PERFORMANCE OPTIMIZATION OF CMLAIR DYNAMIC AND STATIC SOLVERS USING GPUS

By

Tholfaqar Mardan Al Waheed

A report submitted in partial satisfaction of the

requirements for the degree of

Masters of Science, Plan II

in

Mechanical Engineering

at the

University of California, Berkeley

Committee in charge:

Professor David B. Bogy, Chair

Professor Roberto Horowitz

Fall 2016

Abstract

Performance Optimization of CMLAir Dynamic and Static Solvers using GPUs
By
Tholfaqar Mardan Al Waheed
Master of Science in Engineering – Mechanical Engineering
University of California, Berkeley
Professor David B. Bogy, Chair

Air Bearing Slider (ABS) design software was developed by PhD students in Computer Mechanics Lab (CML) over the years. It consists of a number of software packages such as Dynamic, Static (AKA Quick) and Thermal Fly-Height Control (TFC) to do specific tasks in relation to designing the ABS in a Hard Disk Drive (HDD). These tasks include designing the shape, and predicting the Fly-Height, Pitch and Roll properties. This software suite was later known as CMLAir software, which became popular among hard disk drive companies such as Hitachi Global Storage Technologies (Hitachi GST, a Western Digital company), Western Digital, Seagate and SAE. Engineers in those companies have been using the ABS software extensively for decades and run dozens of simulations daily. As a result, the need for faster software was understandable, essential and worth the development.

All of ABS design software packages were written in FORTRAN, which is the fastest scientific programming language in existence. Consequently, increasing the speed can be challenging using traditional methods. An alternative route was needed to make already fast software even faster.

There are numerous methods to speed up performance. One method that has proven to be efficient and affordable is using separate hardware to run scientific calculations. Graphical Processing Unit, also known as GPU, has the ability to run thousands of math-intensive pieces of software in parallel. Their unique architecture, which consists of hundreds of processors that harness large, dedicated Random Access Memory (RAM), can improve speed dramatically by a factor of three times as shown in subsequent chapters.

This work describes the use of GPU hardware via parallel computing techniques and Compute Unified Device Architecture (CUDA) FORTRAN language to enhance speed performance of ABS design software. In addition, this study also demonstrates why using a GPU is faster and much more cost effective than using multiple CPUs. Results, discussion, conclusion and future work are also addressed.

Acknowledgments

I would like to extend my sincere gratitude to my advisor, Prof. David. B. Bogy for his continuous guidance and support throughout my graduate career at Berkeley. Thanks to Prof. Roberto Horowitz for reviewing this work.

Also, thanks to my wife Tabarek for bearing with me through the journey of this project, and to my parents who encouraged me to continue and never give up my dreams.

I would like to express my gratitude to my fellow graduate students at CML for their help in explaining some of the mechanical aspects of the ABS design process, and for making research more enjoyable.

Finally, special thanks goes to Dr. Steve Rennich from High Performance Computing at NVIDIA, Mathew Colgrove, and the developers from NVIDIA/The Portland Group for their generous collaboration and support on this project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The computer code, CMLAir for ABS design, implements a multi grid control volume method to solve the generalized Reynolds equation. It can simulate all the main air bearing surface features such as multiple etch depth, shaped rails, and arbitrary wall profile. Numerical simulation has been an indispensable tool for air bearing design. An efficient numerical code enables us to evaluate many different designs without building and testing each one of them, thus shortening the cycle time and reducing the development cost. The governing equation for the air bearing pressure between the slider and the disk is the generalized Reynold's equation. A control volume formulation is used to discretize the equation. It is very stable and well suited for air bearing problems for the high speed and low spacing in the current magnetic hard disk drives [1].

The basic idea of the control volume method is easy to understand and lends itself to direct physical interpretation. The calculation of the domain is divided into a number of non-overlapping control volumes such that there is one control volume surrounding each grid point. The differential equation is integrated over each control volume. Piecewise profiles expressing the variation of the approximate solution between the grid points are used to evaluate the required integrals. The result is the discretization equation containing the values of approximate solution for a group of grid points [2].

For our case studies, which are considered medium to complex, a grid size of 192x192 points is the minimum for the solution to converge. While 402x402 grid size is common to use, a higher number of points (up to 1000x1000) are required for more complex slider designs with unique wall profiles and multiple etching depths. However, when you increase the size of the grid, the software takes a longer time to solve and converge. Additionally, when you need to run many complex cases of designs, it increases the time required to finish simulations. Hence there is a need for faster software.

Modifying the software to run on a different hardware (the GPU) rather than using the traditional CPU resulted in a noticeable speed improvement. The GPU architecture is unique and has many dozens of processors that can be utilized to solve complex mathematical computations. It also has its own dedicated memory for data manipulation without the need to copy data from/to the CPU, which is a costly process. The main concept behind the dramatic speed improvement (3X in some complex slider designs) is the division of tasks into smaller portions and computing them via GPU in a synchronized and parallel manner rather

than computing them serially in the CPU. Parallel computing has proven its superiority and speed efficiency over sequential computing [3].

The organization of this work was divided it into chapters:

- Chapter Two explains the CMLAir family of GPU software: motivation, need and details.
- Chapter Three presents the problem and the steps carried out to solve it.
- Chapter Four provides technical details about the computer and GPUs used during this project.
- Chapter Five addresses the Results and Discussion obtained from testing GPU Dynamic 5, TFC 2 and Quick 5 codes.
- Chapter Six details the Conclusion and Future work.

Chapter 2

CMLAir GPU software family

2.1 GPU Dynamic 5 code

Numerical simulation has been a powerful and versatile tool for studying head-disk-suspension dynamics. The simulation allows an investigation into more complicated dynamics problems in the disk. It offers fast and accurate prediction of the dynamic flying characteristics of the different head/disk interface designs [4].

There were multiple versions of Dynamic codes. The latest one was Dynamic 4, which runs fully on the CPU. Dynamic code version 5.0, however, is an upgrade from the Dynamic 4 code. We moved large number of potential "do loops" that take considerable amount of time by the CPU to be run on the GPU. With that being done to this version of CMLAir code, we were able to achieve 2.5X speed improvement using a complex slider design. This achievement was done by implementing a new feature called OpenAcc directives. The GPU directives are added to selected "do loops" to make them run on the GPU. This method is easier to use than kernel directives, and much more accurate, resulting in fewer errors when rapping the portion of the code intended to be executed into the GPU hardware. The Dynamic code implemented both methods to enhance both speed and accuracy [5]. This code was introduced in February, and updated in July 2016.

2.2 GPU TFC 2 code

The areal density of magnetic recording in HDDs has been increasing dramatically over the past decades. Because the increase of areal density requires the magnetic data bits to pack closer to each other, the physical spacing or Flying-Height of the magnetic head has to be reduced accordingly in order for reliable read/write operations. The physical spacing in current commercial HDDs is below 3 nm. Thermal Flying-Height control (TFC) technology is now widely used in state-of-the-art hard disk drives as an effective method to reduce the head-disk spacing and improve the hard drive performance and reliability. The TFC concept involves embedding a separate heater into the read/write head [6].

The TFC code version 2.0 includes a set of macro files written in APDL (ANSYS Parametric Design Language), which can be directly executed in the commercial finite element

software, ANSYS. These macros use an iterative approach to solve the flying-attitude and thermal deformation of a slider with micro-heaters built within the TFC slider [7]. Since TFC simulation is a repetitive task, GPU TFC v2.0 will run from 20% up to 100% or 2X faster than CPU TFC v1.0 each iteration depending on the slider design. Thus, using GPU TFC solver will increase the speed of simulation and improve efficiency [8]. This software was introduced in 2014.

2.3 GPU Quick 5 code

GPU Quick 5 is a modified version of Quick 4. With this edition of CMLAir, we were able to move 80% of the Quick code to run on GPU architecture. Furthermore, depending on the shape and initial conditions of the ABS, you can get different speed improvement characteristics compared to Quick code version 4. This can vary with the slider design, as shown in the examples and figures in a later section, from 20% for a traditional and simple ABS designs to as much as 200% or 3X for more complex designs [3]. This software was introduced in 2013.

Chapter 3

Problem Definition

By dividing math calculations inside the code into smaller portions and trying to compute them in parallel rather than serially, it is possible for the ABS code to run faster. One of the platforms that support parallel processing more optimally than others (such as CPUs) is the GPU. Why use a GPU and not a CPU? Here are some considerations that make GPUs a more promising choice:

- GPUs contain more processors than the latest CPU.
- GPUs have a lower price point than CPUs for the amount of internal processors (Core i7 CPU=4, Tesla GPU=2496, GeForce680 GPU=1536).
- GPUs have a more reliable, robust and easy-to-use parallel processing module than CPUs.
- GPUs work separately from CPUs and possess their own hardware (processors and dedicated Random Access Memory (RAM)).
- There are better programming languages available to run codes in parallel (CUDA FORTRAN or CUDA C) on GPUs than OpenMP (Open Multi-Processing) designed to run on CPUs.

The aforementioned considerations can make any engineer eager to start using GPUs to solve mathematical problems. However, changes to the existing code are a necessary first step before a system can achieve faster performance speeds. As for the Dynamic 5 version of ABS design software, we used a different and a new technique called OpenAcc directives. We added 500 lines of code to Dynamic 4 to make it able to run on a GPU. We used GeForce 960 GPU hardware manufactured by NVIDIA for the testing. A speed enhancement of 2.5X was gained [5].

As for the TFC code version 2, we have used a parallel processing technique called CUDA kernels, which basically creates subroutines written in CUDA FORTRAN that communicate between CPU and GPU and off load portions of the code into the GPU and run it in parallel. The following list of changes have been made to the TFC code [8], which is similar to the changes and methodology made to Quick code version 5 using kernel subroutines before running it on the GeForce 680 GPU [3]:

- The Quick 5.0 code has 4,400 lines added to support parallel processing.

- One file has been added, kernels.f90, which has 2,320 lines of code and 26 kernel subroutines.
- 8 new modules have been added to enhance portability of data handled by the code and optimum memory utilization.
- 2 files have been migrated from FORTRAN 77 to FORTRAN 90 to support CUDA FORTRAN.

A speed improvement of 3X was achieved for some of the test cases we will demonstrate in subsequent chapters.

Chapter 4

Technical Details

4.1 Hardware Requirement

During the testing of the new GPU Dynamic 5.0, we used a Dell desktop computer model: XPS 8500. It has powerful core i7-3770 processor, 3.4 GHz (four processors on a chip), 16 GB of Double Data Rate 3 (DDR3) RAM and a PCI express slot version 3.0 that attach the GPU to the motherboard. A decent power supply on board is required since the GPU, during calculations, consumes a relatively large amount of DC power, as shown in table 4.1.The GPU has its rated maximum power that needs to be noted and accommodated. Inside the Dell XPS 8500, we installed the following NVIDIA graphic card model: GeForce GTX 960 with technical details shown in table 4.1. It can be classified as a medium level GPU with powerful features and a compelling price that is used by many video game enthusiasts. A picture of the GPU is shown in figure 4.1a. We chose the Dell XPS 8500 for the following reasons:

- It has the minimum required power supply by the GeForce GTX 960.
- It contains a PCI express 3.0 bus interface on its motherboard.
- Its case and inside space is more than enough to comfortably accommodate the GeForce GTX 960.

NOTE: To protect the power supply from overheating, do not to use the DVD/CD for playing or burning while using GPU Dynamic 5.0. No further action is needed with respect to the heat generated by the card when working at its maximum capacity since the GPU has two large centralized fans on top of the main chips.

As for the testing of GPU TFC 2, we used a Dell XPS 8300 computer with a GeForce GTX 680 GPU installed inside it [8]. Technical details for the GPU can be examined from looking at table 4.2 while a picture of the card is shown in figure 4.1b.
For the GPU Quick 5, we also used the Dell XPS 8300 computer with installed GeForce GTX 680 GPU inside. Furthermore, on a separate experiment, we used a Tesla K20 GPU installed on a server machine belonging to NVIDIA. Tesla K20 GPU technical details are documented in table 4.3, and a picture of the card can be seen in figure 4.1c.

4.2 Software Requirement

After installing the GeForce GTX 960 GPU, we needed to install the latest GPU driver. GPU driver installation can be performed by using the installation DVD, which comes with the GPU, or downloading it directly from NVIDIA GPU driver's website. To ensure greater system compatibility and bug-free software, we downloaded the latest GPU driver software from the www.GeForce.com website. At the time of benchmark testing, the latest GeForce driver was version 361.91 for the GPU Dynamic 5. The installation of the latest GPU driver is a straightforward process [5].

4.3 GPU's Technical Specifications

CUDA Cores	1024
Base Clock (MHz)	1127
Memory Size (GB)	2
Memory interface width	128-bit
Memory bandwidth (GB/sec)	112
Power dissipation (Watts)	20 idle, 120 max
Minimum PC power supply (Watts)	400
Dimensions (Inch)	9.5 length x 4.38 height x 1.5 width
Price ($US)	220

Table 4.1: Technical specifications for GeForce GTX 960 GPU

CUDA Cores	1536
Base Clock (MHz)	1006
Memory Size (GB)	2
Memory interface width	256-bit
Memory bandwidth (GB/sec)	192.2
Power dissipation (Watts)	25 idle, 195 max
Minimum PC power supply (Watts)	550
Dimensions (Inch)	10 length x 4.38 height x 1.5 width
Price ($US)	450

Table 4.2: Technical specifications for GeForce GTX 680 GPU

CUDA Cores	2496
Base Clock (MHz)	706
Memory Size (GB)	5
Memory interface width	320-bit
Memory bandwidth (GB/sec)	208
Power dissipation (Watts)	25 idle, 225 max
Minimum PC power supply (Watts)	650
Dimensions (Inch)	10.5 length x 4.38 height x 1.5 width
Price ($US)	3600

Table 4.3: Technical specifications for Tesla K20 GPU

4.3.1 GeForce GTX 960

The NVIDIAs GeForce GTX 960 GPU card has interesting characteristics that can be utilized for math calculations as shown in table 4.1. It has 1024 cores running at 1.1 GHz speed. In addition, it has 2 GB of fast Graphics Double Data Rate 5 (GDDR5) memory with 128-bit bus width divided between cores. This unique memory feature reduces the need for transferring data back and forth between the GPU and CPU. It also reduces the heap (memory reserved for intermediate calculations), which speeds up problem solving. Having a PCI express 3.0 data bus takes data transfer to a new level compared to older GPUs. This explains the ultrafast transfer rate of 112 GB/second between the GPU and CPU and its peripherals. However, such a GPU needs substantial power to perform its operations when running simulations. Price-wise, the GeForce 960, which is priced at $220, is in the medium price range between the high-end, Tesla K20 cards (priced at $3,600 US) and the low-end older GeForce models such as GeForce 460 (priced at $120) with fewer and slower processors and smaller memory [5].

4.3.2 GeForce GTX 680

This GPU is an older model than the 960. The details are shown in table 4.2. It has 1536 cores running at 1 GHz speed. In addition, it has 2 GB of fast GDDR5 memory with 256-bit bus width divided among the 1536 processors. It has PCI express version 3.0 bus, which has a great data transfer between CPU and GPU. The transfer rate is 192.2 GB/second priced at $450 in 2013 [3].

4.3.3 Tesla K20

By contrast, the Tesla K20 card is a high-end GPU card. It has more CUDA cores, larger memory, a wider memory interface bus, increased memory bandwidth, larger maximum power and a higher price point than the GeForce GPUs as shown in table 4.3. We do not recommend the Tesla K20 unless there is budget for it. GeForce cards can perform comparably close to Tesla cards without the added expense.

(a) GeForce GTX 960

(b) GeForce GTX 680

(c) Tesla K20

Figure 4.1: NVIDIA GPU's

Chapter 5

Results and Discussion

In this section we will discuss the benchmarking and explain the results we obtained for the new GPU codes: Dynamic 5, TFC 2 and Quick 5. To perform an accurate evaluation, we used the verity of slider designs as case studies. We used 7 slider designs when testing Dynamic 5, 3 sliders for TFC 2, and 8 sliders for testing the Quick 5. Those slider designs range from a simple shape to a rather complex design. These sliders were borrowed from CML PhD students or from CMLAir users in the HDD industry.

5.0.1 GPU Dynamic 5

Looking at figures 5.1a and 5.1b, we can clearly see that using GPU Dynamic code gives much faster speed of convergence. Furthermore, looking at figures 5.2a and 5.2b, we can see that Fly-Height (column 2), Pitch (column 3) and Roll (column 4) are exactly identical in both cases. The Dynamic 5 is faster than Dynamic 4 and gives same output results as Dynamic 4. Figures 5.3a and 5.3b show the simulations' final outputs for both Dynamic 5 and Dynamic 4 using slider 6. We can also notice that there is big difference in speed improvement between slider 6 design shown in figure 5.7b and the rest of sliders. In slider 6 case we have achieved 150% or 2.5X speed improvement as shown in figure 5.1b. In order to better understand the reason behind the difference in performance between the different slider designs, we have to say that slider 6 shown in figure 5.7b has unique features and etching depth compared to other sliders. Hence the difference in performance.

5.0.2 GPU TFC 2

For testing GPU TFC 2 code, we used three different sliders:

- Case 1 with slider 7 shown in figure 5.7c.
- Case 2 which is similar in complexity to case 1.
- Case 3 with slider 8 shown in figure 5.7d.

Different slider designs or different heater power settings gave different execution times and outputs. Figures 5.4a, 5.4b and 5.4c show those effects. Speed improvements range from 20% to 103% faster than the TFC version 1.0. The higher the heater power, the bigger the difference found in execution times between the GPU and CPU. The slider in case 2 showed

the biggest speed improvement. In this case, TFC v1.0 took 61 minutes to get the converged final Fly-Height, while GPU TFC took only 30 minutes when 25mW heater power is applied. This translates to 103% or 2X speed enhancement.

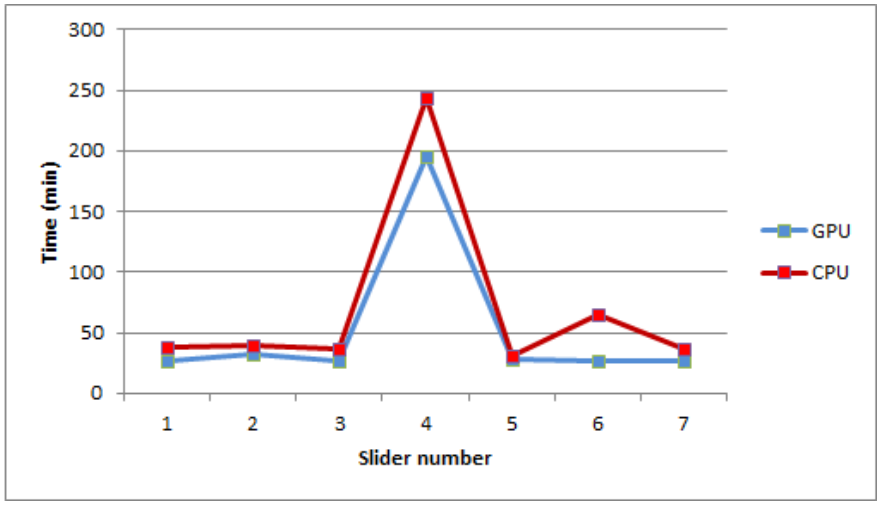
5.0.3 GPU Quick 5

During the process of benchmarking the GPU Quick 5, we used both GPU cards: Tesla K20 and GeForce GTX 680 to show the speed enhancement over CPU, and the relationship between performance and price of a GPU. Figures 5.5a, 5.5b and 5.5c illustrate those speed enhancements and performance/price relationships.

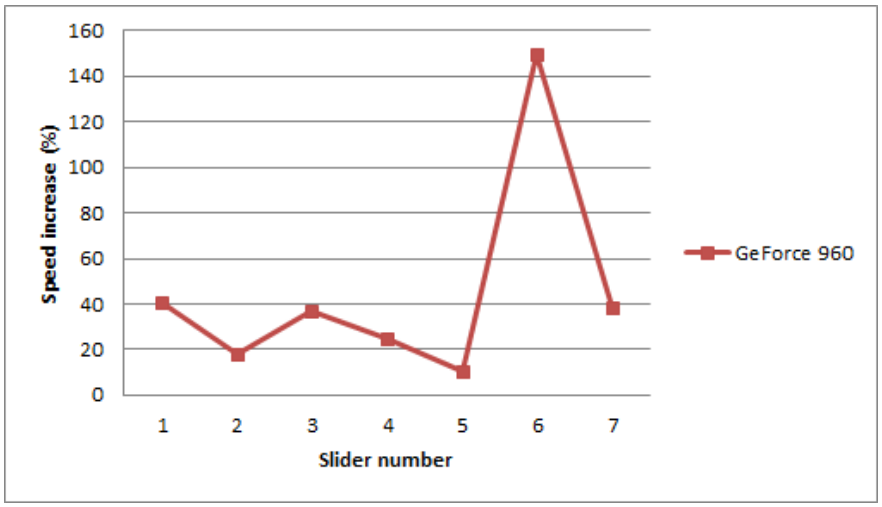
As you can see in the case using slider 1, the reason why there was no speed improvement or 0% speed increase is because the slider is not complex in design. This means both the CPU and GPU solvers are solving the problem so fast that you cannot see any difference in execution time. While for other complex sliders, such as slider 3 and 5, both have many rails/etching depths. This caused the high-speed improvement of 200% and 150% respectively when using Tesla K20 card. Hence, GPU is recommended for complex slider designs, which is the common case in the new hard disk drives.

With some of the complex slider designs, there is a noticeable difference in Quick 5 computation time as shown in figure 5.5a. This difference can be explained by looking at the desktop computer used to run Tesla K20 and GeForce GTX 680 GPUs. Since the Tesla K20 card installed on a desktop tower with 2.2 GHz CPU clock slower than the 3.4 GHz Intel i7 inside the Dell XPS8300. This explains why there is bigger difference between Tesla-CPU in figure 5.5a compared to the difference between GeForce-CPU in figure 5.5b.

Ultimately, according to the discussions and results in the previously mentioned tables/figures, we have concluded that an affordable $450 GeForce GTX 680 or $220 GeForce GTX 960 GPUs can perform faster than core i7 CPU. Also, GeForce GTX 680 is relatively comparable in performance to a $3,600 Tesla K20 GPU. With that fact, we eliminated the need for a higher priced/high-end GPU and made speed enhancements using GeForce family GPUs desirable.

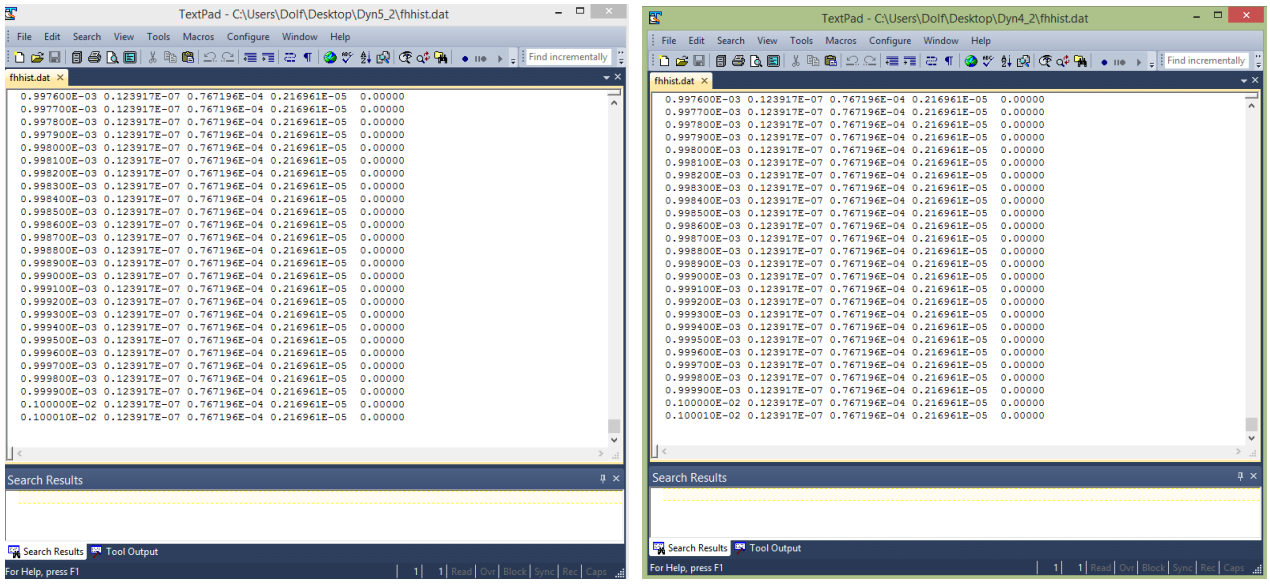


(a) GeForce 960 vs Dynamic 4



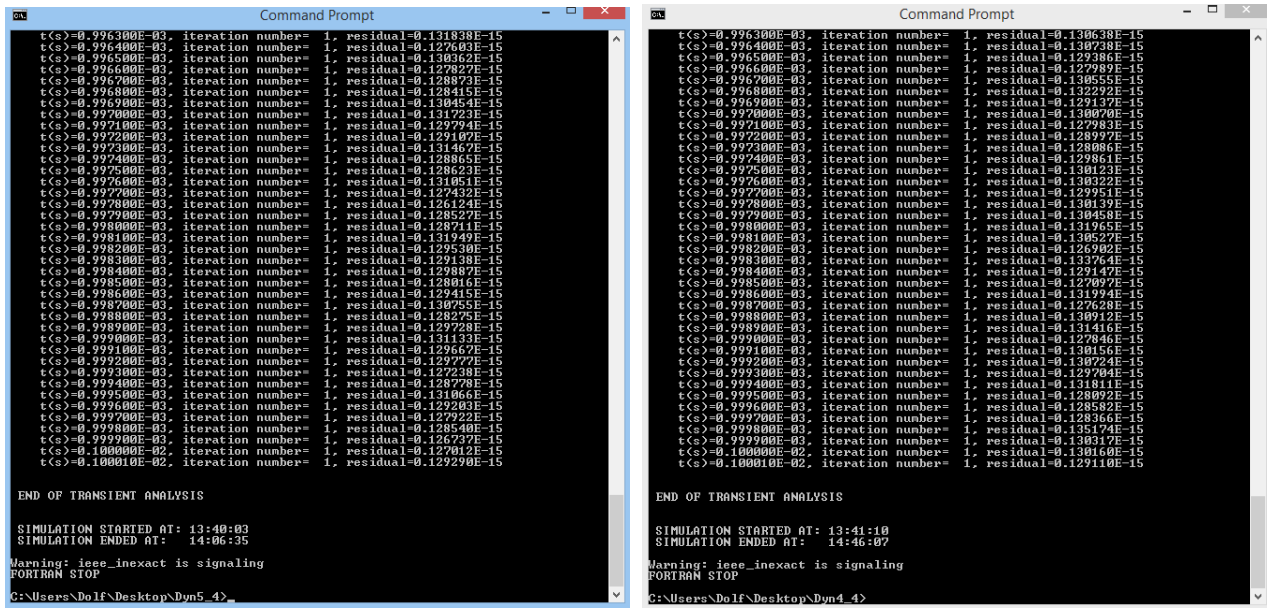
(b) Speed increase

Figure 5.1: GPU Dynamic 5 vs CPU



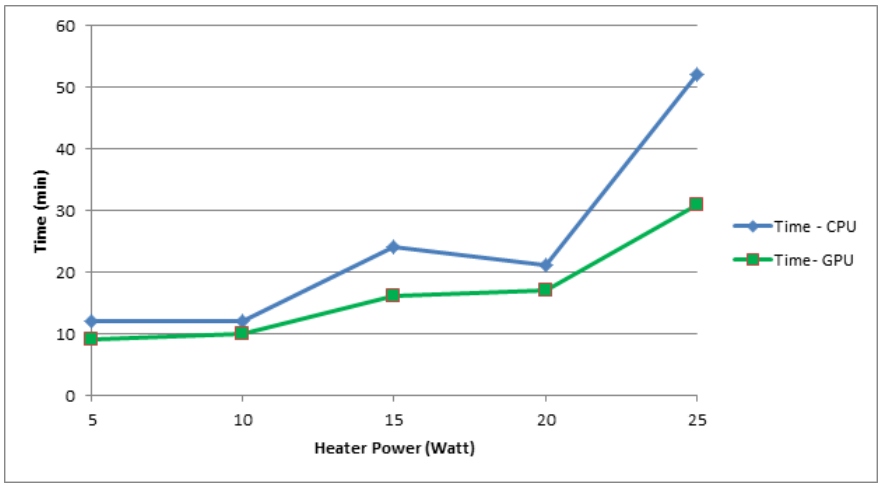
(a) Dynamic 5 (b) Dynamic 4

Figure 5.2: GPU Dynamic 5 vs CPU - Fly-Height history using Slider 6

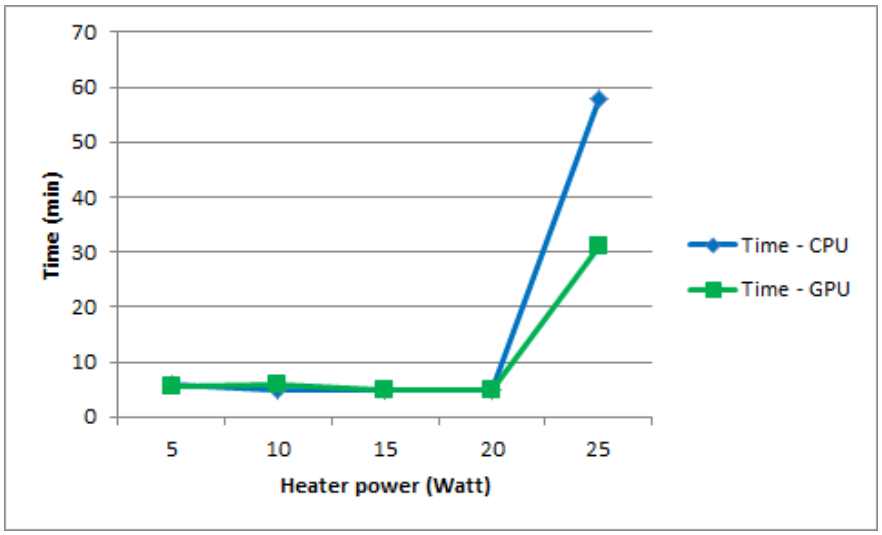


(a) Dynamic 5 (b) Dynamic 4

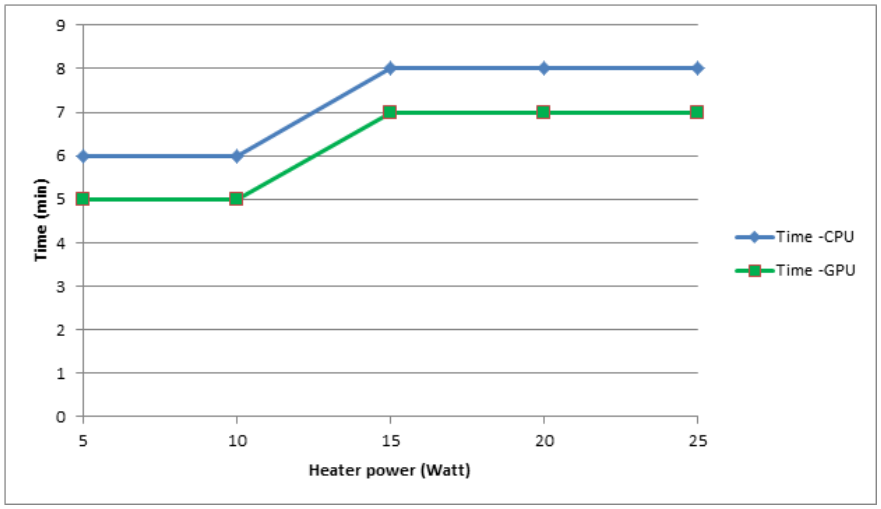
Figure 5.3: GPU Dynamic 5 vs CPU - simulation output using Slider 6



(a) Slider 7 design

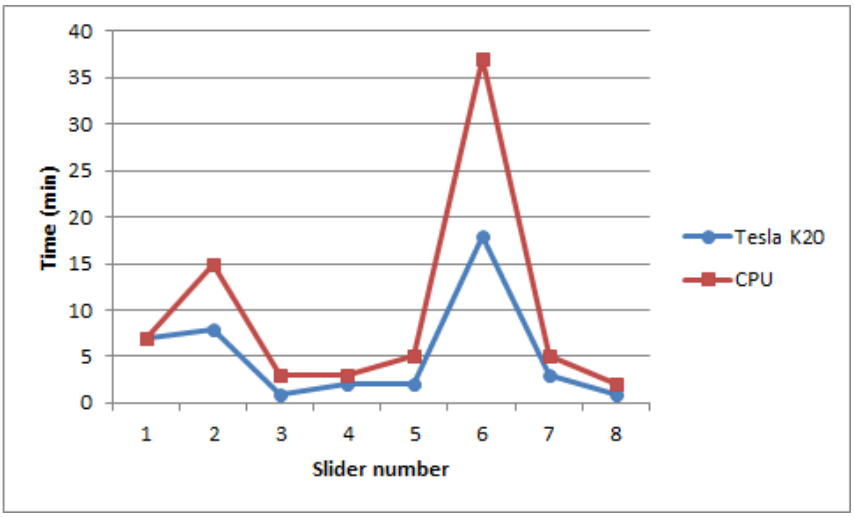


(b) Slider case 2

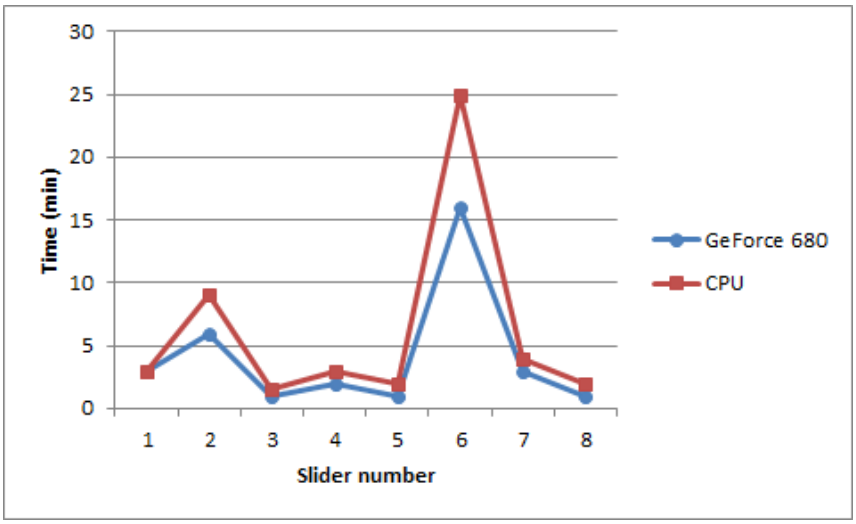


(c) Slider 8 design

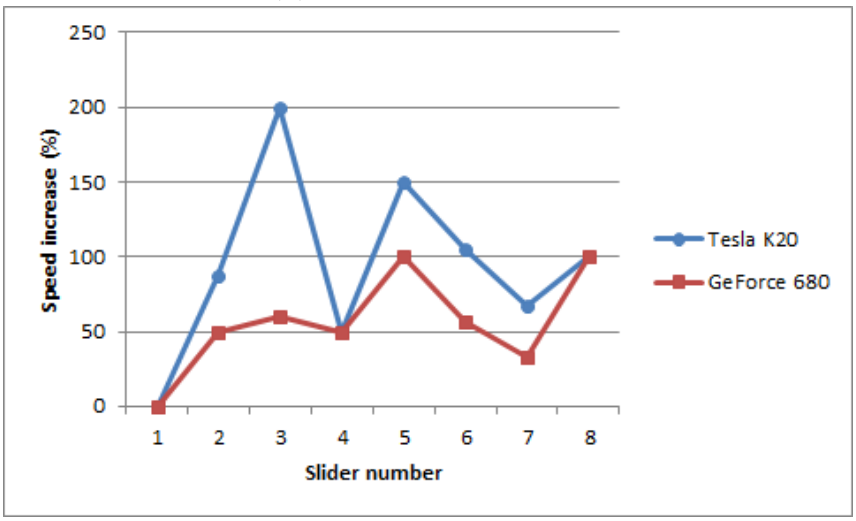
Figure 5.4: GPU TFC 2 vs CPU



(a) Tesla K20 vs CPU



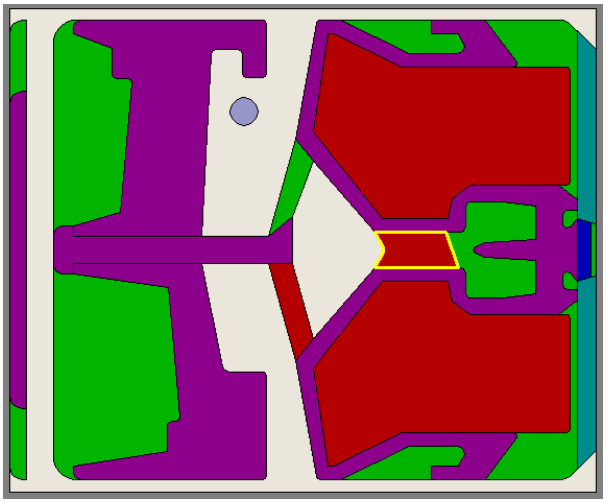
(b) GeForce 680 vs CPU



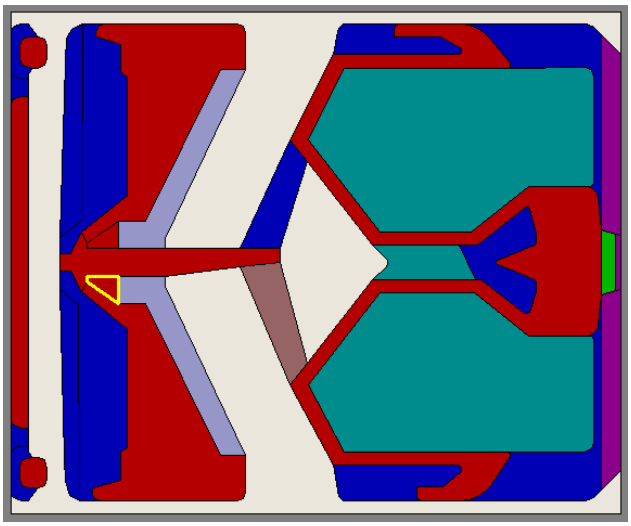
(c) Tesla K20 vs GeForce GTX 680

Figure 5.5: GPU Quick 5 vs CPU

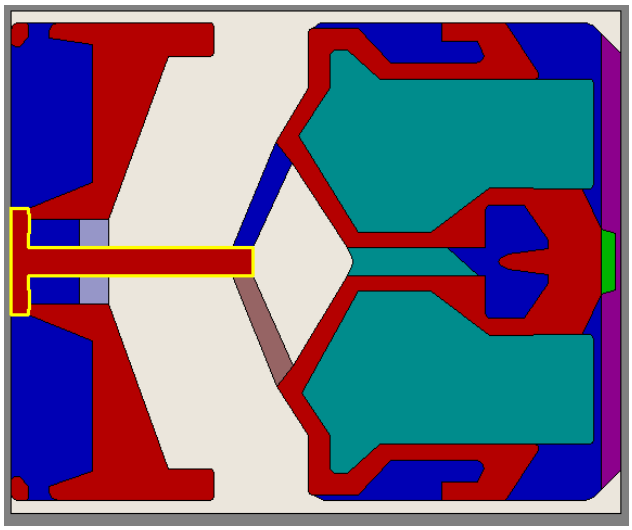
(a) Slider 1 design



(b) Slider 2 design

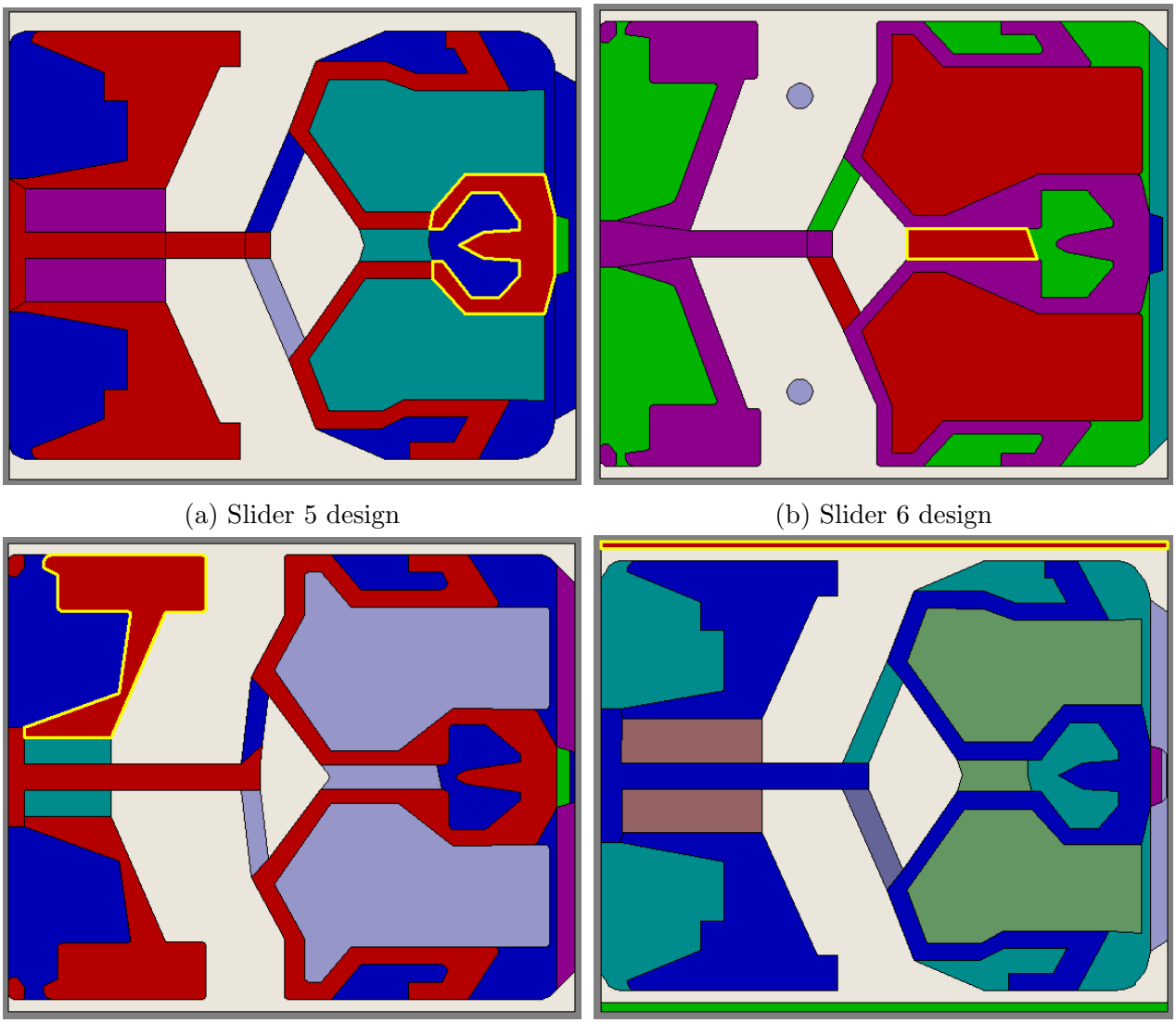


(c) Slider 3 design



(d) Slider 4 design

Figure 5.6: Slider designs: 1 to 4



(a) Slider 5 design

(b) Slider 6 design

(c) Slider 7 design

(d) Slider 8 design

Figure 5.7: Slider designs: 5 to 8

Chapter 6

Conclusion and Future Work

6.1 Conclusion

Looking closely at performance results of the different experiments discussed in the previous chapter, we concretely concluded that using GPUs is the optimum solution for faster ABS design software. Moreover, it gives you several folds of speed enhancement, ranging from 20% up to 3X. Additionally, the study showed that a GPU could be a more feasible choice than using multiple CPUs when it comes to price and ease of programming. Finally, the experiment showed that medium priced GeForce family GPUs could perform as well as the high-end Tesla family GPU cards with the speed enhancements.

6.2 Future Work

Even though we were able to demonstrate a 3X speed improvement with Quick 5, 2X with TFC 2 and a 2.5X with Dynamic 5, there is more that can be done in the future to obtain an even better performance including:

- Port in more portions of the code to run in parallel and increase performance.
- Modify the new code to be able to run on multiple GPUs while keeping cost effectiveness into account at all times.
- Forge the matrix factorization algorithms of CMLAir software into a more efficient ones.

Bibliography

[1] Sha Lu. *Numerical Simulation of Slider Air Bearings*. PhD Thesis, University of California at Berkeley, 1997.

[2] Suhas Patankar. *Numerical Heat Transfer and Fluid Flow*. Hemisphere, 1980.

[3] Tholfaqar Mardan Al Waheed and David B. Bogy. *Quick 5 user manual, A GPU version of CMLAir*. University of California at Berkeley, 2013.

[4] Yong Hu. *Head-Disk-Suspension Dynamics*. PhD Thesis, University of California at Berkeley, 1996.

[5] Tholfaqar Mardan Al Waheed and David B. Bogy. *Dynamic 5 user manual, A GPU version of CMLAir*. University of California at Berkeley, 2016.

[6] Jinglin Zheng. *Dynamics and Stability of Thermal Flying-height Control Sliders in Hard Disk Drives*. PhD Thesis, University of California at Berkeley, 2012.

[7] Jinglin Zheng and David B. Bogy. *Thermal Fly-Height Control TFC code user's manual*. University of California at Berkeley, 2009.

[8] Tholfaqar Mardan Al Waheed and David B. Bogy. *CML GPU TFC Code User's Manual*. University of California at Berkeley, 2014.