Parallelisation of an Android Application with Pyjama

Group 31: Yueyi Song, Stephanie Wiggins, Yiqun Gao

Department of Electrical and Computer Engineering
The University of Auckland
June 2, 2015

Abstract

This project is involves the parallelisation of an existing Open Source Android application written in Java, using Pyjama. The application that been chosen is an image processing software – "Effects Pro". It is an easy-to-use tool for applying filters to a selected image. The current limitation of this application is that it runs slowly. The user must be patient while an image is being processed. It is expected that applying parallel computation techniques to this project will accelerate the processing time. The report begins with the basic concepts of parallel computing, then proceeds to cover the implementation process. Finally it concludes with an objective evaluation of our results.

1.0 Background

Traditionally, a software program is executed on a single processor and would been written in serial sequence. At one time, there would only be one instruction being executed. This is known as serial computing. However, in real world, many problems are large and complex, becomes impractical to be solved within a single computer with limited memory and processing elements.

Parallel computing is about using multiple resources to solve computationally intensive problems simultaneously. Compared to sequential computing, using parallel computing is best suited more complex systems. It involves breaking a problem down into discrete parts, from now on referenced as tasks, which can be solved concurrently. Each task can then be executed simultaneously, on different processors. An overall control mechanism is also required in parallel computing systems, to schedule the necessary tasks onto a given processor. In theory, using more computation resources will shorten the completion time of the program, or in other words, to produce a speedup. We can use the speedup value as a criterion to indicate the performance of the paralleled program.

2.0 Open-Source Application Selection

For this project it was imperative to select an application with a significant amount of processing, which could be sun-divided and executed in parallel. The image processing applications seems suitable to fulfil this characteristic. The application we have chosen – "Effects Pro" is a practical open sourced application which applies a range of different filters to a chosen image. It provides the choice of approximately 20 kinds of filters. There is a considerable shortcoming of this application – That is that the processing speed is very limited, and the application appears very slow when applying a filter to the image. This project hopes to improve both the user experience and the computation time with the improvement of the parallel computation. This will be achieved by successfully accelerating the

processing speed of this application. Another initial expectation is: we also hope that the paralleled application could give users better user experiences by assigning the foreground (GUI) and background tasks (Image Processing) onto different processors. In the original version of "Effects Pro", would lose interactivity while the image was being processed. Thus our implementation strive to improve not only the speed at which the image is processed so that the user is not left waiting, but to also allow for the user input to remain responsive while such a task is being executed.

3.0 Methodology

3.1 Parallel Programming Architectures

Parallel programming has becomes a main trend in software industries. According to a frequently referenced taxonomy conducted by Michael J. Flynn, the parallel architectures have been classified into four categories: Single Instruction Single Data (SISD), Single Instruction Multiple Data (SIMD), Multiple Instruction Single Data (MISD) and Multiple Instruction Multiple Data (MIMD).

Another way to classify parallel architectures regarding the memory location, the memory is either centralized or distributed with the processors. These four exist parallel architectures have been implemented on two different memory structures - centralized memory and shared memory.

OpenMP is one of the most widely used APIs that supports multi-platform shared memory multiprocessing programming. It is a set of compilers that were designed to support parallel programming in C/C++/Fortran.

3.2 AsyncTask

The traditional method for the parallelisation of a Java android application is to use the AsyncTask class. This class allows the designer to separate the GUI and the main processing tasks into two different threads.

AsyncTask has three methods which need to be executed:

1) *doInBackground*(): Contains the computationally intensive algorithms
2) *onProgressUpdate*(): Displays a form of progress to the user via the UI thread, used to show the user that the app is busy processing, and has not gone unresponsive.
3) *onPostExecute*(): This method is used to synchronise the background thread with the UI thread

The *executeOnExecutor*() method can be used to parallelise AsyncTask.

This implementation is limited as it is recommended for use when the background processing tasks take no more than few seconds to execute.

3.3 Pyjama or Parallel Task

We have been given two options to implement the task of parallel an Android application written in Java: Pyjama and Para Task. Based on our initial research we determined that according to the developers documentation pyjama was specifically designed with the development of android applications taken into consideration, whereas Parallel Task was targeted for Desktop Applications. It was upon further exploration of the two that it was determined that both Parallel Task and Pyjama are both suitable, however with the overall structure of the application, pyjama offered the more appropriate implementation. The current implementation of the image processing is based upon simple for loops, thus the Pyjama *for* directive allows for a simple parallel implementation.

4.0 Implementation

It was identified early on in the development that the important factors to consider during the parallelisation of an android application, is the GUI responsiveness, as well as the overall speedup. It also became apparent from our initial research that until a finer grain implementation was utilised, the main improvements would be seen through the increase in user experience while using this application.

4.1 GUI Responsiveness:

Our desired implementation includes the separation of the user input from the image processing using Pyjama's *freeguithread* and *GUI* directives. However this resulted in too many issues due to the source to source compilation, which resulted in this section being excluded from the final implementation.

Thus our final implementation consists of the parallelisation of the image processing elements only.

4.2 Image Parallelisation:

The processing of the images in the original application was executed via simple for loops, therefore when it came to the pyjama implementation, the *for* directive could be implemented to create a simple form of parallelisation for the image. To extend on the simple implementation we also tested using both static and dynamic scheduling, our initial testing was set to the default to obtain an initial idea of whether there was any apparent speed-up from our implementation. Without any apparent speed-up it would be determined that the parallelisation of this application would not be appropriate.

Pyjama's for directive only converts the outer loop to a parallel region, thus for some of the effects currently included in the application parallelisation results in an increase in execution time. Below is an example of implementation of the parallelisation of one of the filters.

```
//#omp parallel shared(width,height,A,R,G,B,pixel,type)
{
        //#omp for
        for (int x = 0; x < width; ++x) {
                for (int y = 0; y < height; ++y) {
                        //BOOST FILTER MANIPULATION
                }
        }
}
```

4.3 Mobile Limitations:

The implementation of a parallel application using pyjama, becomes limited by the type of mobile device on which the application is deployed. Mobile devices have developed significantly over time, however due to their size, the processing power is still limited. High end mobile devices have quad core processors, which is one of the main reasons why the implementation of parallel application is plausible, however they still have very limited processing power. This significantly limits the performance of any application, particularly those with heavily computational sections, thus even with our parallel version of the application, elements such as the image size is limited.

5.0 Results and Discussion

5.1 Speed-Up on Android Virtual Machine

Our running environment is API level 19, RAM 512 and the processor is Intel Core i5-3210M. We use boost filter as an example to demonstrate the contrast between the original program and parallelized program. The data obtained in figure 1 is from the average of several records (more than 30). We can see that basically bitmap processing time increases as the size of image increases and the program version parallelised by Pyjama has a noticeable speed-up.

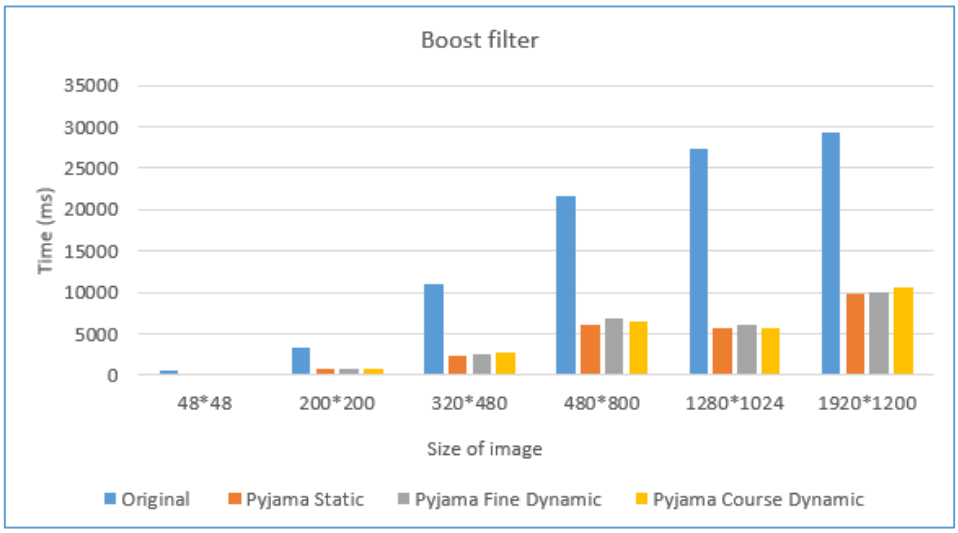

Figure 1: Pyjama Implementations and Original Implementation Comparison

Figure 2 shows that the range of speed-up is from 3 to 5. However, execution in parallel cannot always achieve a speed-up. For instance, if the Gaussian Blur filter is run in parallel, the processing time would be longer than when run in sequential. This is because the degree of parallelism increases as the number of threads grows, thus the parallel overhead will also increase at the same time. For a situation where, you have a large number of small tasks being executed in parallel, there becomes a significant overlap in data communication and as a result the waiting time for data access will be increased.

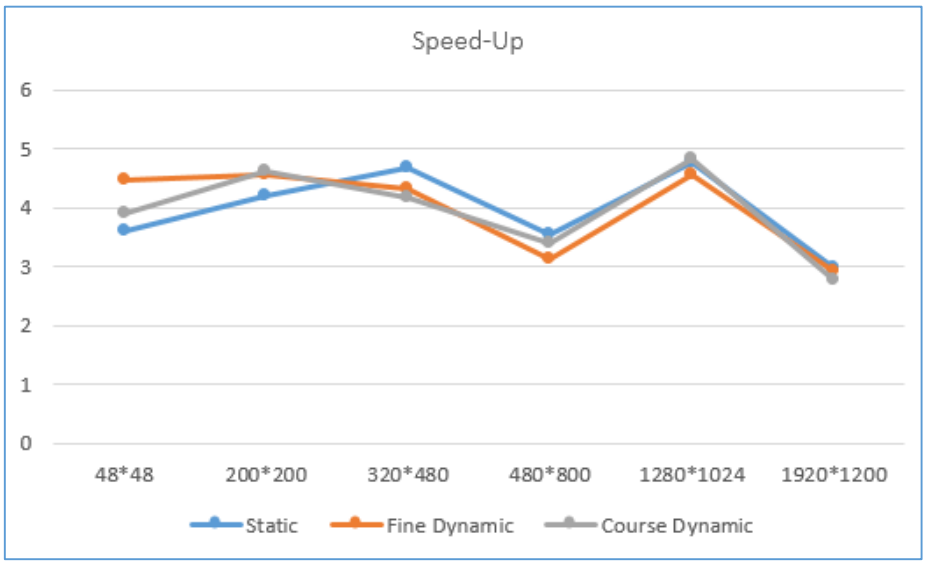

Figure 2: Speed Up of the Boost filter for Various Image Sizes

5.2 Speed-Up on Samsung S7562

These graphs illustrate the difference in execution times, and the achieved speed-up when the various parallel versions, as well as the original application are executed on a physical android device.

It becomes apparent from the graphs below, that testing on the virtual machine yields significantly different results to those executed on an actual device. Although the number of processes running on physical device can be limited to an extent, a point where no other processes are contending for processor access is not plausible. The above graph illustrates that three Pyjama implementation offer a very similar execution time, all of which are lower than the original implementation, however the difference between them is a lot smaller.

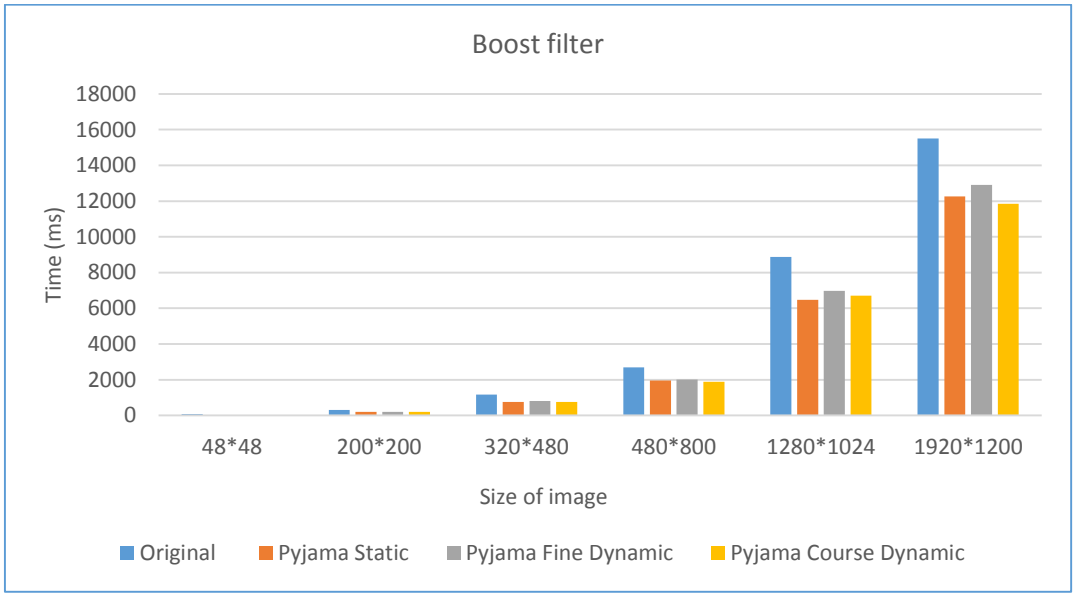

Figure 3: Boost Filter execution time comparison on Samsung S7562

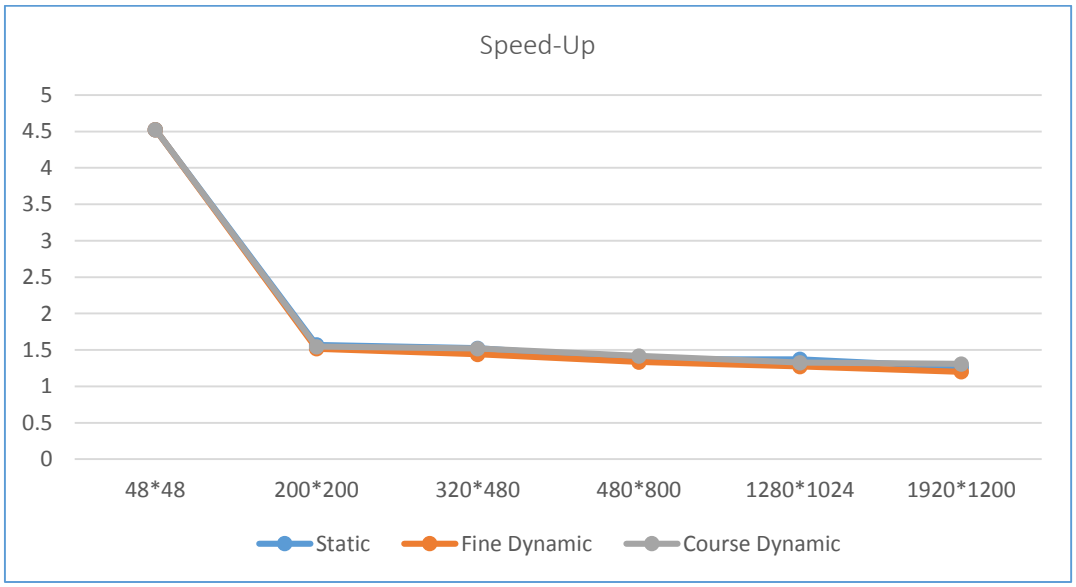

Figure 4: Speedup comparison, for execution on a Samsung S7562

6.0 Limitations, Future Improvements, and Conclusions

6.1 Pyjama Limitations

In terms of this particular project the most influential limitation of Pyjama, is the missing support for nested for-loops. As the image processing sections of the program range from a double for-loop to a quadruple nested for loop the parallelisation of the application becomes very limited. This is because only the outer for loop will be generated as a parallel region. Limiting the extent of the granularity of the parallelisation.

Secondly it has been identified that the conversion of a for-each loop is not supported in Pyjama, which caused some source to source conversion errors, which had to be modified for directives such as the freeguithread or gui. Further errors occurred with the implementation of the freeguithread directive, such that previously defined variables become out of scope when the .pj file is converted to the .java file. This error is due to the scope in which the variable is defined changing when the parallel content is included in the .java file.

6.2 Existing Problems and Future Work

Although the program have a significant speed-up performance after introduced Pyjama, there are still some unsolved problems in it. In the following part of this report, we will focus on analysing the existing problems and give out suggestions for future work.

Existing Problems:

1) If you choose “run as android application” the program can run properly while if you choose “debug as android application” the program will hang.
2) When the program runs in Samsung Note3, the program will stop unexpectedly.
3) Limitations on image size due to the current parallelisation technique, as well as the limitation of mobile devices.

Future Work:

Future improvements, included testing the implementation as a Parallel Task solution to see whether or not this will offer any benefits for the user in terms of speed-up and general responsiveness. However our initial improvements would include the correctly separating the GUI and the processing tasks to maintain the overall responsiveness for the Pyjama implementation, and to debug further whether or not the similarities between the static and dynamic scheduling policies is expected.

It would also be useful to limit the number of filters available to those which the current mobile devices hardware can handle. Some of the effects that require a lot more processing or are unsuitable for parallelisation with the current Pyjama release, should be removed to ensure maximum user-experience.

Using the source code from other people means that we cannot guarantee the quality of the code, or that it will be structured in a manner that will allow for effective parallelisation. Thus in future, it would be better to take the concepts implemented in the original application and apply them in a new application, where the code is structured in a manner that is best for parallelisation via Pyjama directives.

6.3 Conclusions

The results of this project illustrate the difference in execution times for the various pyjama implementations, as well as the AsyncTask implementation (Original).

The current hardware limitations of mobile devices, means that they are incapable of handling intense image processing. Thus the extent of speedup achieved in this project, is expected to be limited compared to that of a desktop implementation.

Based on our results obtained from the execution of various parallel versions, as well as the original application, it becomes apparent that our implementation does provided a speed-up, although limited on the physical android device. Due to the fact that the virtual machine is simulated on a much faster, and robust processor than the on present in the mobile device, it was expected to see, an 'ideal' result coming from the execution measurements compared to those from the execution on the mobile device itself. Overall the use of Pyjama directives to parallelise this android application, has resulted in some form of speedup when applying certain effects to an image.

With the growing capabilities of mobile devices and their need to be able to perform large computationally intensive tasks, the effective parallelisation of the application become a crucial part in the applications development. Throughout the course of this project it became apparent that there are two crucial criteria to meet, in order to ensure the effective parallelisation of the android application. The first of which is to ensure the responsiveness of the application, and secondly to maximise the speed-up via further parallelisation of the larger computational tasks. In terms of the application chosen to demonstrate these aspects. It can be seen that working with pre-existing software that was not structured with parallelisation in mind, proves difficult to incorporate parallel techniques. Also that image processing, although appropriate for smaller images on mobile devices, is more suitable for a desktop application at this point in time.

Furthermore, our efforts throughout this program have highlighted some errors/ bugs present in the Pyjama software which would need to be resolved to ensure that it becomes a desired tool for developers when they are considering the parallelisation of their applications.

7.0 Table of Contributions

Group Member	Research	Application Selection	Parallelisation Implementation	Debugging	Presentation	Report
Yueyi Song	30%	35%	40%	35%	35%	30%
Stephanie Wiggins	35%	35%	25%	30%	35%	40%
Yiqun Gao	35%	30%	35%	35%	30%	30%

Figure 5: Work Distribution: Reasonably even amount of work completed but each group member

8.0 References

[1.] Keeping Your App Responsive. *http://developer.android.com/training/articles/perf-anr.html*

[2.] Android background processing with Handlers, AsyncTask and Loaders. *http://www.vogella.com/tutorials/AndroidBackgroundProcessing/article.html*

[3.] AsyncTask. *http://developer.android.com/reference/android/os/AsyncTask.html*

[4.] O. Sinnen, N. Giacaman, Vikas. *Pyjama: OpenMP-like implementation for Java, with GUI extensions*. 2013

[5.] Pyjama – About. *http://homepages.engineering.auckland.ac.nz/~parallel/ParallelIT/PJ_About.html*

[6.] Parallel Task – About. *http://homepages.engineering.auckland.ac.nz/~parallel/ParallelIT/PT_About.html*