CUDA ACCELERATED LARGE SCALE VEHICULAR AREA NETWORK SIMULATOR

A Thesis by

Chok Meng Yip

Bachelor of Engineering, Wichita State University, 2011

Submitted to the Department of Electrical Engineering and Computer Science
and the faculty of the Graduate School of
Wichita State University
in partial fulfillment of
the requirements for the degree of
Master of Science

August 2014

CUDA ACCELERATED LARGE SCALE VEHICULAR AREA NETWORK SIMULATOR

The following faculty members have examined the final copy of this thesis for form and content, and recommend that it be accepted in partial fulfillment of the requirement for the degree of Master of Science, with a major in Computer Networking.

_____

Abu Asaduzzaman, Committee Chair

_____

Ramazan Asmatulu, Committee Member

_____

Yi Song, Committee Member

DEDICATION

To the Almighty, my loving wife, parents, in-laws, brother, and cousins for their ultimate encouragement throughout my education and for incomparable advice throughout my life

ACKNOWLEDGMENTS

I am very thankful to my thesis advisor Dr. Abu Asaduzzaman for his continuous encouragement and support throughout my research work. His timely supervision of my work and guidance allowed this research work to be completed on time. He always had time and patience to guide me in accomplishing this work in spite of his busy schedule and offered me assistance from time to time. It has been an honor to work for him as a graduate research assistant.

I express my gratitude towards Dr. Ramazan Asmatulu and Dr. Yi Song for their valuable encouragement and I would also like to thank them for taking time from their busy schedule and to be the part of my committee member.

I take pleasure in recognizing, the efforts of all those who encouraged and assisted me both directly and indirectly with my experimental research. I specially want to thank lab colleagues Mark Allen and Kishore Chidella for precious input and continuous motivation and support. Finally, I acknowledge the WSU CAPPLab research group and facilities for providing me with all the necessary resources to prepare my research work.

ABSTRACT

Both size and computational activities of Vehicular Area Network (VANET) are growing. Simulation of VANETs not only requires the simulation of network standards, but also the mobility of nodes. Such dynamic system involves computation of node distance, routing protocols, application layers, data send, data receive, etc. The simulation model of VANET requires both hardware and software supports to deal with massive computational problems. Currently available network simulators, like network simulator 3 (NS-3), are not adequate for simulating large-scale VANET systems. In this work, we propose a dual-stream Compute Unified Device Architecture (CUDA)-assisted VANET simulation model for multicore Central Processing Unit (CPU) / manycore Graphics Processing Unit (GPU) platform to increase computational throughput. The proposed CUDA assisted VANET simulator uses NS-3 as the core engine and improves throughput by exploiting massively parallel processing on the GPU. Experimental results show that the computation throughput can be increased up to 75x by splitting workload between CPU and GPU

TABLE OF CONTENTS

Chapter Page

1. INTRODUCTION.....1

1.1 Wireless Networking Technologies.....
1.1.1 Physical Layer.....
1.1.2 Medium Access Control.....
1.1.3 IEEE 802.11p.....
1.2 Mobility Modelling…….....
1.2.1 Vehicular Area Network.....
1.2.2 Driving Behavior.....
1.3 Applications.....

2. LITERATURE SURVEY.....

2.1 Problem Statement
2.2 Thesis Contributions.....

3. Wireless Networking Technologies.....

3.1 Network Simulator
3.2 NS-3 Simulator
3.2.1 NS-3 Architecture
3.3 NS-3 VANET Simulation
3.4 Large Scale NS-3 Simulation

3. PROPOSED CPU-MEMORY REGROUPING TECHNIQUE……..……

3.1 Traditional CPU to GPU Memory Mapping……
3.2 Proposed CPU to GPU Memory Mapping……

4. PROPOSED NS-3 VANET SOLUTION……..

4.1 CUDA…….
4.2 Current NS-3 VANET Implementation…….
4.3 Proposed NS-3 VANET Implementation……

5. RESULTS AND DISCUSSION……..

5.1 Assumption……
5.2 Experimental Setup……..
5.3 Validation of CUDA Implementation……..

TABLE OF CONTENTS (continued)

Chapter Page

6. CONCLUSION AND FUTURE EXTENSIONS…….…….

6.1 Conclusion……

6.2 Future Extensions……..

REFERENCES…….

LIST OF TABLES

Table Page

1. FCC Allocation of VANET Networks.....

2. Hardware Setup.....

3. Node Position…….

LIST OF FIGURES

Figures Page

1. Hidden Node Problem……..

2. RTS-CTS Illustration……..

3. Simple NS-3 simulation code…….

4. Simulation Results……..

5. Detailed Output……..

6. NS-3 Architecture…….

7. Example Callback Code……..

8. Source Code Excerpt……

9. Visualized Vehicular Trace File……

10. CUDA Logical Organization……..

11. Fermi GPU Floor Plan……..

12. Fermi Streaming Multiprocessor……..

13. CUDA Code……..

14. Data Structure of VANET Simulator……

15. VANET NS-3 Subroutine Calls……

16. Distance Computation……

17. Distance Matrix…….

18. Position and Movement Matrix……

19. Proposed NS-3 workflow……

20. CUDA code for Distance Matrix Computation…….

LIST OF FIGURES (continued)

Figures Page

21. VANET Highway Illustrated…….

22. Matrices representing results……..

23. Execution Time vs Problem Size……..

24. Calculated Speed Up……

25. Comparison against Iterations……

26. Comparison Against Number of Nodes……

LIST OF ABBREVIATIONS

VANET	Vehicular Area Network
MAC	Medium Access Control
OSI	Open Systems Interconnection
PSK	Phase Shift Keying
QAM	Quadrature Amplitude Modulation
DSSS	Direct Sequence Spread Spectrum
OFDM	Orthogonal Frequency Division Multiplexing
RTS	Ready To Send
CTS	Continue To Send
WAVE	Wireless Access for Vehicular Environment
DSRC	Dynamic Short Range Communication
SCH	Service Channel
CCH	Control Channel
BER	Bit Error Rate
V2V	Vehicular to Vehicular Network
V2I	Vehicular to Infrastructure
V2X	Vehicular to Other Units
SUMO	Simulation of Urban Mobility
IDM	Intelligent Driving Model
MOBIL	Minimal Overall Braking decelerations Induced by Lane Changes
CUDA	Compute Unified Device Architecture

IDE	Integrated Development Environment
GUI	Graphical User Interface
GPU	Graphics Processing Unit
GPGPU	General Purpose GPU
SM	Streaming Multiprocessors

CHAPTER 1

INTRODUCTION

In this chapter, we discuss the simulated behaviors of the NS-3 network simulator, specifically on wireless networking simulation and vehicular movement simulation. These are fundamental models necessary to simulate realistic behavior.

1.1 Wireless Networking Technology

In the present era, wireless networking technologies have become a common networking interface for modern consumer devices. The contemporary usage of computer networks presents many challenges for network engineers to provide scalable and reliable networks. The evolution of wireless networks in the last decade has changed the scale of devices. Network designers had to adapt network topologies and modify existing protocols in order to adapt to wireless networks [1].

Consumer wireless networks on modern devices are governed by IEEE 802.11 standard, also known as WiFi. This standard is to ensure devices interoperate across all manufacturers and implementations. As consumer devices become more network functional, consumers expect devices to be more ubiquitous and reliable without centralized controls. Wireless Ad-Hoc network is a standard that governs network connections without central access point. The network layer which governs route formations is responsible for the robustness of Mobile Ad-hoc Networks. Vehicular Area Network is an extension of Mobile Ad-hoc Networks, Vehicular area networks are formed without central access point; therefore they are similar to Mobile Ad-hoc Networks. The networks in vehicular area networks changes rapidly, and traditional mobile

ad-hoc networks standards does not adapt to these conditions. The rest of subsections discuss about each layer and solution that will be implemented in standards

1.1.1 IEEE 802.11 Wireless Communication

The IEEE 802.11 standard defines Medium Access Control (MAC) and physical layer (PHY) of the open systems interconnection (OSI) model to ensure compatibility across all manufacturers. The MAC layer controls fair sharing of network bandwidth. The physical layer defines the signal coding and frequency of wireless medium.

Physical Layer:

The physical layer of wireless networks in the OSI model is responsible for signal generation. Analog signals received by physical layer are known as symbol. Symbols are a group of waveforms, which can be easily recognized by the demodulator. There are many modulation methods to encode a signal into symbols. The common modulation applied in IEEE802.11 are Phase-Shift Keying (PSK) and Quadrature Amplitude Modulation (QAM). Both methods modulates binary digital signal into analog form and encodes more bits per symbol at the expense of transmission error rate.

Direct Sequence Spread Spectrum (DSSS) is further applied to modulated signal to further enhance success rate. The signal gained by applying such process is the process gain. DSSS split symbols (PSK or QAM signals) into sequence of channels. The sequences of channels are known by transmitter and receivers. Doing so makes signal resilient to channel jamming or noise.

The bandwidth is further increased by using multiple channels to transmit data. Orthogonal Frequency Division Multiplexing (OFDM) combines neighboring channel to further increase signal bitrate.

Propagation of Radio Waves

The radio wave signals have been heavily researched. Free-space propagation formula calculates received power over distance (eq1). Pr is the power received, Pt is the power transmitted from station, Gr is the receiver's antenna gain, Gt is transmitting antenna amplification, and λ is the wave length. Power received is the ratio of distance multiplied by distance.

$$\frac{P_r}{P_t} = \left[\frac{\sqrt{G_r G_t}\lambda}{4\pi d}\right]^2 \quad (1.1)$$

Distance plays a big role in wireless modeling, and it affects quality of signal being transmitted. Signal loss due to the distance is defined as the path loss. Wireless network design involves transmission amplitude and receiver gain. A high amplitude may cause too much noise generated at long distance, and transmit amplitude that is too low would not be enough to overcome path loss. Ideal amplitude would be sufficient to transfer data without interference to other nodes.

$$P_b = \mathrm{Q}\left(\sqrt{\frac{2E_b}{N_0}}\right) \quad (1.2)$$

The NS-3 simulator does not encode bits to symbols; however the bitrate and error rate can be computed as a function of distance and noise. The error rate calculation can be computed using equation 1.2. Eb is the energy required per bit, and N0 is the Gaussian Noise Level. Using Equation 1.1 and 1.2, it is sufficient to calculate bit error probability given a distance and transmit power. The higher encoding bit rate, the higher the probability of error. For example, QPSK encoding has bit error rate of Pb = Ps/2 [2]

1.1.2 Medium Access Control

For a shared wireless medium, transmission nodes are required to contend for the channel. Signal collision happens when two nodes transmit using the same channel, resulting in interference and hence packet lost. To overcome such inefficiency, ad-hoc nodes uses distributed coordinated function (DCF) to contend for signal transmission. This function optimizes for fair usage of wireless medium even in a high utilization rate. [3] One common problem in this mechanism is the hidden node problem.

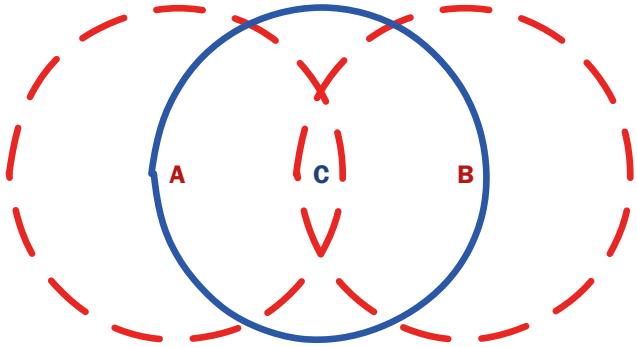

Figure 1: Hidden Node Problem

The hidden node can be illustrated using Figure 1. In this situation, Node A and Node B does not interfere. When node B transmit to node C, node A will not sense the medium being busy, and cause interference, resulting a failed packet transmission in Node C.

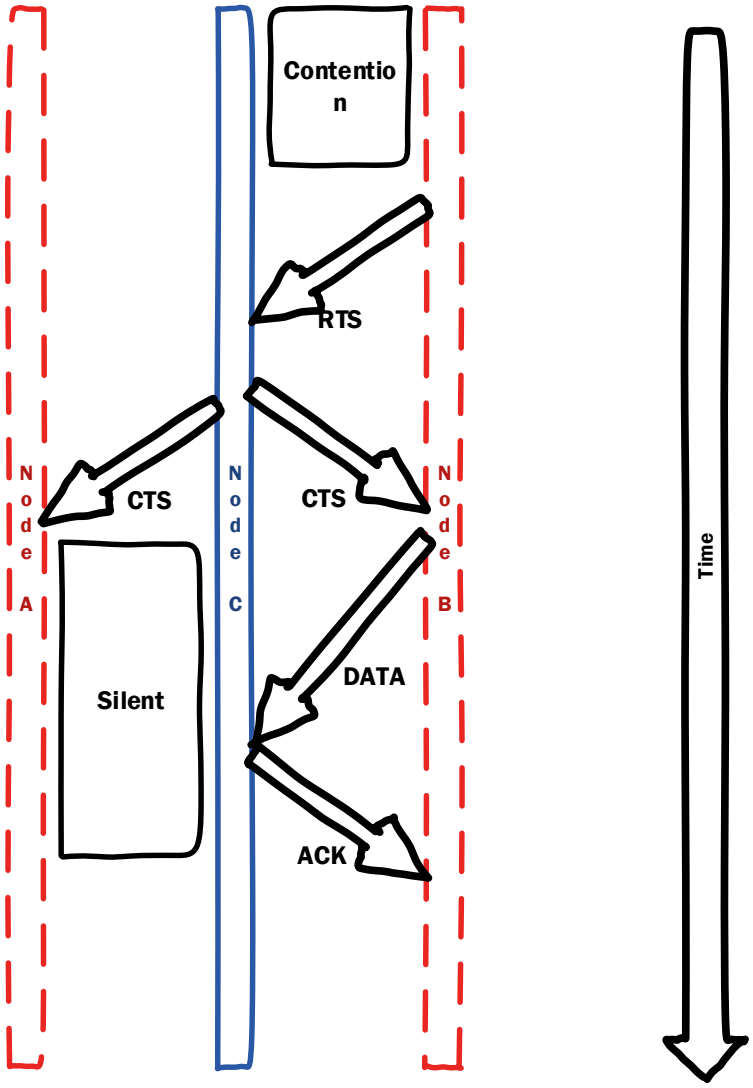

Figure 2 RTS-CTS illustration

To overcome such problem, IEEE 802.11 uses RTS-CTS mechanism. Before a packet is transmitted, ready-to-send (RTS) packet control packet is transmitted to receiver. Receiver will acknowledge RTS request by replying continue-to-send (CTS) packet. The hidden node-A which receives CTS will enter silent period to avoid channel conflict.

Signal collision between nodes in the same channel can be simulated accurately. By simulating signal collision behavior, network engineer can place nodes strategically to optimize a network.

1.2 IEEE 802.11p

The IEEE 802.11p is an extension of existing IEEE 802.11 standard. The Wireless Access for Vehicular Environment (WAVE) and DSRC (Dedicated Short Range Communications) is established in European Countries and United States as a standard for vehicular network. This Federal Communications Commission standard allocated 75MHz spectrum at 5.9GHz bandwidth for vehicular communication [4].

Table 1-FCC Allocation of VANET networks

Frequency (Ghz)	5.850	5.860	5.870	5.880	5.890	5.900	5.910	5.920
Channel Number		172	174	176	178	180	182	184
Uses	Spacing	Critical Safety	Service Channels		Control Channel	Service Channels		Public Safety

Vehicular Network has short link times, thus transmission windows are optimized to exchange data efficiently. Table 1 presents the allocated spectrum and frequency for current IEEE 802.11p standards. Current DSRC standard divides 75 MHz spectrum into 10 MHz channels. Each channel in DSRC is managed. Control Channels (CCH) are reserved for critical messages and Service Channels (SCH) is used for data packets. All nodes switch channels periodically to receive messages from different channels. Since all terminals are required to switch to CCH during the CCH interval, critical messages is transmitted during CCH interval and guaranteed to be received [4].

The MAC Layer of 802.11p is similar to 802.11e. The Enhanced Distributed Channel Access Function (EDCAF) is split into multiple channel access (CA). Each channel maintain a

contention window such that transmit opportunity of one channel does not affect the other channels [5].

The Physical layer of 802.11p uses Orthogonal Frequency Division Multiplexing (OFDM) to make use different channels. IEEE 802.11p states optional enhanced receiver performance requirements; this allows each channel to be encoded in different rate, resulting in different Bit Error Rate (BER). By introducing sub-carrier encoding, each channel is customized to trade-off between transmit range and bandwidth.

1.3 Mobility Modelling

This section discusses simulated behavior beyond the context of networking. These behaviors are necessary model vehicular movement and road traffic scenarios.

1.3.1 Vehicular Area Network

The main task of Vehicular ad-hoc networking is distributing data to other nodes. The distribution is split into three categories. Vehicle to Vehicle transfer (V2V) is concerned in transferring data from vehicle nodes to vehicle nodes. Vehicle to Infrastructure (V2I) focuses on transferring data between vehicular nodes and infrastructure such as internet backhaul or access points. Vehicle to Other Units (V2X) looks at interaction between Vehicular Nodes and other nodes that is not infrastructure or vehicular, such as the road side units.

The data distribution behavior can be modelled by epidemiological model and isolated epidemiological model proposed by Nekovee [6]. The model is computed based on speed, position, and path of the node. Driving behavior and route plays a big role in simulation. The routes can be imported from scenario generator software such as BonnMotion or Simulation of Urban Mobility (SUMO) [7]. This software uses real-world scenario maps to create mobility position and waypoints. The positions and position changes of nodes is simulated on the NS-3

simulator. By using real-world maps, the simulated positions and routes are closely related to real-world scenarios.

1.3.2 Driving Behavior

The mobility models in NS-3 consist of position and routes. The traffic routes are derived using real-world scenarios. Each vehicle in map holds the value of position, velocity, and is considered as a network node. NS-3 simulator uses Treiber's Intelligent Driving Model (IDM) to calculate the change in velocity and position [8]. The driving behavior also incorporates lane changing behavior; the model named "Minimizing Overall Braking decelerations Induced by Lane changes (MOBIL)" is introduced for realistic lane changing behavior [9]. These models have been verified against realistic data derived from experiments [10].

$$\frac{dv}{dt} = a\left[1 - \left(\frac{v}{v_0}\right)^{\delta} - \left(\frac{s^*(v,\Delta v)}{s}\right)^2\right] \tag{1.3}$$

where

$$s^*(v,\Delta v) = s_0 + \max\left[0, \left(vT + \frac{v\Delta v}{2\sqrt{ab}}\right)\right] \tag{1.4}$$

The IDM model in equation 1.3 and 1.4 defines change in velocity as a function of acceleration and current velocity. On a free-road, change in velocity dv, depends on acceleration and deceleration a, comfortable braking value b, current velocity v, desired velocity v0 and desired distance of the vehicle in front s* and actual distance of the vehicle in front s.

$$acc'(M') - acc(M) > p\,[acc\,(B) + acc(B') - acc'(B) - acc'(B')] + a_{hr} \tag{1.5}$$

$$acc'(B') > -b_{save} \quad (1.6)$$

The lane changing behavior simulates decision of drivers to change lane. It first calculates the "incentive" using equation 1.5 based on current lane and "incentive" of changing lane. When changing lane has more "incentive", it checks if it would be safe to change lane using equation 1.6. The *acc'* denotes the acceleration of target lane and *M* denotes 'Me' and *M'* denotes 'Me' after changing to target lane. *B* refers to vehicles in the target lane and *B'* refers to target lane after making the decision. *p* is the politeness factor that weights importance of other vehicles against itself. a_{thr} is the weight threshold to avoid changing lane if conditions are equal or differences are negligible. b_{safe} is the maximum safe deceleration or vehicles of the other lanes.

1.4 Applications

Applications utilizing VANET can be categorized into safety and user applications. Safety applications are essential to enhance driving, while user applications provide entertainment and commerce functionality. Safety applications use CCH while user applications use SCH.

Safety applications enhance user's driving experience by communicating with other nodes or road-side units. This application reduces road accidents, improves intersections, and reduces road congestions.

Vehicles travelling at high speed have very little response time to respond to accidents ahead. Furthermore, drivers do not usually see beyond vehicle in front. The safety application can communicate over a distance, preventing the pile-up accidents from happening. The early warnings give drivers more response time, or even automate the vehicle to stop.

Driving through uncontrolled intersections poses as a challenge for drivers, given limited viewing range, and the need for drivers to look into many directions (e.g., intersections without a traffic light). In VANET, safety application detects and warns drivers of an oncoming vehicle to prevent accident, and vehicles can coordinate in an intersection to prevent collision.

Road congestion can be reduced using VANET. This is done by properly planning the route to destination. Since vehicles are connected, congested road is avoided. It also indirectly reduces traffic accidents [11] because drivers would be less frustrated and more inclined to follow traffic regulations.

User applications in VANET use the Service Channel (SCH) and provide drivers and passengers with network capabilities. This may include internet service on the road, or other network services like commercial advertisements or location directory. VANET appears to be invisible layer, and thus existing applications can be applied to VANET.

The NS-3 simulator is built on top of these rules and behaviors. They are sufficient to simulate real-world conditions and generate data outputs by applying basic networking theories, and applying VANET standard above it. We then model the driving behavior and realistic maps to generate realistic data.

1.5 Thesis Organization

In chapter 2, this chapter discusses current state of the simulation technology, and the proposed solution of this thesis.

Chapter 3 explains about the current simulators that exist and the study of some applications that are commonly used.

In chapter 4, the chapter presents a clear idea about the proposed solution and the methodologies to improve the compute performance.

In chapter 5, the chapter evaluates proposed solution by using common workload between NS-3 simulator and proposed NS-3 simulator.

In chapter 6, an ultimate conclusion of the work done and the future scope for this particular solution is discussed.

CHAPTER 2

PROBLEM DESCRIPTION AND CONTRIBUTIONS

The current VANET Simulator uses NS-3 to simulate networking and driving models. Simulation is the first step of designing networks; network designers optimize Quality of Service by tweaking parameters. Cristea et al [12] states the usefulness of large scale VANET simulators. Contemporary large scale VANET simulation is proved to be time consuming and often energy consuming.

Some solution such as Mobile wireless Vehicular Environment Simulator (MoVES) uses distributed computing to simulate large scale networks [13]. These simulators achieve higher computation throughput by distributing workload across computers, but power and hardware costs for such solutions are expensive [14]. There is also solution proposed by Moritz et al that use mathematical model to optimize simulation by reducing computation workload thus trading off output accuracy [15]. The computation resources play an important role in computation performance.

The solution of this thesis concentrates on how to design energy efficient high performance vehicular area network simulator for large scale networks. In this thesis work we have proposed using General Purpose Graphics Processing Unit (GPGPU) to assist VANET simulation to improve computation performance. This solution is known for large scale and efficient simulation. Many simulators adapted to such solution yield promising speed up[16]. In this thesis work, a Compute Unified Device Architecture (CUDA) is used to offload expensive computations from NS-3 simulator to improve simulation throughput.

2.1 Problem Statement

Simulation of VANETs not only requires the simulation of network standards, but also the mobility of nodes. Such dynamic system involves computation of node distance, routing protocols, application layers, data send, data receive, etc. The simulation model of VANET requires both hardware and software supports to deal with massive computational problems. Currently available network simulators, like network simulator 3 (NS-3), are not adequate for simulating large-scale VANET systems.

2.2 Thesis Contributions

After studying the challenges which are to be surmounted in designing of a large scale vehicular area network simulator, we propose a solution to offload heavy computations to a SIMD processor. The major contributions of this research include:

- A CUDA-assisted VANET simulation model for multicore CPU-GPU platform to increase computational throughput.
- CUDA/C programs for fast simulation of large scale VANET network.
- CUDA/C programs to solve massively parallel big data problems faster.
- Evaluation technique to measure the accuracy of the proposed CUDA-assisted VANET simulator.

CHAPTER 3

LITERATURE SURVEY

This chapter provides a detailed overview of the two main aspects of this dissertation, the NS-3 network simulator and VANET extension of NS-3. Hadi Arbabi and Michele C. Weigle [17] extended NS-3 to simulate VANET based on models described in Chapter 1.

3.1 Network Simulators

Before going into details of NS-3 simulator, this subsection describes the other network simulators available for research and explains why author prefers NS-3 simulator. The simulators studied include OMNET++ [18], NCTuns [19], and iTETRIS platform [20]. These simulators share the same models and objective. Design of simulators varies and inherits different features.

OMNET++ inherits the Eclipse Integrated Development Environment (IDE) to assist users in simulation. It features a graphical user interface (GUI) to design networks. Its open source license allows and users to modify simulator to customize a model. This platform is easy to use because it inherits a powerful GUI.

NCTuns is a network simulator created by Network and System Laboratory in National Chiao Tung University, Taiwan. This simulator features a simulation engine and a loosely-coupled GUI for designing networks. This simulator was intentionally developed as commercial software. Before version 6.0, the simulation engine is open source software, while the GUI is closed source, and the simulator will not work without the GUI client[21]. As of version 8.0, the simulator is fully commercial, and is being marketed as Estinet simulator, featuring ease of use and robust simulation [22].

The iTETRIS platform is a project funded by European Commission, and mainly used by and developed by European nations. This platform serves as a connector for NS-3 simulator and Simulation of Urban Mobility (SUMO) simulator to simulate Intelligent Transportation System (ITS). The platform is being commercially used but developed as open source software. It is distributed by invitation or request only. It features three realms simulation, including traffic management, network communications, and ITS facilities support.

The NS-3 simulator is open source software and fully developed by academia for academia purposes. Software can be modified for specific uses, and due to the large user base, it has large community support. NS-3 does not require a GUI to use, but extensions of GUI are being developed to review simulation results. NS-3 is not compatible with NS-2, it has been written from scratch based on python and C++ programming language. In a recent study [23], the NS-3 simulator yields better performance than OMNET++ simulator. NCTuns is commercial licensed, and researchers no longer have access to the source code. NS-3 is the suitable target because it uses the similar programming language as CUDA. It is also open source; users are free to modify the source code to their needs. NS-3 also inherits the best simulation speed.

3.1 The NS-3 Simulator

NS-3 simulator is a discrete event simulator. Model behavior is simulated by generating events which represent an event happened in reality. For example, when a node sends a packet, Application layer will first generate an event, which will be processed by the Network Layer. Network layer will then generate events to the MAC Layer, and the events cascade until the packet reaches its destination.

```
 1  #include "ns3/core-module.h"
 2 #include "ns3/network-module.h"
 3 #include "ns3/internet-module.h"
 4 #include "ns3/point-to-point-module.h"
 5 #include "ns3/applications-module.h"
 6
 7 using namespace ns3;
 8
 9 NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");
10
11 int
12 main (int argc, char *argv[])
13 {
14  Time::SetResolution (Time::NS);
15  LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
16  LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
17
18  NodeContainer nodes;
19  nodes.Create (2);
20
21  PointToPointHelper pointToPoint;
22  pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
23  pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));
24
25  NetDeviceContainer devices;
26  devices = pointToPoint.Install (nodes);
27
28  InternetStackHelper stack;
29  stack.Install (nodes);
30
31  Ipv4AddressHelper address;
32  address.SetBase ("10.1.1.0", "255.255.255.0");
33
34  Ipv4InterfaceContainer interfaces = address.Assign (devices);
35
36  UdpEchoServerHelper echoServer (9);
37
38  ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));
39  serverApps.Start (Seconds (1.0));
40  serverApps.Stop (Seconds (10.0));
41
42  UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
43  echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
44  echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.0)));
45  echoClient.SetAttribute ("PacketSize", UintegerValue (1024));
46
47  ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));
48  clientApps.Start (Seconds (2.0));
49  clientApps.Stop (Seconds (10.0));
50
51  Simulator::Run ();
52  Simulator::Destroy ();
53  return 0;
54 }
```

Figure 3 Simple NS-3 simulation codes

Figure 3 presents an example of NS-3 simulation; this code simulates point-to-point communication between two nodes. NS-3 simulation is written in C++, it is perceived as a subset of C++ libraries. Line 19 defines two communicating entities, and line 28 and 29 installs network layer stack to created nodes. Line 39 and Line 40 generates an event to start and stop an echo server. Line 48 and 49 generates events to start and stop echo client. Line 51 starts the simulation by generating events. The execution step will remain in line 51 until all events have been completed or Simulator::Stop is called. Simulator::Destroy frees allocated memory. Notice Line 15 and 16 requests the object to log the results; this sends received packets to log file. The parameters (e.g., the network speed and network address) of a network are configured before simulation is run.

```
At time 2s client sent 1024 bytes to 10.1.1.2 port 9
At time 2.00369s server received 1024 bytes from 10.1.1.1 port 49153
At time 2.00369s server sent 1024 bytes to 10.1.1.1 port 49153
At time 2.00737s client received 1024 bytes from 10.1.1.2 port 9
```

Figure 4 Simulation results

The resulting output from the simulation generates simple result. The first packet sent is at 2 second; this is defined at line 48, and it is limited to only one packet at line 43. The delay defined by channel is 2ms, and subsequently, the server received packets at 2ms delay plus latency by layer propagation.

The NS-3 Simulator class is the main access point for event scheduling facilities. When one or more events are scheduled to run (e.g., Simulator::run is called), the simulator class will start processing events. Each event may or may not generate more events, and simulator will stop when none of the events are left to execute, or Simulator::stop is called. [24]

Events are logged in the output of the simulation. Logs are important for precise output. To find out exact propagation delay in example in figure 4, we need a more detailed output. More packets can be logged by changing line 15 and 16 replacing LOG_LEVEL_INFO to LOG_LEVEL_ALL.

```
1  0s UdpEchoServerApplication:UdpEchoServer(0x165023c0)
2  0s UdpEchoClientApplication:UdpEchoClient(0x16502890)
3  0s UdpEchoClientApplication:SetDataSize(0x16502890, 1024)
4  1s UdpEchoServerApplication:StartApplication(0x165023c0)
5  2s UdpEchoClientApplication:StartApplication(0x16502890)
6  2s UdpEchoClientApplication:ScheduleTransmit(0x16502890, +0.0ns)
7  2s UdpEchoClientApplication:Send(0x16502890)
8  2s UdpEchoClientApplication:Send(): At time 2s client sent 1024 bytes to 10.1.1.2 port 9
9  2.00369s UdpEchoServerApplication:HandleRead(0x165023c0, 0x165029d0)
10 2.00369s UdpEchoServerApplication:HandleRead(): At time 2.00369s server received 1024 bytes
   from 10.1.1.1 port 49153
11 2.00369s UdpEchoServerApplication:HandleRead(): Echoing packet
12 2.00369s UdpEchoServerApplication:HandleRead(): At time 2.00369s server sent 1024 bytes to
   10.1.1.1 port 49153
13 2.00737s UdpEchoClientApplication:HandleRead(0x16502890, 0x16502fc0)
14 2.00737s UdpEchoClientApplication:HandleRead(): At time 2.00737s client received 1024 bytes
   from 10.1.1.2 port 9
15 10s UdpEchoClientApplication:StopApplication(0x16502890)
16 10s UdpEchoServerApplication:StopApplication(0x165023c0)
17 UdpEchoClientApplication:DoDispose(0x16502890)
18 UdpEchoServerApplication:DoDispose(0x165023c0)
19 UdpEchoClientApplication:~UdpEchoClient(0x16502890)
20 UdpEchoServerApplication:~UdpEchoServer(0x165023c0)
```

Figure 5 Detailed Output

Detailed output shows in figure 5 line 7 that send function is called, while at line 9 shows Handle Read subroutine is executed. The propagation delay in this case is 3.69ms.

3.2.1 NS-3 Architecture

From previous subsection, we witnessed some usage of NS-3 simulator, it is driven by events and it is based on C++. This subsection explains how the simulation libraries work. The NS-3 internals are categorized into hierarchy.

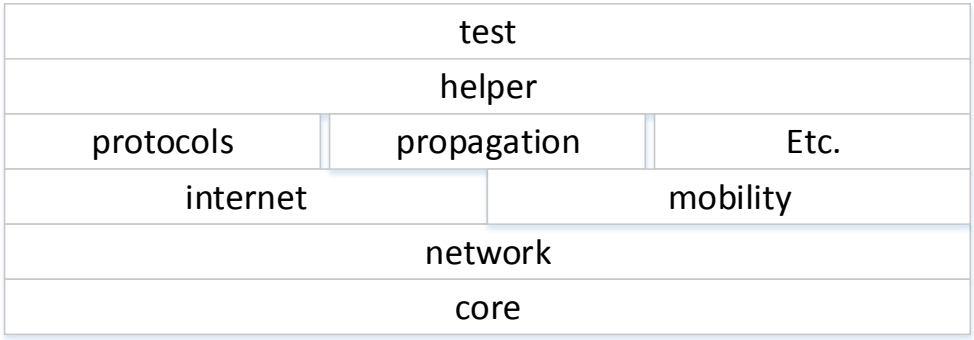

Figure 6 NS-3 architecture

Modules in NS-3 simulator are categorized into libraries. Figure 6 presents the NS-3 architecture. Each library simulates certain functionalities. A simulation scenario is created by combining one or more libraries. In the previous example, Internet, UDP Application, and Point-to-point libraries are used. Each of these libraries simulates a subset of functionality of a network, from the node, to wire, or wireless propagation is simulated using these libraries.

Core module provides the event scheduler, logging and tracing, and random number generator facilities. Event scheduler maintains an event queue, which will be used to run the simulator. The queue is added in chronological order (see figure 7); this serves as the mechanism for NS-3 simulator.

```
148 uint32_t
149 Node::AddApplication (Ptr<Application> application)
150 {
151  NS_LOG_FUNCTION (this << application);
152  uint32_t index = m_applications.size ();
153  m_applications.push_back (application);
154  application->SetNode (this);
155  Simulator::ScheduleWithContext (GetId (), Seconds (0.0),
156  &Application::Initialize, application);
157  return index;
158 }
```

Figure 7 Example callback code

The modules work together by calling back module functions by the core scheduler. Callback works by first storing function pointer to scheduler. Figure 7 presents an example use of callback mechanism commonly used by NS-3 simulator; this is the actual source code excerpt of *node.cc*. In line 155 and 156, simulator scheduled an event to be called, and the actual function is Application::Initialize, which initializes an arbitrary application, such as UDP server in previous example. This mechanism allows developers to extend the modules by using custom callbacks; the module names need not to be known during compile time, allowing modules to be extended by changing the callbacks made.

The network module defines objects and data types such as the node object, channel, and packet. Modules on top of it depend on the network module to define a model. The Internet module extends network module by defining types of packets such as IPv4 and TCP packets. The mobility module constructs node in Cartesian coordinate, allowing nodes to change its position. The helpers are pre-defined configuration for example, the WiFi node; the WiFi node inherently requires free-space propagation modelling and Internet for IPv4 connection.

Further details of this simulator can be resolved from NS-3 documentation [25], but this section is sufficient to further extend NS-3 to simulate VANET environments.

3.3 NS-3 VANET Simulation

Previous sub-chapter discussed about the architecture of NS-3 and how extension can be made possible. This sub-chapter discusses about the VANET extensions.

Michelle Weigle et al developed VANET Highway simulator. This implementation is verified against other models for consistency [17]. This implementation does not use real maps

and only models highway and intersection environments. The simplistic model is sufficient to simulate environments such as four way intersections, but does not use real maps for simulation.

The simulator configuration is stored in XML format. The configuration file stores values such as the speed limit and the length of the highway. This VANET simulator generates a network and a vehicle trace file. The vehicle trace file can be visualized using a JAVA application created by the same author, as seen in figure 9.

```
void HighwayProject::Start() {
    m_vehTrace.open(m_vehTraceFileName.c_str());
    m_netTrace.open(m_netTraceFileName.c_str());
    for(list<Ptr<VehicleGenerator> >::iterator it = m_vehGens.begin(); it != m_vehGens.end();
it++) {
        (*it)->init();
    }
    for(list<Ptr<TrafficLightGenerator> >::iterator it = m_trafficGens.begin(); it !=
m_trafficGens.end(); it++) {
        (*it)->Start();
    }
    Simulator::Schedule(Seconds(0.0), &Step, this);
    Simulator::Stop(Seconds(m_projectXml.GetTotalTimeInSeconds()));
}

void HighwayProject::Step(HighwayProject* project) {

    for(map<int, Ptr<Highway> >::iterator it = project->m_highways.begin(); it != project-
>m_highways.end(); it++) {
        Ptr<Highway> highway = it->second;
        Highway::Step(highway);
    }
    for(map<int, Ptr<Highway> >::iterator it = project->m_highways.begin(); it != project-
>m_highways.end(); it++) {
        Ptr<Highway> highway = it->second;
        highway->HandleTransfers();
    }

    for(map<int, Ptr<Highway> >::iterator it = project->m_highways.begin(); it != project-
>m_highways.end(); it++) {
        Ptr<Highway> highway = it->second;
        for(int i = 1; i <= highway->GetNumberOfLanes(); i++) {
            list<Ptr<Vehicle> >* vehList = highway->GetVehiclesInLane(i);
            if(vehList != NULL) {
                for(list<Ptr<Vehicle> >::iterator it2 = vehList->begin(); it2 != vehList-
>end(); it2++) {
                    Ptr<Vehicle> veh = (*it2);
                    project->m_vehTrace << Simulator::Now().GetNanoSeconds() << "," << veh-
>GetVehicleId() << "," << veh->GetVehicleType() << "," << veh->GetPosition().x << "," << veh-
>GetPosition().y << "," << veh->GetDirection() << ","
                            << veh->GetVelocity() << "," << veh->GetAcceleration() << endl;
                }
```

Figure 8 Source code excerpt

In figure 8, the simulator uses a linked list to store vehicles and uses maps to store highway paths. Highway paths are defined in XML file, for each simulation step, vehicles positions are re-evaluated based on the driving models.

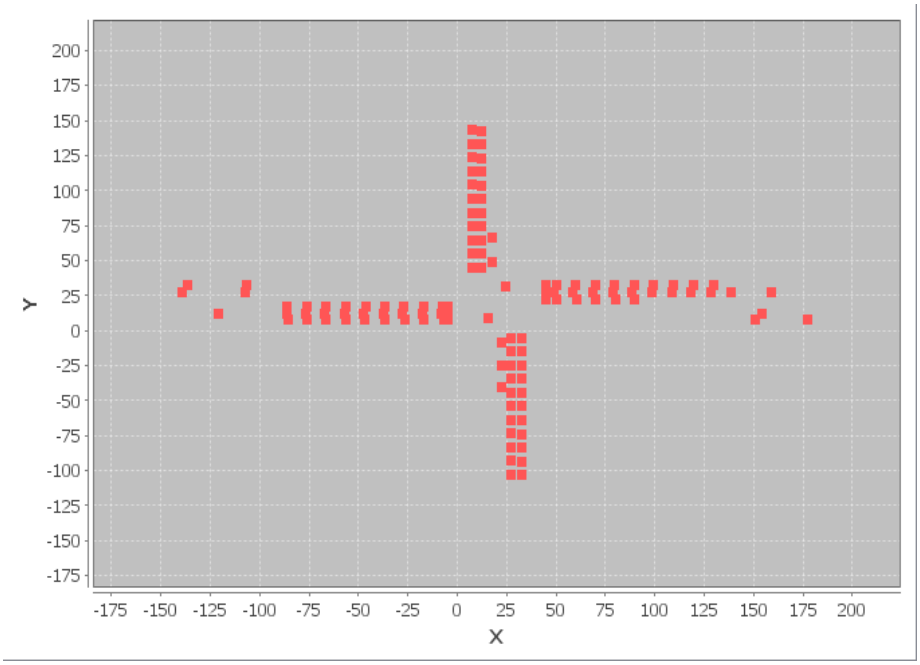

Figure 9 Visualized vehicular trace file

3.4 Large Scale NS-3 Simulation

Large scale network simulations are fundamental part of active networking research. NS-3 supports distributed simulation. This effectively parallelizes process across a network using multiple computers. Pelkey and Riley [26] recent study yield 2.4 times speed up using distributed simulation. The author experimented with node size up to 5000 nodes. The authors also pointed out that synchronization is the big factor to consider when running a simulation for speed up.

CUDA architecture accelerates processes by applying large scale SIMD processors. The details of CUDA architecture will be discussed in next chapter. Swenson et al [27] developed routing models to make use of NS-3 simulator running on CUDA GPGPU processor. This is the

first use of CUDA in NS-3 simulator to simulate large scale nodes. The author translated Floyd-Warshall algorithm to Graph-Matrix format to effectively use GPGPU resources. The author also claims the speed up of 3.5 over simulation of 5000 nodes connected in BRITE topology. The simulation showed promising results, using significantly less resources and yield higher speed up than MPI implementation.

These two outcomes suggest that large number of nodes needs to be parallelized in order to create a scalable simulation. Swenson et al stated the impact of parallelization; in one experiment, a single run took 30 minutes to run. VANET simulation can yield the same speed up using this approach. MPI and CUDA implementation can be combined to yield even better speed up, and would take a lot of effort to develop such solution.

This chapter concludes that NS-3 is the proper platform for this thesis work. The NS-3 simulator yields better performance and is natively supported by CUDA. NS-3 simulator is simple to use, and its architecture allows developers to easily extend the NS-3 simulator functionality. Furthermore, the NS-3 VANET simulation model is developed and verified against other simulators. Finally, large scale network simulation is commonly used and scalable solutions are being researched.

CHAPTER 4

PROPOSED SOLUTION

The main objective of the proposed solution is to improve speed up using CUDA architecture. Previous chapter, we discussed some knowledge of NS-3 VANET implementation allowing us to arrive at this solution. We also briefly discussed about parallelization and methods of implementation. This chapter propose a solution by focusing on parallelization using CUDA and how to integrating the solution into NS-3.

4.1 CUDA

CUDA is a general purpose parallel computing platform and programming language to optimally use NVIDIA Graphics Processing Unit (GPU)[28]. CUDA is perceived as a co-processor to offload programmer's workload into GPU hardware. GPU has the advantage of handling many threads in parallel using the same instruction. A CUDA program which runs on GPU hardware is called a kernel. When kernel is called, CPU can either wait for GPU to complete its computation, or it may continue processing other tasks until kernel execution is complete.

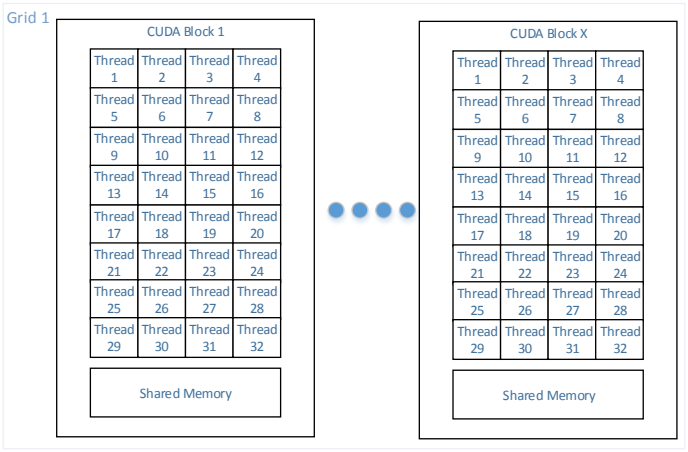

Figure 10 CUDA Logical Organization

CUDA code execution hierarchy is based on Grid, Block, and Threads. Figure 10 is used to visualize the grid, block and thread organization of CUDA architecture. Grid is a group of Blocks; and Blocks are a group of Threads. Each block is guaranteed to execute in parallel in CUDA. Each thread in the same block shares the same Shared Memory and executes the same instruction. The Shared memory is the fastest memory built in GPU Streaming Multiprocessors. Due to the fact that each block of threads are guaranteed to execute in parallel, number of threads in a block is limited to 32 in each dimension, or 1024 maximum number of threads per block[29]. The shared memory is also limited to 64KB for Fermi Architecture. The programmer does not need to be aware of the hardware used in order to get the program to run; but a good programmer needs to know the limitation of the hardware to write an optimal program.

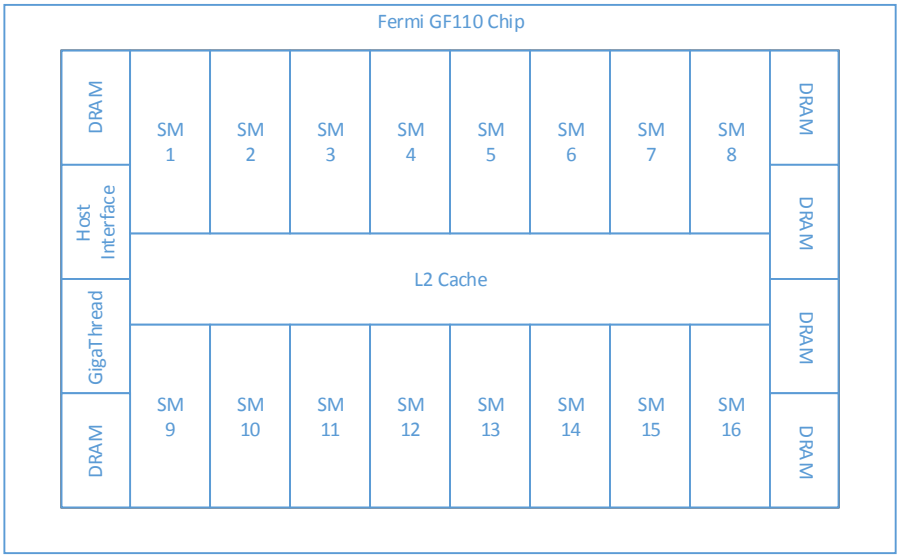

Figure 11 Fermi GPU Floor Plan

Physically, the GPU chip is organized into streaming multiprocessors (SM) and the SMs are surrounded by memory units. This is layout is consistent with the logical organization of the CUDA software; each grid contain multiple blocks where each block populate a Streaming Multiprocessor.

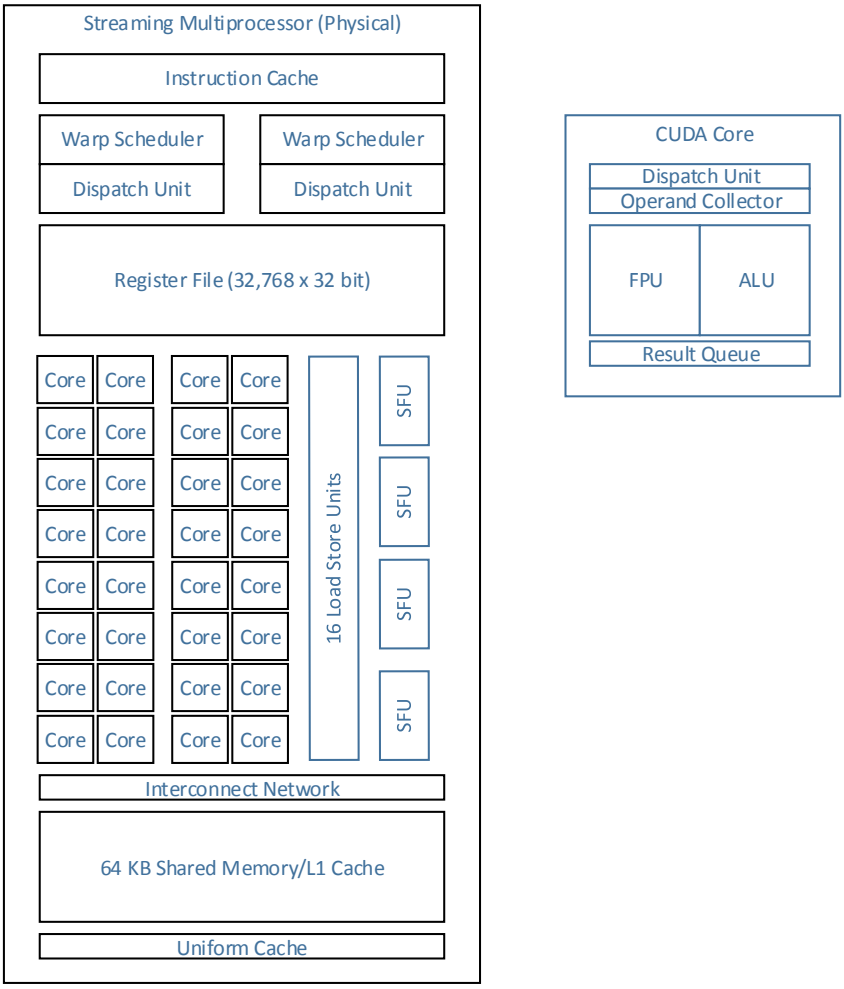

Figure 12 Fermi Streaming Multiprocessor

Figure 12 illustrates streaming multiprocessor in a GPU chip. Each Streaming Multiprocessor contains 32 CUDA cores, 16 load/store unit, Special Function Units, and two warp schedulers. It is also worth noticing that each streaming multiprocessor contains 64KB of configurable memory; this can be used as shared memory, or L1 Cache. The warp schedulers select a core and schedule instructions to run in each core. This guarantees two instructions to be scheduled at any time. Inside each CUDA Core, dispatch unit receives instructions from warp scheduler, and operand collector receives operands from the register file. The Arithmetic Logic Unit or Floating Point Unit will then process the input data and write it to result queue.

The Streaming Multiprocessor is also consistent with the CUDA Blocks. Developers are allowed to use 48KB of shared memory and 16KB of L1 Cache, or 16KB of Shared memory and 48KB of L1 Cache in each block, which is the hardware limit of the SM. Each block is guaranteed to execute in parallel by scheduling instructions to the cores in SM until the all the threads in a block is completed.

F	`__global__ void add(int *a, int *b, int *c){`
	`  *c[threadIdx.x] = *a[threadIdx.x] + *b[threadIdx.x];`
	`}`

	`void main(){`
	`  int a, b, c;`
	`  int *dev_a, *dev_b, *dev_c;`
	`  int size = sizeof(int) * 4 ;`

A	`  cudaMalloc( (void**)&dev_a, size );`
	`  cudaMalloc( (void**)&dev_b, size );`
	`  cudaMalloc( (void**)&dev_c, size );`

	`  a = {1,2,3,4};`
	`  b = {10,20,30,40};`
	`  b = {0,0,0,0};`

B	`  cudaMemcpy( dev_a, &a, size, cudaMemcpyHostToDevice );`
	`  cudaMemcpy( dev_b, &b, size, cudaMemcpyHostToDevice );`
	`  cudaMemcpy( dev_c, &c, size, cudaMemcpyHostToDevice );`

C	`  add<<< 1,4 >>>( dev_a, dev_b, dev_c );`
D	`  cudaMemcpy( &c, dev_c, size, cudaMemcpyDeviceToHost );`

E	`  cudaFree( dev_a );`
	`  cudaFree( dev_b );`
	`  cudaFree( dev_c );`
	`}`

Figure 13 CUDA Code

Figure 13 presents an example CUDA code to add 1x4 matrix. In section A, the memory is allocated inside the GPU. Section B copies matrix A, B, and C to the GPU. At this point, the matrix A and B is already residing in GPU. The GPU kernel is then executed in Section C. The arrow bracket *<<<1,4>>>* orders GPU to execute the kernel with 1 block and 4 threads per block. In this case, there are 4 parallel threads executed concurrently. Section F executes the

matrix add code, the *__global__* identifier tells compiler to compile this subroutine as GPU code. Notice *threadIdx.x* is not a defined keyword, but it is implicitly defined keyword in CUDA; this variable is the thread identifier, each thread in a block has a unique thread identifier. Once execution is completed, Section D copies the resulting matrix C back to host memory. Section E frees the reserved memory. The allocated memory remains in the GPU until *cudaFree()* is called.

4.2 Current NS-3 VANET Implementation

In the previous subtopic, we know that CUDA is efficient in computing SIMD problems. This subtopic explains how Weigle and Arbabi's NS-3 VANET is modified to SIMD CUDA code.

Recall that NS-3 simulation is event based, each event creates a callback; callback executes a subroutine. By translating the subroutines to CUDA subroutines, we can offload the workload to GPU, and process it faster. The VANET simulator classes are organized in hierarchy, with specific purpose for each class. Most of the computational code is held in two subroutine of the VANET simulator, which are the distance computation and movement subroutine.

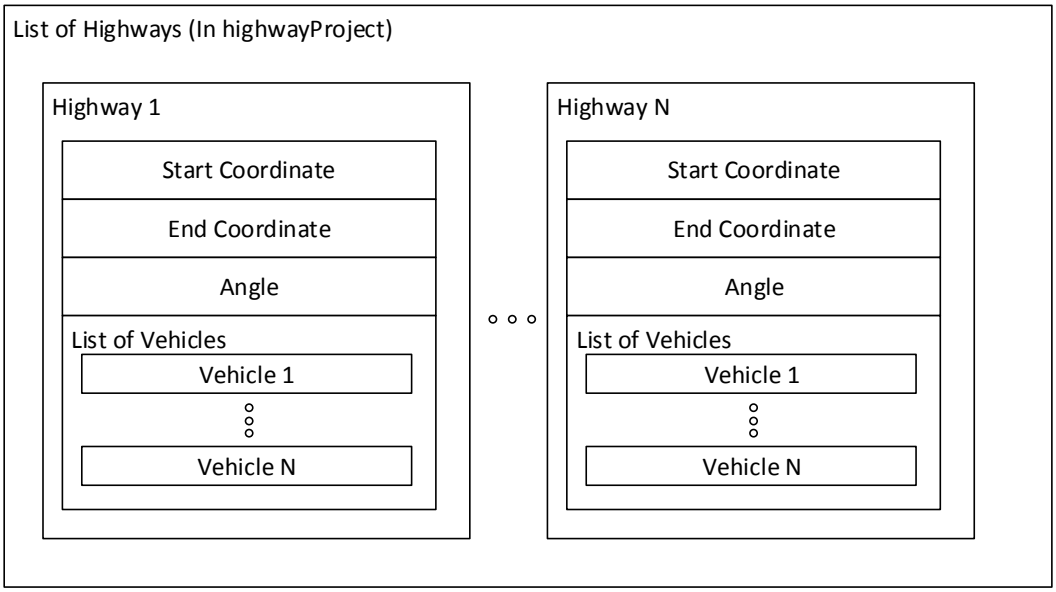

Figure 14 Data Structures of VANET simulator

In Weigle and Arbabi's simulator, mobility module is enhanced to simulate Vehicles, while the default networking stack is used. The highway mobility module is named *highwayProject*. The *highwayProject* instance contains a list of *highway*. Highway object defines highway length, direction, position, lane width, highway position, and list of vehicles. Vehicles and Highways are stored in a linked list. Each instance of *highway* contains a list of *vehicle*s. The *vehicle* object represents a network entity. This node can send and receive packets using the mobility module. The *vehicle* class is responsible for computing distance and change of position.

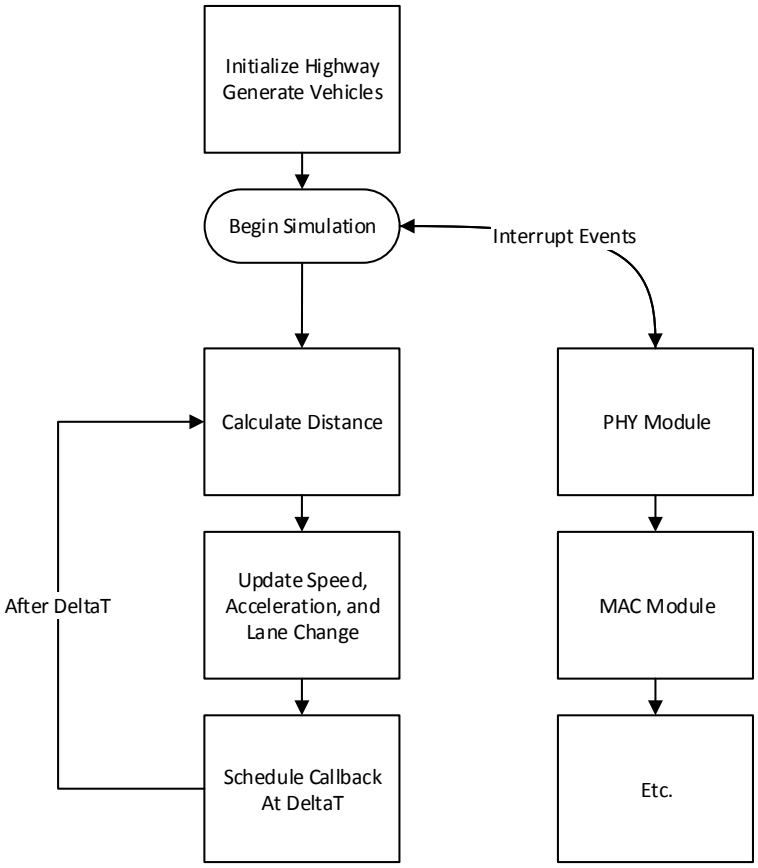

Figure 15 VANET NS-3 subroutine calls

The VANET simulator schedules a callback to calculate distance every nominal time *deltaT*. This value is set to 0.1 as default. This will schedule a callback to re-evaluate position after 0.1s simulation time. Updating speed and changing lane is done after 10 steps of position evaluation subroutine. This will avoid erratic driving behavior and lane changing. The networking stacks are simulated by NS-3 built-in libraries. Each vehicle is installed with an *IEEE 802.11p* model. The networking module can be changed if user prefers to use otherwise.

```
for each N in MAP
  for each I in MAP
    if (distance of I not computed and I not N)
      distance of I = sqrt( Px(I - N)² + Py(I - N)² )
    else
      return distance of I
```

Figure 16 Distance Computation

To calculate the distance between each vehicle, the subroutine visits each vehicle in a linked list and calculates the distance based on Cartesian coordinates as displayed in figure 16. The computational complexity of this algorithm has big-O notation of $O(n^2)$. As number of nodes increases, we expect the computational time to increase exponentially.

4.3 Proposed NS-3 VANET Implementation

The proposed simulator inherits speeds up from CUDA by offloading distance and movement computation. These are the computationally expensive parts of the simulation; while the network simulation can be implemented using CUDA, but it necessarily increases complexity. The GPU has advantage of computing SIMD workload, e.g. matrix calculation, therefore solution involving matrix computation is theoretically fruitful [30].

$$d = \begin{pmatrix} 0 & \sqrt{(A_x - B_x)^2 + (A_y - B_y)^2} & \sqrt{(A_x - C_x)^2 + (A_y - C_y)^2} & \cdots & \sqrt{(A_x - N_x)^2 + (A_y - N_y)^2} \\ \sqrt{(B_x - A_x)^2 + (B_y - A_y)^2} & 0 & \sqrt{(B_x - C_x)^2 + (B_y - C_y)^2} & \cdots & \sqrt{(B_x - N_x)^2 + (B_y - N_y)^2} \\ \sqrt{(C_x - A_x)^2 + (C_y - A_y)^2} & \sqrt{(C_x - B_x)^2 + (C_y - B_y)^2} & 0 & \cdots & \sqrt{(C_x - N_x)^2 + (C_y - N_y)^2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \sqrt{(N_x - A_x)^2 + (N_y - A_y)^2} & \sqrt{(N_x - B_x)^2 + (N_y - B_y)^2} & \sqrt{(N_x - C_x)^2 + (N_y - C_y)^2} & \cdots & 0 \end{pmatrix}$$

Figure 17 Distance Matrix

The coordinates of vehicles in the linked list will first be serialized, and stored as a matrix. The coordinate matrix can then be used to calculate distance matrix from figure 17. For

example, the distance between node A and node B can be retrieved in element *1,2* of the matrix.[28]

$$\mathbf{Pt} = \begin{pmatrix} A_x & A_y \\ B_x & B_y \\ C_x & C_y \\ D_x & D_y \\ E_x & E_y \end{pmatrix}; \mathbf{Mt} = \begin{pmatrix} M_{ax} & M_{ay} \\ M_{bx} & M_{by} \\ M_{cx} & M_{cy} \\ M_{dx} & M_{dy} \\ M_{ex} & M_{ey} \end{pmatrix}$$

Figure 18 Position and Movement Matrix

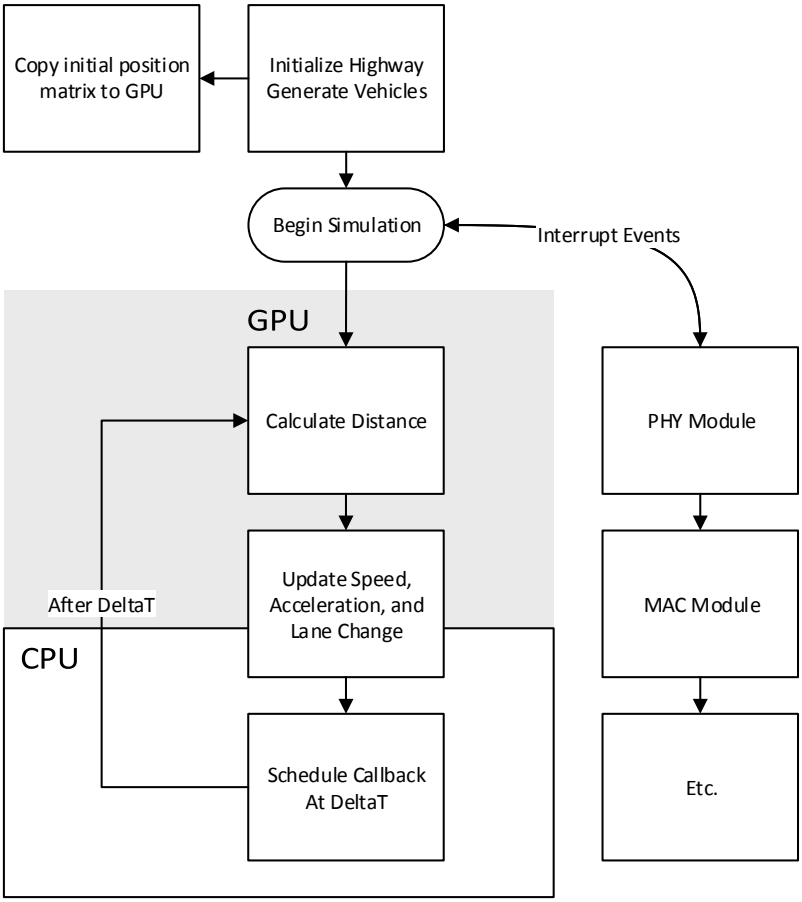

Figure 19 Proposed NS-3 workflow

The position matrix in figure 18, ***Pt*** stores coordinate for each node, and ***Mt*** stores the movement matrix for each step. Figure 19 represents a flowchart of the proposed design. During initialization, ***Pt*** and ***Mt*** are first copied to the GPU. On subsequent steps, the output matrix *Dt* is

copied to CPU for networking stack computation. In any iteration, ***Mt*** is added to ***Pt*** to change the coordinates, simulating the vehicular position change. Using equation 1.3, new movement matrix ***Mt'*** can be computed by substituting ***Mt*** into ***v***. Because ***Mt*** is a matrix, the same operation is applied to the matrix, GPU can use all cores to apply the same operation across all elements. This workload is categorized as a SIMD instruction.

```
__global__ void computeDistanceMatrix(Vector3D *a, Vector3D *b, double *c,
int numberOfNodes){
      int i = blockIdx.x * blockDim.x + threadIdx.x;
      int j = blockIdx.y * blockDim.y + threadIdx.y;
      double dx = b[i].x - a[i].x;
      double dy = b[i].y - a[i].y;
      double dz = b[i].z - a[i].z;
      if ( (i<numberOfNodes) && (j < numberOfNodes)){
            c[i*numberOfNodes + j] = sqrt(dx*dx + dy*dy + dz*dz);
      }
}
```

Figure 20 CUDA code for Distance matrix computation

The subroutine in figure 20 executes a CUDA kernel for distance computation and movement computation is initiated by callback functions. Parallel processing is done by calculating distance matrix for all vehicles concurrently. Each position for a vehicle can then be retrieved from the matrix. This generates a distance matrix, matrix *d*.

$$Mt' = a\left[1 - \left(\frac{Mt}{v_0}\right)^{\delta} - \left(\frac{s^*(Mt, \Delta v)}{s}\right)^2\right] \tag{4.1}$$

The movement vector is also calculated and applied for every iteration. Equation 4.1 is derived from equation 1.3 in chapter 1 to calculate Treiber's IDM in matrix form. Using CUDA programming technique, this equation can be computed in parallel and acceleration matrix ***Mt'*** is generated. Matrix $\boldsymbol{M_t}$ is then used to calculate the acceleration. By applying the change in velocity to movement matrix, we can compute the position of nodes.

CHAPTER 5

RESULTS AND DISCUSSIONS

Chapter 4 described the technique and implementation used to improve the network simulation speed. This chapter verifies correctness of the output, and discusses speed improvement of the VANET simulator.

5.1 Assumption

For verification, the model uses same random number generator, and we assume the same set of movement and random variables are used. This number is particularly important to influence driver's decision to change lanes. The assumption can be made by using the same random generator seed.

For timing experiments, the workstation and server is assumed to have no other users sharing the same resources. This can be done by running the experiments repeatedly for three times and check for consistency. The results can be checked for consistency.

5.2 Experimental Setup

The workload is being executed on workstation and supercomputer. Workstation represents an ideal machine commonly used in lab setup. Supercomputer is commonly used to compute big problems. Table 2 presents the specification of the machines used for this experiment.

Table 2: Hardware Setup

	Supercomputer	**Workstation**
CPU	AMD Opteron 6134	Intel Xeon E5506
CPU Cache Size	512KB	4096KB
Cores	32	8
RAM Size	64GB	12GB
GPU	NVidia Tesla K20m	NVidia Tesla C2075
GPU Architecture	Keper	Fermi
GPU Core Clock	0.71GHz	1.15GHz
GPU Memory Size	5GB	6144MB
GPU Cores	2496	448

This setup shows the strength of supercomputer, the CPU cores and memory are significantly higher than a workstation. While the number of GPU cores in Tesla K20m are significantly higher than Tesla C2075, the clock speed suggests that Tesla C2075 (Fermi) has more advantage for small workload.

5.1 Validation of CUDA Implementation

The developed technique computes change of physical position in a Vehicular Area Network. The simulation uses models of Treiber IDM, and proposed solution solves this modeling equation in matrix form. The movement matrix is calculated using Equation 1.3. Resulting acceleration is added to movement matrix, and then applied to position matrix. The distance between each node is computed using Cartesian distance equation, and stored as a distance matrix format.

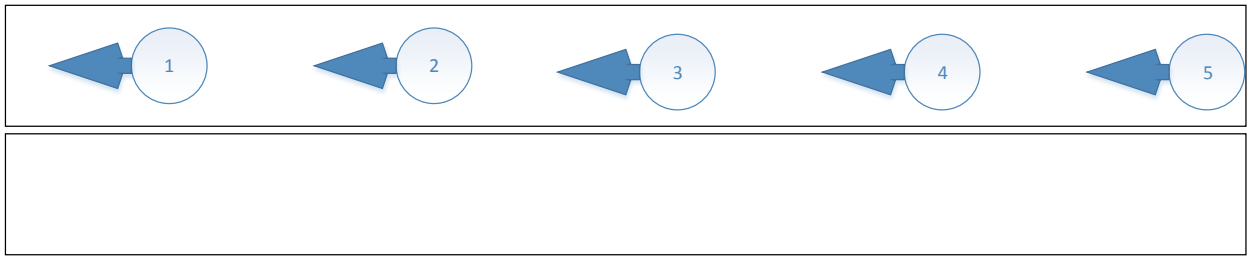

Figure 21: VANET Highway illustrated

$$P_0 = \begin{pmatrix} 1 & 1 \\ 3 & 1 \\ 5 & 1 \\ 7 & 1 \\ 9 & 1 \end{pmatrix} M_0 = \begin{pmatrix} 3 & 0 \\ 3 & 0 \\ 3 & 0 \\ 3 & 0 \\ 3 & 0 \end{pmatrix} P_1 = \begin{pmatrix} 3 & 1 \\ 6 & 1 \\ 8 & 1 \\ 10 & 1 \\ 12 & 1 \end{pmatrix} P_5 = \begin{pmatrix} 16 & 1 \\ 21 & 1 \\ 23 & 1 \\ 25 & 1 \\ 27 & 1 \end{pmatrix} D_1 = \begin{pmatrix} 0 & 2 & 4 & 6 & 8 \\ 2 & 0 & 2 & 4 & 6 \\ 4 & 2 & 0 & 2 & 4 \\ 6 & 4 & 2 & 0 & 2 \\ 8 & 6 & 4 & 2 & 0 \end{pmatrix}$$

$$\text{Distance } (A, C) = \sqrt{(A_x - C_x)^2 + (A_y - C_y)^2}$$

Figure 22 Matrices representing results

On a highway, assuming normal driving behavior, vehicles follow each other separated by a safe distance. Figure 1 illustrate the movement of the nodes in the same direction. The position matrix $\boldsymbol{P_0}$ represents the X and Y coordinate of each node. Each node is represented by a row. This initial position is stored in both CPU and GPU for comparison purposes. The movement matrix $\boldsymbol{M_0}$ is an initial movement matrix. The initial movement matrix for normal driving behavior has constant velocity e.g. no change in speed. In ideal situation, each vehicle moves at the same speed, (3m/s in this case), therefore the movement matrix shares the same element. At second iteration (t=0.2, dt=0.1), movement matrix is calculated; given constant distance and v=3, movement matrix remains the same. The new position $\boldsymbol{P_1}$ is calculated by adding $\boldsymbol{M_0}$. Distance matrix at second iteration is calculated as matrix $\boldsymbol{D_0}$. After five iterations, $\boldsymbol{P_5}$ is computed by repeating the steps.

Table 3: Node Position

Node Number	X position		Y position	
	Before	After	Before	After
Node 1	1	16	1	1
Node 2	3	21	1	1
Node 3	5	23	1	1
Node4	7	25	1	1
Node 5	9	27	1	1

The resulting matrix is then compared with NS-3 output. Table 3 compares the NS-3 result using original output. From comparison, proposed solution and traditional NS-3 generates the same output.

5.3 Isolated Workload

The first set of benchmark isolates workload to only nodes movement. On this workload, only Treiber IDM is modelled, while the network simulation is not executed. This allows us to compare the improvement of CUDA implementation by itself.

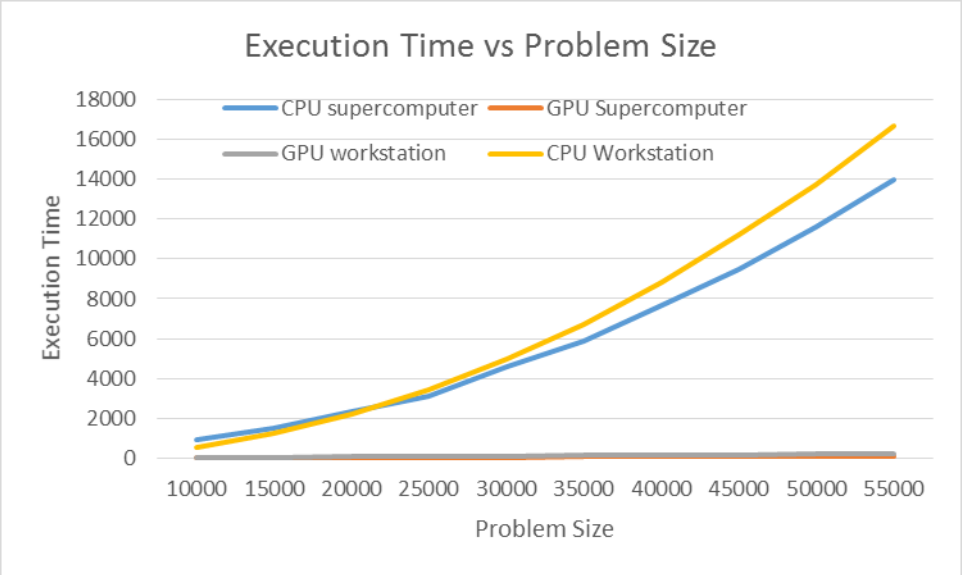

Figure 23 Execution time vs Problem Size

We anticipate more time taken as the number of nodes increases. The workload ranged from 10000 nodes to 55000 nodes. The workload consists of calculating the position based on movement and distance. The workload has compute complexity of $O(n^2)$, a linear increase in number of nodes expected to increase time exponentially. Supercomputer is marginally faster than a workstation for this workload. Since nodes information is stored in a Linked List in CPU implementation, no parallelization is possible for this data structure. By converting the linked list to a matrix, the GPU workload uses all the cores possible.

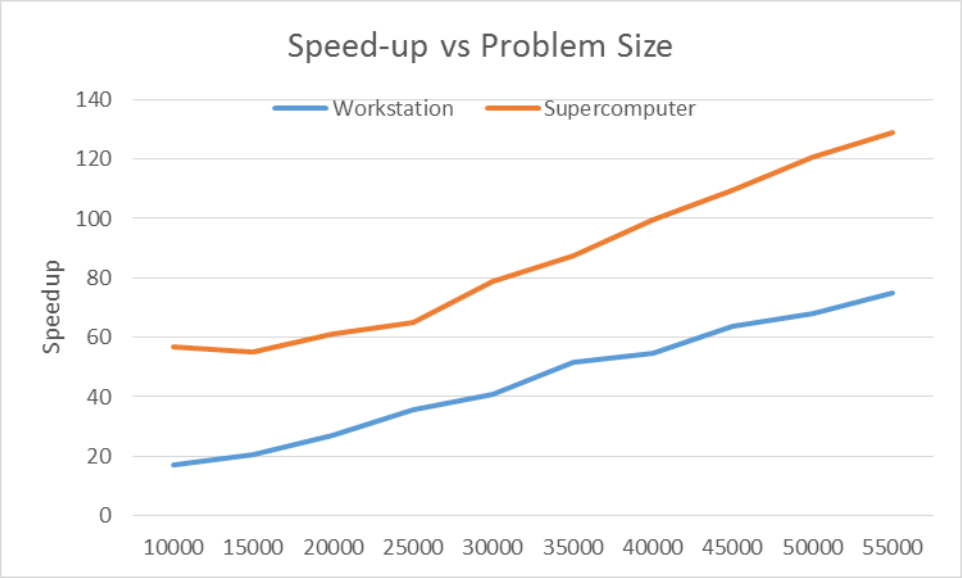

Figure 24 Calculated Speed up

Figure 5.3 represents the speed up over CPU calculated. Speed up improves as workload increases. The CPU implementation results in exponential time increase against problem size. The supercomputer achieved better speed up because GPU has more cores.

5.3 Full Workload

This final workload tests for the full workload for including mobility (vehicular movement) and the network simulation. The network simulation uses only CPU, and does not

use GPU for computation. In proposed solution, as workload increases, CPU becomes the bottleneck of the problems, and speed-up saturates.

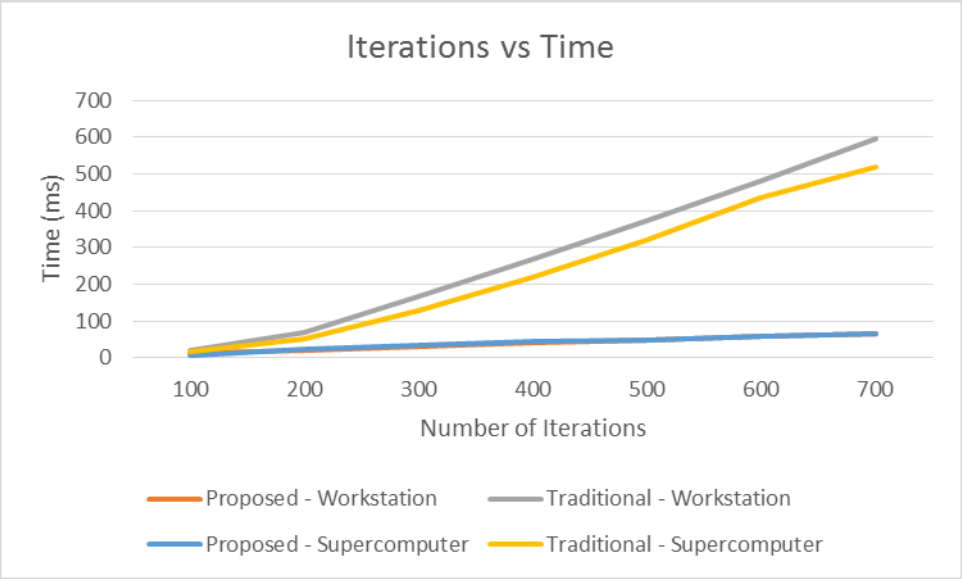

Figure 25 Comparison against iterations

The first scenario in resulting figure 26 simulates 100 nodes and varies iteration, *i.e. dt=0.1, t=10,20,..70*. The proposed solution offloads SIMD operations while CPU consumes data from GPU. Traditional CPU-only implementation requires processor to compute both SIMD and MIMD operations. Supercomputer has faster processor, resulting marginally faster simulation for traditional simulator. For proposed solution, the resulting time taken is almost similar on both platforms.

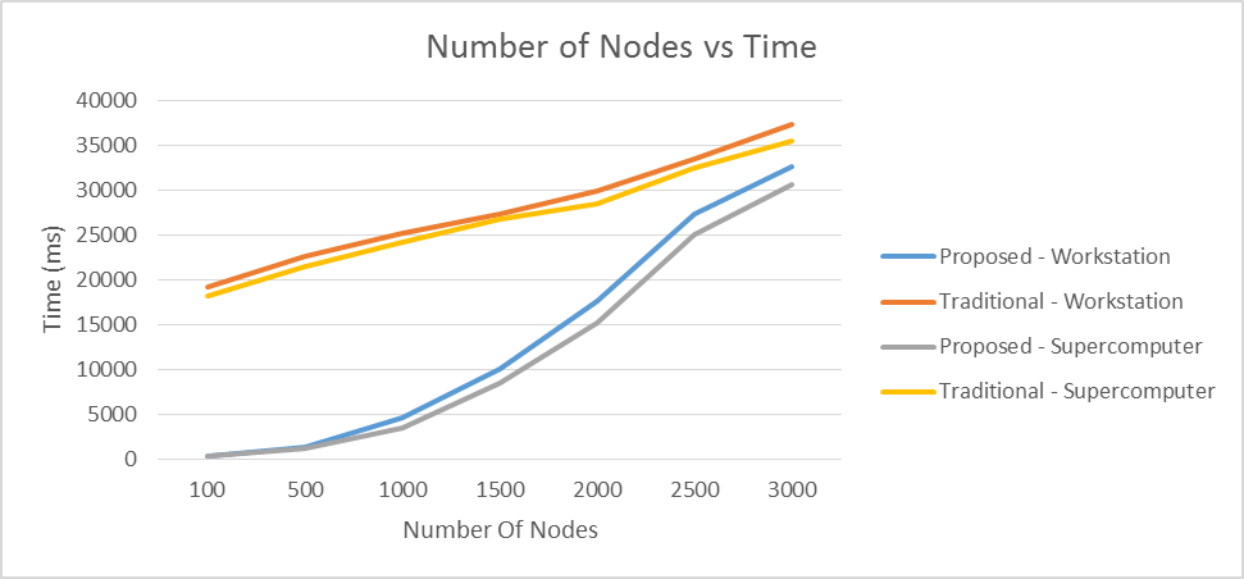

Figure 26 Comparison against number of nodes

When comparing number of nodes versus time, the memory complexity and time increases exponentially. At saturation point (number of nodes = 2500), CPU is becoming the bottleneck; e.g., each iteration waits for CPU to complete before next iteration can be executed. The results show up to 40x increases in speed up. This comparison against single-core CPU and many core GPU is not a fair comparison, therefore the significant speed-up is expected.

5.3 Energy Consumption Analysis

The energy consumed by simulation is analyzed based on the time used to run the simulation. In isolated and full workload traditional simulator analysis, the power is calculated as a function of time used for simulation. For isolated GPU only analysis, only GPU are running at full workload, while CPU is idle. For full workload on proposed simulation, both CPU and GPU are running at full workload.

In isolated workload, the GPU performs calculation without dependency on the CPU. In this best case scenario, the GPU executes at a much faster rate. With the steep speed up, GPU only takes little energy to compute the worklaod.

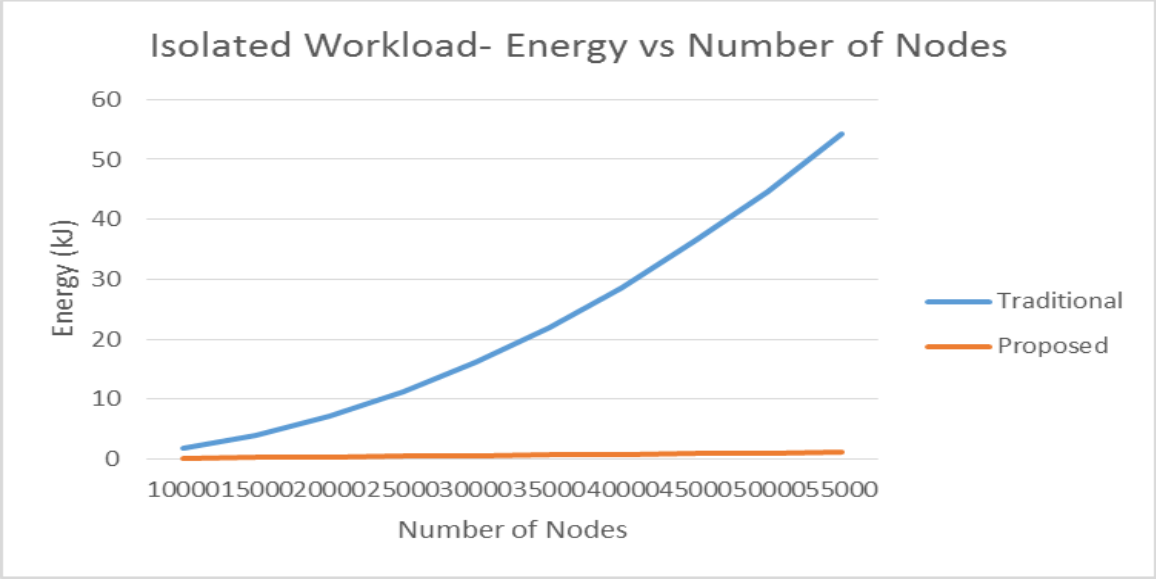

Figure 27 Power Consumed in Isolated Workload

Figure 27 presents the energy consumed by workstation in an isolated workload. Energy required to compute large workload in GPU are less than 1.15 kJ, due to the minimal time consumed by GPU to perform isolated calculation.

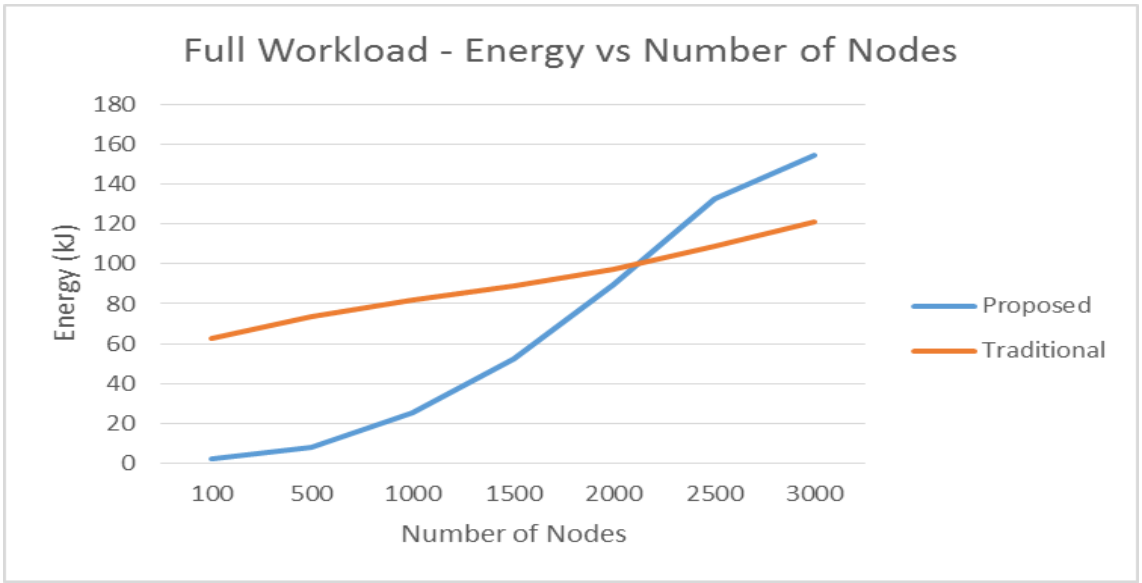

Figure 28 Power Consumed in Isolated Workload

In figure 28, power consumed by proposed solution initially uses 60x less energy to perform same simulation compared to traditional implementation. As workload increases, time

taken to compute increases and speed up saturates at 2000 nodes, where the GPU waits for CPU before an iteration can be completed.

CHAPTER 6

CONCLUSION AND FUTURE SCOPES

We hope the discussion presented in the thesis motivates the interested scholars into considering research in emerging technologies in Vehicular Area Networking and GPU Computing. Vehicular Area Networking poses challenging networking problems and solutions. On the other hand, the use of GPU in simulation gives advantages in throughput.

6.1 Conclusion

By accelerating computation using GPU, we achieve speed up of 75x. We can further exploit this method to develop even faster simulation by integrating more models into NS-3 simulator. CPU had long hit the performance wall, and GPU computation has been a trend to improve efficiency.

NS-3 allows modules to be added or removed due to low-level design. By adding modules which offload task to GPU, simulation throughput can be improved. Many work has been done to make use of GPU, for example the GPU-based simulation models such as BRITE [27] on NS-3. As this trend follows, a higher throughput and more cost-efficient method can be utilized for simulation. By combining multiple GPU-based modules BRITE model [27] will inherently speed up simulation.

New development of NS3 has been using distributed framework such as MPI [26] yields even higher speed up on large scale simulation. With integration of GPU, large scale simulations can benefit higher magnitude speed up.

6.2 Future Extensions

This work makes VANET simulator more efficient. The CUDA implementation can be extended to run including:

- *NS-3 Core Components:* By extending CUDA implementation to NS-3 core, the workload can be completely offloaded by GPU without the bottleneck of CPU. This allows CPU to do further work.
- *Self-Optimization:* Chung et.al. Suggested simulation approach for self-optimizing network. That work focuses on simulating network over different configurations to generate the best results. In his expensive simulation, GPU can be used to simulate events at faster rate [31].
- *Data-Regrouping:* Gummadi et.al. Suggested improving GPU performance by using data-partitioning technique. This technique improves GPU computation performance by regrouping data based on locality principle. This will make GPU computation more efficient [x2].

REFERENCES

REFERENCES

[1] Yousefi, Saleh, Mahmoud Siadat Mousavi, and Mahmood Fathy. "Vehicular ad hoc networks (VANETs): challenges and perspectives." In *ITS Telecommunications Proceedings, 2006 6th International Conference on*, pp. 761-766. IEEE, 2006.

[2] Goldsmith, Andrea. *Wireless communications*. Cambridge university press, 2005.

[3] Bianchi, Giuseppe. "Performance analysis of the IEEE 802.11 distributed coordination function." Selected Areas in Communications, IEEE Journal on 18, no. 3 (2000): 535-547.

[4] Andrews, Scott, and Michael Cops. "Final report: Vehicle infrastructure integration proof of concept executive summary–Vehicle." *US DOT, IntelliDrive (sm) Report FHWA-JPO-09-003* (2009).

[5] Dulmage, J., M. Tsai, M. P. Fitz, and B. Daneshrad. "A Case Study in Incremental Prototyping with Reconfigurable Hardware: DSRC Software Defined-Radio,‖ IEEE Tridentcom." (2007).

[6] Nekovee, Maziar. "Modeling the spread of worm epidemics in vehicular ad hoc networks." In *Vehicular Technology Conference, 2006. VTC 2006-Spring. IEEE 63rd*, vol. 2, pp. 841-845. IEEE, 2006.

[7] Krajzewicz, Daniel, Georg Hertkorn, C. Rössel, and P. Wagner. "Sumo (simulation of urban mobility)." In Proc. of the 4th middle east symposium on simulation and modelling, pp. 183-187. 2002.

[8] Kesting, Arne, Martin Treiber, and Dirk Helbing. "Enhanced intelligent driver model to access the impact of driving strategies on traffic capacity."Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 368, no. 1928 (2010): 4585-4605.

[9] Gluck, Jerome S., Herbert S. Levinson, and Vergil G. Stover. Impacts of access management techniques. No. 420. Transportation Research Board, 1999.

[10] Ibrahim, Khaled, and Michele C. Weigle. "ASH: Application-aware SWANS with highway mobility." In INFOCOM Workshops 2008, IEEE, pp. 1-6. IEEE, 2008

[11] Toor, Yasser, Paul Muhlethaler, and Anis Laouiti. "Vehicle ad hoc networks: Applications and related technical issues." Communications Surveys & Tutorials, IEEE 10, no. 3 (2008): 74-88

[12] Cristea, Valentin, Victor Gradinescu, Cristian Gorgorin, Raluca Diaconescu, and Liviu Iftode. "Simulation of vanet applications." Automotive Informatics and Communicative Systems (2009).

[13] Bononi, Luciano, Marco Di Felice, Marco Bertini, and Emidio Croci. "Parallel and distributed simulation of wireless vehicular ad hoc networks." InProceedings of the 9th ACM international symposium on Modeling analysis and simulation of wireless and mobile systems, pp. 28-35. ACM, 2006

[14] Zhai, Yan, Mingliang Liu, Jidong Zhai, Xiaosong Ma, and Wenguang Chen. "Cloud versus in-house cluster: evaluating amazon cluster compute instances for running mpi applications." In State of the Practice Reports, p. 11. ACM, 2011.

[15] Killat, Moritz, Felix Schmidt-Eisenlohr, Hannes Hartenstein, Christian Rössel, Peter Vortisch, Silja Assenmacher, and Fritz Busch. "Enabling efficient and accurate large-scale simulations of VANETs for vehicular traffic management." In Proceedings of the fourth ACM international workshop on Vehicular ad hoc networks, pp. 29-38. ACM, 2007.

[16] Asmatulu, R., Asaduzzaman, A.; Yip, C.M.; Kumar, S.S.A.;, "An effective CUDA based simulation for lightning strike protection on nanocomposite materials," Southeastcon, 2013 Proceedings of IEEE , pages.1-5, 2013. doi: 10.1109/SECON.2013.6567368

[17] Arbabi, Hadi, and Michele C. Weigle. "Highway mobility and vehicular ad-hoc networks in ns-3." In *Proceedings of the Winter Simulation Conference*, pp. 2991-3003. Winter Simulation Conference, 2010.

[18] Varga, András. "The OMNeT++ discrete event simulation system." InProceedings of the European Simulation Multiconference (ESM'2001), vol. 9, p. 185. sn, 2001.

[19] Wang, S. Y., and C. L. Chou. "NCTUns tool for wireless vehicular communication network researches." Simulation Modelling Practice and Theory17, no. 7 (2009): 1211-1226.

[20] Kumar, Vineet, Lan Lin, Daniel Krajzewicz, Fatma Hrizi, Oscar Martinez, Javier Gozalvez, and Ramon Bauza. "itetris: Adaptation of its technologies for large scale integrated simulation." In Vehicular Technology Conference (VTC 2010-Spring), 2010 IEEE 71st, pp. 1-5. IEEE, 2010.

[21] Simulator, NCTUns Network. "Emulator." (2007).

[22] Wang, Shie-Yuan, Chih-Liang Chou, and Chun-Ming Yang. "Estinet open flow network simulator and emulator." IEEE Communications Magazine 51, no. 9 (2013): 110-117.

[23] Weingartner, Elias, Hendrik Vom Lehn, and Klaus Wehrle. "A performance comparison of recent network simulators." In Communications, 2009. ICC'09. IEEE International Conference on, pp. 1-5. IEEE, 2009.

[24] Henderson, Thomas R., Mathieu Lacage, George F. Riley, C. Dowell, and J. B. Kopena. "Network simulations with the ns-3 simulator." SIGCOMM demonstration (2008).

[25] Riley, George F., and Thomas R. Henderson. "The ns-3 network simulator." InModeling and Tools for Network Simulation, pp. 15-34. Springer Berlin Heidelberg, 2010.

[26] Pelkey, Joshua, and George Riley. "Distributed simulation with MPI in ns-3." InProceedings of the 4th International ICST Conference on Simulation Tools and Techniques, pp. 410-414. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2011.

[27] Swenson, Brian Paul, and George F. Riley. "Simulating large topologies in ns-3 using BRITE and CUDA driven global routing." In Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques, pp. 159-166. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2013

[28] Patterson, David. "The top 10 innovations in the new NVIDIA Fermi architecture, and the top 3 next challenges." NVIDIA Whitepaper (2009).

[29] Alfakih, Abdo Y., Amir Khandani, and Henry Wolkowicz. "Solving Euclidean distance matrix completion problems via semidefinite programming." Computational optimization and applications 12, no. 1-3 (1999): 13-30.

[30] Ryoo, Shane, Christopher I. Rodrigues, Sara S. Baghsorkhi, Sam S. Stone, David B. Kirk, and Wen-mei W. Hwu. "Optimization principles and application performance evaluation of a multithreaded GPU using CUDA." In Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming, pp. 73-82. ACM, 2008.

[31] Chen, Chung Shue, and François Baccelli. "Self-optimization in mobile cellular networks: Power control and user association." In Communications (ICC), 2010 IEEE International Conference on, pp. 1-6. IEEE, 2010