measure the copying performance between resources. The API offers multiple ways on how data can be copied which includes various types of heaps where the resource can be allocated. There are also different command queues that are used to issue commands to the GPU. All possible combinations of heaps and queues were used on three Nvidia graphics cards to validate and compare the copying performance across generations of graphics hardware. Nvidia GeForce Game Ready Driver 445.87 was the graphics driver used for all the tests as this was the most up to date driver at the time of this research. The graphics cards were tested with a locked clock frequency because there could not be any guarantees on how it would affect the results if the GPUs are in an unstable state. The tests will therefore not represent a real case usage of the GPU as graphics cards normally adapts the frequency to the workload. Locking the frequency does however give a more predictable result, from which a comparable result can be collected. The source code for this study can be found in the appendix [1].

3.1 Procedure

The benchmarking of the copying performance was performed with a DirectX 12 implemented prototype which performed and measured all of the possible ways of copying data that was related to our research. In DirectX 12 data is stored as resources that can be allocated on the heap types Default, Upload and Readback. The largest difference with these is in which memory the resource gets allocated in, see Table 3.1 and what accessibility a device has to the resource. Resources can be in different states and during a copying operation, a resource can be either a copy source or destination. While a resource allocated on a Default heap can be used as both source and destination, Upload can only be used as a source and Readback as a destination. This meant that four different heap combinations could be tested, see Table 3.2. In DirectX 12 the developer issues commands to the GPU through a command queue which there are three different types of Direct, Copy and Compute. All three of these were tested to see if there are any performance differences when used to execute a copying operation. DirectX 12 has asynchronous rendering which means that the CPU does not wait for the GPU to be finished with its operations. To be able to know when the copying operation has been completed, the Fence function [4] of the API was used. Time was measured from the execution of the command queue until the GPU signaled the CPU through a Fence. All data sizes were also tested through the memcpy function which is a way of copying data on system RAM without the DirectX 12 API. This was done to get a reference to the copying speed when not using any specific API or graphics cards. To get a reliable result each test was repeated 10 times and an average was calculated.

Heap type	Memory location
Default	Video RAM
Upload	System RAM
Readback	System RAM

Table 3.1: The different heap types and in which memory they allocate in.

Source	Destination
Upload	Default
Default	Default
Default	Readback
Upload	Readback

Table 3.2: The combinations of heap types of which a copying operation is possible

To measure how the performance scales with increasing data sizes an increment of 64 MB was used as this is equivalent to 1024 memory pages, where each page is 65536 bytes [5]. At the end of each test, the data that was copied across the resources were validated to ensure that there was no data loss or corruption. The test requires that a minimum of two resources are in memory at the same time which led to a maximum size of a resource to be 2 GB. This size was chosen as it would not allocate all of the VRAM on any of the graphics cards that were tested while being large enough to see how the performance scales. To save on memory and thereby be able to allocate larger sizes, not all of the resources were allocated at the same time. The test was therefore divided into two parts. Part 1 tested *Upload to Default, Default to Default* and *Default to Readback* while part 2 tested *Upload to Readback*. After each part, all of the allocated resources were deallocated before the test continued.

With DirectX 12 it is possible to monitor the memory usage with the QueryVideoMemoryInfo function [6]. This was used to put the testing program to halt until the amount of allocated memory could be observed as deallocated. Because drivers can be deceiving when they actually free up memory, extra precautions were taken to ensure that it was safe to proceed with the testing. A wait-function was implemented to keep the program waiting for a chosen duration to give the computer extra time to deallocate the memory. The diagnostic tool in Visual Studio 2019 was used to manually find a good duration for how long the wait-function would run. A wait duration of 1 second was enough for most data sizes but when the total allocation was lower than 256 MB it required 7 seconds wait time before all memory was deallocated. In Figure 3.1 the first and second parts of the test are indistinguishable from each other during the first data size increment while in Figure 3.2 both peaks are distinguishable from each other and all the memory is deallocated. The wait time was the same on all of the tested GPUs and the memcpy function [7] used a wait time of 1 second. Do note that these durations were only tested on one computer so they might have to be adjusted if tested on different hardware.

Figure 3.1: A graph from Visual Studios diagnostic tool showing the memory usage of five data increases with a 1 second wait time.

Figure 3.2: A graph from Visual Studios diagnostic tool showing the memory usage of two data increases with a 7 second wait time.

The following steps describe how the test case was carried out:

Memcpy tests:

1. Increment the data size. Step 1-7 is repeated until reaching the maximum size.
2. Create two arrays with the wanted data size, one as the source and one as the destination.
3. Fill the source array with data.
4. Copy the data from the source array to the destination array with the memcpy function and measure the time.
5. Validate that the data has correctly been copied over.
6. Deallocate both arrays.
7. Call a wait-function so the CPU has time to deallocate the memory before continuing with the next test.

DirectX 12 test:

1. Increment the data size. Step 1-9 is repeated until reaching the maximum size.
2. Switch command queue. Step 2-9 is repeated until all queues has been used.
3. Measure the memory usage and then create the resources with the following heaps:
    a. One Upload.
    b. Two Default.
    c. One Readback.
4. Copy data with the given size to the Upload resource. This is done with the MemCpy function and is not measured as it does not involve the API.
5. Execute the following copy operations and measure the time with the Fence function:

a. *Upload to Default*.
b. Default to Default.
c. *Default to Readback*.

6. When the last copying operation is done, validate that all of the returned data is correct.
7. Deallocate all of the resources.
8. Before continuing take precautions that the memory is deallocated before continuing. This is done by comparing the memory usage and calling a wait-function.
9. Repeat steps 3-8, this time with one resource with the heap type Upload and one with Readback. The copy operation is done as *Upload to Readback*.

3.2 Hardware

The whole experiment was conducted on the same computer with the use of three different graphics cards, each component specified in Table 3.3 and 3.4. The GPU and memory clock on each graphics card had a locked frequency to give a more stable result. This was done by enabling Stable Power State [8] in DirectX 12. Another option is to use the software EVGA Precision X1 [9] but this solution does not work with every graphics card, in this case the GTX Titan.

CPU	CPU Frequency	Memory Type & Size	Memory Frequency
Intel i7 6700	3.4 GHz	DDR4 16 GB	2133 MHz

Table 3.3: The CPU and memory specifications.

Graphics card	Architecture	GPU Frequency (Used)	Memory Frequency (Used)	Memory Bandwidth (Used)	Memory Type & Size
GeForce GTX TITAN	Kepler (2012)	836 MHz (836 MHz)	6008 MHz (3004 MHz)	288.4 GB/s (144.2 GB/s)	GDDR5 6 GB
GeForce GTX 1080	Pascal (2016)	1607 MHz (1695 MHz)	10008 MHz (5005 MHz)	320.3 GB/s (160.2 GB/s)	GDDR5X 8 GB
GeForce RTX 2080	Turing (2018)	1515 MHz (1515 MHz)	14000 MHz (7000 MHz)	448.0 GB/s (224 GB/s)	GDDR6 8 GB

Table 3.4: The specifications of the graphics cards that were used.

Chapter 4

RESULT

From our test model was data collected, from which the differences in performance and behavior will be presented in the following order: heaps, queues and graphic cards.

4.1 Heaps

The results for the GTX Titan show that copying data from *Upload to a Readback* is the slowest of the four copying operations, see Table 4.1, Figure 4.1, 4.2 and 4.3. The fastest one being *Default to Default* with around 4.5% faster than Upload to Readback on average. While *Upload to Default* and *Default to Readback* performs the same. The test results also show that the copying time scales linearly with the increasing data size.

The GTX 1080 shows the same performance order but a larger difference in relative copying performance compared to the GTX Titan with a gap of 71% between *Default to Default* and *Upload to Readback*, see Table 4.1, Figure 4.4, 4.5 and 4.6. Even in this test *Upload to Default* and *Default to Readback* performs the same and the copying time scales linearly as well.

The RTX 2080 shows almost the same results as the GTX 1080 but with a performance difference of 78% compared to 71% when comparing *Default to Default* with *Upload to Readback* on both of the graphics cards, see Table 4.1, Figure 4.7, 4.8 and 4.9. The result of not locking the GPU and memory frequency on the RTX 2080 can be seen in Figure 4.10.

Copy operation	**GTX Titan Average time (ms)**	**GTX 1080 Average time (ms)**	**GTX 2080 Average time (ms)**
Default-->Default	93.3	28.4	20.8
Default-->Readback	94.5	84.2	84.4
Upload-->Default	94.7	84.6	84.9
Upload-->Readback	97.7	97.5	96.8

Table 4.1: Average times for each copy operation across all queues and data sizes on the GTX Titan.

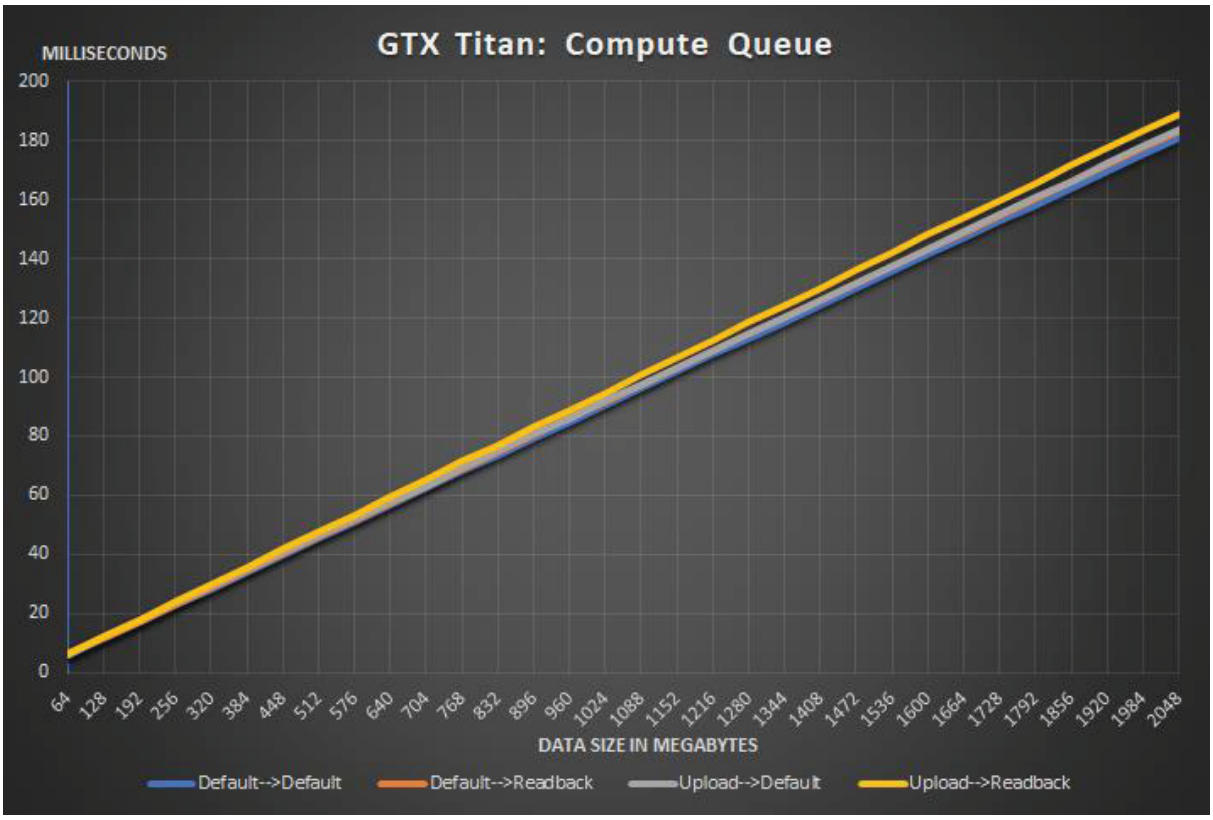

Figure 4.1: The results of copying data between all the heap types through a Compute queue on a GTX Titan.

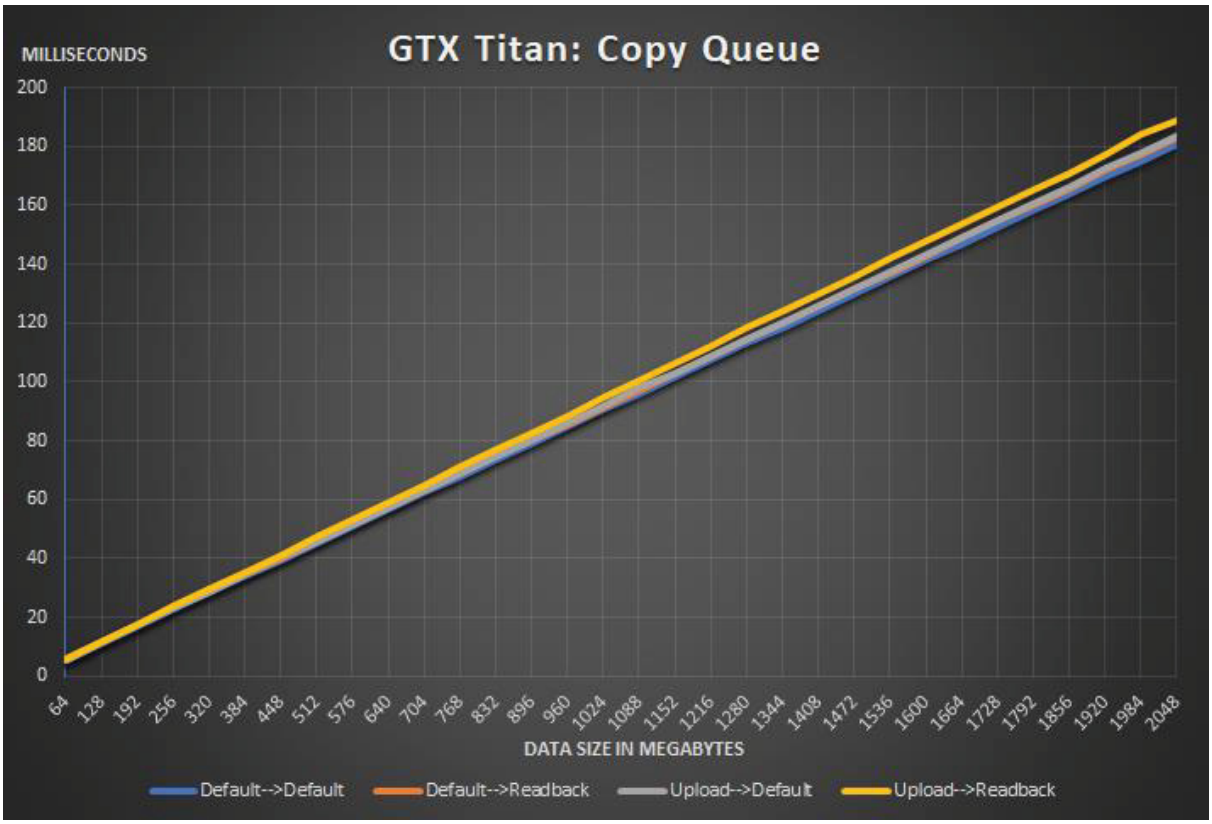

Figure 4.2: The results of copying data between all the heap types through a Copy queue on a GTX Titan.

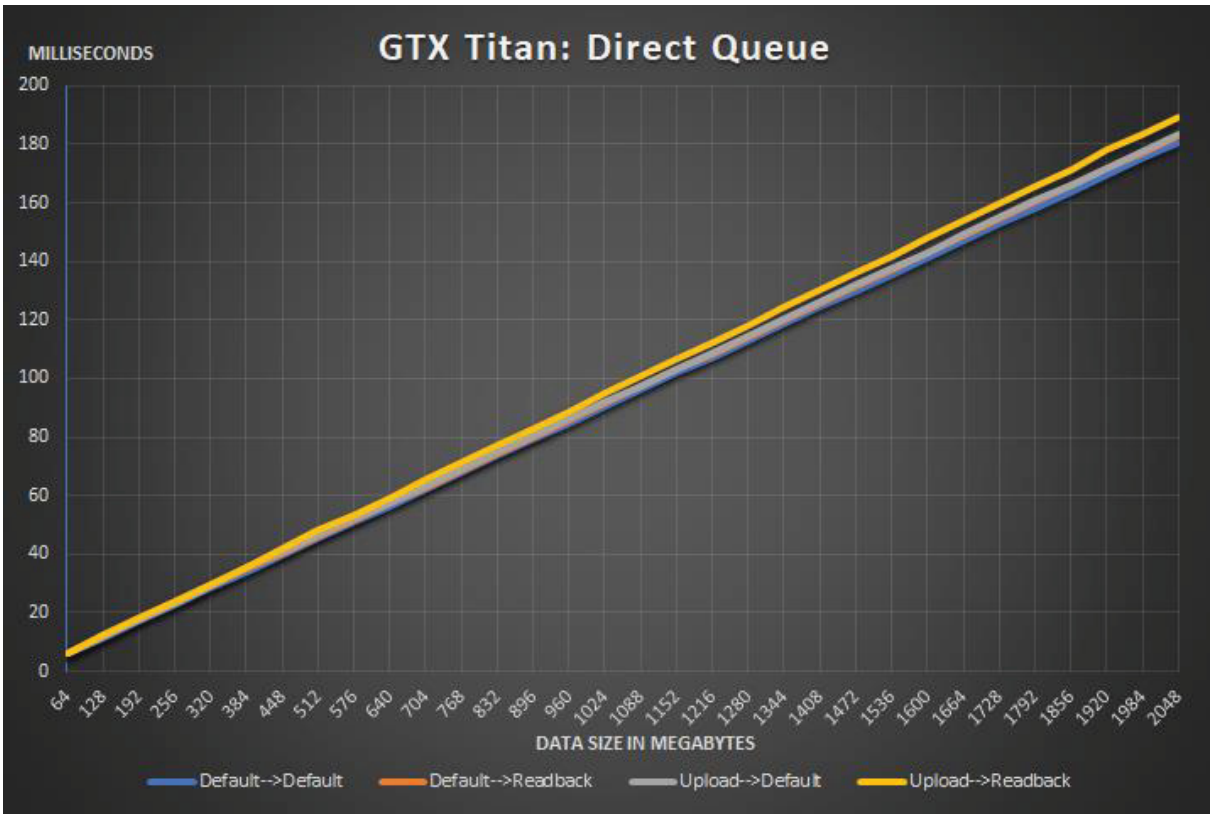

Figure 4.3:The results of copying data between all the heap types through a Direct queue on a GTX Titan.

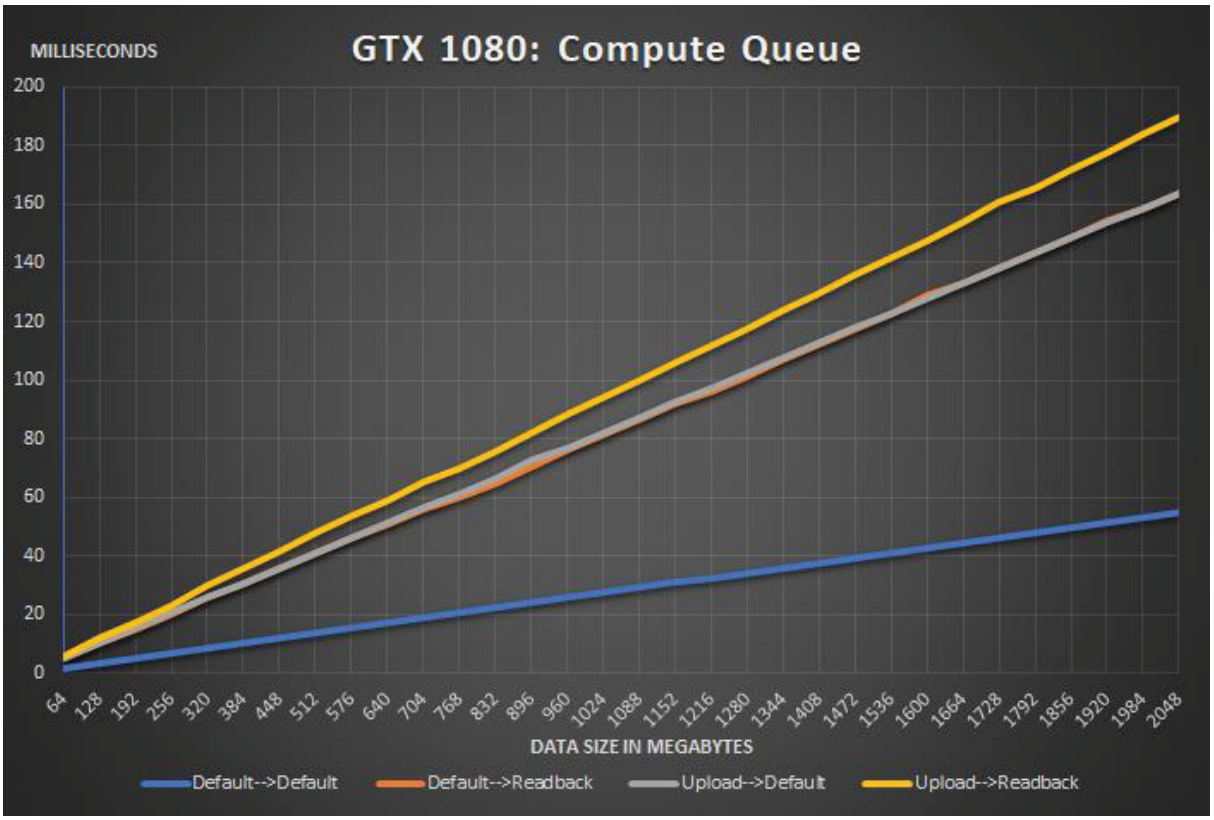

Figure 4.4: The results of copying data between all the heap types through a Compute queue on a GTX 1080.

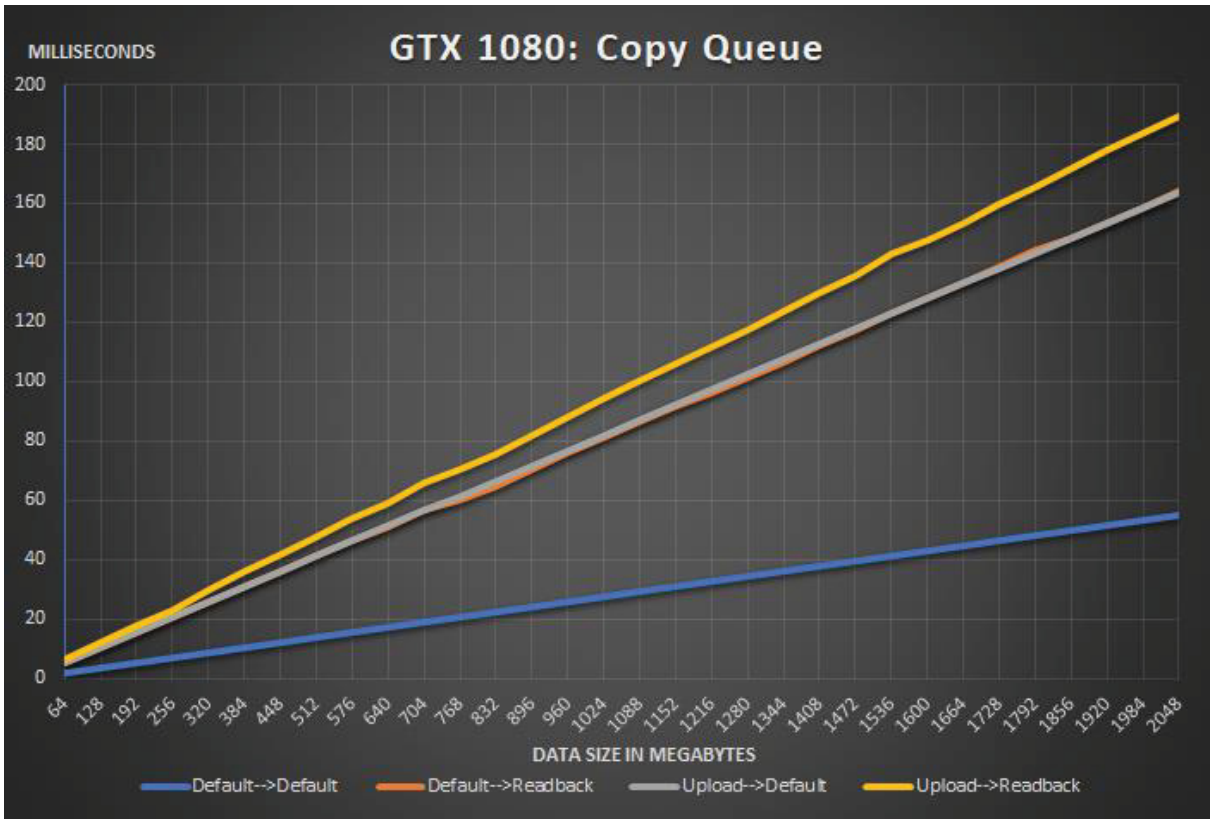

Figure 4.5: The results of copying data between all the heap types through a Copy queue on a GTX 1080.

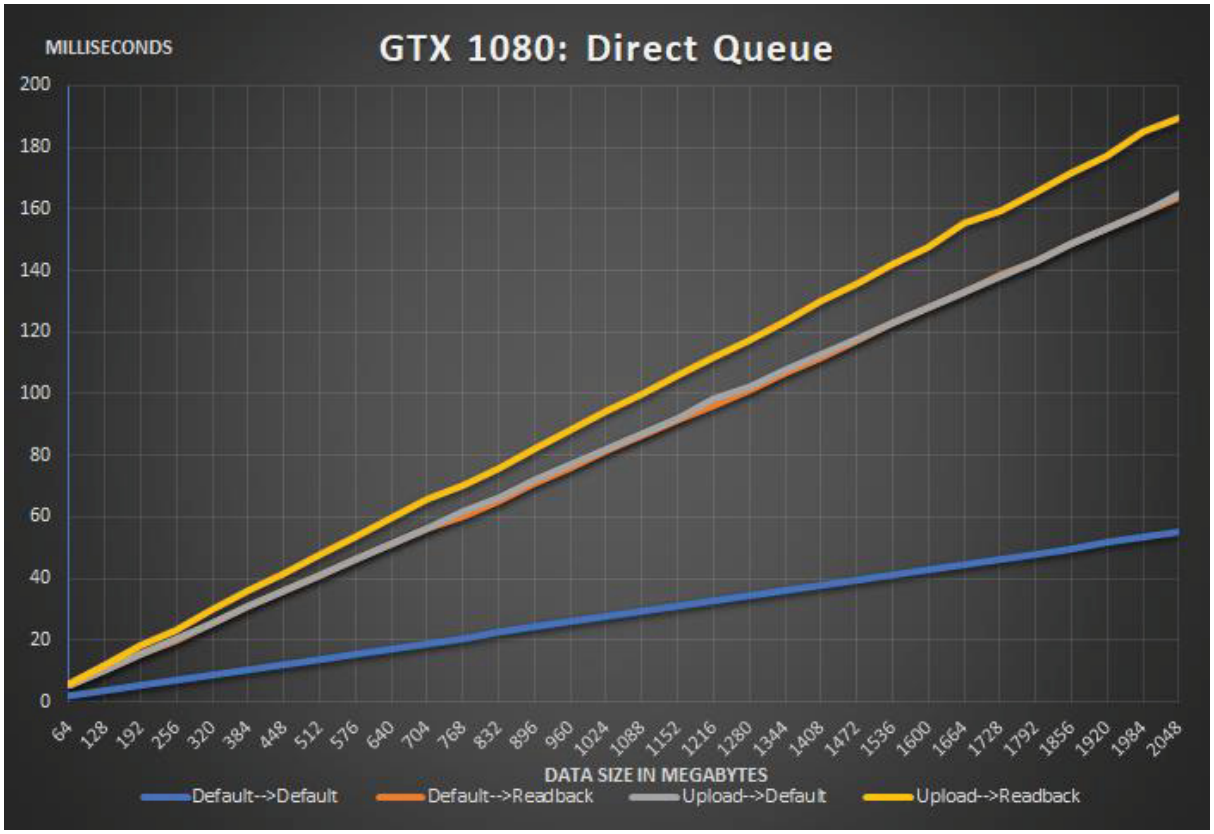

Figure 4.6: The results of copying data between all the heap types through a Direct queue on a GTX 1080.

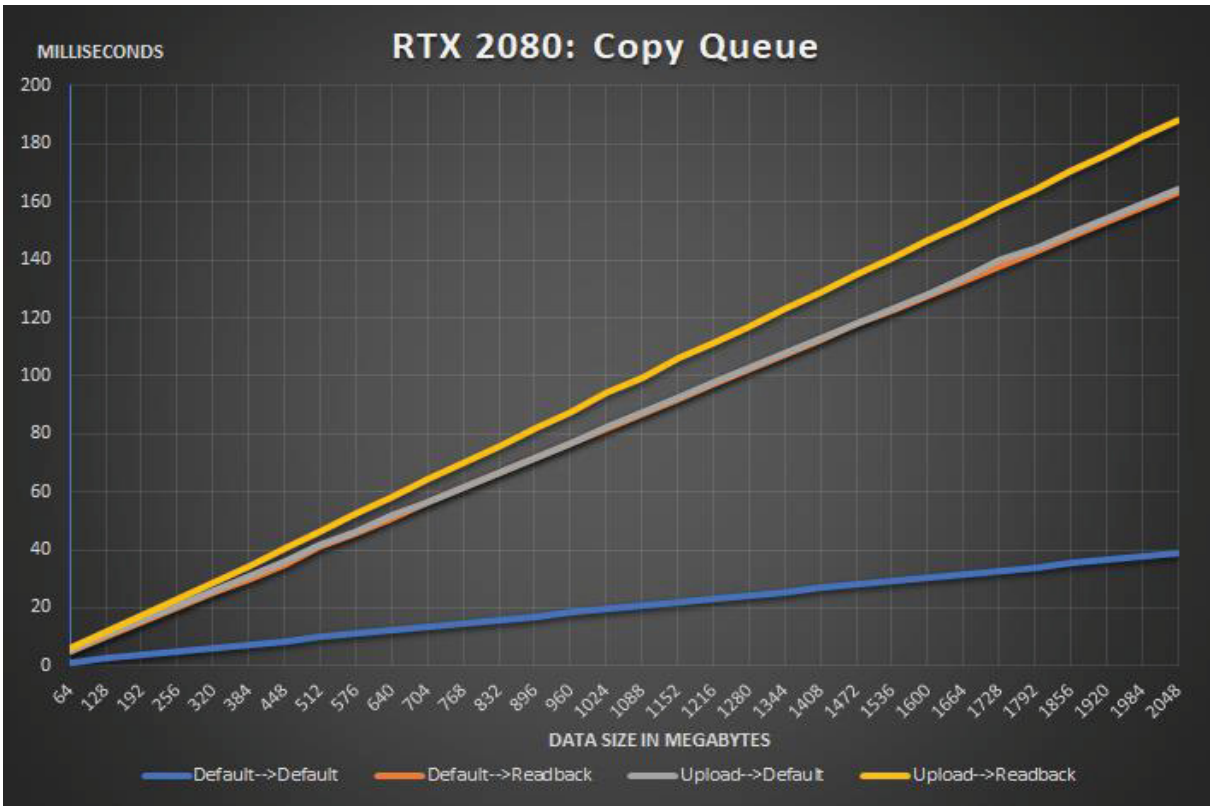

Figure 4.7: The results of copying data between all the heap types through a Compute queue on a RTX 2080.

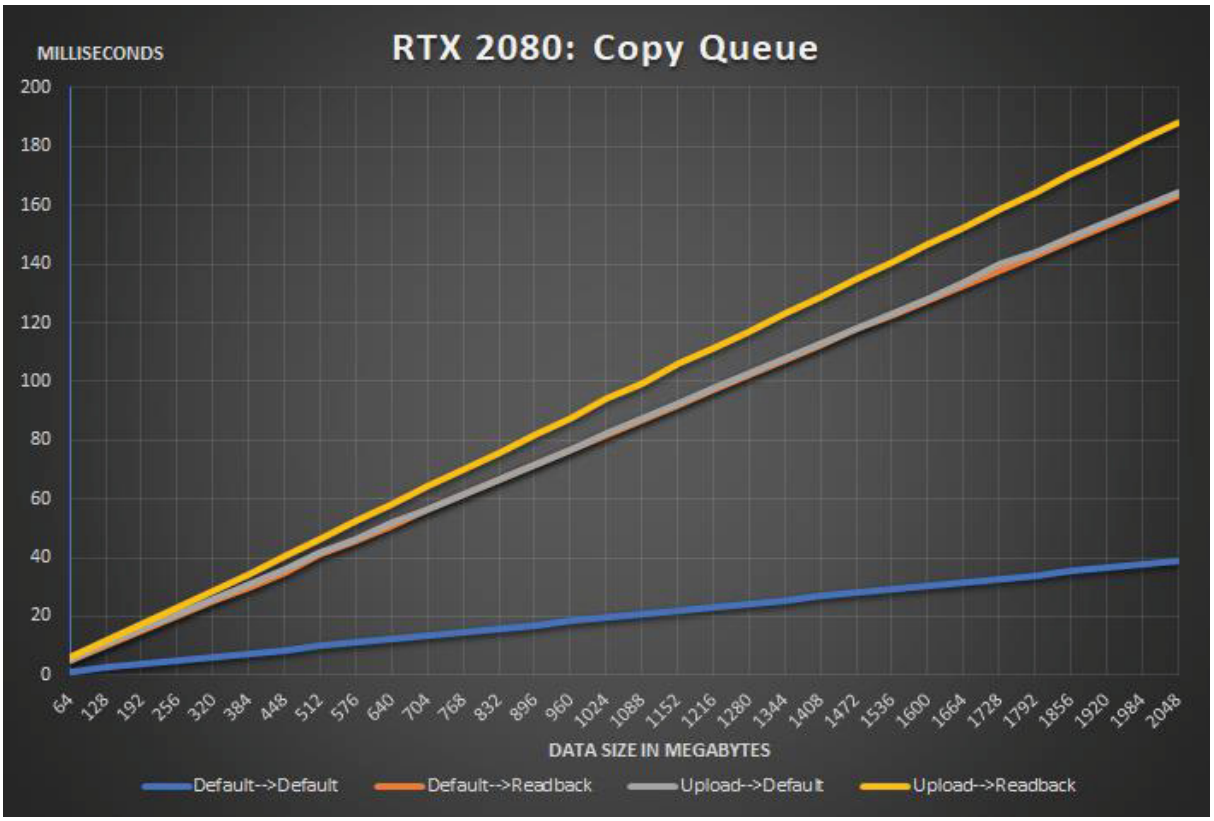

Figure 4.8: The results of copying data between all the heap types through a Copy queue on a RTX 2080.

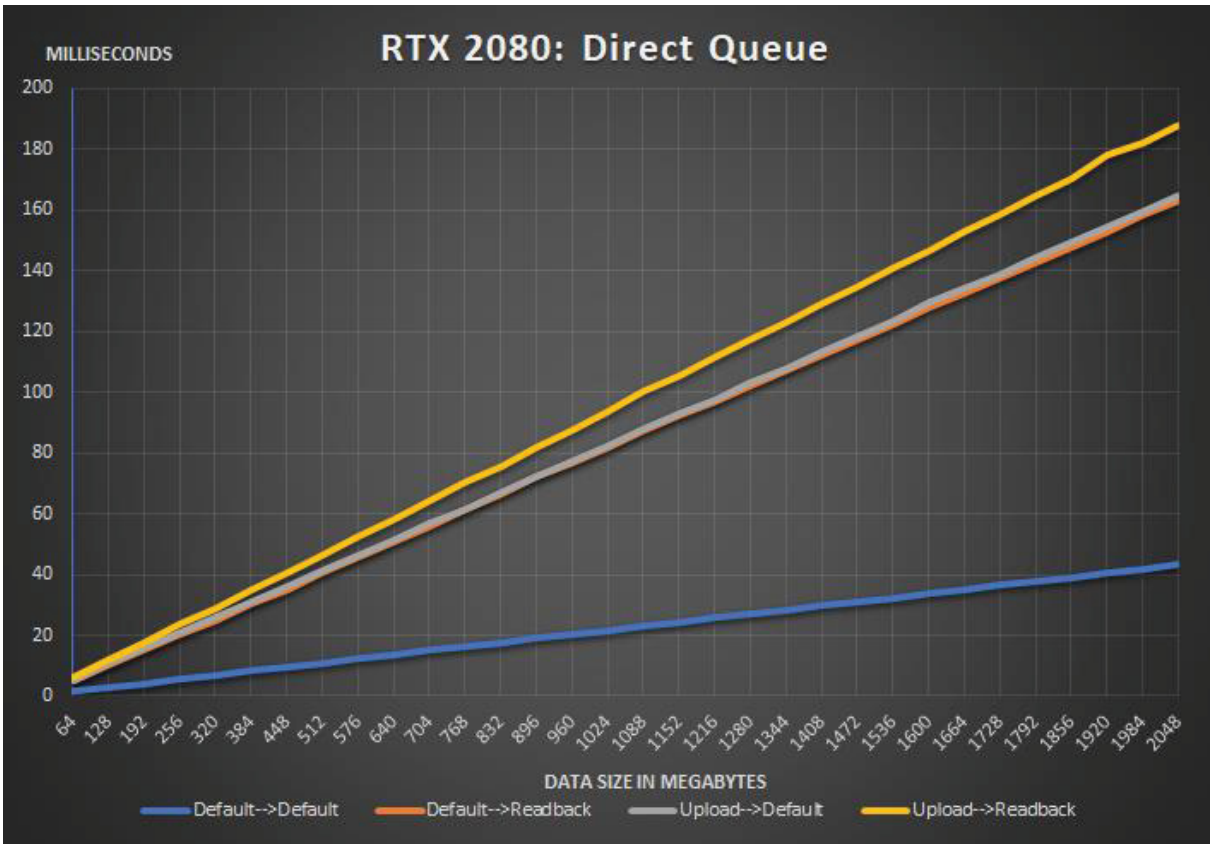

Figure 4.9: The results of copying data between all the heap types through a Direct queue on a GTX 2080.

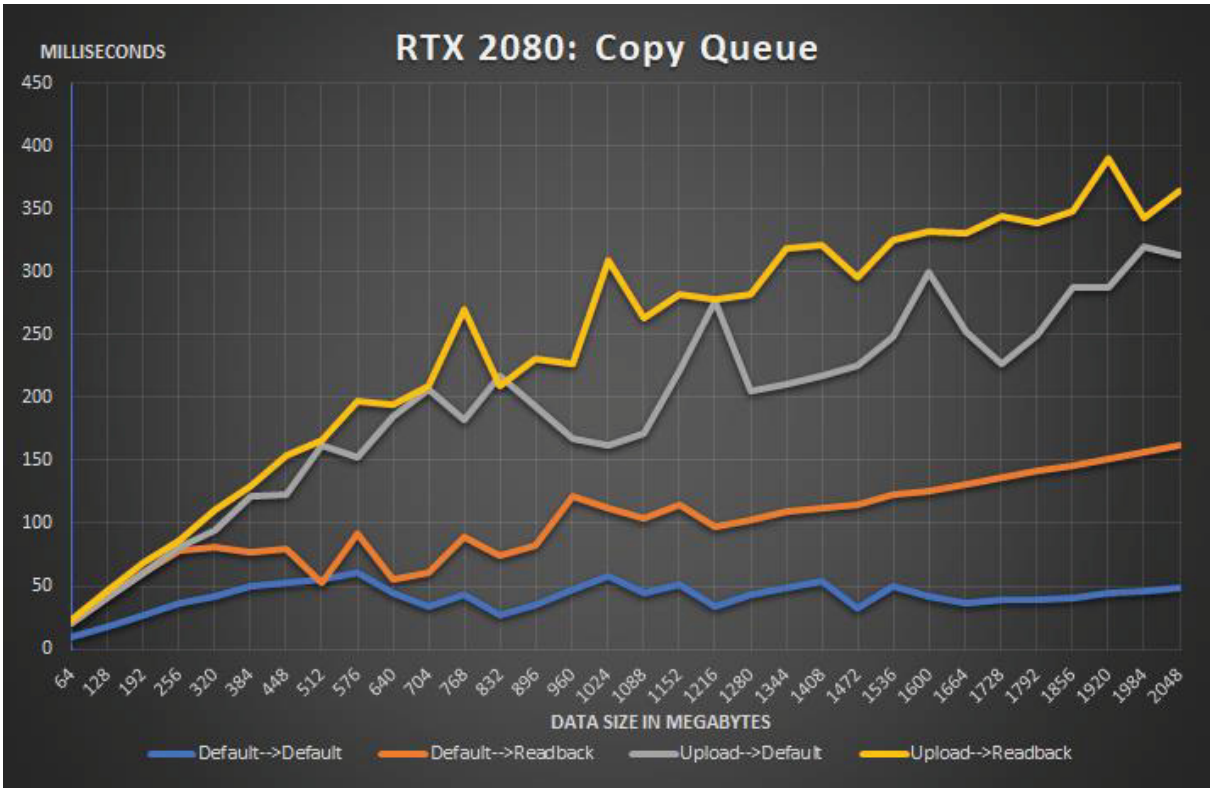

Figure 4.10: An example of the result of not locking the GPU and memory frequency on the RTX 2080.

4.2 Queues

When looking at the results between the data sizes 64 to 256 MB, there are some minor differences. The GTX Titan shows that operations done on the copy queue are completed faster and the difference is bigger on 64 MB than the larger sizes, see Figure 4.11. However, when comparing the averages in Figure 4.12 the difference is close to none. While the GTX 1080 shows a more uniformly behavior in performance between the queues in Figure 4.13, the average values in Figure 4.14 show no difference in performance. Almost the same can be seen with the RTX 2080 in Figure 4.15 and 4.16, with the exception of a 10% slower copying time on the Direct queue when copying from *Default to Default*.

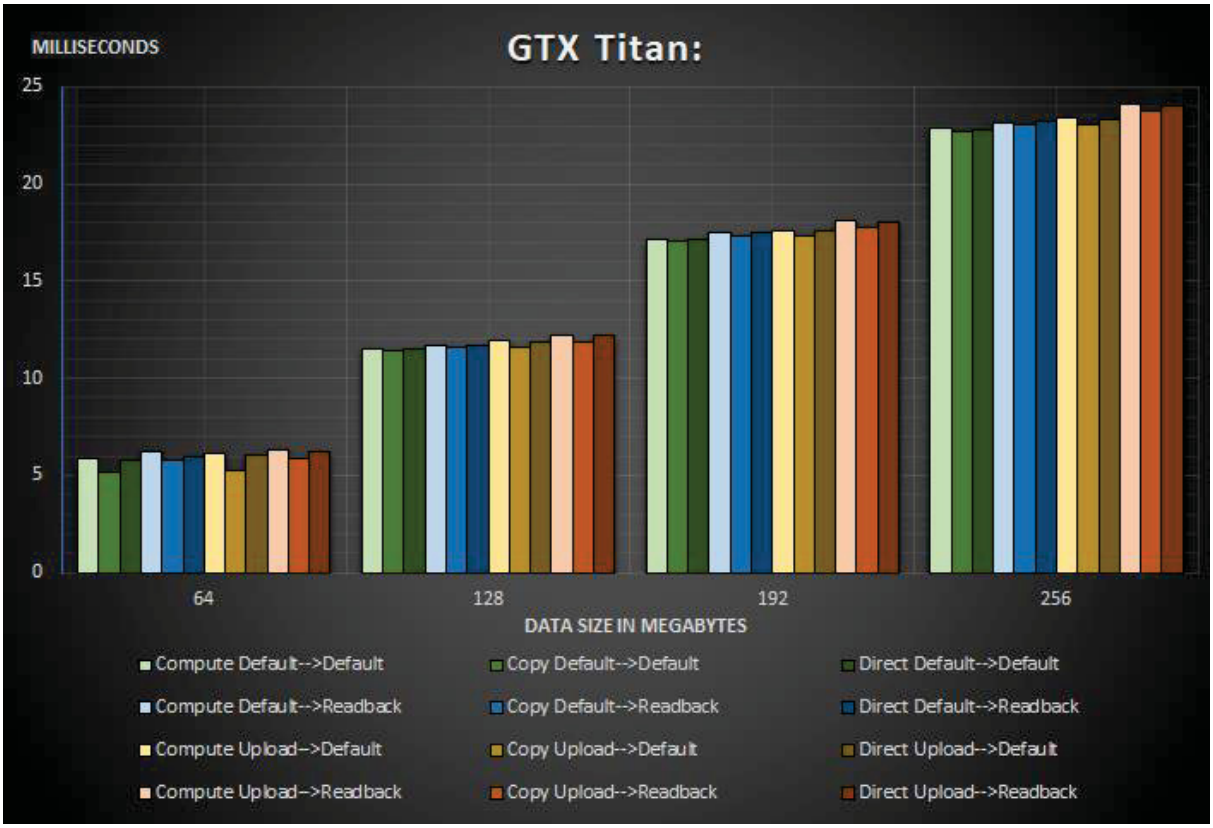

Figure 4.11: Comparison between the command queues when copying 64-256 MB of data between the heap types on the GTX Titan.

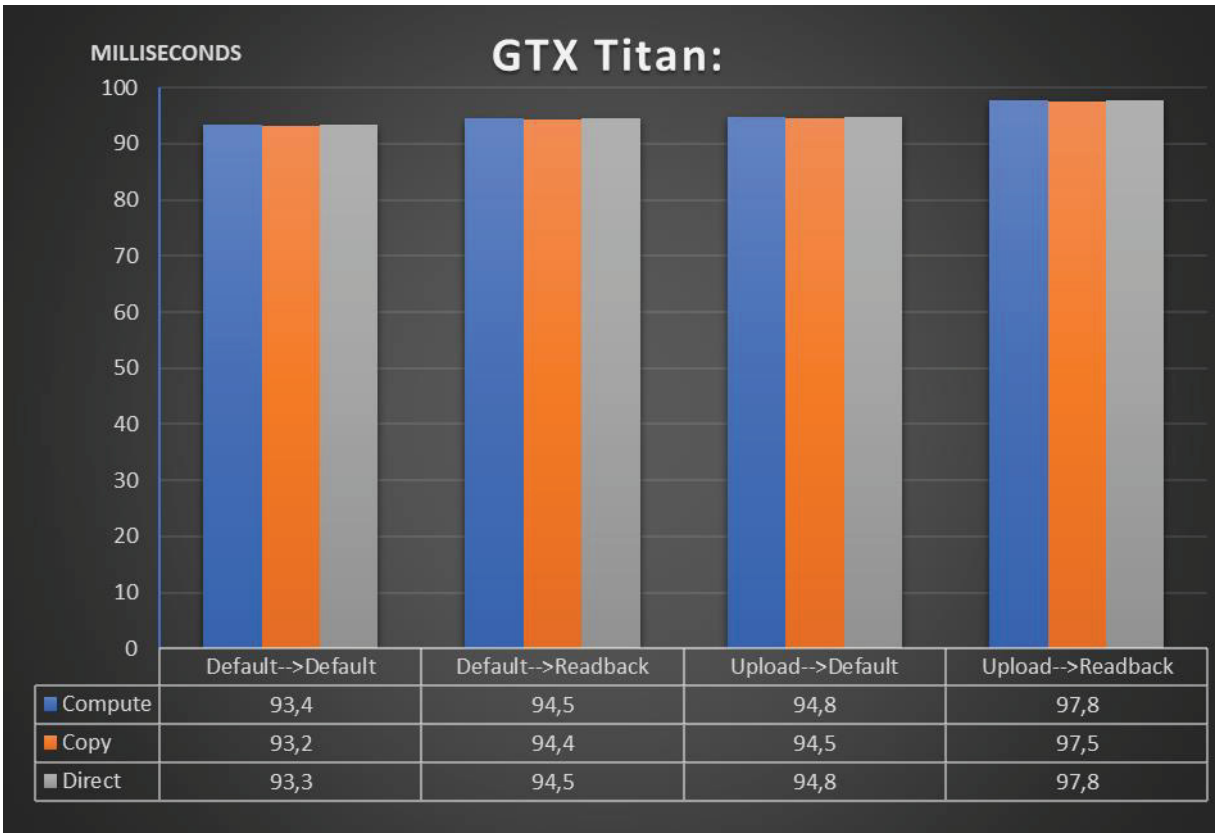

Figure 4.12: Comparison between the average copying time across all command queues and data sizes when copying data between the heap types on the GTX Titan.

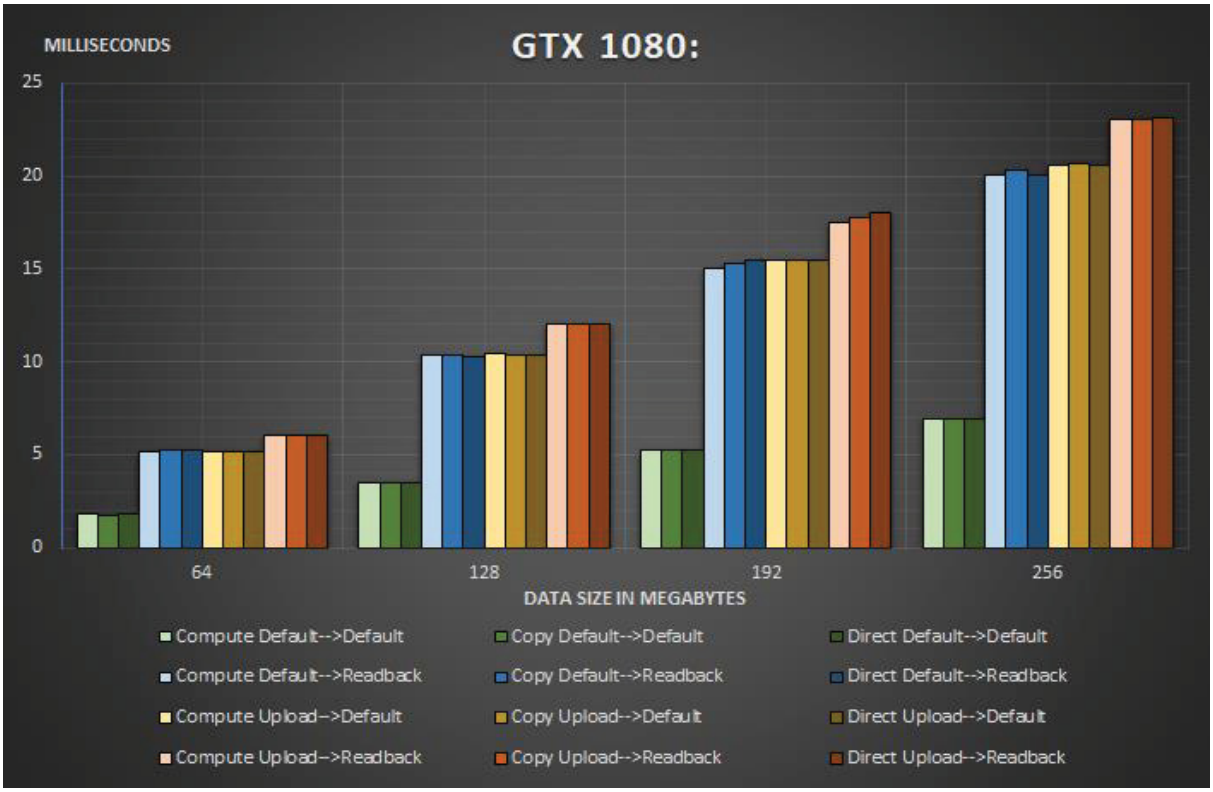

Figure 4.13: Comparison between the command queues when copying 64-256 MB of data between the heap types on the GTX 1080.

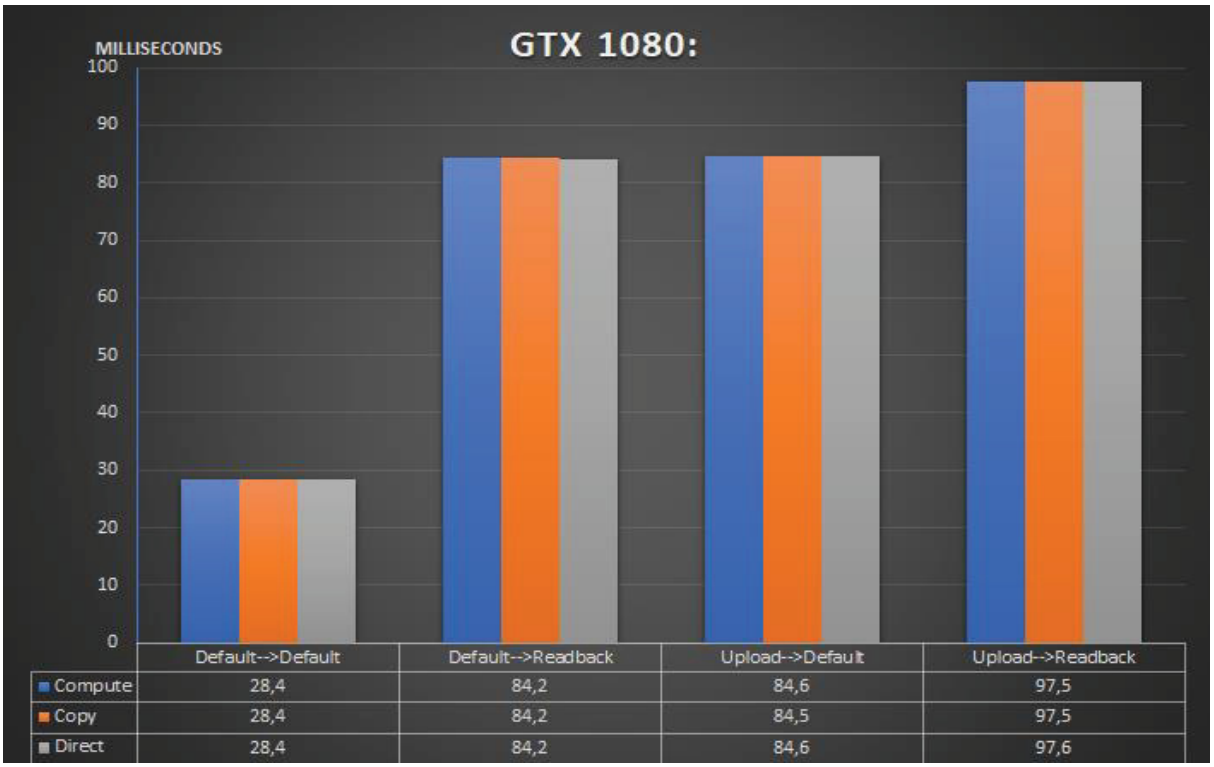

Figure 4.14: Comparison between the average copying time across all command queues and data sizes when copying data between the heap types on the GTX 1080.

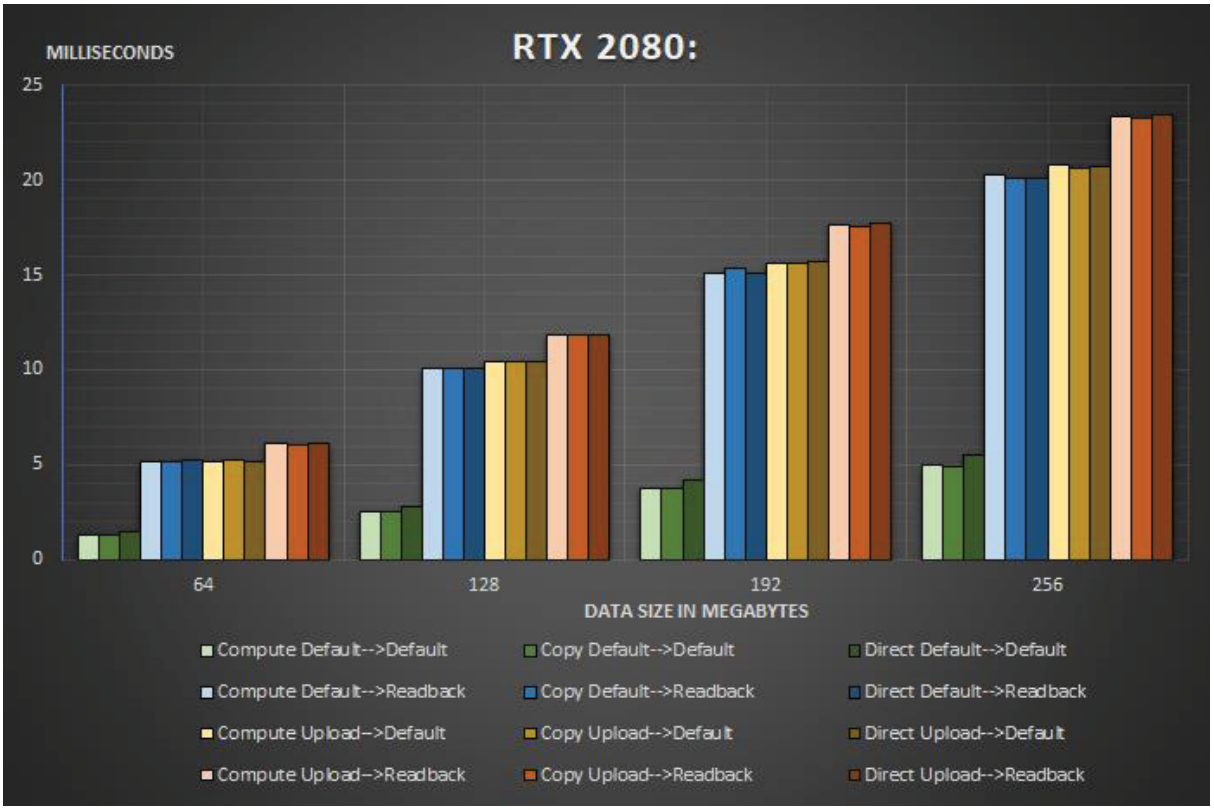

Figure 4.15: Comparison between the command queues when copying 64-256 MB of data between the heap types on the RTX 2080.

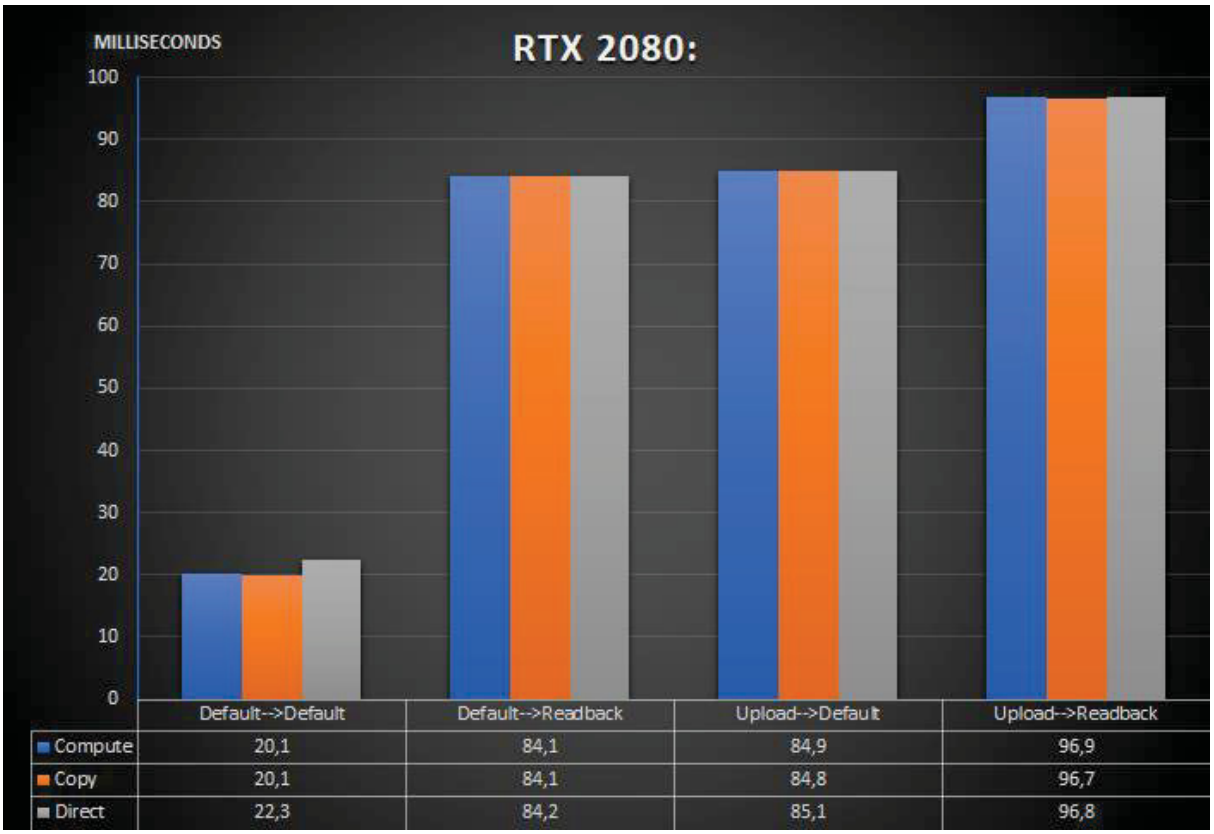

Figure 4.16: Comparison between the average copying time across all command queues and data sizes when copying data between the heap types on the RTX 2080.

4.3 Graphics cards

The results of comparing the graphics cards show that the biggest difference is when copying from *Default to Default*, see Figure 4.16. The GTX 1080 performs 1.52 times slower than the RTX 2080 and the GTX Titan is 4.65 times slower. *Default to Readback* and *Upload to Default* shows identical results with the GTX 1080 and RTX 2080 performing the same while the GTX Titan is 1.12 times slower, see Figure 4.17 and 4.18. Lastly, Figure 4.19 shows no performance difference between the cards when copying from *Upload to Readback* nor any driver overhead when compared with memcpy.

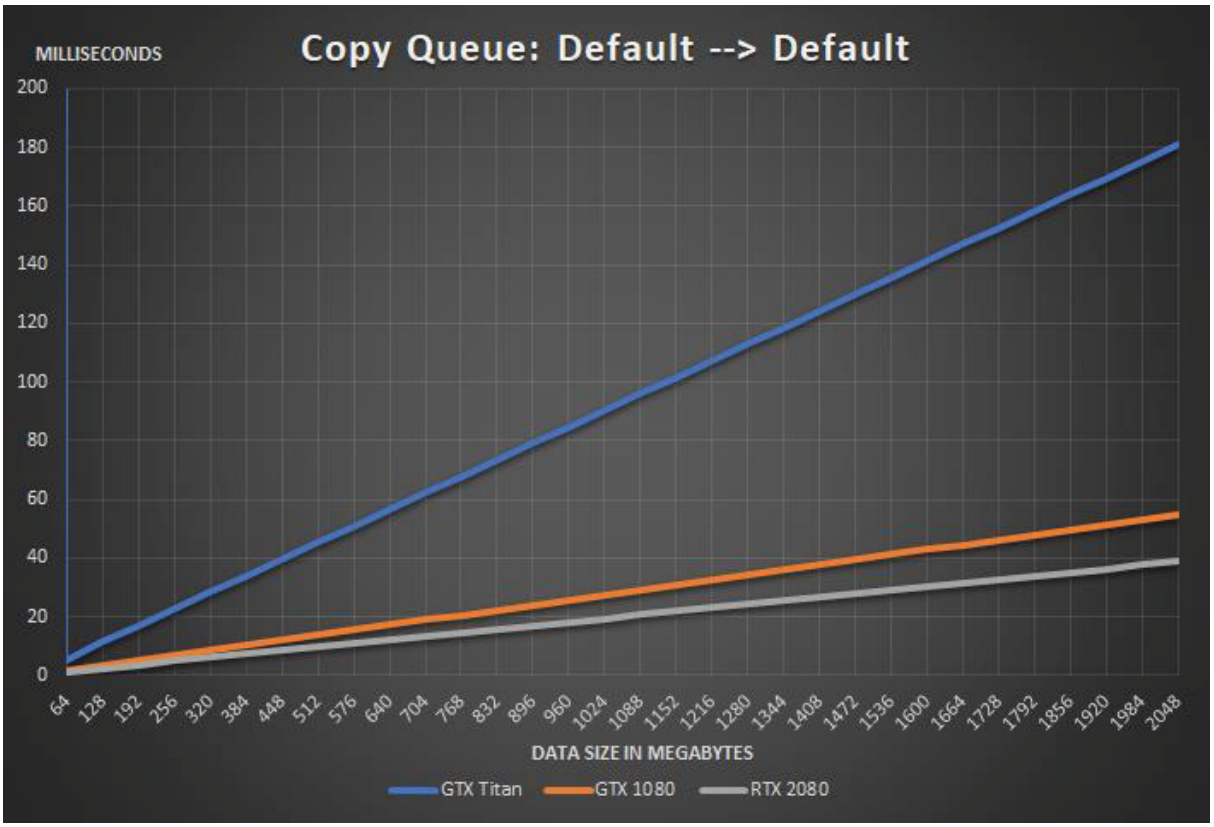

Figure 4.16: Comparison of copying data from Default to Default with a Copy queue across all graphics cards.

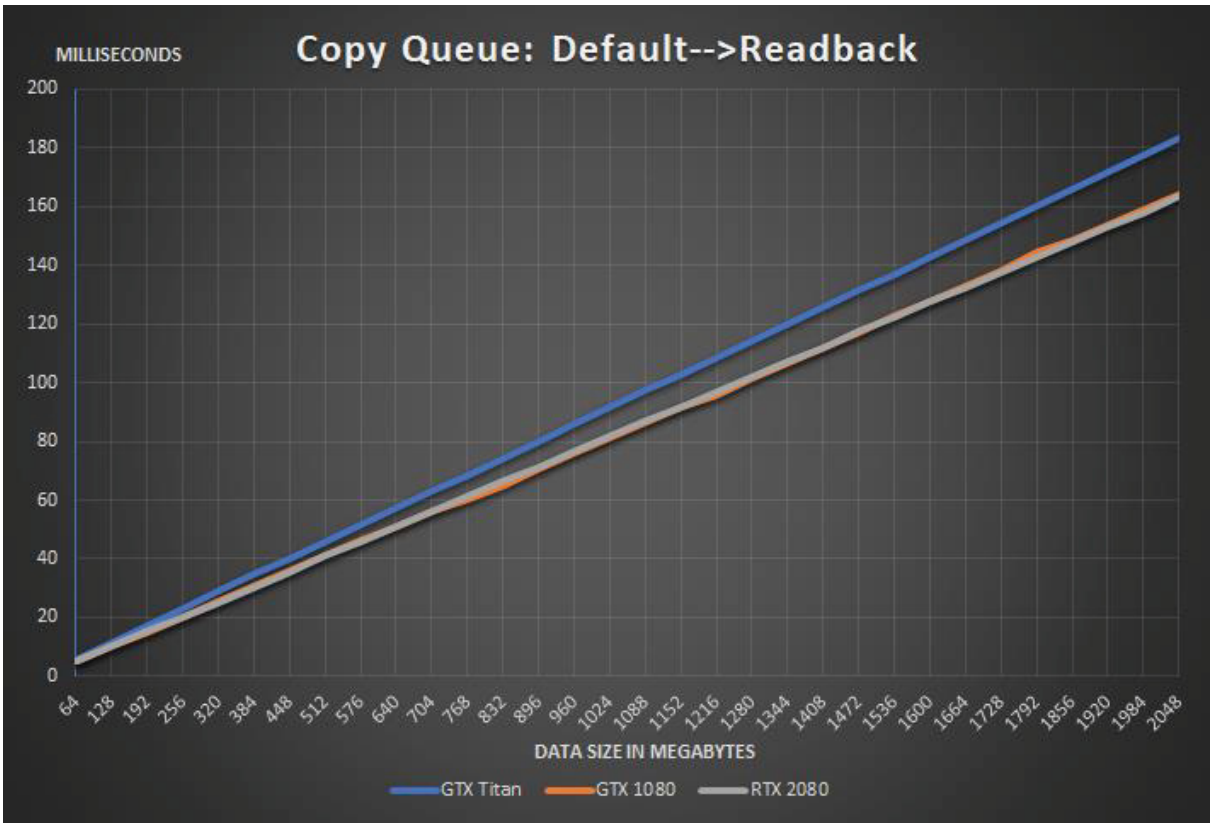

Figure 4.17: Comparison of copying data from Default to Readback with a Copy queue across all graphics cards.

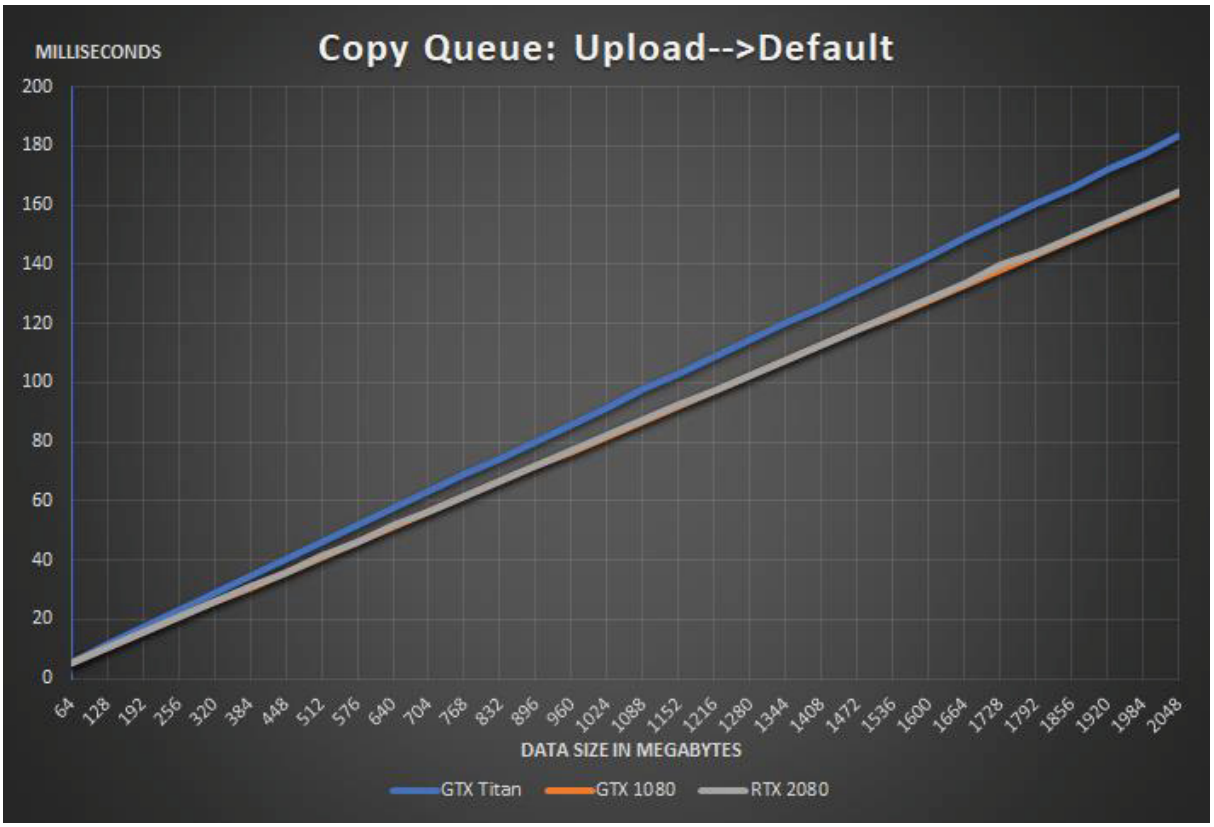

Figure 4.18: Comparison of copying data from Upload to Default with a Copy queue across all graphics cards.

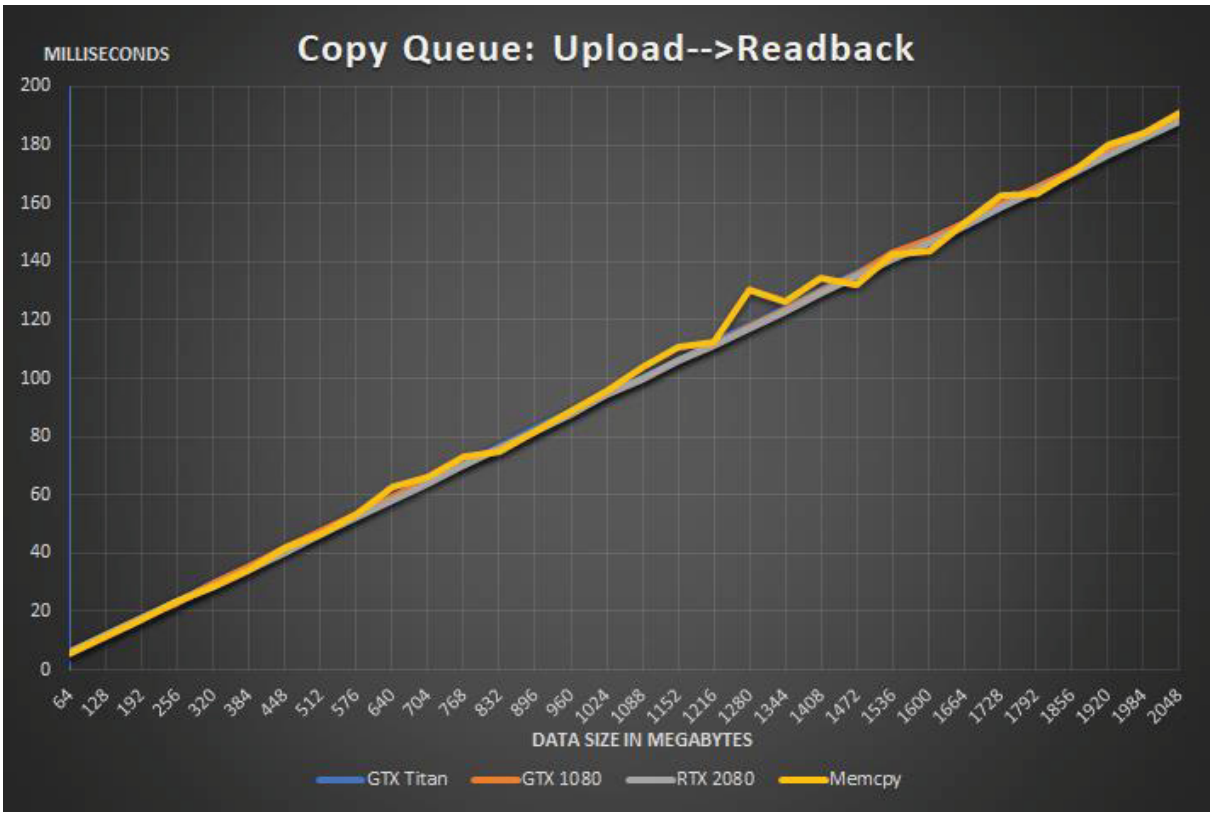

Figure 4.19: Comparison of copying data from Upload to Readback with a Copy queue across all graphics cards. The memcpy function is shown as a comparison.

Chapter 5

DISCUSSION AND ANALYSIS

When comparing the test results, we could observe that the heap types follow the same performance order between the graphics cards, with the following order from fastest to slowest.

1. *Default to Default.*
2. *Upload to Default / Default to Readback.*
3. *Upload to Reaback.*

Our tests show that the *Default to Default* always had the fastest copying time and this could be explained by that the data is copied internally in the VRAM. Comparing this to the *Upload to Readback* which is an internal copy operation in system RAM shows how much faster the VRAM on the tested graphic cards is.

In contrast to the other cards, the GTX Titan's VRAM copying performance was significantly slower. Comparing the specifications of the cards with the data from our study in Table 5.1 show that the GTX Titan only used 8% of its bandwidth while both the GTX 1080 and RTX 2080 used 22%. It should be noted that GTX Titan has a much older architecture and about half of the GPU clock frequency than the newer graphics cards. This could be two contributing factors for the much slower copying speed.

Hardware	**Memory Frequency**	**Bus Width**	**Bandwidth**	**Used Bandwidth**
GTX Titan	3004 MHz	384 bit	144 GB/s	11 GB/s (8%)
GTX 1080	5005 MHz	256 bit	160 GB/s	35 GB/s (22%)
RTX 2080	7000 MHz	256 bit	224 GB/s	50 GB/s (22%)
DDR4 Memory	2133 MHz	64 bit	17 GB/s	10 GB/s (59%)

Table 5.1 The specifications of the tested graphics cards and system RAM compared to the test results. The used bandwidth is calculated from the copying time.

The tests *Upload to Default* and *Default to Readback* had an insignificant difference in performance for all the graphic cards. Both of them copy data between VRAM and system RAM through the PCI Express bus and these results show that direction the data is copied has no impact on performance.

While the result from the memcpy function is less stable than copying data through DirectX 12, it shows that there is no difference in performance between these two options. The difference in stability suggests however that it matters from which processor the instructions are sent from. Our hypothesis is that when using the graphics cards which in our tests had a locked GPU frequency meant that every instruction could be sent at a constant rate. The CPU on the other hand used an unlocked frequency which can lead to fluctuating results as seen with the RXT 2080 in Figure 4.10.

The results from *Upload to Readback* tests show that there is no difference in performance when tested on different graphics cards except when it comes to having a locked GPU frequency. This suggests that the big difference in GPU frequency between the GTX Titan and GTX 1080 had no impact on the results of this copy operation. To paraphrase, a fluctuating GPU frequency has an impact on performance when copying on system RAM while having higher frequency does not. To achieve higher copying

performance in system RAM, higher memory frequency could be tested as this results with increased bandwidth.

No significant difference in performance could be observed between the queues on either of the graphic cards, with the exception of the RTX 2080 in the Default to Default with the direct queue. This showed on an average a 10% slower copying performance compared to the other queues which is a deviation we cannot explain, but an architectural change in the RTX 2000-series of cards could be a possible explanation.

Chapter 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

From all the test results that have been collected, can the question "*How does data copy times differ between buffer heap types with increasing data sizes in DirectX 12?"* be answered.
The results show that the performance scales linearly with the data sizes but there can be a big difference in the performance of copying data between different heap types. The fastest copying time for all graphic cards was the *Default to Default* copy operation, where the copying of the resources only occurred internally in the GPU's VRAM. The second fastest copy operations where the *Upload to Default* and *Default to Readback* which had an insignificant performance difference in our tests. Lastly, copying data from *Upload to Readback* was shown as the slowest.

This study shows a large difference in performance on the GTX 1080 and RTX 2080 when copying data internally in the VRAM compared to copying it between the two memory types over the PCI-E bus. The GTX Titan on the other hand does not have as large performance difference in any of the copy operations. This is explained by lower VRAM bandwidth utilization of just 8% compared to 22% on the other cards. The reason for this lower utilization could be caused by the comparably low GPU frequency or that this older card could have a less efficient architecture.

All the graphics cards gave the same results on the *Upload to Readback* test. This can be explained by that all the copy operations were done within the same system memory and therefore giving a consistent result. The test that was done with the memcpy function confirmed that there is no driver overhead of copying data on the system RAM through the DirectX 12 API. Our study also shows that fluctuating GPU frequency can impact the copying result on all heaps, even when using the API to copy internally in system RAM.

All the tests were executed with the different types of queues that are available with DirectX 12. From the results there was no difference between these which leads to the conclusion that there is no gain in performance when choosing one queue over the other when it comes to copying data. One exception to this was the use of a Direct queue when copying data from *Default* to *Default* on the RTX 2080 which resulted in a 10% slower copying time. Why this occurred could not be answered in this study.

6.2 Future work

This work does not cover all the aspects of copying resources on a GPU, but the tests show that hardware and its configuration have a great impact on the performance from where further work can be built upon. Figure 4.10 shows that the results can variate significantly depending on what frequency it is executed with. From this could the question be asked "How much does the GPU's clock frequency affect the copying speed?".

The tests were executed on the same hardware with different GPUs, from where the system RAM has had a major part in the tests. We have seen from the results that the slowest copy operation was when moving data internally in system RAM. When including another device and transferring the data over the PCI-E to VRAM there was a lower latency. Then the question is "How much would the copying speed be affected if the clock frequency of the system RAM were changed and how much influence has it with combination of the GPUs clock frequency?".

This work focused on the DirectX 12 API to structure a guideline for programmers on how to use the API more effectively. There is a gap in this work where only GPUs from Nvidia were used which does not cover the whole distribution of the GPU market.
To get a better picture of the overall usage for DirectX 12, this work should be tested with GPU's from another brand like AMD.

From the results of the RTX 2080 card could a deviation be noticed when copying data from *Default to Default* with the Direct queue. It would be interesting to see if there is a repeated pattern of this type within the Turing cards.

REFERENCES

[1] Besedin K.Y., Kostenetskiy P.S., Prikazchikov S.O. (2015) Increasing Efficiency of Data Transfer Between Main Memory and Intel Xeon Phi Coprocessor or NVIDIA GPUS with Data Compression. In: Malyshkin V. (eds) Parallel Computing Technologies. PaCT 2015. Lecture Notes in Computer Science, vol 9251. Springer, Cham

[2] S. Chien, I. Peng and S. Markidis, "Performance Evaluation of Advanced Features in CUDA Unified Memory," *2019 IEEE/ACM Workshop on Memory Centric High Performance Computing (MCHPC)*, Denver, CO, USA, 2019, pp. 50-57, doi: 10.1109/MCHPC49590.2019.00014.

[3] Chaibou, A. and Sie, O. (2015) Improving Global Performance on GPU for Algorithms with Main Loop Containing a Reduction Operation: Case of Dijkstra's Algorithm. *Journal of Computer and Communications*, **3**, 41-54. doi: 10.4236/jcc.2015.38005.

[4] Microsoft. 2018. ID3D12Fence interface. [Online] Available at: https://docs.microsoft.com/en-us/windows/win32/api/d3d12/nn-d3d12-id3d12fence [Accessed 14 June 2020].

[5] Microsoft. 2018. D3D12_HEAP_DESC structure. [Online] Available at: https://docs.microsoft.com/en-us/windows/win32/api/d3d12/ns-d3d12-d3d12_heap_desc [Accessed 14 June 2020].

[6] Microsoft. 2018. IDXGIAdapter3::QueryVideoMemoryInfo method. [Online] Available at: https://docs.microsoft.com/sv-se/windows/win32/api/dxgi1_4/nf-dxgi1_4-idxgiadapter3-queryvideomemoryinfo?redirectedfrom=MSDN [Accessed 14 June 2020].

[7] C++ reference. 2020. std::memcpy. [Online] Available at: https://en.cppreference.com/w/cpp/string/byte/memcpy [Accessed 14 June 2020].

[8] Microsoft. 2018. D3D12Device::SetStablePowerState method. [Online] Available at: https://docs.microsoft.com/sv-se/windows/win32/api/dxgi1_4/nf-dxgi1_4-idxgiadapter3-queryvideomemoryinfo?redirectedfrom=MSDN [Accessed 14 June 2020].

[9] EVGA. 2020. Precision X1 [Online] Available at: https://www.evga.com/precisionx1/ [Accessed 14 June 2020].

Appendix

[1] Link to GitHub for source code. https://github.com/J4ck5ilver/DirectX12-Resource-Testing

Faculty of Computing, Blekinge Institute of Technology, 371 79 Karlskrona, Sweden