Department of Computer Engineering and Information Technology
Amirkabir University of Technology

B.Sc. Thesis

Design and Implementation of a Real-Time Object Tracking system

Submitted in partial fulfillment of the degree

Bachelor of Science
in
Computer Engineering

Submitted by
Sareh Soltani

Conducted in
Aolab (The Academy for Amirkabir IoT Laboratory)

Under the guidance of
Prof. Bahador Bakhshi

June 2019

Abstract

In recent years, IoT technology has grown significantly and has been able to meet diverse and complex needs in various fields. One of the applications of the Internet of Things is real-time tracking. The positioning and tracking system has provided reliable solutions to ensure the safety of people and vehicles, and also has a significant impact on optimizing the quality of monitoring. This system can be used for a broad range of applications such as traffic management and vehicle tracking/anti-theft system, and finally, traffic routing and navigation. it can be applied in many business cases, like public transportation, so passengers can track their buses and trains by following the vehicle account on social networks.
In this project, a system was developed to accurately determine the location, direction of movement, and speed of moving objects at any given time using SIM808 Module. In this system, each object is equipped with a GPS (Global Positioning System) module that receives its location every two minutes from the satellite and sends it to the software servers via the GSM modem. Software servers analyze the information after receiving it. In this part of the project, a web application was developed to process the transmitted data and then store it in a database and finally convert the stored information to visible to users. This allows you to see the speed, direction of movement, and the object's locations on the map. Furthermore, a heatmap was implemented to show the frequently visited places by users.

Contents

List of Figures

Chapter 1

Introduction

The concept of the Internet of Things refers to objects with a unique identity and the ability to transmit data over the network, without the need for human interaction and intervention. Its primary purpose is to make objects intelligent and provide a platform through which objects can send and receive information. The Internet of Things (IoT) refers extensively to the development of computing capabilities and network communications of objects, devices, sensors, or any other item that is not typically considered a computer. These smart objects can collect, analyze and manage data remotely [1].

The Internet of Things is a wide range of sensors and actuators that measure and process various environmental conditions. In recent years, IoT technology has grown significantly and has met many complex needs in various fields. Due to the expansion of new technologies, the production of intelligent sensors, the growth of communication technologies, and the complexity of requirements, the Internet of Things has gained a lot of power. It has led to the expansion of intelligent systems in the environment [2].

These systems must interact with each other to have a positive impact on the environment. IoT-based technologies have different requirements compared to other technologies. Typically, these systems have less memory, power consumption, and bandwidth than other systems. Most smart systems are battery-powered and located in a remote location where they cannot be charged continuously. As a result, the power consumption and coverage range for these systems, especially those implemented at the macro level, such as smart farming, smart city and home, and tracking issues, is very important. There are several wireless communication protocols, each with its unique characteristics.

The Internet of Things is a new communication platform for communicating between intelligent objects. The introduction of this platform has provided

new facilities for solving problems, such as locating and tracking moving objects from vehicle attacks at the city, region, or country level. The Internet of Things (IoT) is rapidly gaining access to telecommunications scenarios, and it is expected that the exchange of information in global resource chains will facilitate.

Widely, Internet of Things can be used as the mainstay of pervasive systems and the activation of intelligent environments for ease in identifying objects and retrieving information from the Internet. From a conceptual point of view, the Internet of Things relies on three principles related to intelligent objects' ability: Ability to identify, transfer and interact either among themselves or with other users. Objects are usually classified either individually or as a member of a category.

One of the most important issues today is the tracking of moving objects, which refers to the immediate tracking of a specific moving object's current position. Moving object tracking systems is a solution to many problems, including security issues. It is a technology used to determine the location of an object.

One of the important applications of IoT is tracking systems that are used in various technologies. Tracking systems were developed for the transportation industry. One of the basic needs of the owners of this industry is to check the position of vehicles. The earliest systems for locating were passive systems that stored information in memory and could only be accessed when a vehicle was available. These systems are not suitable for real-time applications because they need to provide information to the user immediately. For solving this issue, active systems were created that allow the use of in-vehicle hardware and remote tracking servers.

Today, the safety of people and vehicles has become a public concern. The tracking system has provided reliable solutions to ensure people's and vehicles' safety. It has a significant impact on optimizing the quality of monitoring and management of public transportation, vehicle movements, people (children and the elderly), or It has every other moving object. Tracking system is a technology that makes it possible to determine the exact position and track of people, vehicles, or any other moving object using various methods such as the Global Positioning System [3].

In addition to vehicles, tracking systems also play an important role in remote monitoring and environmental monitoring applications. For example, tracking animals, humans, and locating objects are some of this system's applications. In human monitoring, this system can be beneficial for the elderly who have certain diseases such as Alzheimer's and are more likely to get lost, or for the safety of children. Families can use this system to find the position of their elderly or child [4].

The system we have implemented in this project can be used in various cases. One of the applications that can be imagined for the Internet of Things is implementing a system that can determine the exact position and path of each moving object at any time. In this project, we intend to build a tracking system that can identify a moving object's exact position and path.
In this project, our communication will be one-way in that the coordinates of the moving object are continuously measured by the GPS module and sent to a server. This module is continually connected to the satellite to get coordinates. GPS data is sent to the Arduino. Finally, the GSM modem sends this information to the software server. In this project, the software servers analyze the information after receiving it, and we will not have a request from the server since our communication will be one-way. In this part of the project, web-based software will be developed to process the submitted information, store it in the database, and finally convert the stored information into a display for users. An application written using stored data displays the location on a map. In this way, the object or person can be found at the current time. In the end, the current position and direction of the person will be displayed on the map. Another application of this system is to obtain high-traffic locations by analyzing the collected information.
In the continuation of this dissertation, we talk about the general architecture of the tracking system and its components. Then, we introduce the implementation method of this system using the mentioned components. Finally, we talk about the results of the system implementation and what can be done in the future based on this project.

Chapter 2

Tracking system components

The main goal of our project is to design and implement a system that can determine the exact position of each moving object at any time. The mentioned system should be cost-effective in addition to proper performance. To design such a system, we must first identify the system requirements and the overall architecture of the system we want. Then we use this architecture to implement the appropriate modules. In this chapter, in section 2.1, we first explain the general design of the tracking system and then in section 2.2, we introduce the components used in this design.

2.1 System design and architecture

In this section, we will design our system. According to the project's requirements, we must specify the transmitter and receiver modules, communication protocol, and application program for displaying information. The primary purpose of a tracking system is to track a specific object and gain its path. The tracking system provides information about the current location and speed of the object.
In this project, our communication has been one-way, in which the coordinates of the moving object are continuously measured and sent to a server. Then, the necessary processing of this information is performed on the server-side. According to the above explanations, we can mention three main parts in this system [5]:

- Obtain the location of a moving object using the GPS module
- Send location information to software servers by GSM modem
- Store location information on the server-side and implement an application to display the object's path on the map

As we have seen, the architecture of our system has four main parts. The first part is about locating the object from the satellite using the GPS module. The second part is related to sending the received information to the server using a GSM modem. The third part is the development of an application that uses the received data to display the location of the object on the map.

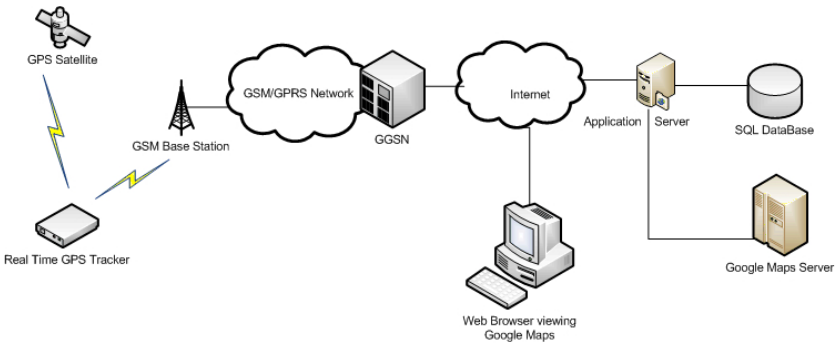

Figure 2.1: Block Diagram of Tracking System [6]

Figure 2.1 shows an overview of the designed system architecture and the relationship between its parts. For selecting the modules, it is necessary to know the task of each section accurately and choose the desired module for it [7].

- In the first part, we need to measure the location of the object continuously. As soon as the object moves, the GPS module consistently receives the moving object's coordinate from the satellite. The signal received from the satellite is weak, so we must use an antenna to amplify the desired signal, and at the end, sends the amplified signal to the Arduino board.

- In the second part, the information received from the GPS module is sent to the server by the GSM modem.

- Software servers analyze the information after receiving them. Our communication in this project is one-way, and we will not have a request from software servers. In this part of the project, web-based software will be developed to process the submitted information and store them in the database. In the last section, this information will be displayed in the designed web page.

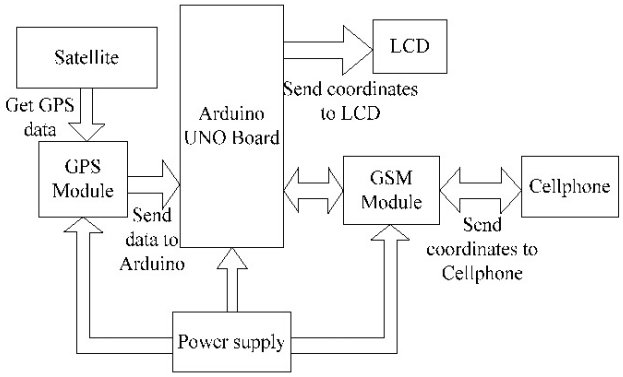

Figure 2.2: Architecture of Tracking System[3]

2.2 System components

In the previous section, we defined the system architecture. Now we will express and introduce the components of this architecture in detail.

2.2.1 Hardware components

The hardware components used to implement this system are:

- Arduino module
- SIM808 module
- GPS antenna
- GSM modem

2.2.1.1 Arduino module

Arduino is an open-source microprocessor suitable for writing applications that interact with the environment and objects outside. This board is ideal for prototyping, and its software and hardware design are freely available to all people. Any interested person, even with a little knowledge and experience in electronics, can use Arduino to do their projects.
Arduino has a simple programming environment that anyone with a little knowledge of C and C++ can program in this environment and run the program written in Arduino. Various sensors can be connected to the Arduino microcontroller and controlled. The microprocessor used on the Arduino board is based on the Arduino programming language and does not require

any additional software or compiler for coding.
Figure 2.3 shows the Arduino Uno board:

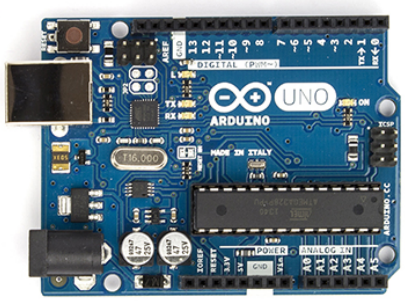

Figure 2.3: Arduino Uno board

2.2.1.2 SIM808 module

The 808 SIM module is a combination of GSM / GPS PRS 9 module and GPS module with 1900 MHz support for data transmission, SMS and voice calling. This module has a SIM card socket in which the SIM card is inserted. Using the GSM / GPRS modem and the 808 SIM module, you can exchange data over the GSM network via the USB interface and access the information of devices located in remote locations.
In figure 2.4, You can see this chip.

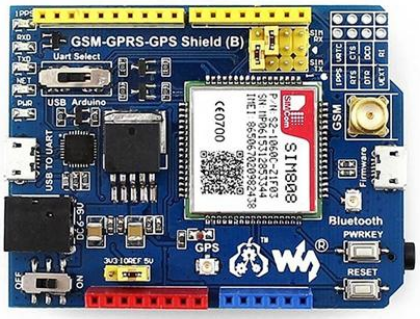

Figure 2.4: SIM 808 module

2.2.2 Software components

2.2.2.1 Arduino IDE

The software used for programming is Arduino software, which you can see in Figure 2.5. Using the C language, you can write the required program, and after compiling, the generated hex code is uploaded on the Arduino. There are different libraries that make coding easier. The program is written using this software to receive data from satellite and send it to a mobile phone.

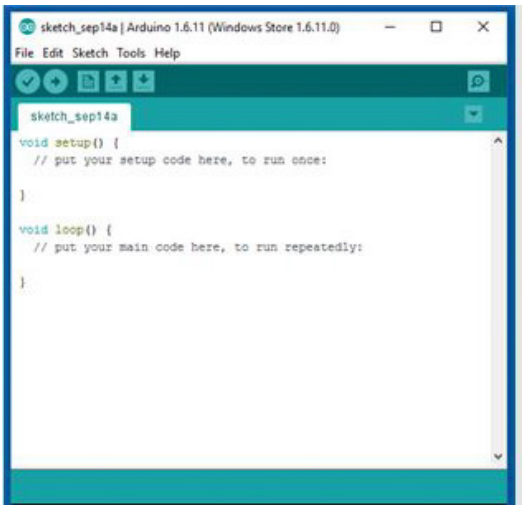

Figure 2.5: Arduino IDE

Chapter 3

Implementation and Performance Analysis

3.1 Analysis of Naive Template Matching

In this section, we'll analyze the implementation and the performance of the *Naive Template Matching* algorithm. First of all, let's consider the serial implementation case.

3.1.1 Bitmap Images

For this project, we've used the *bitmap* image library by *Arash Partow* which is accessible here.

3.1.2 Serial Implementation of NTM

The serial implementation of this algorithm consists of 2 main loops to iterate over image pixels. There is also a need for each pixel we are iterating through, to calculate the *Sum of Absolute Deviations* which is the similarity measure between the main image pixels and the template image. Since the task is mainly focused on counting the number of occurance inside the main images, we can obtain this by finding the minimum value of *SAD* while iterating through the main image pixels and counting the number of occurrences of the found minimum inside the main image matrix.

3.1.3 Timing the Serial NTM

We can obtain the elapsed time for the serial iteration using the *chrono* library.

3.1.4 CUDA Implementation of NTM

3.1.4.1 Compute Sad Array Kernel

The image data is being stored in an 1-D array. Thus, the best choice for CUDA kernel launch parameters would be to set grid dimensions as $(image\_width/block\_size\_x, image\_height/block\_size\_y, 1)$. The main purpose of this choice is not the way we have stored the data, it actually depends on the input data. Since we are dealing with images it is more suitable to have 2-D blocks of threads. Further explanations focus on the kernel implementation. Initially, each thread finds its own global 2-D identification as rows and columns. Then, each thread virtually sees a kernel (template) around it self. This is the core of the parallelization section. All threads can see all of the iterations in one place. We then compute the SAD of each pixel and load it inside an SAD array. The next section describes the reason to do so.

3.1.4.2 Find Minimum in Array Kernel

We can obtain the best possible match by finding the minimum SAD in SAD array. We'll do so using shared memory to speed up the process. Each thread helps loading the data inside a block into the shared memory of that block. We then use *fminf* to find the minimum inside each block. Since we are using the shared memory, we need to do a *reduction* to find the minimum across different blocks. We can obtain this reduction task defining a *mutex* to control the global memory write by first thread of each block.

3.1.4.3 Find Number of Occurrences

Since the main task is to find the number of occurrences of template image in main image, we should count the number of occurrences of minimum SAD in the SAD array. To speed up the process we've used shared memory again. The intuition of using shared memory is exactly the same as the previous section.

3.1.5 Launch Parameters on Test Cases

The test has been done on three different sizes of images. One of the medium sized images is a *bitmap* image with dimensionality of $1112 * 1500$. The template image is a $116 * 116$ image in this case. Considering a block size of 1024 on a *Nvidia 850m GTX* (compute capability of 5), the launch parameters are provided below:

	X	Y	Z
Grid Dimensions	38	47	1
Block Dimensions	32	32	1

3.1.6 Occupancy Analysis

Before getting into the details, it is important to mention that maximum number of active warps in theory is 64. *Nsight* profiler provides great details on the number of active warps per SM. In this case, there are 63.83 active warps on the first kernel (Compute Sad Array Kernel). The occupancy can obtained by dividing the number of active warps to the number of maximum warps available:

$$occupancy = \frac{63.83}{64} = 99.74\% \tag{3.1}$$

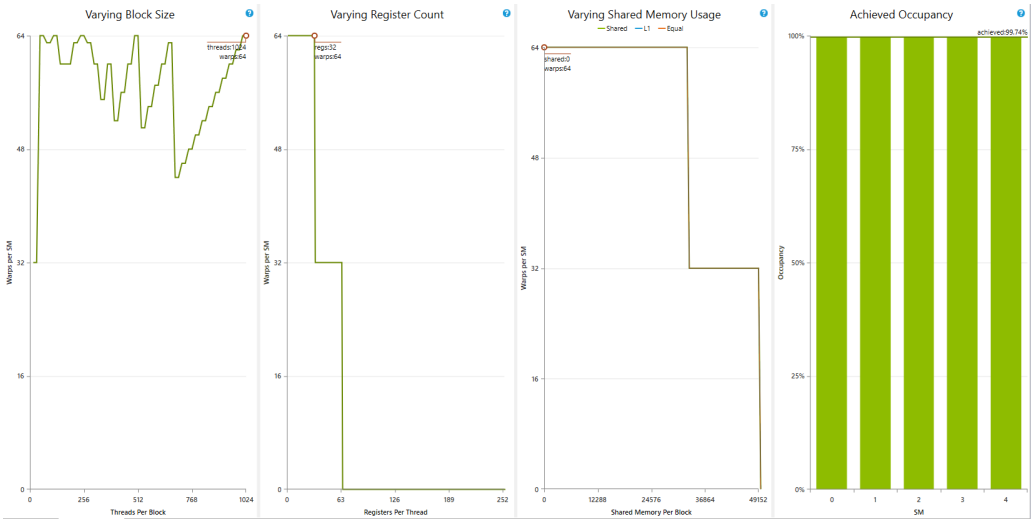

Figure 3.1: Illustration of varying the size of blocks, registers and shared memory on the achieved occupancy.

According to figure 3.1, varying the number of threads per block might yield the same result in other scenarios. The circled point shows the current number of threads per block and the current upper limit of active warps. Note that the number of active warps is not the number of warps per block (that is threads per block divided by warp size, rounded up). If the chart's line goes higher than the circle, changing the block size could increase occupancy without changing the other factors. Also, increasing the number of variables (registers per block) will drastically decrease the occupancy of the first kernel.

3.1.7 SM Activity Analysis

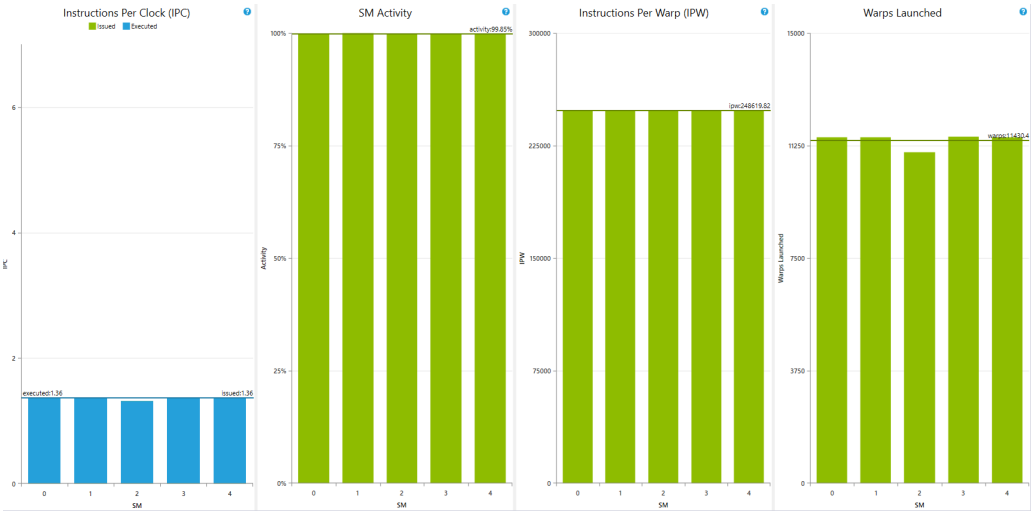

Figure 3.2: Issue efficiency and SM activity charts for the first (bottleneck) kernel.

The IPC chart demonstrates the achieved instructions throughputs per SM for both, issued instructions and executed instructions. The theoretical maximum peak IPC is a device limit and defined by the compute capabilities of the target device.

3.1.7.1 Issued IPC

The average number of issued instructions per cycle accounting for every iteration of instruction replays. Optimal if as close as possible to the Executed IPC. As described in the background section of this document, some assembly instructions require to be multi-issued. Hence, the instruction mix affects the definition of the optimal target for this metric. In this case, the issued IPC throughput is minimum since there weren't any multi issue instructions in the first kernel.

3.1.7.2 Executed IPC

The average number of executed instructions per cycle. Each warp scheduler of a multiprocessor can execute instructions independently, a target goal of executing one instruction per cycle means executing on average with an IPC equal to the number of warp schedulers per SM. The maximum achievable target IPC for a kernel is dependent on the mixture of instructions executed.

3.1.7.3 SM Activity Chart

The second chart provides information on the activity of each SM during the kernel launch. A multiprocessor is considered to be active if at least one warp is currently assigned for execution. An SM can be inactive - even though the kernel grid is not yet completed - due to high workload imbalances. Such uneven balancing between the SMs can be caused by a few factors: Different execution times for the kernel blocks, variations between the number of scheduled blocks per SM, or a combination of the two. In this case, non of the above has occurred and the activity result is almost 100%.

3.1.7.4 IPW Chart

One of the most common patterns for varying IPWs is conditionally executed code blocks. IPW can be useful while having variations in SM activity chart which in this case, non of these have occurred.

3.1.8 Speedup Analysis

In this section, we'll analyze the possible theoretical speedup. For this purpose, we'll calculate the serial computation time for the *collection* and *collection_coin* images as described above. This table describes the average speedup of parallel naive template matching algorithm compared to serial implementation. As can be seen in the figure 3.3, the speedup obtained on GPU is dedicated to many different parameters. In order to calculate the theoretical speedup a direct use of *Amdahl's law* doesn't fit in this problem. One might say that the number of cores in this case is N (Same as the number of threads), but GPU does not have as many physical cores as the number of threads that can be launched. Additionally, significant portion of kernel time is begin spent just waiting for the data to be *read/written* from/to global memory. Also, the frequency and operational power of CPU cores is higher than GPU cores. In this section, we'll use a modified version of the Amdahl's law [**?**]:

$$S = \frac{1}{(1-p) + (k * p/N)} \qquad k = freq(CPU)/freq(GPU) \tag{3.2}$$

The core frequency of my Intel 4710HQ processor is 3490 MHz. The core frequency of each GPU core is 902 MHz. These information are extracted using beloved *CPU-Z* and *GPU-Z* programs. Replacing these information into 3.2 yields

$$S = \frac{1}{(1-p) + 3.86 * p/N} \tag{3.3}$$

since the size of input is constant in this law, we'll assume that 3.3 is correct for the third experiment. Let's suppose that $\frac{1}{4}$ of the main program is parallelized, thus $p = \frac{1}{4}$ and the number of threads to run is 1828864 (1786 blocks of 1024). Theoretical speedup will be

$$S = \frac{1}{(\frac{3}{4}) + 3.86 * \frac{\frac{1}{4}}{1828864}} = 1.33 \tag{3.4}$$

	Exp 1	Exp 2	Exp 3	Exp 4
*Image * Template*	(630 * 459) * (116 * 116)	(1203 * 460) * (116 * 116)	(1211 * 1500) * (116 * 116)	(3840 * 2160) * (214 * 214)
Serial NTM	10787 msec	22615 msec	92204 msec	1722131 msec
Parallel NTM	339 msec	644 msec	1975 msec	27983 msec
Parallel NTM + Data Transfer	345 msec	665 msec	1980 msec	28977 msec
Achieved Speedup	***31.82***	***35.11***	***46.68***	***61.54***

Figure 3.3: Computation and data transfer time for naive serial and parallel implementation.

3.1.9 Performance Analysis

In this section, we'll analyze the performance of the *Compute Sad Array Kernel*. Firstly, lets find the arithmetic intensity of this kernel. Arithmetic intensity I, also called *operational intensity*, is the ratio of *arithmetic operations* or *work* (W), to the *memory traffic* (Q) [**?**]:

$$I = \frac{W}{Q} \tag{3.5}$$

and denotes the number of operations per byte of memory traffic. When the work is represented as *FLOPS*, the arithmetic intensity will be *FLOPS/Byte*. According to the *Naive Roofline Model*, the *attainable performance* is bound either by the *peak performance* or *memory bandwidth * arithmetic intensity*. In this case, the operational intensity can be obtained by following the memory access patterns and the work being done in the main kernel: It is given that the *single precision* processing power of a *Nvidia 850m GTX* is 1155 GFLOPS [**?**]. Given these results, the kernel is actually memory bound since $0.5 < 1155/32 = 36$. The kernel arithmetic intensity should be at least 36 to be considered a compute bound kernel.

Statement	*Operations*	*# Global Memory Writes * bytes*	*# Global Memory Reads * bytes*	*# Constant Memory Reads * bytes*	*# Shared Memory Reads * bytes*
row, col	()()(* + * +)	0	0	0	`6 * sizeof(int)`
1st conditional	()()(< <)()	0	0	`2 * sizeof(int)`	0
1st loop	()()(<)()	0	0	`1 * sizeof(int)`	0
2nd loop	()()(<)()	0	0	`1 * sizeof(int)`	0
m_r	()()(*)()+ + + * +	0	`1 * sizeof(unsigned char)`	`1 * sizeof(int)`	0
m_g	()()(*)()+ + + * +	0	`1 * sizeof(unsigned char)`	`1 * sizeof(int)`	0
m_b	()()(*)()+ + + * +	0	`1 * sizeof(unsigned char)`	`1 * sizeof(int)`	0
t_r	()()(*)() + * +	0	`1 * sizeof(unsigned char)`	`1 * sizeof(int)`	0
t_g	()()(*)() + * +	0	`1 * sizeof(unsigned char)`	`1 * sizeof(int)`	0
t_b	()()(*)() + * +	0	`1 * sizeof(unsigned char)`	`1 * sizeof(int)`	0
SAD Write conditional	()()(*<)() +	`1 * sizeof(int)`	0	`1 * sizeof(int)`	0
Final Operational Intensity	5 / (6 * sizeof(unsigned char) + 1 * sizeof(int) = 0.5				
Nvidia 850m GTX DDR Bus Frequency	1001 MHz				
Nvidia 850m GTX DDR Bus Bandwidth	2 * 128 = 256 bits				
Nvidia 850m GTX Memory Bandwidth	32 GB/s				

Figure 3.4: Calculation of first kernel operational intensity and GPU memory bandwidth.

3.2 Implementation of FFT Based Template Matching

In this section, we delve into the details of *FFT* based template matching using *cufft* library from *Nvidia*. Note that this implementation is still in progress and there are multiple serial loops conducted in preprocessing step which slows down the process compared to the *naive template matching* procedure. As mentioned before, the overall procedure is to convert the spatial domain of input images to the frequency domain. In the frequency domain, we can apply a correlation based operation. This operation yields a signal of the same size as main image size. We then apply an inverse Fourier transform to turn the results back to the spatial domain. We then find the maximum of that signal and the number of occurrences of that maximum value in that signal. The procedure is well defined below[**?**]:

$$c = real(IFFT_{2D}(FFT_{2D}(main\_image).*FFT_{2D}(template\_image))) \tag{3.6}$$

according to 3.6, c contains a list of values which we are desired to find the maximum of it (for maximum similarity). To obtain this in CUDA, we've used multiple functions which are described further. Note that $*$ symbol shows the complex conjugate of the second signal.

3.2.1 Real to Complex Conversion

For this part, we have not focused on the performance. We use simple *for* loops to convert the image data matrix to the proper format of *cufft* library which is called *cufftComplex*. *cufftComplex* is a simple typedef of *float* type in C.

3.2.2 Padding Data

Before performing a *2-D Forward Fourier Transform* we are urged to perform a data padding on template signal to make it the same size as the main signal. Figure 2.1 illustrates this phenomenon pretty well. The duty of data padding is simple. Allocate new memory with a new size. Copy signals into their corresponding location in new allocated memory and set other places to 0.

3.2.3 2-D FFT Planning

cufft library uses the concept of *plan* to provide the baseline setup for the main APIs provided in the library. A plan can initiate a 1-D, 2-D or 3-D data processing API from *cufft* library. In this implementation, we need a setup for a 2 dimensional Fourier transform, thus we use *cufftPlan2d* to prepare the library for the main transformations.

3.2.4 Forward FFT Launch

In order to launch a forward *complex to complex* Fourier transform using *cufft*, we'll use *cufftExecC2C*. The constant *CUFFT_FORWARD* shows the direction of the Fourier transform which is *Forward* in this case.

3.2.5 Complex Conjugate Kernel

This kernel implements the conjugation procedure. The implementation is pretty straightforward. Each thread multiples the corresponding signal point's complex part to -1 and loads it back to the input signal.

3.2.6 Complex Point-wise Kernel

This kernel provides a point to point multiplication manner for each of the points in main and template input signals. It uses another kernel called *ComplexMul* to perform a *complex multiplication*. It then scales the result

using *ComplexScale*. Scaling or normalization is mainly done by dividing the signal values to the size of the signal.

3.2.7 Inverse FFT Launch

Finally, we'll perform an inverse Fourier transform on the result of the *Complex Point-wise Kernel*. The resulting signal is in the spatial domain.

3.2.8 Maximum Value and Number of Occurrences

To find the number of occurrences, we *serially* iterate over the resulting signal of the previous section. Note that these steps are not *100%* efficient and are still in improvement.

Chapter 4

Future Work

We admit the possible drawbacks and inefficiencies existing in this project, thus we have the following key improvement to-do list:

- Reorganization of the project structure, header files and .c files.
- Reorganization of the project source code for a C unified implementation.
- Usage shared memory techniques to improve the performance of the *NTM* implementation.
- Usage of highly-optimized image libraries such as *OpenCV* for *image rotation*, *image matrix extraction* and general *bitmap processing*.
- Implementation of CUDA kernels for real-to-complex signal conversions.
- Implementation of CUDA kernel to find the number of occurrences of template signal in main signal in *Fourier* based template matching.

Chapter 5

Conclusion

In this project, we analyzed various algorithms for the task of *Image Template Matching*. It is fair to say that there are already highly optimized libraries with template matching capabilities such as *OpenCV*, but as mentioned before, the main purpose of this project is to improve the understanding of the CUDA environment and parallel implementation of highly-demanded computer vision tasks such as template matching. We analyzed two main methods to perform this task:

1. Naive Template Matching
2. FFT Based Template Matching

For further read, please refer to methods such as *Stereo Matching*, *Image Registration* and *Scale Invariant Feature Transform*.

References

[1] M. Mukhtar, "*GPS based Advanced Vehicle Tracking and Vehicle Control System,*" International Journal of Intelligent Systems and Applications, Feb. 2015.

[2] S. Hussain Shah and I. Yaqoob, "*A survey: Internet of Things (IOT) technologies, applications and challenges,*" 2016 IEEE Smart Energy Grid Engineering (SEGE), Aug. 2016.

[3] Md. Rahman, J.Mou, K. Tara and Md. Sarkar, "*Real time Google map and Arduino based vehicle tracking system,*" 2016 IEEE Smart Energy Grid Engineering (SEGE), Aug. 2016.

[4] J. Saranya and J. Selvakumar, "*Implementation of children tracking system on android mobile terminals,*" Implementation of children tracking system on android mobile terminals, Apr. 2013.

[5] B. Bidabad and A. Tayebi, "*Design and implementation of vehicle tracking system using GPS/GSM/GPRS technology and smartphone application,*" 2014.

[6] M. Omar, H. Rashaand and T. Nicolae, "*Design and implementation of real time tracking system based on Arduino Intel Galileo,*" 2016 8th International Conference on Electronics, Computers and Artificial Intelligence (ECAI), Jun. 2016.

[7] T. Agrawal and M. Qadeer, "*Tracing Path with Arduino Uno using GPS and GPRS/GSM,*" 2018 International Conference on Computing, Power and Communication Technologies (GUCON), Sep. 2018.