UNIVERSITY OF CALIFORNIA
RIVERSIDE

Post-Processing Acceleration of OCT Data

A Thesis submitted in partial satisfaction
of the requirements for the degree of

Master of Science

in

Electrical Engineering

by

Ziying Deng

June 2022

Acknowledgments

I am grateful to my advisor, Dr. B. Hyle Park, without whose help, I would not have been here. He gave me the opportunity to do research and provided invaluable guidance throughout this research. His dynamism, vision, sincerity and motivation have deeply inspired me. He has taught me the methodology to carry out the research and to present the research works as clearly as possible. It was a great privilege and honor to work and study under his guidance. I am extending my heartfelt thanks to his wife, family for their acceptance and patience during the discussion I had with him on research work and thesis preparation.

I would also like to say thanks to my research lab members for both their help throughout this research work and for the friendship that made my graduate life colorful.

Finally, my thanks go to all the people who have supported me to complete the research work directly or indirectly.

I dedicate my thesis work to my family.

A special feeling of gratitude to my loving parents, Hai Deng and Wenqian Chen whose words of encouragement and all the support to the dream I am pursuing. Thank you for their love, caring and sacrifices and preparing me for my future.

I also dedicate this thesis to my beloved grandparents who have meant and continue to mean so much to me. Although some of them are no longer of this world, their memories continue to regulate my life.

ABSTRACT OF THE THESIS

Post-Processing Acceleration of OCT Data

by

Ziying Deng

Master of Science, Graduate Program in Electrical Engineering
University of California, Riverside, June 2022
Dr. Hyle Park, Chairperson

Optical coherence tomography (OCT) is an optical imaging method based on low-coherence interferometry. OCT uses light waves to take cross-section images of your internal microstructure such as the retina by measuring light backscattered from the sample. It can provide reliable and high-resolution images of biological tissues at different levels.

Real-time OCT can be a challenge due to the heavy computational load required to process acquired data streams. Reconstructing functional images with multi-functional OCT imaging requires additional processing, further increasing processing time.

So we introduced graphics processing unit (GPU) processing and some code optimization strategies of MATLAB to accelerate the image processing. Also, we implemented a web application to run MATLAB program remotely for the convenience of the researchers. It can allow multiple users to remote control MATLAB programs and do some basic operations through both mobile phone and computer.

In Chapter1, I will explain the OCT data processing workflow and describe the motivation of OCT data processing acceleration.

Chapter2 will mainly focus on the methods that I used to improve the performance of the MATLAB code and show the comparison of the total processing time.

In Chapter3, I will introduce the features of the web application as well as the structure of the whole application for both front-end and back-end.

Chapter4 is the conclusion part of the entire post-processing acceleration project, some future work will also be listed in chapter4.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction to Optical Coherence Tomography (OCT)

OCT (Optical Coherence Tomography) is an optical interferometric medical imaging modality widely used in clinical practice and research. By using low coherence interferometry, OCT extracts optical scattering information from subsurface tissue and generates depth-resolved tomographic images. When compared to traditional imaging techniques like ultrasound and magnetic resonance imaging (MRI), OCT provides high resolution and imaging depth, contributing to its potential as a clinical imaging tool. In most practical scenarios, real-time image acquisition, processing, and display are required. Because of the large amount of data acquired, image processing in an OCT system takes a very long time. This demands the use of high-speed computation to analyse acquired data and display images in real-time.

1.2 Introduction to OCT Data Processing Workflow

The typical data process flow for OCT is first reading raw data, subtracting the raw data with reference, and after subtracting it the spectrum will center on this zero line. Then we will apply calibration and dispersion. So the goal in the calibration iteration process is to force the phase lines to overlap so that the wavenumbers on x-axis are re-assigned as a function of array index which means the spectral fringes are interpolated by the new calibrated wavenumber array The reason why we introduce dispersion is that different wavelength have different refraction index in various optical component like fiber or lens. In OCT we have a reference arm and a sample arm so we can not make sure that they have the exact same optical component which cause different optical path length. Then we can get the depth profile by doing FFT for one single A-line and we utilize this results to form the intensity image, the intensity image is consist of 1024 A-lines. Then we process the other images like flow image and attenuation image.

For Spectral Domain OCT which is one of the frequency domain OCT, a spectrometer is used to record spectral fringes that result from the interference of a reference beam with light reflected from a sample. The grating will have diffraction and the spectrum with different wavelengths will not evenly spaced on the camera we use to detect. Since the wavelength mapping on the CCD array is uneven, as shown in Figure 1.3, the initial spectrum received will contain modulations with a range of frequencies instead of a sinusoidal modulation. Thus, the peak in the depth profile becomes broader, leading to blurred images and low SNR. In the case, wavelength calibration is therefore critical.

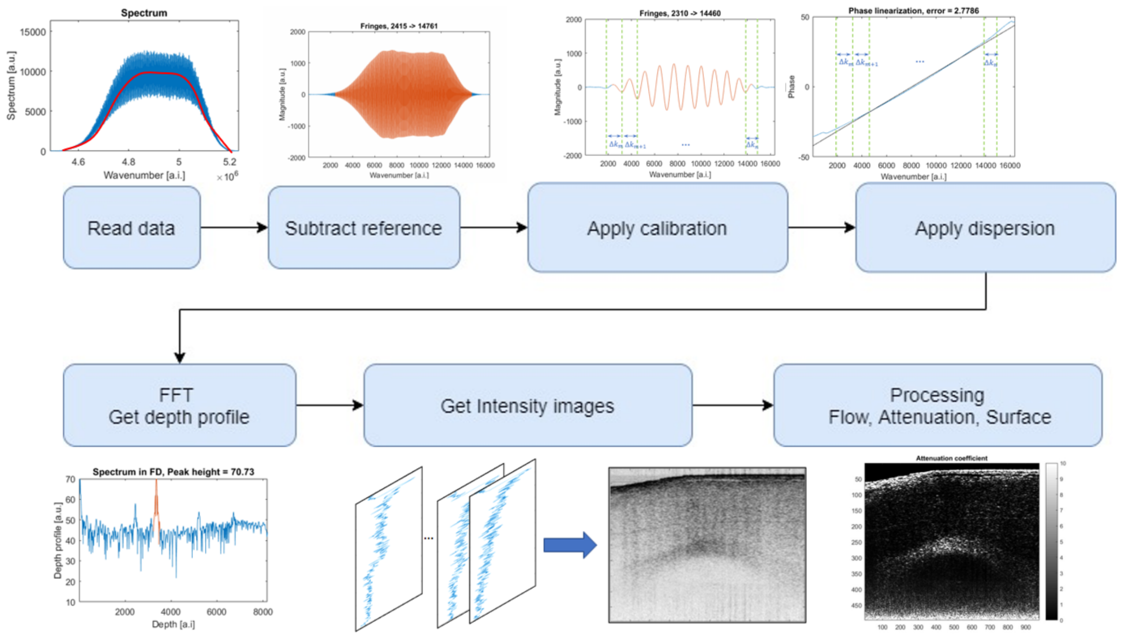


Figure 1.1: OCT Data Process Workflow

In swept source system which is another frequency domain OCT system[1], instead of a broad bandwidth source, a narrow band laser is swept across a broad spectrum. The mechanical wavelength tuning creates small variations in every wavelength sweeps, The swept-source laser and the data-acquisition hardware is instable, so it will result in a non-linear wavenumber sampling and a shift in the wavenumber range which is the k-space.

For the apply calibration part, the reflecting sample data was processed using an iterative auto calibration technique that allocates wavelengths based on the phase linearity of perfect sinusoidal modulation. For future data processing, an array with adjusted wavelength distribution is saved. So you can see the comparison before and after the wavelength calibration in Figure 1.5. Column A is the result without calibration, the peak highlighted in A2 is broad in width which indicates the detected spectrum is a result of the summa-

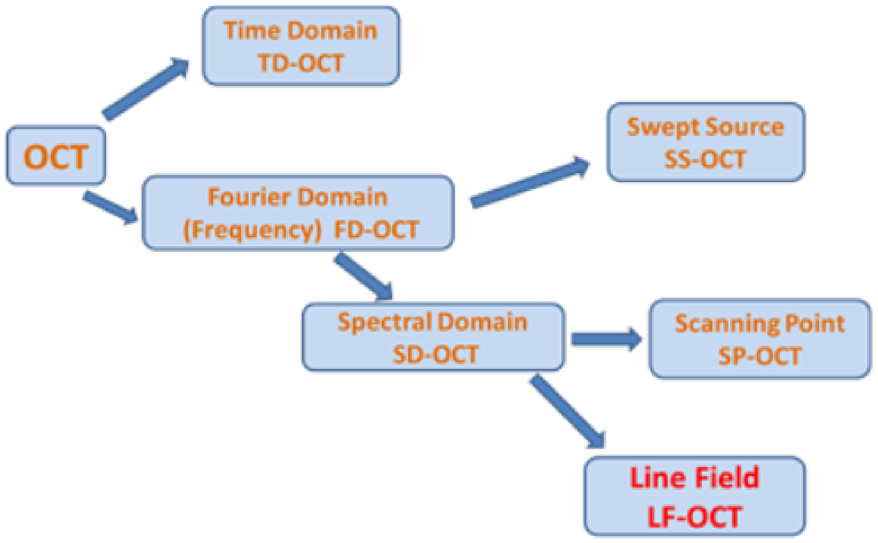


Figure 1.2: Different OCT Systems

tion of a wide range of multiple frequencies due to the unevenly k-space distribution. as demonstrated by the blue curve in Figure 1.5 (A4), the phase of interference fringes exhibits non-linearity when compared to the perfect linear black line. This black line represents an ideal phase estimation based on accurate wavelength assignment. The purpose of the calibration iteration procedure is to overlap the phase lines and re-assign the wavenumbers on the x-axis as a function of array index. The new calibrated wavenumber array interpolates the spectral interference fringes, and zero-padding is used to increase interpolation quality. We repeat this procedure until the error level reached a number smaller than a preset threshold which means calibrated phase curve is well aligned with the reference curve.

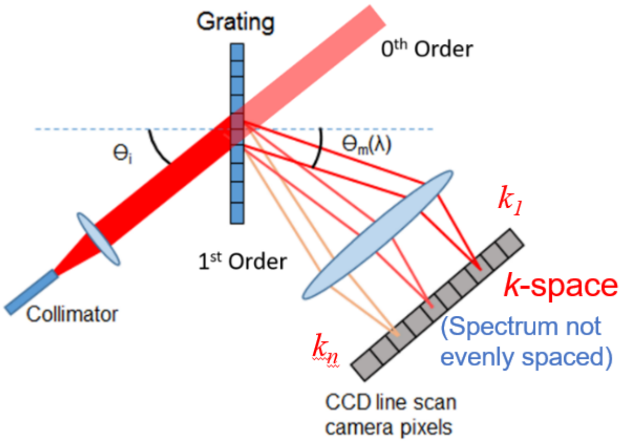


Figure 1.3: Unevenly Spaced Spectrum

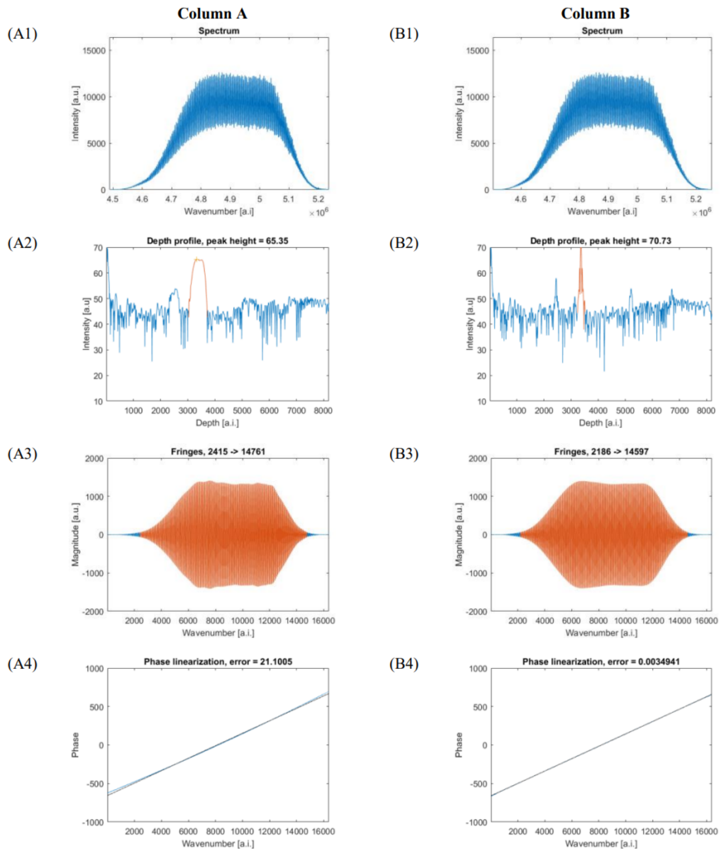


Figure 1.5: Initial Data (left) and Calibrated Data (right)

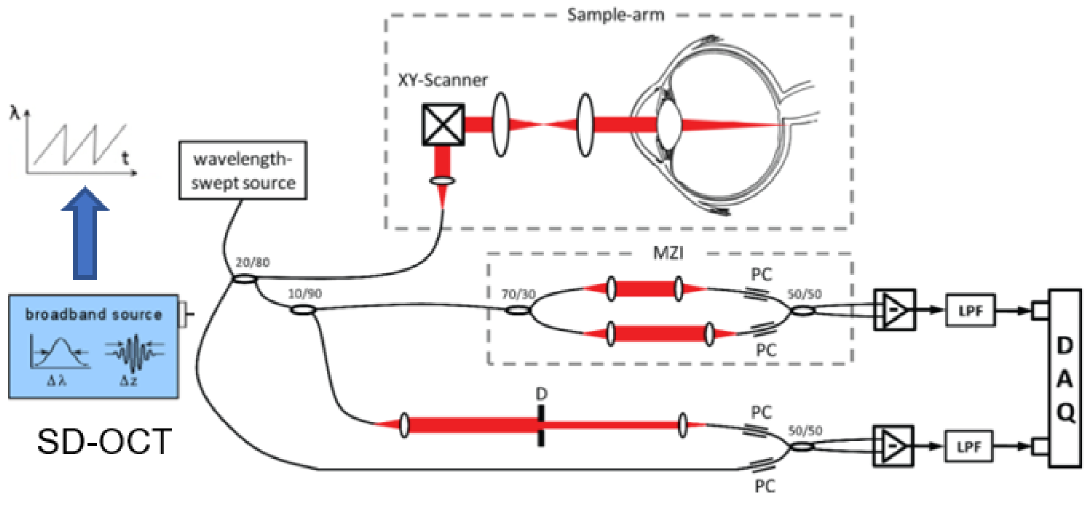
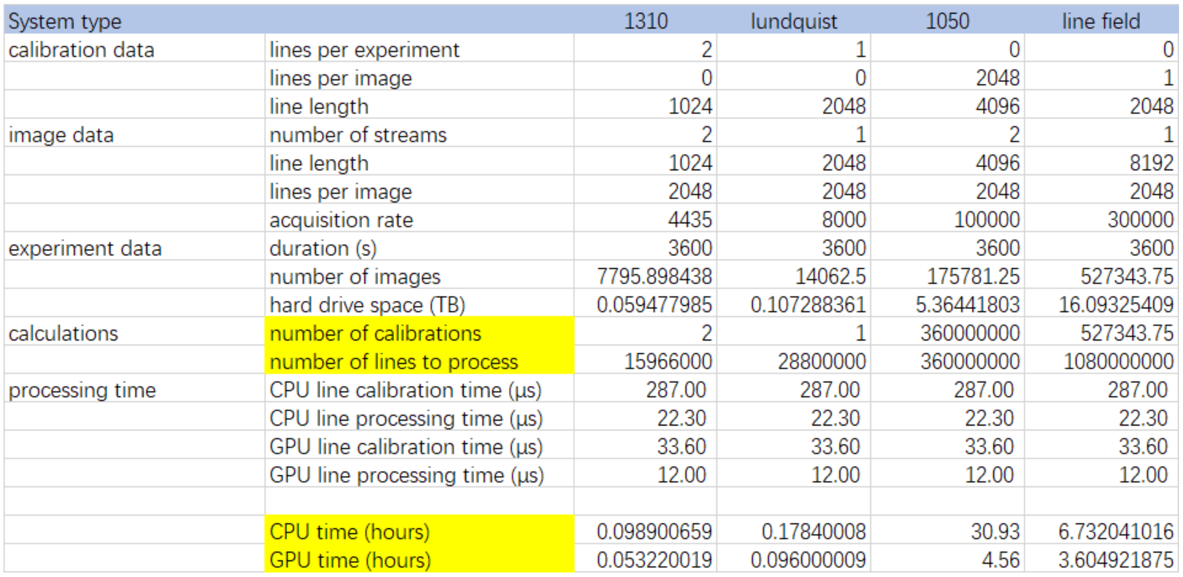


Figure 1.4: Swept-Source OCT (SS-OCT) Setup

1.3 Motivation

With the advances in hardware, the line acquisition rates of spectrometer based OCT system has been increased, and swept-source OCT system has reached to 2.5 million Hz. However, the high computing load required to process the acquired data stream prevents real-time OCT from being realized, especially multi-functional OCT imaging, which processes not only intensity images but also functional images like flow and polarization. During acquisition, rapid visualization allows for quick detection of various biological sample features. Further than the processing of OCT intensity images, which includes spectral resampling, interpolation, and a fast Fourier transform (FFT), multi-functional OCT imaging requires additional processing for reconstruction of both PS-OCT and Doppler OCT images, which adds to the overall processing time.



System type		1310	lundquist	1050	line field
calibration data	lines per experiment	2	1	0	0
	lines per image	0	0	2048	1
	line length	1024	2048	4096	2048
image data	number of streams	2	1	2	1
	line length	1024	2048	4096	8192
	lines per image	2048	2048	2048	2048
	acquisition rate	4435	8000	100000	300000
experiment data	duration (s)	3600	3600	3600	3600
	number of images	7795.898438	14062.5	175781.25	527343.75
	hard drive space (TB)	0.059477985	0.107288361	5.36441803	16.09325409
calculations	number of calibrations	2	1	360000000	527343.75
	number of lines to process	15966000	28800000	360000000	1080000000
processing time	CPU line calibration time (µs)	287.00	287.00	287.00	287.00
	CPU line processing time (µs)	22.30	22.30	22.30	22.30
	GPU line calibration time (µs)	33.60	33.60	33.60	33.60
	GPU line processing time (µs)	12.00	12.00	12.00	12.00
	CPU time (hours)	0.098900659	0.17840008	30.93	6.732041016
	GPU time (hours)	0.053220019	0.096000009	4.56	3.604921875

Figure 1.6: CPU and GPU Time for Different OCT System With the Parameters

The acceleration is especially important for swept-source system(1050 OCT system and line field OCT system), since both acquisition rate and the number of calibration is higher, as you can see in Figure 1.6. So by utilizing GPU implementation and other software level optimization, it allows rapid visualization of biological samples and make the processing time more.

This is a table that explains the motivation of the post-processing acceleration of OCT data. The data in the first two sections are pre-set parameters of different OCT systems. For 1310 and Lundquist systems it only needs 2 times and 1-time calibration per experiment respectively,(click) while for 1050 OCT system(click) it needs to calibrate for every single line because of the changing of wavelength and for line field system it needs to calibrate for every image. You can see this is the acquisition rate for different systems. The experiment data section means this table calculates the processing time of one hour OCT data acquisition. The processing time section is acquired from a real test. The calibration time on CPU per line is 287us compared with 33.6us on GPU. and the processing time per line on CPU is 22.3us compared with 12us on GPU. Just to clarify the calibration time is the execution time of apply calibration function and the processing time is the execution time of getting the depth profile function. They are both just part of the important process of the whole workflow as I mentioned on the last slide. The total CPU time equals the number of calibrations multiply time per line plus the number of lines to process multiple times per line. So for just one hour OCT data acquisition, the performance comparison is listed in the last two rows. the GPU is 2 times faster than the CPU on 1310 system, Lundquist system, and line field system, the performance improved most on 1050 which needs more calibrations. GPU speeds up around 7 times than CPU processing.

1.4 Introduction to MATLAB

1. **Simple configurations**

   MATLAB is more of an application. Environment configuration and code compilation are performed automatically by the program background, just write the code and execute it.

2. **Simple language**

   MATLAB focuses more on algorithms than code implementation. The algorithm here refers to the way of human thinking to the problem, so the algorithm written in the MATLAB program, and the algorithm itself has a great similarity. MATLAB to achieve a certain algorithm, just create a matrix and matrix operations.

3. **High visualization**

   Often some mathematical operation code implementation is cumbersome, such as: calculate FFT (fast Fourier transform), MATLAB provides FFT () function, engineers have optimized the underlying code of the function to be as efficient as possible. So we don't have to design our own code and modify it, just use it. In addition to the need for a lot of rich toolbox, and simple and clear graphical interface.

4. **Widespread use in academia**

   Finally, in fact, the most important reason is because the academic use of MATLAB is quite a lot. MATLAB is often used in scientific research.

Chapter 2

MATLAB Program Acceleration Based on MATLAB Feature and GPU Programming

2.1 Introduction

In most practical application situations, real-time OCT including image acquisition, data processing, and result visualization are required. As a result, fast computing speed is required. Frame rates significantly below real-time will limited OCT retinal imaging. Slow imaging speeds cause motion artifact, resulting in effective resolutions that are lower than those set by the system's specifications. It also restricts the use of OCT to analyze optical tissues during ophthalmic treatment or when the patient moves their eyes. Especially, for some swept-source OCT system, every wavelength sweep needs its own calibration. Swept-source optical coherence tomography (SS-OCT) is an imaging tool capable of providing three-dimensional information of biological tissues in vivo with high imaging speed, high sensitivity, and high resolution. This desirable performance of SS-OCT is mostly attributable to the laser source performance. SS-OCT uses a wavelength-swept laser as the light source; the wavelength sweep rate and instantaneous linewidth of the laser used in the imaging system determine the imaging speed and imaging depth of the system, respectively.

2.2 Techniques to Improve Performance

2.2.1 Optimization Based on MATLAB Feature

Vectorization Programming

In the new version of MATLAB, the execution efficiency of the FOR loop has been greatly improved, but for some complex operations in the for loop, such as changing the dimension of the array in the for loop, vectorization is still an effective means to improve the execution efficiency of MATLAB.

Vectorization Operation MATLAB documentation says "MATLAB is a matrix language, which means it is designed for vector and matrix operations. You can often speed up your M-file code by using vectorizing algorithms that take advantage of this design. Vectorization means converting for and while loops to equivalent vector or matrix operations". To improve, there are two ways to do this: Try to use vectorization operations instead of loop operations. The speed will be greatly increased. The most common functions that use Vectorizing techniques are: all, diff, ipermute, permute, reshape, squeeze, any, find, logical, prod, shiftdim, sub2ind, cumsum, ind2Sub, ndgrid, repmat, sort, sum and so on. When multiple loops must be used, and the executed time of two loops are different and the number of loops in the outer loop is less, the number of loops in the inner loop is more, this can significantly increase speed.

Calculating by Column MATLAB inherits the column storage feature of Fortan, which means higher memory hit ratio and faster speed when calculating by column.

Preallocated Memory

There are multiple ways to improve the running speed of MATLAB program. Firstly, arrays in MATLAB do not need to be explicitly defined and specified before they are used. When the subscript of the assigned element exceeds the existing dimension, MATLAB will enlarge the dimension of the array or matrix once, which will greatly reduce the efficiency of the program. Repeatedly resizing arrays often requires MATLAB® to spend extra time looking for larger contiguous blocks of memory, and then moving the array into those blocks. Therefore, before the loop, the pre-allocation of memory can effectively improve the execution speed of the program. Use the appropriate preallocation function for the kind of array you want to initialize:

- zeros for numeric arrays
- strings for string arrays
- cell for cell arrays
- table for table arrays

Built-in Function

MATLAB most of the built-in functions are M files, itself is also a MATLAB program, the execution efficiency is low. However, some underlying basic functions, such as sum, Max, find, exp, and FFT, are executed more efficiently. The built-in functions are preferentially used to achieve faster execution speed.

Using Local Functions

When MATLAB calls a function, the function needs to be searched and matched. The search priority is as follows:

1. variable
2. Import Specifies the imported package function
3. A nested function within the current function
4. Local functions in the current file
5. Private functions
6. The object function
7. Class constructor in @ folder
8. Load the Simulink model
9. Functions in the current folder
10. Functions at other locations in the path

As you can see, the functions in the folder are of low priority when searching. If you use local functions, the function will be searched faster.

Sparse Matrix

If there are only a few non-zero values in a matrix, it can be represented by a sparse matrix. The sparse matrix records only the non-zero values in the matrix. When the sparsity of the matrix is large enough, the matrix can be converted into a sparse matrix, which can reduce the memory footprint and reduce the amount of computation. Note: If the sparsity of the matrix is not large, forced conversion into sparse matrix will lead to larger memory footprint and lower execution efficiency.

2.2.2 Optimization Based on Hardware Feature

CPU Parallel Computation PARFOR/SPMD

The PARFOR technique, which is similar to OpenMP, is a simple way to have FOR loops execute in parallel. The simplest parallel programming command in MATLAB is PARFOR. When a loop can be parallelized, PARFOR substitutes the FOR command. The loop iterations are split up among several ”workers,” executed in an uncertain sequence, and the results fed back to MATLAB's main copy. There are computation limitations since parallel loop iterations are carried out in an uncertain order, therefore all required data must exist before the loop begins. An iteration cannot rely on previous iteration outcomes. Single Program, Multiple Data is what SPMD stands for. The SPMD command is a simple version of MPI. One client process supervises workers who work together on a single program. Each worker has an identifier, and can determine its behavior based on that ID.

GPU Acceleration

Scientists, engineers, and financial analysts have been able to speed up computationally intensive applications in a range of fields attributable to multicore processors and hyper-threading technology. Today, another sort of hardware, the graphics processing unit, offers even higher computing performance (GPU). GPUs, which were originally designed to speed up graphics rendering, are increasingly being utilized in scientific calculations. A GPU contains a massively parallel array of integer and floating-point processors, and also dedicated, high-speed memory, unlike a standard CPU, which has only a few cores. Hundreds of these tiny units make up a typical GPU.

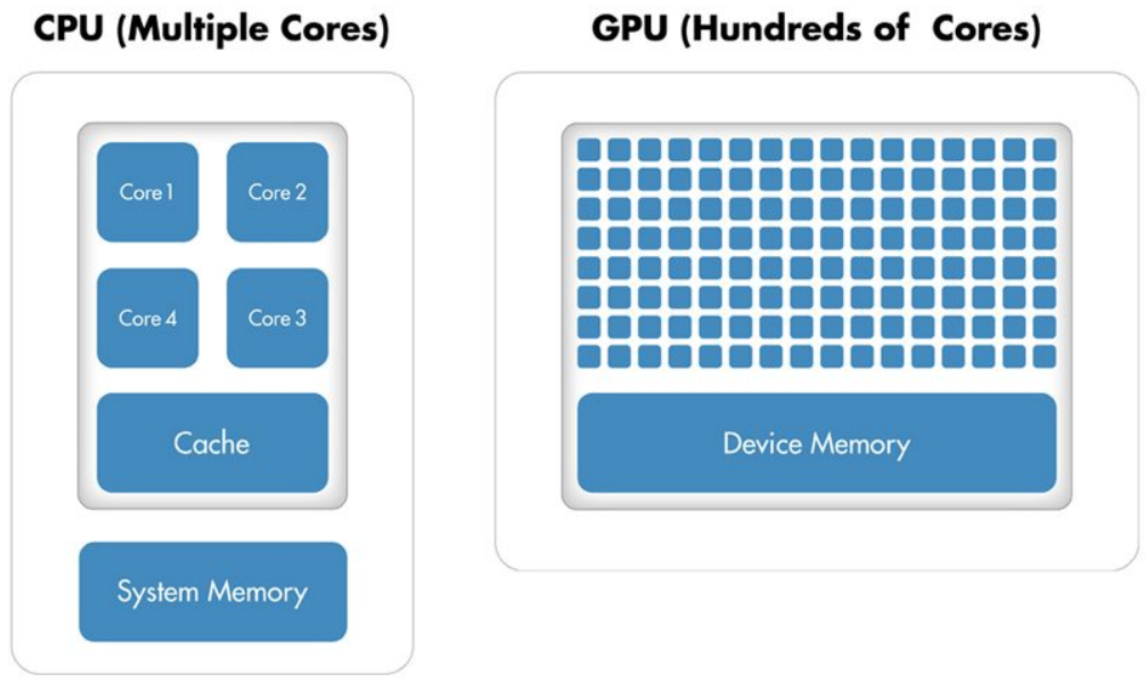


Figure 2.1: Comparison of the Number of Cores on a CPU System and a GPU

Distributed Computation

MATLAB can be run on Clusters or Clouds. Multiple computers work together to greatly improve the computing speed for large-scale computing. Prerequisites: Multiple computers on the same LAN, the same MATLAB version, and Parallel Computing Toolbox and MATLAB Parallel Server installed.

2.2.3 Optimization Based on Programming Feature

Multi-Language Program

MATLAB provides a flexible, two-way integration with other programming languages, allowing you to reuse legacy code.

MEX Programming A MEX file is a function, created in MATLAB, that calls a C/C++ program or a Fortran subroutine. A MEX function behaves just like a MATLAB script or function.

CUDA Programming Write a MEX-File Containing CUDA Code. As with any MEX-files, those containing CUDA code have a single entry point, known as mexFunction. The MEX-function contains the host-side code that interacts with gpuArray objects from MATLAB and launches the CUDA code. The CUDA code in the MEX-file must conform to the CUDA runtime API.

2.3 Method We Use

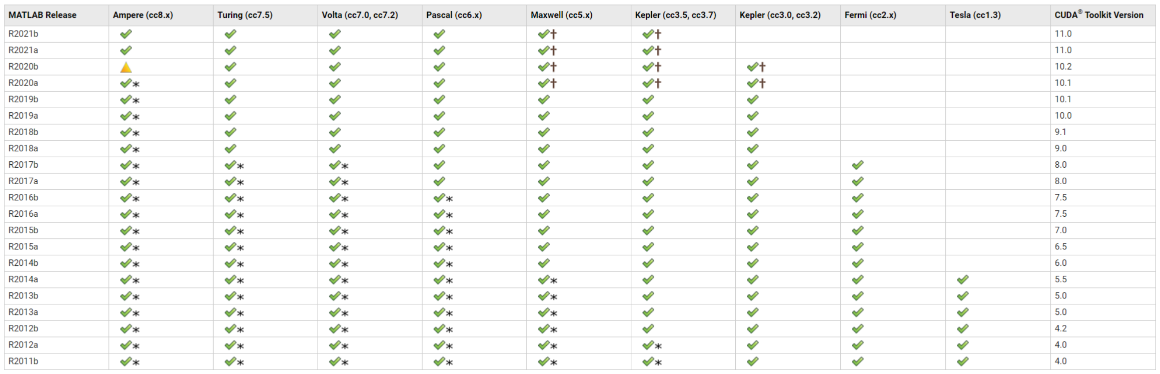
2.3.1 GPU Acceleration

To speed up the running speed of MATLAB code, in addition to code optimization, I also use GPU to speed up the calculation. A GPU can accelerate an application if it fits both of the following criteria: 1. Computationally intensive. The time spent on computation significantly exceeds the time spent on transferring data to and from GPU memory. 2. Massively parallel. The computations can be broken down into hundreds or thousands of independent units of work. Applications that do not satisfy these criteria might actually run slower on a GPU than on a CPU.

GPU Card

First of all, you need a supported GPU, currently only Nvidia GPU is supported, different versions of MATLAB support different architecture of GPU. You can run the gpuDevice command to check whether a GPU is supported.If you have multiple GPUs, then you can use gpuDeviceTable to examine the properties of all GPUs detected in your system. You can use gpuDevice to select one of them, or use multiple GPUs with a parallel pool.

Upgrading GPU With the advancement and innovation on GPU development, newer versions of GPU cards have more number of processor cores as well as larger memory. We previously had two GPU cards: Tesla P4, which is the computation card and Quadro P400 which is the display card. Now we replaced them by Nvidia Grforce RTX 3080. You can see



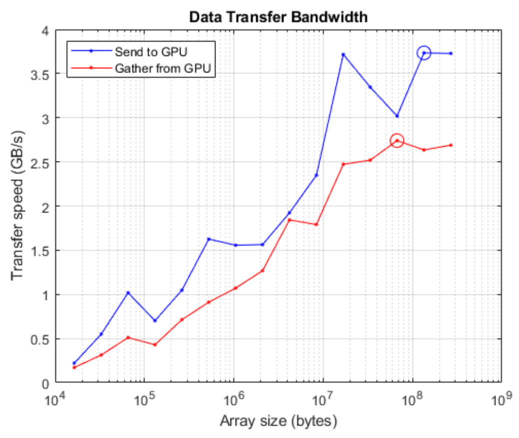
MATLAB Release	Ampere (cc8.x)	Turing (cc7.5)	Volta (cc7.0, cc7.2)	Pascal (cc6.x)	Maxwell (cc5.x)	Kepler (cc3.5, cc3.7)	Kepler (cc3.0, cc3.2)	Fermi (cc2.x)	Tesla (cc1.3)	CUDA® Toolkit Version
R2021b	✓	✓	✓	✓	✓†	✓†				11.0
R2021a	✓	✓	✓	✓	✓†	✓†				11.0
R2020b	▲	✓	✓	✓	✓†	✓†	✓†			10.2
R2020a	✓*	✓	✓	✓	✓†	✓†	✓†			10.1
R2019b	✓*	✓	✓	✓	✓	✓	✓			10.1
R2019a	✓*	✓	✓	✓	✓	✓	✓			10.0
R2018b	✓*	✓	✓	✓	✓	✓	✓			9.1
R2018a	✓*	✓	✓	✓	✓	✓	✓			9.0
R2017b	✓*	✓*	✓*	✓	✓	✓	✓	✓		8.0
R2017a	✓*	✓*	✓*	✓	✓	✓	✓	✓		8.0
R2016b	✓*	✓*	✓*	✓*	✓	✓	✓	✓		7.5
R2016a	✓*	✓*	✓*	✓*	✓	✓	✓	✓		7.5
R2015b	✓*	✓*	✓*	✓*	✓	✓	✓	✓		7.0
R2015a	✓*	✓*	✓*	✓*	✓	✓	✓	✓		6.5
R2014b	✓*	✓*	✓*	✓*	✓	✓	✓	✓		6.0
R2014a	✓*	✓*	✓*	✓*	✓*	✓	✓	✓	✓	5.5
R2013b	✓*	✓*	✓*	✓*	✓*	✓	✓	✓	✓	5.0
R2013a	✓*	✓*	✓*	✓*	✓*	✓	✓	✓	✓	5.0
R2012b	✓*	✓*	✓*	✓*	✓*	✓	✓	✓	✓	4.2
R2012a	✓*	✓*	✓*	✓*	✓*	✓*	✓	✓	✓	4.0
R2011b	✓*	✓*	✓*	✓*	✓*	✓*	✓	✓	✓	4.0

Figure 2.2: GPU Support by Release

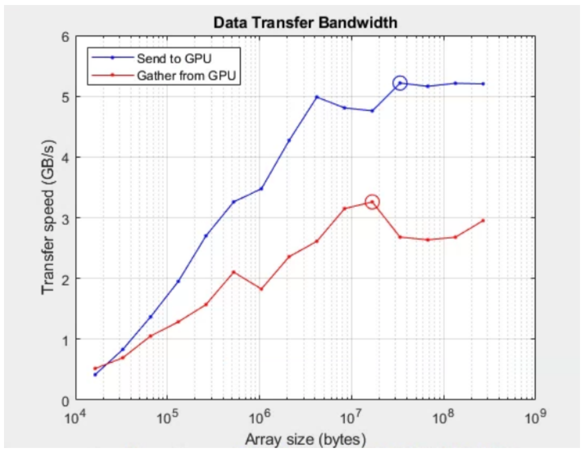
the some performance comparison between the two cards in figure 2.3, figure 2.4, figure 2.5. the Host which is the CPU, in these two figures, they are the same one, so the scale on y axis is different, look at the Y axis in the read and write bandwidth the maximum number is 800 on the left side while only 90 on the right side. The same for matrix to matrix multiply the Host is same CPU card, previously the GPU card performance is always worse than the CPU performance no matter what size is the matrix but now if it is enough computationally intensive the GPU calculation is better. So the GPU card performance has greatly improved after upgrading it.

PCT (Parallel Computing Toolbox)

Parallel Computing Toolbox supports more than 700 functions that let you use GPU computing. Any GPU-supported function automatically runs using your GPU if you provide inputs as GPU arrays, making it easy to convert and evaluate GPU compute performance for your application. PCT allows you to use multicore processors, GPUs, and

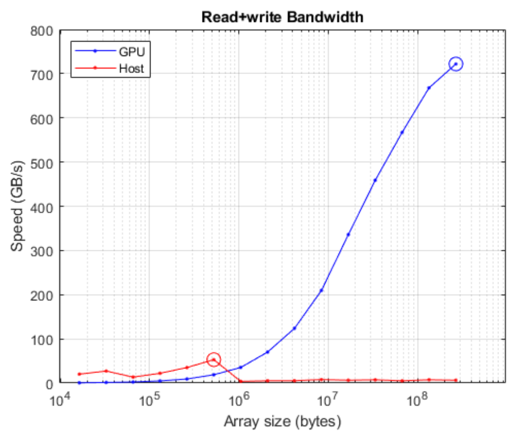


(a) Data Transfer Bandwidth(3080)

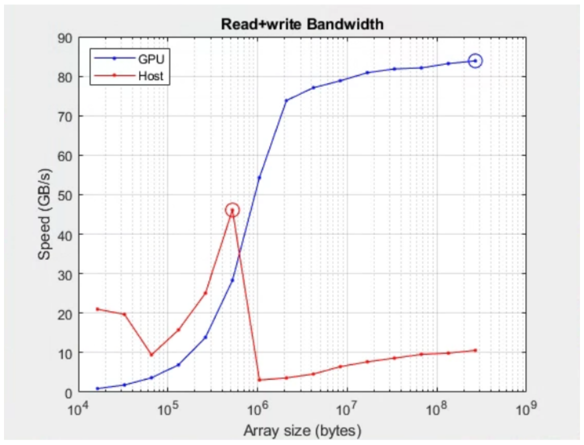


(b) Data Transfer Bandwidth(Tesla P4)

Figure 2.3: Data Transfer Bandwidth

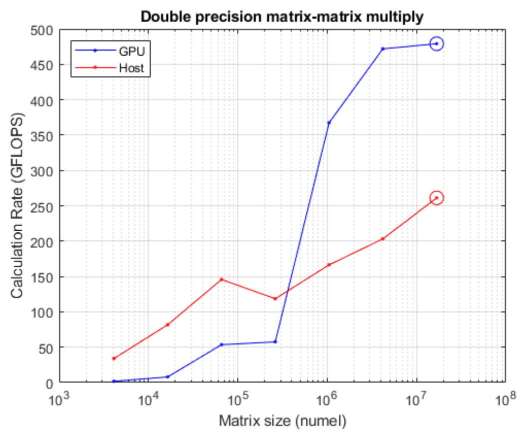


(a) Read+Write Bandwidth(3080)

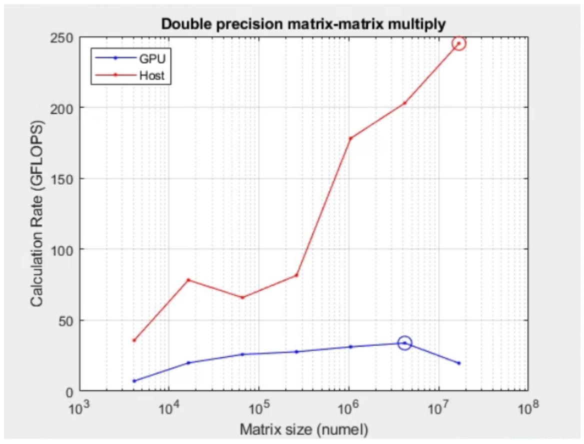


(b) Read+Write Bandwidth(Tesla P4)

Figure 2.4: Read+Write Bandwidth



(a) Double Precision Matrix-Matrix Multiply(3080)



(b) Double Precision Matrix-Matrix Multiply(Tesla P4)

Figure 2.5: Double Precision Matrix-Matrix Multiply

computer clusters to solve computationally and data-intensive tasks. You may parallelize MATLAB® applications without using CUDA or MPI programming while using high-level structures like parallel for-loops, specific array types, and parallelized numerical techniques. In MATLAB and other toolboxes, the toolbox allows you to use parallel-enabled functions. The toolbox can be used with Simulink® to run multiple model simulations in parallel. Interactive and batch modes are available for programs and models. The toolbox enables you to take use of multicore desktops full processing capacity by running applications on workers (MATLAB computational engines). You may run the same programs on clusters or clouds (using MATLAB Parallel ServerTM) without modifying the code. The toolbox may also be used with MATLAB Parallel Server to run matrix calculations that are too massive to fit in a single machine's memory. When considering how to accelerate this computation using Parallel Computing Toolbox, we will focus on the code that performs computations for each time step. The computations involve MATLAB operations for which GPU-enabled overloaded functions are available through Parallel Computing Toolbox. These operations include FFT and IFFT, matrix multiplication, and various element-wise operations. As a result, we do not need to change the algorithm in any way to execute it on a GPU. We simply transfer the data to the GPU using *gpuArray* before entering the loop that computes results at each time step.

Use GpuArray and Gather Commands

”gpuArray” can transfer existing arrays in memory to GPU memory. All built-in functions that can generate matrices can directly generate GPU arrays in memory by adding the parameter ”gpuArray”. The gather command can extract arrays from GPU memory into memory. The gpuArray and gather commands can transfer data between CPU and GPU memory. This process is also time-consuming. If the computation is small, the data transfer cost may be larger than the computation time. Hundreds of functions in MATLAB® and other toolboxes run automatically on a GPU if you supply a gpuArray argument. Whenever you call any of these functions with at least one gpuArray as a data input argument, the function executes on the GPU. The function generates a gpuArray as the result, unless returning MATLAB data is more appropriate (for example, size). You can mix inputs using both gpuArray and MATLAB arrays in the same function call.

2.3.2 Vectorization Operation

Since MATLAB is a matrix-based language, every input can be treated as a matrix. The reason why GPU is slower than the CPU for this problem is that the for-loop is executing the FFT, multiplication, and interpolation operations on individual columns of length 4096. The best way to increase the performance is to vectorize the code, so that a single MATLAB function call performs more calculation. The FFT operation is easy to vectorize: $fft(A)$ computes the FFT of each column of a matrix A. I can perform a multiply of the filter with every column in a matrix at once using the MATLAB binary scalar expansion function bsxfun. In conclusion, vectorizing the code helps both the CPU

Function Execution Time Before and After Vectorization			
	CPU time(s)	Tesla P4 time(s)	Nvidia Grforce RTX 3080 time(s)
No vectorization	0.21312	1.0959	1.06979
Vectorization	0.18044	0.026738	0.0069877

Table 2.1: Execution Time Before and After Vectorization for CPU and GPU-CPU Hybrid Processing

and GPU versions to run faster. However, vectorization helps the GPU version much more than the CPU. The improved CPU version is nearly 1.18 times as fast as the original; the improved GPU version is 55 times faster on old card than the original and . The GPU code went from being 40% slower than the CPU in the original version, to about 10 times faster in the revised version.

2.4 Result

Figure 2.6 is the comparison of MATLAB code running on CPU and GPU(Nvidia Geforce RTX 3080) . we can see after vectorization operation and using the PCT, the whole processing time of a volumn of data on GPU is faster than CPU from the beginning, even if we add the data transfer time. The transfer time here includes both time of transferring from CPU to GPU and time of transferring back from GPU to CPU.

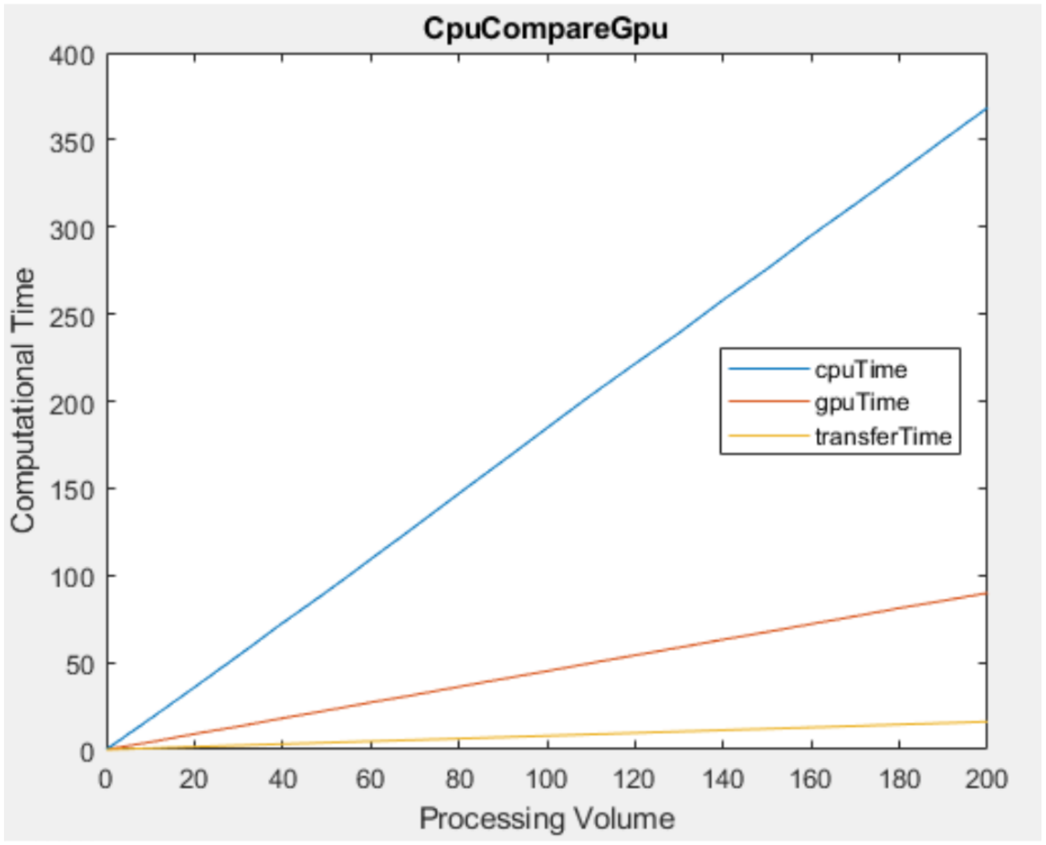


Figure 2.6: Comparison of MATLAB Code in CPU and GPU Execution Efficiency

Chapter 3

Web application

3.1 Introduction

In order to further improve the work efficiency, our lab has already tried some remote control softwares like Teamviewer, google remote desktop etc. So we can remotely run the program even though we are not physically in the lab or in the collaborative lab, we can also save time to run the program overnight. But this method only allows one client to control the PC at a time. So this will cause a lot of conflict with the data processing PC we use most commonly. Actually, we do not need control of the whole PC, we just need some simple operations. Once we make sure the code runs properly, we can leave the PC process and it is unnecessary to monitor the program all the time. Thus, my idea is we need a panel that allows multiple clients to control the MATLAB engine and can do some basic operations of the MATLAB programs. This web application is designed to be cross-platform and available on both computers and mobile devices.

3.2 Mobile Application v.s Web Application

3.2.1 Mobile Application

Native mobile apps are built for a specific platform, such as iOS for the Apple iPhone or Android for a Samsung device. They are downloaded and installed via an app store and have access to system resources.

Mobile apps are more expensive to develop than web apps, and because they are platform-specific, launching an app across different platforms pretty much means starting from scratch in terms of design and development.

Native mobile apps are built using specific languages and Integrated Development Environments (IDE), depending on the intended platform. Apple devices run on the iOS native operating system, so Apple apps are built using either Objective-C or Swift, and the Xcode IDE. Native apps for Android are written in Java and are commonly built using the Android Studio or Eclipse IDE.

However, they are much faster, and tend to be more advanced in terms of features and functionality.

Pros: 1. Notifications can be pushed more on time than the web. 2. Faster than web apps. 3. Greater functionality as they have access to system resources. 4. Can work offline. 5. Safe and secure, native apps must first be approved by the app store.

Cons: 1. Compatibility with different platforms (i.e. iOS and Android) usually means designing and building the app from scratch. 2. Expensive to maintain and update. 3. App Store approval can take extra time and money cost. 4. Download and installation required.

3.2.2 Web Application

Web apps, on the other hand, are accessed via the internet browser and will adapt to whichever device you're viewing them on. They are not native to a particular system, and don't need to be downloaded or installed.

Web apps tend to be built using JavaScript, CSS, HTML, and Python. Unlike mobile apps, there is no standard software development kit for building web apps. However, developers do have access to templates. Compared to mobile apps, web apps are usually quicker and easier to build, but they are much simpler in terms of features.

Pros: 1. Compatibility (PC and mobile devices - showing different layout adaptively). 2. Do not need to be downloaded or installed—web apps function in-browser. 3. Easy to maintain. 4. Will update themselves. 5. Quicker and easier to build than mobile apps. 6. Do not require app store approval, so can be launched quickly.

Cons: 1. Login needed (Use cookies to solve). 2.Slower than mobile apps, and less advanced in terms of features. However, as the current requirements are not that much, so it's not a big deal.

3.2.3 Conclusion

Since our need is processing data remotely as well as checking the result, also it need to allow multiple users to access, based on the current occasion, I decided to develop web app. Considering a lot of factors including quick delivery, limited features, easy maintenance and less cost of both money and time, web application is a good fit. Most importantly, a good user experience is also take into consideration. A web application do not need download, installation and update, which provides great convenience and flexibility to all kinds of users in our lab.

3.3 Web Frameworks

A software usually has a front-end and a back-end. The front-end is usually responsible for interface rendering, that is, how to make it look good; while the back-end handles the front-end request logic, accesses the database to get the corresponding data, and returns the data to the front-end, which is rendered by the front-end. Front end usually refers to the client. Clients usually include browsers, various clients installed on computers, and those installed on mobile phones (also called mobile terminals).The back-end is more about interacting with the database to process the corresponding business logic. What needs to be considered is how to implement functions, data access, platform stability and performance, etc. The back-end mainly provides interfaces to the front-end MVC is a layered idea, a design pattern, and a way of dealing with problems generated by the backend. M is the Model layer, V is the View layer, and C is the Controller control layer. When the back-end receives the request from the front-end, the control layer C usually processes the logic, deploys different service processing according to different logic, and then encapsulates the data into a Model and returns it to the specified view for display.

Modern Web frameworks, no matter what design ideas and development philosophies they adopt, have the same basic working mode: they all receive HTTP requests, process various HTTP parameters, route to the corresponding user-implemented processor, and then obtain the returned results to generate HTTP Response. In fact, it is not a must to use a server-side Web framework to develop a website, but using a framework can help developers improve the quality and efficiency of Web development work, greatly reducing the development workload. However, nowadays the Internet is full of a variety of Web development frameworks, which can provide a variety of functional extensions for developers' projects. How to choose becomes a thorny problem.

A good Web framework allows you to write simple syntactic code that generates code to handle these requests and responses. This means that your work gets simpler, your interactions get simpler, and you use highly abstract code instead of low-level code.

Factors that Influence the Decision

Cost of Learning Learning a Web framework depends on your familiarity with the underlying language, the consistency of its APIS, the quality of the documentation, and the size and activity of the community. It needs to be open source, constantly evolving, with a passionate community of people who create documentation and help users on discussion boards, and be used on many mature, high-quality sites.

Efficiency Efficiency is a measure of how quickly you can create a new feature once you are familiar with a framework, including the cost of writing and maintaining the code. Choose a framework that encourages good development instants: for example, a framework that encourages the Model-view-controller structure to separate code into logical functions will be more maintainable code. In addition, the visual development and configuration of Web development framework also greatly improves the development efficiency.

Caching Support As your site becomes more and more successful, you may find that it can no longer handle the volume of requests it receives. At this point, you might start thinking about adding caching support. Caching is an optimization in which you save all or most of your site requests so they don't need to be recalculated in subsequent requests. Returning a cache request is much faster than recomputing. Caches can be built into your code, or into a server. Web frameworks have varying degrees of support for defining cacheable content.

Powerful Extension Function No good framework can cover everything, and the reality of every case is different, so the extensibility of the framework is required to be strong enough to meet the needs of new businesses.

Network Security Some Web frameworks provide better support for addressing common network attacks. For example, JavaScript input from the client will not be run. Many frameworks provide this kind of functionality, but it is usually not turned on directly by default.

3.3.1 Front-End Framework: React

The front end has grown particularly fast in recent years. In 2022, three frameworks will dominate the market: Vue, React and Angular. Angular was developed by Google and first released in 2010. It is a typescript-based JavaScript framework. Vue, also known as vue.js. It was developed by former Google employee Evan You in 2014. The current stable release is 3.0, which was released in September 2020 (a number of smaller incremental releases have followed). React was developed by Facebook and originally released in 2013.

Facebook uses React extensively in its products (Facebook, Instagram and WhatsApp). The current stable release is 17.x, released in October 2020 (with some small incremental updates). Here's Google's data on search popularity trends for the three frameworks over the past year.

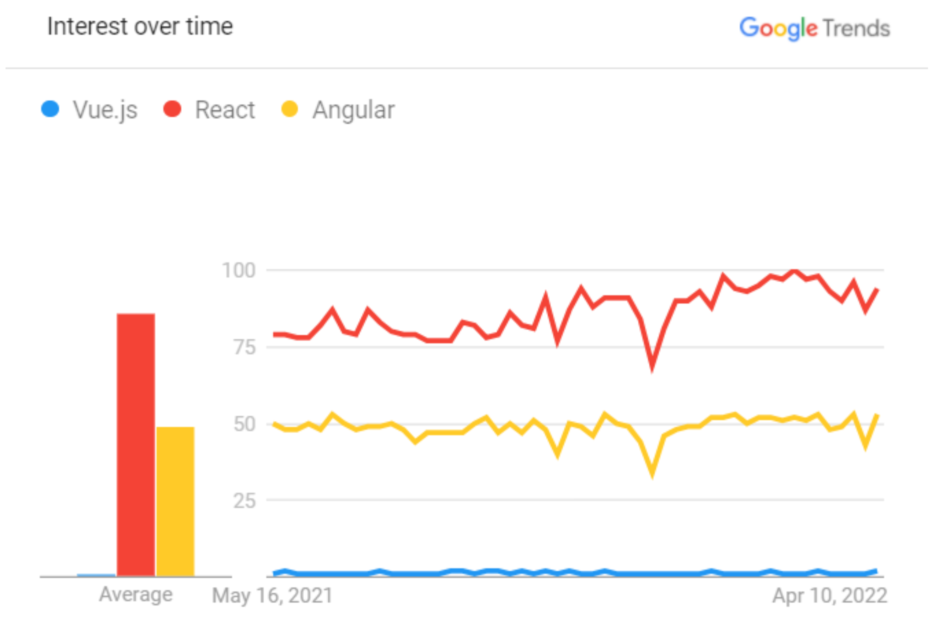


Figure 3.1: Google Trend of Front-End Framework

Finally, based on the factors mentioned above, I chose React as my front-end framework. React (also known as React. Js or ReactJS) is a JavaScript library for building user interfaces.

The modular structure enables it to have flexible code, saving time and cost. Assist in high-performance implementation of complex applications. React front-end development makes code maintenance much easier. Support mobile native apps for Android and iOS platforms.

Advantages of React

Componentization The React code is made up of entities called components. Everything is Component: Code is more modular, React projects have a clearer structure, code is easier to reuse and more maintainable. Components can use the React DOM library to render to a specific element in the DOM. When rendering a component, you can pass in a value called "props ". The two main ways to declare components in React are through function function components and class-based components.

Virtual DOM Another notable feature is React's use of the virtual document Object model, also known as the virtual DOM. React efficiently updates the DOM displayed by the browser by creating an in-memory cache of data structures that calculates the change differences and renders only the children of the actual change.

JSX JSX, or JavaScript XML, is an extension to the syntax of the JavaScript language. Similar in appearance to HTML, JSX provides a syntax-structured component rendering approach familiar to developers. The React component is typically written in JSX.

Unidirectional Data Flow React supports the use of flux mode to transfer data between components, which is called unidirectional data flow. The advantage of unidirectional data flow is compared to angularJS two-way data binding. Therefore, all changes are predictable and controllable.

3.3.2 Back-End Framework: SpringBoot

For the back-end, I chose Spring Boot which is an open resource framework based on JAVA. In the past, when we used SpringMVC+Spring+Mybatis framework for development, we built and integrated the three frameworks. We needed to do a lot of work, such as configuring the Web. XML, configuring Spring, configuring Mybatis, and integrating them together. The Spring Boot framework revolutionized the development process, ditching the tedious XML configuration process and simplifying the development process with a large number of default configurations.

SpringBoot is a further encapsulation and simplification of Spring application development using Maven. It is designed to simplify the initial setup and development process of Spring applications. SpringBoot Initialization helps developers perform lazy initialization mode. Enabling this feature helps developers create beans on demand, rather than during application startup. Therefore, the lazy initial mode can reduce the time required for the application to start.

Advantages of Spring Boot

Spring applications can be created with minimal effort, development process, and configuration. We can simply start these applications using the java-jar command. If the boot process of a Spring-Boot application fails at first, the built-in fault analyzer provides a path to resolve the problem. Spring-boot supports embedded servers such as Tomcat and Jetty. Therefore, there is no need to use the.war file in depth on an external server. To simplify Maven configuration, it provides a useful launcher POM. It has the ability to automatically configure Spring. Spring-boot can be integrated with other frameworks as well as hibernation and hibernation. It provides production-ready configurations, such as metric health and externalized configurations. With Spring-Boot, we don't need to do XML configuration or code generation. Spring-boot reduces developer workload by using transformations based on the configuration software design paradigm.

3.4 The Structure of the Front-End and Back-End

The Model-View-Controller (MVC) is an architectural pattern that separates an application into three main logical components: the model, the view, and the controller. MVC is one of the most frequently used industry-standard web development framework to create scalable and extensible projects.

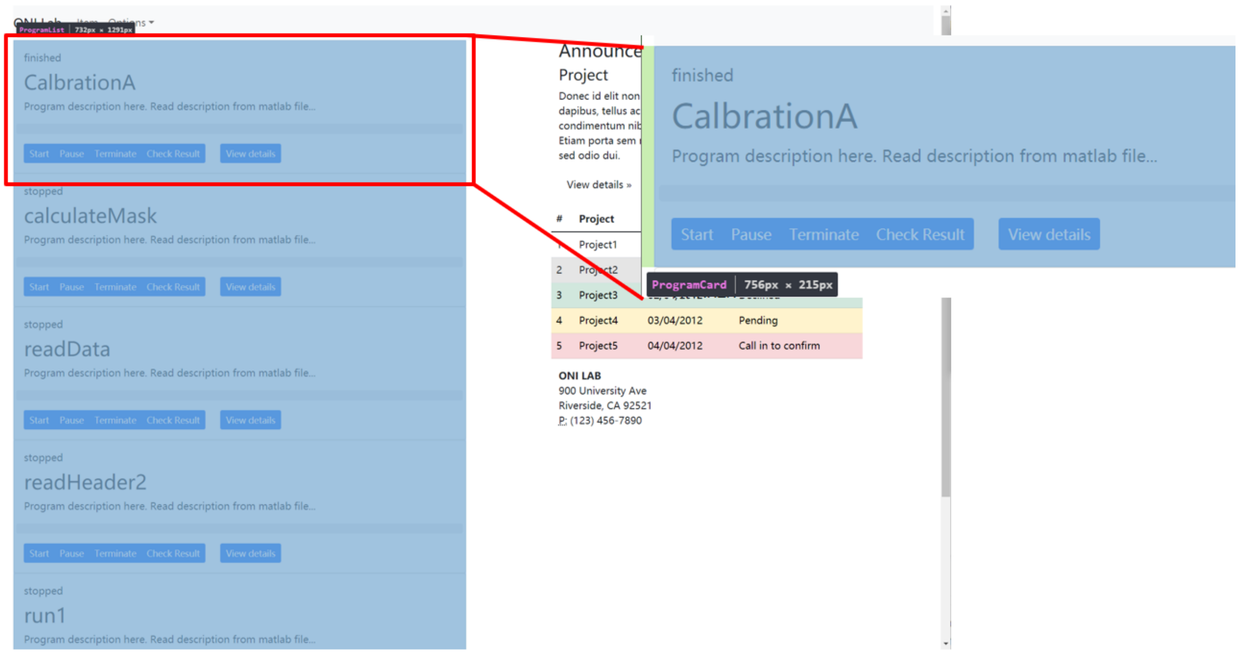
Model: This component represents the data and the business logic of the application. The model component focus on keeping track of the state of the application.It also defines business rules for data, means how the data can be changed and manipulated.

View: The view provides the user interface (UI) for the model. The main work (function) of the view represents the information in user understandable format. It uses UI Components such as HTML, CSS, Jquery etc.

Controller: Controller act as a mediator between view and model. it is responsible to control the data transmission between the model and the view. It maps the user action into model updates.The controller layer is helpful to select the most appropriate view and delivers it to the user.

3.4.1 Front-end

If we treat the whole web page as a container, it is split to a navigation bar and a main body. The main body is split to three different width columns: the Program list, Right column(table shows all the program and its corresponding status) and a gap column between the previous two columns. The first column which is the ProgramList is consist of some ProgramCards. In each ProgramCard, there are CardTitle which is the status of the program, the program infomation, the ProgressBar and the controlButton group including: start, terminate. check result and view details.



ONI Lab
ProgramList 732px × 1291px
finished
CalbrationA
Program description here. Read description from matlab file...
Start Pause Terminate Check Result View details
stopped
calculateMask
Program description here. Read description from matlab file...
Start Pause Terminate Check Result View details
stopped
readData
Program description here. Read description from matlab file...
Start Pause Terminate Check Result View details
stopped
readHeader2
Program description here. Read description from matlab file...
Start Pause Terminate Check Result View details
stopped
run1
Program description here. Read description from matlab file...
Announce
Project
View details »
ProgramCard 756px × 215px

Donec id elit non
dapibus, tellus ac
condimentum nib
Etiam porta sem
sed odio dui.

#	Project		
1	Project1		
2	Project2		
3	Project3		
4	Project4	03/04/2012	Pending
5	Project5	04/04/2012	Call in to confirm

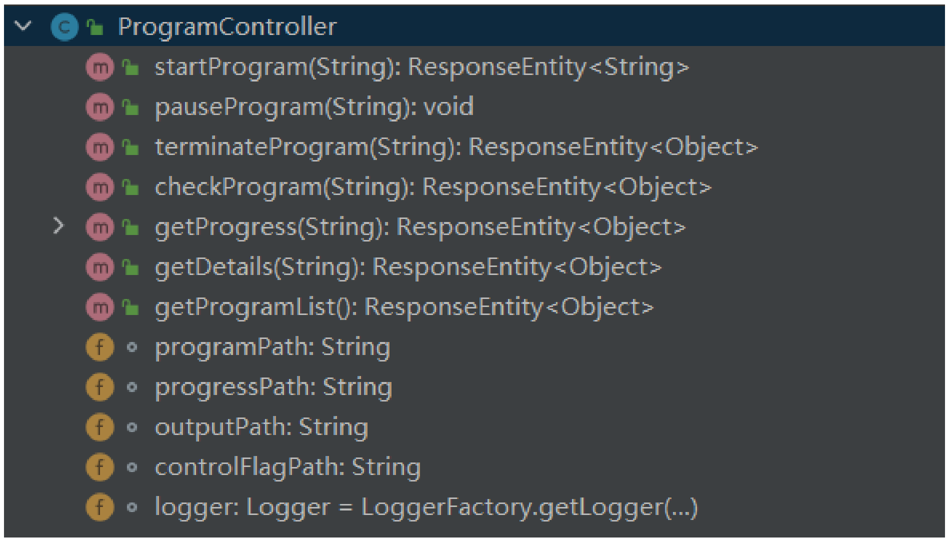
ONI LAB
900 University Ave
Riverside, CA 92521
P: (123) 456-7890

Figure 3.2: Front-End Structure

3.4.2 Back-end

Controller: ProgramController

A controller is responsible for controlling the way that a user interacts with an MVC application. A controller contains the flow control logic for an MVC application. The controller responds to the user input and performs interactions on the data model objects. The controller receives the input, optionally validates it and then passes the input to the model. A controller determines what response to send back to a user when a user makes a browser request. A controller is just a class, here we call it $ProgramController$ Notice



```
ProgramController
    startProgram(String): ResponseEntity<String>
    pauseProgram(String): void
    terminateProgram(String): ResponseEntity<Object>
    checkProgram(String): ResponseEntity<Object>
  > getProgress(String): ResponseEntity<Object>
    getDetails(String): ResponseEntity<Object>
    getProgramList(): ResponseEntity<Object>
    programPath: String
    progressPath: String
    outputPath: String
    controlFlagPath: String
    logger: Logger = LoggerFactory.getLogger(...)
```

Figure 3.3: ProgramController Structure

that the $ProgramController$ has some methods that correspond to the actions exposed by the controller. For example, The URL $/program/start/ + programName$ invokes the $ProgramController.startProgram()$ method.

Spring Web MVC Annotations

@GetMapping Spring4.3, @getmapping, @postmapping, @putmapping, @deletemapping, and @patchmapping are introduced to help simplify the mapping of commonly used HTTP methods and better express the semantics of annotated methods. Take @getMapping as an example. The official Spring documentation says: @getMapping is a combined annotation that is short for @requestMapping (method = requestMethod.get). This annotation maps HTTP Get to a specific processing method.

@Value The @Value annotation is a common feature in the Spring framework. It is used to annotate constants, values from configuration files, and property values of other beans into variables as their initial values. Methods of use: @Value (" constant ") Constants, including strings, URLs, file paths; @Value (" $ ": default_value) reads the configuration file, etc. In this project we use the second method, reading configuration files. @Value ("${}") reads the Value from the configuration file and injects it into a variable. The configuration files are divided into default configuration files application.properties and custom configuration files. Using the @value annotation to read the configuration has advantages in practice. One is to centrally manage variables in configuration files for later maintenance and expansion. Second, it greatly improves the expansibility and universality of the code.

Model

An MVC model contains all of your application logic that is not contained in a view or a controller. The model should contain all of your application business logic, validation logic, and database access logic. In general, you should strive for fat models and skinny controllers. The controller methods should contain only a few lines of code. A controller should only contain the bare minimum of logic required to return the right view or redirect the user to another action (flow control). Everything else should be contained in the model.

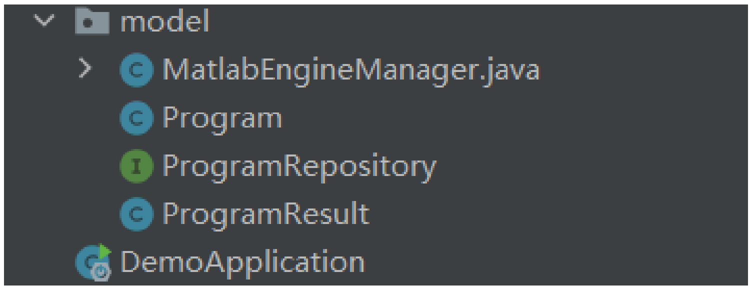
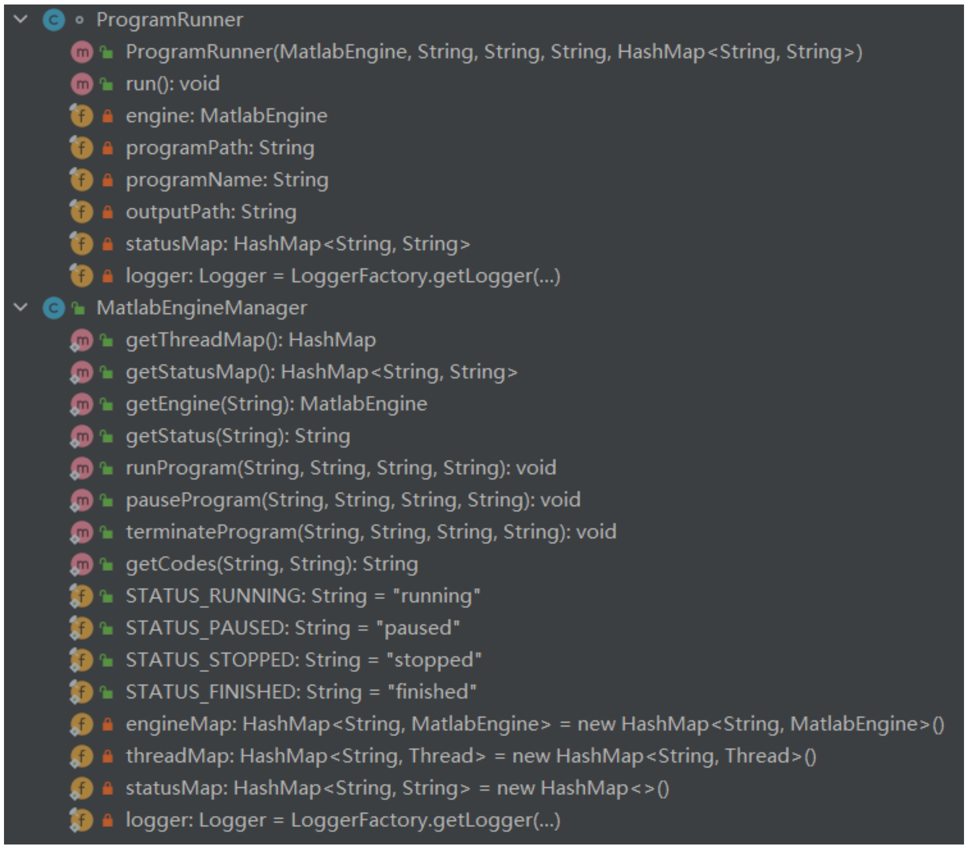

```
model
    MatlabEngineManager.java
    Program
    ProgramRepository
    ProgramResult
DemoApplication
```

Figure 3.4: Model Structure

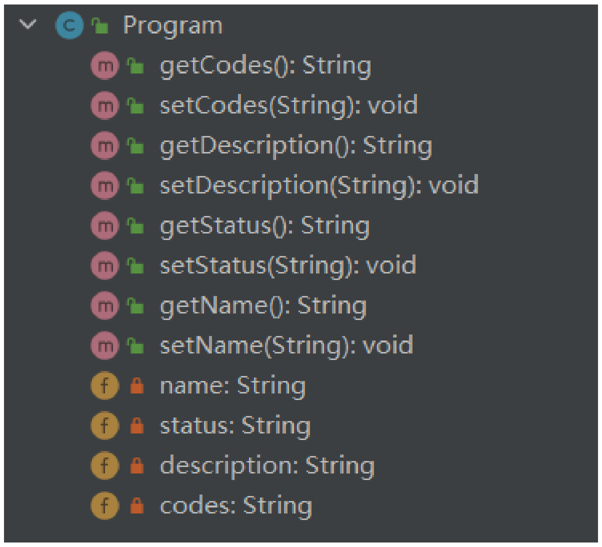
MATLABEngineManager MATLAB development environment provides a rich application interface program (API), using these interface functions can easily realize C, C++, Java, FORTRAN and other languages call MATLAB commands and data files. The MATLAB application Program Interface (API) also provides a set of engine functions to launch and call MATLAB functions. Through MATLAB engine technology, realize the call of background MATLAB function, can take into account the advantages of two aspects, one is to maintain the high efficiency of Java programming language, the other is to directly call MATLAB rich engineering application functions, including complex mathematical analysis and signal processing functions. In the applications of this project, you can use the Java language human-computer interfaces, the realization of the user interface interface and data communication, at the same time call MATLAB engine to realize complex mathematical analysis and the signal analysis and processing functions, this can greatly simplify the difficulty of the complicated operation function with Java language, shorten the development cycle of the application, At the same time, the application program has high reliability. The MATLAB® Engine API for Java® enables Java programs to interact with MATLAB synchronously or asynchronously, including: Start and terminate MATLAB. Connect to and disconnect from MATLAB sessions on the local machine. Call MATLAB functions with input arguments passed from Java and output variables returned from MATLAB. Evaluate MATLAB statements in the MATLAB base workspace. Pass variables from Java to MATLAB and MATLAB to Java. Java programs can pass data to and get data from the MATLAB workspace.

In this project, for the implementation of starting and terminating the program, we have a MATLAB enginemanager corresponding to them to control the program: Every program has its own independent thread controlling the engine asynchronously and has a hashmap to search whose key is the programName. There are 3 hashmaps: engineMap, threadMap, statusMap respectively.



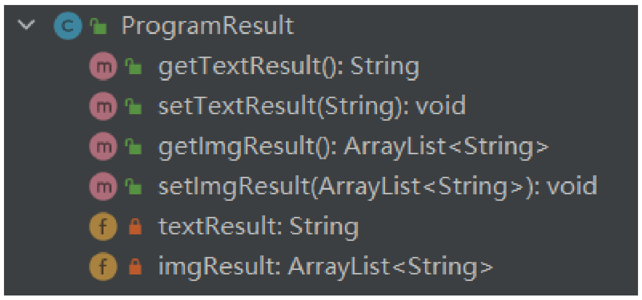
```
ProgramRunner
    ProgramRunner(MatlabEngine, String, String, String, HashMap<String, String>)
    run(): void
    engine: MatlabEngine
    programPath: String
    programName: String
    outputPath: String
    statusMap: HashMap<String, String>
    logger: Logger = LoggerFactory.getLogger(...)
MatlabEngineManager
    getThreadMap(): HashMap
    getStatusMap(): HashMap<String, String>
    getEngine(String): MatlabEngine
    getStatus(String): String
    runProgram(String, String, String, String): void
    pauseProgram(String, String, String, String): void
    terminateProgram(String, String, String, String): void
    getCodes(String, String): String
    STATUS_RUNNING: String = "running"
    STATUS_PAUSED: String = "paused"
    STATUS_STOPPED: String = "stopped"
    STATUS_FINISHED: String = "finished"
    engineMap: HashMap<String, MatlabEngine> = new HashMap<String, MatlabEngine>()
    threadMap: HashMap<String, Thread> = new HashMap<String, Thread>()
    statusMap: HashMap<String, String> = new HashMap<>()
    logger: Logger = LoggerFactory.getLogger(...)
```

Figure 3.5: MATLABEngineManager Structure



```
Program
    getCodes(): String
    setCodes(String): void
    getDescription(): String
    setDescription(String): void
    getStatus(): String
    setStatus(String): void
    getName(): String
    setName(String): void
    name: String
    status: String
    description: String
    codes: String
```

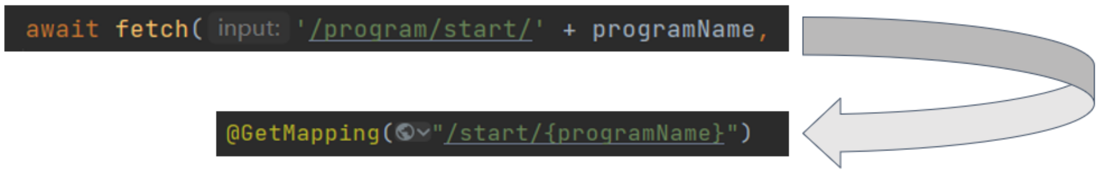
Figure 3.6: Program Structure



```
ProgramResult
    getTextResult(): String
    setTextResult(String): void
    getImgResult(): ArrayList<String>
    setImgResult(ArrayList<String>): void
    textResult: String
    imgResult: ArrayList<String>
```

Figure 3.7: ProgramResult Structure

When we want to start a program, say program1, after clicking the start button a "get request" will be sent from front-end thru the RESTful API On the right side for the code level, it will trigger a handleStart method inside the control button class in the front-end Then the controller method with the corresponding "Getmapping" annotation will be invoked and be passed the parameters, here the corresponding method is startprogram and the parameter is program1.



```
await fetch( input: '/program/start/' + programName,
```

```
@GetMapping("/start/{programName}")
```

Figure 3.8: Interaction Between Front-end and Back-end

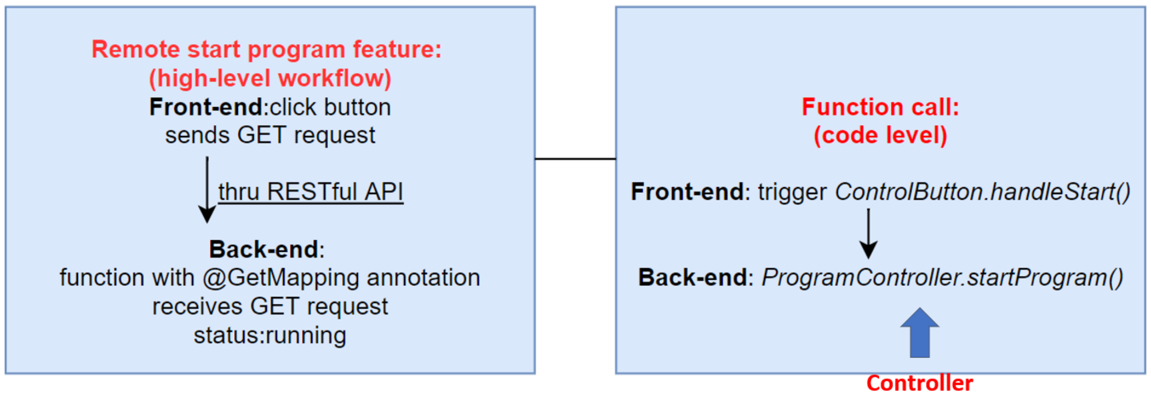


Figure 3.9: Workflow for Remote Control

3.5 Features Implemented

3.5.1 Generate the MATLAB Program List

Firstly, in the method *getProgramList*, it returns all the filenames under program path and add them to program list. Ideally, all the MATLAB programs are located in one folder, otherwise a slightly modification is needed in back-end *application.properties* configuration file.

```java
try {
    Files.list(new File(programPath).toPath())
    .forEach(path -> {
        String fileName = String.valueOf(path.getFileName());
        String name = null;
        String extension = null;
        try {
            name = fileName.substring(0, fileName.lastIndexOf("."));
            extension = fileName.substring(fileName.lastIndexOf("."));
        } catch (StringIndexOutOfBoundsException e) {
            logger.warn("File name " + fileName + " is not a MATLAB program.
                ");
        }
        if (name != null && extension != null && extension.equals(".m")) {
            Program program = new Program();
            program.setName(name);
            program.setDescription("Read description from MATLAB file... ");
            program.setStatus(MATLABEngineManager.getStatus(name));
            programList.add(program);
        }
    });
} catch (IOException e) {
    e.printStackTrace();
}
```

3.5.2 Task Status Recording System

For better logic control of the whole project, a task status recording system is a must. The system is a hashmap whose key is the program name and value is one of three given strings listed below.

```java
public static final String STATUS_RUNNING = "running";
public static final String STATUS_STOPPED = "stopped";
public static final String STATUS_FINISHED = "finished";
private static HashMap<String, String> statusMap = new HashMap<>();
```

3.5.3 Progress Bar

It's a good idea to offer both a status tracker feature and progress updates. Having only a status tracker may induce anxiety, as users end up obsessively checking the tracker to see if there has been a change since they last checked. So we have a real time progress bar which fetch progress information from back-end at a certain frequency and shown in a smoothly transition animation effect. In order not to occupy too much Internet and CPU, it only fetches the progress when the program's status is running.

For the back-end part, it read the progress information from a certain file, pass it to front-end together with the status. The *progress* class has two attribute: *status* and *fraction*

```
Path path = Path.of(progressPath + programName);
        logger.info("get progress of MATLAB program: " + programName);
        String status = MATLABEngineManager.getStatus(programName);
        double fraction = 0.0;
        try {
            fraction = Double.parseDouble(Files.readString(path,
                StandardCharsets.UTF_8));
        } catch (IOException e) {
            e.printStackTrace();
        }
        return ResponseEntity.ok(new Progress(status, fraction));
```

After fetching these two attributes from back-end in a 800ms interval, front-end start rendering the progress bar.

```
const ProgressBar = ({programName, status, setStatus}) => {
    const [now, setNow] = useState(0);

    useEffect(() => {
        // Check at interval, but only fetch when the program is running.
        const id = setInterval(() => {
            if (status === "running") {
                console.log(programName + " is running, check progress...")
                fetch('/program/getProgress/' + programName, {
                    method: 'GET',
                    headers: {
                        'Accept': 'application/json',
                        'Content-Type': 'application/json'
                    },
                })
                    .then(data => {
                        return data.json();
                    })
                    .then(data => {
                        let n = data.fraction * 100
                        setNow(parseInt(n));
                        setStatus(data.status);
                        console.log("executing...");
                    })
            }
        }, 800); //interval time
        return () => clearInterval(id); //clean up the effect.
    }, [status]); //only trigger useEffect when status changes
```

```
                //since no matter which value changes on the page
                    useEffect always be triggered
    return (
        <Progress animated value={now}>{now}%</Progress>
    );
};
```

3.5.4 Control the MATLAB Program Remotely

View Details

Only distinguishing the program with its name sometimes may confused people. So you can also read the code of the selected program from front-end and displayed in a well formatted popup modal.

Start Program

After clicking the start button, the program status will be set to running.

```
async function handleStart() {
    setDisabled(true);
    await fetch('/program/start/' + programName, {
        method: 'GET',
        headers: {
            'Accept': 'application/json',
            'Content-Type': 'application/json'
        },
    });
    setDisabled(false);
    setStatus("running");
}
```

For the back-end, after receiving the GET request from front-end thru RESTful API, it mapping to the *startProgram* method in *ProgramController*

```
 @GetMapping("/start/{programName}")
public ResponseEntity<String> startProgram(@PathVariable String programName)
     {
    MATLABEngineManager.runProgram(programPath, outputPath, controlFlagPath,
         programName);
    return ResponseEntity.ok("Started successfully.");
}
```

Then it invokes the method *runProgram* in the *MATLABEngineManager* class.

```
public static void runProgram(String programPath, String outputPath, String
    controlFlagPath, String programName) {
        if (threadMap.containsKey(programName) && !threadMap.get(programName
            ).isAlive()) {
            threadMap.remove(programName);
        }
        if (threadMap.containsKey(programName) && threadMap.get(programName)
            .isAlive()) {
            return;
        }
        var flagPath = Path.of(controlFlagPath + programName);
        try {
            if (!Files.exists(flagPath)) Files.createFile(flagPath);
        } catch (IOException e) {
            e.printStackTrace();
        }
        try {
            FileWriter myWriter = new FileWriter(controlFlagPath +
                programName);
            myWriter.write(0);
            myWriter.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        // process starts
        var path = Path.of(outputPath + programName);
        try {
            // This runs before getting the engine at the very beginning.
                Otherwise, could check the result of last run.
            Files.deleteIfExists(path);
        } catch (IOException e) {
            e.printStackTrace();
        }
        MATLABEngine engine = getEngine(programName);
        logger.info("Gotten the engine.");
        if (engine == null) {
            logger.error("Engine instance is null!");
            return;
        }
        try {
            // act like environment variable (path)
```

```
37             engine.feval("addpath", programPath);
38         } catch (InterruptedException | ExecutionException e) {
39             e.printStackTrace();
40         }
41         logger.info("Making new runner to run program " + programName);
42         var runner = new ProgramRunner(engine, programPath, outputPath,
               programName, statusMap);
43         Thread thread = new Thread(runner);
44         thread.start();
45         threadMap.put(programName, thread);
46         statusMap.put(programName, STATUS_RUNNING);
47 
48     }
```

And in order to implement the multi-thread feature, in line42 in the code above, I instantiates a class named *ProgramRunner* which inherits from *Runnable* interface. Then override a method called *run* inside the *Runnable* interface. Inside the *run* method, line20 is where actually use the MATLAB engine to start running the MATLAB program.

```
1 public ProgramRunner(MATLABEngine engine, String programPath, String
      outputPath, String programName, HashMap<String, String> statusMap) {
2         this.engine = engine;
3         this.programPath = programPath;
4         this.programName = programName;
5         this.outputPath = outputPath;
6         this.statusMap = statusMap;
7     }
8 
```

```java
    @Override
    public void run() {
        try {
            // act like environment variable (path)
            var path = Path.of(outputPath + programName);
            try {
                if (!Files.exists(path)) Files.createFile(path);
            } catch (IOException e) {
                e.printStackTrace();
            }
            // Output file will be written by MATLAB program.
            engine.feval(programName);
        } catch (InterruptedException | ExecutionException e) {
            e.printStackTrace();
        }
        finally {
            statusMap.put(programName, MATLABEngineManager.STATUS_FINISHED);
        }
    }
```

Terminate Program

The same logic as the start program process, after clicking the terminate button it trigger the *handleTerminate* function in the *ControlButton* class. Using @*GetMapping* annotation, it sends the request to back-end *terminateProgram*. The main process is in *terminateProgram* of the *MATLABEngineManager* class.

```java
public static void terminateProgram(String outputPath, String progressPath,
    String controlFlagPath, String programName) {
        var flagPath = Path.of(controlFlagPath + programName);
        try {
            if (!Files.exists(flagPath)) Files.createFile(flagPath);
        } catch (IOException e) {
            e.printStackTrace();
        }
        try {
            FileWriter myWriter = new FileWriter(controlFlagPath +
                programName);
            myWriter.write(1);
            myWriter.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        threadMap.remove(programName);
        statusMap.put(programName, STATUS_STOPPED);

        // Delete output file.
        var path = Path.of(outputPath + programName);
        try {
            Files.deleteIfExists(path);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

Check Result

This feature allows you to check the MATLAB output in real-time, showing in a popup modal. The modal has both text and image output.

```java
    Path outputFilePath = Path.of(outputPath + programName);
    logger.info("checkResult MATLAB program: " + programName);
    if (!Files.exists(outputFilePath)) {
        logger.warn("output file doesn't exist. Check later.");
        return ResponseEntity.ok("Still running, check later");
    }
    String text = null;
    try {
        text = Files.readString(outputFilePath, StandardCharsets.UTF_8);
    } catch (IOException e) {
        e.printStackTrace();
    }
    result.setTextResult(text);
```

Figure Output For the back-end of figure output, it reads bytes from the output file and encodes into base64 scheme.

```java
Base64 base64 = new Base64();
ArrayList<String> imgs = new ArrayList();
try {
    Files.list(new File(outputPath+programName+"_img").toPath())
            .forEach(path -> {
                try {
                    String encodedString = new String(base64.encode(Files.
                        readAllBytes(path)));
                    imgs.add(encodedString);
                } catch (IOException e) {
                    logger.error("Something wrong with img converting to
                        base64.");
                    e.printStackTrace();
                }
            });
} catch (IOException e) {
    e.printStackTrace();
}
ProgramResult result = new ProgramResult();
result.setImgResult(imgs);
return ResponseEntity.ok(result);
```

For the front-end, browser will decode the base64.

```
async function handleCheckResult() {
        await fetch('/program/checkResult/' + programName, {
            method: 'GET',
            headers: {
                'Accept': 'application/json',
                'Content-Type': 'application/json'
            },
        }).then(res => {
            return res.json();
        }).then(data => {
            let text = data.textResult;
            let imageStrings = data.imgResult;
            let images = Array();
            imageStrings.forEach(
                imgString => {
                    let src = "data:image/png;base64," + imgString;
                    images.push(src);
                });
            setData(text);
            setImages(images);
            setSize("lg");
            setShowModal(true);
        });
    }
```

Also, in order to improve the user's experience, I imported *TransformComponent* in the *PopupModal* class to implement the zoom feature using wheel on mouse.

```
images.map((imageSrc) => {
    return (
        <TransformWrapper wheel={{step:0.1}} maxScale={4}>
            <TransformComponent>
                <img src={imageSrc} className="img-thumbnail" width={800}/>
            </TransformComponent>
        </TransformWrapper>);
})
```

Chapter 4

Conclusions and Future Work

In order to improve the working efficiency and the user experience, I accelerate the OCT data post-processing from both code level and practical application level.

For the first part, a GPU card was added to the process including the apply calibration and fft part, then the vectorization operation for apply calibration and processing attenuation image. The apply calibration function is 10 times faster than the original version and get depth profile function is 2 times faster using GPU. So for Lundquist system, the data processing was sped up to 10 times for the whole process except the save result code and display figure code and if including all the codes, the program sped up to 3 times in total. For other OCT systems, especially those swpet-source systems that have huge amount of calibrations, the final result after acceleration is even better than Lundquist system.

Second part of the project is implementing a website that lab members can start the selected program as well as receive the output of the program and check the status through the web UI on their mobile devices. This application allows multiple client to control the MATLAB engine, providing users a concise UI and easy hands-on experience. This practical application also frees the researchers from the constraints of monitoring the data processing, allows them to work anywhere and take the usage of data processing time to do something more meaningful which greatly save time of the researcher on repeated and boring workload. I chose React as the front-end framework and Springboot as the back-end framework. I used restful API to implement the interaction between front-end and back-end. I also utilized Java to call MATLAB API to implement the remote control part.

Future work on first part is to do code optimization, algorithm optimization and hardware optimization, such as the other different methods I mentioned in the thesis previously including the CUDA and MEX programming. Also we can try distributed computing to decrease the execution time for huge data.

Future work on the web application can include implement the feature of allowing users to input some parameters and interacting with MATLAB. In addition, to improve the users experience, a waiting list feature can be included in the ONI website, so if it has already reached the limit number of programs(according to the CPU and GPU performance), the new request will be put on the waiting list and when the previous code finished running, a notification will be pushed to the previous person to say that your task is completed then a notification to the next person on the waiting list in real-time via email to say your program start running. Thus, the processing time will not slow down when too much requests are received and make better usage of CPU and GPU. In order to further improve the efficiency, real-time notification to the mobile devices is also part of the future work, once there is an error message or the code finished running, so that people can do some adjustment or do the next step.

Bibliography

[1] Boy Braaf et al. “Phase-stabilized optical frequency domain imaging at 1-µm for the measurement of blood flow in the human choroid”. In: *Opt. Express* 19.21 (Oct. 2011), pp. 20886–20903. DOI: 10.1364/OE.19.020886. URL: http://opg.optica.org/oe/abstract.cfm?URI=oe-19-21-20886.

[2] Hansford C. Hendargo et al. “Doppler velocity detection limitations in spectrometer-based versus swept-source optical coherence tomography”. In: *Biomed. Opt. Express* 2.8 (Aug. 2011), pp. 2175–2188. DOI: 10.1364/BOE.2.002175. URL: http://opg.optica.org/boe/abstract.cfm?URI=boe-2-8-2175.

[3] B. J. Vakoc et al. “Phase-resolved optical frequency domain imaging”. In: *Opt. Express* 13.14 (July 2005), pp. 5483–5493. DOI: 10.1364/OPEX.13.005483. URL: http://opg.optica.org/oe/abstract.cfm?URI=oe-13-14-5483.