2014

OpenCL Implementation of LiDAR Data Processing

Alexander Bussiere
University of Windsor

Follow this and additional works at: https://scholar.uwindsor.ca/etd

Recommended Citation
https://scholar.uwindsor.ca/etd/5243

This online database contains the full-text of PhD dissertations and Masters’ theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000 ext. 3208.
OpenCL Implementation of LiDAR Data Processing

by

Alexander Bussiere

A Thesis
Submitted to the Faculty of Graduate Studies through the Department of Electrical and Computer Engineering in Partial Fulfillment of the Requirements for the Degree of Master of Applied Science at the University of Windsor

Windsor, Ontario, Canada

2014

© 2014 Alexander Bussiere
OpenCL Implementation of LiDAR Data Processing

by

Alexander Bussiere

Approved By:

_________________________________________________________________________

Dr Z. Kobt
School of Computer Science

_________________________________________________________________________

Dr R. Muscedere
Department of Electrical and Computer Engineering

_________________________________________________________________________

Dr R. Maev, Advisor
Department of Physics
Department of Electrical and Computer Engineering (Cross Appointed)

August 29, 2014
Declaration of Originality

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone’s copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.
Abstract

When designing a safety system, the faster the response time, the greater the reflexes of the system to hazards. As more commercial interest in autonomous and assisted vehicles grows, the number one concern is safety. If the system cannot react as fast as or faster than an average human, then the public will deem it unsafe. In this thesis, I explore the feasibility of using GPU hardware to perform the algorithms used for determining robotic obstacle avoidance. These obstacle avoidance algorithms are ideally suited to reacting to emergency hazards. The product of this research will be a library of OpenCL accelerated functions designed for processing environmental data from LiDAR sensors. The results show that by adopting algorithms to take advantage of the parallel architecture of GPUs, processing times significantly decrease for large data sets.
Acknowledgements

I first have to acknowledge my parents and family, who have always encouraged me to push myself in science and mathematics. I have to thank them for raising me in an environment that encouraged education and curiosity. I also have to thank them for all the food and support.

Secondly I have to acknowledge my Master’s supervisor, Dr Roman Maev, who not only has lead me through my thesis, but has shown the connection between academic success and applications of research. I have to thank him for hiring myself to work in the Physics department when I was in high school. Showing the connection of fundamental research to commercial application has been important in my studies. He has always encouraged research and fostered the ideas of students.

Lastly, I have to thank the Engineering Department of the University of Windsor. Through the department I have had many opportunities in the competitive fields of robotics design, as well as being able to encourage high school students into pursuing engineering through FIRST Robotics mentorship, and as a member of Engineering Outreach Program.
Contents

Declaration of Originality iii
Abstract iv
Acknowledgements v
List of Figures x
List of Tables xi
Abbreviations xii

1 Introduction 1
  1.1 Research Objective ........................................... 1
  1.2 What is Artificial Intelligent Driving? .......................... 3
  1.3 Problems Inherent with Collision Avoidance ...................... 4
  1.4 Importance of GPU Accelerated Collision Avoidance Algorithms, a Brief Review 5
  1.5 OpenCL (Open Computing Language) ............................. 6
  1.6 Related Work ................................................ 9

2 Background Information and Theory 11
  2.1 Understanding GPU versus CPU Execution ......................... 11
  2.2 GPU Accelerated Library ....................................... 13
    2.2.1 LiDAR Depth Data to 3D coordinates ....................... 13
    2.2.2 Data Filtering .............................................. 14
    2.2.3 Global Geometric Translation ............................. 15
    2.2.4 Scale Data ................................................ 16
    2.2.5 Gaussian Blur .............................................. 17
    2.2.6 Divide and Conquer with Commutative Summation .......... 18
    2.2.7 Sort Points ................................................. 19
    2.2.8 Sobel ....................................................... 19
  2.3 Environmental Analysis ......................................... 20
    2.3.1 Magnetic Field Algorithm .................................. 20
    2.3.2 Tentacle Path Analysis .................................... 22

3 Experimental Setup 26
3.1 Research Methodology .................................................. 26
  3.1.1 Software .......................................................... 26
  3.1.2 Hardware ......................................................... 28
  3.1.3 Power Analysis - WattsUp Pro ................................... 30
  3.1.4 Virtual Hardware ................................................ 32

4 Experiment Setup ....................................................... 36
  4.1 OpenCL Accelerated Depth Data Analysis Library ............... 36
  4.2 Environment Analysis Application of OpenCL library .......... 38

5 Discussion ............................................................... 41
  5.1 Overview ............................................................ 41
  5.2 Results - Experiment 1 ............................................. 41
    5.2.1 LiDAR Depth Data to 3D Coordinates ......................... 42
    5.2.2 Data Filtering .................................................. 43
    5.2.3 Global Geometric Translation ................................ 44
    5.2.4 Scale Data ...................................................... 45
    5.2.5 Gaussian Blur .................................................. 47
    5.2.6 Divide and Conquer with Commutative Summation ............ 48
    5.2.7 Sort Points ...................................................... 49
    5.2.8 Sobel ............................................................ 50
  5.3 Results - Experiment 2 ............................................. 52
    5.3.1 Magnetic Field .................................................. 52
    5.3.2 Tentacle Algorithm ............................................. 53

6 Conclusion ............................................................... 57

A Appendix A - Source Code ............................................ 61
  A.1 LiDAR Sensor Data Interpreter- Matlab Source Code .......... 61
  A.2 LiDAR TCP Convert .................................................. 62
  A.3 OpenCL Kernels - Library Functions .............................. 63
    A.3.1 Translation ...................................................... 63

B Appendix B - OpenCL Hardware Overview ............................ 64
  B.1 OpenCL Hardware Attributes ...................................... 64
  B.2 Central Processing Units ......................................... 68
    B.2.1 AMD FX™-6200 .................................................. 68
    B.2.2 Intel® Core™ i5-3230M CPU @ 2.60GHz ..................... 71
  B.3 GPUs ................................................................. 73
    B.3.1 AMD Radeon 7970 .............................................. 73
    B.3.2 NVIDIA GeForce GT 740M ................................... 76

C Appendix C - Tables & Charts ........................................ 79
  C.1 DARPA Ground Vehicle Challenge - Sensors .................... 79
C.2 Charts
  C.2.1 AMD FX\textsuperscript{TM}-6200 - System Wattage ........................................ 82
  C.2.2 AMD FX\textsuperscript{TM}-6200 & Radeon 7970 System Wattage .......................... 83
  C.2.3 AMD FX\textsuperscript{TM}-6200 & Geforce 210 System Wattage .......................... 83
  C.2.4 Intel Core i5 System Wattage ........................................................................ 84
  C.2.5 Intel Core i5 & Geforce 740M System Wattage .............................................. 84

D Appendix D - OpenCL Kernel Source Code
  D.1 Depth Data Coordinate Conversion ................................................................. 85
     D.1.1 3D Data Point Translation - Integer Point Data .............................................. 85
     D.1.2 3D Data Point Translation - Floating Point Data .............................................. 86
     D.1.3 2D Data Point Translation - Floating Point Data .............................................. 87
     D.1.4 2D Data Point Translation - Floating Point Data - ONLY XY- NO ROTATION ... 88
  D.2 Data Scaling .................................................................................................... 89
     D.2.1 2D Polar Scaling - Float ............................................................................. 89
     D.2.2 2D Cartesian Scaling - Float ....................................................................... 89
     D.2.3 2D Polar Scaling - Int .................................................................................. 90
     D.2.4 2D Cartesian Scaling - Int ........................................................................... 91
  D.3 Differentiate .................................................................................................... 91
     D.3.1 Differentiate 2D - Float ............................................................................. 91
     D.3.2 Differentiate 2D - Int .................................................................................. 92
  D.4 Lidar Depth Data to 3D Coordinates ................................................................ 93
     D.4.1 Lidar RAW to Polar - Int ............................................................................ 93
     D.4.2 Lidar RAW to Polar - Float ........................................................................ 94
     D.4.3 Lidar RAW to Cartesian - Int ..................................................................... 94
     D.4.4 Lidar RAW to Cartesian - Float ................................................................. 95
  D.5 Data Filtering .................................................................................................. 95
     D.5.1 Low Pass Filter - Int ................................................................................... 95
     D.5.2 Low Pass Filter - Float ............................................................................... 96
     D.5.3 High Pass Filter - Int ................................................................................ 97
     D.5.4 High Pass Filter - Float ............................................................................. 98
     D.5.5 Band Pass Filter - Int ............................................................................... 99
     D.5.6 Band Pass Filter - Float ............................................................................ 100
  D.6 Gaussian Blur Filter ...................................................................................... 101
     D.6.1 Gaussian 1D - Int ..................................................................................... 101
  D.7 Divide and Conquer Summation .................................................................... 102
     D.7.1 Summation 1D - Int ................................................................................... 102
     D.7.2 Summation 1D - Float .............................................................................. 103
     D.7.3 Summation 2D - Int .................................................................................. 104
     D.7.4 Summation 2D - Float .............................................................................. 105
     D.7.5 Summation 3D - Int .................................................................................. 105
     D.7.6 Summation 3D - Float .............................................................................. 106
Contents

Bibliography 108
Vita Auctoris 110
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>LiDAR Sensor Layout</td>
<td>13</td>
</tr>
<tr>
<td>2.2</td>
<td>2D Translation</td>
<td>15</td>
</tr>
<tr>
<td>2.3</td>
<td>A front-wheel-steering vehicle [2]</td>
<td>23</td>
</tr>
<tr>
<td>2.4</td>
<td>A point inside region, find minimum distance on arc path</td>
<td>24</td>
</tr>
<tr>
<td>3.1</td>
<td>Intel DN2800MT Marshaltown Low Profile Mini ITX Motherboard</td>
<td>28</td>
</tr>
<tr>
<td>3.2</td>
<td>MSI GeForce 210 1GB GDDR3 PCI-Express 2.0</td>
<td>29</td>
</tr>
<tr>
<td>3.3</td>
<td>Hokuyo URG-04LX-UG01 LiDAR</td>
<td>30</td>
</tr>
<tr>
<td>3.4</td>
<td>WattsUp Pro - Power Load Meter</td>
<td>31</td>
</tr>
<tr>
<td>3.5</td>
<td>Velodyne HDL 64E S2</td>
<td>33</td>
</tr>
</tbody>
</table>
List of Tables

2.1 1D Gaussian Blur Constants  .............................................. 18

3.1 Platform 3 ................................................................. 29
3.2 Platform 2 ................................................................. 29
3.3 Platform 1 ................................................................. 29
3.4 Velodyne HDL 64E S2 - LiDAR Network Protocol ................. 34
3.5 Velodyne HDL 64E S2 - Data Packet  ....................................... 35

5.1 Tentacle Algorithm - Process Parallel - 1024 Data Points ........ 53
5.2 Tentacle Algorithm - Process Parallel - $2^{30}$ Data Points ........ 53
5.3 Tentacle Algorithm - Process Parallel - 1024 Data Points ............ 54
5.4 Tentacle Algorithm - Process Parallel - $2^{30}$ Data Points ............ 54
5.5 Tentacle Algorithm - Data Parallel - 1024 Data Points ................. 55
5.6 Tentacle Algorithm - Data Parallel - $2^{30}$ Data Points ................. 55

B.1 OpenCL Attributes - AMD FX™-6200  ..................................... 68
B.2 OpenCL Attributes - Intel® Core™ i5-3230M  .................................. 71
B.3 OpenCL Attributes - AMD Radeon 7970  ................................... 73
B.4 OpenCL Attributes - NVIDIA GeForce GT 740M ........................... 76

C.1 DARPA Ground Vehicle - Sensors and Systems  ......................... 79
C.2 DARPA Ground Vehicle - Sensors and Systems  ......................... 81
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASIC</td>
<td>Application Specific Integrated Circuit</td>
</tr>
<tr>
<td>CUDA</td>
<td>Compute Unified Device Architecture</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field-Programmable Gate Array</td>
</tr>
<tr>
<td>GPU</td>
<td>Graphics Processing Unit</td>
</tr>
<tr>
<td>LiDAR</td>
<td>Light Detection And Radar</td>
</tr>
<tr>
<td>OpenCL</td>
<td>Open Computing Library</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Research Objective

The goal of this research is to explore, implement and compare algorithms for analyzing LiDAR sensor data that would be used to determine optimal path detection based on obstacle avoidance. The algorithms that will be compared is a magnetic field algorithm for omni-drive robotic platforms, and a tentacle path detection algorithm used for holonic drive vehicles. Both of these algorithms can be CPU intensive for large data sets, but have a potential for being applied to multi-core hardware for a parallel computation that would reduce the processing time. The foundations of the analysis algorithms will incorporate an OpenCL based library, a product of this research. This library contains algorithms that are optimized to work with the data sets involved with 3D depth information; transforming, translating, sorting, etc. This library allows for flexibility of hardware, on both CPU and GPU.

Performance of these algorithms will then be compared with the performance of these algorithms implemented with OpenCL to take advantage of the computing capabilities of GPU as well as CPU hardware.

The performance figures that will be compared are the following properties:
Chapter 1. *Introduction*

- Memory Usage
- Wattage
- Processing Time

The ideal results will be a robust algorithm that has low memory usage, low wattage, and minimal processing time. These algorithms will be implemented and tested on a mobile hardware platform with two different hardware setups, plus a desktop hardware setup. The algorithms will process data gathered by a LiDAR sensor. For testing large datasets, virtual hardware will be used. This virtual hardware is data recordings from a Velodyne HDL 64E S2 LiDAR sensor recordings in an urban driving environment.

The objective of this research is to show that the performance per watt can be maximized by using a GPU or multi-core setup with OpenCL library support. Algorithms can be optimized to take advantage of multiple computation cores in various mathematical applications. The common technique for initial start is loop unrolling. Loop unrolling is the process of converting a FOR LOOP which would run consecutively, into a set of streams that would be computed concurrently. In many applications of image processing, this is easily realized as a simple way to execute an algorithm to multiple pixels concurrently, reducing the overall processing time.

GPUs are designed to handle geometric and mathematical calculations required in generating images and 3D environments. GPUs rely on massive hardware multi-threading to keep arithmetic units occupied. In general computing, this architecture to handle massive number of streams can be applied to mathematical operations on large sets of data. [1] The comparison between a CPU’s and GPU’s operational performance difference can be measured in FLOPS. The following table breaks down the performance comparison:

Comparing these specifications, it is a simple conclusion that the performance, price and performance/watt is an advantage of the GPU over the CPU. The advantage that the CPU
has is the core frequency. The higher the frequency the more operations per second a single thread can perform. The goal is to show that this performance can be gained in the application of path detection algorithms to OpenCL hardware.

1.2 What is Artificial Intelligent Driving?

A simple definition of artificial intelligence is the study of making computers do things which, at the moment, humans do better [6]. One problem with this definition is that it assumes that computers are capable of processing the same way human minds do, i.e. diagnose, advise, and understand. This problem can be avoided by saying that artificial intelligence is the development of computers whose observable performance has features, which, in humans, we would attribute to mental processes [6]. Artificially Intelligent driving is commonly associated with autonomous driving, but the AI systems do not need to be fully autonomous to be considered intelligent.

Many systems have been developed that are in line with the above definition, such as systems for medical diagnosis, navigation and image recognition. However the holy grail of artificial intelligence research is not merely to create systems that can carry out complex functions, but to create systems that comprehend what it is that they do [3]. As the method of teaching often used with children states: learn first, understand later. This means that in order to understand our environment we must first know it, and it is this learning step that this thesis is concerned with. Building a map of the environment a robot must operate in is a method of organizing and validating the information the robot can extrapolate using both its sensors and past information. The task of comprehending the stored information is beyond the scope of this work.

Many commercially available vehicles are featuring systems that would be classified as artificially intelligent systems; 2013 Ford Fusion offered an adaptive cruise control system that features lane keeping system, automatic parallel parking and accident avoidance systems. This system still requires the driver to maintain control of the vehicle in use, but sensors analyze the environment and determine and correct driver mistakes when in adaptive cruise control mode. The 2014 Mercedes S-Class features autonomous steering, lane keeping, and
accident avoidance. Many other manufacturers have announced plans for commercial availability of autonomous vehicles; 2015 Audi, Cadillac, and Nissan will have a vehicles which can autonomously steer, break and lane keeping. By 2025, Volvo, Mercedese Benze, Audi, Nissan, BMW, Chrysler, and Ford are planning to have fully autonomous vehicles. The IEEE predicts that by 2040 75% of all vehicles will be autonomous.[12].

In the field of autonomous robotics, many of the research platforms involved are nonholonic drive systems (one axle is steerable), or omni drive (can change the direction of motion on the spot). These platforms range from very small mouse sized platforms to large commercial vehicle size. Military and commercial autonomous, semi-autonomous, and remote controlled drones are increasingly being used in real world applications.

1.3 Problems Inherent with Collision Avoidance

Artificially intelligent driving algorithms typically deal with large amounts of data. This data must be processed in real time, with minimal delay from input to output, in order to provide a safe system. These driving systems typically involve a combination of LiDAR, ultrasonic, and image based sensors. To maintain real time performance with devices that are continually getting better in terms of increasing data rates, larger data sets, while the need for portable, and lower powered devices are needed. GPU acceleration provides a method for concurrent computing, and reduce the load on the CPU (which was experimentally determined to require more watts per GFLOP). Studying the final designs in the DARPA Ground Vehicle Challenge, the primary method of detecting the vehicles surrounding objects was a combination of video cameras and LiDAR sensors.

For commercial potential, consumers will need to feel confident in the safety of a system that is taking control of the vehicle. This means proving the safety performance of the algorithms and the reliability of the intelligent systems. In order to come to market faster, the time needed to take the algorithms from theoretical to practical needs to be reduced, as well as allowing for easy modifications during the testing. While end goal of the research stage would be a dedicated piece of hardware (An ASIC chip), the need to change the algorithm,
or parameters during the research stage is a necessity.

In other applications, the need for a platform that can be programmed according to known environment might be necessary. The ability to change path detection algorithms could be necessary given the environment of the application. GPU implementation of the algorithms necessary could provide a reduced development time to reach a prototype stage. Algorithms can be tuned specifically to the application before final implementation. The diversity of GPU’s available can provide a variety of devices to choose between, allowing for low powered GPU’s to be used in applications where less processing power is an option.

1.4 Importance of GPU Accelerated Collision Avoidance Algorithms, a Brief Review

The importance of collision avoidance in autonomous vehicle implementations is simple: in a dynamic environment a vehicle must be able to avoid objects (static and dynamic) in real time with assistance from a virtual mapped environment, from real time sensor data, or both. Collision warning systems can include functions such as forward collision warning, blind spot warning, lane change/merge warning, lane departure warning, backup warning, rear impact warning, roll over warning systems, and adaptive cruise control. The information gathered from these algorithms can then be used to control the vehicle, and prevent such a collision from occurring.

As the data from the sensors gathering information about the vehicles environment get more detailed and precise, the amount of data to be processed becomes larger. Algorithms such as the Fortune’s Sweep Line Voronoi path detection algorithm require exponentially more calculations as more data points are introduced. The more calculations require more calculations per seconds to maintain real time performance. Combining data from different sensors, possibly a heterogeneous setup, requires transposition of the data before analysis. These types of operations are an ideal candidate for concurrency. Breaking down the algorithms used in collision avoidance systems into task or data parallel algorithms, GPU’s
can be used to massively parallelize the computations involved in the processing of this data.

GPU accelerated code can provide a major advantage for researchers. Mobile test platforms need to minimize the wattage of computing platforms, the processing power needs to be maximized, and physical size of the computing platform must be minimized. The mobile research platforms have limited power and space, and due to financial as well as time restrictions, dedicated ASIC implementations might not be feasible. Commercial GPU's can provide a cost efficient computing platform, and provide an attractive hardware option.

Some of these systems have been implemented in commercial vehicles. Adaptive cruise control has been implemented by most major manufacturers, but most are limited to constantly motion traffic. Only BMW has an adaptive cruise control system that is capable of stop-and-go traffic situations. Collision avoidance systems are currently being developed by General Motors, funded by USDOT (35M prototyping rear-end collision avoidance system) [2]. Backup warning systems using ultrasonic sensors are currently offered by most vehicle manufacturers, many now offering camera based feedback for the driver. These vehicle backup cameras still require humans to process the images. These collision detection systems have been tested and shown to improve accident rates (upwards of 20 decrease in accidents shown in experimental test of 7500 vehicles in Sweden). The question is no longer whether it is worth implementing these systems, and it has changed to how to effectively implement these systems.

1.5 OpenCL (Open Computing Language)

OpenCL (Open Computing Language) is an open framework developed by Khronos Group, in collaboration with technical teams at AMD, Intel, IBM and NVIDIA. Apple submitted the original proposal for OpenCL in November 2008 to Khronos, and together the final public release was in December 2008. OpenCL was developed to be implemented on heterogeneous platforms based on central processing units (CPU), graphical processing unit (GPU), digital signal processors (DSP) and other processors. OpenCL API's have been written for C, and C++. 
A competing framework, CUDA, was developed by NVIDIA. OpenCL was chosen over CUDA based on the fact that CUDA code can only be executed on NVIDIA GPU’s, i.e., a non-heterogeneous computing system. With OpenCL 1.2, support has been included to execute code on CPU + OpenCL supported devices. The advantage of applications written with OpenCL support is the ability for the kernel code to be supported on many different architectures and setups. The kernel code written to be executed on a GPU can be run on the CPU, with the limitation that the main code is written to look for both types of devices. This allows the same code to be tested on many different configurations, and is especially useful in the research in this thesis. Support has also been made through the PortableCL library, which is a portable support library that offers the ability for kernels to be run on ARM, x86, x64, as well as PowerPC processors. The PortableCL library was developed by MIT, and offers C platform support for portable devices. The timeline for official support is unknown, but intent has been announced.

The OpenCL framework consists of three components: the platform layer, the runtime, and the compiler [3]. The platform layer allows developers to gather information about OpenCL-capable devices and create contexts on said devices. Devices can be GPUs, multi-core processors, or any other device that supports OpenCL [4]. Developers can query the number of devices, a specific device’s vendor, model, or other information. Additionally, developers can query specific architectural details, such as cache sizes, how shared memory is implemented, shared memory size, etc. This can be used with the OpenCL compiler layer to select the device for which a given kernel should be compiled. The OpenCL compiler maps abstract kernels onto a device-specific architecture. Kernel source code is passed to the compiler during an application’s runtime and is compiled and linked into an image that can be executed by a device. This paradigm of runtime compilation is used in graphics shader languages (such as GLSL and HLSL) to increase portability. An OpenCL application can safely be moved to a different machine without static recompilation, since device-specific binaries are recreated at runtime. To run compiled kernels, developers use the OpenCL runtime layer.

The runtime layer provides functions for managing device memory, running kernels, and
transferring data to devices. Tasks can be issued asynchronously, so the runtime provides mechanisms for ensuring synchronization when necessary. When kernels execute, one instance is run for every point in a defined index space, as described in the OpenCL specification. Each kernel instance is a work-item. Workitems are organized into clusters called work-groups. Within a work-group, work-items can share data in local memory and all work-items within a group execute on the same multiprocessor. On multicore processors, this can be used to control cache sharing between work items. Work groups can similarly be used to share data using shared memory on a GPU. The index space used in OpenCL is similar, though not identical to the grid construct used in Nvidia’s CUDA language [5], [4]. OpenCL kernels are written in a superset of the C99 standard with extensions to support data parallelism [4]. The OpenCL language supports vector data types, such as float4 and int16. These types can provide performance benefits on architectures with SIMD instructions, such as the x86 (through SSE), Larrabee [6] (which has 512-bit SIMD instructions), and Cell SPE instruction sets [3]. Vector types are always aligned on a memory boundary equal to their size in bytes. Function intrinsics exist for synchronizing threads within a work-group and fetching a thread’s work-item.

The other advantage to this library is the ability to concurrently execute multiple kernels. While this is not explored in this research, it is an important ability as it makes it possible for future systems to concurrently process LiDAR data as well as process image data. This can be attained through heterogeneous architecture in three ways. The first method is to execute one kernel on one device, and another kernel on a separate device. A second method is to allocate different computing cores to different kernels. The third method is a combination of the first two methods. This is an attractive feature for this field, as different sensors could be processed in parallel on the same hardware. An example of this in practice would be a vehicle platform that has multiple camera’s and multiple LiDAR sensors.
1.6 Related Work

While the application of OpenCL libraries to work specifically with LiDAR data has not been explored at the point of this research work, there has been a lot of research and application of OpenCL and CUDA libraries to the area of image processing. Image processing is an importan aspect of intelligent and autonomous driving. OpenCV (Open Computer Vision) library has implemented CUDA GPU libraries to the processing of vision sensor data [3]. These functions are related to video processing, stereo-vision acceleration, and various filters and matrix summations. The choice for OpenCV to use CUDA as the hardware acceleration library was due to the maturity of CUDA versus the immaturity of the OpenCL library at the time. Many research applications in the field of robotics have used CUDA to accelerate the processing of robotic vision sensors [7],[8].

Researchers have also explored the comparison of CUDA vs OpenCL in accelerating functions and have shown that in most cases the performance difference was deemed negligible (less than 10%) and in the cases there was a difference, OpenCL was the better [9]. More interesting is the performance for smaller data sets, in this sense referring to data set sizes smaller than 5000 data points. In the results of [9], OpenCL computation had a lower performance for smaller data sets than the CUDA version, for both GPU and CPU implementations.

In the field of intelligent driving, many researchers have developed algorithms for interpreting environmental data, and computing a vehicle trajectory for safe movement in known and unknown environments. Vehicles involved in these research endeavors range from large vehicles [7] to small omni-drive robotics [10]. These vehicles have limited power and space available for computing platforms. The DARPA 2010 vehicles all used multiple computer systems to spread the computation load across. They used multiple LiDAR and vision sensors, producing millions of data points every second, of which they need to be processed in real time. The vehicles that did the best in the competition all relied on multiple server racks to handle the data loads, and consuming as much as 8000 watts of power [8]. The
International Ground Vehicle Competition is similar to the DARPA Ground Vehicle Challenge, with the notable difference being in the size of the vehicle. With much more limited power and space, the number of sensors used far less [referen]. Only a few teams used FPGA chips to provide hardware acceleration for either their vision or LiDAR data sets. One team explored the possibility, and deemed that it was impractical given the time and experience constraints the team had [8].
Chapter 2

Background Information and Theory

2.1 Understanding GPU versus CPU Execution

On runtime of the OpenCL enabled program, the program must set up the information about the hardware available and essentially customize the distribution and set up of the kernel code. This kernel is the instruction set that will be loaded into each process or core (NVIDIA or ATI/AMD have different terminology for their hardware). The program gathers information about the number of parallel cores, the maximum number of dimensions, memory, and many others. These variables are used to customize the performance of the functions during runtime, but these specifications can also help select the hardware platform. These specifications can be determined before kernel compilation, but they are also made available by many of the manufacturers in their documentation. Research into these specifications can help select a hardware platform that matches the research application. For example, some devices support different vector widths, and this can cause certain algorithms to run slower if the vector size available is less than the vector size used. Specifically listed here [3], [11], cite19 in Appendix B

This available information of the hardware can allow the main code to adjust the method of execution, and optimize based on hardware available automatically. Parameters such as local memory available can allow the adjustment in the execution of the kernels on CPU’s, where the availability of local memory is minimal (for this hardware it is local cache, that
might only be kilobytes or one or two megabytes in size). This very limited local memory makes it important for the kernels to not overload the local cache with data, and focus on loading the data to global memory structures, which is typically the available RAM. The available RAM in the system can by gigabytes instead of kilo/megabytes, allowing for much more information to be loaded. Exact implementation difference are explored further in the final implementation of memory structures used in Chapter 4.

Parameters such as maximum clock frequency, and maximum number of work items can allow the software to intelligently decided which hardware should be used and how it should be used. Part of the intent of this research is to explore the cross over points where the standard C++ implementation of algorithms is better than OpenCL versions executed on CPU and GPU. This can allow for a library to be optimized on various hardware without intervention of the programmer. There have been a lot of research that shows that the parallelization of the GPU in OpenCL can greatly improve the execution and performance of code in various fields. By combining this capability, with an intelligent decision making about the type of kernel execution, an efficient library can be put together that can adapt to the variety of hardware implementations available in the field of robotics.

The parameters regarding partitions are also important to efficiently researching the abilities of OpenCL to various hardware types. One method of research would be to test a multitude of GPU’s with a range of memory sizes and number of parallel cores/ processes. The method used in this research is to take a GPU that supports a large number of parallel cores (for this research the ATI Radeon HD 7970 was the GPU selected for this research method) and partition the device so only a certain number of these parallel cores can be used. Essentially, the cores are being ignored in software. For lower performance GPU’s, this is done in hardware, by turning off cores that have imperfections in the manufacturing. This method of testing is available for OpenCL devices that offer support for OpenCL 1.2 specifications.
Chapter 2. Background Information

2.2 GPU Accelerated Library

A library of functions needed for optimizing the functions used in path detection was a necessary first start. The following sections explain the functions accelerated, their uses, and the version created. Combinations of these functions will be used later for accelerating the two versions of a complete path detection algorithm.

2.2.1 LiDAR Depth Data to 3D coordinates

This algorithm converts the depth data received from the LiDAR to either Cartesian or Polar. The sensors send the data serially, starting from beginning angle, incrementally until the maximum angle is reached. The data sent is only the depth information, without any position information, assuming the point only contains the information $P(d_i)$. Figure 2.1 shows the layout of the sensor and the sensor data. $P_i$ is the data point that will be converted to polar coordinates. $\theta_i$ refers to the angle from the vertical axis. $\theta_{\text{min}}$ and $\theta_{\text{max}}$ refer to the minimum and maximum scanning angles of the LiDAR sensor. $D_i$ is the depth of the point $P_i$. Finally, $i$ is the index of the point.

![LiDAR Sensor Layout](image)

**Figure 2.1:** LiDAR Sensor Layout
The conversion uses the following equation to convert the received data to polar coordinates with the simple equations (2.1) and (2.2) and will result in the data \( P(\theta_i, d_i) \).

\[
\theta_i = \theta_{\text{min}} + \frac{\theta_{\text{max}} - \theta_{\text{min}}}{n} \times i
\]  
(2.1)

\[R_i = d_i\]  
(2.2)

The other conversion is from the LiDAR depth information to Cartesian Coordinates, \( P(x_i, y_i) \). The equations (2.3) and (2.4) show the conversion to Cartesian.

\[X_i = d_i \times \cos \theta_i\]  
(2.3)

\[X_i = d_i \times \sin \theta_i\]  
(2.4)

While the actual conversion is a simple algorithm, the lack of logical comparisons and branches makes it an ideal algorithm to convert from sequential calculations to parallel. In application, each thread will compute the conversion of each depth point from the sensor. Experimentation will determine the crossover point between the number of points in a dataset needed for GPU / CPU parallelization is more efficient than CPU sequential computation.

### 2.2.2 Data Filtering

A simple operation of filtering out data points that are returned as null points can reduce the size of the data set, as well as make allow other algorithms to be executed without additional logical operations can help enhance the performance of an application. By filtering out data points known to be invalid, other algorithms can make better assumptions. For example, the LiDAR may return a value of -1 for a data point that may not have reflected properly, and unable to return a depth point. Eliminating these points from the arrays can allow the elimination of the IF statements in other algorithms, ie converting from polar to cartesian does not have to check for negative depth values.
Three algorithms were produced, a high pass filter (allows for values over a minimum value to pass), a low pass filter (allows for values under a maximum value to pass), and finally a bandpass filter (allows for values above a minimum and below a maximum to pass).

### 2.2.3 Global Geometric Translation

When adding points from a LiDAR or similar sensor to a point cloud, 3D translation and rotation are necessary actions to perform on the complete data set before inserting into the point cloud. Four versions of this algorithm were produced, a 2D and 3D translation, both in Cartesian-to-Cartesian and polar-to-polar. The format of the array is shown in (2.5) for three dimensions and in (2.6) for two dimensions.

\[
[\Delta X, \Delta Y, \Delta Z, \Delta \theta, \Delta \Phi]
\]

(2.5)

\[
[\Delta X, \Delta Y, \Delta \theta]
\]

(2.6)

To translate the points in 2D are shown in (2.7) and (2.8), where \(x_t\) and \(y_t\) are the translational values, and \(\Theta\) is the rotation angle.

\[
x_i^1 = x_i \cos \Theta + y_i \sin \Theta + x_t
\]

(2.7)
\[ y_i^1 = -x_i \sin \Theta + y_i \cos \Theta + y_t \]  

(2.8)

Versions of the algorithms eliminate the angular translation. While the main option would be to just set the angular variable to zero, this is done to streamline the number of clock cycles the kernel must execute for the operation, if it is known that the angular translation will not be used.

To translate the points in 3D are shown in the rotation matrices (2.9), (2.10) and (2.11), where \( x_t, y_t, \) and \( z_t \) are the translational values, and \( \Theta \) is the rotation angle. These equations are combined into one equation, (2.12).

\[
R_x(\Theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \Theta & -\sin \Theta \\ 0 & \sin \Theta & \cos \Theta \end{bmatrix} 
\]

(2.9)

\[
R_y(\Theta) = \begin{bmatrix} \cos \Theta & 0 & \sin \Theta \\ 0 & 1 & 0 \\ -\sin \Theta & 0 & \cos \Theta \end{bmatrix} 
\]

(2.10)

\[
R_z(\Theta) = \begin{bmatrix} \cos \Theta & -\sin \Theta & 0 \\ \sin \Theta & \cos \Theta & 0 \\ 0 & 0 & 1 \end{bmatrix} 
\]

(2.11)

\[
R_z(\Theta) \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = R_x * R_y * R_z * \begin{bmatrix} x^o \\ y^o \\ z^o \end{bmatrix} + \begin{bmatrix} x_t \\ y_t \\ z_t \end{bmatrix} 
\]

(2.12)

### 2.2.4 Scale Data

Data scaling refers is essentially translating the value read from the sensor to another range of values necessary for analysis by other functions. An example of this is if a sensor returns a depth as a value from 0 to 1023, and a value of 0 actually refers to 50cm, and a value of 1023 refers to 300cm. While it is possible to set up the algorithms to operate on the original data, the simplest way to interact with environmental data and vehicular dimensions and movement could be to interact with the data in real world dimensions. Each data point can
be interacted on independently, and can be parallelized to potentially decrease computation time.

2.2.5 Gaussian Blur

Gaussian Filter 1D and 2D blurring filters can be used to reduce the noise that sensor data might have. This is commonly applied before applying an 1D or 2D derivative filter, when used for edge detection. By convolution the sensor data with either (2.13) for 1D data array or (2.14) equations for 2D data array. These equations are evaluated for discrete data sets, that will be stored as a constant arrays to minimize the clock cycles need for calculations.

These discrete sets are calculated for different windows sizes.

\[ G(x) = \frac{1}{\sqrt{2\pi}\sigma^2} e^{-\frac{x^2}{2\sigma^2}} \]  
\[ G(x) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \]

The evaluated discrete values used in the algorithm are shown in table 2.1. These values were calculated with Matlab code, show in Section A.1. The user can chose a window size, and a \( \sigma \) value that matches the application. The mask is then loaded along with the input data and the result is based on the equation (2.15), where \( i \) is the center data point, and \( n \) is the window size. \( G \) is selected by the \( \sigma \) chosen. Because of the symmetry of the Gaussian curve, values of the centre point and one side of the curve only have to be loaded to reduce the size of the constant. By reducing this size the, the number of memory operations to load each constant into the local memory will be reduced.

\[ Y_i = G_0 \ast X_i + \sum_{k=1}^{n} G_k \ast (X_{i-k} + X_{i+k}) \]

(2.15)
### Table 2.1: 1D Gaussian Blur Constants

<table>
<thead>
<tr>
<th>Window</th>
<th>$\sigma$</th>
<th>1</th>
<th>1.5</th>
<th>2</th>
<th>2.5</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G_0$</td>
<td></td>
<td>0.4744241752</td>
<td>0.3873666012</td>
<td>0.3354698081</td>
<td>0.3000597020</td>
<td>0.2739418777</td>
</tr>
<tr>
<td>$G_1$</td>
<td></td>
<td>0.2339237072</td>
<td>0.2417650767</td>
<td>0.2355630418</td>
<td>0.2261364887</td>
<td>0.2164184276</td>
</tr>
<tr>
<td>$G_2$</td>
<td></td>
<td>0.0280411950</td>
<td>0.0587771359</td>
<td>0.0815583242</td>
<td>0.0967964782</td>
<td>0.1067091509</td>
</tr>
<tr>
<td>$G_3$</td>
<td></td>
<td>0.0008172099</td>
<td>0.0055662987</td>
<td>0.0139231294</td>
<td>0.0235328437</td>
<td>0.0328383032</td>
</tr>
<tr>
<td>$G_4$</td>
<td></td>
<td>0.0000057900</td>
<td>0.0002053372</td>
<td>0.0011719602</td>
<td>0.0032494900</td>
<td>0.0063071240</td>
</tr>
<tr>
<td>$G_5$</td>
<td></td>
<td>0.0000000099</td>
<td>0.0000029506</td>
<td>0.0000486403</td>
<td>0.0002548484</td>
<td>0.0007560555</td>
</tr>
</tbody>
</table>

### 2.2.6 Divide and Conquer with Commutative Summation

Divide and conquer summation is designed to divide an array of integers or float values over a number of processing cores, and decrease the computation time for computing a summation of the values. The algorithm divides the array between all processing cores in cue. Each kernel computes a summation of the locally stored values. Once all streams have computed the local summations, these local summations are transfered to a global variable. The following algorithm is then used to compute to total summation.

\[
localSum_i
\]

\[n = \text{number of kernels}\]

\[\text{while } n \neq 1 \text{ do}\]

\[\text{if } \text{KernelID} \leq \text{NumberofKernels} \text{ then}\]

\[localSum_i \leftarrow localSum_i + kernelSum[i + n/2]\]

\[globalSum_i \leftarrow localSum_i\]

\[n \leftarrow n/2\]

\[\text{else}\]

\[\text{Break;}\]

\[\text{end if}\]

\[\text{end while}\]

The differences between the 1D, 2D and 3D functions is the vector size used. There are slight changes in the addition of points, where the difference being the local parallelization of the summations. OpenCL 1.2 compilers evaluate the the vector structures used in the kernel to determine the optimal execution on the hardware. That means that the pipelining
of the vector functions will have different execution performance of 1D data type casted to 3D vector structures and containers [11].

### 2.2.7 Sort Points

The sort function can sort the data according to four axis: X, Y, Z, θ. The sort function uses the radix sort method, which has been show to be effective in parallel execution [11], [12]. This method divides the data set into smaller groups, and then sort based on least significant digit and then by most significant digit. By performing this comparison of small groups, iteratively, the data set is efficiently sorted. While the Voronoi optimal path planning algorithm (a very commonly used algorithm with LiDAR data) hasn’t effectively been implemented in parallel execution [1], being able to sort the data points in parallel is one method of decreasing the performance time of the system. The bitonic sort method was also considered, but previous research has shown that the performance does not match the radix sort method [12]. The functions developed include different functions for 1D, 2D and 3D data sets in both float and integer arrays.

### 2.2.8 Sobel

The Sobel Operator is in essence, an edge detection method which emphasizes any edges and sharp transitions. It’s application is typically in image processing, but by applying its process to LiDAR data received from devices such as the Velodyne LiDAR, this method can detect the edges and corners of objects in the LiDAR space. This can be then used to isolate or classify vehicles or other objects. One of the variables being passed to the kernel is the direction variable. This determines if the data will be convoluted using (2.16) or (2.17), where A is the input data.

\[
G_x = \begin{bmatrix}
-1 & 0 & +1 \\
-2 & 0 & +2 \\
-1 & 0 & +1 \\
\end{bmatrix} * A
\]  

(2.16)
Chapter 2. Background Information

\[ G_y = \begin{bmatrix} -1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A \]  

(2.17)

2.3 Environmental Analysis

Once these OpenCL accelerated algorithms have been implemented, two environmental analysis algorithms will be tested with these functions. The goal is to offload the task from the CPU to the GPU for the analysis and computations on the data points. These two algorithms that were implemented with these functions were as a magnetic field analysis algorithm, and a tentacle path analysis algorithm. The magnetic field algorithm is generally used by omni-drive vehicles, and the tentacle path analysis is designed for holonic drive vehicles.

2.3.1 Magnetic Field Algorithm

The magnetic field algorithm simply treats all obstacle points as magnetic field point sources. The vehicle and obstacles have equal magnetic field strengths propagating from a point source. These magnetic obstacles apply a force to the vehicle, the sum of the forces is the direction that the obstacles are pushing the vehicle to move towards. The benefit of this algorithm is the minimal calculations required (compared to the Tentacle and Voronoi algorithms). Equation (2.18) calculates the sum of the forces applied to the vehicle along the X-axis. Equation (2.19) calculates the sum of the forces applied to the vehicle along the Y-axis. Equation (2.20) calculates the magnitude of the overall force applied to the vehicle. Equation (2.21) calculates the angle of the magnetic field applied to the vehicle. \( i \) is the index of the array being processed. \( F_x \) and \( F_y \) are the vector components of the magnetic force. \( r_i \) and \( \Theta_i \) are the radial distance and angle of the point being processed, respectively.

\[ F_x = \sum_{i=0}^{n} \frac{1}{r_i} \sin \Theta_i \]  

(2.18)
\[ F_y = \sum_{i=0}^{n} \frac{1}{r_i} \cos \Theta_i \]  
\tag{2.19}

\[ |F| = \sqrt{F_x^2 + F_y^2} \]  
\tag{2.20}

\[ F_\theta = \tan \frac{F_x}{F_y} \]  
\tag{2.21}

From these equations, it can be seen that obstacles that are further away from the vehicle have a much smaller effect on the driving direction as compared with objects that are closer to the vehicle. While this algorithm is relatively quick to execute, it is not without its flaws. This algorithm is not robust in terms of detection a blocked path. A simple example is shown in figure 2.1.1 and figure 2.1.2

Figure 2-1 and Figure 2-2 would result in a force in the x-direction equaling 0, since the obstacles are symmetrical along the Y-axis. Looking at the data points, it can be clearly seen that this will pose a serious problem in the robustness of the algorithm. It can be seen that Figure 2-2 is a blocked path, but the Magnetic Field algorithm cannot determine if a path is clear for driving. It is only capable of determining a driving direction, in this case the theta value would be \( n/2 \) radians. One possible solution is to apply the Tentacle algorithm to inspect the chosen direction to determine if the path is clear (or alternatively the maximum drivable distance in the calculated direction. In order to take advantage of the mass parallelization of work items, the code is designed create a thread for each data point. The thread calculates the magnetic force. The thread then computes a divide-and-conquer addition that reduces the total summation cycles from \( n \) (where \( n \) is the number of data points) to \( \log_2 n \) [9]. This step also reduces the memory delay when transferring an array of vectors, versus only sending a single vector containing the summation value. The reasoning behind choosing this test is the simplicity of the algorithm. This algorithm can be easily implemented on CPU architecture to run in real time, but can be shown to test the point between the bottleneck of transferring the data to the GPU, and the bottleneck of the computations.

The algorithm used for the divide-and-conquer method works by using each work term to add two force vectors. Since there is a work item created for each data point, half of the work items begin to add two data points each. After this step has completed, one quarter of
the work items sum two of the summations created in the first step. This process continues until the number of summation points is 1. Once this is complete, a single vector is sent back to the host.

### 2.3.2 Tentacle Path Analysis

The Tentacle Path Detection algorithm is based on Ackermann steering model[13], and can easily be adapted to work with parallel and rear wheel steering vehicles, as well as vehicles with trailers[2]. The portable robotic platform produced the challenge of balancing the processing power needed for high performance algorithms with the weight of batteries needed to power such a platform. The second issue was the need to produce a system that could maintain real-time performance.[4, 5, 14]

The tentacle algorithm compares depth data to predetermined drivable paths. These drivable paths are determined by the control system for steering. Initial design of the data analysis process was inspired by the Tentacle algorithm used by Hundelshausen et al[1] which correlated colored lanes to polynomial curves. Since the lanes that were being detected involved unknown vehicles as well as physical objects, this method could not be used. The method created is more brute force than correlative, but the result is a robust algorithm. The tentacle algorithm calculates the occupancy of traversable paths. These traversable paths are defined by rectangular spaces (analysis of straight driving), arc spaces (used for steering paths) as well as skewed rectangles (veers used to reduce calculations of circle paths)[15–17].

The first tentacle path is defined by boundaries of vertical equations defined by the width of the vehicle plus a buffer width. This path is really a special case of the tentacles based on circular motion paths. The straight path is simplified from (2.22) to vertical linear boundaries. This is shown in equation (2.23), where \( w_{\text{vehicle}} \) is the width of the vehicle, and \( w_{\text{buffer}} \) is the width of the buffer spacing. If a depth value, with coordinates \((x_i, y_i)\), fails this test, the maximum traversable distance is recorded for future comparison with the control algorithm.

\[
r_i = \lim_{R \to +\infty}
\]

(2.22)
\[ |x_i| > w_{vehicle} + w_{buffer} \] (2.23)

If a data point is within the rectangular region, the maximum traversable distance, \( z_{\text{min}} \), must be recorded, shown in equation (2.24).

\[ z_{\text{min}} = \min(z_{\text{min}}, y_i) \] (2.24)

The main tentacle paths are defined by arc regions. The arc regions have a center radius defined by the potential radii of steering. The definition of the center curve path is defined by Equation (2.33).

\[ R = \sqrt{a_2^2 + l^2 \cot^2 \delta} \] (2.25)
\[ \cot \delta = \frac{\cot \delta_o + \cot \delta_i}{2} \] (2.26)
\[ R_{\text{min}} = R - \frac{w}{2} \] (2.27)
\[ R_{\text{max}} = R + \sqrt{\left(R + \frac{w}{2}\right)^2 + (l + g)^2} \] (2.28)

Equation (2.25) defines the steering radius of the vehicle with reference to the center of mass. \( a_2 \) is the length from the rear axle to the center of mass. \( l \) is the distance between...
the front and rear axle. \(w\) is the width between the centers of the wheels. \(\delta\) is the angle from the horizon to the line that goes from the center of the turning radius to the center of the front axle. This distance of the center of mass from the rear axle was experimentally determined to by setting the steering servo to the maximum steering angle, and driven on a carpet surface (tile and concrete surfaces were not enough friction to accurately drive in a circle without slip). The equation was rearranged for \(a_2\), shown in equation (2.29).

\[
a_2 = \sqrt{R^2 - l^2 \cot^2 \delta} \tag{2.29}
\]

The value was averaged from 25 trials, and determined to be 20.5 cm.

If a data point is within the arc region, the maximum traversable distance must be calculated (2.30) in reference to the center of the vehicles path and then record the minimum distance (2.32). The equation (2.31) is based on the law of cosines, calculating the magnitude of the angle from the triangles side lengths.

\[
L = \sqrt{(|x_i| - R)^2 + y_i^2} \quad \text{(2.30)}
\]

\[
\phi = \cos^{-1}\left(\frac{R^2 + L^2 - \sqrt{x_i^2 + y_i^2}}{2RL}\right) \quad \text{(2.31)}
\]

\[
z_{\text{min}} = \min(z_{\text{min}}, L\phi) \quad \text{(2.32)}
\]

\textbf{Figure 2.4:} A point inside region, find minimum distance on arc path
The equations (2.34) and (2.35) are the definitions for the outer and inner boundaries of the arcs, respectively. The +- refers to the steering direction (- referring to steering to the right, + for steering to the left)

\[ y = \sqrt{\text{radii}_j^2 - x^2} \]  \hspace{1cm} (2.33)

\[ y_i > \sqrt{(R_{\text{max}} + w_{\text{buffer}})^2 \pm x_i^2} \]  \hspace{1cm} (2.34)

\[ y_i < \sqrt{(R_{\text{min}} - w_{\text{buffer}})^2 \pm x_i^2} \]  \hspace{1cm} (2.35)

The veers were created as a way for simplifying the number of CPU intensive math functions. The veers were designed to eliminate the need for large radii curves. The equations (2.36) and (2.37) define the upper and lower bounds of the veer paths. \( \theta \) is the angle of the veer from the center of the vehicle. Substituting veer paths for large radii curves has been beneficial for low performance hardware, or software platforms where the amount of processing time for path detection needs to be reduced.

\[ y_i > x_i \cot(\theta) + \frac{w/2 + w_{\text{buffer}}}{\cos(\theta)} \]  \hspace{1cm} (2.36)

\[ y_i < x_i \cot(\theta) - \frac{w/2 + w_{\text{buffer}}}{\cos(\theta)} \]  \hspace{1cm} (2.37)
Chapter 3

Experimental Setup

3.1 Research Methodology

3.1.1 Software

Analyzing the kernel performance is critical to this research. The tool chosen to perform this for the AMD CPU and the ATI Radeon HD 7970 is AMD’s CodeXL tool. This tool can run as a stand alone application or as an add on to Microsoft Visual Studio. In this research, it is used as an add on to Microsoft Visual Studio 2013. CodeXL is an OpenCL kernel debugger as well as a CPU and GPU profiler tool for running analysis on code. CodeXL is developed by AMD, and is designed to debug OpenCL and OpenGL kernels. The debugger is designed to find any bugs in the kernel code, as well as optimizing performance. The debugger is designed to monitor the memory resources, core performance, memory leaks, and areas causing less than optimal kernel execution. The CPU Profiler is designed to analyze the code execution on CPU’s and monitor the resources used during execution. The GPU Profiler collects and visualizes GPU data, kernel occupancy, and hotspot analysis for AMD APUs and GPUs. The Static Kernel Analysis tool allows for kernel code to be compiled for various AMD hardware and simulate the execution time and performance without running the application. This allows for theoretical benchmarking on hardware not in the system. For the Intel CPUs, the Intel OpenCL SDK Debugger was chosen. This debugger does
not provide as many features as AMD’s CodeXL, Intel’s debugger has been designed and optimized for Intel CPUs. For the NVIDIA GPU, the NVIDIA Nsight debugger add on for Microsoft Visual studio has been chosen as it is designed and optimized to analyze the performance of NVIDIA GPUs in the execution of OpenCL kernels.

The implemented software is grouped into two main test groups: the library functions, implementation using library functions. The library of OpenCL accelerated functions are kernels written that can be executed individually on an OpenCL compatible device, or be used as a reference function from another kernel. The OpenCL enabled library of LiDAR based functions will be individually benchmarked and tested against the C++ implemented code. These functions were discussed in Chapter 2. Since AMD’s CodeXL, Intel’s OpenCL SDK Debugger, and NVIDIA’s Nsight can all profile C++ code, it will be tested with all three profilers. That is, when testing the ATI Radeon HD 7970, CodeXL will profile the C++ implementation as well as the kernel implementation. This will minimize the discrepancies in analysis that could come from using three different profilers.

These algorithms were chosen based on the difficulty of concurrency to implement. The virtual force field obstacle avoidance algorithm is a simple path direction planning, and OpenCL implementation is through a simple loop unrolling plus tree-based summation. Tentacle path analysis is a little more complicated as it is more geometrical calculations, plus logical comparisons. The fortunes sweep line Voronoi path detection is the most complex, but also the most robust of the algorithms. This algorithm is the least explicitly parallelization of the three. These algorithms are first implemented in C++, to determine a baseline for performance. Once this base code has been implemented and experimentally tested, these algorithms are rewritten to be implemented with the OpenCL framework. These algorithms will be then tested again, benchmarked, and compared with the baseline results.
3.1.2 Hardware

The computer hardware selection was geared to three hardware setups: high wattage system, mobile hardware, and a low power system. The computing hardware for the mobile platform consists of an Intel Atom 1.8GHz Mini-ITX motherboard setup, with 4 GB of DDR3 1333MHz RAM. This first test platform requires that the hardware used for implementation must be lightweight, low power. Specifications of this system in Figure 3.1. This setup is a combination of a ultra-low power processor with a low power graphics card. This setup is targeted towards a mobile platform that needs low power and small footprint. This platform is built on the Intel DN2800MT Marshaltown Low Profile Mini ITX Motherboard, shown in ???. This has a vertical profile of 20mm. The other benefit is the flexibility of power input, with a power range of 10 to 19V DC. This allows for a lithium polymer battery to be connected directly to the motherboard without a DC to DC power converter. The board also features a smart power feature that can automatically shut off if the voltage drops under 11V, protecting batteries from dropping to damagingly low voltages. The MSI GeForce 210 GPU, shown in Figure 3.2, is connected through a PCI Express 1x to 16x ribbon.

These requirements also match the ideal characteristics of a platform that would be necessary for a full size vehicle implementation. The desktop hardware setup is a six core 3.6GHz AMD bulldozer CPU, with 16GB of RAM, and has a Radeon 7970 GPU. The GPU has 2048 streaming processes and 3GB of memory. The hardware properties are summarized in
Chapter 3. *Experimental Setup*

**Table 3.1:** Platform 3

<table>
<thead>
<tr>
<th>CPU</th>
<th>1.8GHz Intel Atom</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM</td>
<td>(2) 2GB 1333MHz DDR3</td>
</tr>
<tr>
<td>GPU</td>
<td>MSI GeForce 210</td>
</tr>
</tbody>
</table>

**Table 3.2:** Platform 2

<table>
<thead>
<tr>
<th>CPU</th>
<th>2.6GHz Intel Core i5</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM</td>
<td>(2) 4GB 1866MHz DDR3</td>
</tr>
<tr>
<td>GPU</td>
<td>NVIDIA 740M</td>
</tr>
</tbody>
</table>

**Table 3.3:** Platform 1

<table>
<thead>
<tr>
<th>CPU</th>
<th>3.6GHz AMD 6-Core</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM</td>
<td>(4) 4GB 1866MHz DDR3</td>
</tr>
<tr>
<td>GPU</td>
<td>ATI Radeon HD 7970</td>
</tr>
</tbody>
</table>

The sensor used is a Hokuyo URG-04LX-UG01 LiDAR sensor, shown in Figure 3.3. The sensor returns a maximum data set of 683 points, with an angular resolution of $0.3515625^\circ$. The device communicates over USB 2.0. The scanning time is 100 milliseconds per scan, or
in other words a maximum scanning frequency of 10Hz. The detectable range of the sensor is shown in Figure 2.

3.1.3 Power Analysis - WattsUp Pro

One of the parameters that is used as a performance metric is the power wattage of the computer during idle, during CPU load, and during GPU load. This lead to researching various load meters. There are many off-the-shelf load meters available. P3 Kill-A-Watt, Belkin Conserve Insight are popular consumer targeted load meters. Testing began with the P3 Kill-A-Watt. One issue was apparent, while the product boasts an accuracy of 0.2% for wattage, the issue was accurately measuring the wattage. The system does not record data, it only displays data. This limits the accuracy of recording to viewing the device. Any quick power spikes might be missed, and the data cannot be reviewed later.

To analyze the power usage of the code execution, the WattsUp Pro power load meter was selected. The WattsUp Pro is capable of measuring between 1000 to 32000 records
depending on the number of parameters analyzed with each record. Parameters available are Current Watts, Minimum Watts, Maximum Watts, Power Factor, Volt Amp (apparent PWR), Cumulative Watt Hours, Average Monthly KiloWatt Hour, Elapsed Time, Duty Cycle, Frequency (Hz), Cumulative Cost, Average Monthly $, Line Voltage, Minimum Volts, Maximum Volts, Current Amps, Minimum Amps, Maximum Amps. The WattsUp Pro is capable of 1Hz power analysis, and has an accuracy of 1.5% for all parameters over 60watts. Under 60 watts, the accuracy of current and power factors is significantly less at approx 10% accuracy. It has a maximum rated power of 1800watts. [insert reference]

The data can either be read in real time with the WattsUp Pro Realtime software, or downloaded from the internal non-volatile memory on board the WattsUp Pro over USB. Some sample data is shown in 3.1.3.
To get an accurate power reading for the analysis, the computer hardware will have to be under test load for more than a second. This will reduce the probability that the sampling of the load meter will not accurately take a reading of the test load power, since the test conditions state that the performance metric is to run LiDAR sampling at a rate of 60Hz or faster. In order to accurately test the average power consumed, the software analyzes sample data for 300 readings, which equates to 5 minutes of testing.

An important metric to analyze the power consumed during a non-active state. Background applications have been kept to a minimum and only services needed by the operating system, Windows 8, have been left running. Power will then be measured, and can be compared to the power consumed during data analysis.

### 3.1.4 Virtual Hardware

Experimental data from the Velodyne HDL 64E S2 was provided by the research group LAAS-CNRS (Laboratory for Analysis and Architecture of Systems) at the University of Toulouse. The data was recorded during testing of an autonomous research vehicle in different environments, driving conditions, and driving situations. This data allowed for real world data results to be tested with the OpenCL accelerated functions for 3D Lidar data
Figure 3.5: Velodyne HDL 64E S2

manipulation. To work with the data, the raw communication data has to be read and organized in such a way that it can be read and analyzed in real time as if the data was being provided by a live sensor. The recorded data is a combination of data from the Velodyne HDL 64E S2 and a GPS unit. The GPS unit provides an X, Y, Z coordinate, as well as the pitch, roll and yaw of the unit. The Lidar Network Protocol is outlined in Table 3.4. Table 3.5 breaks the data packet down further into each piece of information received per data packet. In the case of the data samples provided by the 2RT-3D project, the number of points sent with each data packet was 7200.
Table 3.4: Velodyne HDL 64E S2 - LiDAR Network Protocol

<table>
<thead>
<tr>
<th>Section</th>
<th>Subsection</th>
<th>Data Structure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>ID</td>
<td>8 byte String</td>
<td>&quot;RDR file&quot;</td>
</tr>
<tr>
<td></td>
<td>Options</td>
<td>1 Byte</td>
<td>bit0 : 1 if the file contains a lookup table, 0 otherwise bit1-7 : reserved</td>
</tr>
<tr>
<td></td>
<td>LT Address</td>
<td>uint64</td>
<td>Position of the first byte of the LOOKUP TABLE block in the file if present.</td>
</tr>
<tr>
<td>Data Block</td>
<td>Timestamp</td>
<td>uint64</td>
<td>Milliseconds since the start of the record</td>
</tr>
<tr>
<td></td>
<td>Packet</td>
<td>27+7*n Bytes</td>
<td>A Lidar network protocol packet</td>
</tr>
<tr>
<td>Lookup Table</td>
<td>Total size</td>
<td>uint64</td>
<td>Total number of packets in the file</td>
</tr>
<tr>
<td></td>
<td>Table Size</td>
<td>uint32</td>
<td>Number of references listed in table</td>
</tr>
<tr>
<td>Reference Block</td>
<td>Table Size</td>
<td>Table Size * 16 bytes</td>
<td>Timestamp and Byte Offset</td>
</tr>
</tbody>
</table>
Table 3.5: Velodyne HDL 64E S2 - Data Packet

<table>
<thead>
<tr>
<th>Section</th>
<th>Subsection</th>
<th>Data Structure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>Type</td>
<td>uint8</td>
<td>Identify the source of the packet [0x01 : real time packet (coming from a live acquisition), 0x02 : simulation packet, 0x03 : replay packet]</td>
</tr>
<tr>
<td></td>
<td>N</td>
<td>uint16</td>
<td>Indicates the number of impacts described in the IMPACT DATA block. Keep in mind that small packets with a high transmission rate will easily clutter the network. Consider sending many impacts in the same pack</td>
</tr>
<tr>
<td>Sensor Pose</td>
<td>X position</td>
<td>float</td>
<td>West-East axis, east being positive, in meter</td>
</tr>
<tr>
<td></td>
<td>Y position</td>
<td>float</td>
<td>South-North axis, north being positive, in meter</td>
</tr>
<tr>
<td></td>
<td>Z position</td>
<td>float</td>
<td>Vertical axis, up being positive, in meter</td>
</tr>
<tr>
<td></td>
<td>Yaw Angle</td>
<td>float</td>
<td>In degrees. 0 points to east. 90 points to north</td>
</tr>
<tr>
<td></td>
<td>Pitch Angle</td>
<td>float</td>
<td>In degrees. 0 points to the horizon. 90 points down</td>
</tr>
<tr>
<td></td>
<td>Roll Angle</td>
<td>float</td>
<td>In degrees. 0 is horizontal. 90 means the left side of the vehicle is pointing up</td>
</tr>
<tr>
<td>Impact Data</td>
<td>Yaw</td>
<td>uint16</td>
<td>Between 0 and 35999, the unit is the 100th of degrees. 0 is the forward direction. Positive angles go to the left (counter-clockwise when seen from the top of the vehicle)</td>
</tr>
<tr>
<td></td>
<td>Pitch</td>
<td>uint16</td>
<td>Between 0 and 35999, the unit is the 100th of degree. 0 is the horizontal plane in the vehicle’s reference frame. Positive angles go down (counter-clockwise when seen from the left of the vehicle)</td>
</tr>
<tr>
<td></td>
<td>Distance</td>
<td>uint16</td>
<td>Distance in 0.2 centimeters increments</td>
</tr>
<tr>
<td></td>
<td>Intensity</td>
<td>uint8</td>
<td>Intensity of the reflected laser. 255 is the most intense return</td>
</tr>
</tbody>
</table>
Chapter 4

Experiment Setup

4.1 OpenCL Accelerated Depth Data Analysis Library

The first step in the analysis of the performance of OpenCL in processing sample LiDAR data was to benchmark the functions written to process the data in terms of:

- Processing time on OpenCL device
- Processing time on CPU without acceleration
- Memory used
- Power Consumed

The algorithms are tested with randomly generated data of various data lengths, as well as sample data from the Hokuyo and the Velodyne HDL 64E S2. The purpose of testing the randomly generated depth data of various lengths is to determine the point at which the efficiency of the processor is greater than using the OpenCL device. For this purpose, data sets of length beginning at 512 to 4294967296 in steps following the equation (4.2), where \( L \) is the length of the data set, and \( i \) is the step index.

\[
L_i = 2^{i+9}
\]  

\[(4.1)\]
Testing the kernels with actual data is a necessary step for thorough testing. Sample data from the Hokuyo URG-04LX-UG01 LiDAR will be sampled, and recorded to a data file. The data file will then be used to test each of the kernel functions. Testing will also be done with recorded data provided by Robosoft Inc, France. This data will be from the Velodyne HDL 64E S2 recordings, which were taken from a moving vehicle in an urban environment.

Processing time is analyzed as the time the data is received by the CPU. This is an important benchmark location. For the OpenCL device, at this point all that needs to be done is to initiate the memory transfer from the RAM to the OpenCL device RAM, and then initiate the execution of the kernel. Receiving the data from the depth data device that is used is the same whether a CPU used with the C++ code, or an OpenCL device is used. What needs to be measured though, is the impact that transferring the data to device and the time it takes to execute have in comparison to regular code execution on the CPU. Investigation of the bottleneck in the process is crucial for optimization and measuring the execution time, as well as detailed analysis of how that time is spent is critical to understanding what is happening. AMD CodeXL provides an in depth analysis of the kernel execution and memory transfers, which provides the feedback necessary for this analysis. The end point for the time analysis is when the data has completed transfer off the OpenCL device, and the data is ready for use by the CPU. For consistency of testing the CPU devices, the dynamic clocks on the devices have been disabled. While this does restrict the devices from peak performance, this reduces the discrepancy in time analysis between different tests. These discrepancies arise from how the devices select the operating frequency. The parameters for control are commonly the temperature of the device cores, and power consumption. Something as simple as the environment temperature fluctuating could cause the frequency to be more restricted in one test compared to another.

Memory is an important metric to analyze. With small data sets, the devices’ RAM and on board memory might not be important, but with large data sets the amount of available RAM and memory could become a performance limitation. In devices like GPU’s with multiple different layers of usable memory, analysis of which layer and the bandwidth available can be a limitation. Characteristics analysed include memory transfer bandwidth, the maximum amount of memory used, and the global and local memory breakdown on GPU’s. AMD’s CodeXL will provide the interface and the ability to measure these characteristics.
Power consumed is an important metric to compare hardware efficiency. As an example, if a performance increases by 40% but the power consumed increases by 80%, the performance gain might not be favorable for the application. To minimize power fluctuations from devices not directly necessary for the experiment, have either been controlled or removed. Heat sink fans of the devices have been controlled to a set fan speed. As well, secondary drives in the system have also been removed to minimize power consumption not related to the experiments.

4.2 Environment Analysis Application of OpenCL library

The first step in the analysis of the performance of OpenCL in processing sample LiDAR data was to benchmark the functions written to process the data in terms of:

- Processing time on OpenCL device
- Processing time on CPU without acceleration
- Memory used
- Power Consumed

These benchmarks were isolated based on the research of [9] and [11] to be the critical factors when optimizing the execution of the kernel code.

The algorithms are tested with randomly generated data of various data lengths, as well as sample data from the Hokuyo and the Vel. The purpose of testing the randomly generated depth data of various lengths is to determine the point at which the efficiency of the processor is greater than using the OpenCL device. For this purpose, data sets of length beginning at 512 to 4294967296 in steps following the equation (4.2), where \( L \) is the length of the data set, and \( i \) is the step index.

\[
L_i = 2^{i+9}
\]  

(4.2)
Testing the kernels with actual data is a necessary step for thorough testing. Sample data from the Hokuyo URG-04LX-UG01 LiDAR will be sampled, and recorded to a data file. The data file will then be used to test each of the kernel functions. Testing will also be done with recorded data provided by Robosoft Inc, France. This data will be from the Velodyne HDL 64E S2 recordings, which were taken from a moving vehicle in an urban environment.

Processing time is analyzed as the time the data is received by the CPU. This is an important benchmark location. For the OpenCL device, at this point all that needs to be done is to initiate the memory transfer from the RAM to the OpenCL device RAM, and then initiate the execution of the kernel. Receiving the data from the depth data device that is used is the same whether a CPU used with the C++ code, or an OpenCL device is used. What needs to be measured though, is the impact that transferring the data to device and the time it takes to execute have in comparison to regular code execution on the CPU. Investigation of the bottleneck in the process is crucial for optimization and measuring the execution time, as well as detailed analysis of how that time is spent is critical to understanding what is happening. AMD CodeXL provides an in-depth analysis of the kernel execution and memory transfers, which provides the feedback necessary for this analysis. The end point for the time analysis is when the data has completed transfer off the OpenCL device, and the data is ready for use by the CPU. For consistency of testing the CPU devices, the dynamic clocks on the devices have been disabled. While this does restrict the devices from peak performance, this reduces the discrepency in time analysis between different tests. These discrepancies arise from how the devices select the operating frequency. The parameters for control are commonly the temperature of the device cores, and power consumption. Something as simple as the environment temperature fluctuating could cause the frequency to be more restricted in one test compared to another.

Memory is an important metric to analyze. With small data sets, the devices’ RAM and onboard memory might not be important, but with large data sets the amount of available RAM and memory could become a performance limitation. In devices like GPU’s with multiple different layers of usable memory, analysis of which layer and the bandwidth available can be a limitation. Characteristics analyzed include memory transfer bandwidth, the maximum amount of memory used, and the global and local memory breakdown on GPU’s. AMD’s CodeXL will provide the interface and the ability to measure these characteristics.
The two environmental analysis algorithms will process the same data, both simulated and sample data. The tentacle path detection algorithm will have 2048 driveable paths, and will have vehicle dynamics matching the test vehicle, which was a 2009 Ford Escape. The specifications for the vehicle physics were determined from test driving the same model, and recording the specifics, which are shown in Table:
Chapter 5

Discussion

5.1 Overview

Experiment 1 involves the testing of the execution time of the implementations of the various algorithms on the GPU vs CPU. Using CodeXL, the memory operations and execution time can be analyzed and compared. Using the WattsUp AC Power Analyzer, the power consumed between the CPU and GPU versions can also be analyzed. The cross over point will also be determined. This is the point where the data set size becomes large enough to become more efficient to use the GPU instead of the CPU for the operations.

5.2 Results - Experiment 1

The functions created for this OpenCL accelerat ed library were tested based on the criteria outlined in Chapter 4. The process for code writing and testing was iterated to produce kernel code that was optimized and tested on various hardware platforms. These optimizations include reducing memory operations, memory size needed, as well as work group size adjustments. Initial results of kernel performance with little adjustment did show real time performance in most cases (broken down further for each function type), greater performance was gained by changing kernel code to better perform on the OpenCL hardware.
The hardware platform for final evaluation was the AMD 6 Core CPU with the Radeon 7970 GPU. This platform was selected as the focus hardware, and showed the performance differences between CPU and GPU implementations.

### 5.2.1 LiDAR Depth Data to 3D Coordinates

Final results of CPU and GPU performance of both C++ implementation, BOOST multithreading, and OpenCL implementations are shown in the chart 5.2.1 shown below.

As it can be seen, the GPU significantly outperforms the CPU implementations of the same calculations. This did however come at a cost of power consumed. During the CPU implementation for the C++, OpenCL, and Boost versions had a maximum power wattage of 220 watts on the AMD system. The GPU version averaged 372 watts on full load. This is a significant increase, 69.1%, of consumed power. There was a performance difference between the OpenCL implementation and the Boost multi threaded versions (in the chart they are overlapping). Both of these were on average 4.3 and 4.4 times performance improvement over the C++ version for OpenCL and Boost library variations, respectively. The crossover point for the GPU, this is the point where the GPU outperforms the CPU is 3072 data points. For the large data set, the GPU was 205 times faster.
Analyzing the memory operations, 72.4% of the total evaluation time was spent on memory transfers from the host to the GPU, and from GPU to host. This is a significant portion of the total operation time, and shows the bottleneck that the system encounters in GPU computing.

The different variations of data types showed similar performance results in terms of computation time. The real difference was the memory occupied during operation, which in the cases of the integer variations showed a memory reduction of approximately 24%. This set of functions required very little optimizations from the initial design. The only optimization was changing the number of work items to match the number of possible parallel threads, and dividing the total data set between the work items. The original implementation treated each data point as a work item. This change resulted in a 4% decrease in computation time, which was from a decrease in time spent on queuing.

5.2.2 Data Filtering

Final results of CPU and GPU performance of both C++ implementation, BOOST multithreading, and OpenCL implementations are shown in the chart 5.2.2 shown below.

As it can be seen, the GPU significantly out performs the CPU implementations of the same calculations. This did however come at a cost of power consumed. During the CPU
implementation for the C++, OpenCL, and Boost versions had a maximum power wattage of 220 watts on the AMD system. The GPU version averaged 372 watts on full load. This is a significant increase, 69.1%, of consumed power. There was a performance difference between the OpenCL implementation and the Boost multi threaded versions (in the chart they are overlapping). Both of these were on average 4.5 times performance improvement over the C++ version for OpenCL and Boost library variations. The crossover point for the GPU, this is the point where the GPU outperforms the CPU is 3112 data points. For the large data set, the GPU was 215 times faster.

Analyzing the memory operations, 82% of the total evaluation time was spent on memory transfers from the host to the GPU, and from GPU to host. This is a significant portion of the total operation time, and shows the bottleneck that the system encounters in GPU computing. This bottleneck is increased due to the increased delay in transferring the output to the host. Since the data set might not be smaller than the input due to filter that was removed, the threads now have to push data back asynchronously, and this can cause a performance delay if the memory is being used by another thread.

5.2.3 Global Geometric Translation

Final results of CPU and GPU performance of both C++ implementation, BOOST multi-threading, and OpenCL implementations are shown in the chart 5.2.3 shown below.

![Global Geometric Translation Chart](image-url)
As it can be seen, the GPU significantly out performs the CPU implementations of the same calculations. This did however come at a cost of power consumed. During the CPU implementation for the C++, OpenCL, and Boost versions had a maximum power wattage of 220 watts on the AMD system. The GPU version averaged 372 watts on full load. This is a significant increase, 69.1%, of consumed power. There was a performance difference between the OpenCL implementation and the Boost multi threaded versions (in the chart they are overlapping). Both of these were on average 4.5 times performance improvement over the C++ version for OpenCL and Boost library variations. The crossover point for the GPU, this is the point where the GPU outperforms the CPU is 3112 data points. For the large data set, the GPU was 265 times faster. This is a much greater increase in performance of GPU vs CPU, and this comes from the advantage the GPU has for implementing 3D translation and floating point math, being optimized for 3D calculations.

Analyzing the memory operations, 56.9% of the total evaluation time was spent on memory transfers from the host to the GPU, and from GPU to host. This is a significant portion of the total operation time, and shows the bottleneck that the system encounters in GPU computing.

5.2.4 Scale Data

Final results of CPU and GPU performance of both C++ implementation, BOOST multi-threading, and OpenCL implementations are shown in the chart 5.2.4 shown below.
As it can be seen, the GPU significantly outperforms the CPU implementations of the same calculations. This did however come at a cost of power consumed. During the CPU implementation for the C++, OpenCL, and Boost versions had a maximum power wattage of 220 watts on the AMD system. The GPU version averaged 372 watts on full load. This is a significant increase, 69.1%, of consumed power. There was a performance difference between the OpenCL implementation and the Boost multi threaded versions (in the chart they are overlapping). Both of these were on average 4.5 times performance improvement over the C++ version for OpenCL and Boost library variations. The crossover point for the GPU, this is the point where the GPU outperforms the CPU is 3072 data points. For the large data set, the GPU was 242 times faster. This is a much greater increase in performance of GPU vs CPU, and this comes from the advantage the GPU has for implementing 3D translation and floating point math, being optimized for 3D calculations.

Analyzing the memory operations, 68.7% of the total evaluation time was spent on memory transfers from the host to the GPU, and from GPU to host. This is a significant portion of the total operation time, and shows the bottleneck that the system encounters in GPU computing.
5.2.5 Gaussian Blur

Final results of CPU and GPU performance of both C++ implementation, BOOST multi-threading, and OpenCL implementations are shown in the chart 5.2.5 shown below.

As it can be seen, the GPU significantly out performs the CPU implementations of the same calculations. This did however come at a cost of power consumed. During the CPU implementation for the C++, OpenCL, and Boost versions had a maximum power wattage of 220 watts on the AMD system. The GPU version averaged 372 watts on full load. This is a significant increase, 69.1%, of consumed power. There was a performance difference between the OpenCL implementation and the Boost multi threaded versions (in the chart they are overlapping). Both of these were on average 4.8 times performance improvement over the C++ version for OpenCL and Boost library variations. The crossover point for the GPU, this is the point where the GPU outperforms the CPU is 3072 data points. For the large data set, the GPU was 205 times faster.

Analyzing the memory operations, 82.4% of the total evaluation time was spent on memory transfers from the host to the GPU, and from GPU to host, as well as from global to local. This is a significant portion of the total operation time, and shows the bottleneck that the system encounters in GPU computing. The bottleneck comes from the extra memory operations that occur when loading the data into each local memory required not just he
input data, but depending on the size of the Gaussian filter, up to 8 other data points before and after each work item. This creates a lot of work items that are requesting memory transfers, and creates a memory bottleneck.

5.2.6 Divide and Conquer with Commutative Summation

Final results of CPU and GPU performance of both C++ implementation, BOOST multi-threading, and OpenCL implementations are shown in the chart 5.2.6 shown below.

As it can be seen, the GPU significantly outperforms the CPU implementations of the same calculations. This did however come at a cost of power consumed. During the CPU implementation for the C++, OpenCL, and Boost versions had a maximum power wattage of 220 watts on the AMD system. The GPU version averaged 372 watts on full load. This is a significant increase, 69.1%, of consumed power. There was a performance difference between the OpenCL implementation and the Boost multi-threaded versions (in the chart they are overlapping). Both of these were on average 4.8 times performance improvement over the C++ version for OpenCL and Boost library variations. The crossover point for the GPU, this is the point where the GPU outperforms the CPU is 1024 data points. For the large data set, the GPU was 309 times faster.
Analyzing the memory operations, 76.0% of the total evaluation time was spent on memory transfers from the host to the GPU, and from GPU to host, as well as from global to local. This is a significant portion of the total operation time, and shows the bottleneck that the system encounters in GPU computing.

Some optimizations that had to occur were the placement of waiting points in the kernels. Since the kernel has to be able to work with any size of data set, the number of iterations is unknown. To solve this problem, and to reduce the amount of memory used for this process, one buffer stored the temporary data. To prevent reading data that was not yet written to by another work item, the work item has to pause and wait for all work items in the process to reach the same point. At this point, all the work items have written their output, and now all work items still active can begin reading the values for the next stage. This caused an increase of approximately 27% of the computation time, this prevents any errors from improperly timed memory transfers.

### 5.2.7 Sort Points

Final results of CPU and GPU performance of both C++ implementation, BOOST multithreading, and OpenCL implementations are shown in the chart 5.2.7 shown below.
As it can be seen, the GPU significantly outperforms the CPU implementations of the same calculations. This did however come at a cost of power consumed. During the CPU implementation for the C++, OpenCL, and Boost versions had a maximum power wattage of 220 watts on the AMD system. The GPU version averaged 372 watts on full load. This is a significant increase, 69.1\%, of consumed power. There was a performance difference between the OpenCL implementation and the Boost multi-threaded versions (in the chart they are overlapping). Both of these were on average 4.8 times performance improvement over the C++ version for OpenCL and Boost library variations. The crossover point for the GPU, this is the point where the GPU outperforms the CPU is 1024 data points. For the large data set, the GPU was 245 times faster.

Analyzing the memory operations, 89.9\% of the total evaluation time was spent on memory transfers from the host to the GPU, and from GPU to host, as well as from global to local. This is a significant portion of the total operation time, and shows the bottleneck that the system encounters in GPU computing.

Some optimizations that had to occur were the placement of waiting points in the kernels. Since the kernel has to be able to work with any size of data set, the number of iterations is unknown. To solve this problem, and to reduce the amount of memory used for this process, one buffer stored the temporary data. To prevent reading data that was not yet written to by another work item, the work item has to pause and wait for all work items in the process to reach the same point. At this point, all the work items have written their output, and now all work items still active can begin reading the values for the next stage.

5.2.8 Sobel

Final results of CPU and GPU performance of both C++ implementation, BOOST multi-threading, and OpenCL implementations are shown in the chart 5.2.8 shown below.
As it can be seen, the GPU significantly outperforms the CPU implementations of the same calculations. This did however come at a cost of power consumed. During the CPU implementation for the C++, OpenCL, and Boost versions had a maximum power wattage of 220 watts on the AMD system. The GPU version averaged 372 watts on full load. This is a significant increase, 69.1%, of consumed power. There was a performance difference between the OpenCL implementation and the Boost multi threaded versions (in the chart they are overlapping). Both of these were on average 4.8 times performance improvement over the C++ version for OpenCL and Boost library variations. The crossover point for the GPU, this is the point where the GPU outperforms the CPU is 3072 data points. For the large data set, the GPU was 223 times faster.

Analyzing the memory operations, 64.3% of the total evaluation time was spent on memory transfers from the host to the GPU, and from GPU to host, as well as from global to local. This is a significant portion of the total operation time, and shows the bottleneck that the system encounters in GPU computing.

Some optimizations that had to occur to increase the performance of the kernel. At first each work item was given one point to focus on of the dataset, and read the two data points next to this data point. This created a lot of memory operations that were redundantly read multiple times for different points. By dividing the total data set over the possible parallel
threads at one time, the input data could be loaded locally into the work item in large sets, and reduce the number of global read operations.

5.3 Results - Experiment 2

5.3.1 Magnetic Field

While the actual algorithm is not considered too complex, this simple algorithm can take up many CPU clock cycles for execution. The goal of the OpenCL LiDAR library was to accelerate the processing time for this algorithm for large data sets. The CPU implementation in essence is a vector summation after performing some data filtering. With the OpenCL implementation, the complexity of shared memory operations complicates the process. By using the distributed summation followed by a commutative summation, the execution time is reduced by upwards of 945 times the C++ single core execution (Comparing the Radeon 7970 execution vs the AMD CPU), and up to 182 times faster than the Boost Threaded version of the C++ code. This shows a considerable increase in processing capabilities.

While the GPU execution showed a great decrease in computation time, the OpenCL kernel execution on the CPU’s was less dramatic, and when compared with the Boost Threaded versions, there was a slight decrease in performance.

It should also be noted that the final version of the Magnetic Field kernel process the data in a combination of data and process parallel method. The kernel starts by processing a section of the total amount of data. This section length is calculated from the total data set divided by the number of processes. This method allows for the storage of the section of code to be loaded into the process’ local memory, and concentrate the number of memory operations performed. By copying sections of the data set, the memory controller is able to predict the next memory transfer and begin the process before the actual memory transfer has been requested. In the end, this method was able to decrease the processing time of the large data set by 2%.
5.3.2 Tentacle Algorithm

The tentacle path analysis algorithm had the potential to be executed in either process parallel, or in a data parallel method. The first method experimented was the process parallel method. This method was able to analyze smaller data sets without much consideration of memory size. Unfortunately, on data sets larger than 105,000 points, the amount of local memory in each process was not enough to load the full data set into the local process memory. Breaking the data sets into smaller blocks allowed the process parallel method to be executed. The advantage to analyzing the data in a process parallel method was the lack of a need to commutatively check the availability of each path. This allows the freedom to not need waiting points in the kernel for processes where memory transfers need to be completed before other tasks are done.

The summary of the results of 1024 data pints are shown in Table 5.3, where all results are shown in milliseconds. C++ refers to the C++ code implementation which gets executed on a single CPU core. Boost C++ refers to the Boost threading library for threaded execution of the processes across multiple CPU cores. All data is shown in microseconds. In Table 5.4 show the results for processing $2^{30}$ data points. These values are shown in milliseconds. Both of these summaries are for computing 2048 driveable paths.

By analyzing these results, it is clear to see the improvement that can be achieved with OpenCL when analyzing large data sets. The acceleration brings down the computation time from 11.2 seconds down to 17.78 milliseconds. On the mobile platform, it dropped the computation time from 23.3 seconds down to 108.09 milliseconds. It should be noted here
that this does not meet the requirement set to maintain real time data analysis from the Velodyne sensors. To maintain real time, a smaller window of time would have to be analyzed, or fewer driveable paths analyzed. The amount of GPU memory used for the execution was around 1.4% of global memory and 12.2% of local memory for the Radeon 7970, 5.7% of global memory and 26.3% of local memory for the Geforce 740M, and 23.9% of global memory and 42.0% of local memory for the Geforce 210. These numbers show that there is still room for more data to be processed, and is not the limiting factor for execution.

For CPU implementation, the process had a maximum of 79.9MB for the AMD CPU, and 78.7MB used for the Intel CPU. This again is a small fraction of available RAM, and is not the performance limiting factor. For smaller data sets, the performance of the CPU device performance of the OpenCL kernel was very close to performing as fast as the Boost Threaded C++ version, with Boost having a slight edge. This is completely different from the GPU execution. Analyzing the results of the GPU execution of the kernel vs CPU implementation showed that for 1024 data points, the GPU is much slower. Analyzing with AMD CodeXL for the AMD GPU execution showed that 92.2% of the time was spent waiting for the GPU device and memory transfer. This dramatically impacts the performance of the GPU, and the CPU without any acceleration is much faster than any of the GPU kernel executions.

In a data parallel method, the data set is split up to each process. Each process transfers 1024 points to local memory from the section of the data set that is assigned to that process. Each data point is then compared to driveable paths. After each data point is processed, the minimum traversable distance is stored locally compared with the previously locally
calculated points for each of the paths. Once all the data points in the entire dataset have been evaluated, then the processes ranging from Process 0 to Process N-1, where N is the total number of paths, begin processing the data points in parallel to determine the minimum traversable path. In terms of actual performance of this kernel, the results are shown in Table 5.5 for 1024 data points and Table 5.6 for $2^{30}$ data points.

<table>
<thead>
<tr>
<th>CPU</th>
<th>GPU</th>
<th>C++</th>
<th>Boost C++</th>
<th>OpenCL-CPU</th>
<th>OpenCL-GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMD 6-Core</td>
<td>Radeon 7970</td>
<td>1.33</td>
<td>0.37</td>
<td>0.45</td>
<td>8.07</td>
</tr>
<tr>
<td></td>
<td>Geforce 210</td>
<td>1.33</td>
<td>0.37</td>
<td>0.45</td>
<td>12.53</td>
</tr>
<tr>
<td>Intel Core i5</td>
<td>Geforce 740M</td>
<td>2.46</td>
<td>1.89</td>
<td>2.05</td>
<td>6.62</td>
</tr>
</tbody>
</table>

Table 5.5: Tentacle Algorithm - Data Parallel - 1024 Data Points

<table>
<thead>
<tr>
<th>CPU</th>
<th>GPU</th>
<th>C++</th>
<th>Boost C++</th>
<th>OpenCL-CPU</th>
<th>OpenCL-GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMD 6-Core</td>
<td>Radeon 7970</td>
<td>$1.28 \times 10^4$</td>
<td>$2.34 \times 10^3$</td>
<td>$2.33 \times 10^3$</td>
<td>19.91</td>
</tr>
<tr>
<td></td>
<td>Geforce 210</td>
<td>$1.28 \times 10^4$</td>
<td>$2.34 \times 10^3$</td>
<td>$2.34 \times 10^3$</td>
<td>1.54 \times 10^3</td>
</tr>
<tr>
<td>Intel Core i5</td>
<td>Geforce 740M</td>
<td>$2.59 \times 10^4$</td>
<td>$8.15 \times 10^3$</td>
<td>$8.60 \times 10^3$</td>
<td>120.54</td>
</tr>
</tbody>
</table>

Table 5.6: Tentacle Algorithm - Data Parallel - $2^{30}$ Data Points

As it can be seen from the summary of the results, the data parallel method for the Tentacle Analysis is a decrease in performance from the process parallel execution. Similar to the process parallel method, the GPU kernel execution is dramatically slower than the CPU execution for small data sets. For larger data sets, the Radeon 7970 kernel execution is the only device capable of maintaining real time analysis without a reduction of the number of points or the number of paths analyzed. The amount of GPU memory used for the execution was around 2.2% of global memory and 13.2% of local memory for the Radeon 7970, 8.4% of global memory and 32.8% of local memory for the Geforce 740M, and 45.0% of global memory and 57.2% of local memory for the Geforce 210. These numbers show that there is still room for more data to be processed, the amount of memory used for this execution is increased from the data parallel kernel. This increase is due to the extra memory required for the reduction process of the kernel which uses a 2048x1024 arrays for temporary storage, as well as additional temporary storage for local reductions. For CPU implementation, the process had a maximum of 82.2MB for the AMD CPU, and 79.1MB used for the Intel CPU. This is a small fraction of available RAM, and is not the performance limiting factor. For smaller data sets, the performance of the CPU device performance of the OpenCL kernel was very close to performing as fast as the Boost Threaded C++ version, with Boost having
a slight edge. This is completely different from the GPU execution. Analyzing the results of
the GPU execution of the kernel vs CPU implementation showed that for 1024 data points,
the GPU is much slower. Analyzing with AMD CodeXL for the AMD GPU execution showed
that 90.8% of the time was spent waiting for the GPU device and memory transfer. This
dramatically impacts the performance of the GPU, and the CPU without any acceleration
is much faster than any of the GPU kernel executions.
Chapter 6

Conclusion

From the results, it can be concluded that OpenCL does allow for the CPU and GPU hardware to process LiDAR data in real time. [6] showed that the performance of OpenCL kernels on CPU’s cannot achieve the performance of threaded libraries such as Intel threaded or Boost threaded libraries. But the one key advantage that was experimentally found was the portability of the OpenCL kernels. OpenCL 1.2 kernel code can be executed on any hardware that supports the OpenCL 1.2 libraries. Since it is up to the hardware designer to publish the OpenCL drivers for each device, and not up to the developers of OpenCL to develop and maintain these drivers, compatibility of hardware is left to the hardware designers. Since it is up the hardware developers to meet the OpenCL standards, it allows a variety of hardware to be compatible, from CPU’s, GPU’s, FPGA, and ASIC chips. OpenCL allows for the same kernel code to be executed on this variety of hardware without the need of changing the kernel when changing hardware platforms.

This portability allows for selecting hardware based on the amount of processing capabilities needed. For small amounts of data, boards like the Parallella, which are low power ASIC assisted boards could be used. For larger data sets, a GPU could be selected to balance the number of processing cores and wattage requirements. The ability to split tasks to multiple devices allows even more flexibility.

The experimental results showed that the OpenCL devices can allow off the shelf parts to achieve real time performance. Off the shelf parts mean that students, research departments,
and commercial research and development can start testing algorithms faster and cheaper. OpenCL accelerated LiDAR sensor algorithms allow entry level budgets create and explore competitive research and applications with a diverse selection of devices. In the cases studied in this thesis, for large data sets, the Radeon 7970 was capable of maintaining real time data processing. On the other hand, smaller data sets could use hardware that has fewer cores and processing capabilities which could allow for smaller hardware and for less power consumed. This ability to run the same code on various hardware allows the trial and error of hardware choices to find the balance of computing power, wattage, and physical size needed for a project, without the need to alter the kernel code.

From a development side, the difficulty of developing OpenCL kernels lies in the immaturity of the library. NVIDIA’s CUDA certainly has the advantage of many more resources, communities and conferences for GPU computing. OpenCL is not as developer friendly when it comes to debugging broken code. This is due to the fact that each chipset manufacturer is responsible for the compiler drivers for the hardware, and not one single compiler for all hardware. Developer’s such as AMD are pushing tools that aid developers in analyzing the execution of OpenCL kernels, and break down the performance and pin point the bottlenecks that are occurring. More tools like this are needed in this field, as it is chipset manufacturer dependent. Again, this is a relatively immature library, and future development and growth will make it the future of OpenCL more interesting. With support from major manufactures such as AMD, Intel, ARM, NVIDIA and many FPGA manufacturers, the commitment to improvement and collaboration has created a great tool for programmers.

OpenCL continues to be supported by hardware designers and manufacturers, and are giving feedback into the future development of the library. Users are also impacting the evolution of this library, and the features available continue to grow. The diversity of new products open the doors up to new research platforms. The goal of this research was to allow for a diversity of hardware platforms to be supported, and with a growing number of low power OpenCL compatible hardware platforms being released, development of low power, small robotic research platforms can be developed. Testing of the implementation of the accelerated LiDAR functions on FPGA hardware would also be very interesting.
Future Work 59

OpenCL 1.3 will bring with it the capability for OpenCL kernels to be compiled to FPGA hardware. Current hardware essentially creates multiple CPU’s on the FPGA chip to execute the kernel. OpenCL 1.3 will allow for optimized compilation and only require the exact hardware needed for the kernel. Increased ability for splitting up kernel execution across devices, as well as executing multiple kernels on a single device, would bring interesting research into the performance capabilities of such setups.

Next generation OpenCL compatible devices are being released and are in development that will bring OpenCL kernels to new branches of low power hardware. The Parallela board is coming to market with 18 cores (2 ARM cores + 16 core ASIC ) that only consumes a maximum of five watts at maximum performance, as well as being roughly the size of a credit card. Future generations will incorporate an ASIC with 64 or 100 cores. Boards like this will allow projects with smaller budgets, low power, and small package size to be feasible. With LiDAR technology being developed into smaller packages and more availability, the need for increasing computing density will be necessary. Drones and other robotic platforms are being increasingly used for tasks that are either menial, or would be safer by removing the human aspect of the operation.

Solutions like this library allow developers to reuse the same hardware for multiple purposes without needing to switch out hardware for different target operations. Future development on the kernels would include the adoption of OpenCL 2.0, which the specifications for the standard was released July 22, 2013. Manufacturers are now releasing OpenCL drivers that meet this new standard and requirements. OpenCL 2.0 allows for many new features such as memory pipes, shared virtual memory, as well as Android support. Future development would also include expanding the library of functions based on feedback from other users. OpenCL 2.0 memory structures could open up the ability to develop a Voronoi algorithm, which is a commonly used optimal path detection algorithm. This algorithm in its current data structures requires many memory operations that would cause the kernel to essentially run in a single thread, and some research is now being done on developing a parallel method for this algorithm. OpenCL 2.0 also improves the compilation and execution of kernels on FPGA hardware, which would be another direction for future testing.
The next goal for this research is in the distribution and application of this library with other research departments in the field of robotics. This application would help facilitate more research in OpenCL acceleration for robotics research, as well as continue to grow the number of functions accelerated. The ability to select off the shelf hardware for a certain application allows for more opportunities to create test and research platforms that are both flexible and high performance.
Appendix A

Appendix A - Source Code

Write your Appendix content here.

A.1 LiDAR Sensor Data Interprettor- Matlab Source Code

sigma = 3;
halfwid = 5;
format long
%1D Mask
x = linspace(-halfwid, halfwid, 2*halfwid+1);
tmp1 = exp(-1./(sqrt(2.*sigma.^2))*(x.^2))
sum = 0;
for i=1:(2*halfwid+1)
    sum = sum+tmp1(i);
end
tmp1 = tmp1./sum;
sprintf('%0.10f & ', tmp1)
A.2 LiDAR TCP Convert

```matlab
%filename = uigetfile('All Files (*.*)')
fid = fopen('20100802_171300_TrajLigneDroite3.dat');
fName = 'output2.txt'; %# A file name
fout = fopen(fName,'w');

%Read Header File
F_ID = fread(fid, 8, '*char') %First 8 Bytes
F_Options = fread(fid, 1) %Options - 1 Byte
F_Lookup_Table_Address = fread(fid, 1, 'uint64')
fseek(fid, 17, 'bof');

%READ Time Stamp

%Read Network Packet
while true
    timestamp = fread(fid, 1, 'int64')
    header_type = fread(fid, 1, 'int8')
    header_n = fread(fid, 1, 'int16')
    sensor_X = fread(fid, 1, 'float')
    sensor_Y = fread(fid, 1, 'float')
    sensor_Z = fread(fid, 1, 'float')
    sensor_Yaw = fread(fid, 1, 'float')
    sensor_Pitch = fread(fid, 1, 'float')
    sensor_Roll = fread(fid, 1, 'float')

    for i = 1:header_n
        if ftell(fid) ~= F_Lookup_Table_Address
            data_yaw(i) = fread(fid, 1, 'uint16');
            data_pitch(i) = fread(fid, 1, 'uint16');
            data_distance(i) = fread(fid, 1, 'uint16');
        end
    end
end
```
data_intensity(i) = fread(fid, 1, 'uint8');
fprintf(fout,'%d\t%d\t%d\t%d\n',data_yaw(i), data_pitch(i), data_distance(i), data_intensity(i),

else
    break;
end
end
fprintf(fout, '\n');
if ftell(fid) == F_Lookup_Table_Address
    break;
else
end
fclose(fout);
fclose(fid);

A.3 OpenCL Kernels - Library Functions

A.3.1 Translation
Appendix B

Appendix B - OpenCL Hardware Overview

This section goes over the specifics of the OpenCL compatible hardware used for testing.

B.1 OpenCL Hardware Attributes

{NAME, VENDOR, PROFILE, TYPE} This parameter contains information about the hardware name, the vendor of the hardware, profile name supported by the device, as well as the type of hardware device.

NATIVE_VECTOR_WIDTH.{CHAR, INT, LONG, SHORT, DOUBLE, HALF, FLOAT} Returns the native ISA vector width. The vector width is defined as the number of scalar elements that can be stored in the vector.

PREFERRED_VECTOR_WIDTH.{CHAR, INT, LONG, SHORT, DOUBLE, HALF, FLOAT} Preferred native vector width size for built-in scalar types that can be put into vectors. The vector width is defined as the number of scalar elements that can be stored in the vector.

PREFERRED_INTEROP_USER_SYNC if the device’s preference is for the user to be responsible for synchronization, when sharing memory objects between OpenCL
and other APIs such as DirectX, CL_FALSE if the device / implementation has a performant path for performing synchronization of memory object shared between OpenCL and other APIs such as DirectX.

**ADDRESS_BITS** The default compute device address space size specified as an unsigned integer value in bits. Currently supported values are 32 or 64 bits.

**AVAILABLE** Is CL_TRUE if the device is available and CL_FALSE if the device is not available. Needed when executing two or more sets of kernels on different hardware.

**BUILT_IN_KERNELS** A semi-colon separated list of built-in kernels supported by the device. An empty string is returned if no built-in kernels are supported by the device.

**COMPILER_AVAILABLE** Is CL_FALSE if the implementation does not have a compiler available to compile the program source. Is CL_TRUE if the compiler is available. This can be CL_FALSE for the embedded platform profile only.

**{DOUBLE, HALF, SINGLE}.FP_CONFIG** Describes double precision floating-point capability of the OpenCL device.

**ENDIAN_LITTLE** Is CL_TRUE if the OpenCL device is a little endian device and CL_FALSE otherwise.

**EXTENSIONS** Returns a space separated list of extension names (the extension names themselves do not contain any spaces) supported by the device. The list of extension names returned can be vendor supported extension names and one or more of the following Khronos approved extension names:

**ERROR_CORRECTION_SUPPORT** s CL_TRUE if the device implements error correction for the memories, caches, registers etc. in the device. Is CL_FALSE if the device does not implement error correction. This can be a requirement for certain clients of OpenCL.

**EXECUTION_CAPABILITIES** Describes the execution capabilities of the device. This is a bit-field that describes one or more of the following values: the OpenCL device can execute OpenCL kernels or the OpenCL device can execute native kernels.
GLOBAL_MEM_CACHE.\{SIZE, TYPE\} Size of global memory cache in bytes. Type of global memory cache supported.

GLOBAL_MEM.\{CACHLINE_SIZE, SIZE\} Size of global memory cache line in bytes.

HOST.Unified_MEMORY

IMAGE_MAX.\{ARRAY, BUFFER\}_SIZE Max number of images in a 1D or 2D image array. Max number of pixels for a 1D image created from a buffer object.

IMAGE_SUPPORT Is CL_TRUE if images are supported by the OpenCL device and CL_FALSE otherwise.

IMAGE2D_MAX.\{WIDTH, HEIGHT\} Max height/width of 2D image in pixels.

IMAGE3D_MAX.\{WIDTH, HEIGHT, DEPTH\} Max height/width/depth of 3D image in pixels.

LOCAL_MEM.\{TYPE, SIZE\} Type of local memory supported. Size of local memory arena in bytes.

MAX.\{READ, WRITE\}_IMAGEARGS

MAX_CLOCK_FREQUENCY Maximum configured clock frequency of the device in MHz.

MAX_COMPUTE_UNITS The number of parallel compute units on the OpenCL device. A work-group executes on a single compute unit. The minimum value is 1.

MAX_CONSTANT.\{ARGS,BUFFER_SIZE\}

MAX.\{MEM_ALLOC, PARAMETER\}_SIZE Max size of memory object allocation in bytes. The minimum value is max (1/4th of CL DEVICE_GLOBAL_MEM.SIZE, 128*1024*1024). Max size in bytes of the arguments that can be passed to a kernel.

MAX_SAMPLERS Maximum number of samplers that can be used in a kernel.

MAX_WORK_GROUP_SIZE Maximum number of work-items in a work-group executing a kernel on a single compute unit, using the data parallel execution model.
MAX_WORK_ITEM_{DIMENSIONS, SIZES}  Maximum dimensions that specify the
global and local work-item IDs used by the data parallel execution model. Maximum
number of work-items that can be specified in each dimension of the work-group

MEM_BASE_ADDR_ALIGN The minimum value is the size (in bits) of the largest
OpenCL built-in data type supported by the device (long16 in FULL profile, long16
or int16 in EMBEDDED profile).

OPENCL_C_VERSION OpenCL C version string. Returns the highest OpenCL C ver-
sion supported by the compiler for this device that is not of type CL_DEVICE_TYPE_CUSTOM.

PARENT_DEVICE Returns the cl_device_id of the parent device to which this sub-device
belongs. If device is a root-level device, a NULL value is returned.

PARTITION_AFFINITY_DOMAIN Returns the list of supported affinity domains for
partitioning the device using CL_DEVICE_PARTITION_BY_AFFINITY_DOMAIN

PARTITION_MAX_SUB_DEVICES Returns the maximum number of sub-devices that
can be created when a device is partitioned.

PARTITION_{PROPERTIES, TYPE} Returns the properties argument specified in
clCreateSubDevices if device is a subdevice. This is used for subdivisions of the hard-
ware for execution of multiple kernels on single hardware.

PLATFORM The platform associated with this device.

PRINTF_BUFFER_SIZE Maximum size of the internal buffer that holds the output of
printf calls from a kernel.

PROFILING_TIMER_RESOLUTION Describes the resolution of device timer. This
is measured in nanoseconds.

QUEUE_PROPERTIES Describes the command-queue properties supported by the de-
vice.

REFERENCE_COUNT Returns the device reference count. If the device is a root-level
device, a reference count of one is returned.
VENDOR_ID, CL_{DEVICE, DRIVER}_{VERSION} A unique device vendor identifier. An example of a unique device identifier could be the PCIe ID. OpenCL version string. Returns the OpenCL version supported by the device.

## B.2 Central Processing Units

### B.2.1 AMD FX™-6200

<table>
<thead>
<tr>
<th>ATTRIBUTE</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_DEVICE_ADDRESS_BITS</td>
<td>32</td>
</tr>
<tr>
<td>CL_DEVICE_AVAILABLE</td>
<td>1</td>
</tr>
<tr>
<td>CL_DEVICE_COMPILER_AVAILABLE</td>
<td>1</td>
</tr>
<tr>
<td>CL_DEVICE_ENDIAN_LITTLE</td>
<td>1</td>
</tr>
<tr>
<td>CL_DEVICE_ERROR_CORRECTION_SUPPORT</td>
<td>0</td>
</tr>
<tr>
<td>CL_DEVICE_EXECUTION_CAPABILITIES</td>
<td>3</td>
</tr>
<tr>
<td>CL_DEVICE_EXTENSIONS</td>
<td>cl_amd_device_attribute_query</td>
</tr>
<tr>
<td></td>
<td>cl_amd_fp64</td>
</tr>
<tr>
<td></td>
<td>cl_amd_media_ops</td>
</tr>
<tr>
<td></td>
<td>cl_amd_media_ops2</td>
</tr>
<tr>
<td></td>
<td>cl_amd_popcnt</td>
</tr>
<tr>
<td></td>
<td>cl_amd_printf</td>
</tr>
<tr>
<td></td>
<td>cl_amd_vec3</td>
</tr>
<tr>
<td></td>
<td>cl_ext_device_fission</td>
</tr>
<tr>
<td></td>
<td>cl_khr_3d_imageWrites</td>
</tr>
<tr>
<td></td>
<td>cl_khr_byte_addressable_store</td>
</tr>
<tr>
<td></td>
<td>cl_khr_d3d10_sharing</td>
</tr>
<tr>
<td></td>
<td>cl_khr_fp64</td>
</tr>
<tr>
<td></td>
<td>cl_khr_gl_sharing</td>
</tr>
<tr>
<td></td>
<td>cl_khr_global_int32_base_atomics</td>
</tr>
<tr>
<td></td>
<td>cl_khr_global_int32_extended_atomics</td>
</tr>
</tbody>
</table>
Appendix B. OpenCL Hardware Overview

CL_DEVICE_GLOBAL_MEM_CACHE_SIZE
CL_DEVICE_GLOBAL_MEM_CACHE_TYPE
CL_DEVICE_GLOBAL_MEM_CACHELINE_SIZE
CL_DEVICE_GLOBAL_MEM_SIZE
CL_DEVICE_HOST_UNIFIED_MEMORY
CL_DEVICE_IMAGE2D_MAX_HEIGHT
CL_DEVICE_IMAGE2D_MAX_WIDTH
CL_DEVICE_IMAGE3D_MAX_DEPTH
CL_DEVICE_IMAGE3D_MAX_HEIGHT
CL_DEVICE_IMAGE3D_MAX_WIDTH
CL_DEVICE_IMAGE_SUPPORT
CL_DEVICE_LOCAL_MEM_SIZE
CL_DEVICE_LOCAL_MEM_TYPE
CL_DEVICE_MAX_CLOCK_FREQUENCY
CL_DEVICE_MAX_COMPUTE_UNITS
CL_DEVICE_MAX_CONSTANT_ARGS
CL_DEVICE_MAX_CONSTANT_BUFFER_SIZE
CL_DEVICE_MAX_MEM_ALLOC_SIZE
CL_DEVICE_MAX_PARAMETER_SIZE
CL_DEVICE_MAX_READ_IMAGE_ARGS
CL_DEVICE_MAX_SAMPLERS
CL_DEVICE_MAX_WORK_GROUP_SIZE
CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS
CL_DEVICE_MAX_WORK_ITEM_SIZES
CL_DEVICE_MAX_WRITE_IMAGE_ARGS
CL_DEVICE_MEM_BASE_ADDR_ALIGN
CL_DEVICE_NAME
CL_DEVICE_NATIVE_VECTOR_WIDTH_CHAR

cl_khr_local_int32_base_atomics
cl_khr_local_int32_extended_atomics
16384
2
64
2147483648
1
8192
8192
2048
2048
2048
1
32768
2
3800
3812
6
8
65536
1073741824
4096
128
16
1024
3
"-102
8
1024
AMD FX(tm)-6200 Six-Core Processor
16
| CL_DEVICE_NATIVE_VECTOR_WIDTH_DOUBLE | 0 |
| CL_DEVICE_NATIVE_VECTOR_WIDTH_FLOAT  | 4 |
| CL_DEVICE_NATIVE_VECTOR_WIDTH_HALF   | 4 |
| CL_DEVICE_NATIVE_VECTOR_WIDTH_INT    | 4 |
| CL_DEVICE_NATIVE_VECTOR_WIDTH_LONG   | 2 |
| CL_DEVICE_OPENCL_C_VERSION          | 1.2 |
| CL_DEVICE_PREFERRED_VECTOR_WIDTH_CHAR| 16 |
| CL_DEVICE_PREFERRED_VECTOR_WIDTH_DOUBLE| 0 |
| CL_DEVICE_PREFERRED_VECTOR_WIDTH_FLOAT| 4 |
| CL_DEVICE_PREFERRED_VECTOR_WIDTH_HALF| 4 |
| CL_DEVICE_PREFERRED_VECTOR_WIDTH_INT| 4 |
| CL_DEVICE_PREFERRED_VECTOR_WIDTH_LONG| 2 |
| CL_DEVICE_PROFILING_TIMER_RESOLUTION| 268 |
| CL_DEVICE_QUEUE_PROPERTIES          | 2 |
| CL_DEVICE_SINGLE_FP_CONFIG          | 191 |
| CL_DEVICE_TYPE                       | 2 |
| CL_DEVICE_VENDOR                    | AuthenticAMD |
| CL_DEVICE_VENDOR_ID                 | 4098 |
| CL_DEVICE_VERSION                   | 1.2 |
| CL_DRIVER_VERSION                   | "1084.4 (sse2" |
|                                      | "2.0 (sse2" |
B.2.2 Intel® Core™ i5-3230M CPU @ 2.60GHz

Table B.2: Intel Core i5-3230M

<table>
<thead>
<tr>
<th>ATTRIBUTE</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL DEVICE_ADDRESS_BITS</td>
<td>32</td>
</tr>
<tr>
<td>CL DEVICE AVAILABLE</td>
<td>1</td>
</tr>
<tr>
<td>CL DEVICE_COMPILER_AVAILABLE</td>
<td>1</td>
</tr>
<tr>
<td>CL DEVICE_ENDIAN_LITTLE</td>
<td>1</td>
</tr>
<tr>
<td>CL DEVICE_ERROR_CORRECTION_SUPPORT</td>
<td>0</td>
</tr>
<tr>
<td>CL DEVICE_EXECUTION_CAPABILITIES</td>
<td>3</td>
</tr>
<tr>
<td>CL DEVICE_EXTENSIONS</td>
<td>cl_ext_device_fission</td>
</tr>
<tr>
<td>cl_intel_dx9_media_sharing</td>
<td></td>
</tr>
<tr>
<td>cl_intel_exec_by_local_thread</td>
<td></td>
</tr>
<tr>
<td>cl_intel_printf</td>
<td></td>
</tr>
<tr>
<td>cl_khr_byte_addressable_store</td>
<td></td>
</tr>
<tr>
<td>cl_khr_d3d11_sharing</td>
<td></td>
</tr>
<tr>
<td>cl_khr_dx9_media_sharing</td>
<td></td>
</tr>
<tr>
<td>cl_khr_fp64</td>
<td></td>
</tr>
<tr>
<td>cl_khr_gl_sharing</td>
<td></td>
</tr>
<tr>
<td>cl_khr_global_int32_base_atomics</td>
<td></td>
</tr>
<tr>
<td>cl_khr_global_int32_extended_atomics</td>
<td></td>
</tr>
<tr>
<td>cl_khr_icd</td>
<td></td>
</tr>
<tr>
<td>cl_khr_local_int32_base_atomics</td>
<td></td>
</tr>
<tr>
<td>cl_khr_local_int32_extended_atomics</td>
<td></td>
</tr>
<tr>
<td>CL DEVICE GLOBAL MEM CACHE SIZE</td>
<td>262144</td>
</tr>
<tr>
<td>CL DEVICE GLOBAL MEM CACHE TYPE</td>
<td>2</td>
</tr>
<tr>
<td>CL DEVICE GLOBAL MEM CACHELINE SIZE</td>
<td>64</td>
</tr>
<tr>
<td>CL DEVICE GLOBAL MEM SIZE</td>
<td>2147352576</td>
</tr>
<tr>
<td>CL DEVICE HOST UNIFIED_MEMORY</td>
<td>1</td>
</tr>
<tr>
<td>CL DEVICE IMAGE2D MAX HEIGHT</td>
<td>16384</td>
</tr>
<tr>
<td>8192</td>
<td></td>
</tr>
<tr>
<td>CL DEVICE IMAGE2D MAX WIDTH</td>
<td>16384</td>
</tr>
<tr>
<td>Variable</td>
<td>Value</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>CL_DEVICE_IMAGE3D_MAX_DEPTH</td>
<td>2048</td>
</tr>
<tr>
<td>CL_DEVICE_IMAGE3D_MAX_HEIGHT</td>
<td>2048</td>
</tr>
<tr>
<td>CL_DEVICE_IMAGE3D_MAX_WIDTH</td>
<td>2048</td>
</tr>
<tr>
<td>CL_DEVICE_IMAGE_SUPPORT</td>
<td>1</td>
</tr>
<tr>
<td>CL_DEVICE_LOCAL_MEM_SIZE</td>
<td>32768</td>
</tr>
<tr>
<td>CL_DEVICE_LOCAL_MEM_TYPE</td>
<td>2</td>
</tr>
<tr>
<td>CL_DEVICE_MAX_CLOCK_FREQUENCY</td>
<td>2600</td>
</tr>
<tr>
<td>CL_DEVICE_MAX_COMPUTE_UNITS</td>
<td>4</td>
</tr>
<tr>
<td>CL_DEVICE_MAX_CONSTANT_ARGS</td>
<td>480</td>
</tr>
<tr>
<td>CL_DEVICE_MAX_CONSTANT_BUFFER_SIZE</td>
<td>131072</td>
</tr>
<tr>
<td>CL_DEVICE_MAX_MEM_ALLOC_SIZE</td>
<td>536838144</td>
</tr>
<tr>
<td>CL_DEVICE_MAX_PARAMETER_SIZE</td>
<td>3840</td>
</tr>
<tr>
<td>CL_DEVICE_MAX_READ_IMAGE_ARGS</td>
<td>480</td>
</tr>
<tr>
<td>CL_DEVICE_MAX_SAMPLERS</td>
<td>480</td>
</tr>
<tr>
<td>CL_DEVICE_MAX_WORK_GROUP_SIZE</td>
<td>1024</td>
</tr>
<tr>
<td>CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS</td>
<td>3</td>
</tr>
<tr>
<td>CL_DEVICE_MAX_WORK_ITEM_SIZES</td>
<td>(1024,1024,1024)</td>
</tr>
<tr>
<td>CL_DEVICE_MAX_WRITE_IMAGE_ARGS</td>
<td>480</td>
</tr>
<tr>
<td>CL_DEVICE_MEM_BASE_ADDR_ALIGN</td>
<td>1024</td>
</tr>
<tr>
<td>CL_DEVICE_NAME</td>
<td>Intel(R) Core(TM) i5-3230M CPU</td>
</tr>
<tr>
<td>CL_DEVICE_NATIVE_VECTOR_WIDTH_CHAR</td>
<td>16</td>
</tr>
<tr>
<td>CL_DEVICE_NATIVE_VECTOR_WIDTH_DOUBLE</td>
<td>2</td>
</tr>
<tr>
<td>CL_DEVICE_NATIVE_VECTOR_WIDTH_FLOAT</td>
<td>2</td>
</tr>
<tr>
<td>CL_DEVICE_NATIVE_VECTOR_WIDTH_HALF</td>
<td>0</td>
</tr>
<tr>
<td>CL_DEVICE_NATIVE_VECTOR_WIDTH_INT</td>
<td>4</td>
</tr>
<tr>
<td>CL_DEVICE_NATIVE_VECTOR_WIDTH_LONG</td>
<td>2</td>
</tr>
<tr>
<td>CL_DEVICE_OPENCL_C_VERSION</td>
<td>1.1</td>
</tr>
</tbody>
</table>
1.2
CL_DEVICE_PREFERRED VECTOR_WIDTH_CHAR 1
16
CL_DEVICE_PREFERRED VECTOR_WIDTH_DOUBLE 1
2
CL_DEVICE_PREFERRED VECTOR_WIDTH_FLOAT 1
4
CL_DEVICE_PREFERRED VECTOR_WIDTH_HALF 0
CL_DEVICE_PREFERRED VECTOR_WIDTH_INT 1
4
CL_DEVICE_PREFERRED VECTOR_WIDTH_LONG 1
2
CL_DEVICE_PROFILING TIMER_RESOLUTION 394
CL_DEVICE_QUEUE_PROPERTIES 3
CL_DEVICE_SINGLE_FP_CONFIG 7
CL_DEVICE_TYPE 2
CL_DEVICE_VENDOR Intel(R) Corporation
CL_DEVICE_VENDOR_ID 32902
CL_DEVICE_VERSION 1.1
1.2
CL_DRIVER_VERSION 1.1
1.2

B.3 GPUs

B.3.1 AMD Radeon 7970

<table>
<thead>
<tr>
<th>ATTRIBUTE</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_DEVICE_ADDRESS_BITS</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>64</td>
</tr>
<tr>
<td>CL_DEVICEAVAILABLE</td>
<td>1</td>
</tr>
<tr>
<td>--------------------</td>
<td>---</td>
</tr>
<tr>
<td>CL_DEVICE_COMPILERAVAILABLE</td>
<td>1</td>
</tr>
<tr>
<td>CL_DEVICE_ENDIANLITTLE</td>
<td>1</td>
</tr>
<tr>
<td>CL_DEVICE_ERRORCORRECTION_SUPPORT</td>
<td>0</td>
</tr>
<tr>
<td>CL_DEVICE_EXECUTIONCAPABILITIES</td>
<td>1</td>
</tr>
<tr>
<td>CL_DEVICE_EXTENSIONS</td>
<td>cl_amd_c1x_atomics, cl_amd_device_attribute_query, cl_amd_fp64, cl_amd_media_ops, cl_amd_media_ops2, cl_amd_meminfo, cl_amd_popcnt, cl_amd_printf, cl_amd_vec3, cl_ext_atomic_counters_32, cl_khr_3d_image_writes, cl_khr_byte_addressable_store, cl_khr_d3d10_sharing, cl_khr_dx9_media_sharing, cl_khr_fp64, cl_khr_gl_sharing, cl_khr_global_int32_base_atomics, cl_khr_global_int32_extended_atomics, cl_khr_image2d_from_buffer, cl_khr_int64_base_atomics, cl_khr_int64_extended_atomics, cl_khr_local_int32_base_atomics, cl_khr_local_int32_extended_atomics</td>
</tr>
<tr>
<td>CL_DEVICE_GLOBAL_MEM_CACHE_SIZE</td>
<td>0</td>
</tr>
<tr>
<td>CL_DEVICE_GLOBAL_MEM_CACHE_TYPE</td>
<td>0</td>
</tr>
</tbody>
</table>
### OpenCL Hardware Overview

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL_DEVICE_GLOBAL_MEM_CACHELINE_SIZE</td>
<td>0</td>
</tr>
<tr>
<td>CL_DEVICE_GLOBAL_MEM_SIZE</td>
<td>2147483648</td>
</tr>
<tr>
<td>CL_DEVICE_GLOBAL_MEM_SIZE</td>
<td>3221225472</td>
</tr>
<tr>
<td>CL_DEVICE_GLOBAL_MEM_SIZE</td>
<td>6442450944</td>
</tr>
<tr>
<td>CL_DEVICE_GLOBAL_MEM_SIZE</td>
<td>838860800</td>
</tr>
<tr>
<td>CL_DEVICE_HOST_UNIFIED_MEMORY</td>
<td>0</td>
</tr>
<tr>
<td>CL_DEVICE_IMAGE2D_MAX_HEIGHT</td>
<td>16384</td>
</tr>
<tr>
<td>CL_DEVICE_IMAGE2D_MAX_WIDTH</td>
<td>8192</td>
</tr>
<tr>
<td>CL_DEVICE_IMAGE2D_MAX_WIDTH</td>
<td>16384</td>
</tr>
<tr>
<td>CL_DEVICE_IMAGE2D_MAX_WIDTH</td>
<td>8192</td>
</tr>
<tr>
<td>CL_DEVICE_IMAGE3D_MAX_DEPTH</td>
<td>2048</td>
</tr>
<tr>
<td>CL_DEVICE_IMAGE3D_MAX_HEIGHT</td>
<td>2048</td>
</tr>
<tr>
<td>CL_DEVICE_IMAGE3D_MAX_WIDTH</td>
<td>2048</td>
</tr>
<tr>
<td>CL_DEVICE_IMAGE_SUPPORT</td>
<td>1</td>
</tr>
<tr>
<td>CL_DEVICE_LOCAL_MEM_SIZE</td>
<td>32768</td>
</tr>
<tr>
<td>CL_DEVICE_LOCAL_MEM_TYPE</td>
<td>1</td>
</tr>
<tr>
<td>CL_DEVICE_MAX_COMPUTE_UNITS</td>
<td>28</td>
</tr>
<tr>
<td>CL_DEVICE_MAX_COMPUTE_UNITS</td>
<td>32</td>
</tr>
<tr>
<td>CL_DEVICE_MAX_CONSTANT_ARGS</td>
<td>8</td>
</tr>
<tr>
<td>CL_DEVICE_MAX_CONSTANT_BUFFER_SIZE</td>
<td>65536</td>
</tr>
<tr>
<td>CL_DEVICE_MAX_MEM_ALLOC_SIZE</td>
<td>1877999616</td>
</tr>
<tr>
<td>CL_DEVICE_MAX_MEM_ALLOC_SIZE</td>
<td>209715200</td>
</tr>
<tr>
<td>CL_DEVICE_MAX_MEM_ALLOC_SIZE</td>
<td>2950692864</td>
</tr>
<tr>
<td>CL_DEVICE_MAX_MEM_ALLOC_SIZE</td>
<td>2951741440</td>
</tr>
<tr>
<td>CL_DEVICE_MAX_MEM_ALLOC_SIZE</td>
<td>536870912</td>
</tr>
<tr>
<td>CL_DEVICE_MAX_MEM_ALLOC_SIZE</td>
<td>805306368</td>
</tr>
<tr>
<td>CL_DEVICE_MAX_PARAMETER_SIZE</td>
<td>1024</td>
</tr>
<tr>
<td>CL_DEVICE_MAX_PARAMETER_SIZE</td>
<td>4096</td>
</tr>
<tr>
<td>ATTRIBUTE</td>
<td>VALUE</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>CL_DEVICE_ADDRESS_BITS</td>
<td>32</td>
</tr>
<tr>
<td>CL_DEVICE_AVAILABLE</td>
<td>1</td>
</tr>
<tr>
<td>CL_DEVICE_COMPILERAVAILABLE</td>
<td>1</td>
</tr>
<tr>
<td>CL_DEVICE_COMPUTE_CAPACITY_MAJOR_NV</td>
<td>3</td>
</tr>
<tr>
<td>CL_DEVICE_COMPUTE_CAPACITY_MINOR_NV</td>
<td>0</td>
</tr>
<tr>
<td>CL_DEVICE_ENDIAN_LITTLE</td>
<td>1</td>
</tr>
<tr>
<td>CL_DEVICE_ERRORCORRECTION_SUPPORT</td>
<td>0</td>
</tr>
<tr>
<td>CL_DEVICE_EXECUTION_CAPABILITIES</td>
<td>1</td>
</tr>
<tr>
<td>CL_DEVICE_EXTENSIONS</td>
<td>cl_khr_byte_addressable_store</td>
</tr>
</tbody>
</table>
Appendix B. *OpenCL Hardware Overview*

<table>
<thead>
<tr>
<th>CL DEVICE GLOBAL MEM CACHE SIZE</th>
<th>32768</th>
</tr>
</thead>
<tbody>
<tr>
<td>CL DEVICE GLOBAL MEM CACHE TYPE</td>
<td>2</td>
</tr>
<tr>
<td>CL DEVICE GLOBAL MEM CACHELINE SIZE</td>
<td>128</td>
</tr>
<tr>
<td>CL DEVICE HOST UNIFIED MEMORY</td>
<td>0</td>
</tr>
<tr>
<td>CL DEVICE IMAGE2D MAX HEIGHT</td>
<td>32768</td>
</tr>
<tr>
<td>CL DEVICE IMAGE2D MAX WIDTH</td>
<td>32768</td>
</tr>
<tr>
<td>CL DEVICE IMAGE3D MAX DEPTH</td>
<td>4096</td>
</tr>
<tr>
<td>CL DEVICE IMAGE3D MAX HEIGHT</td>
<td>4096</td>
</tr>
<tr>
<td>CL DEVICE IMAGE3D MAX WIDTH</td>
<td>4096</td>
</tr>
<tr>
<td>CL DEVICE IMAGE SUPPORT</td>
<td>1</td>
</tr>
<tr>
<td>CL DEVICE LOCAL MEM SIZE</td>
<td>49152</td>
</tr>
<tr>
<td>CL DEVICE LOCAL MEM TYPE</td>
<td>1</td>
</tr>
<tr>
<td>CL DEVICE MAX CLOCK FREQUENCY</td>
<td>895</td>
</tr>
<tr>
<td>CL DEVICE MAX COMPUTE UNITS</td>
<td>2</td>
</tr>
<tr>
<td>CL DEVICE MAX CONSTANT ARGS</td>
<td>9</td>
</tr>
<tr>
<td>CL DEVICE MAX CONSTANT BUFFER SIZE</td>
<td>65536</td>
</tr>
<tr>
<td>Key</td>
<td>Value</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>CL_DEVICE_MAX_MEM_ALLOC_SIZE</td>
<td>536870912</td>
</tr>
<tr>
<td>CL_DEVICE_MAX_PARAMETER_SIZE</td>
<td>4352</td>
</tr>
<tr>
<td>CL_DEVICE_MAX_READ_IMAGE_ARGS</td>
<td>256</td>
</tr>
<tr>
<td>CL_DEVICE_MAX_SAMPLERS</td>
<td>32</td>
</tr>
<tr>
<td>CL_DEVICE_MAX_WORK_GROUP_SIZE</td>
<td>1024</td>
</tr>
<tr>
<td>CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS</td>
<td>3</td>
</tr>
<tr>
<td>CL_DEVICE_MAX_WORK_ITEM_SIZES</td>
<td>1024</td>
</tr>
<tr>
<td>CL_DEVICE_MAX_WRITE_IMAGE_ARGS</td>
<td>16</td>
</tr>
<tr>
<td>CL_DEVICE_MEM_BASE_ADDR_ALIGN</td>
<td>4096</td>
</tr>
<tr>
<td>CL_DEVICE_NAME</td>
<td>GeForce GT 740M</td>
</tr>
<tr>
<td>CL_DEVICE_NATIVE_VECTOR_WIDTH_CHAR</td>
<td>1</td>
</tr>
<tr>
<td>CL_DEVICE_NATIVE_VECTOR_WIDTH_DOUBLE</td>
<td>1</td>
</tr>
<tr>
<td>CL_DEVICE_NATIVE_VECTOR_WIDTH_FLOAT</td>
<td>1</td>
</tr>
<tr>
<td>CL_DEVICE_NATIVE_VECTOR_WIDTH_HALF</td>
<td>0</td>
</tr>
<tr>
<td>CL_DEVICE_NATIVE_VECTOR_WIDTH_INT</td>
<td>1</td>
</tr>
<tr>
<td>CL_DEVICE_NATIVE_VECTOR_WIDTH_LONG</td>
<td>1</td>
</tr>
<tr>
<td>CL_DEVICE_OPENCL_C_VERSION</td>
<td>1.1</td>
</tr>
<tr>
<td>CL_DEVICE_PREFERRED_VECTOR_WIDTH_CHAR</td>
<td>1</td>
</tr>
<tr>
<td>CL_DEVICE_PREFERRED_VECTOR_WIDTH_DOUBLE</td>
<td>1</td>
</tr>
<tr>
<td>CL_DEVICE_PREFERRED_VECTOR_WIDTH_FLOAT</td>
<td>1</td>
</tr>
<tr>
<td>CL_DEVICE_PREFERRED_VECTOR_WIDTH_HALF</td>
<td>0</td>
</tr>
<tr>
<td>CL_DEVICE_PREFERRED_VECTOR_WIDTH_INT</td>
<td>1</td>
</tr>
<tr>
<td>CL_DEVICE_PREFERRED_VECTOR_WIDTH_LONG</td>
<td>1</td>
</tr>
<tr>
<td>CL_DEVICE_PROFILING_TIMER_RESOLUTION</td>
<td>1000</td>
</tr>
<tr>
<td>CL_DEVICE_QUEUE_PROPERTIES</td>
<td>3</td>
</tr>
<tr>
<td>CL_DEVICE_SINGLE_FP_CONFIG</td>
<td>63</td>
</tr>
<tr>
<td>CL_DEVICE_TYPE</td>
<td>4</td>
</tr>
<tr>
<td>CL_DEVICE_VENDOR</td>
<td>NVIDIA Corporation</td>
</tr>
<tr>
<td>CL_DEVICE_VENDOR_ID</td>
<td>4318</td>
</tr>
<tr>
<td>CL_DEVICE_VERSION</td>
<td>1.1</td>
</tr>
<tr>
<td>CL_DRIVER_VERSION</td>
<td>311</td>
</tr>
</tbody>
</table>
# Appendix C

## Appendix C - Tables & Charts

### C.1 DARPA Ground Vehicle Challenge - Sensors

<table>
<thead>
<tr>
<th>Team</th>
<th>Sensors</th>
</tr>
</thead>
<tbody>
<tr>
<td>University of Texas at Austin</td>
<td>Velodyne HDL-64E (10hz), two SICK LiDAR (Line), 4 Sterovision Cameras 4 Lidars, 10 Cameras 5 sterovision cameras 2xSICK 2D LMS-291 LiDAR, Velodyne LiDAR, Stereovision 4xSICK LM220</td>
</tr>
<tr>
<td>US and Israeli defense R&amp;D groups</td>
<td></td>
</tr>
<tr>
<td>Axion Racing</td>
<td></td>
</tr>
<tr>
<td>University of Pennsylvania</td>
<td></td>
</tr>
<tr>
<td>California Institute of Technology/Jet Propulsion Laboratory</td>
<td></td>
</tr>
<tr>
<td>Technische Universität Carolo-Wilhelmina zu Braunschweig</td>
<td></td>
</tr>
<tr>
<td>University of Florida</td>
<td></td>
</tr>
<tr>
<td>Ford Motor Company Research &amp; Advanced Engineering</td>
<td></td>
</tr>
<tr>
<td>Insight Racing Team</td>
<td></td>
</tr>
<tr>
<td>Team MIT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7 Lidar’s</td>
</tr>
<tr>
<td></td>
<td>Velodyne HDL-64E, Reigl LiDAR</td>
</tr>
<tr>
<td></td>
<td>7 SICK LMS 291 LiDARs</td>
</tr>
<tr>
<td>Team Name</td>
<td>LiDARs/ LADARs &amp; Rate</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>Team Mojavaton</td>
<td>2 SICK LMS 291 LiDARs</td>
</tr>
<tr>
<td>Team ODY-ERA</td>
<td>3 640x480, 3 320x240 (10fps)</td>
</tr>
<tr>
<td>Ohio State University</td>
<td>2x LMS221, Ibeo Alasca XT</td>
</tr>
<tr>
<td>Princeton University</td>
<td>4 LADAR (13ms rate)</td>
</tr>
<tr>
<td>SciAutonics / Auburn Engineering</td>
<td>Velodyne HD, two IBEO Alasca XT</td>
</tr>
<tr>
<td>Standford</td>
<td>5 x SICK LMS-291</td>
</tr>
<tr>
<td>Team Sting</td>
<td>SICK LMS 291-S05/S14 Lidar, Velodyne HDL-64, Continental ISF 172 Lidar</td>
</tr>
<tr>
<td>Carnegie Mellon University</td>
<td>3 x IBEO Alasca XT, 2 x SICK LMS 291</td>
</tr>
<tr>
<td>VictorTango</td>
<td>Velodyne HDL-64E (10hz), two SICK LiDAR (Line)</td>
</tr>
<tr>
<td>Team AnnieWAY</td>
<td>(5) Intel Core 2 Duo Quad-core</td>
</tr>
<tr>
<td>Team Autonomous Solutions</td>
<td>(2) SICK Lidar, IBEO-LD LIDAR</td>
</tr>
<tr>
<td>Team Berlin</td>
<td>(2) Alasca XT LIDAR sensors, (3) SICK LMS</td>
</tr>
<tr>
<td>Team CajunBot</td>
<td>(3) SICK LMS</td>
</tr>
<tr>
<td>Case Western Reserve University</td>
<td>(3) SICK LMS, (1) Alasca XT</td>
</tr>
<tr>
<td>Cornell</td>
<td>(1) LADAR</td>
</tr>
<tr>
<td>Team Cybernet</td>
<td>(2) Ibeo ALASCA XT</td>
</tr>
<tr>
<td>Team Gray</td>
<td>(4) SICK LMS</td>
</tr>
<tr>
<td>Team Jefferson</td>
<td>(2) SICK LMS</td>
</tr>
<tr>
<td>Team Juggernaut</td>
<td>(3) SICK LUX LiDAR</td>
</tr>
<tr>
<td>Team-LUX</td>
<td>(3) COTS LIDAR</td>
</tr>
<tr>
<td>TEAM OSHKOSH</td>
<td>(6) SICK LMS</td>
</tr>
<tr>
<td>University of Central Florida</td>
<td>(5) SICK LMS</td>
</tr>
<tr>
<td>Team Urbanator</td>
<td>Velodyne HDL-64E, (2) Sick LMS-221, (4) Sick LMS-291</td>
</tr>
<tr>
<td>The Golem Group</td>
<td>SICK LMS-221</td>
</tr>
<tr>
<td>University of Utah</td>
<td></td>
</tr>
</tbody>
</table>
### Table C.2: DARPA Ground Vehicle - Computers

<table>
<thead>
<tr>
<th>Team</th>
<th>Sensors</th>
</tr>
</thead>
<tbody>
<tr>
<td>University of Texas at Austin</td>
<td>4 x (Opteron Dual Core + 2 NVIDIA 7900 GT)</td>
</tr>
<tr>
<td>US and Israeli defense R&amp;D groups</td>
<td>IBM Servers</td>
</tr>
<tr>
<td>Axion Racing</td>
<td>(5) Dell 2650 dual Xeon Servers</td>
</tr>
<tr>
<td>University of Pennsylvania</td>
<td>(8) Mac Mini</td>
</tr>
<tr>
<td>California Institute of Technology/Jet Propulsion Laboratory</td>
<td>N/A</td>
</tr>
<tr>
<td>Technische Universität Carolo-Wilhelmina zu Braunschweig</td>
<td>N/A</td>
</tr>
<tr>
<td>University of Florida</td>
<td>10 AMD X2 4600</td>
</tr>
<tr>
<td>Ford Motor Company Research &amp; Advanced Engineering</td>
<td>N/A</td>
</tr>
<tr>
<td>Insight Racing Team</td>
<td>N/A</td>
</tr>
<tr>
<td>Team MIT</td>
<td>Fujitsu Blade Server</td>
</tr>
<tr>
<td>Team Mojavaton</td>
<td>(2) Intel 5130 Dual Core Xeon processors</td>
</tr>
<tr>
<td>Team ODY-ERA</td>
<td>N/A</td>
</tr>
<tr>
<td>Ohio State University</td>
<td>N/A</td>
</tr>
<tr>
<td>Princeton University</td>
<td>(4) Intel Core 2 Duo processors,</td>
</tr>
<tr>
<td>SciAutonics / Auburn Engineering</td>
<td>N/A</td>
</tr>
<tr>
<td>Stanford</td>
<td>N/A</td>
</tr>
<tr>
<td>Team Sting</td>
<td>(8) Dual-Core Intel XEON 5120</td>
</tr>
<tr>
<td>Carnegie Mellon University</td>
<td>N/A</td>
</tr>
<tr>
<td>VictorTango</td>
<td>(2) HP Proliant DL140 with a 2 Ghz quad core CPU</td>
</tr>
<tr>
<td>Team AnnieWAY</td>
<td>AMD dual-core Opteron</td>
</tr>
<tr>
<td>Team Autonomous Solutions</td>
<td>(6) Matrox Odyssey XPRO</td>
</tr>
<tr>
<td>Team Berlin</td>
<td>(5) Rack mounted computer</td>
</tr>
<tr>
<td>Team CajunBot</td>
<td>(3) 1.8 GHz Pentium M.</td>
</tr>
<tr>
<td>Case Western Reserve University</td>
<td>(3) Mac Mini, (3) National Instruments PXI</td>
</tr>
</tbody>
</table>
### C.2 Charts

#### C.2.1 AMD FX\textsuperscript{TM} -6200 - System Wattage

<table>
<thead>
<tr>
<th>Team</th>
<th>Processor Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cornell</td>
<td>N/A</td>
</tr>
<tr>
<td>Team Cybernet</td>
<td>N/A</td>
</tr>
<tr>
<td>Team Gray</td>
<td>N/A</td>
</tr>
<tr>
<td>Team Jefferson</td>
<td>N/A</td>
</tr>
<tr>
<td>Team Juggernaut</td>
<td>N/A</td>
</tr>
<tr>
<td>Team-LUX</td>
<td>(4) Pentium M processor running at 1.8 GHz</td>
</tr>
<tr>
<td>TEAM OSHKOSH</td>
<td>N/A</td>
</tr>
<tr>
<td>University of Central Florida</td>
<td></td>
</tr>
<tr>
<td>Team Urbanator</td>
<td>N/A</td>
</tr>
<tr>
<td>The Golem Group</td>
<td>(8) dual core 2.2 GHz Opteron 848HE</td>
</tr>
<tr>
<td>University of Utah</td>
<td>(2) Dual core Athlon 64 bit 3800+</td>
</tr>
</tbody>
</table>

![AMD FX-6200 Idle and Full Load](image-url)
Appendix C. Tables & Charts

C.2.2 AMD FX™-6200 & Radeon 7970 System Wattage

AMD FX-6200 w/ Radeon 7970 Idle and Full Load

C.2.3 AMD FX™-6200 & Geforce 210 System Wattage

AMD FX-6200 w/ Geforce 210 Idle and Full Load
Appendix C. Tables & Charts

C.2.4 Intel Core i5 System Wattage

![Graph showing Intel Core i5 Idle and Full Load wattage over time](image1)

C.2.5 Intel Core i5 & GeForce 740M System Wattage

![Graph showing Intel Core i5 & GeForce 740M Idle and Full Load wattage over time](image2)
Appendix D

Appendix D - OpenCL Kernel Source Code

D.1 Depth Data Coordinate Conversion

D.1.1 3D Data Point Translation - Integer Point Data

```c
__kernel void TranslateFlat3D(__global int3 * origData, __constant float * transData,
                              __global int3 * outputData) {
    __local int i;
    __local float3 tempPoint;
    __local float3 tempFinalPoint;
    __local float3 tempPointSpherical;

    i = get_global_id(0);

    tempPoint = convert_float3(origData(i));
    tempPointSpherical = convert_float3(tempPoint);

    tempFinalPoint = tempPoint;
    tempFinalPoint[0] += transData[0];
    tempFinalPoint[1] += transData[1];
    tempFinalPoint[2] += transData[2];
    tempFinalPoint[3] += transData[3];
    tempFinalPoint[4] += transData[4];

    outputData[i] = convert_int3(tempFinalPoint);
}
```
// Convert data points from Cartesian to Spherical Coordinate System
tempPointSpherical.s0 = hypot(tempPoint.x, tempPoint.y); // s0 refers to the radius component

tempPointSpherical.s1 = (tempPoint.x != 0) * atan2(tempPoint.y, tempPoint.x); // s1 refers to the theta (azimuth) component // Condition checks the possibility for infinity

tempPointSpherical.s2 = acos(tempPoint.z / tempPointSpherical.s0); // s2 refers to the phi (inclination) component

// Rotate the data points
tempPointSpherical.s1 += transData[3];
tempPointSpherical.s2 += transData[4];

// Convert data points from Spherical to Cartesian coordinate system

// Rotate the data points

// Translate the Points in X/Y/Z direction in Cartesian coordinate system

tempFinalPoint.x += transData[0];
tempFinalPoint.y += transData[1];
tempFinalPoint.z += transData[2];

// Copy data to the output array
outputData(i) = convert_float3(tempFinalPoint(i));
}

D.1.2 3D Data Point Translation - Floating Point Data

__kernel void TranslateFlat3D(__global float3 * origData, __constant float * transData, __global float3 * outputData) { // Float inputs
__local int i; // Stores the local id
i = get_global_id(0);

__local float3 tempPoint;
__local float3 tempFinalPoint;
tempPoint = origData(i);

// Store the spherical data point
__local float3 tempPointSpherical;

// Convert data points from Cartesian to Spherical Coordinate System
tempPointSpherical.s0 = hypot(tempPoint.x, tempPoint.y); // s0 refers to the radius component

tempPointSpherical.s1 = (tempPoint.x!=0)*atan2(tempPoint.y, tempPoint.x); // s1 refers to the theta (azimuth) component // Condition checks the possibility for infinity

tempPointSpherical.s2 = acos(tempPoint.z/tempPointSpherical.s0) // s2 refers to the phi (incliination) component

// Rotate the data points
tempPointSpherical.s1 += transData[3]; // Azimuth

tempPointSpherical.s2 += transData[4]; // Inclination

// Convert data points from Spherical to Cartesian coordinate system
tempFinalPoint.x =
    tempPointSpherical.s0*cos(tempPointSpherical.s1)*sin(tempPointSpherical.s2);
tempFinalPoint.y =
    tempPointSpherical.s0*sin(tempPointSpherical.s1)*sin(tempPointSpherical.s2);
tempFinalPoint.z = tempPointSpherical.s0*cos(tempPointSpherical.s2);

// Translate the Points in X/Y/Z direction in Cartesian coordinate system
tempFinalPoint.x += transData[0];
tempFinalPoint.y += transData[1];
tempFinalPoint.z += transData[2];

// Copy data to the output array
outputData(i) = tempFinalPoint(i);

// ------------------------ 2D Data Point Translation - Floating Point Data
------------------------//
__kernel void TranslateFlat2D(__global float2 *origData, __constant float *transData,
                   __global float2 *outputData) {
   __local int i;  // Stores the ID locally
   i = get_global_id(0);

...
__local float2 tempPoint;
__local float2 tempFinalPoint;
tempFinalPoint = origData(i);

// Store the spherical data point
__local float3 tempPointSpherical;

// Convert data points from Cartesian to Spherical Coordinate System
tempPointSpherical.s0 = hypot(tempPoint.x, tempPoint.y); // s0 refers to the radius component

// Condition checks the possibility for infinity

// Rotate the data points

// Convert data points from Spherical to Cartesian coordinate system

// Translate the Points in X/Y/Z direction in Cartesian coordinate system

// Copy data to the output array
outputData(i) = tempFinalPoint(i);
}

D.1.4 2D Data Point Translation - Floating Point Data - ONLY XY- NO ROTATION

// ------------------------ 2D Data Point Translation - Floating Point Data - ONLY XY - NO ROTATION ------------------------/

// transData should be an array of length 3 (X, Y, Z)
__kernel void TranslateFlat2Dxy(__global float2 *origData, __constant float *transData,
    __global float2 *outputData) {
    __local int i; // Stores the ID locally
    i = get_global_id(0);

    __local float2 tempPoint;
    __local float2 tempFinalPoint;
tempPoint = origData(i);

Appendix D. OpenCL Kernel Source Code

D.2 Data Scaling

D.2.1 2D Polar Scaling - Float

```c
// Float Version
// transData is of type float4 [MaxSensor, MinSensor, MaxActual, MinActual] and assumes
// that the origData is in polar form
__kernel void scale2DPolar(__global float2 *origData, __constant float4 *transData,
                          __global float2 *outputData) {
    // Stores the ID locally
    __local int i;
    i = get_global_id(0);
    __local float2 localData = origData[i];
    __local float2 scaledData;

    // Manipulates the radius in polar coordinate system
    // MAXactual - MINactual
    scaledData.y = (transData.w - transData.z);
    // MAXsensor - MINSensor
    scaledData.y /= (transData.y - transData.x);
    // Scale by sensor data
    scaledData.y *= origData.y;
    scaledData.x = origData.x

    outputData = scaledData;
```

D.2.2 2D Cartesian Scaling - Float
__kernel void scale2DRect(__global float2 *origData, __constant float4 *transData, __global float2 *outputData) {
    // Stores the ID locally
    __local int i;
    i = get_global_id(0);

    __local float2 localData = origData[i];
    __local float2 scaledData;

    // Convert the rectangular coordinates to Polar
    localData = pol2rect(localData);

    // Manipulates the radius in polar coordinate system
    // MAXactual - MINactual
    scaledData.y = (transData.w - transData.z);
    // MAXsensor - MINsensor
    scaledData.y /= (transData.y - transData.x);
    // Scale by sensor data
    scaledData.y *= origData.y;
    scaledData.x = origData.x;

    // Convert back to rectangular form
    outputData = rect2pol(scaledData);
}

D.2.3 2D Polar Scaling - Int

__kernel void scale2DPolar(__global int2 *origData, __constant int4 *transData, __global int2 *outputData) {
    // Stores the ID locally
    __local int i;
    i = get_global_id(0);

    __local int2 localData = origData[i];
    __local int2 scaledData;

    // Manipulates the radius in polar coordinate system
    // MAXactual - MINactual
    scaledData.y = (transData.w - transData.z);
    // MAXsensor - MINsensor
    scaledData.y /= (transData.y - transData.x);
    // Scale by sensor data
    scaledData.y *= origData.y;
D.2.4 2D Cartesian Scaling - Int

```c
__kernel void scale2DRect(__global int2 *origData, __constant int4 *transData, __global int2 *outputData) {
    // Stores the ID locally
    __local int i;
    i = get_global_id(0);

    __local float2 localData = (float2)origData[i];
    __local float2 scaledData;

    // Convert the rectangular coordinates to Polar
    localData = pol2rect(localData);

    // Manipulates the radius in polar coordinate system
    // MAXactual - MINactual
    scaledData.y = (transData.w-transData.z);
    // MAXsensor - MINSensor
    scaledData.y /=(transData.y-transData.x);
    // Scale by sensor data
    scaledData.y **origData.y;
    scaledData.x = origData.x

    // Convert back to rectangular form
    outputData = (int2)rect2pol(scaledData);
}
```

D.3 Differentiate

D.3.1 Differentiate 2D - Float
// Floating Point
__kernel void differentiate2D(__global float2 *origData, __constant int size, __global float2 *outputData) {
__local int i; // Stores the ID locally
i = get_global_id(0);
__local int workItems;
workItems = get_global_size(0);

// the number of items has to overlap to insure proper overlap of differentiation
int numItems = 1+(size/workItems);

float2 data[numItems];
float2 diffData[numItems-1];

// Input Data - Bring to local
for (int j = 0; j< numItems ; j++) {
  data[j] = origData[i*numItems+j];
}

// Differentiate
for (int j = 0; j< numItems - 1; j++) {
  diffData[j] = data[j+1]-data[j];
}

// Output data
for (int j = 0; j< numItems - 1; j++) {
  outputData[j] = diffData[j];
}
}

D.3.2 Differentiate 2D - Int

// Integer
__kernel void differentiate2D(__global int2 *origData, __constant int size, __global int2 *outputData) {
__local int i; // Stores the ID locally
i = get_global_id(0);
__local int workItems;
workItems = get_global_size(0);

// the number of items has to overlap to insure proper overlap of differentiation
int numItems = 1+(size/workItems);

int2 data[numItems];
int2 diffData[numItems-1];

// Input Data - Bring to local
for (int j = 0; j< numItems ; j++) {
  data[j] = origData[i*numItems+j];
}

// Differentiate
for (int j = 0; j< numItems - 1; j++) {
  diffData[j] = data[j+1]-data[j];
}

// Output data
for (int j = 0; j< numItems - 1; j++) {
  outputData[j] = diffData[j];
}
D.4 Lidar Depth Data to 3D Coordinates

D.4.1 Lidar RAW to Polar - Int

```c
__kernel void convertRaw2Polar(__global int *origData, __constant int3 rangeTheta,
                                __global int2 *outputData) {
    //rangeTheta = (minTheta, MaxTheta, array Size)
    //Get Local ID
    __local int i;
    i = get_global_id(0);
    //Retrieve Data
    __local int2 tempData;
```
tempData.y = origData[i];

// Calculate angle relative to center
__local int thetaStep;
thetaStep = (rangeTheta.y-rangeTheta.x)/rangeTheta.z;
tempData.x = rangeTheta.x + i*thetaStep;
outputData[i] = tempData;
}

D.4.2 Lidar RAW to Polar - Float

__kernel void convertRaw2PolarFloat(__global float * origData, __constant float2 rangeTheta, __constant int size, __global float2 * outputData) {
    // rangeTheta = (minTheta, MaxTheta)
    // Get Local ID
    __local int i;
    i = get_global_id(0);
    // Retrieve Data
    __local float2 tempData;
    tempData.y = origData[i];

    // Calculate angle relative to center
    __local float thetaStep;
    thetaStep = (rangeTheta.y-rangeTheta.x)/size;
    tempData.x = rangeTheta.x + i*thetaStep;
    outputData[i] = tempData;
}

D.4.3 Lidar RAW to Cartesian - Int

__kernel void convertRaw2Cart(__global int * origData, __constant int3 rangeTheta, __global int2 * outputData) {
    // rangeTheta = (minTheta, MaxTheta, array Size)
    // Get Local ID
    __local int i;
    i = get_global_id(0);
// Retrieve Data
__local int2 tempData;
tempData.y = origData[i];

// Calculate angle relative to center
__local int thetaStep;
thetaStep = (rangeTheta.y - rangeTheta.x)/rangeTheta.z;
tempData.x = rangeTheta.x + i*thetaStep;
outputData[i] = pol2cart(tempData);
}

D.4.4 Lidar RAW to Cartesian - Float

// Theta is float value
__kernel void convertRaw2CartFloat(__global float * origData, __constant float2 rangeTheta,
__constant int size, __global float2 * outputData) {
// rangeTheta = (minTheta, MaxTheta)

// Get Local ID
__local int i;
i = get_global_id(0);

// Retrieve Data
__local float2 tempData;
tempData.y = origData[i];

// Calculate angle relative to center
__local float thetaStep;
thetaStep = (rangeTheta.y - rangeTheta.x)/size;
tempData.x = rangeTheta.x + i*thetaStep;
outputData[i] = pol2cart(tempData);
}

D.5 Data Filtering

D.5.1 Low Pass Filter - Int
__kernel void lowPassFilter(__global int *origData, __local int *value, __global int *outputData) {
    // Stores the ID locally
    __local int i;
    i = get_global_id(0);

    // the number of items has to overlap to insure proper overlap of differentiation
    int numItems =1+(size/workItems);

    int data[numItems];
    int filtered[numItems];

    // Input Data - Bring to local
    for(int j = 0; j<numItems; j++) {
        data[j] = origData[i*numItems+j];
    }

    __local count;
    count = 0;
    for(int j = 0; j<numItems; j++) {
        if(data[j] < value) {
            filtered[count] = data[j];
            count++;
        }
    }
    for(int j = 0; j <count; j++) {
        // Push output data to output variable. Since the actual size of the returned value
        // is not returning an array the same size as the input array, and some values have
        // been eliminated, push_back() is used to send back the values. Values can be
        // returned in an order different than returned, as the return is not synchronized
        // with other processes.
        push_back(filtered[j], outputData);
    }
}

D.5.2 Low Pass Filter - Float
__kernel void lowPassFilter(__global float *origData, __local float *value, __global float *outputData)
{
    // Stores the ID locally
    __local int i;
    i = get_global_id(0);

    // The number of items has to overlap to insure proper overlap of differentiation
    int numItems = 1 + (size / workItems);

    float data[numItems];
    float filtered[numItems];

    // Input Data - Bring to local
    for (int j = 0; j < numItems; j++)
    {
        data[j] = origData[i * numItems + j];
    }

    __local count;
    count = 0;
    for (int j = 0; j < numItems; j++)
    {
        if (data[j] < value)
        {
            filtered[count] = data[j];
            count++;
        }
    }

    for (int j = 0; j < count; j++)
    {
        // Push output data to output variable. Since the actual size of the returned value
        // is not returning an array the same size as the input array, and some values have
        // been eliminated, push_back() is used to send back the values. Values can be
        // returned in an order different than returned, as the return is not synchronized
        // with other processes.
        push_back(filtered[j], outputData);
    }
}

---

D.5.3 High Pass Filter - Int

__kernel void lowPassFilter(__global int *origData, __local int *value, __global int *outputData)
Appendix D. OpenCL Kernel Source Code

D.5.4 High Pass Filter - Float

```c
__kernel void lowPassFilter(__global float *origData, __local float *value, __global float *outputData)
{
    // Stores the ID locally
    __local int i;
    i = get_global_id(0);

    // the number of items has to overlap to insure proper overlap of differentiation
    int numItems = 1 + (size / workItems);

    int data[numItems];
    int filtered[numItems];

    // Input Data - Bring to local
    for (int j = 0; j < numItems; j++)
    {
        data[j] = origData[i * numItems + j];
    }

    __local count;
    count = 0;
    for (int j = 0; j < numItems; j++)
    {
        if(data[j] > value)
        {
            filtered[count] = data[j];
            count++;
        }
    }

    for (int j = 0; j < count; j++)
    {
        // Push output data to output variable. Since the actual size of the returned value
        // is not returning an array the same size as the input array, and some values have
        // been eliminated, push_back() is used to send back the values. Values can be
        // returned in an order different than returned, as the return is not synchronized
        // with other processes.
        push_back(filtered[j], outputData);
    }
}
```
Appendix D. OpenCL Kernel Source Code

```
__local int i;
i = get_global_id(0);

// the number of items has to overlap to insure proper overlap of differentiation
int numItems = 1 + (size / workItems);

float data[numItems];
float filtered[numItems];

// Input Data - Bring to local
for (int j = 0; j < numItems; j++)
{
    data[j] = origData[i * numItems + j];
}

__local count;
count = 0;
for (int j = 0; j < numItems; j++)
{
    if (data[j] > value)
    {
        filtered[count] = data[j];
        count++;
    }
}
for (int j = 0; j < count; j++)
{
    // Push output data to output variable. Since the actual size of the returned value
    // is not returning an array the same size as the input array, and some values have
    // been eliminated, push_back() is used to send back the values. Values can be
    // returned in an order different than returned, as the return is not synchronized
    // with other processes.
    push_back(filtered[j], outputData);
}
```

---

D.5.5 Band Pass Filter - Int

```
__kernel void bandPassFilter(__global int *origData, __local int2 *value, __global int *outputData)
{
    // Stores the ID locally
    __local int i;
i = get_global_id(0);
```
// the number of items has to overlap to insure proper overlap of differentiation
int numItems = 1 + (size / workItems);

int data[numItems];
int filtered[numItems];

// Input Data - Bring to local
for (int j = 0; j < numItems; j++)
{
    data[j] = origData[i * numItems + j];
}

__local count;
count = 0;
for (int j = 0; j < numItems; j++)
{
    if (data[j] < value.x && data[j] > value.y)
    {
        filtered[count] = data[j];
        count++;
    }
}

for (int j = 0; j < count; j++)
{
    // Push output data to output variable. Since the actual size of the returned value
    // is not returning an array the same size as the input array, and some values have
    // been eliminated, push_back() is used to send back the values. Values can be
    // returned in an order different than returned, as the return is not synchronized
    // with other processes.
    push_back(filtered[j], outputData);
}

---

D.5.6 Band Pass Filter - Float

__kernel void lowPassFilter(__global float *origData, __local float2 *value, __global
float *outputData)
{
    // Stores the ID locally
    __local int i;
i = get_global_id(0);

    // the number of items has to overlap to insure proper overlap of differentiation
Appendix D. OpenCL Kernel Source Code

8 \text{int numItems =1+(size/workItems);} \\
9 \\
10 \text{float data[numItems];} \\
11 \text{float filtered[numItems];} \\
12 \\
13 //Input Data - Bring to local \\
14 \text{for(int j = 0; j< numItems; j++)} \\
15 \{ \\
16 \text{data[j] = origData[i*numItems+j];} \\
17 \} \\
18 \\
19 \_\_local count; \\
20 \text{count = 0;} \\
21 \text{for(int j = 0; j< numItems; j++)} \\
22 \{ \\
23 \text{if(data[j] < value.x \\&\& data[j] > value.y)} \\
24 \{ \\
25 \text{filtered[count] = data[j];} \\
26 \text{count++;} \\
27 \} \\
28 \} \\
29 \text{for(int j = 0; j <count; j++)} \\
30 \{ \\
31 // Push output data to output variable. Since the actual size of the returned value \\
32 // is not returning an array the same size as the input array, and some values have \\
33 // been eliminated, push_back() is used to send back the values. Values can be \\
34 // returned in an order different than returned, as the return is not synchronized \\
35 // with other processes. \\
36 \text{push_back( filtered[j], outputData);} \\
37 \} \\
38 \\

D.6 Gaussian Blur Filter

D.6.1 Gaussian 1D - Int

1 \_\_constant int gauss3[3] = \{1,2,1\}; \\
2 \_\_constant int gauss5[5] = \{1,4,6,4,1\}; \\
3 \_\_constant int gauss7[7] = \{1,6,15,20,15,6,1\}; \\
4 \_\_kernel void GaussianConvolution3(_\_global const int *A, _\_global int *B) \{ \\
5 \\
6 //Local offset load size
Appendix D. OpenCL Kernel Source Code

D.7 Divide and Conquer Summation

D.7.1 Summation 1D - Int

```c
__kernel void summation1D(__global int *origData, __global int *tempX, __global int *outputData) {
    //tempX allows for the global sharing of summation data
    //Stores the ID locally
    __local int i;
    i = get_global_id(0);
    //Stores the Global Size of the kernel execution
    __local int globalSize;
    globalSize = (int) get_global_size(0);

    __local uint half;
    half = globalSize;

    while(half!=0)
    {
        //barrier(CLK_GLOBAL_MEM_FENCE);
        if(i < half/2)
```
D.7.2 Summation 1D - Float

```c
__kernel void summation1D(__global float *origData, __global float *tempX, __global float *outputData)
{
    // tempX allows for the global sharing of summation data
    // Stores the ID locally
    __local int i;
    i = get_global_id(0);
    // Stores the Global Size of the kernel execution
    __local int globalSize;
    globalSize = (int) get_global_size(0);
    __local uint half;
    half = globalSize;
    while(half!=0)
    {
        // barrier(CLK_GLOBAL_MEM_FENCE);
        if(i < half/2)
        {
            tempX[i] = tempX[i]*tempX[i+(half/2)]*(half%2 && i==0)*tempX[half-1];
        }
        else
        {
            break;
        }
        half = half/2;
    }
}
```

// The index 0 thread returns the summation value
if (i == 0) {
    outputData.x = tempX[i];
}

D.7.3 Summation 2D - Int

__kernel void summation2D(__global int2 *origData, __global int2 *temp, __global int2 *outputData) {
    // temp allows for the global sharing of summation data
    // Stores the ID locally
    __local int i;
    i = get_global_id(0);
    // Stores the Global Size of the kernel execution
    __local int globalSize;
    globalSize = (int) get_global_size(0);

    __local uint half;
    half = globalSize;
    temp[i] = origData[i];
    while (half != 0) {
        // barrier(CLK_GLOBAL_MEM_FENCE);
        if (i < half / 2) {
            temp[i].x = temp[i].x + temp[i + (half / 2)].x + (half % 2 && i == 0) * temp[half - 1].x;
            temp[i].y = temp[i].y + temp[i + (half / 2)].y + (half % 2 && i == 0) * temp[half - 1].y;
        } else {
            break;
        }
        half = half / 2;
    }
    // The index 0 thread returns the summation value
    if (i == 0) {
        outputData = temp[i];
    }
}
D.7.4 Summation 2D - Float

```c
__kernel void summation2D(__global float2 *origData, __global float2 *temp, __global float2 *outputData) {
    //temp allows for the global sharing of summation data
    //Stores the ID locally
    __local int i;
    i = get_global_id(0);
    //Stores the Global Size of the kernel execution
    __local int globalSize;
    globalSize = (int) get_global_size(0);

    // local uint half;
    half = globalSize;
    temp[i] = origData[i];
    while(half!=0)
    {
        // barrier(CLK_GLOBAL_MEM_FENCE);
        if(i < half/2)
        {
            temp[i].x = temp[i].x+temp[i+(half/2)].x+(half%2 && i==0)*temp[half-1].x;
            temp[i].y = temp[i].y+temp[i+(half/2)].y+(half%2 && i==0)*temp[half-1].y;
        }
        else
        {
            break;
        }
        half = half/2;
    }
    //The index 0 thread returns the summation value
    if(i == 0)
    {
        outputData = temp[i];
    }
}
```

D.7.5 Summation 3D - Int

```c
__kernel void summation3D(__global int3 *origData, __global int3 *temp, __global int3 *outputData) {
    //temp allows for the global sharing of summation data
    //Stores the ID locally
    __local int i;
    i = get_global_id(0);
```
// Stores the Global Size of the kernel execution
__local int globalSize;
globalSize = (int) get_global_size(0);

__local uint half;
half = globalSize;
temp[i] = origData[i];
while(half!=0)
{
    // barrier (CLK_GLOBAL_MEM_FENCE);
    if(i < half/2)
    {
        temp[i].x = temp[i].x + temp[i+(half/2)].x + (half%2 && i==0)*temp[half-1].x;
        temp[i].y = temp[i].y + temp[i+(half/2)].y + (half%2 && i==0)*temp[half-1].y;
        temp[i].z = temp[i].z + temp[i+(half/2)].z + (half%2 && i==0)*temp[half-1].z;
    }
    else
    {
        break;
    }
    half = half/2;
}
// The index 0 thread returns the summation value
if(i == 0)
{
    outputData = temp[i];
}

---

D.7.6 Summation 3D - Float

__kernel void summation3D(__global float3 *origData, __global float3 *temp, __global float3 *outputData) {
    // temp allows for the global sharing of summation data
    __local int i;
i = get_global_id(0);
    // Stores the Global Size of the kernel execution
    __local int globalSize;
globalSize = (int) get_global_size(0);

    __local uint half;
    half = globalSize;
temp[i] = origData[i];

while (half != 0)
{
    // barrier(CLK_GLOBAL_MEM_FENCE);
    if (i < half / 2)
    {
        temp[i].x = temp[i].x * temp[i + (half / 2)].x * (half % 2 && i == 0) * temp[half - 1].x;
        temp[i].y = temp[i].y * temp[i + (half / 2)].y * (half % 2 && i == 0) * temp[half - 1].y;
        temp[i].z = temp[i].z * temp[i + (half / 2)].z * (half % 2 && i == 0) * temp[half - 1].z;
    }
    else
    {
        break;
    }
    half = half / 2;
}
// The index 0 thread returns the summation value
if (i == 0)
{
    outputData = temp[i];
}
Bibliography


NAME: Alexander Bussiere
PLACE OF BIRTH: Waterloo, ON, Canada
YEAR OF BIRTH: 1989
EDUCATION: University of Windsor, B.A.Sc., Windsor, ON 2011
             University of Windsor, M.A.Sc., Windsor, ON, 2014