# FIT 3162 Computer Science Project 2

# FINAL REPORT

## GPU Acceleration for Raster Filter Using APARAPI

Group 9

Christine 29392888

Wan Jack Lee 28848551

Zisong Liao 28418107

# Table of content

# 1. Introduction

Shaded relief image is a raster image that shows changes in elevation using light and shadows on terrain from a given angle and altitude of the sun, derived from the digital elevation model. It is stored in raster form. It has filtered out the non-important terrain feature and accentuates shaded relief features of it. There is plenty of usage of shaded relief images such as cartography research. By applying filters to it, the shaded relief image can remove the unused terrain details. For instance, the mask filter filtered out the edges of the mountains and underlined the low-altitude platform.

To generate shaded relief images, we can use Eduard to apply mask filters to the digital elevation model. Eduard is a piece of software which was generated by the researchers at Monash Faculty of Information Technology and ETH Zurich, Switzerland. It is a JAVA-developed software and thus it is cross-platform supported. All of the raster filters in Eduard used Java Thread Algorithms to achieve concurrent programming. It can be further accelerated through the usage of the Graphics Processing Unit.

There are various competitors of GPGPU technique available such as CUDA, OpenCL, and Metal API. After several discussions, the acceleration technique that was chosen is APARAPI, the reason for choosing APARAPI as the acceleration technique out of those candidates is that the APARAPI is operating system independent. APARAPI is the Java library which compiles the JAVA code to OpenCL system calls during the runtime.

The main focus of the project is to optimize the mask filter algorithm with APARAPI and integrate the Eduard software with the APARAPI acceleration project. As there are changes on this semester, some of the plans have changed. Thus, these reports will conclude the project management plan, methodology that was used, and the project outcomes.

# 2. Background - Literature Review

## 2.1. Introduction to Literature Review

In this section, we will present our literature review on the topic. The purpose of having this literature review is to learn and understand about GPU acceleration, Aparapi framework, and shaded relief image with raster filter. We have done a more general literature review in our proposal, but in this report, we would like to present a more related literature with our project, as we find out what information is essential and what is not.

The aim of our project is to accelerate the GPU in order to make the Eduard program run faster. Most computers today have a GPU installed in it, therefore it is very accessible to most people and the method we will use in this project is applicable not to only this application, but also other similar applications. We needed to learn about how GPUs work and how by using frameworks such as Aparapi, it would run faster. Learning about Aparapi's background and how it functions, in order for it to be applicable to the project. The Eduard program's purpose is to apply filters to a shaded relief image in order to accentuate the desirable parts of it. We have to know the purpose of these filters and how the filters work, which includes the inner workings of its algorithm.

We will first discuss the GPU's background and the interest around general purpose computing. We will also discuss how the GPU can be accelerated and its methods, such as using APIs, particularly a higher level framework, the Aparapi. We will also discuss the shaded relief image and its purpose, along with the filters applied to these images.

## 2.2. APARAPI & Parallel Computing

### 2.2.1. GPU Acceleration

The Graphical Processing Units (GPU) is a processor that is designed to handle complex mathematical calculations in order to generate computer graphics and image processing. The GPU was first seen used in the arcade system boards in the 1970s, however the technology of the GPU then is not as advanced as the GPU processor we see today. The GPU processors available today are much more advanced as, for example, it could generate images of terrains in high detail and at a fast speed, mostly used in computer game graphics. This has been the main focus of the development of the GPU processor and that is why the GPU is designed on the idea of executing instructions in parallel. The GPU is also built rendering engines that are capable of handling math-intensive processes in a short time, which are represented in floating-point numbers.

Due to the GPU's nature of parallel architecture and floating point capacity, the GPU has been the subject of interest in accelerating computer applications outside of graphical purposes. The term General Processing GPU (GPGPU) computing is used to describe this idea. The intention of GPGPU is to offload the work done from the CPU to the GPU in order to finish a particular task at a faster time, mainly for applications with non-graphical and high-parallel requirements.

GPGPU computing models were mostly developed for C or C++ as its programming language, but the increasing popularity of other programming languages, such as Java, has also inspired projects to offload the work of a Java program onto the GPU. In cases of programming languages other than C and C++, they will need to use wrappers, such as the Java Native Interface for the Java programming language. There are two approaches to do this which are providing Java bindings to a lower level language and using a user-friendly API that translates the Java code into the lower level languages. These lower level languages are CUDA and OpenCL, whereas the popular APIs that translated Java into these lower level languages are Java-GPU, Rootbeer, and Aparapi.

GPGPU computing is becoming popular as using the GPU is believed to be faster than the CPU due to its parallel architecture. However, it is difficult to truly compare the performance of the GPU and the CPU, as the optimum algorithm for the GPU may be different to the optimum algorithm for the CPU in the context of comparing for the same application. It is possible to combine the use of the GPU and the CPU for an application as parts of the application can be broken down and be assigned to the GPU or the CPU depending on the reliability of the hardware for the task. The general consensus is that the GPU is better at handling tasks that require processing data and would be advantaged with the GPU's nature of parallelism.

### 2.2.2. Aparapi And Its Reliability in Accelerating the GPU

In GPGPU computing, there are two programming interfaces, which are Open Computing Language (OpenCL) and CUDA. Both OpenCL and CUDA call a piece of code to run on the GPU, however there are differences with these two interfaces. A notable difference is that OpenCL is an open standard for parallel programming for different kinds of processors, such as CPUs, GPUs, and Digital Signal Processors (DSP), whereas CUDA is only compatible for NVIDIA processors.

Comparing OpenCL and CUDA, in an experiment done by Docampo et al. (2013), the performance of CUDA and OpenCL is similar. They have concluded that the actual implementation of the code is the main factor in the performance differences. OpenCL is more suitable for applications on

vector types, such as floats, than CUDA that is suitable for scalar codes in kernels. CUDA does not support much of task parallelism, whereas OpenCL is flexible use of primitives that leads to efficient creation and rearrangement of vector types. It may be more beneficial to use OpenCL when a developer is trying to apply parallelism to an application. Despite these differences, OpenCL and CUDA have similar models and similar syntax in terms of keywords and built-in functions, which allows mapping between OpenCL and CUDA.

Even though OpenCL allows using the GPU to achieve parallelism, it is still hard to implement as it requires the developer to have prior knowledge of C or C++ programming languages, which are not popular to use anymore these days. The developer should write copious amounts of boilerplates just to set up the environment, such as initialising the framework. This does not include the writings for the actual application yet.

Considering these problems, two programming styles have been developed, which are automated approach and semi-automated approach. In an automated approach, low-level details are hidden from the developer. Using the operations, it abstracts data movement and computation. Some of the examples of this approach are Accelerator, LIME, SkelCL, and Chestnut.

A semi-automated approach includes recognizing common parallelizable patterns in the code, such as nested loops, but most would still rely on the developer for recognizing and writing this themselves. These considerations lead to the creation of higher-level GPGPU languages, such as Aparapi and Rootbeer. Both Aparapi and Rootbeer are extensions to the Java programming model, designed to provide developers to write high-level parallel codes. The difference between the two is that Aparapi uses OpenCL and Rootbeer uses CUDA. This roots back to the fact that Aparpai can be used on OpenCL devices, which includes most CPUs and GPGPUs, and Rootbeer can only be used on NVIDIA GPUs.

The Aparapi is developed to allow developers to offload parallelizable sections of a Java code without having to learn low-level languages such as C and C++. Due to OpenCL being compatible with most devices, it makes Aparapi heterogeneously flexible compared to Rootbeer. Even though Aparapi can run on CPUs, the goal is to speed up the code by parallelism. The limitation of a CPU is that it cannot run threads concurrently as much as a GPU, even when it is a multi-core CPU. Therefore focusing on the GPU should be kept in mind when writing in Aparapi and prior knowledge about the GPU is needed.

### 2.2.3. Shaded Relief Image and Raster Filter

Shaded relief image shows changes in elevation of a terrain, this is done by using lights and shadows on terrains. It can be represented in grid, contour, and profile even though the nature of elevation data is continuous. A shaded relief image can be derived from a digital elevation model. One of the applications of shaded relief images is to identify lineaments of different distinct relief and topography, as topographic illumination can be simulated. This technique has also been used in cartography to create an illusion of a 3D relief map. Usually, shaded relief images are used to assist in geological feature recognition, which helps solve problems involving geomorphology and geology researches.

When creating 3D maps, cartographers tried to generalize terrains by removing distracting details and to highlight the landform. Cartographers would remove visually unnecessary details, and to sharpen edges and mountain ridges of a digital terrain. Filters are used to achieve these goals. Filtering is the process of applying the filter on to the terrain. A filter that removes high-frequency details is called the low-pass filter and a high-pass filter accentuates the discontinuities. Low-pass filters are simpler and thus are more commonly used on digital terrains. However, low-pass filter smooths unvaryingly, therefore important features of the terrain are blurred as well, such as sharp edges.

A terrain model can be represented as two-dimensional. The development of a Laplacian pyramid is the same as the development for raster terrain models. Its difference is the data type used, Laplacian pyramid uses integer numbers whereas raster terrain model uses floating point values. Laplacian pyramid is derived from the Gaussian pyramid by putting two levels of the Gaussian pyramid on the same grid and subtracting the corresponding value of the two. As a result produces levels for a Laplacian pyramid.

Other filters that may be better at removing visual noise on digital terrains while preserving important details are suggested. One of it is using a combination of several methods to smooth a 2D dataset, while preserving the discontinuities. A two-step method is also proposed, which first smoothes the field of terrain's normal vectors and then fitting a new terrain model to the smoothed normals. The problem with this method is that it has high complexity and high computation time.

## 2.3. Conclusions

In conclusion, the GPU has the ability to speed up most processes because of its parallelism nature, which creates an interest to apply the GPU for general purposes. This can be done by using APIs such as OpenCL and CUDA. However, it is still hard

to implement as the developer would still need knowledge on programming languages such as C and C++, which are no longer popular these days, and also writing boilerplates. Frameworks such as Aparapi removes these difficulties and allows the developer to implement GPGPU computing with just knowledge on the Java programming language and removes the need to write boilerplates. The shaded relief image is used to see the elevations of a terrain and filters are used to accentuate the features that are desirable and remove undesirable features. Desirable features include edges and mountain ridges, whereas undesirable features include visual noise or unnecessary details.

After doing this literature review, we have more understanding on the subject matter and also point out things that are more relevant to the project compared to the literature review in our proposal. We are able to recognize reasons for the problems we encountered early in the project and the possible development that can be done in the future.

# 3. Methodology

## 3.1. Implementation

In our project, we used GPU to accelerate raster filters in Eduard algorithm, which is an application for creating shaded relief images. The main technique and library we used for acceleration is the APARAPI, which is the high level Java library that abstracts the underlying technology of OpenCL. We were provided the Eduard code which is written in Java, so what we have done was to implement some raster filters in Eduard using GPU-specific languages.
Regarding how we implemented it, the following are some sequential steps.

### 3.1.1. Initial attributes

These attributes are declared initially of the development:

1. inputGridFile: the string of path to input grid file, it was set to null and would be selected in GUI dialog.
2. filename: the string of filename holder.
3. outputGridFile: the string of path to output grid file, it was set to be null and would be selected in GUI dialog.
4. outputImageFile: the string of path to output image file, it was set to null and would be selected in GUI dialog.
5. benchmarkFile: the string of path to output benchmark file, it was set to null and would be selected in GUI dialog.
6. performanceString: the performance string which will show in the benchmark output file.

All of the above are static attributes of the MainLauncher class, which is the driver class that handles the application.

#### 3.1.2. File input

The first thing is to read the input grid from the file. We use the askFile() method in FileUtils class, which is a class of Eduard containing several file related methods. The method will detect the selection in GUI dialog and get the path of the chosen file and return it to the attribute inputGridFile.

#### 3.1.3. Generation of APARAPI grid

Firstly, we call read() method of class EsriASCIIGridImpoter, which returns a grid from a file in specific format, to get the Eduard grid. In addition, we translate the Eduard grid into APARAPI grid for GPU acceleration.

#### 3.1.4. Mask filter execution

This is the main part of APARAPI acceleration, we create a MaskFilter object of the source grid and call the execute() method, which will apply gradient operator, low pass operator and clamp to range operator to the grid respectively. These APARAPI operators all work in GPU by using Kernel command, and finally return the filtered grid.

#### 3.1.5. Grid output

The shallowCopy() method of Eduard is called to create a shallow copy of the input grid as output grid, then we modify it by APARAPI filtered grid to translate to the Eduard output grid.

#### 3.1.6. Image creation

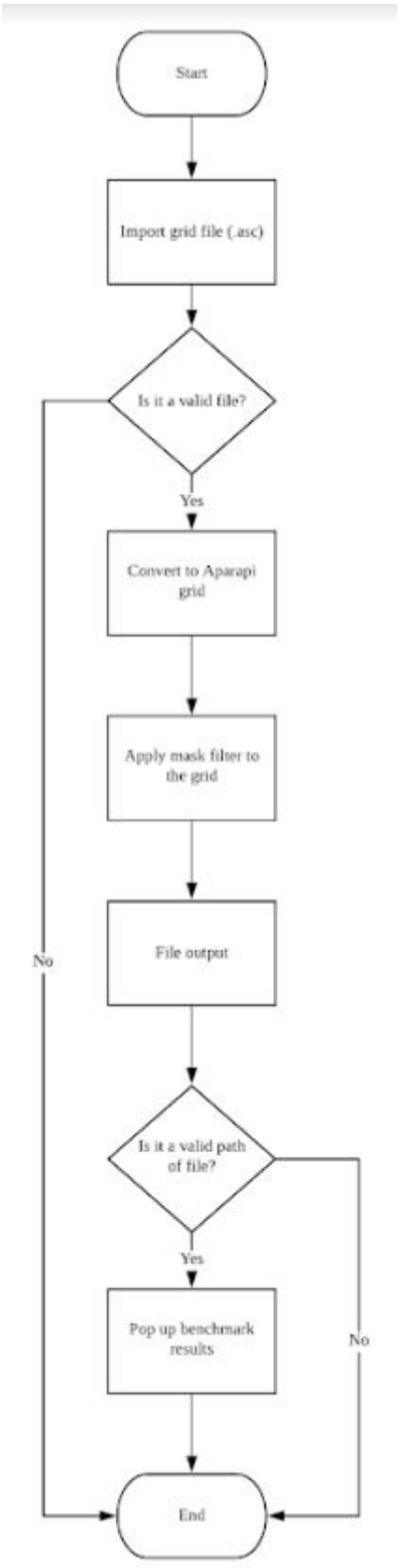
Regarding creating the output image, we get the minimum and maximum value from the grid matrix and use Eduard gridToImageOperator to convert the grid to image. Besides, we also generate benchmark values and export them to a file.

#### 3.1.7. Result output

Finally, we use another GUI dialog to ask users to choose files to export the image and benchmarks to storage.

The following is the flowchart of the project, which shows all processes of the code from the start. All the possible errors are considered and handled,

when they occur, the application will pop up an alert window to tell users the error and end itself, which prevents it from crashing.


Start
Import grid file (.asc)
Is it a valid file?
Yes
No
Convert to Aparapi grid
Apply mask filter to the grid
File output
Is it a valid path of file?
Yes
No
Pop up benchmark results
End

## 3.2. Design

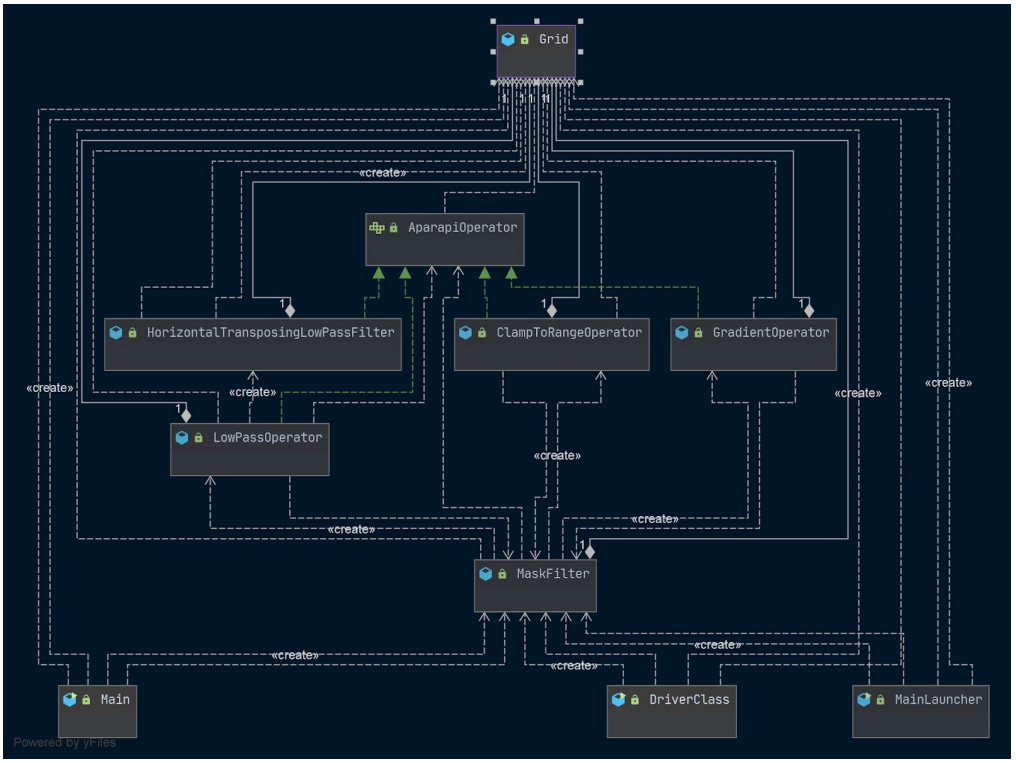
### 3.2.1. Object-oriented design of project

JAVA is an object-oriented language which can make any structure in projects individual and isolated. In our development, the main launcher is the driver class that invokes methods in different objects step by step, while these objects are initialized in separate packages. Every package is responsible for a specific functionality, including file input/output, GUI and raster filter. In our project, some of these packages belong to Eduard code, we just directly use them, and others are written by us. The most significant one is the operator package, which contains APARAPI operators for raster filters.

Regarding how to implement this package, we create an AparapiOperator interface, that is implemented by different concrete operator classes (clamp to range operator, gradient operator and low pass operator). Besides, we create a MaskFilter class to execute all operators to a grid, then we can get the output grid.

The above functionality is only realized by the package itself. Therefore, the package cohesion principle is applied to the design. The operator can be used anywhere if it takes a grid as the input, and it will always apply filters on it and return the output grid. It is reusable and maintainable, and if we want to modify any functionalities of it, we can directly develop on its classes (Jones & Meilir, 1988). Classes and packages outside can simply use it by creating a MaskFilter object and call execute function.
Among different operators and interfaces in the package, we also apply the Liskov Substitutability Principle, which means for each concrete operator class, it conforms to the operator interface (Martin, 2000). The interface has an operate() method and all operators implement this method to apply different raster filters on the grid.

### 3.2.2. Overall design of project




In order to achieve polymorphism, an AparapiOperator interface has been created. The AparapiOperator defines the common method of each Operator, for instance, the execute method, such that it can prevent the code smell. The HorizontalTransposingLowPassFilter (operator), LowPassOperator, ClampToRangeOperator, and GradientOperator are the subclass of AparapiOperator. Each operator has implemented the corresponding algorithm and they are variants of the Eduard original operators. The LowPassOperator is a composition of HorizontalTransposingLowPassFilter and also the low pass algorithm itself. The decision to reimplement the algorithms instead of creating new algorithms will be discussed below.

The MaskFilter object is a composition of each Operator. Apart from that, it has the Slope to Normalized mask algorithms which will be executed after all the operator has been executed. The Grid object (APARAPI) is the counterpart of the Grid object of Eduard. It has important attributes and methods which will be used by these operators.

The MainLauncher class is the main class of the project which will deal with the IO operation and also invoke the mask filter algorithm. The Main class is the original main class which can be used to link with the Eduard ProcessLauncher class. The driver class is only used for testing purposes thus it can be ignored.

## 3.3. Method

### 3.3.1. Hardware Specification

- GPU Acceleration Support

  APARAPI is an open-source framework for executing native Java code on GPU, so the most significant hardware device is the graphic card, which is used for efficiently accelerating the Eduard algorithm by parallel computing. The graphic card is supposed to have powerful performance that can support Open-CL framework on multiple threaded allocation. Our computers and laptops all carry Nvidia GPU or other similar GPUs to meet this requirement.

- Powerful Processor and Storage

  The application is designed to deal with large grid files like maps, there are several raster filters those deal with points in grids concurrently to reduce the total time. Therefore, a powerful processor is necessary to handle this large number of calculations. Our laptops all have Intel Core i5 or i7 processors to support this. In addition, an SSD can also make the process efficient, because it has some large grids and images as input or output, and which can be much faster to be dealt with within an SSD rather than a normal mechanical hard drive. Unfortunately, some of our team do not have large enough SSD to run this project, but we still recommend users to use it.

### 3.3.2. Software Specification

- Version Control

  Version control is particularly significant for synchronisation of file version and backup commit of file changes. In our development lifetime, there were multiple changes of codes taken place by each team members and on their own personal computers. To allow us to work on them simultaneously, Git version control mechanism is quite suitable for our project since it provides a platform that allows us to make entire control on coordinate works.

  Our first chosen is the Monash Gitlab, since we are all familiar with it because of use in some units before. However, Github has more features and is more convenient to use, it is also better for maintenance of codes. Thus, Github is more suitable for us.

- Programming Language and Library

  The provided Eduard code is written in Java, so we chose Java as the programming language. At first, we were using previous JDK version 12.0.2 to implement, and after updating by Java, that version was deprecated, we finished code by using JDK version 13.0.

  APARAPI is the chosen Java high-level API for the GPU acceleration, it is a cross-platform library which applies threads on a graphic processor unit and can be executed on different platforms like Windows and MacOS. We all have devices with the Windows operating system, so APARAPI is a good tool for our project. We use Maven to inject the APARAPI library into codes, where a powerful repository for searching and exploring Java libraries. We add the APARAPI dependency from it instead of directly downing the whole library.

- Development Environment

  Since the programming language we use is Java, there are two popular development environments that can be used on Windows, Eclipse and Intellij. Our previous selection is the Eclipse, as we get used to it. However, during the development, we found that Eclipse is not convenient enough to connect with Github. On the contrary, Intellij supports Git service better, and also has a backup IDE with Visual Studio code. Therefore, we finish our implementation on Intellij.

- File Sharing and Communication Tool

  For documentations like reports and graphs, we use Google Drive for work sharing and allocation. It provides a platform for us to know all members' progress on tasks. Any changes or edits can be noticed immediately after someone makes them, which makes it easier for us to share ideas. If one has any suggestions for improvement about others' work, he can make comments on that part, so Google Drive is also a useful team communication tool

  Regarding code file sharing, as mentioned above we use Github. Generally, we create a git repository for the project, and clone it to our local storage. Then we connect local codes with Github by using intellij. Whenever we make any changes on the local repository, we just commit them and push them to the Github repository, then the change can be accessed by all members in the group. Github reduces many overlaps and mistakes due to incomprehension of

others' works, all team members seem to work on the project concurrently.

We also use Messenger for general communication, like discussing the time of meetings and asking basic questions. It is not as technical as specific tools, but more convenient than them.

# 4. Project Management

## 4.1. Project conceived or managed

### 4.1.1. Conceiving the Project

The project is conceived from the original Eduard program, which is provided by our project sponsor. We first analyze and study the code in order to understand the program. We only focused on the methods and classes that are related to the process of applying the raster filter on to the input file. There was some confusion at first and it creates a block in the conceiving process. After consulting with our tutors and other teams, we were able to figure out the way that we could apply Aparapi to the Eduard program.

The initiation of the project started with preparing all of the member's laptops with the essential tools. We have installed Maven, updated Java to the latest version. We used IntelliJ as our Java Integrated Development Environment (IDE) and we also installed Visual Studio as our backup IDE. Git is set up on our computers so it will be able to connect to the project repository. A new repository for the project is created on GitHub and everyone is made sure to connect to it. The project is then injected with Aparapi dependency.

During the development of the project, we communicate mostly through Facebook Messenger and Zoom. We initially had Skype as our main conference tool, but as Zoom got popular and convenient, we switched to Zoom as our main conference tool. For file sharing, we used the same shared drive as FIT3161's project on Google Drive and we control and check our progress through Microsoft Planner.

### 4.1.2. Management

In our proposal, we have prepared a Work Breakdown Structure that we were planning to execute during the semester break and during the second semester. However, due to communication problems, we did not follow through our initial plan. At the start of the second semester, we created a new plan that is also reflected onto a Work Breakdown Structure. In this plan, we evaluated the time that we have for the project and we realized that there are parts that we may not be able to finish. In this case, we prioritized the functionality of the program over the user interface. In this plan, we also take into consideration the deliverables that we have to prepare that are not directly related to the development of the project.

Even though we have prepared a Work Breakdown Structure, we have decided to have a weekly online meeting, which is done through the Zoom conference software. In these meetings, we would discuss the progress of

our project. From these discussions, we set our goals for the week and then assign the tasks to each member. These tasks are tracked in the group's Microsoft Planner, as shown in Appendix E. Also, when the project is updated, we would notify the Messenger group chat and remind them to pull from the repository before making any changes to the project.

## 4.2. Project Management Approach

We have decided to switch to the waterfall approach for our project. The reason for the switch is because we need to wait until the previous module to be finished before working on the next module. It is difficult to apply the Agile approach as we cannot predict what is needed for the next module.

We implemented the waterfall approach by first recognizing what the scope of the project is. This includes the goal of the project, what we should present, and what has to be produced by the end of the project. We know that the goal of the project is to accelerate the application of the raster filter by using the GPU, which has to be done by using the Aparapi framework. The things that have to be delivered during the duration of the project are presentations, journals and reports on the team management and the project itself. By the end of the project, we need to deliver a working code which function is to apply the raster filter on a shaded relief image at a higher speed than the original Eduard code.

We first came up with a simple design of the planned program. This was reflected on a basic sequence diagram that we presented during the interim presentation which we have attached in Appendix F in this report. We worked on the project with this simple design in mind in order to make sure the functionalities of different modules are kept in its appropriate sections. We implemented the waterfall approach when working on the functionalities of the project by waiting for a module to be finished first before working on the next module. We were not able to move on to other modules first before the current module is done. Therefore, when the project is stuck on a certain part, we would try to solve it as soon as possible, as otherwise we will have an even lesser time to complete the project.

During our final presentation, we were able to present the final project to our project sponsors. In the code demonstration, we showed how we have applied Aparapi to the raster filter. Though the overall time of the program is not faster than the original Eduard program, we were able to apply the acceleration to most functions of the raster filter. The final product of this project was verified by the project sponsors. This is how we have implemented the Waterfall approach.

## 4.3. Project Resources

One of the most important resources is this project is the original Eduard code. The original Eduard code is given by the project sponsor. As our aim in this project is to accelerate the Eduard program, we have to look into how Eduard works. We looked into the functionality of the program, such as the algorithm of the Main class, and branched out to understanding the purpose of the classes that are used in the Main class and also the methods called from it. Most of the explanation of the purpose of the code comes from the docstring that is written in the code itself, however there is no further background on the purpose of the method or class and there is also lacking written documentation provided and online.

We also received 8 sample Action Script Communication files of different kinds of terrains. These are the files that will be inputted into Eduard and applied the raster filter too. These sample Action Script Communication files were also provided by the project sponsors, thus we believe that these files are appropriate for testing whether the produced program fulfills the requirements of the project.

We are aiming to accelerate the GPU for the Eduard program by injecting the Aparapi framework into it. The Aparapi framework is cloned from the Syncleus Github repository and then injected into our project. Resources regarding the background, setting up, and documentations on the Aparapi is available on aparapi.com.

## 4.4. Project Execution

The execution of the project was initially slow. After the long semester break, we needed to refresh our memories on the project. We also needed to do further research on the Eduard code and Aparapi as we realized that our knowledge was not sufficient enough. We also realized that there was not that many documentations on Eduard and Aparapi in the first place. Due to this, there was a moment where we were stuck early in the project. We were able to overcome this by consulting with out tutors and other teams that were doing a similar project.

From the information we got, we were able to move on to the next part of our project, while also conceiving a simple sequence diagram of our program. Considering the time lost when we were stuck and the potential problems we might face in the future, we decided to prioritize the functionalities of the project over the user interface. With that in mind, we focused only on the functionality aspect of the program. In our weekly meetings, we evaluate the progress of the project and then evaluate the time we have before presenting the program. We tried to keep up with the plan created but also consider the member's workload for other units.

When we were working on the different operators of the raster filter, we encountered some difficulties working on the Low Pass Operator, due to its complexity and the limitations of the Aparapi framework at the duration of the project. Even though this operator can't be accelerated, due to the time limitation, we proceed with this result and decided to write the idea that this part could be improved on in this report instead.

## 4.5. Risk management

In our proposal, we have identified four risks that could affect our project. Due to COVID-19, we were able to identify another risk and one of the already existing risk's possibility increased. We have evaluated the risks identified and also reflect on how it affects the project.

Referring to the updated risk register, we identified a new risk this semester and it is ranked first in the updated risk register. This risk is the possibility of that change of tasks. Due to the COVID-19 pandemic, the university is closed and everyone is advised to stay at home and practice social distancing. Due to the quarantine, there was a possibility of tasks being changed due to the limitations. During the development of the project, there are changes to the specifications of the project, such as doing a pre-recorded interim presentation instead of a face-to-face one. There are also indirect changes to tasks caused by this risk, such as the inability to discuss in person leads to the prevention of collaborating on certain tasks, therefore there are more tasks assigned to the members than if we were able to collaborate. This was marked as the highest risk as social distancing was already in practice and therefore the probability of the risk happening is very high and it affects the development of the project. There is not much that we could do to control this risk as the changes in specification of the project is not within our control, but we tried to collaborate on some tasks by conversing through Messenger on ways to solve the problem.

The second risk is a risk that was identified in the making of the project proposal, which is the deficiency of time. We identified this as a risk as we were not sure if the research we did was sufficient enough for developing the project and there is a possibility that this might lead to having to do more research on the subject matter, which would result in delay on the progress of the project and possible underestimation on the overall time taken to complete the project. The possibility of this risk happening is also amplified by the COVID-19 pandemic, which causes hindrance of face-to-face communication.

During the development of the project, we did encounter some problems that caused delay to the project's progress as we are stuck on a certain problem. We tried controlling this risk by changing our priorities and tasks distributed. As we

realize that there might not be enough time to complete the project as planned, we prioritized the functionality of the project and then put the user interface of the project as the last of our priority, which means that we will only complete the user interface if time allows. However, as we develop the project, we realize that it is not possible to complete the user interface with the remainder time, therefore there is very minimum user interface in the project, such as the use of JFileChooser to provide a window for choosing the input file and the output folder.

The third risk is the extension of scope. This risk was based on the assumption that as our understanding of the project is lacking, there might be more things that we needed to do compared to what was initially expected regarding the functionality of the project. While developing our project, we realized that we needed further understanding of the Eduard program and the Aparapi framework. Documentations were limited for both Eduard and Aparapi, but we were able to control this risk by seeking help from our tutors through consultation and discussing with other groups that seem to be having similar problems as ours, regarding the Eduard program.

The fourth risk is the risk of having a small amount of meetings. We identified this as a risk as there is a possibility of not having all the members available during the 3 month semester break. All of the members are going to be travelling outside of Melbourne, therefore the group was not able to hold a face-to-face meeting and there is a small chance of having an online meeting as well due to the differences in timezone in the place that the members were travelling to. We tried to control this risk by instead communicating through the group's Messenger chat and occasionally update on the initiation of the project. However, when the COVID-19 pandemic happened, we were not able to have a face-to-face meeting for the rest of the project. We then decided to communicate through online meetings and also set a weekly meeting in order to keep everything on track.

The fifth risk is the possibility of using the Aparapi framework to accelerate the GPU won't accelerate the Eduard program. We identified the causes to be the member's computer configuration problem and the lack of understanding on the Aparapi framework. During the development of the project, we were able to make sure that the configuration and the specification of each member's computers is a suitable environment for the development of the project. However, the risk arises due to the lack of understanding of the Aparapi framework and the Eduard program. We were able to successfully use Aparapi to accelerate parts of the Eduard program, but we were not able to accelerate the LowPassOperator part of the Eduard program. This is due to the limitation of the kernel that makes it impossible to check the previous row of the grid whereas it is essential in the LowPassFilter which allows the original program's LowPassFilter to be faster. This drawback causes the whole project to be slower than the original Eduard program, even though it is faster algorithm wise as it is run in parallel. There is no actual way to control or avoid this as the purpose of the project is to use the Aparapi framework, which is a criteria that has to be changed to actually allow the LowPassFilter to be accelerated.

### 4.6. Limitations

One of the limitations that is directly related to the project is the lack of documentations. The documentations on the Eduard code are only limited to the docstrings that are written in the code and the in-code comments. This lack of documentation results in the limitations on understanding the purposes of the code. The docstring provided in the original Eduard code gives only a simple explanation on the purpose of a method but it doesn't provide on how exactly the code works. We will have to analyze the code line by line to understand the algorithm of the method, which results in the extension of the time needed when understanding the original Eduard code.

Another limitation directly related to the project is the lack of documentation on the Aparapi framework. The reliable resources are only limited to the website aparapi.com, which does not contain enough documentations to aid the project.

The COVID-19 pandemic is indirectly related to the development of the project. The restrictions and the dangers of the situation does not allow meeting face to face to be possible. As we are not allowed to meet, it is more difficult to discuss and address issues regarding the project. Solutions such as regular online Zoom meetings are also dependent on other factors such as the reliability of the Internet connection and the limitations of the Zoom conference software.

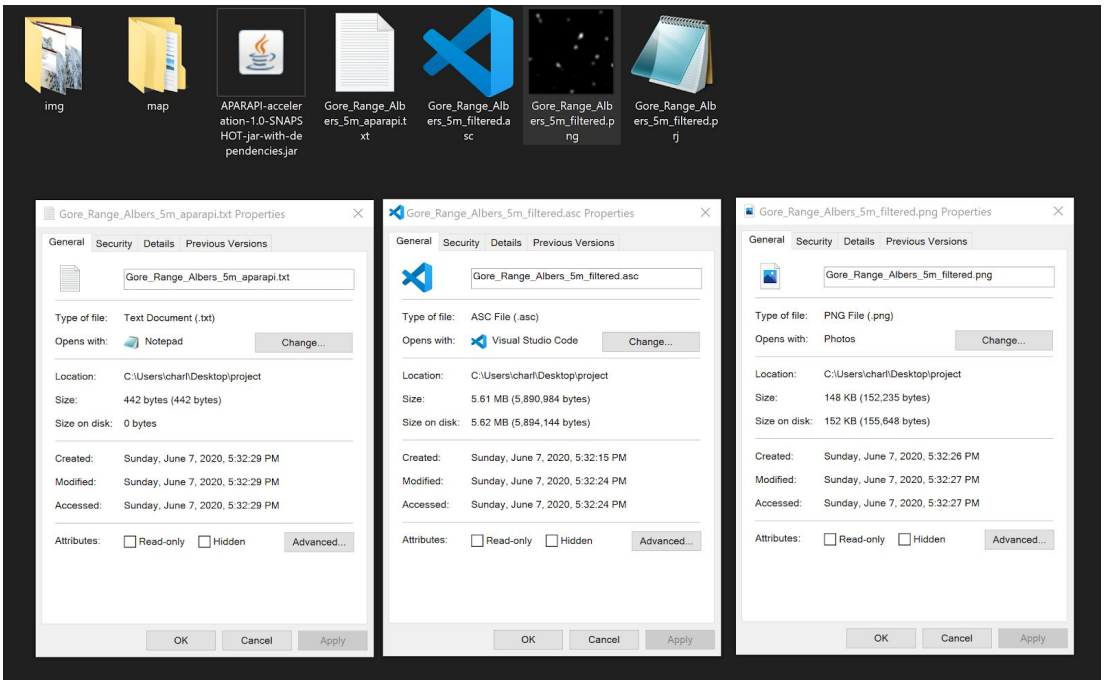
The team morale has also reduced as there may seem to be a lesser amount of responsibility than there actually is. There are instances where members neglected their task because they were trying to prioritize tasks outside of the project and also instances where when a message is sent, there is a prolonged duration to receive a response. This causes the delay of the progress on the project.

## 5. Outcomes

### 5.1. Results achieved and product delivered

In this project, the product that was created is the APARAPI accelerated mask filter generator. When executing the project, there is a file selection window pop-out to prompt the user to choose the input raster file. After the subroutines are executed, there are three file selections windows pop-out to prompt the client to export the post-processed grid file, post-processed filtered image, and performance benchmark file. Besides that, when the execution is done without any error input after all files have been exported, an execution done message will be displayed. However, if the input file type is not compatible, an error message will display and

the execution will be ended. Also, if no input file is given, a warning message will display and the execution will be ended.




The final product of the APARAPI accelerated mask filter generator are the output raster file, a shaded relief image which is generated, and a performance benchmark file which includes the time used to generate the mask. As the background application runs on top of the operating system, the software will not access any sensitive data.

The output raster file of the APARAPI accelerated mask filter has been proven to be identical to the output raster file of its counterpart(Eduard software). The two output raster files are compared using the unit test cases of the APARAPI acceleration software.

## 5.2. Requirements

There are several requirements (scope) in the project and we will list out all the requirements that we have done. There are two categories of requirements: functional requirements and non-functional requirements in this project.

The functional requirement of the project is that the client input a raster grid file (with 'asc' as the file extension name). The raster grid file will include a large amount of data point values which represent each grid. Once finished processing, the software will output the post-processed raster grid file. An image version of the mask filter grid will also be output such that it can preview the raster mask filter easily. These functional requirements are done with using the Eduard classes. The defined classes

have provided the method which can be used to read in and write out the raster grid files, for instance, the EsriASCIIGridExporter will export the grid data to a file and the EsriASCIIGridImporter will read in the grid file and generate an Eduard Grid object. However, due to the separate concern principle, the APARAPI Grid object cannot be directly translated (or converted) to Eduard Grid object and vice versa, thus another layer of linking them has to be implemented. Besides that, the inbuild ImageIO library is used to generate the grid image. Achieving the functional requirement can be done without using a complex GUI.

The other functional requirements include generating a benchmark for the software. The timer method from the inbuilt Java library is used to calculate the time consumed by each APARAPI kernel execution. The timer is started when the kernel is invoked by the GPU whereas the timer is stopped when it is done. The timer doesn't include the time to instantiate the used variables and data. Apart from that, a simple graphic user interface is provided for better user experiences as the client now doesn't have to deal with the command line user interface. In order to provide such functionality, the JFileChooser is used.

The non-functional requirement is that the software should be OS-independent. As the main development language is JAVA, it will be cross-platform supported and thus it can be executed in Windows, Linux or Mac operating systems. The underlying technology of APARAPI is OpenCL and OpenCL is widely supported by most of the graphic cards. As APARAPI will convert the system-call to OpenCL, this means that the software will be cross-platform supported.

## 5.3. Justification of decisions made

In this section, we will discuss the decisions made during the development stage and the justification of decisions made.

The reason that we modify from the original algorithm instead of creating or using a new algorithm is that we can easily reference the speedup. One of the requirements of the project is that implement the selected filters that run slowed in Eduard, thus modify the original algorithms can compare the speedup between the Eduard algorithms with APARAPI-accelerated algorithms.

Apart from that, rather than reusing the Eduard classes, we have created new classes which use similar attributes or methods. One of the reasons is that we initially want to minimize the usage of the Eduard classes, thus reducing the dependencies across two software, this has ensured the separation of concern. Besides that, the APARAPI project has developed in a separate repository thus it will be hard to direct references to the Eduard classes.

Moreover, in the anonymous Kernel object in each APARAPI operator, the reason that some of the methods have been copied instead of using the Eduard or APARAPI classes is that APARAPI doesn't support Object-oriented design. There was an attempt to create an abstract Kernel class such that other Kernel classes inherited the method of the abstract classes. However, it will trigger an exception and APARAPI will fall back to the JAVA Thread Algorithm, which will have an expensive computing cost.

Furthermore, there is a simple Graphic User Interface for input the grid file and export the post-processed grid file. The rationale behind the decision is that it can provide better user experiences such that a client doesn't need to deal with the command user interface.

## 5.4. Results

The Eduard software has fully integrated with the APARAPI pieces of software. There is no modification of code needed to execute the software and read-write file. A simple Graphic User Interface is provided to select the input file and the location to save the output files.

The benchmark of the program has shown that the speedup of most of the operator function is significant compared to the Eduard operator. As shown in the test report, the Copy Void operator of APARAPI-based program is 105 times faster than its counterpart in Eduard software, follow by the Gradient operator of APARAPI-based program which achieved the speedup of 47.34 times compared with Gradient operator of Eduard. The APARAPI version of Clamp to Range operator is 32 times faster than the Eduard version of Clamp to Range algorithm, while the APARAPI version of Slope to Normalized Mask operator is 21 times faster than its counterpart.

However, an interesting result is discovered in the test report. The Low-Pass operator of the APARAPI based software is significantly slower than the Low-Pass operator of the Eduard program, which is 1 times slower than the Eduard software. Besides that, the execution runtime of the Eduard software is significantly faster than the APARAPI implementation. Thus in terms of performance, the speedup for each algorithm has achieved however the overall software runtime is slower.

Based on the performance test and integration test, we can conclude that the acceleration of each algorithm has succeeded, but the runtime of the APARAPI based program is slower than the Eduard.

## 5.5. Limitations of project outcomes

As discussed on the result and test report, we knew that there are several limitations of the project outcomes. For instance, the performance test shows the software has

a faster algorithm runtime compared with Eduard. However, the overall execution time of APARAPI implementation takes longer than Eduard. The reason for causing the execution time to slow down is the limitation of APARAPI. All the array parameters used by APARAPI are required to transport to the GPU memory before APARAPI executes its kernel at runtime and the results will have to manually move back to the main memory after the computing has done.

Another reason for dragging the performance is caused by one of the operators. The Horizontal Transposing algorithm requires read-in multiple rows of the array concurrently, however it is not supported by APARAPI. Thus, the algorithm cannot fully utilize the benefits of GPGPU speedup.

Moreover, the Graphic User Interface that provided by APARAPI is rather simple and no extra function provided, for instance, the preview of raster filter image. Thus, it might be not that user-friendly as all of the output files can only be opened after they are exported. However, the GUI is rather straightforward and it is foolproof and convenient.

## 5.6. Future works and improvements

Several future works can be done to fix the limitations of code. For instance, class compatibility is another related determinant to consider, refactoring the APARAPI classes with Eduard classes can reduce the translation overhead of the object. As shown in the test report, the APARAPI Grid object is not fully compatible with the Eduard original Grid object, thus it requires an extra layer to convert the object (or vice versa). Thus we suggest that the modification of both Eduard and APARAPI classes make them compatible with each other.

Besides that, to resolve the slow runtime of the Horizontal Transposing algorithm, we may need to "invent" or discover another algorithm which has the correct output while reducing the execution time significantly. Apart from this, another acceleration method such as OpenMP library can be applied to the operator to mitigate the issue. Furthermore, the original Java Thread algorithm can be used to mitigate the problem. As proven on the test report, the Abstract Frequence algorithm provides the same output of the Horizontal Transposing algorithm. Thus, besides the need of translating the grid object, the abstract frequency algorithm of Eduard can replace the horizontal transposing algorithm of APARAPI.

Other than that, extra features such as generating line integral filters can be implemented through APARAPI library. A rather more complex Graphic User Interface can be provided to enable a-more-user-friendly experience for clients. Those functionality can be provided on the next update of the software.

## 5.7. Other matter

There is no other issue or matter which to be discussed in this section. However, the results of the project have proven that the APARAPI acceleration technique can and only can be applied on the 1-dimension datasets. If some of the operations on the datasets required multidimensional direct access on it, APARAPI acceleration technique might bring overhead toward it.

# 6. Conclusion

## 6.1. Cohesive conclusion

We would like to express our deep appreciation to FIT3162 Team 3 Aiman Iskandar bin Murhiz, Timotius Putra Pujianto, James Dean Schubach) for their helpful and valuable assistance on the software execution algorithm as well as the integration of software.

To finish the project, all of the team members have done some research on the General Purpose GPU acceleration. One of the reasons that the chosen acceleration technique is APARAPI is that it is straightforward to use and there is no difficulty to deal with it. However, as APARAPI is still being improved by various contributors, it lacks vital functionality that can reduce the overhead of some certain calculations. Furthermore, the research on the raster filter has shown the importance of the shaded relief image and the usage of it, for example, for geological feature recognition and generating 3D maps illusion.

During the developing stage of the project, the project management approach has been changed from the Agile methodology to Waterfall methodology. The reason for switching the methodology is due to the pandemic outbreak. The lack of face-to-face meetings or brainstorming sessions has reduced the flexibility of adapting Agile methodology. Besides that, the deficiency of time has indirectly impacted the team project. Each member has suffered the inconvenience of the pandemic outbreak as all on-campus education activities have transitioned to online classes.

Apart from that, we would like to conclude that we have successfully achieved the project within the scope. There are several scopes in the project. For instance, it is possible to achieve acceleration of mask filter generating algorithms using APARAPI. As shown in the performance test, most of the algorithm has achieved a significant speedup. Although the overall software execution speed is more gradual than the original Eduard software, the reason that causes the issue is due to the limitation of APARAPI. It can be mitigated by using other technology to parallelize the algorithms. Besides that, a benchmark has been suggested as a reference to compare the speedup with the original software. The raster image that is generated such that the client can preview the post-processed effect.

There are future works that can be done to further improve the software such that it can provide a better user experience. The parallel technique that was learned through this project can be moreover applied to other real-world situations.

# 7. References

Page-Jones, M. (1988). The practical guide to structured systems design (2nd ed.). Englewood Cliffs, N.J.: Prentice Hall.

Martin, R. C. (2000). Design principles and design patterns. Object Mentor, 1(34), 597.

A. Padyana, C. D. Sudheer, P. K. Baruah and A. Srinivasan, "High throughput compression of floating point numbers on graphical processing units," 2012 2nd IEEE International Conference on Parallel, Distributed and Grid Computing, Solan, 2012, pp. 313-318, doi: 10.1109/PDGC.2012.6449838.

Leist, A., Playne, D. P., & Hawick, K. A. (2009). Exploiting graphical processing units for data-parallel scientific applications. Concurrency and Computation: Practice and Experience, 21(18), 2400–2437. https://doi.org/10.1002/cpe.1462

Docampo, J., Ramos, Taboada, Exposito, Tourino, & Doallo. (2013). Evaluation of Java for General Purpose GPU Computing. 2013 27th International Conference on Advanced Information Networking and Applications Workshops, 1398–1404. https://doi.org/10.1109/WAINA.2013.234

P. Joseph, I. (2014). Accelerating java on embedded gpu (ProQuest Dissertations Publishing). Retrieved from http://search.proquest.com/docview/1521737324/

Nasira, G. (2014). Exploring the Contrast on GPGPU Computing through CUDA and OpenCL. i-Manager's Journal on Software Engineering, 9(1), 1–8. https://doi.org/10.26634/jse.9.1.3208

Segal, O., Colangelo, Nasiri, Zhuo Qian, & Margala. (2015). Aparapi-UCores: A high level programming framework for unconventional cores. 2015 IEEE High Performance Extreme Computing Conference (HPEC), 1–6. https://doi.org/10.1109/HPEC.2015.7322440

Albert, C., Murray, A., & Ravindran, B. (2014). Applying source level auto-vectorization to Aparapi Java. Proceedings of the 2014 International Conference on Principles and Practices of Programming on the Java Platform, 13, 122–132. https://doi.org/10.1145/2647508.2647519

Harvey, P., Hentschel, K., & Sventek, J. (2015). Parallel Programming in Actor-Based Applications via OpenCL. Proceedings of the 16th Annual Middleware Conference, 162–172. https://doi.org/10.1145/2814576.2814732

Abdullah, Anwar, Akhir, Juhari Mat, & Abdullah, Ibrahim. (2010). Automatic mapping of lineaments using shaded relief images derived from digital elevation model (DEMs) in the Maran-Sungi Lembing area, Malaysia. The Electronic Journal of Geotechnical Engineering, 15, 949–957.

Jenny, Bernhard, Jenny, Helen, & Hurni, Lorenz. (2011). Terrain generalization with multi-scale pyramids constrained by curvature. (Report). Cartography and Geographic Information Science, 38(2), 110–116. https://doi.org/10.1559/15230406382110

shaded relief image| Definition - Esri Support GIS Dictionary. (2019). Retrieved 24 October 2019, from https://support.esri.com/en/other-resources/gis-dictionary/term/701e7093-08a6-4de4-a8e0-258117db9690

Graphics processing unit. (2020, June 11). Retrieved June 12, 2020, from https://en.wikipedia.org/wiki/Graphics_processing_unit

General-purpose computing on graphics processing units . (2020, June 11). Retrieved June 12, 2020, from https://en.wikipedia.org/wiki/General-purpose_computing_on_graphics_processing_units

# Appendix A - Project Work Breakdown

| Task | Person in charge | Date | | Duration | Contribution |
|---|---|---|---|---|---|
| | | Start | End | | Quality Assurance |
| Developing Phase (Implementations) | | | | | |
| Development Environment Setup | All | Week 0 | Week 0 | 1 week | All |
| Implement: (Main) Read-in write-out buffer | Jack | Week 1 | Week 4 | 1 month | Christine |
| Implement: GradientOperator | Jack | Week 6 | Week 8 | 3 weeks | Christine |
| Implement: HorizontalTransposingFilter | Christine | Week 6 | Week 8 | 3 weeks | Jack |
| Implement: LowPassOperator | Christine | Week 6 | Week 8 | 3 weeks | Jack |
| Implement: ClampToRangeOperator | ZiSong | Week 6 | Week 8 | 3 weeks | Jack |
| Implement: MaskFilter | Jack | Week 6 | Week 9 | 1 month | ZiSong |
| Integrate Eduard: MainLauncher | All | Week 3 | Week 11 | 2 months | Jack |
| Interim Presentation | All | Week 5 | Week 6 | 2 weeks | Christine |
| Testing Phase | | | | | |
| Benchmark the acceleration factors | Christine | Week 10 | Week 11 | 2 weeks | Jack |
| Quality Testing | Zisong | Week 11 | Week 11 | 1 week | Christine |
| Software Demonstration | Jack | Week 10 | Week 10 | 1 week | All |
| Finalizing Phase | | | | | |
| Presentation | All | Week 11 | Week 12 | 2 weeks | Christine |
| Final Report | Christine* | Week 11 | Week 12 | 2 weeks | Jack |
| Test Report *Testing | Jack* | Week 11 | Week 11 | 1 week | ZiSong |
| Code Report | Jack* | Week 11 | Week 11 | 1 week | Christine |
| Team Management | Christine | Week 12 | Week 12 | 1 week | ZiSong |
| * : Main representative | | | | | |

# Appendix B - Final Report Work Breakdown Structure

| Task | Person in charge | Date | | Duration | Contribution |
|---|---|---|---|---|---|
| | | Start | End | | Quality Assurance |
| Introduction | Jack | 10/6 | 10/6 | 1 day | Christine |
| Background - Literature Review | | | | | |
| Introduction | Christine | 2/6 | 2/6 | 1 day | Jack |
| Content | Christine | 3/6 | 6/6 | 4 days | Jack |
| Conclusion | Christine | 3/6 | 7/6 | 5 days | Jack |
| Methodology | | | | | |
| Implementation | ZiSong | 3/6 | 9/6 | 7 days | Jack |
| Design | ZiSong | 3/6 | 9/6 | 7 days | Jack |
| Method | ZiSong | 3/6 | 9/6 | 7 days | Jack |
| Project Management | | | | | |
| Project conceived | Christine | 31/5 | 3/6 | 4 days | Jack |
| Approach | Christine | 31/5 | 3/6 | 4 days | Jack |
| Resources | Christine | 31/5 | 3/6 | 4 days | Jack |
| Execution | Christine | 31/5 | 3/6 | 4 days | Jack |
| Planning | Christine | 31/5 | 3/6 | 4 days | Jack |
| Risk Management | Christine | 31/5 | 3/6 | 4 days | Jack |
| Limitation | Christine | 31/5 | 3/6 | 4 days | Jack |
| Outcomes | | | | | |
| Result and delivery | Jack | 31/5 | 1/6 | 2 days | Christine |
| Requirements | Jack | 1/6 | 1/6 | 1 day | Christine |
| Justification | Jack | 4/6 | 4/6 | 1 day | Christine |
| Results | Jack | 4/6 | 4/6 | 1 day | Christine |
| Limitation | Jack | 4/6 | 4/6 | 1 day | Christine |
| Future works | Jack | 5/6 | 5/6 | 1 day | Christine |
| Conclusion | | | | | |
| Cohesive | Jack | 10/6 | 10/6 | 1 day | Christine |

8650 words

# Appendix C - Risk Register

| Risk Register of GPU Acceleration for Raster Filter by using Aparapi | | | | | |
|---|---|---|---|---|---|
| No. | R1 | R2 | R3 | R4 | R5 |
| Rank | 1 | 2 | 3 | 4 | 5 |
| Risk | Changes of Tasks | Deficiency of time | Extension of scope | Few meeting times | Failure of speed up |
| Description | The inability to meet in person may change tasks to be completed. | The project might not be finished by the deadline. | The project is more qualitative in some parts beyond initially expected. | The time of meeting at campus would be short in holiday. | The Aparapi GPU approaches could not effectively accelerate the raster filter. |
| Category | Project Management - Tasks | Project Management - Estimates | Technique - Software | Project Management - Communication | Technique – Hardware, Software |
| Root Cause | Emergence of the COVID-19 pandemic. | Estimation of time by mistake because of insufficient knowledge | Planning of optimize the project | Team members leaving Australia in holiday | Laptop configuration problems or poor understand of Aparapi |
| Triggers | The need to do social distancing. | Lack of time to finish the project | Add or improve functionalities to project | Could not allocate time for meeting | Low power of laptop or bad use of Aparapi algorithm |
| Potential Responses | Mitigate | Mitigate | Acceptance | Mitigation | Avoidance |
| Risk Owner | Jack, Christine, Zisong | Jack and Christine | Zisong | Jack, Christine and Zisong | Jack, Christine and Zisong |
| Probability | 5 | 3 | 2 | 5 | 1 |
| Impact | 5 | 5 | 2 | 3 | 2 |
| Status | | | | | |

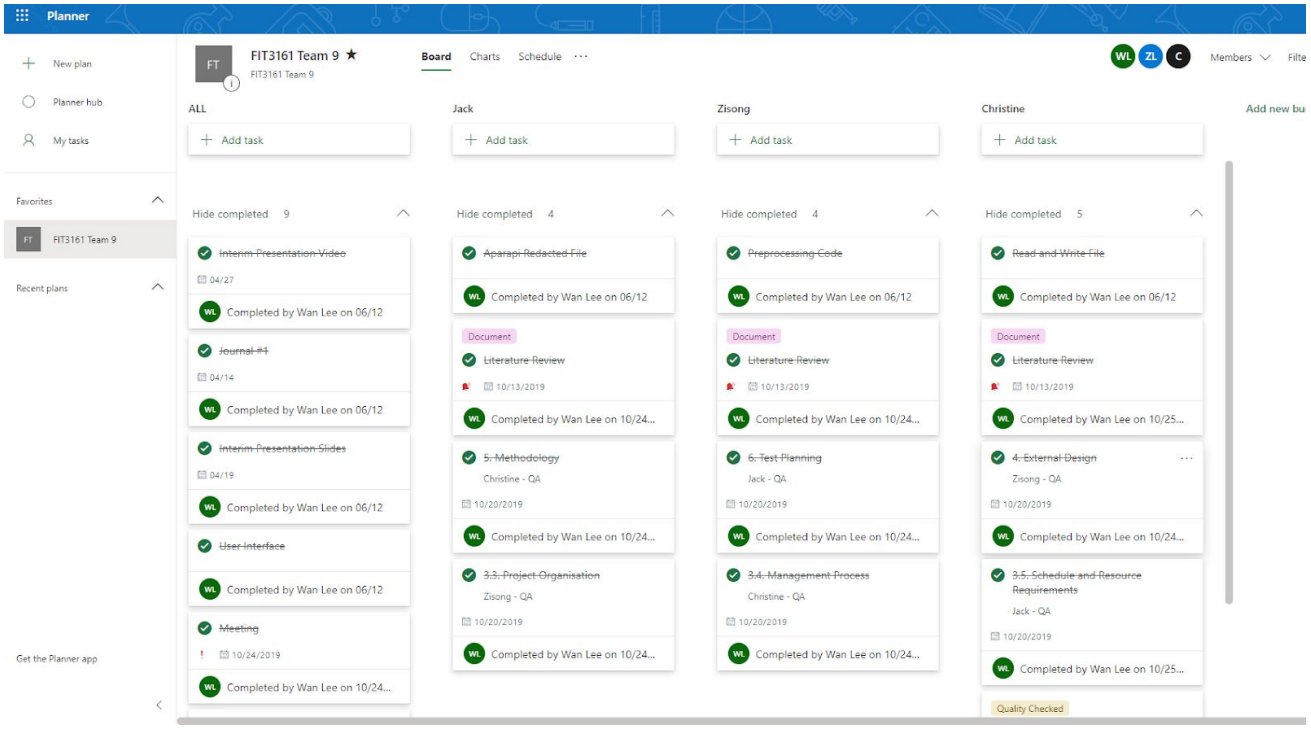
# Appendix D - Risk Matrix

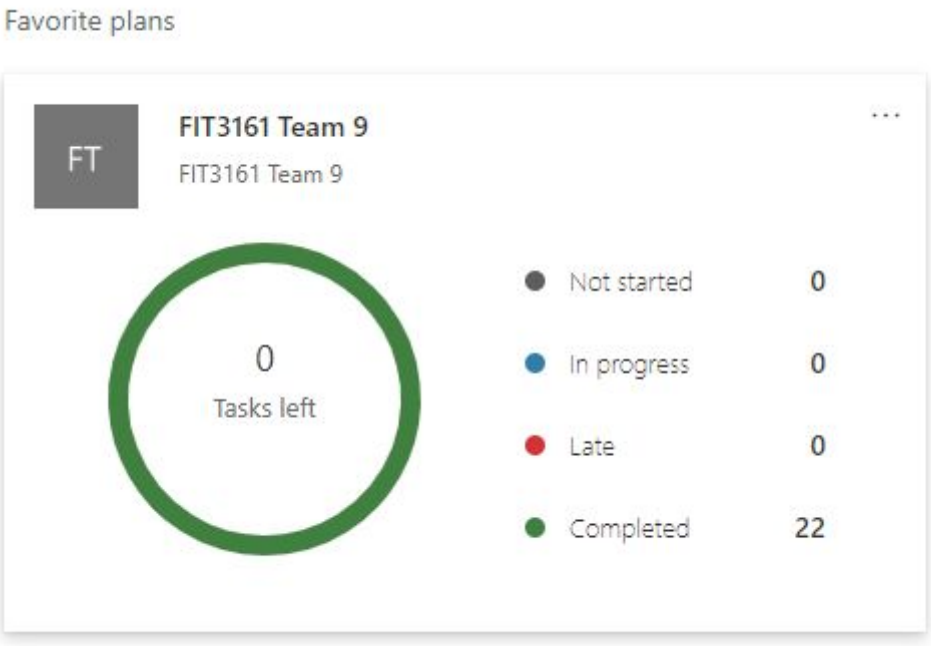
**Probability**

| | | | |
|---|---|---|---|
| High | | Risk 4 | Risk 1 |
| Medium | | Risk 3 | Risk 2 |
| Low | Risk 5 | | |
| | Low | Medium | High |

**Impact**

# Appendix E - Microsoft Planner



# Appendix F - Sequence Diagram