Oregon State University

CUDA-based Parallel Programming Benchmarks

Computer Science Capstone Project

Team Members:
Wade Cline
Taylor Christensen
Andrew Van Delden
Garrett Fleenor

Clients:
David Zier
Rogen Allen

June 9, 2012

Contents

1 Introduction

Historically, software programmers have focused on programming software for the Central Processing Unit (CPU). While this works fine for many applications, many computers have a powerful Graphics Processing Unit (GPU) that sits idle while the CPU runs programs. Furthermore, the GPU is vastly superior to the CPU for performing certain tasks. Part of the reason that software programmers have not taken advantage of the GPU in their programs is the inaccessibility of the GPU. In the past, programmers who wanted to use the GPU for general-purpose programming had to rig graphics APIs such as OpenGL in an ad hoc manner in order to program for general-purpose computations. In order to remedy this situation, NVIDIA made Compute Unified Device Architecture (CUDA), an Application Programming Interface (API) that allows the programmer to easily interact with the graphics card for general-purposes computations.

This project was requested by David Zier of NVIDIA Corporation as part of an effort to develop benchmarks for CUDA. These benchmarks will show potential performance gains from using CUDA and will provide a metric for measuring the performance between different cards and CPUs. By showing the power of CUDA, more programmers will become aware of its performance benefits and will use it in order to develop faster applications.

The clients for this project were David Zier and Roger Allen. David Zier managed the project and provided feedback and ideas. Roger Allen provided technical assistance and helped analyze the technical aspects of working with CUDA. Together the two struck a good balance of both managerial and technical expertise that helped to steer this project in the right direction.

The team members were Wade Cline, Taylor Christensen, Andrew Van Delden, and Garrett Fleenor. Wade Cline managed the team members, implemented code, and did research for the project. Taylor Christensen advised Wade on project management, implemented code, and also did research. Andrew Van Delden and Garrett Fleenor were assigned to work on implementing code and do some of the required coursework.

2 Original Requirements Document

Note: Original document converted from LaTeX source. Please see the enclosed CD to find an exact copy of the original Requirements Document.

2.1 Team

2.1.1 Name

World! Hello,

The team name is a play on words of a common problem of parallel programming, serialization, coupled with the de facto standard introductory program that prints, “Hello, world!”.

2.1.2 Members

Christensen, Taylor – christay@onid.oregonstate.edu
Cline, Wade – clinew@onid.orst.edu
Fleenor, Garrett – fleenorg@onid.oregonstate.edu
Van Delden, Andrew – vandelan@onid.oregonstate.edu

2.1.3 Clients

Zier, David – dzier@nvidia.com
Allen, Roger – RAllen@nvidia.com oistorically, Central Processing Unit (CPU) clock speed has doubled every couple of years. This has allowed software developers to wait a couple of years to see performance gains in program speed. However, recent trends have been breaking this pattern; instead of getting faster clock speed computers have been getting more processi

2.2 Project

2.2.1 Introduction

CUDA is a massively parallel programing architecture built into NVIDIA graphics cards that is designed to harness the graphics pipeline for general purpose computing. Unlike traditional parallel processing, where a device has two

or four powerful cores, GPUs have traditionally had hundreds of small cores that are designed for rapid mathematical calculations. CUDA's purpose is to bring the power of the CPU to general computing by providing the benefits if massively parallel processing to non-graphical applications.

2.2.2 Project Description

Our overall task is to create a set of application benchmarks that clearly show performance gains in CUDA compared to traditional parallel programing APIs, such as pthreads. In order to do this, different programming paradigms from single-core CPUs, multi-core CPUs, and massively parallel GPU architectures must be implemented on various applications. The applications can then be benchmarked for performance and the results analyzed in order to show the performance gains from CUDA.

In order to get our feet wet and develop a good understanding of CUDA, we will implement one of the Princeston Application Repository for Shared-Memory Computers (PARSEC) benchmarks. This should be a relatively simple benchmark to implement and will give each group member some experience before working on an actual application.

The application we have chosen for this project is TrueCrypt. TrueCrypt is free, open-source disk encryption software that runs on Windows, Mac, and Linux. Truecrypt implements three encryption algorithms that we wish to parallelize. The algorithms are the Advanced Encryption Standard (AES), Serpent, and Twofish. The algorithms were originally submitted as part of the Advanced Encryption Standard process in order to ratify a standard encyprtion algorithm. The process was also more open and transparent than its predecesor in order to foster confidence in the cryptographic community. Each algorithm that has been implemented in Truecrypt was at least a finalist in the process, with the AES algorithm being the winner. The openness and ubiquity of these algorithms make them good candidates for parallelization using CUDA.

2.2.3 Requirements

The project is open-source in order to publically show the performance benefits of CUDA. The language used will be either C or C++ in order to utilize the CUDA API. Completion of the requirements is divided into three separate phases. The phases involve finding applications for benchmarking, modifying the applications for benchmarking, then reporting on the results from benchmarking the applications. These tasks will each be focused on in the Fall, Winter, and Spring Terms, respectively. The Spring Term will also focus on the end stages of application development.

Finding applications for benchmarking will consist of research into various programs that benefit from the parallelization offered by CUDA. The applications that are acceptable will likely be dependent on specific algorithms implemented within the program. Algorithms that may benefit from CUDA include compression/decompression, encryption, matrix multiplication, and certain Artificial Intelligence (AI) algorithms. Programs that appear to contain applications that might benefit from CUDA will then be presented to the client and evaluated by the client based on the client's expectations and feasibility.

Modification of applications will then be done by the group members. How the applications are modified will depend on the difficulty and parallelizability of development. If the applications are easy to develop on, each member will work on a different application. Conversely, if the applications are difficult to develop on, members will likely divide the application into different sections and assign those sections to different group members.

Reporting on the results from benchmarking will consist of data gathering, data analysis, and report development. Specific attention will be paid in regards to comparative benchamrking and analysis. This will showcase the capabilities of CUDA in a scientific, controlled environment. Data gathering will likely take place in the Computer Graphics lab over a couple of days, depending on access to resources and the time it takes to benchmark the applications. Data analysis will consist of manual or automated analysis by the team. The team will decide on a report format and which sections of the report will be done by which team member. Each team meber will work on their section of the report in parallel. Report development will also take into account the pros and cons of the previous group's work, with the intention of all development being completed by the time of the Engineering Expo.

The project will be considered complete once the applications have been found, modified, and all reports, including documentation, have been written and finalized.

2.2.4 Versions

Benchmark implementation will determine application versions. The first version will consist of a simple, easy-to-implement benchmark in order to familiarize the group with CUDA. Versions two to four will then focus on

parallelizing TrueCrypt. Each algorithm within TrueCrypt may be implemented as a separate version, thus providing stepping stones for project completion. The fifth version will then consist of developing the wrappers for benchmarking and data output.

2.2.5 Design

Each benchmark will likely be a separate executable, and the entire suite will be run with a script. Splitting up each benchmark will allow for a more modular product and may make it so that each benchmark may be run in parallel.

2.2.6 Specific Tasks to be Undertaken

We plan to start with a simple benchmark in order to get everyone acquainted with the software by implementing a benchmark from the PARSEC benchmarking suite. Since our team is relatively inexperienced when it comes to parallel programming, it would be best to implement an easy benchmark so that everyone becomes familiar with massively parallel programming.

Next, work will begin on TrueCrypt. Each algorithm from TrueCrypt can then be implemented separately in a different version of the software. Once all the algorithms have been implemented, the next step will involve creating a wrapper in order to benchmark the application and get the output data. Last, the data will be gathered, analyzed, and placed into a presentable format.

2.2.7 Risk Assessment

One possible problem we may encounter is that one of the encryption algorithms is written in such a manner that it is either impossible to parallelize or is already parallelized. This would make our efforts to port the TrueCrypt encryption and decryption process to the CUDA architecture a moot point. However, our initial look through the source code of the TrueCrypt software led us to believe that this was not the case. If necessary, we may look into other open source encryption and decryption software packages, such as gzip, in case the TrueCrypt software should fail us.

Possible warning signs of already implemented parallelization are very easy to find. Signs of the usage of either openMP, pthreads or any of the common, or even uncommon, threading libraries are easy to spot and trace. If we spot threaded activity in the encryption and decryption algorithms then there is a very good chance that they are already using a multithreaded algorithm and that a CUDA implementation would be trivial from that point. Should that happen we will need to fall back onto one of the backup encryption projects and start from scratch.

Another possible problem we may encounter is getting people trained up on parallel programming fast enough. Parallel programming can be difficult for some people to grasp, and may take some time to fully understand. We only have a little time to get trained up on parallel programming and are all taking classes as well as trying to get trained on this subject. Should one of us start to fall behind in learning how to parallel program the best course of action would be for the others in the group to help teach him how to parallel program. This may be done through a method such as extreme programming, as this method tends to benefit both the person falling behind and the people teaching the person who is falling behind.

One way to look out for this happening is to meet on a regular basis and discuss how our learning is going and what we are having trouble understanding. This way communication is scheduled, regular, and provides a forum for discussion of ideas and frustrations. Another way to look out for problems is to do small coding projects to test what we have learned.

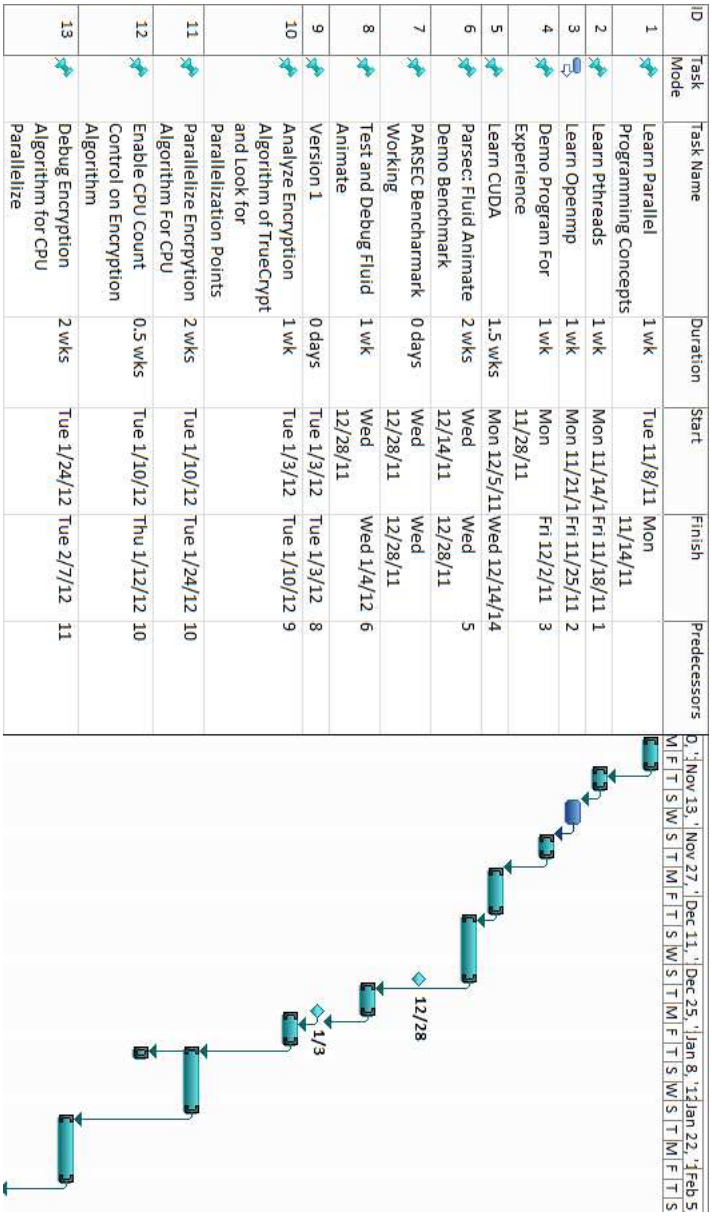
There is also a risk of not being able to read a large enough amount of data from disk in order to keep the GPU fed. Disk Input and Output tends to take long periods of time compared to processing the data. The algorithms are also highly optimized. It is possible that encryption/decryption would complete on the GPU long before another block of data could be read. Thus the goal of decreasing the encyrption/decryption time by processing the data on the GPU would no longer be possible.

2.2.8 Testing

There are several ways to see if our code works. One way is to simply test the encryption algorithm after modification and see if the algorithm still properly encrypts and decrypts files. A more thorough and exhaustive method of testing would be to first disable threading and test to see if the algorithm still encrypts and decrypts properly. We could then slowly enable more and more threads on the CPU and then finally enable CUDA to see if the algorithm still properly encrypts and decrypts. We can also perform some internal checks on the algorithm if we make any

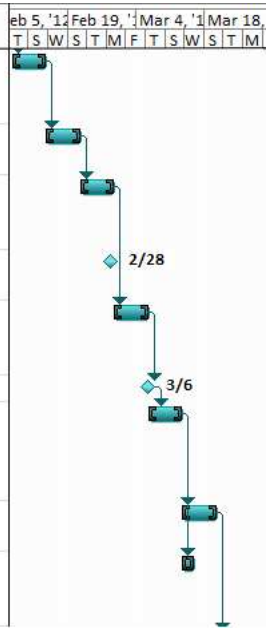
changes; the internal checks can be verified with the original implementation in order to ensure correctness of our implementation.

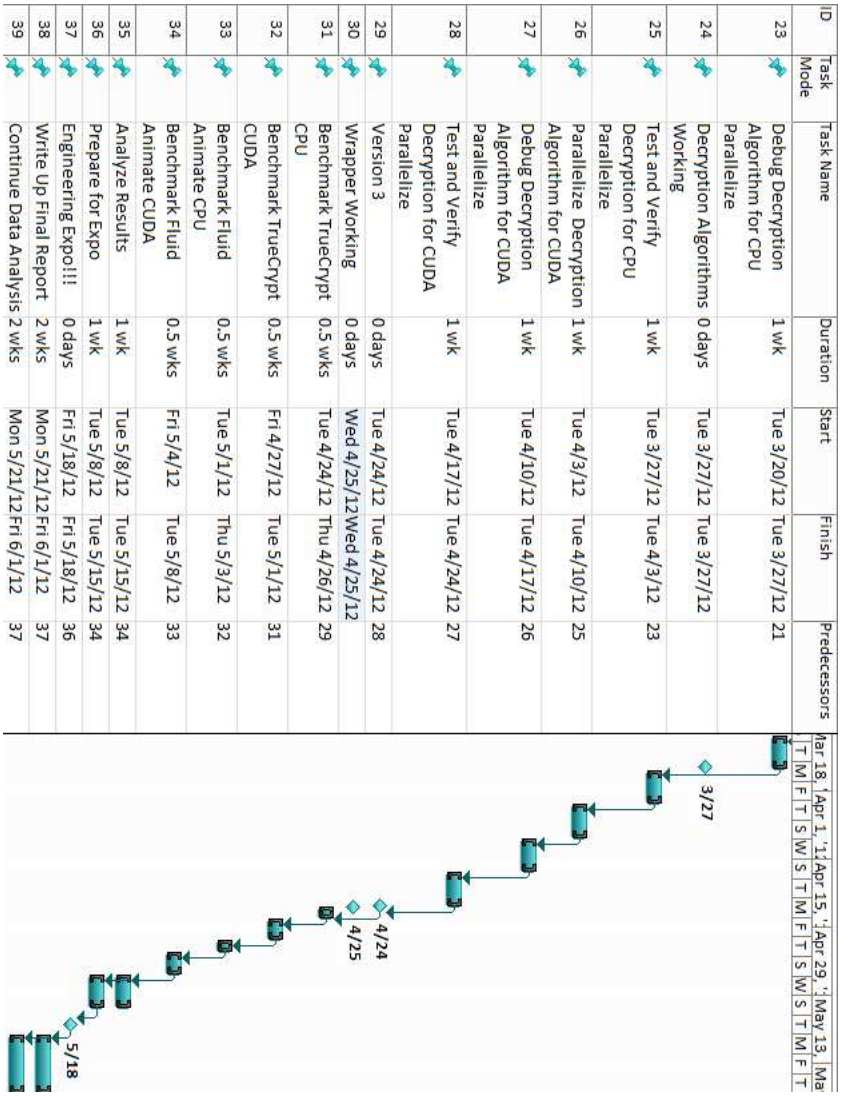
2.2.9 Preliminary Timetable



ID	Task Mode	Task Name	Duration	Start	Finish	Predecessors
1		Learn Parallel Programming Concepts	1 wk	Tue 11/8/11	Mon 11/14/11	
2		Learn Pthreads	1 wk	Mon 11/14/11	Fri 11/18/11	1
3		Learn Openmp	1 wk	Mon 11/21/11	Fri 11/25/11	2
4		Demo Program For Experience	1 wk	Mon 11/28/11	Fri 12/2/11	3
5		Learn CUDA	1.5 wks	Mon 12/5/11	Wed 12/14/14	
6		Parsec: Fluid Animate Demo Benchmark	2 wks	Wed 12/14/11	Wed 12/28/11	5
7		PARSEC Benchmark Working	0 days	Wed 12/28/11	Wed 12/28/11	
8		Test and Debug Fluid Animate	1 wk	Wed 12/28/11	Wed 1/4/12	6
9		Version 1	0 days	Tue 1/3/12	Tue 1/3/12	8
10		Analyze Encryption Algorithm of TrueCrypt and Look for Parallelization Points	1 wk	Tue 1/3/12	Tue 1/10/12	9
11		Parallelize Encrytion Algorithm For CPU	2 wks	Tue 1/10/12	Tue 1/24/12	10
12		Enable CPU Count Control on Encryption Algorithm	0.5 wks	Tue 1/10/12	Thu 1/12/12	10
13		Debug Encryption Algorithm for CPU Parallelize	2 wks	Tue 1/24/12	Tue 2/7/12	11

ID	Task Mode	Task Name	Duration	Start	Finish	Predecessors
14		Test and Verify Encryption for CPU Parallelize	1 wk	Tue 2/7/12	Tue 2/14/12	13
15		Parallelize Encryption Algorithm for CUDA	1 wk	Tue 2/14/12	Tue 2/21/12	14
16		Debug Encryption Algorithm for CUDA Parallelize	1 wk	Tue 2/21/12	Tue 2/28/12	15
17		Encryption Algorithms Working	0 days	Tue 2/28/12	Tue 2/28/12	
18		Test and Verify Encryption for CUDA Parallelize	1 wk	Tue 2/28/12	Tue 3/6/12	16
19		Version 2	0 days	Tue 3/6/12	Tue 3/6/12	18
20		Analyze Decryption Algorithm of TrueCrypt and Look for Parallelization Points	1 wk	Tue 3/6/12	Tue 3/13/12	19
21		Parallelize Decrpytion Algorithm For CPU	1 wk	Tue 3/13/12	Tue 3/20/12	20
22		Enable CPU Count Control on Decryption Algorithm	0.5 wks	Tue 3/13/12	Thu 3/15/12	20





ID	Task Mode	Task Name	Duration	Start	Finish	Predecessors
23		Debug Decryption Algorithm for CPU Parallelize	1 wk	Tue 3/20/12	Tue 3/27/12	21
24		Decryption Algorithms Working	0 days	Tue 3/27/12	Tue 3/27/12	
25		Test and Verify Decryption for CPU Parallelize	1 wk	Tue 3/27/12	Tue 4/3/12	23
26		Parallelize Decryption Algorithm for CUDA	1 wk	Tue 4/3/12	Tue 4/10/12	25
27		Debug Decryption Algorithm for CUDA Parallelize	1 wk	Tue 4/10/12	Tue 4/17/12	26
28		Test and Verify Decryption for CUDA Parallelize	1 wk	Tue 4/17/12	Tue 4/24/12	27
29		Version 3	0 days	Tue 4/24/12	Tue 4/24/12	28
30		Wrapper Working	0 days	Wed 4/25/12	Wed 4/25/12	
31		Benchmark TrueCrypt CPU	0.5 wks	Tue 4/24/12	Thu 4/26/12	29
32		Benchmark TrueCrypt CUDA	0.5 wks	Fri 4/27/12	Tue 5/1/12	31
33		Benchmark Fluid Animate CPU	0.5 wks	Tue 5/1/12	Thu 5/3/12	32
34		Benchmark Fluid Animate CUDA	0.5 wks	Fri 5/4/12	Tue 5/8/12	33
35		Analyze Results	1 wk	Tue 5/8/12	Tue 5/15/12	34
36		Prepare for Expo	1 wk	Tue 5/8/12	Tue 5/15/12	34
37		Engineering Expo!!!	0 days	Fri 5/18/12	Fri 5/18/12	36
38		Write Up Final Report	2 wks	Mon 5/21/12	Fri 6/1/12	37
39		Continue Data Analysis	2 wks	Mon 5/21/12	Fri 6/1/12	37

2.2.10 Roles

The entire team will be involved in the coding, testing and debugging process; however, there are some individual duties that are assigned. Wade is in charge of organizing the weekly group meetings and making sure that we have at least a minimal agenda during meetings. Garret and Taylor will be in charge of organizing research on the encryption methods used in TrueCrypt. Andrew will be in charge of organizing our software suite and keeping our files backed up and secure.

2.2.11 Integration Plan

Our team will be working together for the different programs that we will be benchmarking. As each program is completed, they will be implemented into the entire program suite. Using team programming and an agile process for each benchmark will facilitate easy data integration.

2.2.12 User Interface

No real user interface is necessary for this project. Shell scripts will be used in order to run the benchmarks. The output data from the program will then be graphed by the developers with the appropriate graphing program(s) and shared with the interested parties.

2.3 Resources

- NVIDIA website: http://developer.nvidia.com/what-cuda
- 2010-2011 461 CUDA benchmark project.
- "CUDA by Example. An Introduction to General-Purpose GPU Programming" by Jason Sanders, Edward Kandrot.
- "An Introduction to Parallel Programming" by Peter S. Pacheco
- "Programming Massively Parrellel Processors" by David B. Kirk, Wen-mei W. Hwu

3 Project Changes

The project changed rather significantly as time went on. The PARSEC fluid animate benchmark turned out to be much more difficult to implement than originally expected, and the requirement was eventually dropped. It also turned out that the data pipeline for TrueCrypt was extremely complex and it was decided to take the implementation of the encryption algorithms out of TrueCrypt and benchmark just the implementation of the algorithms.

3.1 Requirements Status Table

Req. #	Requirement	Status	Comments
1	Learn Parallel Programming Concepts	Complete	-
2	Learn Pthreads	Deleted	Not used; OpenMP used instead.
3	Learn OpenMP	Complete	-
4	Demo Program for Experience	Deleted	Demo Program turned out to be quite difficult.
5	Learn CUDA	Complete	-
6	PARSEC: Fluid Animate Demo Benchmark	Deleted	Turned out much more complicated than expected.
7	PARSEC: Benchmark Working	Deleted	Explained above.
8	Test and Debug Fluid Animate	Deleted	Explained above.
9-29	Analyze, parallelize, debug, and test the encryption and decryption algorithms of TrueCrypt	Deleted	Directly using TrueCrypt turned out to be too complicated. Instead, the encryption/decryption algorithms were pulled from TrueCrypt for benchmarking purposes.
30	Parallelize Serpent Encryption Algorithm	Complete	-
31	Parallelize Twofish Encryption Algorithm	Complete	-
32	Parallelize Serpent Decryption Algorithm	Complete	-
33	Parallelize Twofish Decryption Algorithm	Complete	-
34	Test and Validate Parallelized Serpent Encryption/Decryption Algorithms	Complete	Verified by manual inspection (text) and sha256sum hash algorithms.
35	Test and Validate Parallelized Twofish Encryption/Decryption Algorithms	Complete	As above.
36	Finish PARSEC Fluid Animate Benchmark	Deleted	See #6.
37	Parallelize AES Encryption Algorithm	Deleted	Instead performed in-depth analysis of Serpent.
38	Parallelize AES Decryption Algorithm	Deleted	See above.
39	Test and Validate Parallelized AES Encryption/Decryption Algorithms	Deleted	See above.
40	Wrapper Working	Completed	

41	Benchmark TrueCrypt CPU	Modified	Benchmarks will be on individual algorithms.
42	Benchmark TrueCrypt CUDA	Modified	See above.
43	Benchmark PARSEC: Fluid Animate CPU	Deleted	See #6.
44	Benchmark PARSEC: Fluid Animate CUDA	Deleted	See #6.
45	Automated report	Completed	-
46	Statistical analysis	Completed	Sample mean, harmonic mean, and standard deviation.
47	Determine points for analysis	Complete	Spent most of the time analyzing Serpent Decrypt's slower performance on the CPU.
48	Perform analyses	Complete	Thoroughly analyzed Serpent; able to optimize for same-speed encrypt/decrypt on the GPU.
49	Prepare for Expo	Complete	
50	Engineering Expo	Complete	
51	Write Up Final Report	Complete	
52	Continue Data Analysis	Complete	

3.2 Final Gantt Chart

See Figures 1-2 to see the Final Gantt Chart for the project[1].

4 Weekly Reports

4.1 Week 04

Progress

Chosen team name: "World! Hello". Benchmarks have been brainstormed. At least one book containing related material has arrived; the others should arrive soon.

Problems

No significant problems have been encountered so far.

Plans

Action Items (AIs):
All:

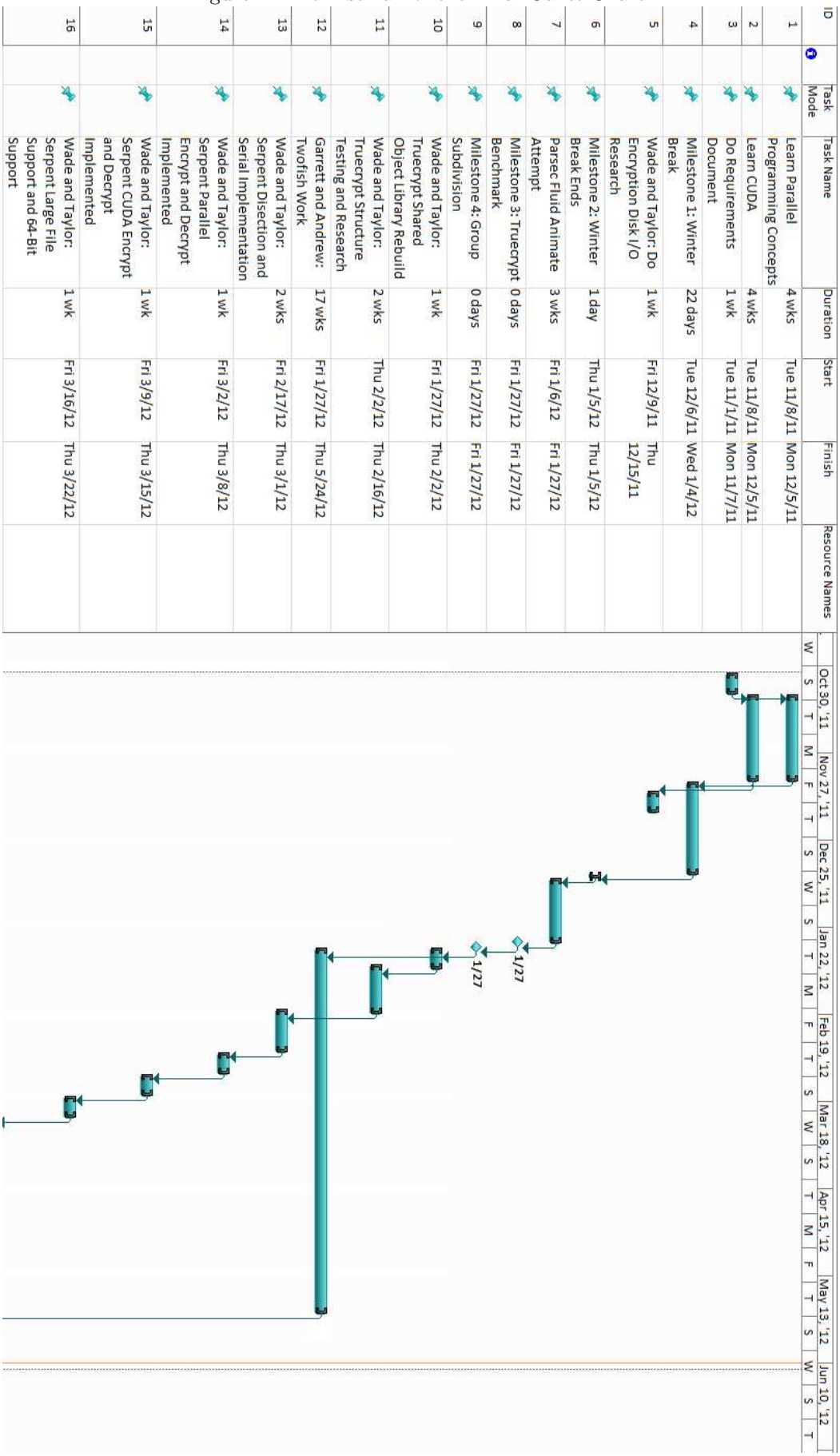
- Complete assigned items for the Requirements Document (not listed here).

Wade:

- Talk to professor bailey about getting keys for an exclusive resource storage area and making copies for group members.
- Update group with status of books when they arrive/are stored.
- Set up Meeting with David Zier for workweek 43.
- Create agenda.
- Set up Doodle poll.

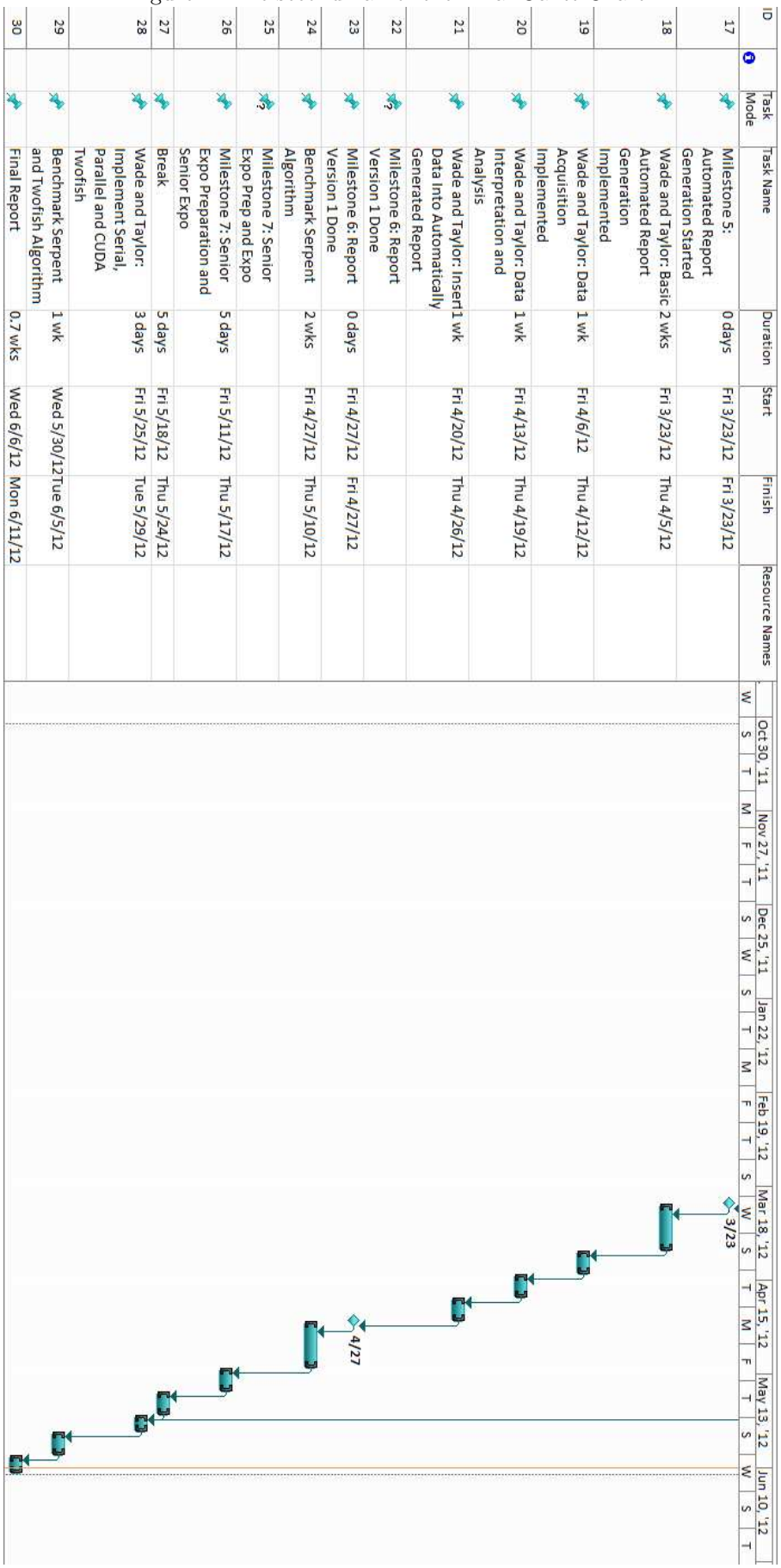
[1]Some parts of the image were modified in order to make the chart more accurate.

Figure 1: The first half of the Final Gantt Chart.



ID	Task Mode	Task Name	Duration	Start	Finish	Resource Names
1		Learn Parallel Programming Concepts	4 wks	Tue 11/8/11	Mon 12/5/11	
2		Learn CUDA	4 wks	Tue 11/8/11	Mon 12/5/11	
3		Do Requirements Document	1 wk	Tue 11/1/11	Mon 11/7/11	
4		Milestone 1: Winter Break	22 days	Tue 12/6/11	Wed 1/4/12	
5		Wade and Taylor: Do Encryption Disk I/O Research	1 wk	Fri 12/9/11	Thu 12/15/11	
6		Milestone 2: Winter Break Ends	1 day	Thu 1/5/12	Thu 1/5/12	
7		Parsec Fluid Animate Attempt	3 wks	Fri 1/6/12	Fri 1/27/12	
8		Milestone 3: Truecrypt Benchmark	0 days	Fri 1/27/12	Fri 1/27/12	
9		Milestone 4: Group Subdivision	0 days	Fri 1/27/12	Fri 1/27/12	
10		Wade and Taylor: Truecrypt Shared Object Library Rebuild	1 wk	Fri 1/27/12	Thu 2/2/12	
11		Wade and Taylor: Truecrypt Structure Testing and Research	2 wks	Thu 2/2/12	Thu 2/16/12	
12		Garrett and Andrew: Twofish Work	17 wks	Fri 1/27/12	Thu 5/24/12	
13		Wade and Taylor: Serpent Disection and Serial Implementation	2 wks	Fri 2/17/12	Thu 3/1/12	
14		Wade and Taylor: Serpent Parallel Encrypt and Decrypt Implemented	1 wk	Fri 3/2/12	Thu 3/8/12	
15		Wade and Taylor: Serpent CUDA Encrypt and Decrypt Implemented	1 wk	Fri 3/9/12	Thu 3/15/12	
16		Wade and Taylor: Serpent Large File Support and 64-Bit Support	1 wk	Fri 3/16/12	Thu 3/22/12	

Figure 2: The second half of the Final Gantt Chart.



4.2 Week 05

Progress

We had a teleconference with David Zier and worked out a list of benchmarks that we are going to implement for our project. We also decided who was going to do what parts for the requirements documents.

Problems

No major problems, our team seems to be on the same page. We just need to write the requirements document now.

Plans

We are going to implement some easy parallel programming benchmarks first to get our feet wet with CUDA and parallel programming in general, then tackle a harder project, like crypto.

4.3 Week 07

Progress

Requirements Document signed and completed.

Problems

No problems have been encountered.

Plans

Each member has been assigned a different task.

- *Wade*: Create CUDA Power-Point slides for team learning.
- *Garrett*: Set up GitHub repository and ensure each member has access.
- *Andrew*: Talk to Professor Bailey and work on T.R.I.P. assignment, taking into account presentation feedback from last year.
- *Taylor*: Test TrueCrypt data throughput. Determine if disk I/O is limited factor for speed in spinning hard disks.

4.4 Week 08

Progress

- *Andrew*: Presentation Slides – No progress yet.
- *Garrett*: Repository – Task completed; gave presentation to team on GitHub.
- *Wade*: CUDA Power-Point slides – 50% done reading.
- *Taylor*: TrueCrypt I/O encrypt/decrypt – Examining TrueCrypt source code.

Problems

Andrew unable to begin Presentation Slide work due to school; should not affect overall deadline. Wade slightly behind reading due to school; should be able to catch up due to a brief respite from schoolwork and due to Thanksgiving Break. Will be unable to meet next week due to Thanksgiving Break. Deadlines for CUDA Power-Point slides and Truecrypt I/O benchmark have been extended until next group meeting.

Plans

Andrew plans to begin Presentation Slides this coming week; will talk to Professor Bailey on Tuesday. Garrett assigned to help Taylor with Truecrypt I/O benchmark. Wade will continue reading and then create Power Point. Taylor will continue looking through and working on benchmarking Truecrypt source code.

4.5 Week 10

Progress

- *Andrew*: Presentation poster and TRIPS both completed successfully and on-time.
- *Garrett*: No progress made helping Taylor with Truecrypt I/O.
- *Wade*: Finished reading "Massively Parallel Programming".
- *Taylor*: Solid-State Drive (SSD) on its way.

Problems

- Truecrypt Disk I/O benchmark: Unfinished due to school work.
- *Wade*: CUDA slides incomplete (due to other schoolwork).

Plans

- *All*:
  - Anyone who can help Taylor complete disk I/O benchmarking early Winter Break, should.
  - Implement PARSEC benchmark in CUDA by the end of the first week of Winter Term.
  - Learn Pthreads.
  - Learn OpenMP.
  - Learn CUDA.
- *Taylor*:
  - Track down the SSD.
  - Finish Truecrypt disk I/O benchmarking.
- *Wade*:
  - Complete this week's P3.
  - E-mail clients in order to set up a meeting a couple days before the first day of Winter Term.
  - Talk to Professor Bailey about the Tesla graphics card.
  - Finish CUDA Power Point slides.
  - E-mail clients with aforementioned slides' topics.
  - Assist Taylor with Truecrypt disk I/O benchmarking.

4.6 Week 11

Progress

- SSD drive obtained by Taylor.
- I/O benchmarking completed by Taylor and Wade.
- CUDA slides finished by Wade.
- Client-meeting set up.

Problems

- No members were able to complete the CUDA implementation of the PARSEC fluid animate benchmark.
- Progress so far:
  - *Taylor* – Compiled, working on CUDA benchmark.
  - *Wade* – Compiled; working on CUDA benchmark.
  - *Garrett* – No progress.
  - *Andrew* – No progress.

Plans

- Continue to work on a CUDA implementation of the PARSEC fluid animate benchmark. Now due by the client meeting on Friday. All members will report on their progress at that time.
- Andrew will complete the Project Significance Document by its specified due date.

4.7 Week 12

Progress

Progress of group members is currently unknown due to meeting cancellation.

Problems

Flood caused group meeting cancellation.

Plans

- Obtain updates from group members via e-mail.
- Re-schedule meeting with client.

4.8 Week 13

Progress

- Team experienced major technical difficulties with hardware.
- Status as follows:
  - *Andrew*: Learned C++; unable to make as much progress as hoped for on PARSEC benchmark.
  - *Garrett*: Has environment set up; difficulty setting up environment on laptop.
  - *Taylor*: Got working CUDA kernel for PARSEC.
  - *Wade*: Got working CUDA kernel for PARSEC.
- Most progress: Halfway through implementation.

Problems

- Taylor wished to express concerns about group performance.
  - Wade needs to lighten up.
  - Garrett needs to cut something from his schedule; unable to find time for CUDA work; has lots of classes + work; unfair to team; something needs to go.
  - Andrew seems kind of lost in everything; does not fully comprehend project; needs some assistance in getting up to speed; needs to work with partner.

Plans
AIs:

- Plan to team up and work on the two encryption algorithms in paralell in two separate groups:
  - Serpent algorithm: Wade and Taylor
  - Twofish algorithm: Andrew and Garrett.
- Wade and Taylor will work on PARSEC after Seprent algorithm.
- Things to keep in mind:
  - Be consistent with benchmarking.
  - Full list of hardware.
  - Command line utility to get hardware and measurements.
  - Detect when you're sending data and getting data back from CUDA.
  - Multiprocessors having issues with time stamps, be aware of issue.

4.9 Week 14

Progress

Wade & Taylor:

- Printed off Serpent specification and searched for parallelization points.
- Found parallelization points in computation of 32 "S-Boxes", in which the 128-bit block cypher is divided into 4 32-bit words.
- May be able to thread each word for massive parallelization.

Andrew & Garrett:

- Met in CGL, found "old CUDA version" on computers there.

Problems

Wade & Taylor:

- Wade has 3 midterms next week; Will be busy until Friday 3 p.m.

Andrew & Garrett:

- Waiting for new CUDA version.

Plans

Wade & Taylor:

- Wade will focus on midterms this week, and will begin meeting with Taylor to code starting Friday at 3 p.m.

Andrew & Garrett:

- Asked Professor Bailey to update CUDA version.

4.10 Week 15

Progress

Wade & Taylor:

- All Truecrypt static object libraries converted into shared object libraries for CUDA compatibility.
- Truecrypt able to compile Serpent algorithm as .cu file.
- Set up Truecrypt – Serpent Branch on GitHub.

Andrew & Garrett:

- Downloaded TrueCrypt in CGL (Computer Graphics Lab).

Problems

Wade & Taylor:

- Truecrypt had bizarre linking errors; turned out to be static object libraries within Truecrypt (CUDA requires shared object libraries).

Andrew & Garrett:

- CGEL running 3.2.1, need 4.1; have e-mailed professor Bailey twice.

Plans

Wade & Taylor:

- Begin implementing parallelized Serpent algorithm.

Andrew & Garrett:

- Waiting for CUDA update in CGL.

4.11 Week 17

Progress

Andrew & Garrett:

- Running code outside of TrueCrypt.
- Getting ready to start porting.

Wade & Taylor:

- Worked on deciphering TrueCrypt API.

Problems

Andrew & Garrett:

- Garrett out of town week after next week.

Wade & Taylor:

- TrueCrypt's API does not lend to CUDA.

Plans

Andrew & Garrett:

- Have Twofish running in CUDA by the end of next week.

Wade & Taylor:

- Isolate serpent algorithm in separate benchmark.

4.12 Week 18

Progress

Taylor & Wade:

- Implemented file wrapper which properly pads, reads & writes blocks, and writes file padding.
- Able to generate Serpent extended key using affine recurrence (for encryption algorithm).
- Implemented serial version of Serpent encrypt.

Andrew & Garrett:

- Twofish working on CUDA with a single block.

Problems

Taylor & Wade:

- None.

Andrew & Garrett:

- Poster not yet completed for reasons unknown to Wade & Taylor; was supposed to be submitted today.
- Garret out of town next week.

Plans

Taylor & Wade:

- Finish serial version of Serpent decrypt tonight; post to GitHub when completed.
- Work on CUDA and pthreads implementation starting at end of workweek.

Andrew & Garrett:

- Get CUDA working on arbitrary number of blocks.

4.13 Week 19

Progress

Taylor & Wade:

- Improved benchmark interface.
- Began CUDA encryption implementation.
- Able to encrypt smaller portion of data through Serpent with CUDA.

Andrew & Garrett

- No progress.

Problems

Taylor & Wade:

- Having trouble allocating large amount of memory on the GPU.

Andrew & Garrett:

- Garrett out of town this week.

Plans

Taylor & Wade:

- Taylor will buy 460 GTX and reconfigure personal computer for benchmarking purposes.
- Finish CUDA implementation of Serpent algorithm.

Andrew & Garrett:

- None given in update.

4.14 Week 20

Progress

Taylor & Wade:

- Serpent now encrypting arbitrary-length file through CUDA (successfully tested on 2GB file).

Andrew & Garrett:

- No update given.

Problems

Taylor & Wade:

- Unable to meet during Finals' Week.

Andrew & Garrett:

- No update given.

Plans

Taylor & Wade:

- Implement Serpent decryption in CUDA after end of Finals.

Andrew & Garrett:

- No update given.

4.15 Week 21

Progress

(This update will include progress made over Finals Week and Spring Break)
Taylor & Wade:

- Implemented Serpent CUDA decrypt.
- Implemented Serpent encrypt and decrypt in parallel through OpenMP.
- Greatly improved file read/write times.
- Added benchmark times to stdout (up to "1 nanosecond" resolution).
- Serpent encrypt and decrypt now programmatically determines multiprocessor and thread count.

- Serpent encrypt and decrypt now uses constant memory for the key and several other variables, greatly increasing the algorithm speeds.
- Ported code over to 64-bit architecture, and back again to 32-bit architecture; should be fairly robust on both architectures now...
- Fixed bug when writing a file that caused file enlargement and corruption.
- Fixed bug that caused errors when truncating extremely large files.
- Began work on automatically-generated reports.
- Report now creates appropriate directory structure for the report based on the time of day.
- Report now automatically creates a LaTeX PDF file.
- Report automatically retrieves system information using `lshw`.
- Report can now automatically write a LaTeX table of data.
- Benchmarking machine successfully set up (it's a beast).

Andrew & Garrett:

- No progress.

Problems

Wade & Taylor:

- Faulty RAM caused much confusion.
- File writing bug was causing file corruption.
- 32-bit and 64-bit fprintf() incompatibility; should be fixed via preprocessor directives.
- Extremely large files (15 GB) were triggering unexpected truncating behaviour.
- Serpent memory buffer needs to be arbitrarily set based on the NVIDIA graphics card; not sure how to programmatically determine.
- `lshw` output in report is ugly and trails off the report, also making the output unreadable; may leave in a separate .txt file referenced by the report.

Andrew & Garrett:

- Garrett has a busy week ahead.

Plans

Wade & Taylor:

- Taylor will replace corrupted RAM.
- Begin merging the report with the benchmark program.
- Perform automated statistical analysis on the report.

Andrew & Garrett:

- "Have Two-Fish encryption ready and testable by the end of the week."

4.16 Week 22

Progress

Wade & Taylor:

- Report now fully integrated with the benchmark.
- Program interface revamped for report.
- Report collects and writes all necessary data.

Andrew & Garrett:

- "Serial Twofish is done".

Problems

Wade & Taylor:

- Taxes ate up Friday.

- Taylor was busy this week.
- Barcamp next week.

Andrew & Garrett:

- Not specified.

Plans

Wade & Taylor:

- Add standard deviation, sample mean, and t-statistic calculations to the report.

Andrew & Garrett:

- Work on CUDA.

4.17 Week 23

Progress

Andrew & Garrett:

- "Making progress implementing Twofish with the wrapper."

Wade & Taylor:

- Report now automatically calculates mean and standard deviation.
- Compilation sanitized; should not need to manually load shared object library.
- Serpent encrypt/decrypt should now automatically reallocate global memory buffer if initial attempt fails (no more manually updating and recompiling).
- Fixed uninitialized key bug.

Problems

Andrew & Garrett:

- "Since twofish makes use of how data blocks and the keys are setup, we had to do an additional data layer with what Wade & Taylor created so that we can index the key as well as the data block".

Wade & Taylor:

- Calculating t-statistic involves a Gamma function... not sure how to implement.
- Uninitialized key bug caused much confusion.

Plans

Schedule a meeting with client to go over progress made and develop a plan for the next couple of weeks.

4.18 Week 25

Progress

Wade & Taylor:

- Submitted poster for Expo.

Andrew & Garrett:

- Some work on TwoFish.

Problems

Wade & Taylor:

- Wade running fever all week; fell behind in classes and have two midterms this coming week.
- Taylor busy all week.

Andrew & Garrett:

- Trouble getting TwoFish to compile.
- Missed poster deadline.

Plans

Wade & Taylor:

- Figure out why encrypt is running faster than decrypt.
- Determine theoretical maximum execution time.

Andrew & Garrett:

- Get TwoFish completed by Expo.

Andrew:

- Work on getting CSV to graph automating script for the report.

4.19 Week 26

Progress

Wade & Taylor:

- Fixed difference between Serpent Encrypt/Decrypt algorithms on CUDA.

Andrew & Garrett:

- No progress.

Problems

Wade & Taylor:

- Unable to figure out what is causing the difference between Serpent Encrypt/Decrypt algorithms on the CPU.
- Attempt to increase algorithm speed via global memory coalescing was unsuccessful.

Andrew & Garrett:

- ???

Plans

Wade & Taylor:

- Send e-mail detailing results of Serpent encrypt/decrypt analysis.
- Prepare for Expo.

Andrew & Garrett:

- ???

5 Final Poster

See Figure 3 for the poster presented at the Engineering Expo.

Figure 3: The poster that was presented at the Engineering Expo; see the CD for better resolution. Please note that, after the expo, the CUDA version of Serpent decryption was fixed to run at the same speed as Serpent encryption.

6 Project Documentation

6.1 Program Structure

Though written in C, the program is written in a more object-oriented manner by having the majority of the functionality invoked by passing structures into their manager modules (each `x.c` and `x.h` file is considered a "module" in this case; the functions within these files "manage" the object). The main structure for this program is the `report_t` structure; this structure is meant to be an abstraction of the overall benchmarking process and contains data for each algorithm and the results from benchmarking the algorithm. Within the report are serveral `section_t`s, each representing a different algorithm to benchmark (such as Serpent and Twofish). Each `section_t` has several `subsection_t`s which represent the different methods for running the algorithm (Serial, Parallel, and CUDA). These `subsection_t`s then contain two `benchmark_data_t`s for encryption and decryption data.

The `main()` function interacts with the `report_t` by calling the appropriate functions in the report module; the function in the report manager then knows when to call the corresponding function in the section module, and so on. The overall result of this is a highly modular design that allows programmers to easily modify the code base without breaking the code.

6.1.1 Call Graph

See Figures 4-6 to view the call graph for the `main()` function.

6.2 Requirements

In order to run the program, the user must have a CUDA-capable device on their computer. Most NVIDIA GPUs developed within the last 6 years support CUDA. Note that there may also be a CUDA-capable device on the motherboard if the motherboard has an integrated GPU made by NVIDIA. To compile the program, the user will need to have the `nvcc` compiler and have the appropriate environment variables set. The system will also need to be running the appropriate CUDA-capable graphics driver. Instructions for configuring the environment variables and CUDA-capable graphics driver are outside the scope of this document; see the NVIDIA website for more details on configuring your system for CUDA.

To automatically generate the report, the user will need to have `latex`, `dvips`, and `ps2pdf` installed. This is free software and should be provided by your package manager. If these files are not installed, the program should still generate the `.csv` files from the data gathered.

System information is gathered by running the `lshw` program. If this program is not installed, then no system information will be generated. Like the LaTeX software, this program is free software and is built into some Linux distributions (such as Ubuntu). In order to generate the most system information through `lshw`, run the command as root.

6.3 Invocation

The program may be invoked by running `report <iteration count> <filename>` where `<iteration count>` is the number of times to run the benchmark and `<filename>` is the filename of the file to benchmark. A high iteration count is desireable for eliminating variablity introduced by the entropy of system scheduling, but will cause a linear increase in execution time. The file to be benchmarked should be a relatively large size in order to accurately take advantage of OpenMP and CUDA (because the initialization time may be longer than the serial runtime for parallel and CUDA processes). Also, the file should not contain sensitive or unique data; this program is for benchmarking purposes only.

Depending on how the library pathing is setup, it may be necessary to add `libccc.so.*` to the library path. This should be possible by appending `env LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:./` as a prefix to the program invocation command. Alternatively, one can copy `libccc.so.*` to the `/usr/lib` directory, but that requires root access and should not be necessary in most cases.

6.4 Doxygen

The source code has been well-commented in most cases and even contains a file for Doxygen named `doxyfile`. If one has Doxygen installed, it should be possible to automatically generate documentation for the code by running the `doxygen` program on the file. The Doxygen file is configured to automatically generate call-stack graphs using GraphViz. Since the doxygen file was generated rather late in the project, not all features are currently working

Figure 4: The first part of the call graph for main().

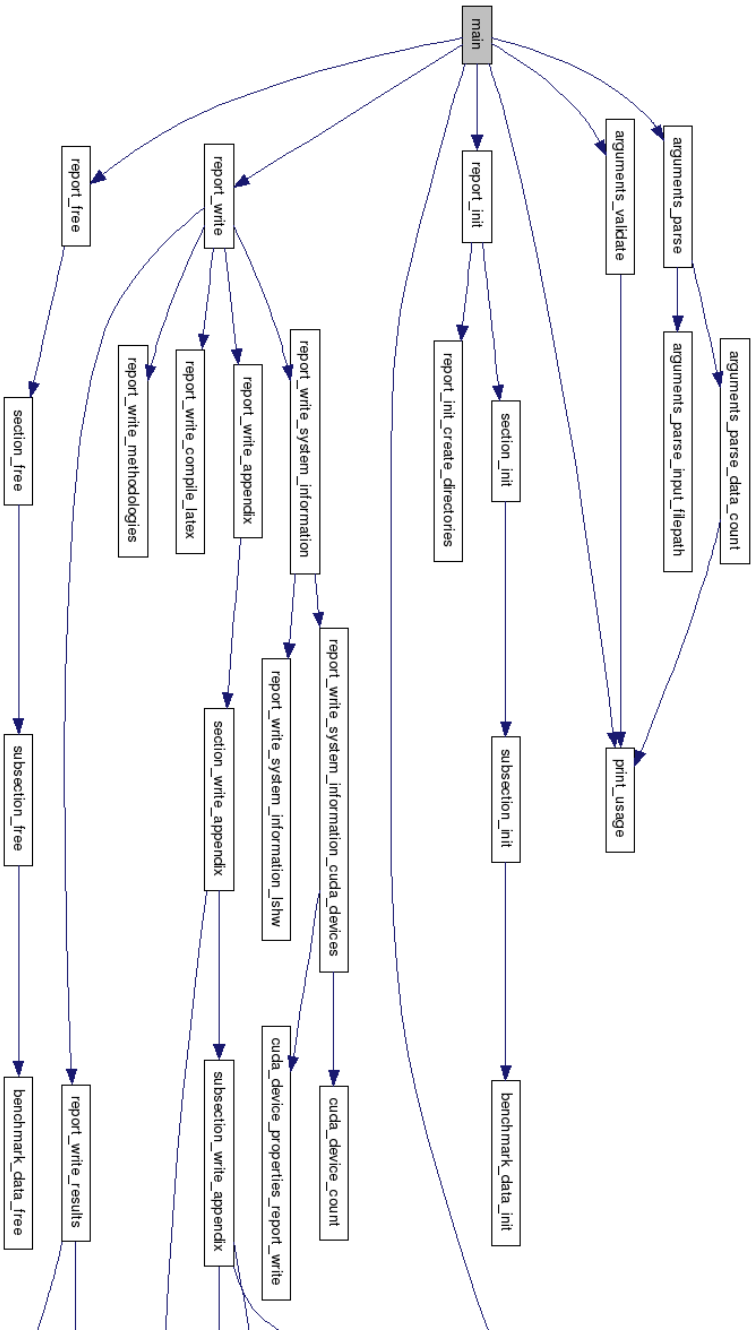


Figure 5: The second part of the call graph for main().

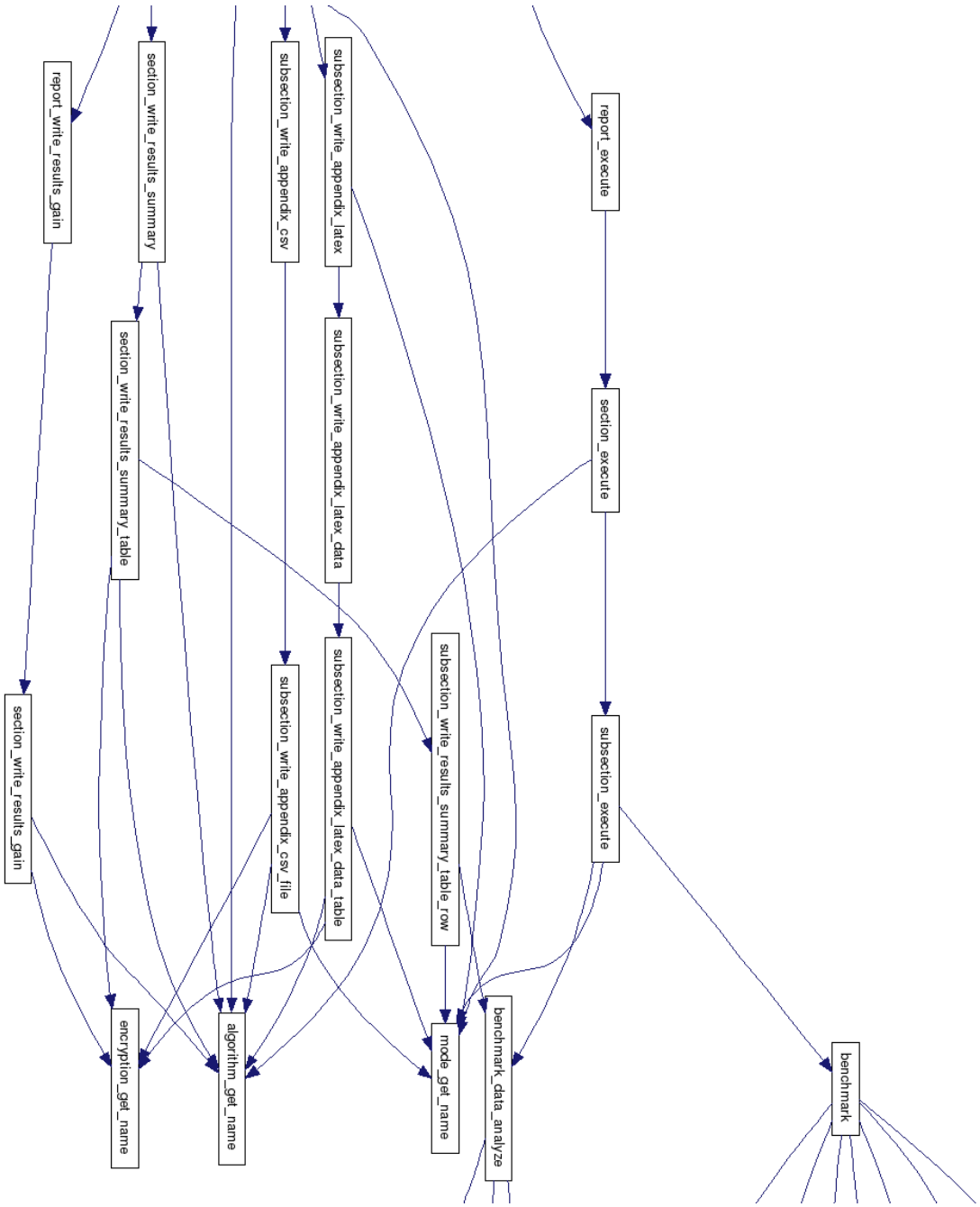
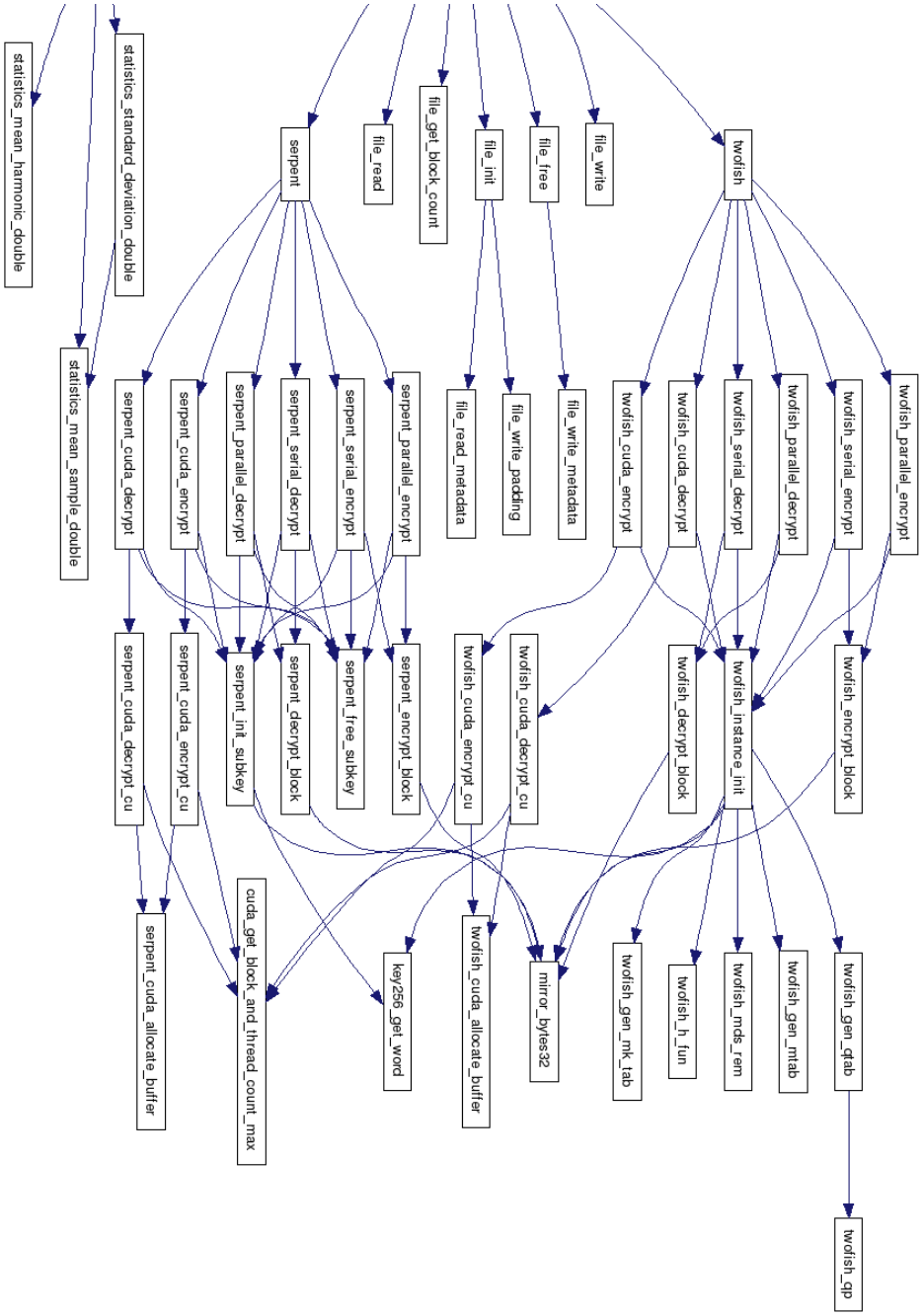


Figure 6: The third part of the call graph for main().



correctly; it may be necessary to manually configure `doxyfile` to get the output one wants. For details on how to run Doxygen and configure `doxyfile`, see the official website.

6.5 Retrospective

The object-oriented approach seemed to work extremely well for this project. The well-formed structure of the code allowed modifications to be easily made without breaking the whole program. Lots of error-checking often caught simple mistakes before they became a problem, saving hours of debugging time.

One thing that could have been implemented better would have been to consider the encryption and decryption algorithms as entirely separate algorithms. This would have made certain portions of the code considerably easier to deal with, since the two were always treated as separate algorithms when the analysis was written and the data was collected.

7 Learning New Technology

The most important part of learning CUDA was having a solid theoretical understanding of GPU architecture and understanding how to use a programming language in order to take advantage of the GPU architecture. This is because, while there are many implementations of the same process that can *compile* on the GPU there are very few that can *run well* on the GPU. The difference is often orders of magnitude, as in the case of this project. The book "Programming Massively Parallel Processors" by David B. Kirk and Wen-mei W. Hwu was the most helpful in this regard. The book clearly illustrated how the graphics card architecture works, what bottlenecks to watch out for, and how to use CUDA to avoid those bottlenecks. The knowledge from this book was used to change the Serpent algorithm from a 3x speedup to a 20x speedup on the testing machine.

Another important technique for learning new technolgy was the ubiquitous Internet search engine. The search engine helped mainly with specific questions regarding programming. Examples include: looking up syntax, interpreting error messages, and referencing API functions. This helped by filling in the gaps of any missing knowledge.

The more concrete book "CUDA by Example: An Introduction to General-Purpose GPU Programming" by Jason Sanders and Edward Kandrop was also useful. The book was full of helpful code samples and also contains some newer features of CUDA. This helped during the actual implementation of the program, when it was sometimes necessary to look up the exact syntax for certain CUDA features. The examples were also more complete than the short snippets of code found on the Internet, allowing one to understand the context in which to use certain features. A good complement to Kirk and Hwu's book.

8 Knowledge Gained

Much of this project involved learning about the GPU, its hardware architecture, and how to properly use the hardware through code. However, it was also necessary to learn skills and concepts such as more esoteric library features, team management, and hardware simulation software. Only by learning all of these was the project successful.

8.1 Programming the GPU

First and foremost we learned how to program the GPU. This involves learning the hardware architecture of the GPU and understanding how the source code maps to the hardware. The key difference between this and regular schoolwork is learning how to write imperative code that actually takes advantage of the imperativeness of the langugage, as compared to settling for the first piece of code that produces the correct output. This is because there are many different ways to write a program to solve a problem, but very few ways to solve that problem efficiently. A well-implemented program will know how to take advantage of the hardware for maximum performance, and that's what we had to learn to implement.

8.2 Large Codebase

The large codebase also taught us how to design and manage a large project. Most school projects can be easily hacked together and forgotten; this project involved planning and maintenance over the course of several months. Techniques for managing code, such a module managers and a consistent and easy method of error-handling had to be developed in order to keep the whole project cohesive and abstract enough for continued development.

8.3 Library Features

The complexity of the code also involved using some slightly more esoteric features of the standard library that we had neither used nor sometimes even heard of before. In order to get functions such as `clock_getres()` working it was necessary to learn and use something called "feature test macros" to grant access to the function. The same features were required for very large files, and 64-bit functions such as `stat64` and `lseek64` had to be tracked down in header files. Porting from 32-bit to 64-bit architectures also taught us about the `fprintf()` macros `PRIuPTR` and `PRIuMAX` defined in `inttypes.h`.

8.4 Valgrind

The performance analysis of Serpent Decrypt on the CPU required us to use cache simulation software in order to see if the cache was having a significant performance impact. The software Valgrind was used for cache simulation. This helped us become more aware of how our program interacted with the cache, and how cache misses can affect program performance. The duration of the simulation also taught us how extraordinarily slow software simulation of hardware is.

8.5 Social

Less technical, but certainly not less important, was learning how to work in groups. Organizing meetings, delegating work, and discussing requirements are all highly beneficial skills to practice. This project provided us with lots of experience in this regard. We also gained lots of experience in working with difficult team situations. This experience will serve us well when difficult situations arise in the future.

8.6 Encryption

Implementing the Serpent and Twofish encryption algorithms also taught us about encryption. This involved understanding how theoretical concepts such as confusion and diffusion relate to a secure cipher. Pragmatic implementations of these theories consist of constructs such as Feistel networks, Substitution-boxes, key-mixing, and input whitening. These constructs are well-known throughout cryptography, and various encryption algorithms make use of them in different manners.

8.7 Retrospective

Knowing what we know now, we would make several changes to how we approached the problem. First, we would have the Requirements Document delayed in order to gain a better theoretical understanding of the GPU; it is important to have this understanding before choosing algorithms to implement. Next, we would have considered talking to several different departments to see if they had any research code that could benefit from CUDA. This would give opportunity for interaction between the various academic disciplines. The downside to this plan is that it would leave only a very small amount of time to actually implement the algorithm and write up the report.

9 Appendix

9.1 CUDA Indexing

This code sample is meant to highlight the importance of properly indexing when writing code for the GPU. The code is for the function `__global__ void serpent_cuda_decrypt_blocks(block128_t* cuda_blocks)`. Note that some variables are globally scoped (these are the constant variables in CUDA). This first example shows a naive way to index:

```
int index = (blockIdx.x * blockDim.x * blocks_per_thread) + (threadIdx.x * \
    blocks_per_thread); // (beginning of multiprocessor segment) + (segment index).
int i;

// Encrypt the minimal number of blocks.
for ( i = 0; i < blocks_per_thread; i++ ) {
    serpent_cuda_decrypt_block(&(cuda_blocks[index + i]), cuda_subkey);
}
```

```
// Encrypt the extra blocks that fall outside of the minimal number of blocks.
index = ( gridDim.x * blockDim.x * blocks_per_thread ) + ((blockIdx.x * \
    blockDim.x) + threadIdx.x); // (end of array) + (absolute thread #).
if ( index < blocks_per_kernel ) {
    serpent_cuda_decrypt_block(&(cuda_blocks[index]), cuda_subkey);
}
```

Imagine an array of blocks of arbitrary length. This method would divide that array into continuous subarrays based on the number of threads and multiprocessors (multiprocessors are known as blocks within the CUDA API, but reffered to as multiprocessors in this case in order to distinguish them from the 128-bit blocks of plaintext/cyphertext). For example, if there were 1000 blocks, 10 multiprocessors, and 10 threads per multiprocessor, then the first thread on the first multiprocessor would process blocks 1-10 in that order, then thread two would process blocks 11-20, and so on, while the first thread on the second multiprocessor would process blocks 101-110, the second thread blocks 111-120, &c. The overall results of this is that, in the first round, blocks 1,11,21,...,991 are being processed.

While this is logically correct, and produces the correct results, it overlooks an important feature of the GPU's memory model. When blocks 1,11,...,991 are being processed, because the data for blocks 1-1000 is continuous but only so much memory may be accessed at once, each block must be serviced separately by global memory. The result is that time is wasted when a thread is waiting for its memory request to be serviced. The better method accesses adjacent segments of global memory as follows:

```
int index = (blockIdx.x * (blockDim.x * blocks_per_thread)) + threadIdx.x;
int i;

// Decrypt the minimal number of blocks.
for ( i = 0; i < blocks_per_thread; i++ ) {
    serpent_cuda_decrypt_block(&(cuda_blocks[index]), cuda_subkey);

    index += blockDim.x;
}

// Decrypt the extra blocks that fall outside the minimal number of blocks.
index = ( gridDim.x * blockDim.x * blocks_per_thread ) + ((blockIdx.x * \
    blockDim.x) + threadIdx.x); // (end of array) + (absolute thread #).
if ( index < blocks_per_kernel ) {
    serpent_cuda_decrypt_block(&(cuda_blocks[index]), cuda_subkey);
}
```

In this implementation, the first thread on the first muliprocessor processes blocks 1,11,...,91 in that order, the second thread blocks 2,12,...,92, and so on, while the first thread on the second multiprocessors processes blocks 101,111,...,191, the second thread blocks 102,112,...,192, &c. In this case, the first round processes blocks 1-10,101-110,...,901-910. Note that this isn't *more* blocks per round, but it's a *smarter* way of indexing threads.

In this case, when blocks 1-10 ask for access to global memory, the hardware detects this and merges, or *coalesces*, all their memory accesses into a single access. The result is significant; the latter implementation takes about 25% less time to execute. This shows how a little knowledge and a small code change can make a drastic difference in GPU programming.

9.2 Serpent Decrypt Analysis

An interesting portion of this project involved trying to find out why Serpent Decrypt was running slower than Serpent Encrypt on the CPU (Note that the Fianl Poster's GPU results contain an artifact from the naive indexing implementation mentioned earlier in this report). One idea was to run a cache analysis using Valgrind's `cachegrind` tool, which took a couple of days to run. The results showed no significant number of cache misses between the two, and also showed that Serpent Encrypt had significantly more instruction cache reads and regular writes, while decrypt had more reads. Yet since none of these were misses, it does not appear to explain the slowness of Serpent Decrypt on the CPU. Likewise, a look at the mathematical operations performed did not reveal a significant difference between the Serpent Encrypt and Decrypt. It is interesting to note that the implemtation of Serpent Decrypt on the GPU actually tends to run *slightly faster* than the implementation of Serpent Encrypt on the GPU.

The reasons for this are currently unknown to and beyond the scope of this team's knowledge. The tables from `cachegrind` are in Figure 7 and may also be found on the CD.

Figure 7: Data tables from running cachegrind on Serpent encrypt and decrypt.

Legend:

IR, *I1mr*, *I2mr*	Instruction cache reads, L1 read misses, and L2 read misses, respectively.
Dr, *D1mr*, *D2mr*	Data cache reads, L1 read misses, and L2 read misses, respectively.
Dw, *D1mw*, *D2mw*	Data cache writes, L1 write misses, and L2 write misses, respectively.

10 runs:

Serial	*IR*	*I1mr*	*I2mr*	*Dr*	*D1mr*	*D2mr*	*Dw*	*D1mw*	*D2mw*
Encrypt	5.04E+011	559	299	3.70E+011	4.69E+007	4.69E+007	3.84E+010	0	0
Decrypt	4.71E+011	524	276	3.93E+011	4.69E+007	4.69E+007	1.84E+010	0	0

10 runs:

Parallel	*IR*	*I1mr*	*I2mr*	*Dr*	*D1mr*	*D2mr*	*Dw*	*D1mw*	*D2mw*
Encrypt	5.04E+011	638	334	3.70E+011	9.38E+007	9.37E+007	3.84E+010	2261	1166
Decrypt	4.71E+011	605	312	3.93E+011	9.38E+007	9.37E+007	1.84E+010	2522	1356

1 run:

Both	*IR*	*I1mr*	*I2mr*	*Dr*	*D1mr*	*D2mr*	*Dw*	*D1mw*	*D2mw*
Encrypt	1.01E+011	136	70	7.40E+010	1.41E+007	1.41E+007	7.69E+009	146	108
Decrypt	9.42E+010	130	66	7.85E+010	1.41E+007	1.40E+007	3.68E+009	306	179

The following comparison tables use 'Encrypt/Decrypt' for the Ratio and 'Encrypt – Decrypt' for the Difference.

10 runs:

Serial	*IR*	*I1mr*	*I2mr*	*Dr*	*D1mr*	*D2mr*	*Dw*	*D1mw*	*D2mw*
Ratio	1.07	1.07	1.08	0.94	1	1	2.09	-	-
Difference	3.26E+010	35	23	-2.27E+010	0	0	2.01E+010	0	0

10 runs:

Parallel	*IR*	*I1mr*	*I2mr*	*Dr*	*D1mr*	*D2mr*	*Dw*	*D1mw*	*D2mw*
Ratio	1.07	1.05	1.07	0.94	1	1	2.09	0.9	0.86
Difference	3.26E+010	33	22	-2.27E+010	-45	-51631	2.01E+010	-261	-190

1 run:

Both	*IR*	*I1mr*	*I2mr*	*Dr*	*D1mr*	*D2mr*	*Dw*	*D1mw*	*D2mw*
Ratio	1.07	1.05	1.06	0.94	1	1	2.09	0.48	0.6
Difference	6.53E+009	6	4	-4.54E+009	-93	21928	4.01E+009	-160	-71