THE IMPLEMENTATION OF PARALLEL COMPUTATION ON CPU AND GPU

by

Kun Xia

Approved:  

Xiaoming Li, Ph.D.  
Professor in charge of thesis on behalf of the Advisory Committee

Approved:  

Kenneth E. Barner, Ph.D.  
Chair of the Department of Electrical and Computer Engineering

Approved:  

Babatunde A. Ogunnaike, Ph.D.  
Dean of the College of Engineering

Approved:  

Ann L. Ardis, Ph.D.  
Senior Vice Provost for Graduate and Professional Education
ACKNOWLEDGMENTS

This study won’t be completed without helping and understanding from Dr Li. Thank you so much for giving me this opportunity to work with you in researching of parallel computation in past year.
# TABLE OF CONTENTS

LIST OF FIGURES .................................................................................................................. vi
ABSTRACT ................................................................................................................................. viii

Chapter

1  INTRODUCTION .................................................................................................................. 1
   1.1 Motivation ......................................................................................................................... 1
   1.2 Background ....................................................................................................................... 2

   1.2.1 Serial Processing and Parallel Processing ................................................................. 2
       1.2.1.1 Definition ............................................................................................................ 2
       1.2.1.2 Why Do We Need Parallel Processing? ......................................................... 4

   1.2.2 Parallel Programming and CPU ................................................................................ 5
   1.2.3 Parallel Programming and GPU ............................................................................... 7
   1.2.4 Parallel Programming Structure ............................................................................. 8

       1.2.4.1 MPI .................................................................................................................... 8
       1.2.4.2 OpenMP ............................................................................................................ 10
       1.2.4.3 OpenCL ............................................................................................................ 13

2  RESEARCH OBJECTIVES ................................................................................................... 17
   2.1 Problems ......................................................................................................................... 17
   2.2 Experiment Goals .......................................................................................................... 19

3  EXPERIMENTAL METHODOLOGY ............................................................................... 21
   3.1 Design ............................................................................................................................. 21
       3.1.1 Tree Algorithm ....................................................................................................... 21
       3.1.2 Serial Computation ............................................................................................... 23
       3.1.3 Parallel Computation ............................................................................................ 27

       3.1.3.1 Implement of Pthread ................................................................................. 27
       3.1.3.2 Implement of CUDA ..................................................................................... 29
3.2 Performance Evaluation and Analysis..................................................... 33
  3.2.1 Serial Program............................................................................. 35
  3.2.2 Pthread Computation................................................................. 36
  3.2.3 CUDA Computation................................................................. 38

4 CONCLUSION .................................................................................. 39

REFERENCES .................................................................................. 40
LIST OF FIGURES

Figure 1.1: Varieties of Parallel and Serial Processing .......................................................... 4
Figure 1.2: The Explanation of serial Processing ................................................................. 6
Figure 1.3: The Explanation of Parallel Processing with Single CPU ..................................... 6
Figure 1.4: The General Ideal of CPU and GPU’s Cores ......................................................... 8
Figure 1.5: Shared Memory Model and Distributed Memory Model (Shao-Ching Huang, 2013) ................................................................. 11
Figure 1.6: OpenMP Execution .......................................................................................... 13
Figure 1.7: Heterogeneous Computing Structure (Cliff Woolley, 2010) .............................. 14
Figure 1.8: Model of OpenCL .......................................................................................... 15
Figure 2.1: The Different Performance of Parallel code based on their algorithms (n.d.) ................................................................. 19
Figure 3.1: Tree Structure (tutorialspoint.com) ................................................................. 22
Figure 3.2: Example of Binary Tree .................................................................................. 23
Figure 3.3: The Structure of Serial Program ...................................................................... 25
Figure 3.4: Solution for Unperfect Tree ............................................................................. 26
Figure 3.5: The Data Flow of CUDA Function (Xiaoming Li) ............................................ 31
Figure 3.6: CPU Information (virtual machine) ................................................................. 34
Figure 3.7: Performance of Serial Computation ............................................................... 35
Figure 3.8: the Performance of Pthread Computation ......................................................... 36
Figure 3.9: The Performance Difference between Pthread Computation and Serial Computation ........................................................................... 37
Figure 3.10: The Performance of CUDA Computation and Serial Computation ....... 38
ABSTRACT

High performance computations have never been failed to attract people since the first computer was created. Before using parallel computations, people tried to increase the performance of single core processor. But the large power consumption and heat releasing became to the biggest performance limit for single core processor. Therefore, the technology of parallel programming was introduced couple years ago and replaced the role of single core processor. Parallel computation is the algorithm that control multiple operations simultaneously. It can be implement not only on multiple processor device, but also implement on single processor. This study is about researching implementation of parallel computing with C. After introducing the basic idea and structures of parallel computation, the implementation of parallel computation will be created with tree algorithm. On the single processor like CPU, parallel computation is applied with pthread function. And for the multiple processor like GPU, CUDA is the best choice for us. Meanwhile, the serial function is also needed for testing the correctness of parallel computing functions and performance analysis.
Chapter 1

INTRODUCTION

1.1 Motivation

Most people have the experience for the parallel computations. People can play video games, watch news, or even work on their PC or Mac when their music is on. The computer used today is able to supply running multiple tasks at the same time. This is the application for the parallel computations. The world in which people live today is full of information and data. A large amount of data is generated and transformed every second all over the world and it doesn’t stop. Based on the paper from Andrew McAfee and Erik Brynjofssion(2012), there were 2.5 exabytes data created every day, and this number will be doubled every forty months. The amount of data transformed through the internet every second is much larger than we have stored in past twenty years. How to collect and process those data have become a huge challenge for human beings. People need to make their computer faster and faster to handle the growth of data. Because of the fast processing of big data, we can shop, watch news, and communicate with friends through the internet smoothly. Companies need high-speed server with parallel processing to provide high-quality service for their customers.

Ways of improving the processing speed is the key for dealing with heavy loads. Generally, the CPU can only process one task at one time, and other tasks will be processed after the previous task is finished. However, the CPU is not keeping running all the time during the task. The most of time is wasted by waiting tasks such
as memory accessing. When people facing this situation, they may want to use the waiting time for other tasks or they may want the CPU do multiple tasks at the same time. Therefore, the concept of parallel programming is introduced to people’s life and it will be extremely helpfully for improving the processing speed and efficiency. In order to increase the efficiency of CPU, the parallel program will allow it to start a new task during the waiting time. If the CPU has multiple cores, it will be able to process multiple tasks simultaneously. This research is studying the concept of the parallel programming and it will be implemented on multiple platforms like CPU and GPU.

1.2 Background

1.2.1 Serial Processing and Parallel Processing

1.2.1.1 Definition

The difference of the principle between the serial processing and parallel processing are about precedence and simultaneity. During the serial processing, multiple tasks will be put into a sequence memory and CPU will process only one task at a time. The next task will get start when the previous one is finished completely. Under the serial processing, CPU just works for one single task, and other tasks are not allowed to be accessed. Waiting in the checkout line in the grocery store is the best example to show the concept of serial processing. People have to wait in line until the last customer finish their payment. Blaise Barney who is working for Lawrence Livermore National Laboratory showed the basic idea about serial computing and processing in 2016. He found that people usually break the large problem into small discrete of serial tasks, and process them sequentially one after another.
If the computer can execute the multiple processes or calculations simultaneously, this type of processing is called parallel processing or parallel computation. In Blaise Barney’s idea, big loads are usually divided into several similar small parts, and those parts will be processed at the same time. The example of checkout line with multiple registers is also a good example to show this concept of parallel processing. Because of the multiple registers, the store can serve multiple customers at the same time. In Gorden’s paper, *Parallel and Serial Processing* (2002), a good example as showed in figure 1.1 represents the difference between parallel and serial processing. In his imagination, there are two different processes, A and B. “If A and B go simultaneous, then process is parallel. If A precedes B or B precedes A, then processing is serial.” In the figure 1.1, the P is the perceptual processes, and M is motor processes.
1.2.1.2 Why Do We Need Parallel Processing?

Along with the development of computer and Internet technology, more and more data need to be collected and processed. Therefore, people need faster computer to handle those data and complex situations. For example, we need faster processor to make robot smarter, and lives easier. Before the implementation of parallel computation, people tried to increased the performance of single processor. However, based on the lecture of Peter Pachoco from Morgan Kaufmann, the faster processor needs smaller transistors. However, because of the limit of our industry technology, it is very hard to make the transistor small enough. Moreover, faster processor also consumes more power, and generate more heat. The increasing heat will reduce the reliability of the processor. Therefore, the parallel computation technology becomes
the best way to solve this problem. The multiple cores processor replaced the single core processor and was applied in people’s computer.

1.2.2 Parallel Programming and CPU

Professor Lu, from Shanghai Jiao Tong University, represented the truth of parallel processing on CPU that we use today. He believed that the parallel processing on the single processor is not the real parallel processing. If we divide the processing period into very small parts, we will not find any simultaneous processing. The CPU assigns those tasks into different times because the single CPU can only execute one task at a time. This concept is illustrated in figure 1.2 and figure 1.3. There are three tasks that need to be executed by CPU, and each task have two steps. Each step takes 1 millisecond, and there is also a two-milliseconds waiting time between two steps. Like the figure 1.1 shows, the new tasks have to wait until the previous task is completed. In the parallel processing, after finishing the step one of the previous task, the next task will get start during the waiting time, and the second step of first task will resume after this task goes into its waiting time. In generally, the CPU will keep switching execution between multiple tasks, and it just take a very short time. Therefore, people can’t find the switching when they use their computer, and in their view that multiple tasks are executed simultaneously. On the other hand, if computer has multiple processors, it can do the real parallel processing, because different processors can do different executions at the same time.
The information introduced by professor Lu is the idea of multithreading. If the CPU can execute multiple threads at the same time, people call this ability as multithreading. Those threads come from the same program, and each thread is executed independently. A lot of programs are designed with multithreading technology. Even though the CPU can only run one thread at a time, people still feel that the multiple threads are executed simultaneously by the quickly switching between different threads. For example, most of video games are multithreading.
programs. Those video games have to make sure the game won’t get stuck during the period of waiting users’ input.

1.2.3 Parallel Programming and GPU

GPU, as known as the Graphics Processing Unit, is the microprocessor designed for the high-speed graph and image creation and computation. Based on the information from NVidia website, the CPU is designed for sequential serial processing with a few cores. For example, the Intel core i7-7920 processor has 4 cores. Meanwhile, in order to handle the complex multiple tasks simultaneously, the GPU contains thousands small high efficient cores. Each core can execute different thread independently. The figure 1.4 show the difference between CPU and GPU, and the small blocks in the figure represent cores in CPU and GPU. Originally, the primary purpose for designing GPU is improving the ability of computing and processing for 3D display. And now, this ability is widely used in a lot of fields like math modeling, science and medical technology.

With the powerful ALU (Arithmetic Logic Unit) and large caches, CPU is designed for dealing with heavy complex loads. Meanwhile, GPU can execute hundreds of threads simultaneously through its thousands of small cores. Therefore, GPU is good at processing the simple and repeated tasks. For example, the computer can be treated as a company, and CPU and GPU are different departments. In GPU department, there are thousands of entry level workers with low level skills. Meanwhile, in the CPU department, there are just a few of high level scientists, and engineers with professional skills. If this company wants to simply make one thousand copies of a book, the GPU department will be much faster than CPU. This task will be divided into thousands small tasks, and each part can be done simultaneously.
However, if this company have to deal with a tough problem like write books, the CPU department will be the best choice.

Because of the different capabilities, the CPU and GPU are used as a combination in people’s computer and deliver the best performance.

Figure 1.4: The General Ideal of CPU and GPU’s Cores

1.2.4 Parallel Programming Structure

1.2.4.1 MPI

MPI, as known as the Message Passing Interface, is a parallel programming API for message passing. The MPI is a routines library rather than a programming language. Sivarama Dandamudi, the professor from Carleton University in Canada
said, “MPI defines the syntax and semantics of a core of library routines useful to a wide range of users writing portable message passing program in Fortran77 or C”. Most of the MPI implementations are open source, and they are supplied by all parallel computers we used today.

After getting the information of MPI, people may feel confused what the Message Passing is. Message Passing is the communication between different processes when those processes serving the same task. For example, computer is like a company, and the master process is boss. Boss assigns the tasks to different employees (slave process in computer) by emails. Message Pass between processes is like those emails between boss and employees.

Back to the beginning of 90s, it is difficult for people to create their parallel programs. As people know, most programming languages used at that time didn’t have ability for parallel processing. Although some companies and people developed their own extensions for parallel programs, they were not available for cross-platform applications.

Along with the development of computer technology, more and more parallel programs are created, and some of parallel program models and libraries became popular. In 1992, people decided to combine those popular models and libraries together, and created a standard interface which would be commonly-available for all major parallel programming structures. After couple years hard working, the interface was created and named as MPI. Because of MPI, the developing of parallel program become much easier. Programmers just needed to know MPI, and they can develop their parallel program on any major platform. It also become possible that multiple different computers or servers executed the same program simultaneously. Now MPI
become to a stable, high efficiency, and high applicable tool which is widely used in every computer.

1.2.4.2 OpenMP

Before diving into detail information, we need understand the concept of parallel programming model on CPU. There are two major models for parallel execution of multiple processors. The first one is Shared Memory Model. In this model, all CPUs share the same memories which means that the shared data is available for all processes or threads. Another one is Distributed Memory Model that every CPU gets their own memory and those memories are not accessible for other CPUs. The figure 1.5 shows the basic idea of Shared Memory Model and Distributed Memory Model below.
In 2011, Doctor He, the HPC Consultant from User Engagement Group in Berkley, CA, represented the definition of OpenMP. She said that “OpenMP is an industry standard API of C/C++ and Fortran for shared memory parallel computing.”
programming”. As Dr. He said, OpenMP is not a computing or programming language, and it only can be used with a sequential program like C/C++ programs. Moreover, OpenMP is using “Fork and Join” model as its execution model. OpenMP program starts with one master thread and executed serially until reach parallel region. In parallel region, the master thread will be divided to small threads, and those threads will work simultaneously in different processors. At the end of parallel region, threads join together and generate another master thread. It is executed serially again until meeting next parallel region. The illustration of OpenMP is shown in Figure 1.6.
OpenCL

OpenCL, also known as Open Computing Language, is a very popular framework for parallel computing. On NVidia website, OpenCL is defined as “a low-level API for heterogeneous computing that runs on CUDA-powered GPUs. Using the OpenCL API, developers can launch compute kernels written using a limited subset of the C programming language on a GPU.” Before getting into the details of OpenCL, we need to know what the heterogeneous computing is. In the beginning of this
chapter, we introduced basic concepts and features of CPU and GPU. As we know that CPU is good at serial complex processing, and GPU is designed for simple parallel computing. But which one do we need to pick when we deal with a large complex task? Therefore, people start to try to combine the CPU and GPU together and let them work for the same tasks. The behavior of incorporating different processors (like CPU and GPU) to participate the same program is called heterogeneous computing. The figure 1.7 shows the idea of heterogeneous computing. The primary work for CPU is logic control, like the manager in the company. CPU analyzes and simplify the into small parts and assign to GPU, and GPU will do the simply computing works through its thousand cores. Heterogeneous computing computer is much faster than any traditional computing methods.

Figure 1.7: Heterogeneous Computing Structure (Cliff Woolley, 2010)
After introducing heterogeneous computing to public, many computer vendors started to develop their own heterogeneous computing systems and methods. For example, NVidia create the CUDA for their GPU users, and made their system thousands time faster than old one. However, users from different companies like AMD or Intel can’s use CUDA to speed up their computers. In order to solve this problem, Apple started to develop an universal heterogeneous computing framework for the majority of users. After hard working with multiple major GPU companies like NVidia, AMD and Intel, OpenCL is created and opened to public for free.

Figure 1.8: Model of OpenCL
As shown in figure 1.8, OpenCL is formed by host and one or more devices. When we execute the OpenCL program, the host (CPU) will generate kernel functions and send them to devices (usually are GPUs, and sometime can be other CPUs). Those devices are formed by many computer units which are processor in GPU or cores in CPU. Those kernel functions will be processed simultaneously in compute unit, and send the result back to host. Because of the good portability and accelerate ability, OpenCL is widely used today, and the its potential will make computers more and more efficient.
Chapter 2

RESEARCH OBJECTIVES

2.1 Problems

Parallel programming has been created for decades, and it was widely used in every computer today. Back to few years ago, there was a race between companies for building the fastest single core processor. However, people found that there was a bottleneck for hardware technology because faster processor would consume more power, and more power consumption brings more heat. However, it was hard to find an appropriate way to get rid of those heats to avoid their process get burned.

Instead of making the impossible processor, people decided to use multiple processors to improve the computer’s performance. The heavy load was divided into multiple small parts, and executed by different processors simultaneously. It is parallel programming technology, and it improved the performance of computer almost thousand times higher than serial one. In past few years, Multiple cores CPUs and GPUs are good examples for parallel processing. More and more cores were built in new processor, and the processor’s performance is also improved significant with same size or power consumption. However, there still are some problems and challenges for parallel programming.

The biggest challenge is that people can’t think parallely. In Steven Wildstrom’s paper, *The World is Parallel: The Opportunities and Challenge of Parallel Programming*, he said “Neuroscientist Jill Bolte Taylor says the right hemisphere of the brain, which processes sensory signals, does parallel processing but
the left hemisphere, which is responsible for analytic thinking, “functions like a serial processor.” For better or worse, programming is a left-brain activity.” As people know in their daily life, people usually can only focus on one thing at a time. For example, students can’t pay attention on their homework while playing video games. Many software developers may have the experience that they sometimes get stuck in some simple problems, and the obvious method doesn’t bring any expected results. Victor Lee, who come from Parallel Computing Lab at Intel, shows the data from Amdahl’s law. It shows the performance based on the different algorithms in figure 2.1. Based on his data, the serial algorithm doesn’t have very good performance in parallel program. Therefore, people have to keep the concept of parallel computation in their mind when they deal with parallel programs. Steven Wildstrom also said that “The biggest mathematical impediment to parallel approaches is that many processes are recursive: each step depends on the result of previous steps. Consider the simple problem of finding the greatest common divisor of two integers. The standard method of doing this, the Euclidean algorithm, has been known for over 2,000 years and uses repeated subtraction.”
The another challenge is how to improve the efficiency of the exist programs through CPU and GPU computing. People have used computer for more than half century. There are a lot of programs and applications created in the past, and a lot of them are still used today. It is easier to create a new program with high efficiency, but we can’t replace all old programs and application. Therefore, how to modify those old applications with high performance parallel computation is still a big challenge for software developer.

2.2 Experiment Goals

Based on the problems and challenges mention on previous section, parallel computation used today is still far from enough in each field, and there is a big potential development for it. As the result, there are two main goals in this study.
The first one is studying and finding out the better parallel programming algorithm to replace traditional serial algorithms. Even if the serial computation need be used in the code, the parallel computation still needs to finish the majority of the tasks. In this study, the parallel programs will be applied on CPU and GPU, and executed with tree algorithm. The input data will be generated into a sequential array, and each element in this array will be treated as a child. Multiple children in every subtree will be added together and return the result to their parent. Then, their parents will be treated as children and do sum algorithm again until root get final result. On the CPU, it will be implement with Pthread (POSIX thread) method, and on GPU, the CUDA is the technology of parallel computation.

The second goal is how to optimize program and reach the high performance. For this goal, we will pay attention on how to access memories wisely.
Chapter 3

EXPERIMENTAL METHODOLOGY

3.1 Design

In order to reach the goals, there are three programs created in this study which are serial program, Pthread parallel program, and CUDA program. The serial program is also used as the test code in this study. Results from other programs will be compared with the serial programs result. If those results are equal, the function is executed correctly. The best way to compare their performance is comparing their execution time. If the program can be finished in shorter time, its performance will be better than others. Moreover, C is the only programming language used in this study.

3.1.1 Tree Algorithm

There are a lot of data structures like array, link list, and stack used today. They are good structure that can be accessed and implement easily. However, they are “Linear Structures”, and they will take a long time when the data size is huge. For example, to find the max value from an array, program has to access each element serially, and it is not suitable for the idea of parallel computation. Therefore, in this study, the tree structure is introduced and implement for parallel programming.
As shown in Figure 3.1, tree structure is formed by nodes, and the node on the top of the tree is called the root. If any node connects forward to at least one node, this node is called the parent node. The node which connects to the parent is called the child node.

In this study, the input data will be stored in an array, and each element in this array can be treated as a child node in the tree structure. After getting the sum of multiple child nodes, the result will be pushed into the parent node and do the summing algorithm again in the next level until getting the total value in the root. There is a binary tree example shown in Figure 3.2. In the beginning, there are eight elements with values 0 to 8. In the lowest level, every two nodes add together and store their result in the parent node. In this study, there is no type limited for type of tree and size of input data.
3.1.2 Serial Computation

In the serial computation, the program will access the elements, and do the summary sequentially. As we can see in figure 3.2, the program access 0 and 1 first, and store the result 1 into their parent. Then, 2 and 3 are accessed and added, and result 5 is stored in their parent. After finished the first level, the second level will repeat the loop again. The program will be end after root getting total value 28. The code below shows the implementation of Serial computation with tree algorithm.

NumberofChild is defined as the type of tree. For example, if NumberofChild equals to 2, the tree will be binary tree. And the TotalNumber represents the total elements of input data. In the beginning of this program, malloc() create the memory space for input data, and we use for loops to initial the value of elements and determine the total level number.
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <time.h>

#define NumberofChild 2
#define TotalNumber 1024*1024*4*16

int TotalNumber;
int *inputData;
int countofLevel=1;
int count=1;
int countforParent=0;

float totaltime;

int main() {
    // create the memory space for input data
    inputData=(int *) malloc(TotalNumber*sizeof(int));
    struct timeval start, end;

    //set value for each element
    for(int i=0; i<TotalNumber;i++){
        inputData[i]=1;
    }
    count=NumberofChild;

    //find the number of tree's level
    while(count!=TotalNumber){
        count=count*NumberofChild;
        countofLevel++;
    }
    count=1;

    gettimeofday(&start, NULL);
    //while for each level
    while(countofLevel>0 ){
        //for loop for each sub tree
        for (int i=0;i<TotalNumber/count;i=i+NumberofChild){
            for (int j=1;j<NumberofChild;j++){
                inputData[countforParent]=inputData[countforParent]+inputData[i+j];
            }
            countforParent++;
        }
        countofLevel=countofLevel-1;
        count = count*2;
        countforParent=0;
    }
}

In the major part of this code, while loop will go through each level in tree structure, there are two for loops under the while loop for each subtree and child. The figure 3.3 shows the structure of those for loops in binary tree with size of 4. The value of each element is got from memory and added together. The result will be return to memory, and replace the old elements. After finishing those for loop, the while loop will move to next level and repeat the same process again.

Figure 3.3: The Structure of Serial Program
Moreover, we also need code to extend the tree date for any tree size. There are functions called `getLast()` and `ArraySub()` which are designed for this problem. As shown in figure 3.4, it is an example of solution for binary tree. The node 4 will be added to root first, and the rest of elements will form a perfect binary tree and do the computations as explained above.

![Figure 3.4: Solution for Unperfect Tree](image)
3.1.3 Parallel Computation

3.1.3.1 Implement of Pthread

The full name of Pthread is POSIX ("Portable Operating System Interface for Computer Environments") thread. The definition created by IBM is that “is an interface standard governed by the IEEE and based on UNIX. POSIX is an evolving family of standards that describe a wide spectrum of operating system components ranging from C language and shell interfaces to system administration.”.

The introduction chapter shows the basic information and applications of thread. However, the thread safety issue become a big problem for software development. For example, multiple different threads access the same memory, and return some unexpected result. Therefore, Pthread uses MUTEX Function to keep threads’ safety. In order to use pthread library, the c code has to import pthread.h.

The methods for initialing input data and getting number of tree’s level are same as the methods for serial program. Then, we use pthread method to replace serial one. As the code below, the first for loop is creating threads for each subtree through Pthread_create() function. And when each thread finish their task, pthread_jion() function is used to terminate those threads. Before join those thread together, pthread_exit() is used for termination of calling thread.
```c
for(t=0; t<smallNumber; t++){
    pthread_create(&tid[t], NULL, Array, NULL);
    CheckThread=CheckThread-1;
}

for(t=0; t<smallNumber; t++){
    pthread_join(tid[t], NULL);
}
```

As shown below, each thread function use `pthread_mutex_lock()` and `pthread_mutex_unlock()` to protect the data in each thread from “Data Inconsistencies”. The “Data Inconsistencies” situation is called race condition which is occurred when multiple threads access the share memory at the same time. In this situation, the order of threads is not controllable, and the result become to unexpectable. `Pthread_mutex_lock` is like a door with locks. When the first thread access the memory, the lock will be activated, and the memory will become unavailable for other threads until unlock the door.
pthread_mutex_lock(&mutex);
//printf("\n\nsubnumleave=%d\n",subNumLeave);

// if the remainder of NUM_LEAVE/NUM_BRANCH is not 0
if(subNumLeave%NUM_BRANCH !=0 && i==0 && stest==1){
    GetLast();
}

//sum of tree branches
a[i]=a[NUM_BRANCH*i];
for(t=1;t<NUM_BRANCH;t++){
    a[i]=a[NUM_BRANCH*i+t]+a[i];
}
//printf("a[%d]=%d @ Pthread%d at %d
with %d\n",i,a[i],i,NUM_BRANCH,NumberOfThread);
i++;
ThreadEnd=ThreadEnd-1;

...

pthread_mutex_unlock(&mutex);

3.1.3.2 Implement of CUDA

CUDA is the “parallel computing platform and programming model”
developed by NVidia. Before using CUDA, there was a similar platform called
GPGPU, but it is hardly to use without professional skills. Therefore, CUDA, as the
simpler and platform “CUDA enable GPU” replaced the old GPGPU. CUDA use
C/C++ as its primary language, and save a lot of time for users from learn a new
language. Moreover, CUDA are using heterogeneous computing structure, and CPU
and GPU will work together when executing CUDA programs.
The CUDA function used in this study is formed by two parts. The first part is main function, and it executed on host (CPU). Meanwhile, the Kernel function is created as the second part and processed on GPU. In order to understand where the function will be executed, the identifier is introduced into function. In this function, __Gloable__ is used to make the kernel function called form the host and executed on GPU.

In the beginning of the main function, the space is created and initialed by the code as shown below. Not like the method for serial function, the variable of input data is created as pointers, and use cudaMemcpyHost, and cudaMemcpy to create space on host and device.

```c
/*allocate and initialize the matrices*/
float *dA, *dC;
float *hA, *hC;

/*set up memory on the device and host*/
int size = N * sizeof(float);
cudaMallocHost((void **)&hA, size);
cudaMallocHost((void **) &hC, size);
cudaMalloc((void **) &dA, size);
cudaMalloc((void **) &dC, size);
```

In the function below, the input data will be created on host, and copied to GPU through cudaMemcpy(). The variable “ha” in the function represented as the input data memory address on the host, and the variable “da” is the input data memory address on GPU. The figure 3.5 shows the processing flow of CUDA function.
After copy the data to GPU, method Tree

\[
\text{Tree}^{\text{numberofBlock, numberofThread}} (dC, dA)
\]

is used to call kernel function. The numberofBlock represents the total number of thread blocks, and numberofThread is the number of threads in each block.
/*transfer to device*/
cudaMemcpy(dA, hA, size, cudaMemcpyHostToDevice);

/*launch kernel*/
cudaEventRecord(start);
Tree<<<numberOfBlock, numberOfThread>>>(dC, dA);
cudaEventRecord(stop);
/*transfer to host*/
cudaMemcpy(hC, dC, size, cudaMemcpyDeviceToHost);

After finishing the function, we need to copy the data back to host, and free
and reset the memory on device like the following functions do.

/*Free host and device memory*/
cudaFree(hA);
cudaFree(hC);
cudaFree(dA);
cudaFree(dC);
cudaDeviceReset();
return EXIT_SUCCESS;

In the kernel function, the GPU uses the block ID and thread ID to access the
memory, and it is same as the for loop on host as shown below.
The for loop in C is shown below.
For (int i =0; i<n; i++) {...}
The kernel loop is shown below.
int i = blockIdx.x*blockDim.x + threadIdx.x;
If (I < n) {...}
/*CUDA kernel*/

/*assert in GPU*/
__global__ void Tree(float *C, float *A)
{
    int tx = typeOfTree *(blockIdx.x*blockDim.x + threadIdx.x);
    float sum=0.0;
    if (tx<N){
        for(int i =0; i< typeOfTree ;i++){
            sum = A[tx+i]+sum;
        }
        //printf("sum = %f  tx = %d\n",sum,tx);
        C[tx/typeOfTree ]=sum;
    }
}

3.2 Performance Evaluation and Analysis

In this section, the results of serial and parallel computation’s performances on three platforms are represented. The execution time is used to measure their performance. If the program took longer time for its execution, the performance of it will be lower. Based on the results of those functions, problems and performance effect factors will be discussed in following sections. And, the current statement of this study and future plan is also included. Moreover, the GPU and CPU’s information is shown below.
device # : 0
=======================
name : Tesla K40c
memory : -768MB
version : 3.5
max threads/block : 1024
max thread size : 1024x1024x64
max grid size : 2147483647x65535x65535
shared mem/block : 48KB
constant memory : 64KB
wrap size : 32
max mem pitch : 2147483647
registers/block : 65536
clock rate : 745MHz
texture align : 512

Memory Info:
free : -866MB
total : -768MB

GPU Information shown above.

Figure 3.6: CPU Information (virtual machine)
3.2.1 Serial Program

There is the result of the serial computation shown below with binary tree example. A long with the increasing of size of input data, the execution time of serial computation is increased. It is also easy to find that the execution time is raised when the size become large. When the input data’s size larger than 33554432, the execution rising speed become faster than before. The reason must relate to the size of memory. When the input data’s size is large enough, it will use multiple memory units like different level of cache or main memory, and waste more time on accessing memories.

![Performance Change of Serial Computation](image)

Figure 3.7: Performance of Serial Computation

The Function now can deal with all kinds of trees like binary tree or tries. Although the potential of speeding up of serial computation is smaller than parallel computation, the performance can be improved by optimize the structure of coding like the way of accessing memory, using more pointer.
3.2.2 **Pthread Computation**

The result of pthread function is shown below in figure 3.8, and the performance is not what I expected. As we can see from figure 3.8, the execution time keeps going up when the size of input data raised. The result looks like correct. However, when we compare it with the serial computation’s results, it is easy to find from figure 3.9 that it takes much time than serial one.

![Figure 3.8: the Performance of Pthread Computation](image-url)
Figure 3.9: The Performance Difference between Pthread Computation and Serial Computation

In general, the pthread algorithm should be faster than serial one. The bed memory accessing structure in my pthread function is the main reason for low performance. Every thread uses the same memory area, and each thread is locked by pthread_mux_lock() function to prevent it from racing condition. However, because of the bed structure, threads are executed serially instead of parallelly. At the same time, the cost of using thread is much higher than serial function. Therefore, the performance of pthread is lower than serial one.

The function now can return the correct result, and it also can handle the different situations like what the serial function does. Moreover, we can also set the number of processing threads at one time. In the future, it still needs to be optimized for improving the performance.
3.2.3 CUDA Computation

As shown in figure 3.10, the result and performance of CUDA function is same as my expectation. Along with the increasing size of input Data, the execution time is just increased slightly. Because thousands of threads executed simultaneously, the increasing input data doesn’t have too much effects on the performance. The CUDA function in this study can do all implements from serial computation and pthread computation with much higher performance.

Figure 3.10: The Performance of CUDA Computation and Serial Computation
Chapter 4

CONCLUSION

Along with the development of technology, people need their computer become faster and faster. Therefore, people have kept trying their best for improving computer performance. After decades’ research and development, people find the parallel computation is best way to reach the high performance of computation because the single high speed processor will bring a lot of heat and power consumptions.

In this study, we used the tree algorithm as the basic computation structure to get the total sum for input data. In order to compare the different performance of different parallel programming technologies, we introduced three different computation technologies which are serial computation, pthread computation, and CUDA computation. The serial and pthread computation is implement on CPU, and the CUDA use the heterogeneous computing which CPU and GPU.

After creating those code and programming in this study, we find a good computing algorithm is very important for improving the computer performance. In the serial computing, we need to find a good way for memory accessing because a bed memory accessing will slow down the program’s execution speed even the output is correct. Pthread computing is the worst part in this study. Even the output value is correct, the performance is extremely low. For solving this problem, we need to figure out how to avoid sharing the same memory address by multiple threads at the same time.
REFERENCES


Blaise Barney (2016), Introduction of Parallel Programming,
URL: https://computing.llnl.gov/tutorials/parallel_comp/.


Lu, Chaojun (n.d.), *Thinking and Method of Computer Programming*, (Ch 10.3.9.3),
URL: https://wizardforcel.gitbooks.io/sjtu-cs902_courseware/content/index.html.

Gordon D. Logon (2002), Parallel and Serial Processing,

Peter Pachoco (2010), *An Introduction to Parallel Programming*, Morgan Kaufmann,
URL: https://eclass.uoa.gr/modules/document/file.php/D186/%CE%8E%CE%BB%CE%B7%202011-12/1.%20PachecoChapter_1.pdf.

NVDIA.COM (n.d.), *What is GPU Accelerated Computing?*

NVDIA.COM (n.d.), *What’s the Difference Between a GPU and a CPU*,


Wes Kendall (n.d.), *MPI Tutorial Introduction*, MPI Tutorial,
URL: http://mpitutorial.com/tutorials/mpi-introduction/

Shao-Ching Huang (2013), *Parallel Computing and OpenMP Tutorial*,
Dr. HE, Yun (Helen) (2011), *Introduction to OpenMP*,
URL: https://www.nersc.gov/assets/Uploads/XE62011OpenMP.pdf.


Tutorialspoint.com (n.d.), *Data Structure and Algorithm – Tree*,

IBM (n.d), *What are Pthread?*, IBM Knowledge center,

Xiaoming Li (n.d.), *The CUDA Programming* (PowerPoint Slide)