MASTER

Accelerating a movie recommender system using VirtualCL on a heterogeneous GPU cluster
big data analysis using distributed accelerators

Bhatnagar, A.

Award date:
2015

Disclaimer
This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
Accelerating a Movie Recommender System
Using VirtualCL on a Heterogeneous GPU Cluster

Big Data Analysis Using Distributed Accelerators

ASHISH BHATNAGAR
Master’s Thesis

Supervisors:
Johnny Öberg
Francesco Robino
Roelof Pieters
Arash Pendari

Examiner:
Ingo Sanders

EIT ICT Labs Master School Embedded Systems Program
School of Information and Communication Technology
KTH Royal Institute of Technology
Stockholm, Sweden

TRITA-ICT-EX-2014:186
Abstract

Present day market offers a large number of movies which overwhelm people with choices. In order to quickly navigate through all the possible movies and find the interesting ones, the user can take advantage of recommender systems for movies. This thesis studies a movie recommender system which uses image processing and computer vision algorithms. The amount of time taken to analyze movies using these computation intensive algorithms is in the order of years. However, exploiting parallel nature of these algorithms using GPUs (Graphics Processing Unit) can help reduce the time many-folds.

The primary goal of the thesis is to build a heterogeneous GPU cluster and use it to accelerate the algorithms of the recommender system. The guidelines and steps to build a heterogeneous GPU cluster given in the thesis can be used by other organizations and researchers. Results indicate that the heterogeneous GPU cluster platform can accelerate algorithms of a movie recommender system up to 5 times. The secondary goal of this thesis is to investigate the benefits of using VirtualCL framework which enables remote access to the GPUs of the cluster. Remote access to the GPUs provides energy efficiency and ease of cluster management. Results show that VirtualCL framework provides remote GPU capability at the cost of degradation in performance. Therefore, VCL framework should be used just for application areas where performance can be traded off for physical portability and ease of management.
Acknowledgement

Foremost, I would like to thank my examiner and Associate Professor at KTH, Mr. Ingo Sanders for accepting my thesis proposal and giving me an opportunity to work on this project. I would like to express my sincere gratitude to Dr. Johnny Öberg and Francesco Robino for their continuous support of my study and research, for their patience, motivation, enthusiasm, and immense knowledge. Their guidance helped me in all the time of research and writing of this thesis.

I would like to thank Arash Pendari, CEO at Vionlabs AB, who not only provided this excellent opportunity but also gave continuous and never ending cooperation for the project. My sincere thanks also goes to Roelof Pieters whose guidance always helped me move in the correct direction. I appreciate my colleagues at Vionlabs AB who made the work environment friendly and productive.

My special thanks are extended to EIT ICT Labs Masterschool for having faith in me and helping me out throughout my master studies. I would like to express my appreciation to Ms. Ipshita Bhattacharya whose inputs made this thesis better and easier to read. Last but not the least, I thank my family and friends who supported and encouraged me throughout the studies.
Contents

1 Introduction 1
  1.1 Background ........................................... 1
  1.2 Problem Description .................................... 2
    1.2.1 Problem ........................................... 2
    1.2.2 Problem Statement ................................ 2
  1.3 Purpose ................................................. 2
  1.4 Goals .................................................. 2
  1.5 Method ................................................ 2
  1.6 Boundaries ............................................. 3
  1.7 Outline ................................................. 3

2 Literature Review 5
  2.1 Recommender Systems .................................... 5
  2.2 Computing Platforms ................................... 6
  2.3 Development Environment ................................. 7
    2.3.1 Parallel Programming Model ........................ 8
    2.3.2 Available Library Support ........................... 10
  2.4 Remote GPU Framework .................................. 10
  2.5 Application Characteristics ............................... 11
  2.6 Optimizations ........................................... 13
    2.6.1 GPU Based Optimizations ............................. 13
    2.6.2 Application Based Optimizations ..................... 14

3 Methodology 17
  3.1 Prototyping ............................................. 17
    3.1.1 Prototyping Technique ............................... 18
    3.1.2 Advantages of Prototyping ........................... 18
    3.1.3 Disadvantages of Prototyping ....................... 19
### CONTENTS

3.1.4 Prototyping Model Selection ........................................ 19
3.2 Test Driven Development ............................................. 21
3.3 Quality Maintenance and Assurance ................................. 21

4 Implementation .................................................................. 23
  4.1 Gathering Requirements .............................................. 23
  4.2 System Architecture .................................................. 24
  4.3 Components of the Architecture ................................. 26
    4.3.1 Selected GPU ................................................. 26
    4.3.2 Selected Parallel Programming Model ..................... 26
    4.3.3 Specifications of Selected Devices ......................... 28
  4.4 Application Selection ............................................... 29
    4.4.1 Application 1 .............................................. 29
    4.4.2 Application 2 .............................................. 31
    4.4.3 Optimizations Implemented ................................. 31
  4.5 Implementation of System Architecture ....................... 33
    4.5.1 Single GPU Implementation ................................ 33
    4.5.2 Multiple GPU Implementation .............................. 34
    4.5.3 Remote Access Implementation ............................. 35

5 Results and Analysis .................................................... 39
  5.1 Evaluation Procedure ................................................ 39
  5.2 Results ..................................................................... 40
  5.3 Analysis ..................................................................... 41

6 Conclusions and Future Work ........................................ 47
  6.1 Future Work .......................................................... 48

Bibliography ....................................................................... 49

Appendices ......................................................................... 52

A Abbreviations ................................................................ 53
List of Figures

2.1 Types of Recommender Systems ........................................ 6
2.2 Architecture Differences Between GPU and CPU ..................... 6
2.3 Remote GPU Framework using VirtualCL .............................. 11

3.1 The Waterfall Model ....................................................... 20
3.2 The Spiral Model .......................................................... 20

4.1 System Architecture ...................................................... 25
4.2 Block Diagram of System Architecture ................................. 25
4.3 GPU Performance Benchmarks ............................................ 27
4.4 Specifications of NVIDIA GTX 770 GPU .............................. 27
4.5 Flowcharts of the Applications ........................................... 30
4.6 Block Diagram After Single GPU Implementation .................... 34
4.7 NVIDIA GTX 770 GPU .................................................. 34
4.8 Block Diagram After Multiple GPU Implementation ................. 35
4.9 Open Air Multiple GPU Rig .............................................. 36
4.10 Block Diagram After Remote Access Implementation ............... 36
4.11 GPU Cluster System ..................................................... 37

5.1 Results of Application 1 .................................................. 40
5.2 Results of Application 2 .................................................. 41
5.3 Movie Analysis using application 1 ..................................... 42
5.4 Movie Analysis using application 2 ..................................... 43
5.5 Acceleration for the movies .............................................. 43
5.6 VCL overhead analysis based on size of data to be transferred .... 44
5.7 VCL overhead analysis based on number of frames to be transferred 44
# List of Tables

2.1 Comparison between CUDA and OpenCL ......................................... 8  
2.1 Comparison between CUDA and OpenCL ......................................... 9  
2.2 Characteristics to predict benefits of porting an application to GPU architecture .................................................. 11  
2.2 Characteristics to predict benefits of porting an application to GPU architecture .................................................. 12  
2.3 GPU based optimizations ............................................................. 13  
2.3 GPU based optimizations ............................................................. 14  
2.4 Application based optimizations .................................................. 14  
2.4 Application based optimizations .................................................. 15  
4.1 Optimizations Implemented in the Applications .............................. 32
Chapter 1

Introduction

1.1 Background

In everyday life, people rely on recommendations from other people by means of spoken words, reference letters, news reports from news media, general surveys, travel guides, and so forth. Recent explosion in web based services such as online shopping sites and movie sharing sites has enabled users to obtain a wide variety of products and information. With an increasing consumerism fuelled by the emergence of the web based services, it is getting difficult for users to search only for necessary information or products that best match their preferences [1]. The buyers are being presented with an increasing range of choices while sellers are being faced with the challenge of increasing the efficiency of their advertising efforts [2].

Recent past has seen a tremendous growth of recommender systems which are powerful tools for extracting additional value for a business from its user databases [3]. Recommender Systems have evolved to fulfil the natural dual need of buyers and sellers by automating the generation of recommendations based on data analysis. They assist and augment the natural social process to help people find the most interesting and valuable information for them [4]. Conversely, they help the business by generating more sales.

With a rapid increase in their popularity, they are becoming a crucial tool on the web. However, it is important for a successful recommender engine to use techniques which generate trustworthy suggestions in order to compliment the decision making process of human beings. Recommender systems being used by Netflix and Amazon are examples of the most powerful and successful engines. Their success and resulting increase in revenue has caused the inception of dozens of other recommender engines. As a result, the architecture of recommender systems has become one of the most active areas of research and development.

The movie recommender system under study is being built by Vionlabs AB, which is a small start-up company in Stockholm. It is being stressed by the huge volume of data to be analysed, and will be stressed even more because the amount of data is bound to increase in the future. A state of the art computer system
takes approx. a day to analyse one of the 700,000 available mainstream movies. As a result, a start-up company which can not afford to buy thousands of computer systems will end up spending years in just analysing the movies.

1.2 Problem Description

1.2.1 Problem

The movie recommender system under study uses algorithms from computer vision, natural language processing and machine learning to analyse movies. A distinguishing feature of applications from these research areas is that they involve high computing complexity and plenty of parallelism. This problem structure naturally matches with the parallel programming model of GPU (Graphics Processing Unit) architecture [5], [6]. GPUs can exploit this feature and provide tremendous acceleration to the recommender engine. Thus, using a GPU cluster to accelerate the recommender systems presents a more elegant and affordable solution compared to traditional CPU - GPU systems.

1.2.2 Problem Statement

How can we build and use a cost effective GPU cluster to accelerate the movie recommender system? How can we optimize applications for recommending movies and compare the acceleration provided by the GPU cluster to the traditional computer systems?

1.3 Purpose

The purpose with the thesis is to provide guidelines for building an affordable GPU cluster. The thesis describes steps taken to port, optimize and run different movie recommending algorithms on the created system. It explains the steps taken to compare the performance of the GPU cluster to traditional computer systems.

1.4 Goals

The goal of the work is to build a GPU cluster accelerator for a movie recommender system and to test its performance against existing computer systems. The guidelines for creating the system can be used by individual researchers and small businesses to efficiently run their computation intensive algorithms.

1.5 Method

This research uses quantitative research methods because the performance of the GPU cluster system is measured and compared using experiments and tests. Quali-
tative methods of research are irrelevant to this study because this research doesn’t involve people’s opinions to obtain the results. The results obtained are independent of the observer and the instruments. Therefore, the research utilizes philosophical assumptions which belong to objective category (positivism). In order to develop the product into a functional product, Boehm’s spiral model is used to create the prototypes. In addition to it, concepts from test driven development are used to test and analyze the performance of the system.

1.6 Boundaries

In order to solve this real world problem, it is important to take some practicalities into consideration. Vionlabs AB is a small sized business looking for a method to accelerate their algorithms. They have limited man power and it is essential for them to have a method which is easy to implement and manage. Algorithm optimization needs time and human resources are scarce for them. Therefore, this research emphasizes on a practical implementation of the GPU cluster and doesn’t focus only on algorithm optimization.

1.7 Outline

The research provides guidelines for the selection of parallel programming framework for a wide range of applications. These guidelines are helpful for other organizations, researchers and individuals who want to use CPU-GPU systems to accelerate their applications. The organization of the thesis is given below. Chapters 2 and 3 present a theoretical background through a comprehensive literature review and a study of related work conducted in the field of heterogeneous computing. Chapter 4 describes the research methods and discusses the design decisions taken for the GPU cluster system. Chapter 5 describes the implementation steps taken during the study. It presents the selected algorithms and the steps taken to create the GPU cluster. Chapter 6 presents the results obtained and their detailed analysis. The thesis ends with chapter 7 containing conclusion and future work.
Chapter 2

Literature Review

This chapter presents an overview of recent developments in the field of recommender systems. It describes different components of the GPU cluster which are essential to maximize its performance. The chapter presents features and characteristics of applications which help in exploiting the parallel architecture of GPUs and discusses optimizations identified in literature which yield high performance improvements.

2.1 Recommender Systems

The traditional information filtering system consisted of various search algorithms which group items together, based on a pre-defined trait or property. Recent advancements in fields like machine learning and computer vision have lead to the development of recommender systems which find interesting properties in products and group them together. Recommender systems have become an active area of research and development following the success of these systems made by Amazon and Netflix. However, use of GPUs to accelerate recommender systems is a relatively new field. The authors in [7], have obtained positive experimental results. They show that their proposed parallel GPU implementations outperformed the serial CPU implementations and the serial GPU implementation.

Recommender systems can be classified not only based on the products they analyse, but also based on the way they analyse data sources. Figure 2.1 shows the types of recommender systems based on the way data sources are analyzed.

- Collaborative recommender systems use historical interactions of users whose interests are similar to those of the given user. Currently, they are the main-stream form of recommending technology.

- Content based recommender systems try to identify interesting items based on features of items in the user profile.

- Hybrid recommender systems attempt to combine both of these designs [8], [4].
The recommender system under study at Vionlabs AB is a hybrid engine which will start as a content based analysis system and slowly increase collaborative inputs when users generate their ratings. The content based analysis of movies is a cluster of application areas like computer vision, natural language processing and machine learning. These application areas have plenty of data level parallelism i.e., data can be processed independently and in any order on different computing elements.

2.2 Computing Platforms

Most of the applications use two major platforms for high performance computing. The first one is the CPU (central processing unit) which was traditionally used to run serial applications and has recently added multiple cores (shown in figure 2.2) to process data in parallel. The second one is the GPU (graphics processing unit) which was used for graphics computation on a lot of small cores. Recently GPUs have added the possibility of being used for general purpose computing which led to the rise of the GPGPU (General Purpose Graphics Processing unit) field. These two platforms are compared below.

Single-core processors have shown an exponential growth in their performance in the past few decades. This exponential growth ended in 2004, mainly limited by the power wall [9]. The mainstream CPUs today can pack only a small number of processing cores (as shown in figure 2.2) to stay within the energy limits.
The CPU core is ideally designed for serial applications. In order to make use of instruction level parallelism present in serial applications, the CPU uses out of order super scalar architecture. The frequently used data is stored in caches and the programmer doesn’t have to deal with the memory management \[5\]. In order to improve performance and reduce the impact of branch diversity, CPUs utilize a branch predictor \[5\], \[10\].

Authors in \[11\] state that increasing parallelism has become the primary source of increasing performance. GPUs provide a better platform for high performance computing applications because they provide a large number of small cores. Each GPU core is relatively simple which makes it possible to pack more cores on a single chip. GPUs provide hardware support for various operations which consume a lot of time on the CPU. These special function units provide tremendous acceleration to high performance computing applications \[5\].

The above discussion clarifies that both the computing platforms have their own advantages and disadvantages. In order to maximize gain, it is important that the best solution is chosen according the chosen application field. CPU platform should be used if the application contains a limited amount of parallelism. We notice that recommender system applications from computer vision and machine learning fields contain large amount of data level parallelism and require high computing power. Therefore, GPU architecture is more suitable to provide acceleration to the movie recommender system under study.

An argument can be made in favour of using a mixed CPU/GPU system instead of using a single CPU with a GPU cluster. In such a system, every GPU will be run using a CPU. The serial part of an application will be run using the CPU core and the parallel part of an application will be run using the GPU cores. A mixed CPU/GPU system would give a higher performance for algorithms where the serial portion dominates the parallel portion of the application. For an application which is highly parallel, a mixed CPU/GPU system will create a very small impact. In addition to the small improvement in performance, all the extra CPUs will result in additional cost because today’s high end CPUs are very expensive. Moreover, the additional CPUs will increase the energy consumption by the system. Since the recommender system deals with highly parallel applications, it was decided to use a single CPU with a GPU cluster.

\[2.3\] Development Environment

GPUs have an immense amount of computation power. However, tapping into the potential is not very easy. Different performance requirements and factors such as energy usage, cost of implementation, etc. influence their selection. Today, there is a plethora of GPUs available from many vendors which differ in their performance benchmarks and are available at a variety of costs. The performance of the GPU based applications depends on the development environment selection. The key parameters of the development framework and their selection criteria are described
There exist many parallel programming models such as CUDA, OpenCL, Direct-Compute, HMPP (Hybrid Multicore Parallel Programming), etc. for programming GPUs [12], [13], [14], [15]. Among these, both CUDA and OpenCL are developed using C/C++ like API’s and they can be easily integrated with the C/C++ code. Since they are the programming models with maximum support for GPGPU computing, they are discussed and compared below.

**CUDA v/s OpenCL**

There have been multiple studies comparing the performances of CUDA and OpenCL. These studies have performed elaborate tests on various parts of an algorithm such as data transfer, computation etc. Results from the prominent ones are mentioned in the table 2.1.

<table>
<thead>
<tr>
<th>Property</th>
<th>CUDA</th>
<th>OpenCL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model Based Differences</td>
<td>Available only for NVIDIA GPUs [16].</td>
<td>Available for all major GPUs and accelerators [16]</td>
</tr>
<tr>
<td></td>
<td>Kernel code is statically compiled and doesn’t add to execution time [16].</td>
<td>Kernel code is dynamically compiled and adds to execution time. Time consuming compiler optimizations may increase total execution time [16], [17].</td>
</tr>
<tr>
<td>Performance Based Differences</td>
<td>CUDA performs better than OpenCL [11], [18].</td>
<td>OpenCL can achieve similar performance to CUDA [11].</td>
</tr>
<tr>
<td></td>
<td>CUDA C compilers are already mature.</td>
<td>Once OpenCL compilers have matured, OpenCL kernels can deliver sustained performance comparable to CUDA kernels[16].</td>
</tr>
<tr>
<td></td>
<td>CUDA doesn’t work for Radeon GPUs</td>
<td>the automatic optimizations by the OpenCL C compiler do not work for Radeon GPUs [16].</td>
</tr>
</tbody>
</table>

Table 2.1: Comparison between CUDA and OpenCL
2.3. DEVELOPMENT ENVIRONMENT

Property | CUDA | OpenCL
--- | --- | ---
Portability Based Differences | CUDA doesn’t provide portability | OpenCL offers portability [11], [16], [19].
 | CUDA is vendor dependent. | OpenCL uses vendor independent terms and provides user options to provide portability [11].

Table 2.1: Comparison between CUDA and OpenCL

Performance v/s Portability

OpenCL claims to provide application portability between parallel processors, but authors in [11], [18] claim that portability is obtained at the cost of decreased performance. The authors in [19] find that CUDA is a better choice for applications when achieving highest possible performance is the main target. In [11] and [20], authors conclude that OpenCL is a very useful tool which enables portability without causing an alarming dip in the performance. These studies indicate that if performance is not the paramount requirement, choice between CUDA and OpenCL can be made based on other factors such as cost and portability.

In [17], [19] and [20], the authors state that functional portability without performance portability is not important because it results in improper usage of the accelerator architecture. The optimal hardware settings for one GPU’s architecture can be far from optimal for another GPU’s architecture. Several papers suggest the use of ‘auto-tuning’ which is a technique to find optimal settings for different GPU architectures [16], [21]. Authors in [22] and [23] have reported benefits of using auto-tuning methods.

In [24], the authors demonstrate the portability through OpenCL without significant performance losses. The authors state that in many cases, users accept reduced efficiency for a gain in code simplicity and maintenance. The difference in performance between a serial application and an OpenCL implementation is dependent upon the extent to which the serial application’s code was optimised for the parallel architecture. Porting an application to the parallel architecture results in high development cost and efforts which increase when the code needs to be highly optimized for performance on a single architecture. In [20], authors present a simple methodology for achieving acceptable levels of performance portability across different architectures.

The discussion on performance versus portability indicates a trade-off and the final decision needs to be taken based on the application characteristics and other factors like time, money and code management. Unlike the old times, today these factors play an important role when deciding to go for portability over performance.
2.3.2 Available Library Support

Recent times have seen a growth in various libraries coming up for GPGPU programming. There are many other 3rd party wrappers for Python, Java, MATLAB etc. Wrappers like pyCUDA and pyOpenCL extend the benefits of using a scripting language like Python to GPGPU Computing. In addition, there are many libraries which provide support to GPU Computing in specific scientific application areas like image processing, computer vision. ImageMagick, OpenCV, Theano are some of the libraries which offer GPU acceleration support.

2.4 Remote GPU Framework

Lately, GPUs (Graphics Processing Units) are not only in use for graphics applications, but also for non-graphics applications, so-called GPU computing or GPGPU Computing [16]. Programming of GPGPUs is becoming increasingly accessible, leading to an increased use of GPUs to accelerate parallel computations such as linear algebra, image processing, finance, molecular dynamics, and graph traversal [9]. Currently, applications that utilize GPU devices run their device-executable code on local devices in their respective hosting-nodes [25]. Users are provided with software development and programming environments that can ease the use of OpenCL devices on a single node, but were not designed to run applications on clusters [26].

As a result of recent advancements, large-scale GPU clusters are gaining popularity in the scientific computing community. Authors in [10] have presented a generalized GPGPU cloud system built on Kepler GPGPUs. However, their deployment and production use are associated with a number of new challenges such as balanced cluster architecture, resource sharing in a cluster environment, programming models, and applications for GPU clusters [27].

Authors in [26] discussed VirtualCL (VCL) cluster platform, which is a wrapper for OpenCL that allows most unmodified applications to transparently utilize many OpenCL devices in a cluster as if all the devices are on the local computer. This is illustrated in figure 2.3 where, the host node accesses OpenCL devices present on the compute node. VCL benefits applications that can use many devices concurrently. The VirtualCL (VCL) cluster platform can run most unmodified OpenCL applications transparently on clusters with many devices, including CPUs, GPUs and Accelerator devices of all vendors. VCL allows programs to run on a cluster without having to be split, by providing the impression of a single host with many devices. Users can start a parallel application on a hosting computer, then VCL manages and transparently runs the kernels of the application on different nodes [26]. In this manner, VCL provides remote access to GPUs of a heterogeneous cluster for all CPUs present in the same network.

In order to provide GPGPU virtualization, major alternatives to VCL framework included rCUDA [28], Zillians vGPU [29], StarPU [30] and SnuCL [31]. As discussed earlier, there was a possibility of using GPUs from various vendors and OpenCL was chosen as the parallel programming model. As a result, the biggest
competitors to VCL framework, such as rCUDA and Zillians vGPU were rejected because they offered the virtualization support for CUDA. Zillians vGPU offered support to infiniband, which was a very expensive alternative to TCP/IP. Among the virtualization options for OpenCL, SnuCL, StarPU and VCL framework were the major candidates for selection. It was decided to use VCL framework because it offered utilization of multiple GPUs in a cluster with zero/minimal changes to the application code.

### 2.5 Application Characteristics

The characteristics of applications that result in a tremendous performance improvement on porting the application to GPUs are given in table 2.2.

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Floating point computation to memory access ratio</td>
<td>A single local or global memory access is costly in terms of clock cycles needed to complete the process. Floating point instructions can be run in parallel to the memory accesses to hide the latency. Authors in [6] suggested the ratio of floating-point instructions and global memory access to be used as one of the characteristics to predict the benefits of porting the application to the GPU architecture.</td>
</tr>
</tbody>
</table>

Table 2.2: Characteristics to predict benefits of porting an application to GPU architecture
<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared memory access to global memory access ratio</td>
<td>An application which has higher shared memory to global memory access ratio, results in better performance after porting.</td>
</tr>
<tr>
<td>Per pixel floating point computation</td>
<td>In [6], the authors state that modern GPUs perform 20 times faster floating point operations compared to CPUs because of specialized hardware. Therefore, an application with a higher percentage of floating point computation will yield more benefits from porting.</td>
</tr>
<tr>
<td>Per pixel memory access</td>
<td>Current GPUs use data parallelism to achieve more operations per unit time and have 10 times higher memory bandwidth than CPUs. An application having more memory accesses per pixel will achieve more acceleration after porting [6].</td>
</tr>
<tr>
<td>Parallel fraction</td>
<td>Amdahl’s law states that code acceleration using parallel processors is limited by the sequential fraction of the program. It implies that the acceleration of an application on GPUs is always limited by the percentage of the sequential section [5]. Thus, an application which has a smaller sequential fraction can benefit the most from porting to a parallel architecture.</td>
</tr>
<tr>
<td>Branch diversity</td>
<td>In order to extract maximum efficiency from a GPU architecture, control statements like if, switch, while etc. should be reduced and if unavoidable, they should be brought to the outer most loops [6]. An application with lower number of control statements is bound to provide a higher acceleration after porting.</td>
</tr>
<tr>
<td>Task dependency</td>
<td>The number of global barrier synchronizations required by an application contribute to task dependency. An application that has a lower amount of task dependency results in better performance after porting.</td>
</tr>
<tr>
<td>Global memory transfer</td>
<td>The image data transfer rate between CPUs and GPUs is a significant bottleneck for applications having a large memory requirement and low floating point computation [6].</td>
</tr>
</tbody>
</table>

Table 2.2: Characteristics to predict benefits of porting an application to GPU architecture
2.6 Optimizations

Apart from the inherent beneficial characteristics mentioned above, there are other methods in which an application can be accelerated. The application can use the optimizations mentioned below in order to obtain more acceleration. These optimizations are divided into 2 main categories: GPU based optimizations and Application based optimizations. GPU based optimizations depend on the specifications of the GPU being used to run the ported applications. These specifications help in identifying the most suitable GPU. Application based optimizations refer to coding techniques which help in increasing the execution speed.

2.6.1 GPU Based Optimizations

The list of GPU based optimizations is presented in the table 2.3.

<table>
<thead>
<tr>
<th>Optimization</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bandwidth</td>
<td>Bandwidth affects kernel performance if either the kernel can hide latency of memory accesses or the kernel can fit the required data on memories like cache and buffers [5]. A GPU with higher bandwidth usually gives a better performance.</td>
</tr>
<tr>
<td>Compute flops</td>
<td>Compute flops can be increased by using more cores and by using higher SIMD width. Scalable performance can be achieved by increasing compute flops and bandwidth [5].</td>
</tr>
<tr>
<td>Work group size</td>
<td>Authors in [20] and [16] found that an optimal work group size affects the performance of an application. A small work-group size reduces the number of active threads and a large work group size produces a lack of hardware resources.</td>
</tr>
<tr>
<td>Cache</td>
<td>A GPU with a higher cache size can fit the data required by different applications in the available cache. This results in compute bound kernels which make scalable performance possible [5].</td>
</tr>
<tr>
<td>Reduction and synchronization</td>
<td>Reduction and synchronization operations obstruct the parallel flow of applications as they do not allow data parallelism. A GPU that has a smaller synchronization overhead is better suited for higher performance.</td>
</tr>
</tbody>
</table>

Table 2.3: GPU based optimizations
Optimization | Explanation
---|---
Gather/Scatter | High performance can be achieved by address aligning the data to operate on. Software operation to achieve this is quite expensive. GPUs having the hardware support for the same can yield a better performance [5].
New features | Authors in [11] mention that the latest GPU architectures offer features which provide more options for optimization and significantly impact the GPU’s performance.
Fixed function units | GPUs provide specialized fixed function units which can be used to accelerate applications. Authors in [5] predict that future CPUs and GPUs will have many such units accelerating applications.

Table 2.3: GPU based optimizations

### 2.6.2 Application Based Optimizations

In [19], the authors correctly noted that the effect of various optimizations varies from architecture to architecture. They state that accurate exploration of the parameter space is essential to maximize performance. Therefore, it is important that optimizations are chosen carefully which provide benefit on all platforms. Some of those optimizations are discussed in table 2.4.

<table>
<thead>
<tr>
<th>Optimization</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiler optimizations</td>
<td>Compiler optimization flags provide the easiest way to obtain optimized code. Examples of some of the flags are '-cl-fast-relaxed-math', '-cl-mad-enable' and '-cl-no-signed-zeros'. Authors in [18] and [16] stated that turning on the mathematical optimization flags makes code noticeably more effective than without optimization.</td>
</tr>
<tr>
<td>Placement of branches</td>
<td>In order to reduce the effect of branching, the flow control statements should be moved up the pipeline for efficient evaluation. Placement of branches inside inner loops degrades performance and should be avoided [32].</td>
</tr>
</tbody>
</table>

Table 2.4: Application based optimizations
Optimization | Explanation
---|---
Pre-computation | Pre-computation of a branch can improve the performance of an application when the branch result doesn’t change for a while. Evaluating the branches only when necessary results in a large performance boost [32].
Floating point precision | Double precision computation should be avoided when high accuracy is not required because it is twice as slow as single precision computation [20].
Using memory layout | Applications where different cores operate on data in separate memories outperform those where cores operate on data in same memory space. An application which exploits spatial and temporal locality of the cache line provides better performance [20].
Coalesced global memory accesses | Authors in [11] and [20] have identified manifold benefits of this optimization. This optimization results in smaller number of larger communication messages that exploit the network bandwidth well.

Table 2.4: Application based optimizations
Chapter 3

Methodology

This research uses quantitative research methods since the performance of the CPU-GPU systems is measured and compared using concrete experiments and test applications. The philosophical assumptions behind the research belong to the objective (positivism) category because the results obtained from the research do not depend on the observer or the instruments of measurement. In order to develop the working GPU cluster, concepts from prototyping and test driven development are used. These concepts help in efficient development process and minimized errors. They are discussed in detail in this chapter.

3.1 Prototyping

Traditionally, the user involvement in the development process was limited to the start and the end of the project. This conventional approach to system design caused the customer to be passive and the developer to be active. This approach often lead to systems which did not meet the customer’s requirements. In [33], the authors identified reasons which render the customer unable to define the system. These reasons are:

- the problem may be too complex and poorly understood by the end-users.
- the end-users may be unable to communicate it properly to analysts.
- the problem may change while the system is being developed.
- the problem may be unstructured, defying clear specification.
- the system may have unclear and volatile system requirements.

In order to remove these problems, it is essential that better methods are found, both of obtaining information from the user and understanding the real requirements. The method should make the customer more active and developer more passive, until they work together towards a mutually acceptable system [34].
Prototyping is an aid to build the right product by the simple process of having more than one go at it and learning from the mistakes [35]. It is a development method to reduce the problems in traditional product development approaches [36], [37]. The objective of creating prototypes is to ensure the efficient development of a real system fulfilling all system requirements [34]. It is in some sense an approximation to the final product.

3.1.1 Prototyping Technique
Naumann and Jenkins identified prototyping as a four stage methodology consisting of:

- identifying the user’s basic information requirements
- developing a working prototype system
- implementing and using the prototype system
- revising and enhancing the prototype system

3.1.2 Advantages of Prototyping
In literature, prototyping has been widely acclaimed as an effective approach to requirements definition and systems development. The advantages of prototyping are as follows:

- **Bridging the Communication Gap**
  The most important benefit of using prototypes is the reduction in the communication gap between the customer and the developer. During the process of building a prototype, the customer is engaged in all the cycles of system development. This exposure provides the customer an insight into usability of the system and improves the acceptance rates of the prototype. Authors of [34] and [38] suggest that even a crude prototype is enough to establish and confirm a continuous communication between the developer and the customer.

- **Early Evaluation and Problem Detection**
  Literature has identified that cost of detecting and fixing a problem is much less earlier in the development cycle. The major challenges for the system developers are to capture the complete set of product specifications and to detect the points of failure. The authors in [36], [39], [40], [41] and [38] identify prototyping as one of the most efficient means to arrive at product specification and to detect failure possibilities.

- **Fast and Efficient Development**
  Although using a prototype consumes some time initially but in the long run, building and utilizing prototypes paves way for quicker and more efficient system development. Prototyping can reduce project backlog [34].
3.1. PROTOTYPING

- **Cost Effectiveness and Flexibility**
  Prototypes facilitate design improvement and device failure detection early in the design process [40]. Such a system leads to a reduction in the long term costs. Flexibility provided by the prototypes helps inexperienced users and developers to obtain an appropriate level of specification detail [38]. This flexibility enables prototypes to provide higher quality information and greater value [33].

3.1.3 Disadvantages of Prototyping

There are a few disadvantages associated with the development of prototypes which lie in following categories.

- **Time, Effort and Cost**
  Building a prototype takes time and effort of the development team. A prototype that is inappropriate or untimely could hamper the progress of the development team at the expense of their considerable effort [34].

- **User Related Problems**
  It is possible for an inexperienced user to confuse a prototype with the real system [35]. They may get latched on to faults of a simple prototype and use them as reasons against the proposed system [34].

3.1.4 Prototyping Model Selection

Prototyping models have been an existing area of extensive research in the past. One of the main development model is the waterfall model which is shown in figure 3.1. The waterfall model consists of the following steps which occur in the serial order and deal with a single activity.

- gather requirements from the user
- quickly decide the design
- build the prototype
- evaluate and refine requirements models.

The waterfall model has many disadvantages. It is unrealistic to expect that the development steps will take place in a serial order. Moreover, the testing phase in waterfall model is encountered at the end of the cycle. If changes are required at a later stage, they may result in major rework and may render the prototype useless.

These drawbacks have resulted in Boehm’s spiral model which is shown in figure 3.2. Boehm’s spiral model for development is more realistic and comprehensive [35]. A single cycle in Boehm’s model constitutes of all the steps in the waterfall model. This cycle is inexpensive and fast and it may be iterated a number of times [34]. The complete product development consists of many iterations of this cycle. Every
Figure 3.1: The Waterfall Model

iteration is carried out starting from the end product of the previous cycle. In this manner, Boehm’s model enables an incremental development of products and overcomes the drawbacks of the waterfall model.
3.2 Test Driven Development

Test Driven Development (TDD) is a software development practice in which unit test cases are incrementally written prior to code implementation [42]. Test driven development consists of smaller but repetitive development cycles in which the development happens incrementally. In a TDD iteration, a test case is developed before developing the system. After the system creation, when it satisfies the test cases, the next iteration is followed for developing the system. Following TDD practices results in improved and faster development. It helps in detecting errors early in the development process. Since TDD follows smaller cycles like Boehm’s spiral model, it was decided to combine the two methods for the development.

3.3 Quality Maintenance and Assurance

During the project, there was a continuous focus to maintain the quality of the research. Therefore, the research was conducted ethically and results of the research are presented truthfully to the best of the knowledge. In order to make the results reliable and dependable, all the tests were run 5 times and an average value of the results are used to draw the conclusions. Demonstration and evaluation of various prototypes was carried out at each iteration. This report covers all the aspects of the implementation and presents all the steps taken while doing the research. The implementation steps can be reused because the research is replicable and does not depend on the instrument or the researcher. The validity of the results was verified by following the test driven development where, the results were validated against already formed test cases.
Chapter 4

Implementation

This chapter presents the architecture of the GPU cluster system that was built to run the applications of the movie recommender system. The specifications of all the components of the cluster system are presented next. The applications and optimizations used to test the performance of the GPU cluster system are presented next. The steps taken to construct and implement the GPU cluster are presented at the end of the chapter. All the prototypes created in order to create the final system are presented here.

4.1 Gathering Requirements

The implementation of the project was started by a continuous interaction with Vionlabs personnel which helped in gathering a comprehensive set of their needs. These needs were translated to the following requirements which formed the basis of the design decisions taken during the remaining work.

- **Cost** One of the main requirements to be satisfied was that the development cost should remain small. As a startup company, Vionlabs AB was interested in receiving the maximum output from their investments. Choosing a high end GPU could deliver the best performance but its cost was a lot more than a mid level GPU. The price difference in the market was so high that it was possible to get better performance by buying multiple lower end GPUs at the cost of a single high end GPU. Therefore the challenge for this requirement was to choose the most cost effective GPU.

- **Performance and Portability** Although performance was the primary objective of using GPGPUs, portability was also an important aspect which needed to be looked at. Advancements in GPU hardware happen at a tremendous rate. New GPUs with better hardware specifications enter the market almost every 6 months. In such a fast changing environment, it was important to keep the future in mind. This fact increased the importance of portability many times. Portability offers flexibility to improve the performance by mov-
ing from one GPU to another without changing the software code. The lack of portability results in rewriting the code for each and every architecture. This makes the algorithm, vendor and architecture dependent. This further increases the difficulty in managing the code base for different versions.

- **Ease of Management** It was required to develop an architecture which was easy to manage and access. The designed architecture was to be sustainable and easy to maintain. Another challenge was to develop an adaptable architecture so that future advancements in technology could easily be integrated. This meant that upgrading the existing GPUs to better ones without changing existing code should be possible. Adding GPUs from different vendors should also be possible in the architecture.

Authors in [20] have identified several benefits from the practicality of a single source approach to application design: (i) it is easier to maintain a single code that targets all platforms, as opposed to separate hand-tuned versions of the same code for each alternative platform; (ii) it reduces the risk of being locked into a single vendor solution; (iii) benchmarking is simplified, as the results can be compared from a single code source; and (iv) it represents a "safer" investment for HPC sites, as new codes (and ported legacy codes) will run on both existing and future architectures.

- **Libraries** The algorithm development which was done in C/C++ and python, made it important to have libraries that have GPU support for them. In addition to these basic libraries, there exist many tools which promise GPU support. Since the recommendation engine algorithms primarily dealt with Image Processing, Computer Vision and Natural Language Processing, it was beneficial to find libraries which used GPUs to their advantage. These applications had a faster development cycle and improved the performance by using GPU enabled libraries.

- **GPU remote access** Another requirement of Vionlabs was to have GPUs accessible from multiple host nodes. This was beneficial to the devices which did not have GPU. A single host device was to be given access to multiple GPUs.

### 4.2 System Architecture

In order to satisfy these requirements, the GPU cluster system was built according to the architecture shown in figure 4.1. The system consisted of servers containing data from movies, a heterogeneous GPU cluster and other computing devices such as CPUs, notebooks and laptops. These computing devices had VirtualCL framework which enabled remote access to the GPU cluster. The GPU cluster had direct access to the servers containing movies. The architecture enabled GPU cluster to obtain movie data to be analyzed directly from the server. The algorithms of
the recommender system were located on the computing devices which ran the computationally intensive algorithms on the remotely accessed GPU cluster. The GPUs accessed the servers containing movie data and analyzed their content. They reported the results back to the computing devices where the recommender system stored the required data. The block diagram of the final architecture is shown in figure 4.2. Construction of various components of this block diagram was carried out in three steps which are mentioned in the implementation section of this chapter. All the components of the architecture are discussed in the remaining chapter in detail.

Figure 4.1: System Architecture

Figure 4.2: Block Diagram of System Architecture
4.3 Components of the Architecture

This section discusses various components of the architecture and presents the reasons for their selection.

4.3.1 Selected GPU

The first and the most important component to be selected for the architecture was the GPU itself. The GPU cluster was decided to be built using NVIDIA GPUs which support the possibility of using both CUDA and OpenCL programming models. This ensures availability of a variety of libraries for algorithm development at Vionlabs AB. This selection was made with focus on both, the present and the future. The present needs of Vionlabs AB could have been satisfied using OpenCL programming model. However, in the future, there was a possibility of using both OpenCL and CUDA frameworks for the applications of the movie recommender system. Moreover, the literature study suggested that AMD Radeon GPUs were not able to deliver the expected performance because OpenCL compiler was not mature enough.

NVIDIA introduced the next generation GPU computing architecture which was codenamed Kepler. In the pre-Kepler architectures, a kernel grid could only be generated by a control step and one kernel grid could not invoke another kernel grid. However, in the Kepler architecture, kernel grids can be executed in a nested arrangement without the assistance of control steps. This feature provides the possibility of recursive algorithms. Kepler architecture directly supports GPU virtualization without the need of invoking a third party middleware; thus making GPGPU cloud computing more broadly applicable [10]. It provides general purpose computing capability in the form of on-demand virtual resources.

After deciding to go for NVIDIA GPUs, a market survey was conducted. Performance indexes for various GPUs were compared and analyzed. It was important to choose a GPU which gave good performance at a low price. A performance benchmark for GPUs is shown in figure 4.3. The value written immediately next to the bar in the figure represents the performance index and the value on the right most column shows the price of the GPU. A higher performance index and a lower price for the GPU are indications of a suitable GPU. From the figure 4.3, we can see that GTX 770 GPUs offer a good trade-off between performance and price. Thus NVIDIA GTX 770 GPUs were selected for creating the prototype. The specifications for NVIDIA GTX 770 GPU are given in the figure 4.4.

4.3.2 Selected Parallel Programming Model

The requirements from Vionlabs AB mentioned earlier in the chapter and the literature review promoted OpenCL to be the choice for the parallel programming language. This decision was motivated by the fact that CUDA only supports NVIDIA GPUs and the GPU cluster, in the future, was likely to consist of GPUs from dif-
4.3. COMPONENTS OF THE ARCHITECTURE

Figure 4.3: GPU Performance Benchmarks
Image taken from videogamebenchmark.net

<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>NVIDIA</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU Series</td>
<td>GeForce 700</td>
</tr>
<tr>
<td>GPU Model</td>
<td>Kepler GK104-42</td>
</tr>
<tr>
<td>Fabrication Process</td>
<td>28nm</td>
</tr>
<tr>
<td>CUDA Cores</td>
<td>1536 cores</td>
</tr>
<tr>
<td>GPU Base Clock</td>
<td>1046 MHz</td>
</tr>
<tr>
<td>Memory Clock</td>
<td>1752 MHz</td>
</tr>
<tr>
<td>Effective Memory Clock</td>
<td>7008 MHz</td>
</tr>
<tr>
<td>Memory Size</td>
<td>2048MB</td>
</tr>
<tr>
<td>Memory Type</td>
<td>GDDR5</td>
</tr>
<tr>
<td>Memory Bandwidth</td>
<td>224.3 GB/sec</td>
</tr>
<tr>
<td>Power Draw</td>
<td>230 W</td>
</tr>
<tr>
<td>Min Required PSU</td>
<td>600 W</td>
</tr>
</tbody>
</table>

Figure 4.4: Specifications of NVIDIA GTX 770 GPU

ferent vendors. OpenCL provides a flexible and portable option which makes it possible to maintain a single code rather than many versions of codes for different
GPUs. Code portability offered by OpenCL gives additional flexibility of using the existing code for exploiting parallelism offered by Intel’s multi core CPUs. The portability, however, comes at the price of a slight loss in performance but OpenCL makes up for the loss in terms of reduction in management cost and independence in choosing GPU vendors. In addition to that, the performance gap between CUDA and OpenCL is expected to reduce in the near future as auto tuning mechanisms evolve and the OpenCL compiler matures. Many libraries like pyOpenCL, theano etc. have extended their GPU support and these communities are growing at a fast pace. These libraries provide extensive support for accelerating algorithm development.

4.3.3 Specifications of Selected Devices

The implementation of the architecture mentioned above resulted in the construction of two devices which were used to implement and test it. The two devices were: GPU cluster system and computing device. The specifications of the computing systems are presented below in order to make the research repeatable and sustainable. The GPU cluster system which was used to develop, port and test the algorithms had the following specification.

Specifications of the GPU Cluster

Operating System : Release 12.04 (precise) 64-bit
Linux Kernel : 3.8.0-39-generic
Desktop : GNOME 3.4.2
RAM : 9.8 GiB
Processor : Intel Core i7 CPU 920 @ 2.67 GHz x 8
GPUs : Nvidia GeForce GTX 770 (x2), Nvidia GeForce GTX 680 (x1)
Drivers : OpenCL 1.1, CUDA 6.0.1
OpenCV : version 2.4.8
VirtualCL : version 1.22
IDE : Eclipse CDT 8.2.1

The computing device which was used to access the GPU cluster remotely had the following specification.

Specifications of the Remote System

Operating System : Release 12.04 (precise) 64-bit
Linux Kernel : 3.11.0-24-generic
Desktop : GNOME 3.4.2
RAM : 5.7 GiB
Processor : Intel Core i3 CPU 2120 @ 3.30 GHz x 4
OpenCV : version 2.4.9
VirtualCL : version 1.22
4.4 Application Selection

Applications have traditionally been divided into two categories:

- Synthetic Applications: Synthetic applications are those which provide ideal instructions to make full use of the underlying hardware. A highly parallel application would lie in this category. An application to calculate squares of numbers is example of such an application.

- Real-world Applications: Such applications include algorithms frequently used in real-world domains [11]. These applications are more complex and doesn’t provide a massive set of ideal instructions. A face detection algorithm is an example of a real-world application.

As stated earlier in the thesis, content based analysis of movies is a conglomeration of ongoing research areas like image analysis, computer vision, natural language processing and machine learning. A distinguishing feature of these application areas is that they have plenty of data level parallelism and the data can be processed independently and in any order on different processing elements for a similar set of operations such as filtering, aggregating, ranking [5]. Applications in this area satisfy most of the requirements which are essential to obtain good acceleration. The characteristics mentioned above enable these applications to use multiple processing elements which are present in GPUs and thus these applications are a great fit for acceleration using GPU. In order to test and analyze the performance of the GPU cluster system, two applications were designed and ported to the GPU platform. The applications are described below and their flowcharts are shown in figure 4.5.

4.4.1 Application 1

The first application was a synthetic application. It was a massively parallel application and was chosen to evaluate the best case scenario for the GPU cluster. The application was built in order to investigate the relation between colours present in different frames of a movie and the genre of the movie. It used the well known fact that various colours are indicative of different emotions. For example, red colour is associated with the emotion of love and purple is associated with the emotion of suspense and mystery. This emotional mapping was used to recommend movies based on the relation between the viewer’s choice and the content of the movie.

Mapping between colours and emotions was created for the following colours: black, blue, brown, green, grey, orange, pink, purple, red, white and yellow. The algorithm divides the video of the movie into various frames and accesses every pixel of every frame of the movie. After initializing OpenCL framework, the algorithm uses a look up table to convert each pixel’s RGB (Red, Green, Blue) values into one of the colours mentioned above. The result for each pixel’s colour are stored in an accumulator array. The movie is recommended to the viewer based on the final composition of colours and the viewer’s previous choices.
The algorithm was selected as a synthetic algorithm because it presents the possibility of being massively parallel. A single movie contains hundreds of thousands of pixels and each pixel can be processed in parallel to other pixels. Therefore, GPUs can be used to exploit the huge amount of inherent parallelism present in the application.
4.4.2 Application 2

The second application was a more complex real-world algorithm. It was chosen in order to test the performance of the prototypes in a real world scenario. The application was built in order to investigate the relation between the number of faces present in the movie and the genre of the movie. The idea behind its implementation was that movies can be categorized based on the number of faces that are shown throughout the movie. For example, a movie which has a lot of faces in its frame could be categorized as drama or a documentary and a movie which has a low number of faces in it could be a horror movie, a movie based on aliens or an animated movie.

The application divides the movie into various frames and accesses them one at a time. It uses Viola Jones face detection algorithm on the frame to detect faces present in a frame. If faces are found on the frame, the algorithm increments the counter of total number of faces by the number of faces found. The movie is classified according to the number of faces and is recommended to the viewer based on the classification and the viewer’s previous choices.

This algorithm was considered to be a real-world application because although it offers avenues for exploitation of parallelism, these avenues are limited. For example, in Viola Jones algorithm, cascade filters are used on the frame to detect haar like features [43]. These cascade filters of various sizes can be run in parallel to each other but the computation for each filter needs to be done serially. The algorithm has some inherent serial parts which do not allow the algorithm to be massively parallel.

4.4.3 Optimizations Implemented

As discussed earlier, optimizing an application for a single architecture can cause degraded performance of the application on different architectures. Since this research deals with running the same application on different architectures, optimization was not the main motive of the research. In the research scenario, Vionlabs AB was ready to trade-off efficiency with maintenance and code writing simplicity. As a result, the emphasis in this research was not on optimization alone. The list of optimizations which were implemented on both the applications is provided in the table 4.1.
Reduction in Synchronization
Synchronization mechanisms and barriers add overhead to the GPU functionality. Therefore, a reduction in the number of barriers results in better performance.

Choice of floating point precision
Double precision floating point operations take more time than single precision floating point operations. Since the recommender engine applications didn’t need high accuracy, all double precision operations were converted to single precision operations.

Selection of work group size
As mentioned earlier, size of the work group affects the performance of the GPU. After testing different values, it was found that a work group of size 64 gave the best performance.

Compiler Optimizations and fast-math optimisations
After trying out various optimization levels provided by the compiler, level 3 optimizations (using -O3 flag) were found to give the best performance. OpenCL optimizations like ‘-cl-fast-relaxed-math’ and ‘-cl-mad-enable’ were used to give performance improvement.

Use built-in functions
Using hardware support for math intensive functions like sqrt resulted in improved performance.

Moving branching up the pipeline
Branching affects the performance of GPU pipeline in a negative manner. Performance can be increased manifold if branching occurs early in the code. A branch present in the inner most loops must be avoided at all times.

GPU memory hierarchy exploitations
Utilizing GPU memory hierarchy in the correct way results in tremendous gains. Global memory is huge in size but accessing it is a costly operation. Copying of data from global memory to the local memory in a large packet and then using local memory to access the data increases the performance of the applications.

<table>
<thead>
<tr>
<th>Optimization</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reduction in Synchronizations</td>
<td>Synchronization mechanisms and barriers add overhead to the GPU functionality. Therefore, a reduction in the number of barriers results in better performance.</td>
</tr>
<tr>
<td>Choice of floating point precision</td>
<td>Double precision floating point operations take more time than single precision floating point operations. Since the recommender engine applications didn’t need high accuracy, all double precision operations were converted to single precision operations.</td>
</tr>
<tr>
<td>Selection of work group size</td>
<td>As mentioned earlier, size of the work group affects the performance of the GPU. After testing different values, it was found that a work group of size 64 gave the best performance.</td>
</tr>
<tr>
<td>Compiler Optimizations and fast-math optimisations</td>
<td>After trying out various optimization levels provided by the compiler, level 3 optimizations (using -O3 flag) were found to give the best performance. OpenCL optimizations like ‘-cl-fast-relaxed-math’ and ‘-cl-mad-enable’ were used to give performance improvement.</td>
</tr>
<tr>
<td>Use built-in functions</td>
<td>Using hardware support for math intensive functions like sqrt resulted in improved performance.</td>
</tr>
<tr>
<td>Moving branching up the pipeline</td>
<td>Branching affects the performance of GPU pipeline in a negative manner. Performance can be increased manifold if branching occurs early in the code. A branch present in the inner most loops must be avoided at all times.</td>
</tr>
<tr>
<td>GPU memory hierarchy exploitations</td>
<td>Utilizing GPU memory hierarchy in the correct way results in tremendous gains. Global memory is huge in size but accessing it is a costly operation. Copying of data from global memory to the local memory in a large packet and then using local memory to access the data increases the performance of the applications.</td>
</tr>
</tbody>
</table>

Table 4.1: Optimizations Implemented in the Applications
4.5 Implementation of System Architecture

This section states the steps for developing a prototype of the final architecture. The decision of prototyping was positioned well towards the beginning of the development life cycle. Authors in [34] suggested that a good prototype must be built on the basis of information already gathered, keeping in view the requirements. The requirements mentioned earlier and the information gathered by interaction with Vionlabs personnel laid a good foundation for prototyping the GPU cluster. Encouraging the Vionlabs AB users to participate early in the process allowed them to have more control over the project and reduced the risk of having undetected mistakes. In a later stage of the cycle these mistakes would have been considerably more difficult and expensive to deal with.

Although a comprehensive list of requirements was important for successful prototyping, there were other important factors such as selecting the correct method. In [36], the authors advocated creating a rapid prototype in the beginning and then using iterative or evolutionary prototypes according to the needs of the project. This approach offered the benefits of fast and incremental evaluation of the system in different stages.

In order to implement the system architecture, the work was divided into creation of three prototypes in three iterations of boehm’s spiral model. Boehm’s model was identified as the most suitable model since it overcomes the drawbacks of the waterfall model. Test driven development strategy was applied to the development of GPU cluster in all the stages of prototyping. Test cases were developed for each prototype which were used to evaluate the prototype’s performance. The prototype was declared successful once it passed the performance evaluation based on the developed test. The three prototypes and their implementation is described below.

4.5.1 Single GPU Implementation

The block diagram implemented in the single GPU implementation is shown in the figure 4.6.

This prototype was created using only one GPU to assess the improvement in performance. It was simple and relatively quick to create, amend and build because a prototype, that takes as long or costs as much to build as the production version, is of little use. This sub-system was used to port and test applications on the GPU platform. The GPU used for building the prototype is shown in figure 4.7. After a demonstration and performance evaluation, a discussion with the users led to the conclusion that performance of the prototype was satisfactory.

This prototype was a cheap prototype which enabled quantification of the expected increase in performance. If this sub-system didn’t perform as expected, it would have been easy to discard. Since the user was made aware of the results to expect from the final system and the development was at a very early stage, the flexibility offered by the prototype offered a possibility to change the requirements,
4.5.2 Multiple GPU Implementation

The block diagram implemented in the multiple GPU implementation is shown in the figure 4.8.

This iteration in the evolutionary prototyping constituted of developing a heterogeneous GPU cluster sub-system which was built to showcase GPU cluster capabilities. The sub-system with multiple GPUs was created and applications were run if necessary.
4.5. IMPLEMENTATION OF SYSTEM ARCHITECTURE

and tested using these GPUs. It consisted of three GPUs and two power supplies. The power supplies were connected using ADD2PSU connector. The GPUs were connected to the motherboard using PCIe extension cables.

The built sub-system can be seen in figure 4.9. This method of prototyping resulted in a reduced cost of implementation because components from the previous iteration of the process were reused. Building the prototype on top of the existing system reduced the time taken to build it in this iteration.

4.5.3 Remote Access Implementation

The block diagram implemented in the Remote Access implementation is shown in the figure 4.10.

This final iteration consisted of enhancing the heterogeneous GPU cluster with remote access capabilities. In order to satisfy the requirement of remote access to multiple GPUs, the GPUs needed to be implemented in the form of a cluster.

In order to realize remote access to the cluster, VirtualCL framework was used. This framework provided a layer of abstraction to the GPUs present in the computer systems on the same network. A system having VirtualCL framework was able to access GPUs present in networked systems as if they were present on the local machine. The framework is divided into the three parts mentioned below.

- VCL Library
- VCL Daemon
- VCL Broker
These parts can be seen in the figure 4.10. The application developed using OpenCL uses the VCL library, which takes care of converting OpenCL calls to VCL
understandable calls. The broker keeps track of GPUs present in the networked systems and contains the updated states of the GPUs. The library contacts the broker and finds if the required GPUs are available to run the code or not. Once the required GPUs are available, the broker distributes the work load among the different GPUs. The broker sends the appropriate data along with the OpenCL commands to the VCL Daemons on networked systems. The daemon has the responsibility of running the commands on GPUs, provide the data to the GPUs and to obtain the results back. Once the results are obtained, the daemon send the result data back to the broker which passes the data to the VCL library. Finally the application retrieves the data from the VCL library.

The network architecture for the recommender system consisted of a single host node and a single compute node. The host node consisted of VCL library and the broker. The compute node consisted of the library, broker and daemon of the VCL framework. This architecture is shown in the figure 4.10. Using the VCL framework, all the GPUs on the compute node were used as OpenCL compute devices. The host node, which did not have a GPU, executed the host program as an OpenCL application. VCL framework provided the illusion of the presence of local GPUs. The framework allowed the OpenCL application to utilize GPUs in a compute node as if they were present in the host node [44]. No change in behaviour of the application was noticed when execution of the code was happening on the compute node.

One of the main advantages of VirtualCL framework is that there is no need to change the application developed for a single GPU. The same application was run on multiple GPUs using the VCL framework. Once familiarity with the framework was gained, it was easy to setup on various computer systems.

The final network of systems is shown in figure 4.11. The system on the left side of the figure was used to remotely access the GPUs on the system on the right side.
The specifications of the two systems has been mentioned earlier in this chapter. The systems were connected using Gigabit LAN which was the fastest available interconnect. The OpenCL applications developed were run on the remote system. The VCL library translated the OpenCL calls to VirtualCL understandable calls. VCL broker on the remote system sent the required data and the calls to the VCL broker present in GPU cluster system using LAN interconnect. The information was passed to the VCL daemon present in the GPU cluster system. The daemon used the GPUs on the cluster and calculated the results. The results were passed back to broker of the remote system through the broker of the GPU cluster. The OpenCL application finally received the results from the VCL library. In this manner, the OpenCL application was able to access the GPUs in the cluster system.
Chapter 5

Results and Analysis

The goal of the thesis was to provide guidelines for building a heterogeneous GPU cluster for accelerating applications of the movie recommender system being built by Vionlabs AB. It was required to build a system using the guidelines and to test its performance against existing computer systems and the prototypes built in different stages of the research. This chapter presents the results obtained from the prototypes described in the previous chapter. The chapter also presents a comprehensive discussion and analysis of the obtained results.

5.1 Evaluation Procedure

In order to evaluate the performance of the GPU cluster, applications 1 and 2 were run on different prototypes. Following prototypes were used to evaluate the performance of the system.

- A) CPU with optimized applications
- B) GPU without using VCL framework
- C) Local GPU through VCL framework
- D) Remote GPU through VCL framework

The applications were tested using following movies: Forrest Gump, Remember the Titans, Saving Private Ryan, Star Wars IV, The Avengers, The Notebook, The Shining, Silence of the Lambs, The Usual Suspects and Vivian Maier. These movies were selected because they were the best movies from their respective genre. The applications were run for all the movies mentioned above and their running time was calculated for all the prototypes. The results obtained from performance analysis of various prototypes is mentioned in the next section.
5.2 Results

For each of the above mentioned prototypes, the execution time of the applications was noted and plotted. The results obtained after running application 1 and application 2 for all the movies are shown in Figures 5.1 and 5.2 respectively. In the figures, Y-axis shows the time taken in seconds for the analysis of movie mentioned on the X-axis. The figures show that the best performance is obtained by using the prototype B which used GPU without using VCL framework. Prototype C which used local GPU with VCL framework provided the next best performance. In case of application 1, prototype D which used GPU cluster remotely through VCL framework outperformed the prototype A which used the optimized CPU version of the application. However, in case of application 2, the prototype A outperformed prototype D which used VCL framework to provide remote access to the GPU cluster.

![Figure 5.1: Results of Application 1](image)

The figure 5.3 illustrates the properties of movies extracted after running the first application for different movies. Each graph shows the distribution of colours for the movies listed above. The x-axis of each graph shows the colours present in each frame of the movie. The y-axis of each graph shows the amount of colour present in each frame.

The comparison of movies based on the results obtained from the second application can be seen in the Figure 5.4. The graph shows the ratio of number of faces to the total number of frames in different movies. This result can be used to recommend movies to various users based on the user’s previous movie choices.
5.3 Analysis

It was seen that applications 1 and 2 provided an improvement of 100x and 66x respectively when the performance of their GPU versions was compared to their un-optimized CPU versions. However, since that comparison would have been flawed, the CPU versions were optimized and the comparison was made again. From the figures 5.1 and 5.2, we can see that the average acceleration on porting of a highly optimized CPU version to a moderately optimized GPU version has approximately been 5x and 1.5x for application 1 and application 2 respectively. Acceleration obtained from the two applications for different movies is shown in the Figure 5.5.

Due to the fact that application 1 represents an application which is close to an ideal, massively parallel application, the results obtained for application 1 are more optimistic than those of application 2. However, it is important to note that if in some application areas, performance is the key, optimizing the GPU code further can provide a better solution.

Figures 5.1 and 5.2 show that the overhead of using VCL framework to access the local GPU is not much. As a result, the second best performance in both the applications is obtained by prototype C which was using local GPU with the VCL framework. However we see that, the penalty of using VCL framework to access the GPU cluster remotely is application dependent. As a result, for application 2, prototype A which used the CPU version performed better than the prototype D with remote access to GPU cluster.
Figure 5.3: Movie Analysis using application 1
Figure 5.4: Movie Analysis using application 2

Figure 5.5: Acceleration for the movies
Figure 5.6: VCL overhead analysis based on size of data to be transferred

Figure 5.7: VCL overhead analysis based on number of frames to be transferred
5.3. ANALYSIS

On analyzing VCL overhead based on figures 5.6 and 5.7, it was noted that a general pattern could be seen. It was seen that in most cases, VCL overhead increased as the data to be transferred increased. Another important thing to be seen was that the VCL overhead increased along with the number of frames to be analyzed. In both the applications, the number of frames to be analyzed were directly proportional to the kernel calls made by the application. As a result, the increase in VCL overhead could be linked to the two factors.

It was seen that the remote access to GPU cluster provided a degraded performance in the two applications. This can be linked to the fact that the processor present in the GPU cluster was much better than the one present in the remote system. The performance of applications could also have been affected by the connection between the host node and compute node. In order to obtain a good performance, high speed Gigabit LAN was the minimum requirement. In the market, better interconnects like infiniband interconnects exist which could reduce the VCL overhead and improve the performance of the VCL framework.

If the GPUs were not being used in a cluster, each GPU would have required a separate CPU for running. By using the GPU cluster lesser number of CPUs were being used to run more number of GPUs. As a result, the total amount of energy spent on running the applications on the cluster was reduced. Using GPU cluster to run the applications also resulted in better utilization of the GPUs because the idle time for the GPUs gets reduced. It can be seen that using GPU cluster provides a energy efficient method of accelerating the movie recommender system.
Chapter 6

Conclusions and Future Work

The guidelines presented in this thesis enables organizations and individual researchers to build suitable GPU clusters on their own. The factors to be considered when building a GPU cluster have been suggested in the thesis. The guidelines were successfully tested by building and evaluating the performance of a heterogeneous distributed GPU cluster. The cluster was built and its performance was tested using two strategically chosen applications. The applications were ported, optimized and evaluated using four different system configurations. The applications were not developed with performance as the only goal and were developed to represent the real world scenario. They were written using OpenCL parallel programming model which provided portability of code across various GPU architectures.

Among the four configurations, the prototype which used GPU without using VCL framework was found to provide the best performance. It was closely followed by the prototype in which local GPUs were accessed using VCL framework. This confirms that GPU architecture is well suited for running analysis algorithms of a recommender system. The increase in performance implies that the time spent in analyzing movies for the recommender system will be reduced manifold. Thus, this method of enhancing the performance of applications is very useful and saves a lot of time.

Although the prototype in which the GPU cluster was accessed remotely showed performance degradation because of VCL overhead, it was found to have other benefits. It provided GPU functionality to any other system present in the same network. This can further result in reduced idle time of the GPUs and reduction in cost and energy consumption. This prototype provides a mechanism to reduce the number of GPUs required for algorithm analysis.

It was found that VCL framework is best suited for applications which have low data to be transferred and more computation on the GPUs. Reducing number of kernel calls in such applications will improve the performance of the remote access to the GPU cluster. The performance of GPU cluster can also be increased by a better host processor and a better interconnect which connects the host systems to the GPU cluster.
The steps taken for prototype development were in accordance with suggestions in the literature and their benefits were clearly seen. Evolutionary prototyping helped the prototype to evolve into the final system. In future, prototyping is expected to grow in popularity as a high-productivity tool. Test Driven Development provided realistic checks and feasibility analysis in different iterations. These checks helped in verifying the usefulness of the prototype.

6.1 Future Work

The research work can be extended by creating and implementing a GPU based cloud architecture. Using the VirtualCL framework, systems in the cloud will be able to access GPUs of other systems over the internet. Such a system will result in building a powerful distributed accelerator system at an affordable cost.

Since this research was not performance oriented, optimization was not the most important goal of the GPU implementation. The research can be re-iterated with focus on optimizations. Literature suggests that optimizing on a single architecture results in over optimized code for other architectures. It will be interesting to find the point of over optimization of the code. It will also be beneficial to test the performance of applications which are capable of using multiple GPUs.

Another interesting area of research is to investigate the VCL framework and quantify the effects of host processor and interconnects. The reasons for poor performance of application 2, when it was remotely accessing the GPU cluster, need to be investigated. It will be interesting to see the effect of optimizing the applications on performance of the system.

VCL framework is still under development and there is a need to improve it. However, there are other frameworks which offer the capability of remotely accessing the GPUs present in a cluster. Some of the prominent frameworks are rCUDA and starPU. The performance results obtained in this study can be extended in the form of a comparison of VCL framework with other frameworks.
Bibliography


Appendix A

Abbreviations

AMD - Advanced Micro Devices
API - Application Programming Interface
CPU - Central Processing Unit
CUDA - Compute Unified Device Architecture
GPGPU - General Purpose computing on Graphics Processing Units
GPU - Graphics Processing Units
HMPP - Hybrid Multicore Parallel Programming
HPC - High Performance Computing
LAN - Local Area Network
OpenCL - Open Computing Language
OpenCV - Open Computer Vision
pyCUDA - Python CUDA
pyOpenCL - Python OpenCL
SIMD - Single Instruction Multiple Data
SM - Streaming Multiprocessor
TDD - Test Driven Development
VCL - VirtualCL