Testing the Feasibility of Running a Computationally Intensive Real-Time Traffic Simulation on a Multicore Programmable Graphics Processor

A Thesis
in STS 402

Presented to

The Faculty of the
School of Engineering and Applied Science
University of Virginia

In Partial Fulfillment
of the Requirements for the Degreee

Bachelor of Science in Computer Engineering

by

Kevin Stammetti

April 4, 2007

Table of Contents

Table of Figures

Glossary

CPU	Central Processing Unit
Device	Another term for GPU
CUDA	Compute Unified Driver Architecture. CUDA is a software interface provided by Nvidia for issuing and managing computations on the GPU as a data-parallel computing device without the need of mapping them to a graphics API [1:1].
GPGPU	Stands for *General-Purpose computation on GPUs*. With the increasing programmability of commodity graphics processing units (GPUs), these chips are capable of performing more than the specific graphics computations for which they were designed [2:1].
GPU	Graphics Processing Unit
Host	Another term for CPU.
MITSIM	Open source microscopic traffic simulator produced by the Intelligent Transportation Systems Program at the Massachusetts Institute of Technology.
Multicore	Computer architecture design that places more than one computational unit on a single integrated circuit.
Parallelism	A situation in which the computer is able to simultaneously execute two or more processes [3:861].
SDK	Software Development Kit. An SDK allows a programmer to learn about the enabled features of a given architecture.
Thread	A unit of computation that contains the minimum internal state and resources [3:869].

Abstract

In the 1960s, a semiconductor scientist named Gordon Moore theorized that the number of transistors would double each year on a single integrated circuit. Through much effort, the semiconductor industry has been able to closely follow “Moore’s Law”, but new information shows this type of progress is not sustainable in the coming years. This realization has implications in both chip fabrication and software development. Instead of making chips with more transistors per unit area, industry now produces newer multicore chips. These multicore chips, which have more than one traditional computational unit, have long been used for computer graphics, but now researchers are putting their improved throughput to use in computational simulation. New research documents efforts to speed up traditional simulations on multicore graphics processing units (GPUs), both to help simulation efforts and to learn about multicore chip potential. In addition to this new research style, given the acronym GPGPU (Graphics Processor as a General Processing Unit), there is a social desire to produce improved automotive traffic safety systems. It follows that research would follow in order to make a faster traffic simulation using state-of-the-art GPUs. In fact, this document details a research project aimed at testing the feasibility of running a traffic simulation using the GPGPU paradigm.

Chapter 1: Introduction

In the 1960s, Intel co-founder Gordon Moore theorized that the number of transistors on an integrated circuit would double each year. Though progress has slowed to doubling the number of transistors to every eighteen months to two years, his theory spawned a concerted effort to make chips faster. Despite such effort, current fabrication techniques can only sustain such progress for a few more years as fabricators get close to atomic limits.

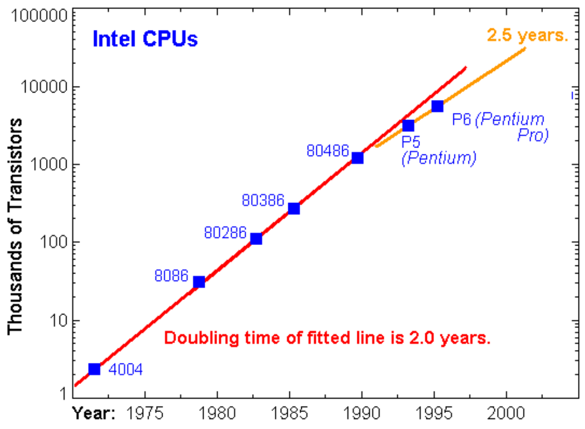


Figure 1: Moore's Law in relation to CPUs [4].

Knowing this, the computer architecture industry believes that multicore chips may be the key to increasing the speed of processors in the future. Industry has traditionally used these multicore chips, which combine multiple traditional central processing units (CPUs) on one circuit board, solely in computationally intensive graphics processing units (GPUs). Now, research is heading in the direction of utilizing these multicore chips, and specifically GPUs, to perform real-time analysis of non-graphical applications [5:41]. The computer architecture industry has dubbed this movement GPGPU, which stands for “Graphics Processor as a General Processing Unit” [2:1]. This thesis research project depicts the use of the GPGPU paradigm in order to adapt an automotive traffic simulation using a new Application Programming Interface (API) for a GPU. In doing this, the project shows how to make use of the potential of a multicore chip, as well as performs a feasibility study as to whether the GPU technology can support a time-stepped simulation.

Some researchers have already begun attempts in GPU simulation, but not many. Because there have not been many documented attempts, related literature has a distinct boundary in relation to the research presented here. Most sources either deal strictly with traffic simulation research or with GPU programming; there are little to no sources that provide a crossover. For example, a researcher named Stanimire Tomov, attempted to use a GPU to do probabilistic simulations. Another research group, led by Weixiong Chen, attempted to produce a parallelized traffic simulation without using GPU technology. The lack of crossover demonstrates this project's novelty in computer industry's and traffic simulation's future.

Traffic simulation is an important area of research because the Federal Highway Administration (FHWA) has made it a priority to reduce traffic accidents in the near future. Recent statistics attribute 43,000 fatalities to automobile accidents in 2003 – and the FHWA sponsors many projects to make cars safer [6:1]. Professor Brian Smith is perhaps doing the most relevant research related to traffic safety here at the University of Virginia. His group is developing a simulation of a conceived technology set to revolutionize safety on the roads, called the Vehicle Infrastructure Integration (VII). Smith's research focuses on a proposed network of computer chips that reside in cars. This hardware will be able to communicate with other cars on the road about road conditions, detours, and even relative speeds [15]. As of now, Smith's group is developing a simulation network to be built on top of a microscopic traffic simulation. However, his new safety network has performance issues because of the underlying traffic simulation's poor performance. With a method for speeding up a traffic simulation, this project benefits his research specifically. More generally, a faster traffic simulation allows researchers to do traffic pattern analysis quicker.

The other engine driving this research is industry's desire for more computationally intense multicore chip applications. Traditionally, the "uni-processor" CPU has been the main computational unit in a computer, with the arithmetic and logic unit (ALU) performing much of the mathematical computation. Figure 2 shows the typical resource allocation of ALUs in a CPU. The computer architecture industry is pursuing knowledge about multicore chip applications because it has a strong desire to make faster machines by increasing the number of computational cores per chip. Figure 3 demonstrates that GPUs contain many more computational cores than a general CPU (the same color scheme is used as in Figure 2, with green representing combinational cores). The current paradigm for future chips seems to be in multicore chip design, as new processors such as the Sony Cell and Intel's dual-core Duo have been released within the past year. Still, industry must do research in order to identify the strengths and weaknesses of current multicore chip design.

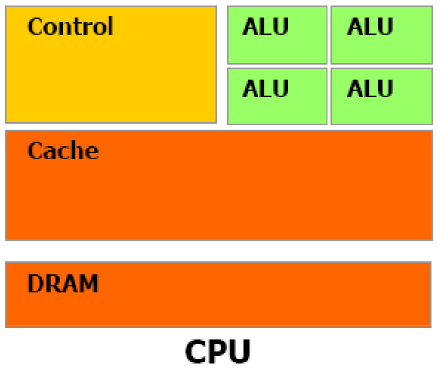


Figure 2: Typical Components of a CPU [1:2]

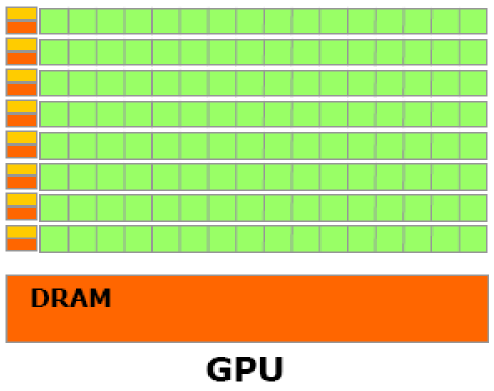


Figure 3: Typical Components of a GPU [1:2]

Justin Rattner, the Chief Technical Officer at Intel, gave a speech in the summer of 2006 about the concerns of the future of multicore development. He outlined the need to optimize the size of each core, the number of cores, and whether it is better to have a larger cache memory or a larger, faster main memory. The computer architecture industry is still looking for benchmark applications so that companies can "plan the technology of 2010 with the applications of 2006" [7]. Improved real-time simulations, such as the one provided in this project, give insight into how multicore chip technology can improve in the years to come.

Along with areas in hardware research, the software community should take interest in these results as well. There are many different types of problems that can utilize the potential of parallelization, and in turn use multicore chips to increase throughput. Using the newest APIs allows Kevin Skadron, this thesis's technical advisor, to explore the newest programming methods. Within his research group, students are working on multiple applications aiming to realize the potential of the GPGPU paradigm. One application aims to simulate a neural network by modeling synapses across a computer network, an application looking for a faster way to traverse linked data structures to improve memory access, and applications dealing with biomedical images. With parallelization on multicore chips, many simulations involving large amounts of data have been sped up. The group's goal for programming is to parallelize traditional algorithms and distribute the calculations over multiple cores, therefore reducing the time to run a simulation in hopes of enabling future research.

Within the framework of GPGPU simulation, this project spawned a traffic simulation that utilizes the basic components of an application called MITSIM. MITSIM is a microscopic traffic simulation developed at the Massachusetts Institute of Technology. The United States Department of Transportation defines a microscopic simulation as a program that simulates traffic on a car by car basis using developed car following and lane changing logic [8:8]. MITSIM is an open source application; the source code and documentation is freely available on the internet. MITSIM uses specific algorithms defining the ways cars move, follow, and change lanes. This research borrowed these algorithms in order to produce two traffic simulations: one "traditional" CPU implementation and one taking advantage of GPGPU technology. These MITSIM based applications form the basis of this project.

This project relies on specific GPGPU paradigms in order to produce valid results in the traffic simulation and computer architecture industries. Nvidia, a major graphics card producer, released an API named CUDA in the fall of 2006 to further its GPGPU efforts. CUDA allows a programmer to take advantage of multicore potential of an Nvidia graphics chip. Using CUDA, this project altered the MITSIM algorithm to take advantage of multiple threads so that this improved simulation could run on a machine with a multicore chip, specifically on an Nvidia GeForce8800 GPU. Computer architect Henry Moreton agrees that programmers can improve simulations like this with multicore chip technology. He has argued that "GPUs are well suited for large data parallel problems" [5:41]. A multicore chip has the possibility of greatly enhancing the performance of a large scale real-time simulation such as MITSIM.

At the beginning of research, this project hypothesized that the GPU technology would greatly speed up the MITSIM core algorithm. There were two main reasons for this hypothesis. First, the algorithm does not require much memory transfer between the CPU and the graphics card itself. The time it takes to load and unload memory from the graphics card was initially assumed to be negligible, and later discussion will back this claim. The other emphasis for a parallelizable algorithm is little dependency between tasks. The automotive traffic simulation follows this model because cars do not need to communicate with each other, just the surrounding environment. From these two initial properties, this project guessed incorrectly that the GPU technology would greatly enhance the traffic simulation.

Although this project's hypothesis turned out to be false, the project overall still demonstrates methods to be used in future applications to take advantage of parallelization in multicore chips. The rest of this report justifies this project and also reports the methods, results, and conclusions that working in this GPGPU environment generated. Chapter 2 is a literature review of related technical literature, showing past traffic simulation research as well as prior parallelization efforts. After that, Chapter 3 places the work in a social and ethical framework using real world examples. In Chapter 4, there are details concerning the experimental procedures that led up to the finished deliverable, which is a GPGPU implementation of a traffic simulation. Chapter 5 provides the qualitative and quantitative analysis of the experiment itself. There is graphical analysis that generates the conclusions given in Chapter 6. This final chapter not only provides the conclusions for this one experiment, but also suggests how future GPGPU research can help improve a traffic simulation. These results and conclusions are not only important to computer chip designing companies but to the academic community as well.

Chapter 2: Review of Related Literature

Researchers from industry and academia alike have recently begun attempts to parallelize computationally intensive simulations in order to run them on GPUs. Because there have not been many documented attempts, related literature has a distinct boundary in relation to the research presented here. Most sources either deal strictly with traffic simulation research or with GPU programming; there are little to no sources that provide important discoveries in both. This demonstrates this project's novelty in computer industry's future. This chapter focuses on related past research – by both industry researchers as well as academics – in the fields of traffic simulation and parallelizable GPU simulations.

2.1 Past Traffic Simulation Research

Initial research uncovered one fundamental source at the heart of this project: the whitepaper associated with the development of the MITSIM application. Written in 1996 by Qi Yang of the Massachusetts Institute of Technology and Haris N. Koutsopoulos of Carnegie Mellon University, the paper discusses the application's internal structure and algorithm. The detailed statistical calculations used to simulate driver decisions are not important to this project; the bits of analysis the authors provide show principal reasons as to why a traffic simulation must be sped up. The paper states that running time "heavily depends…on the number of vehicles that exist simultaneously in the networks" [9:13]. Also, speeds of cars are only updated every 1/10 second, a time unit visible to the naked eye. Finally, the paper shows the flow chart of decisions made per time step (See Appendix A). It seems that a traffic simulation is perfect for parallelization, as multiple cores can tackle the movement of many vehicles at one time. The details Yang and Koutsopoulos provide have direct impact into the implementation strategy's development to speed up this simulation.

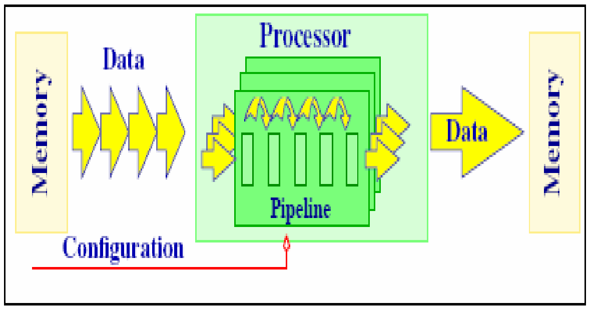
Researchers have also shown that general traffic simulation has intrinsic value. Over time, traffic simulation has played an important role into the development, analysis, and maintenance of real-life traffic systems [8:11]. Jim Clark, an engineer at the Federal Highway Administration, concluded his analysis of traffic patterns in Des Moines, Iowa and Fargo, North Dakota saved travelers "countless hours" and saved the cities "millions of dollars of right of way and construction costs" [10:8]. He used a government-sponsored microscopic traffic simulation called CORSIM, an application much like MITSIM. Clark's and his partner Gene Daigle's research, and other research like it, have helped sustain an interest in microscopic traffic simulation because such analysis provides fiscal savings.

Improvements in traffic simulation will likely follow such monetary success seen by Jim Clark and Gene Daigle. Weixiong Chen of the National Science Academy in China made initial attempts in parallelizing a simple traffic simulation algorithm. By creating multiple compute nodes (much like a model of a GPU) and communicating amongst them, he was able to develop a distributed network on which to run his traffic simulation. With his simple system, he concluded that four compute nodes could speed up the simulation up to 2.52 times over a single node system [11:5]. Multi-processor simulation has had success in the past.

2.2 Past GPU Programming Research

Beyond traffic simulation, this project demonstrates methods for programming simulations in GPU environments. An innovative idea has been published in the past few years that makes this process much easier. Henry Moreton, a design engineer at Nvidia (a major graphics processor design firm), wrote in 2005 that traditional simulations were the next step in GPU programming because of their good parallel capabilities [5]. He also describes in detail the workings of a general GPU, and specifically the Nvidia GeForce 6800. This chip is part of the CUDA hardware, so his research into the architecture and capabilities of traditional programming on the GPU is very relevant to this project. Another leader in GPGPU research is ATI Technologies, one of Nvidia's chief competitors. In 2006 they developed an Application Programming Interface (API) that makes traditional programming much easier. Traditional programming contrasts graphics programming, which requires the use of data sets involving more complicated data structures, such as vertices and line segments. With this API, which is a Data Parallel Virtual Machine (DPVM), it is easier to program on a GPU without using an inappropriate API, such as OpenGL (the traditional graphics programming API). Mark Segal, one of the authors of ATI's whitepaper regarding the DPVM, concludes that "the DPVM provides a straightforward programming model for data parallel applications" [12:6]. ATI and Nvidia provide developments and research that are keystones to the GPGPU movement, and in turn make this project a possibility. Without an API such as the DPVM (and similarly Nvidia's CUDA), executing a traditional simulation on a GPU would be extremely difficult; now, it remains a daunting task but is a reality.

People outside of the GPU manufacturing industry are making



headway in GPU programming as well. In September 2005, a member of Germany's Caesar Research Group, Robert Strodzka, published an article in the "Simulation Modeling Practice and Theory" technical journal. Strodzka and his team of researchers offer strategies for distributing memory on graphics cards. Figure 4, taken from Strodzka's article, shows that data can be arranged in arrays, which GPUs process at very fast speeds. They conclude that

Figure 4: 2D data streaming through a GPU [13:668]

APIs (such as CUDA, which was unreleased at the time) will eventually make GPU programming more like programming on a traditional CPU [13:670] Along the same lines, a team from Brookhaven National Laboratories in Upton, New York approached a similar problem. Without an API such as DPVM, the team was able to successfully simulate a Monte Carlo algorithm on a GPU. The Monte Carlo method sub-divides a circle into many small squares in order to approximate area. The team states that a GPU-programmed "32-bit floating point performance can be 2–6 times faster than the CPU for certain applications" [14:79]. Along with Moreton and Segal's work within the chip manufacturing, this research is very exciting. Strodzka's and Tomov's work, coupled with a desire to have a faster traffic simulation, lead directly to the research presented here.

Many areas of research directly relate to this project. From an initial desire to do traffic simulation, and then a desire to make this traffic simulation go faster, comes this project. Along with a request from Brian Smith's VII team, speeding up an algorithm like MITSIM's seems like a natural progression. This GPU programming approach has been the recent topic of research, both within the GPU manufacturing industry as well as from sources outside of industry. Together, the potential speed-up of a general simulation using GPU programming and the necessity of making MITSIM faster combine to form the inspiration for this project. The next chapter sets this project and its implications in a social and ethical context.

Chapter 3: Social and Ethical Context

The biggest contribution this research gives to society is in the area of traffic safety. Despite effort from researchers and government officials alike to improve traffic safety, the government released information showing that 43,000 deaths occurred on the road in 2003 [6:1]. Brian Smith's proposed VII system represents an attempt to increase traffic safety. Using wireless Dedicated Short Range Communications (DSRC), the proposed VII system allows chips integrated within a vehicle to "talk" to a dedicated road infrastructure as well as other vehicles equipped with DSRCs [15:4]. This chapter describes the societal impact of such traffic safety improvements as well as impact of further exploring the GPGPU paradigm.

This thesis project describes strategies to make a traffic simulation take advantage of parallel resources in the GPGPU paradigm. One has to question why industry needs to keep making faster computers; multiple factors help to answer these questions. First, there are economic reasons. If a competitive company does not produce faster hardware, consumers have no incentives to buy its new machines, and the company struggles to pay its employees. Next, there are cultural reasons. American society values everything moving faster. One does not have to look much further than the widespread growth of high-speed internet to realize that Americans value speed and efficiency. Finally, there are knowledge-seeking reasons. Researchers have the opportunity to speed up many computationally expensive simulations in order to do intensive research faster. The more research that can be done in a given time increases the chance that the given problem will be solved sooner. Multicore chip design provides one possible strategy to speed up simulation.

A fast-moving traffic simulation provides researchers and government officials alike more information about the next possible vehicle safety breakthrough: a wireless communication infrastructure. Over the past twenty years, the United States Department of Transportation has made a concerted effort to make driving safer for everyone. The USDoT has mandated technological breakthroughs, ranging from seat belts to standard driver's side airbags and eventually to an "advanced" – meaning computer based – technology like anti-lock brakes. Recently, lawmakers have been debating the proposal of a law requiring cars to have electronic stability control, which is a system that distributes braking in cases where a roll seems imminent, decreasing chances of that roll [16:1]. With technology proposed by Brian Smith's research, the possibilities are endless as to what could be automated in driving. In his initial investigation about VII architecture, he suggests examples such as "collision avoidance, intersection decision support systems, sensing using probe vehicles, route guidance systems, adaptive signal control systems, cooperative adaptive cruise control, as well as various other applications in commercial vehicle operations" [15:5]. Figure 5 shows just how dependent the VII system is on an underlying traffic simulation. Allowing Smith to possibly simulate his VII system in real time shows a tremendous technological leap into realizing a VII-type system and the safety enhancements that come along with it. However, critics debate whether these types of safety enhancements will decrease the amount of traffic accidents.

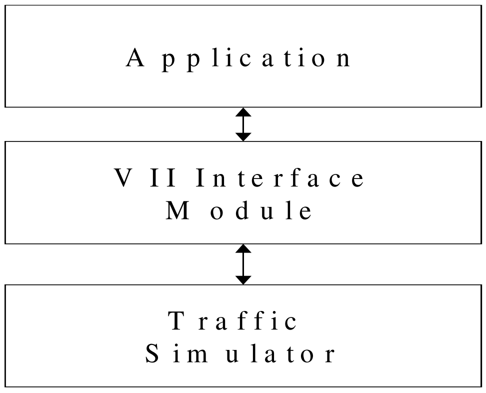


Figure 5: High-Level view of VII architecture [15:7]

A research faction interested in traffic safety research believes that increased technology in traffic safety devices actually have a negative impact on traffic safety statistics. Proponents of the "offset hypothesis" include University of Purdue Civil Engineering faculty member Fred Mannering. He defines the offset hypothesis as "consumers adapt[ing] to innovations that improve safety by becoming less vigilant about safety" [17:1]. Mannering and his team of researchers conclude that the purchasing of anti-lock brakes and driver's side airbags in the early 1990s led to greater "driver mobility". This mobility – faster driving and more frequent lane changes – mean that drivers get to their destinations quicker. Unfortunately, with the same factors, the monetary value of insurance claims per accident went up for drivers with the advanced safety features [17:12]. It seems that increased safety technology does not solely have positive benefits to society.

The implications of a faster traffic simulation realized through multicore chip technology have both positive and negative effects on society. First, introducing systematic methods that improve computer program efficiency seems to only have positive effects in the American economy. A technologically advanced infrastructure such as Brian Smith's VII has the ability to provide drivers with more safety features, possibly saving lives. This thesis project, testing the feasibility of using the GPGPU paradigm to speed up a traffic simulation algorithm, helps to guide the future for VII's implementation. VII and other future safety advancements will continue to attempt to combat the effects of the "offset hypothesis", and for these reasons, this research pursued its goals.

Chapter 4: Materials and Methods

The experiment this project details follows the simple scientific method taught throughout grade school. First, Kevin Skadron and Brian Smith produced a problem: to attempt to use the GPGPU paradigm to speed up a traffic simulation. After initial research, this project hypothesized that the GPU parallelization has the possibility of supporting a traffic simulation and speeding it up. The next logical step in the scientific method was to design an experiment. This experiment has two phases. The 'control' phase is the CPU implementation of the MITSIM core algorithm. The 'experimental' phase is the GPU implementation that takes advantage of a structured parallelization method named "Foster's Design Methodology" described in Ian Foster's book, "*Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*" [18]. These implementations produce timing results that differentiate the two, allowing for analysis. This chapter specifically discusses the technical elements of the two phases of the experiment, while the next chapter analyzes the graphical data.

4.1 CPU Implementation

This research produced a CPU implementation modeled strictly following the flow of MITSIM's execution. It does not have many of the features of MITSIM, but does contain the most parallelizable portion of the MITSIM simulation loop. This new implementation, nicknamed TrafficSim, uses the MITSIM logic to simulate cars moving down a two-way stretch of interstate, complete with exits every mile and the ability to switch lanes.

The implementation takes advantage of a data structure in order to simulate a vehicle. This structure, called Vehicle, stores important data about a virtual car. The structure stores the vehicle's speed, position, direction, lane, and destination exit. TrafficSim instantiates one of these data members for each car using a random seed in order to approximate real-life conditions. The structure also keeps a data member which is important in the comparison of the two implementations: if each car successfully exits or not. This data structure is necessary for simulating traffic.

TrafficSim uses simple assumptions in order to complete its execution. If a car is more than five exits away from its intended exit, it either stays in or changes to the left lane. If the car is within a quarter mile of its intended exit, it must get in the right lane to exit. When changing lanes, the car must check in front, next to, and behind itself in the target lane. If any other car occupies these spots, the program does not allow the car to change lanes. Finally, if a car misses its intended target because it cannot change lanes, TrafficSim assumes the car exits the interstate at the next exit. These simple approximations of real-world driving behavior help make TrafficSim credible as a traffic simulation.

TrafficSim's program flow comes directly from Yang and Koutsopoulis's initial MITSIM research paper. The CPU implementation models two phases of their simulation: the Update Phase and the Advance Phase. Figure 6 shows that

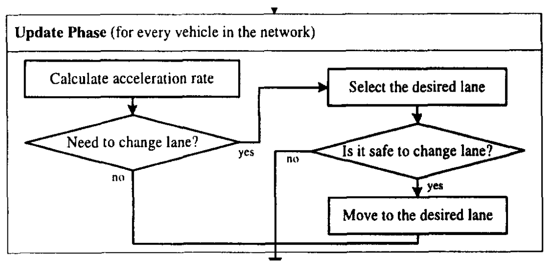

Update Phase (for every vehicle in the network)
Calculate acceleration rate
Need to change lane?
yes
no
Select the desired lane
Is it safe to change lane?
no
yes
Move to the desired lane

Figure 6: Flowchart for the Advance Phase of MITSIM [9:115]

the Update Phase checks to see if the car needs to accelerate or change lanes. If the vehicle does need to change lanes, the update phase checks to make sure the lane change is safe. If it is, the car changes lanes. From here, the simulation enters the Advance Phase. As Figure 7 points out, MITSIM's original Advance Phase is more involved than TrafficSim's

Advance Phase. TrafficSim lacks the traffic signals and sensors of

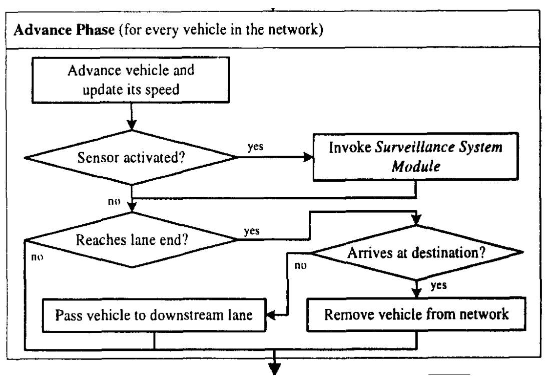

Advance Phase (for every vehicle in the network)
Advance vehicle and update its speed
Sensor activated?
yes
Invoke Surveillance System Module
no
Reaches lane end?
yes
no
Arrives at destination?
no
yes
Pass vehicle to downstream lane
Remove vehicle from network

Figure 7: Flowchart for the Advance Phase of MITSIM [9:115]

MITSIM's more in depth implementation. In TrafficSim's Advance Phase, each vehicle updates its speed and checks its position. If the car is at the desired location, TrafficSim removes the vehicle from its lane. If not, it advances the position based on the speed. After performing the Update Phase and the Advance Phase for each car in the network, the entire process starts over during the next time step. The full MITSIM flow chart can be found in Appendix A.

TrafficSim is a program that required very few special materials. Using standard C/C++ programming language properties and the Microsoft Visual Studio C++ compiler running on a Windows XP machine, this research project produced TrafficSim. The more "bleeding edge" materials did not enter the experiment entered the GPU implementation phase.

4.2 GPU Implementation

This project's production of a parallelized simulation followed a strict analysis of the MITSIM algorithm. The analysis using Foster's design methodology for parallel programming showed an implicit parallel structure within the algorithm. The final parallelized structure adheres to the methods set forth by Ian Foster and simulates each vehicle on its own thread using Nvidia's GeForce 8800 multicore architecture.

There are four main tenets in Ian Foster's design methodology for parallel programming. Algorithm analysis must produce methods of partitioning, communication, agglomeration, and mapping in relation to the program's execution. Partitioning simply means that the overall computation must be split into smaller pieces. Having independent threads is an easy way to partition a parallel program. In regards to communication, the design methodology suggests that both local and global communication must be strictly defined. Each thread can communicate on a small scale with a small number of "neighbor" threads (local communication) and can communicate on a large scale with all threads (global communication). The CUDA architecture, described in the following paragraph, makes these definitions quite simple. Next, agglomeration forces the program design to "group tasks into larger tasks to improve performance or simplify programming". Finally, the design must take "mapping" into account, meaning that the algorithm must efficiently take advantage of the different processors of a multicore architecture [18]. The experiment characterizes the traffic simulation using Foster's four tenets to produce a working implementation.

CUDA's parallel architecture works well with Foster's design methodology; together they helped spawn the GPU traffic simulation. Within CUDA, a thread is the smallest unit of parallelization [19:4]. Threads are grouped together in blocks, which are in turn grouped together in grids. Every level has its own memory with different access parameters. Threads have access to thread memory, block memory, and global grid memory. A thread can communicate with threads on its block through the shared block memory or through global memory, which all blocks have access to. Figure 8 shows the properties of the different levels of memory architecture. From this figure, it is easy to

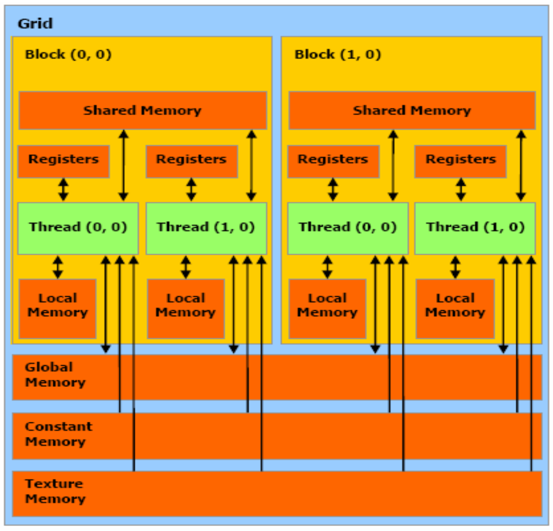


Figure 8: CUDA architecture memory structure [1:11]

see how Foster's design methodology applies to the traffic simulation. In the simulation, if each car receives its own thread, it has dedicated memory to store its own state. A car communicates with threads on its block through shared memory and with all cars through the global memory, where the highway lanes reside. These realizations complete the first two stages of the design methodology. Using these threads also easily identifies agglomeration: they simplify the programming model. Each car thread executes the same program thread, making the overall design easier to program. Finally, the CUDA architecture does the "mapping" for the programmer. The defined grids, blocks, and threads automatically try to utilize the multiple processors efficiently while reducing the amount of inter-thread communication. The TrafficSim GPU implementation utilizes these distinct features of the CUDA architecture.

The algorithm for the TrafficSim GPU implementation is not very different from the CPU implementation, but the source code differs because of the CUDA specific functionality. First, the program has to determine a block size. This block size is the amount of threads that run per block. From this number, the program defines the number of blocks, which is simply the number of cars divided by the block size. Because the cars are stored in a one-dimensional array, it only makes sense that the program defines the blocks in a one-dimensional grid. After defining grid and block dimensions, TrafficSim must allocate the memory to store the cars on the graphics card. Once the program allocates and copies the memory, the GPU invokes its kernel function, which is very similar to the main function of the CPU algorithm. This is the function that each thread runs to completion. Once each thread exits the kernel function, the program copies the car memory from the GPU memory, stops the timer, and calculates the same statistic as in the CPU implementation: the total time for all cars to finish their trip. The overall program flow is very similar; the programming style of the most important function is quite different.

To be fully efficient, the kernel function needs to run the same code for all threads on the same block. This makes the programming style non-intuitive for each thread. At the end of each time step, one would think that if a car finishes, the thread should exit. However, because of CUDA's architecture, all threads on the block should exit at the same time. The exit condition should be block-specific, not thread-specific. TrafficSim accomplishes this using the shared block memory. At the beginning of execution, the program defines a variable named "cars_left". As a car exits, the thread decrements cars_left and proceeds as a "dummy" car. A dummy car will execute all code within the thread but will not change the state of the highway lanes that reside in global memory. The program runs this way so that all threads on a block run the same set of instructions.

The traffic simulation requires a high degree of synchronization on the GPU to avoid write-after-write errors. Because it is a time-step simulation, TrafficSim needs for each thread to have the same global state at the start of each step. Fortunately, Nvidia provides a standard method for doing this: syncthreads(). When the simulation invokes this function, each thread waits until all threads get to the syncthreads() call. Without this synchronization, one thread could access a portion of the highway at the same time as another, leading to two threads writing to the same memory location. Every car also avoids these problems whenever it updates the position. A car checks in front and to the side when moving forward or changing lanes. Figure 9 shows this property: for the

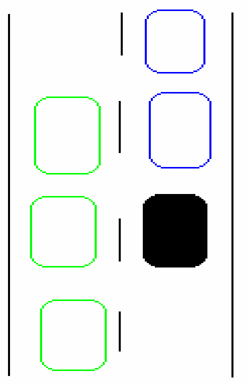


Figure 9: Free spaces necessary to move a car. As drawn by author.

black car, each green oval must be clear to switch lanes and each blue oval must be clear in order for it to move forward. Each thread on the block also has to synchronize access to the shared block memory. Without synchronization, one thread could possibly overwrite the correct "cars_left" value with an incorrect value. Syncthreads helps to ensure there are no write-after-write errors, which in turn ensures correct global memory access.

After the initial experiment ended, the project produced a third implementation to help explain exactly why the GPU implementation was slower than the CPU implementation. This new application did not use syncthreads, so is not viable as a true traffic simulation because it is not “time-stepped”. However, its performance is important in demonstrating exactly why the CPU implementation outperforms the GPU simulation. This new implementation will be described as the “unsynchronized” implementation.

This chapter details the high level implementation of the CPU and GPU implementations of TrafficSim. The source code is listed in Appendix C. Each simulation utilizes the same basic functionality on different hardware. Because of this similarity, one can credibly compare the timing results between the two to see that the CUDA architecture may not be the ideal hardware to run a traffic simulation. The next chapter discusses the results that lead to this conclusion.

Chapter 5: Results

To satisfy both motivations behind this work (traffic simulation and the GPGPU paradigm), this thesis project produced two sets of results. The actual experimental results are in the form of timing data. The CPU implementation of TrafficSim competed against two different sets of GPU implementations – a synchronized implementation and an unsynchronized implementation. The first section of this chapter displays the timing results comparing the CPU implementation and GPU implementations in line graphs. The section after details appropriate software development methods to take advantage of CUDA's parallel architecture.

5.1 Experimental Timing Results

This experiment produced data for different values of cars and lane lengths. The values correspond to increases in orders of magnitude. For example, the number of cars ranges from 100 to 10,000. All three implementations of TrafficSim executed each (number of cars, lane length) pair five times. The graphs below display the average of these five runs for each implementation, comparing the CPU implementation to each of the GPU implementations.

Figure 10 displays the comparison between the synchronized GPU implementation and the CPU implementation. The only data that the reader can see is from the GPU implementation simply because the CPU performs so much better than the GPU. The amount of time for GPU execution dwarfs the amount of time for CPU execution; the CPU timing does not even fit the scale. In the case of 10,000 cars on a lane length of 10,000, the GPU implementation is close to 450 times slower. Synchronization hurts the GPU performance dramatically.

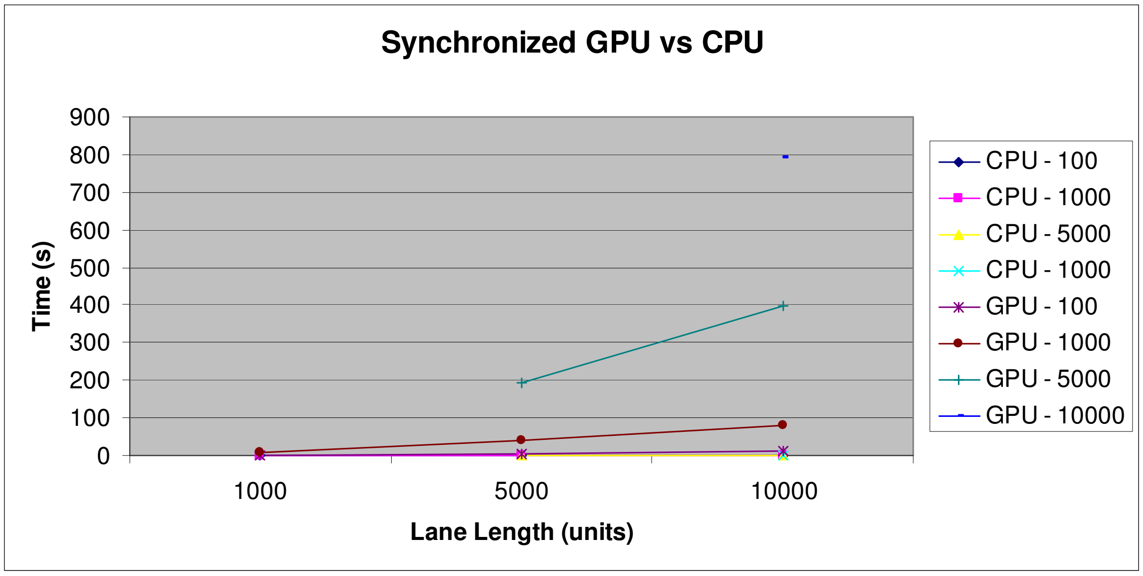


Figure 10: Time comparison of Synchronized GPU Implementation and CPU Implementation. Created by author.

Figures 11 and 12 show the comparison between the unsynchronized GPU implementation and the CPU implementation. Figure 11 depicts the timing differences while Figure 12 displays the speedup factor for each (number of cars, lane length) pair. In Figure 11, the slowest performances come from the CPU implementation. Figure 12 shows the speedup ranges from 1 to 25, depending on the length of the road and the number of cars. The parallel performance manifests itself in this data; syncthread calls do not hinder the unsynchronized GPU implementation. This project initially assumed these results would characterize the performance of the synchronized GPU implementation.

The experimental data can be found in Appendix B.

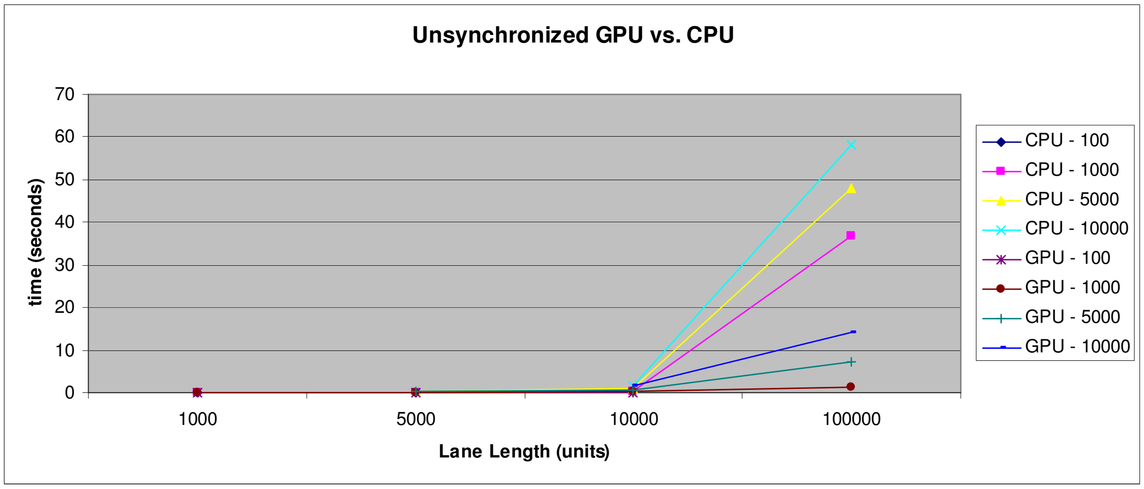


Figure 11: Time comparison of Unsynchronized GPU Implementation and CPU Implementation. Created by author.

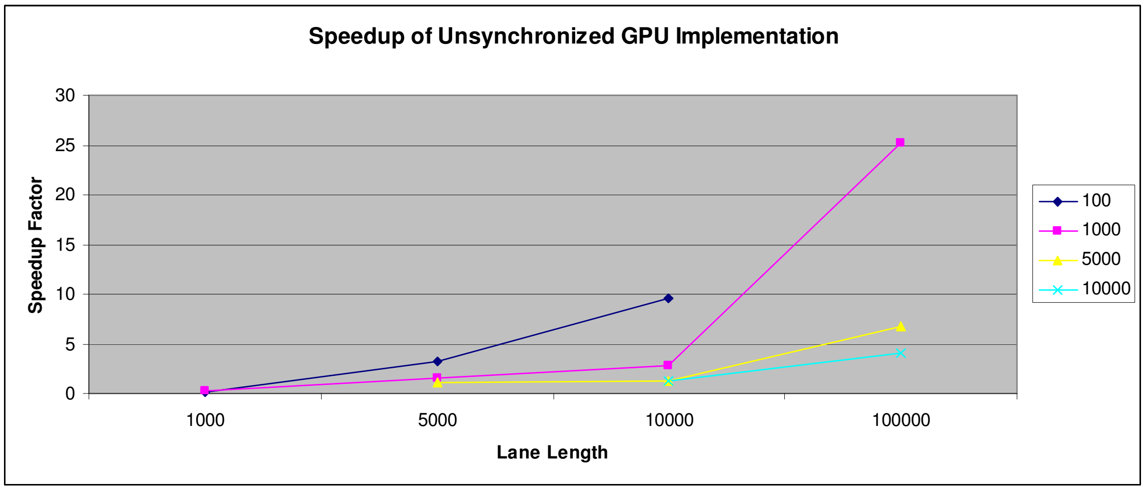


Figure 12: The speedup factors for the Unsynchronized GPU implementation vs. CPU implementation. Created by author.

5.2 CUDA Software Development Methods

Programming efficiently on the CUDA architecture is a very subtle science. To take advantage of the multicore technology, a program has to be structured in a very certain way so that it keeps all threads, blocks, and grids in use. To be truly efficient, the CUDA architecture requires many concurrent threads and a high ratio of mathematical calculations to memory accesses [1: 6].

One of the most important properties for a CUDA program is independence. Each thread should have minimal contact with other threads. This means minimal shared memory accesses and little to no shared data between threads [1: 6]. This complete independence does not require synchronization of any kind and truly takes advantage of the multicore potential of a GPU. Not all algorithms have this property, and it seems as if a true traffic simulation requires too much synchronized memory access to take advantage of the CUDA architecture parallelism.

CUDA programs need to keep thread flow as similar as possible. This keeps blocks very efficient; all threads on the block need to execute the same instructions. Divergent branches lead to great inefficiencies, and the traffic simulation has many branches that can take the execution flow in many different directions. Also, a block needs to have all of its threads running together to reduce inefficiency. This is why each thread in the traffic simulation has an exit condition based on a shared block variable; all threads finish at the same time. Unfortunately, this exit condition requires a great deal of synchronization, which hurts performance.

TrafficSim has very few properties that make it viable for good performance on the CUDA architecture. While the time to transfer data between the host memory and the device memory is very small, the actual execution time is very large compared to the CPU implementation. This is mainly because of the large amount of synchronization required among the many threads in the simulation. Divergent branching also contributes to poor performance. A parallelizable algorithm needs thread independence in both time and data access.

Chapter 6: Conclusions

It has been this project's goal to determine the feasibility of running an automotive traffic simulation on an Nvidia CUDA architecture graphics card. While technically feasible, running a traffic simulation using the GPGPU paradigm might not be the ideal in terms of computer performance. Through extensive research and experimentation, this project finds that, at the time of writing, a traffic simulation algorithm does not suit the CUDA architecture well.

This thesis project's main experiment demonstrates the incompatibility between the MITSIM core algorithm and the CUDA architecture. First, the experiment's methods called for a CPU reproduction of MITSIM's main flow. From here, this project produced a duplicate of this implementation to run on an Nvidia GeForce 8800 graphics card. The CPU implementation runs faster than the GPU implementation, much to this project's surprise. Once the experiment produced initial results, a new goal surfaced. This project decided to explain exactly why the GPU performed so poorly when running the traffic simulation. After creating a "non-synchronized" (non-time stepped, negating any credibility as an actual traffic simulation) GPU implementation, this project determined that the level of synchronization that a traffic simulation requires is not ideal for GPU implementation. The traffic simulation does not take advantage of the GPU's parallel capabilities well enough to warrant full use of a GPGPU traffic simulation.

These findings are not without limitations, however. The CPU implementation has very few bases in real world traffic behavior. In fact, the CPU implementation only follows the flow chart described in the initial MITSIM research document. While this may limit some of this research's credibility in terms of a traffic simulation, the lack of real world basis does not limit the feasibility study. TrafficSim's algorithm exactly mirrors MITSIM's core algorithm. As a feasibility study, this thesis project remains credible because of this similarity. Also, the experimental results lack a full test spectrum. This is mainly because the small results sample held a large amount of information. Any increases in either the number of cars or lane length would only have led to longer execution times in the GPU. The small test sample characterizes the GPU's poor performance without needing an exhaustive data set.

Although the project's initial hypotheses proved incorrect, this research still holds importance in both traffic safety and multicore chip design. This initial feasibility study shows that traffic simulation designers should not necessarily hold out for a GPGPU speedup; traffic researchers can look to other avenues for increased performance. This project also shows multicore chip designers some of the limitations to algorithms that perform well on their newest hardware. To be taken seriously in realistic simulations, industry needs to address limitations regarding synchronization and the execution flow of different threads. This project presents relevant information and can help direct future research.

This project's ultimate conclusion is that this type of research should not stop. Researchers can advance this project's basis for a GPGPU traffic simulation. TrafficSim's GPU implementation can certainly be fine-tuned in order to remove some of the efficiency-killing conditional statements. The simulation can certainly retain some of its realism with a more strict control flow rather than the many if-then-else situations that present themselves in the current version. Furthermore, future researchers can extend this type of work to a more professional traffic simulation, specifically MITSIM. Any results that a future GPGPU-MITSIM research effort could produce would have more credibility than this project's results. Such results could more easily influence academia and industry down a new research path. In relation to hardware research, industry researchers can rework their hardware to make it more compatible with time-step simulations. This project's technical advisor, Kevin Skadron, will be heading to Nvidia on sabbatical in order to present this and other projects' findings. He has stated that he wishes Nvidia to support divergent branching more efficiently. This type of hardware change would most certainly improve the performance of TrafficSim and other simulations that do not have a predefined execution flow.

Through research and experimentation, this project has shown that it is certainly feasible to run a time-stepped simulation on a GPU. The Nvidia CUDA hardware architecture makes it possible to run close to any algorithm on a GPU. However, the multicore GPU does not necessarily improve the performance of all simulations. In fact, the automotive traffic simulation that this project produced had a major performance decrease on the supposedly faster GPU. In order to determine whether or not to create a GPGPU algorithm, it is important to determine the level of parallelism inherent in the original algorithm. A traffic simulation does not meet the requirements for running an algorithm efficiently on a GPU. While society needs improvements in traffic simulation, following the GPGPU paradigm does not seem to be the best course of action.

Bibliography

[1] "Nvidia Cuda: Compute Unified Device Architecture Programming Guide". Internet:http://developer.download.nvidia.com/compute/cuda/0_8NVIDIA _CUDA_Programming_Guide_0.8.pdf, Feb. 12, 2007 [Mar. 26, 2007].

[2] "Introduction." Internet: www.gpgpu.org, Mar. 13, 2007 [Mar. 26, 2007].

[3] Gary Nutt. *Operating Systems, Third Edition*. Boston, MA: Pearson Education, Inc, 2004, pp. 844-72.

[4] "Moore's Law for Intel CPUs". Internet: www.physics.udel.edu/~watson/ scen103/intel.html, Apr. 20, 2000 [Mar. 27, 2007].

[5] Henry Moreton. "The GeForce 6800". *IEEE Micro*, vol. 25, pp. 41-5, Mar.-Apr. 2005.

[6] "Road Fact Safety Sheet". Internet: http://safety.fhwa.dot.gov/ facts/road_factsheet.htm, Mar. 20, 2007 [26 September 2006].

[7] Justin Rattner. "Cool Codes for Hot Chips: A Quantitative Basis for Multicore Design".Intel, 2006. http://www.hotchips.org/hc18/docs/ keynote1_hc18.pdf, Aug. 21, 2006 [Sep. 11, 2006].

[8] "Traffic Analysis Toolbox, Volume 1." Internet: http://ops.fhwa.dot.gov/ trafficanalysistools/tat_vol1/Vol1_Primer.pdf, Oct. 19, 2006 [Sep. 8, 2006].

[9] Qi Yang and Haris N. Koutsopoulos. "A Microscopic Traffic Simulator for Evaluation of Dynamic Traffic Management Systems." *Transportation Research, Part C, Emerging Technologies*, vol. 3, pp. 113-129, 1996.

[10] Jim Clark and Gene Daigle. "The Importance of Simulation Techniques in ITS Research and Analysis". *Winter Simulation Conference, Proceedings,* 1997, pp. 1236-43.

[11] Weixiong Chen. "A parallel model for traffic simulation." *Intelligent Transportation Systems, Proceedings,* 2003, pp. 1595-9.

[12] Mark Segal and Mark Peercy. "A Performance-Oriented Data Parallel Virtual Machine for GPUs". *ACM Siggraph, Proceedings*, 2006, pp. 1-6.

[13] Robert Strodzka et al. "Scientific computation for simulations on programmable graphics hardware". *Simulation Modelling Practice and Theory,* vol. 13, pp. 667-80, Nov. 2005.

[14] Stanimire Tomov. “Benchmarking and implementation of probability-based simulations on programmable graphics cards”. *Computers & Graphics*, vol. 29, pp. 71-80, Feb. 2005.

[15] Brian Smith et al. “Development of a Simulation Architecture and Implementation to EvaluateVehicle Infrastructure Integration”. *Unpublished Report.* September 2006.

[16] Tom Incantalupo. “New Car Safety Rule Expected.” Internet: http://www.newsday.com/business/nyzslug4890298sep14,0,4780562.story ?coll=ny-business-print, Sep. 14, 2006 [Sep. 27, 2006].

[17] Clifford Winston, Vikram Maheshri, and Fred Mannering. “An exploration of the offset hypothesis using disaggregate data: The case of airbags and antilock brakes”. *Journal of Risk and Uncertainty*, vol. 32, pp. 83-99, Mar. 2006.

[18] Anthony A. Aaby. “Design Methodology for Parallel Algorithms.” Internet: http://cs.wwc.edu/~aabyan/460/Design.html, Jun. 3, 2004 [Mar. 30, 2007].

[19] David Kirk and Wen Mei Hwu. “Lecture 14-15: CUDA Performance.” Internet: http://courses.ece.uiuc.edu/ece498/al/lectures/lecture14-15-CUDA-performance.ppt, 2007, [Mar. 27, 2007].