Utilising the Graphics Processing Unit of an Android Powered Device

William Scott-Jackson, 082588603

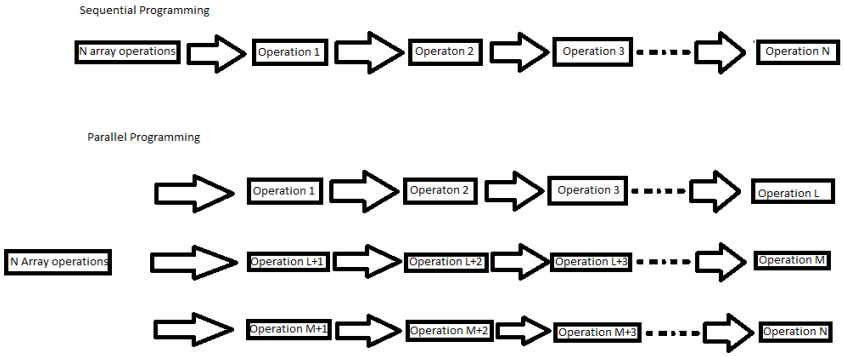
Table of Contents

Over the past few years, many Android powered smartphones have penetrated the telecommunications market in direct competition with Apple, Windows phones, and other mobile phone developers. But recently companies have been trying to provide the processing power comparable to that of a computer, but with a reduced size and a capacitive touch screen interface similar to that of today's mobile smart phones. This brought about the birth of the tablet PC, as with all hardware they have limits and application developers strive to get the most out of their applications by optimising them with various programming techniques. These techniques often involve use of the built in graphics chip to offload a heavy, repetitive mathematical operations. In the world of PC programming, there exists Microsoft Accelerator which allows a programmer to quickly optimise an application with no exposure to a new Application Programming Interface (API). The aim of this project was to build a working API which would allow a user to equally easily optimise an Android application without exposing them to other more complicated APIs.

Introduction

Computers and Central Processing Units

Every computer contains a Central Processing Unit (or CPU), which is an electronic component containing millions upon millions of nanometre sized transistors which are capable of switching on and off incredibly quickly. As a result a CPU can perform millions upon millions of mathematical operations in a second, these days; the types of operations performed include data processing, signal processing, and simulations. When developers write programs for CPUs, they adopt what is known as the sequential paradigm, the instructions in a program are executed in the sequence they are written in. Since the 1960's the amount of transistors that can fit into a given space has doubled every two years as given by Moore's Law[1]. This has given rise to an exponential rise in processing power increase, but in recent years, the rate of this increase has been slowing, due to us reaching the limits of Moore's Law. This has caused manufactures to abandon just going for faster processors and opting in for more processors which in turn has demanded that software developers adapt their techniques appropriately. Such techniques include multithreading, which is where a process or processes can be broken down into threads and run in parallel. A thread is a concurrent unit of execution that can run in parallel with other threads. This is useful for processing large arrays of numbers, or even applying filters to images etc. This brings rise to a practice known as data parallel programming; this is simply where the same operation is performed on several different threads or processors, all working towards processing the same set of data.
Below is a visual representation of parallel programming, the example is the processing of N array elements, in the sequential paradigm they are processed one after the other until the it reaches the end. But with the parallel paradigm, sections of the array are divided up into separate threads and processed separately, assume for this example that both paradigms process elements at the same speed (although this is often not true, as some processors trade off clock speed for more threads), then it's safe to conclude that the parallel paradigm is faster:



[1] http://www.intel.com/content/www/us/en/silicon-innovations/moores-law-technology.html

In this diagram, the operation is divided up into thirds; from top to bottom are the first third, second third and final third where it processes the Nth element. Of course in practice the amount of threads created will depend on requirement of the operation and the processor available.
Most computers these days come with either a dual or quad core processor as standard, but these can only multithread to an extent; however there is another computing component that can further exploit this parallel programming paradigm.

Computers and Graphics Processing Units

Since 1999, another major component of all modern PCs, the graphics processing unit (GPU) has been evolving at an alarming rate. A GPU is used for performing operations that eventually lead to an object or image being displayed on a screen of sorts. These are particularly useful for programs and systems that require some form of graphical interface, such as Computer Aided Design programs, simulations and video games. A GPU trades off clock speed for an increased amount of cores, a typical consumer GPU such as the NVidia GTX 460 contains nearly four hundred cores. This is invaluable for the various graphical demands that modern software demands from computer systems.
This has led to the practice of General Purpose Graphics Processing Unit programming (or GPGPU programming), which is harnessing the hugely parallel architecture for programming normally done on a CPU.
Nvidia's solution to this was to develop their CUDA architecture[2], which allows a user to program in a language very similar to C and these programs can harness the power of the GPU. Microsoft also has a solution in place; Microsoft Accelerator[3], which is a library of functions for programmers using C# or C++. The reason Accelerator may be preferable to using CUDA or a similar implementation, is that Accelerator is designed for someone wanting to optimise an already existing code for a machine that has a GPU or a multicore CPU that may not have been available when the program was first written.

Tablet Computing and the Android Operating System

In the last few years, smartphones have penetrated the portable telecommunications market, especially with the release of Apple's iPhone. Following hot on the heels of Apple was Google with their Android Operating system. Even more recently was the development of the tablet computer, again Apple leading the competition with the iPad, with Android tablet developers following behind. Because tablet computers are more restrictive in terms of size in order to maintain portability, this has considerable restrictions on potential processing power (a more powerful processor would require a cooling system to stop it overheating, a system not feasible on something portable). Most tablets are equipped with either a dual or quad core processor, but these won't have a clock speed anywhere near that of a modern day laptop or desktop PC. Similarly, they will also have a graphics chip that won't be able to compare to that of a PC graphics chip. The most common graphics chip found in an Android device is the Nvidia Tegra chip, as of writing this report, the current standard model that ships with all tablets is the Tegra 3[4], but months prior to this the Tegra 2[5] was the standard, and also the graphics chip that this project was designed, tested and implemented on. The Nvidia Tegra 2 contains eight cores and is clocked at 667MHz; this is enough to warrant using for general purposes to supplement the limited processing power tablet computers currently possess; but with the promise of further improvements to tablet computers on the horizon, it makes even more sense to invest in this practice. At present, there are a limited number of freely available tools that can allow a programmer to do some GPGPU programming on an Android powered device.

---

[2] CUDA: By Example kindle edition, Addison Wesley, Jason Sanders and Edward Kandrot, location 351 of 4339
[3] http://research.microsoft.com/en-us/projects/accelerator/
[4] http://www.nvidia.com/object/tegra-3-processor.html
[5] http://www.nvidia.co.uk/object/tegra-2.html

The advantage with the Android operating system is that a most of the application development is all open source and that anyone with a suitable machine can write programs and applications without having to pay for a developers license, unlike those that choose to program for Apple devices such as the iPhone and iPad. This implies more people may be inclined to take up Android programming as hobbyists, these sorts of people may not have much training in programming and won't be aware of the techniques available to utilise every last clock cycle of processing power.

Project Aim

The aim of this project was to create an Application Programming Interface (API) that will allow a novice programmer to do some GPGPU programming, but very much in a similar way to Microsoft Accelerator, where a program can be easily optimised by the addition of a few extra commands from the user. The project was built and tested on an Asus Eee Pad Transformer TF101 running on Android 4.03 Operating system and fitted with an Nvidia Tegra 2 graphics chip.

Research Procedure

Before any code was written, it was important to find some reading material to do extensive research and hence find some direction on how the project should proceed. The following areas were researched:

- Microsoft Accelerator, how it works and how to use it.
- Programming in Java, and hence programming for Android devices
- OpenGL and C++ development for Android, this was one of the routes in which this project could have taken.
- RenderScript development on Android, this was another of the routes this project may have taken.

Literature Review

Microsoft Accelerator

The first step of this project was to research into Microsoft Accelerator, learning how it works and how it can be used to optimise a program for a multicore CPU system, or a system fitted with a GPU. The Microsoft Research website contains a section on Microsoft Accelerator, within this site are some documents detailing how Accelerator came about, how it works and how to use it.

Accelerator Introduction Document[6]

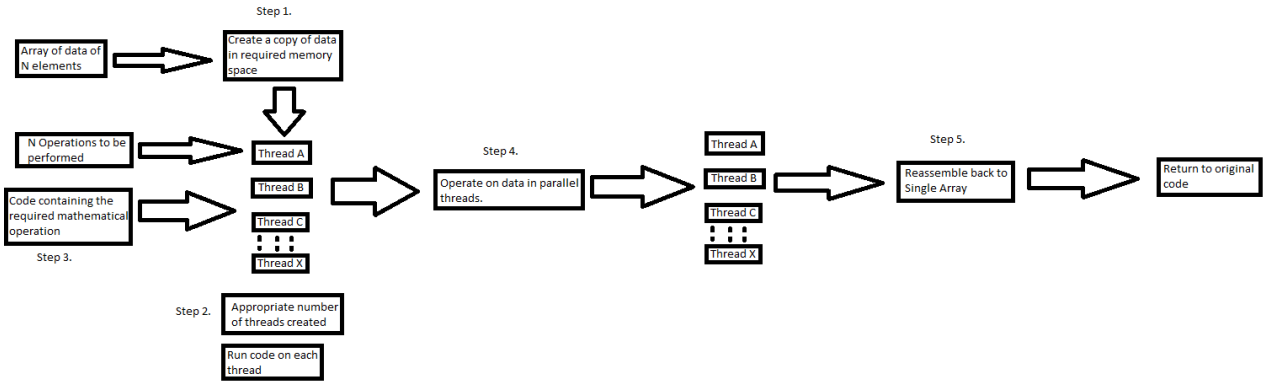
This document contains a brief introduction into Microsoft Accelerator, but more importantly a simple development guide for those wishing to use it.
On page 15, there is a detailed description of the process which Accelerator goes through to perform a data processing operation using a GPU. Any data that needs to be processed will be converted into a texture; a texture is an image that can be mapped to a surface of an object in a computer program. The elements of the data are mapped to individual pixels, and like an image, the pixels are assigned a colour based on the value. This makes it possible for the GPU to interpret the incoming data. The actual operations that need to be performed (i.e. arithmetic operations such as addition) are converted into programs that can run in GPUs such as a pixel a shader. A pixel shader is a program that operates on an individual pixel to compute its value. Once this operation has been done for every element of the array (which corresponds to every pixel in the texture); the data is converted from a texture back to an array and transfers it back to the computers system memory.
Microsoft Accelerator uses the DirectX graphics API to do the necessary processing on the GPU; DirectX offers various graphics and multimedia functionality on any Microsoft computer fitted with a suitable GPU. At the time of writing this report (April 2012) the standard version of DirectX is DirectX 9, but a lot of newer programs are retailing with some DirectX 11 functionality.

[6] http://research.microsoft.com/en-us/projects/accelerator/accelerator_intro.docx

On page 16, of this document it describes the process which the Accelerator goes through to run an array operation on a multicore CPU (also applicable to GPUs).

1. The first step is to create a copy of the data to work with.
2. A set of threads are created so that the operation can be divided between these threads.
3. A kernel is spawned; a kernel is a piece of code that is designed to run on hardware.
4. This kernel is run on each thread.
5. The output data is reassembled and returned to where it was called from.

These five steps are demonstrated in the diagram below:



Using the basic development guide included within this document, it was possible to develop a basic benchmark program and test it on a couple of systems that can use Microsoft Accelerator, the source code is included in the attached CD, but a segment of this code is included in Appendix 1 and a brief discussion on this code will be included later on in this report.

Java and Android Application Development

Of course all of this research into creating a data parallelism library wouldn't be much use if some time didn't go into learning how to implement such a library onto an Android system. Basic Android applications are developed using the Java programming language; this language was developed at Sun Microsystems by one James Gosling[7], and has become incredibly popular since being used on a vast array of devices. Java is a commonly used programming language that boasts the ability for the user to make a user interface with relative ease when compared to languages such as C/C++.

The resource used to learn how to program in Java was Sam's Teach Yourself Java in 24 Hours by Rogers Cadenhead; this book contained all of the information required to create programs in Java, as well as some of the concepts that this language and others utilise, mainly, the use of classes. A class is a separate source file that contains various functions that the programmer writes, which can be reused over and over again between different programs with a minimal amount of extra coding, a concept heavily adopted in this project.

But this on its own is still not quite enough, because Android devices are built with a capacitive touchscreen interface and have completely different architectures to that of a PC, it was necessary to learn to program for Android systems too. The first source for this was the Android 3.0 Application Development Cookbook by Kyle Merrifield Mew. This ran through all of the Android specific aspects of programming such as creating a user layout, creating functions that respond to touch and accelerometer inputs amongst other things. A basic guide on how to create a basic Android application is included in Appendix 2.

C++ and OpenGL in Android

Since Android systems have started penetrating the market, the pressure is on developers to be creating much more high performance software to meet consumer demand. But high performance isn't as easy to achieve with Java as it is with other programming languages. One of the trade-offs made with the Java programming language was the fact that Java has what is known as a garbage

[7] Sam's Teach Yourself Java in 24 Hours Kindle Edition, Sam's, Rogers Cadenhead, location 308 of 9011

collector[8] which allocates and frees memory when it is no longer required, but this requires extra processing power deciding what memory to free and what memory to leave compared to that of a program where the programmer has done manual memory management. But other programming languages such as C++ offer this level of performance.
One of the more recent developments in the Android development tools available is the Native Development Kit (NDK) which allows a programmer to insert C++ functionality into their applications. Whilst this can allow a significant performance boost, on the flip side development complexity is increased greatly and in order for someone to develop an application using the NDK they will need a large amount of experience in two programming languages (C++ and Java).
Android uses the graphics API OpenGL, which gives programmers access to various functions that allows them to create complex shapes, apply textures and lighting effects to these objects. It is possible to exploit these functions in a similar way that Microsoft Accelerator does, but that uses the DirectX API. For example if a programmer wanted to add two arrays together, an Accelerator like library would convert these two arrays into textures, where the individual elements of the arrays are represented as pixels. Equivalent pixel shader programs would operate on these sets of pixels on the GPU and generate an output texture. This output texture would then be converted into an output array and sent back to where the programmer had originally called the function from.
The main drawback of attempting to implement this project using this method is the sheer complexity and scale of the project given the time frame. Going down this avenue would require a lot more time and effort, probably more suitable to a team of programmers experienced with graphics programming.
A brief guide on how to develop Android applications using the NDK is given in Appendix 3.

RenderScript in Android

Over the last year, Android tablet computers have entered the market and have come with their own operating system (OS) Android 3.0 or Honeycomb (Android operating systems are often named after some form of confectionary). With this new operating system came some new developer features, namely the RenderScript API. RenderScript[9] is both a graphics and compute API which allows users to inject some high performance into their applications but without the steep learning curve of using the NDK. Developers will have to be familiar with the C programming language (C99 standard), so to an extent there is still some added complexity, but nowhere the added complexity of the NDK.
Most of the information on RenderScript can be found on the Android developer's website, but a small amount information was found in the text Pro Android Apps Performance Optimisation by Hervé Gulhot[10].
The massive advantage offered by RenderScript is the performance boost; it can run on various types of device, for example the CPU, GPU or even a digital signal processing (DSP) chip if one is available without having to target each device. This already trumps Microsoft Accelerator in that respect as with Accelerator the programmer has to choose a device which they want their code to run on. This because the RenderScript code is always compiled and cached at run time, so it can decide where to offload the calculations then.
The main problem though is that RenderScript is barely out of its infancy, so some features are still missing, namely the features that allow the RenderScript code to be run on a GPU or DSP, (correct at the time of writing of this report) so anything produced in RenderScript will only run on the CPU[11]; this is even true for Android devices running the latest version of their Ice Cream Sandwich (4.0 and above) operating system. But an advantage here is that the nature of RenderScript, specifically the

[8] Pro Android Apps Performance Optimisation Kindle Edition, APress, Hervé Guihot, location 2614 of 5885
[9] http://developer.android.com/guide/topics/renderscript/index.html
[10] [10] Pro Android Apps Performance Optimisation Kindle Edition, APress, Hervé Guihot
[11] http://blogs.arm.com/multimedia/617-gpu-computing-in-android-with-arm-mali-t604-renderscript-compute-you-can/

fact it will run on various architectures without having to target them specifically is that a developer using RenderScript can future proof their code to an extent (i.e. won't have to write their code to target specific hardware when the features become available).
A guide on how to include RenderScript in Android applications is given in Appendix 4.

At this point in the project, a decision had to be made, go down the more complicated route using the Android NDK and OpenGL, or tackle the more feasible RenderScript route. Given the time frame, the skills and resources available, it made more sense to tackle this project using RenderScript instead of OpenGL.

Procedure

Once a decision was made, a clear plan was required in order to give some direction to the progression of the project. These are the tasks that were tackled in order to reach the main project aim:

- Create an Android application which can be used as a test bed for any code produced.
- Write a basic Android benchmark test which can be used to test the performance of any Android code running using Java. This benchmark test will comprise of a simple program operating on a given number of random array elements and time how long it takes to perform. The development process of this can be found in Appendices 5, 6, 7 and 8.
- Write some RenderScript code that can perform some basic mathematical operations on the elements of an array. The development process can be found in Appendix 9.
- Make any improvements on the paradigm from the code written in the previous step if possible.
- Once the above code has been optimised, implement it for different operations and different data types.
- Find a way to test the performance of the RenderScript code and compare it to the Android Java code.

The basic requirements of the RenderScript Accelerator API were:

1. A function should be invoked with a few simple lines of code, keeping user exposure to a minimum. Once the function has been called, it should take care of everything without the user knowing (i.e. memory management, array reassembly etc.).
2. Be able to create a copy of the data arrays and send them to the appropriate memory space, in this case the GPU memory.
3. Call a RenderScript operation to process the elements found in the GPU memory space. The appropriate number of threads needs to be created depending on the operation and what the Android system decides is the appropriate processor to run on.
4. Reassemble the arrays and send them back to the user as a simple output array that they can process further.

Results

Microsoft Accelerator

Before any other work was done towards the main aim of the project, it was important to write a simple program using Microsoft Accelerator to get acquainted with it, how a user can optimise their code with the use of a few simple commands. The code produced was a basic benchmark test (performance test) as shown in Appendix 1; there are two versions of the code, one that doesn't use Microsoft Accelerator and one that does. They are both benchmark programs to test the effectiveness of Microsoft Accelerator in terms of performance and ease of use.

Java Benchmarking

The best way to test the performance of the RenderScript built Accelerator API, was to build a performance test for Java to compare the performance. The development process and prototypes are documented in Appendices 5, 6 and 7 with an extract of the final benchmark program in Appendix 8.

The benchmark program gets the current time from the Android system clock and stores it in a set of variables, with resolution up to microseconds. A mathematical operation is called; say for example repeated integer addition, floating point subtraction etc. At the end of each of these operations, the time is again retrieved from the Android system clock and stored into another set of variables. The difference between the start time and end time would be resultant time taken, this is then displayed on the screen of the tablet.

Building the RenderScript API

Now that a basic Java program had been written to test the performance of that programming language was prepared, it was time to move onto the main aim of the project, building an API that will accelerate the same operation but made in such a way that it is easy for anyone to use.

A basic development guide on how to include RenderScript in Android applications is given in Appendix 4 and covers some of the details of how to interact RenderScript code with Java code to save the repeating explanations of certain lines of code i.e. creating the RenderScript context and creating object from generated classes.

The first prototype code that was developed is shown in Appendix 9, this was a very inefficient code, but further development led to the code in Appendix 10.

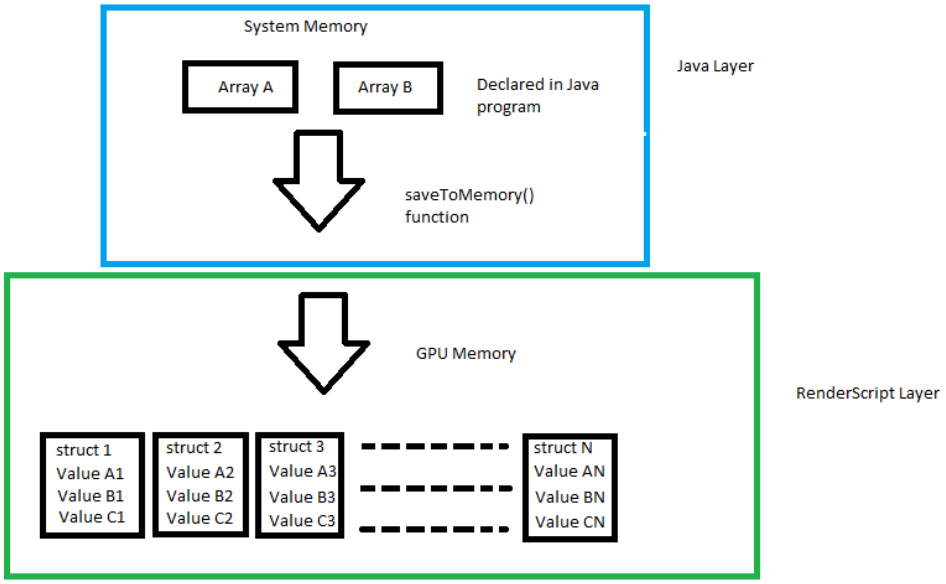
Appendix 10 contains the source code for the second RenderScript prototype which also leads to the development of the final benchmark program. The first important component of this second prototype is the saveToMemory() function which dynamically allocates memory on the GPU depending on the size of the array that the user wants to process. The saveToMemory() requires the input arguments of the two arrays to store, the RenderScript context and the object from the generated class. The first line of code within this function is particularly important:

```
ScriptField_Values Arrays = new ScriptField_Values(mRs, A.length,
Allocation.USAGE_GRAPHICS_VERTEX);
```

This creates a new object called Arrays, from a class called ScriptField_Values, this is a source file that is generated in response to some RenderScript code being written. To interact with memory, the use of structures[12] is recommended, a structure can store information of different data types, unlike arrays which can only store information of one data type. This is recommended because RenderScript currently doesn't support arrays or array manipulation, so structures are the most suitable alternative. In the case of this program, the structure will contain the values of the two numbers to add A, B and the result of the operation C.

So the above line of code creates a new object and allocates memory for the structure(s), the first argument is the RenderScript context mRs, the second argument is the number of structures that need to be created so that enough memory can be allocated. This number is equal to the length of the array. And the final argument is where the memory should be allocated; in this case the memory is being allocated in the GPU memory, more specifically where the data on vertices is stored. A visual representation of this memory allocation process is shown below:

[12] http://developer.android.com/guide/topics/renderscript/index.html#struct



As demonstrated here, the saveToMemory() function creates as many structures as there are array elements to process and puts a pair of values in each one within the GPU memory space. There is also a variable C to represent the result (calculated later).

The next line of code binds the create object to the structure pointer "values" defined in the RenderScript code.

```
mScript.bind_values(Arrays);
```

A pointer refers to a particular position in memory as shown below for this particular implementation:

```
typedef struct __attribute__((packed, aligned(4))) Values {
      int A;
      int B;
      int C;
} Values_t;
Values_t *values;
```

As can be seen, this code creates structure to store the variables in, the a pointer *values is created so that is possible to refer to this memory space elsewhere. The words packed and aligned are to increase memory performance, packed does exactly that, packs the data and decreases the system memory footprint. Aligned places the members of the structure at an offset position in memory defined by the input argument (e.g. 4 bytes), this improves memory performance[13].

This means that any changes made to the values within the structures can be accessible to both the RenderScript layer and the Java layer as the code is interacting directly with memory.

The next line creates a new object that is of type Item, this is RenderScript specific and allows a programmer to interact with the RenderScript structure.

```
Item i = new ScriptField_Values.Item();
```

This is followed by a for loop:

```
for (int j = 0; j < A.length; j++) {
                    i.A = A[j];
                    i.B = B[j];
                    Arrays.set(i, j, true);
      }
```

This for loop will run for an amount of iterations that equate to the length of the arrays. It will access each structure with the index j, and set the variables A and B within each of these structures to the elements in arrays also accessed with the index j. After that has done, the third line tells the program to synchronise these values to memory, by again pointing to the structure using the Item object i, it points to the particular structure with the index j, and the Boolean true tells the program to copy the

[13] C in a Nutshell, O' Reilly, Peter Prinz & Tony Crawford, Page 145

values to memory at that specific time. Alternatively the programmer could input false, but call a function that will synchronise everything later on.
This completes step two as detailed in the procedure.
Now taking a look at the RenderScript function intAdd():

```
void intAdd() {
	Values_t *pValues = values;
	int index = rsAllocationGetDimX(rsGetAllocation(values));
	for (int i = 0; i < index; i++) {
		pValues->C = pValues->A + pValues->B;
		rsSendToClientBlocking(pValues->C, (const void *) &pValues->C, i);
		pValues++;
	}
}
```

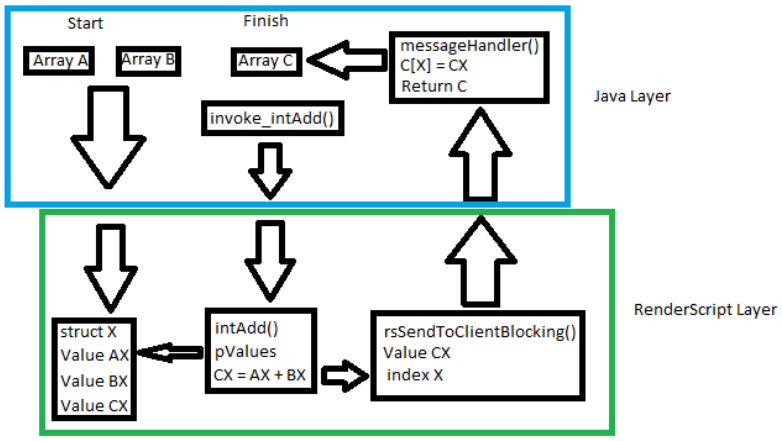
When this is invoked, the first line of code creates a pointer to the structure, so that the values (members of the structure to use proper terminology) can be accessed and operated on.
The second line of code calculates the size of the index, hence how many times to repeat the operation by finding the size of the x dimension of the memory allocated to the multiple structures. A for loop runs which iterates for the amount of times that equate to the size of the index i.e. how many individual structures there are, which equates to the length of the input arrays. The first line within this for loop accesses the individual members of the structure using the syntax:

```
pValues->C = pValues->A + pValues->B;
```

This looks at the structure referred to the pointer pValues, retrieves the elements A and B, then adds to them together and stores them in C.
The following line is a RenderScript function that allows a programmer to send data back to the Java layer with the help of a message handler (that will be shown next), the word Blocking, means that the program will block until the message has been received on the Java layer before moving on. The final line increments the pointer, and hence moves onto the next structure to perform the operation again.

Because RenderScript calls are asynchronous, there needs to be away to retrieve the data from the operations, as mentioned above the rsSendToClientBlocking() function is used on the RenderScript layer. The input arguments are an integer value, a const void value and another integer value. The first integer is the result of the operation, this is the data required to send back to the user, the second input argument for this case is considered useless but something needed to be inputted for the program to actually work. The final input argument is the index of the current operation which is important in the array reconstruction process in the message handler.
The Java layer needs to have a message handler to retrieve the message passed from RenderScript otherwise the program will report an error and close down. This is where the messageHandler() function comes in also given in Appendix 10. The first line of this function creates a new array that is the size of the input arrays; this will gradually fill with elements returned for the RenderScript program. The next block will only run in response to a message being returned from RenderScript, the next line is a logging function that retrieves the values and stores them in variables. The final line in that block puts the values into the array D which is indexed by the variable mLength. Once the array has been filled up it returns the array from where it was called, and from there the array is returned to the Java code the user has called it from. A visual representation of this process is shown below:

This demonstrates the flow of the RenderScript built Accelerator function for the integer addition implementation. The arrays A and B are stored into a set of structures, each of which containing a pair of values. The inAdd() function is invoked, the result of which is sent back one value and index at the time. The result is reassembled using the messageHandler() function and returned as array C at the end.



There is one major flaw though in this, again arising from the asynchronous nature of RenderScript. Once the RenderScript function is invoked, the Java code continues to run regardless of whether all of the data has been returned, this can result in either a zero array being returned or an array with some values missing. A very basic and prohibiting solution has been put in place, a delay has been added so that the RenderScript has enough time to run and save the values to the output array. The only delay that would guarantee successful data transfer for any number of arrays is two seconds.
Another flaw is that this API will not run and return values of arrays over one thousand elements, so another class was written called **IntSplitArrays.java** which will divide an array over one thousand elements into many arrays each of one thousand elements, run them one at a time and reconstruct the array appropriately, then send it back. An extract of **IntSplitArrays** is given in Appendix 11.
Now with these two flaws coupled together severely limit the performance of this prototype. If there is an array that needs processing containing over one thousand elements, then there will be a delay which equates to two seconds multiplied by the number of thousands of array elements. This is a fatal rate determining step.
Another minor flaw is that because the source files of the current version of this API are of mixed programming languages, it is not possible to package up into a java library that can be simply imported into a program using an import statement. The source files have to be manually imported and the package name at the top of each source file have to be changed to the user's package name. This is a minor flaw that can be easily rectified by even the most inexperienced Java programmer.
The code shown in the appendices is only for one operation and for one data type. Although in Appendix 12 there is an extract of this API implemented for the floating point data type, again the entire implementation is given on the attached CD.
The instructions on how to use this API in an Android application are included in Appendix 17.

Benchmarking the RenderScript API

Those flaws aside, it was possible to benchmark the RenderScript API to an extent and compare the results to the Java equivalent test. A sample piece of the RenderScript benchmark test is contained in Appendix 14. The way the code works is almost identical to that previously described, with the exception of all of the values in the array are being sent to memory at once and all of which are processed at once, but the values are not sent back to the Java layer. But what has been added is a timing scheme similar to that of the Java benchmark test, where a system time stamp gets saved into a variable prior to the operation starting, and once again after the operation has finished; the difference between the two time stamps is the time taken. This time the rsSendToClientBlocking() function is used to send back the time taken for the operation to execute. The resolution of this timer is in microseconds, the same as the one developed in Java. A screen shot of the result of the two tests is given in Appendix 15 and a table of results of performance tests are given in Appendix 16.

From the table of results, it is apparent that the RenderScript test ran faster at smaller numbers of array elements than the Java equivalent. But when more operations were required then the RenderScript test ran significantly slower than the Java equivalent, this is most likely down to the fact that the array values are stored in GPU memory and it takes time to access them, whereas the Java equivalent is retrieving values from system memory. The time taken to access the results form GPU memory quickly offsets the performance gain from using RenderScript. Both of the benchmark tests only time how long it takes to process the operations, and for the RenderScript test doesn't take into account the time it takes to store the values into GPU memory, and it doesn't send any values back to the Java layer. So it is safe to say that the current build of this RenderScript Accelerator API is not as fast as the Java equivalent program, apart for really small amounts of array operations.

Discussion

Pros of and Cons of the Current API Build

As it stands the current build of the Accelerator style API performs three out of the four steps detailed in the procedure in their entirety. It performs step two exactly to specification, it creates a copy of the data and stores it in the GPU. Once the data has been processed it will reassemble the data correctly and send it back the Java layer for further processing. Although it takes far too long to send back the processed results to make it a viable tool.

As can be seen in the instructions in Appendix 17, some lines of code needed to be added to allow the use of the RenderScript API functions, namely the creation of the RenderScript context, and creating of an object from the generated class. Although it's only two lines of code, it would desirable to have them in the class files, but they need to be included in the Activity class (the source file that the user interacts with directly). This is only a problem as it's a brief exposure to a foreign API which is counterproductive to what this project is trying to achieve.

So far the API has been implemented for integer data types and floating point data types, but there is a problem with the floating point implementation. The rsSendToClientBlocking() function has input functions of two integers and a const void*, so any data sent back via this will either be converted to an integer or a const *void, either of which require more processing power and time. If it is converted to an integer then some data will be lost, unless it is divided up and the constituent components (i.e. the numbers in the decimal point) get sent over separately. This will require more time and processing power to reassemble the numbers AND reassemble the arrays. This is particularly undesirable as its completely counterproductive to being an Accelerator API.

Although the RenderScript API has the ability to run on the CPU, GPU or DSP (the latter two are planned for later Android OS releases), it doesn't multithread so it doesn't use the two CPU cores or eight GPU cores to full affect, which would greatly improve performance.

Further Development

To overcome the data transfer problem, it's possible to retrieve the values directly from memory using a similar function that was used to transfer the value over to memory in the first place:

```
Arrays.get_A(j);
```

This will get the variable A from the structure indexed by the value j, but there is a problem here. Because of the asynchronous nature of RenderScript, it's difficult to tell when the value will be processed and ready for retrieval. It's possible to write some code to act as a flag to tell the Java layer that the data is now ready for retrieval from memory. Although if the program becomes multithreaded (explained below), then some threads may be finished before others and this could lead to some gaps in the data. Because the number of threads created would be decided at runtime then it will be difficult to predict what will be finished and when.

The most plausible and desirable solution for the floating point data transfer again will be to access the values directly from memory when the operation is complete. The same problem exists though

is that the Java layer doesn't know when the RenderScript layer has completed the operation, so some form of flag would need to be created to tell the Java layer to retrieve the function. Therein lies the problem as the Java code will continue to proceed regardless of whether the operation has completed or not, so it's also important to make the Java code block whilst it waits for the data to be ready for retrieval.
To ensure that the API multithreads the program as in stage three detailed in the procedure, this will require the use of the "For Each"[14] strategy of RenderScript. This "For Each" strategy is often used in Android development in image processing, on the Java side the programmer would create a memory allocation object from an image file and call the RenderScript code. The code is a kernel written in such a way that it's meant to operate on one pixel, but the "For Each" strategy tells the RenderScript to operate on each pixel using that kernel and can spawn the threads it needs to do this. Implementing this would in theory perform the entirety of step three as mentioned in the procedure.
The main problem is that RenderScript is quite heavily targeted towards processing images, so the functions are often designed for this purpose, as is the "For Each" strategy. It may be possible to use the "For Each" strategy directly on arrays, but failing that it may also be possible to convert the arrays to an image of sorts such as bit map file, where the elements of the array are mapped to pixels and given "colours" instead of numerical values.
Assuming these obstacles could be overcome with further development time and resources, it would also make sense to implement this API for other data types such as long and double data types. But it would also make sense to implement other functions rather than just the simple mathematical functions. For example some digital signal processing functions such as the implementation of a digital filter or one of the various Fourier transforms. This sort of implementation makes sense as a RenderScript can target a digital signal processing chip's architecture (once the feature to run on this sort of device is released of course).
Once all of those issues were tackled, then if possibly, somehow consolidate the two lines of code to the class files to limit programmer exposure to the API, but in the grand scheme of what needs fixing this can be considered an afterthought.

Conclusion

Microsoft Accelerator is a very quick and easy way for someone with a moderate amount of experience with C# or C++ programming to quickly optimise a program to use data parallelism that were originally written to use the classical sequential paradigm. But the implementation wasn't architecture agnostic, it still required some input from the user to target a particular architecture, be it a multicore CPU or a GPU.
As it stands, the RenderScript built Accelerator API is a good proof of concept that it is possible to implement a library that can easily be imported and used to optimise applications written in the sequential paradigm. Although a lot of the features that were used to create this and still are in their infancy (i.e. the lack of availability to use the GPU or DSP chip), the architecture agnostic nature of RenderScript can mean that this code will run on any processor on the device without the user having to target it specifically, the RenderScript runtime will decide where it is best to offload the operation, making it incredibly portable and future proof. Although there are performance issues due to the fact a delay was added to ensure proper data transfer back from GPU memory. Also it may have been possible to package everything into a single file to be imported easily, although it was not immediately obvious due to the fact that the API was written in two different programming languages and a lot of the program interacts with automatically generated code in a different folder that shouldn't be manipulated.

[14] Pro Android Apps Performance Optimisation Kindle Edition, APress, Hervé Guihot, location 5047 of 5885

Given more time and resources, this could have evolved into a potentially more powerful API that can be used not only by programming novices but also engineers looking for some portable processing power.

Project Evaluation

Overall the project has progressed mostly according to that planned in the project proposal given in Appendix 18 and the projected Gantt chart given in Appendix 19. In terms of the technical details of the project, it was originally proposed to either use Java on its own or the NDK along with OpenGL. But instead the projected moved to a combination of both Java and RenderScript, to implement a data parallelism library.

During the research phase, the projected research was mostly centred on how to use the NDK and OpenGL. Although once RenderScript came up, this proved to be more of a promising avenue for this project, so more time went into research that.

When it came to the implementation, the Java benchmark programs were started quite early in semester one, but were continually refined and tweaked until the end of semester two. The implementation of the RenderScript Accelerator library started quite soon in semester two as projected on the Gantt chart. Although the final testing and debugging kept going for a few weeks longer than projected and continued until the end of semester two.

There were no major setbacks to the projects progression, which is satisfactory given the scope of the project and the steep learning curve of all of the tools.

There were no costs for this project, no software or hardware purchases were required, it was already available and the software tools are all free and readily available.

Appendices

Appendix 1: Microsoft Accelerator Code

Below are two pieces of code written in C#, they are both basic benchmark tests that time how long it takes to perform one thousand addition operations. One is basic sequential C# code whereas the other uses Microsoft Accelerator to parallelise the operation on a GPU. An Annotated version of the first block of code can be found on the attached CD at:
BenchmarkWithoutAccelerator\BenchmarkWithoutAccerator\Program.cs
It is recommended that if you wish to view this right click the file and open with Notepad or WordPad. A working build of the code below can be found at:
BenchmarkWithoutAccelerator\BenchmarkWithoutAccelerator\bin\Release\BenchmarkWithoutAccelerator.exe

```
//Benchmark program without Microsoft Accelerator
using System;
using System.Diagnostics;
using System.Runtime.CompilerServices;

class Benchmark {
    static void Main(string[] args) {
        15umber15nt _arrayLength = 1000; Random ranf = new Random();
        float[] inputArray1 = new float[_arrayLength];
        float[] inputArray2 = new float[_arrayLength];
        float[] outputArray = new float[_arrayLength];
        var s1 = Stopwatch.StartNew();

        for (int j = 0; j < _arrayLength; j++) {
            inputArray1[j] = (float)ranf.NextDouble() * 10;
            inputArray2[j] = (float)ranf.NextDouble() * 10;
            outputArray[j] = inputArray1[j] + inputArray2[j];
            Console.WriteLine(outputArray[j].ToString());
        }
        s1.Stop();
        Console.WriteLine(((double)(s1.Elapsed.TotalMilliseconds * 1000000) /
_arrayLength).ToString("0.00 ns"));
        Console.Read();
    }
}
```

An annotated version of the following block of code can be found on the attached CD at:
BenchmarkWithAccelerator\BenchmarkWithAccelerator\BenchmarkWithAccelerator.cs
Again it is recommended you view this with either Notepad or WordPad unless you are running a system with Microsoft Visual Studio C# 2010 Express, then this projected can imported and view that way.
A working build of this code can be found at:
BenchmarkWithAccelerator\BenchmarkWithAccelerator\bin\Release\BenchmarkWithAccelertor.exe

```
//Benchmark program with Microsoft Accelerator
using System;
using System.Diagnostics;
using System.Runtime.CompilerServices;
using Microsoft.ParallelArrays;
using FPA = Microsoft.ParallelArrays.FloatParallelArray;
using PA = Microsoft.ParallelArrays.ParallelArrays;

class Benchmark {
    static void Main(string[] args) {
        int i, j;
```

```
        16umber16nt _arrayLength = 1000;
        Random ranf = new Random();
        float[] inputArray1 = new float[_arrayLength];
        float[] inputArray2 = new float[_arrayLength];
        float[] outputArray = new float[_arrayLength];

        DX9Target evalTarget = new DX9Target();

        for (j = 0; j < _arrayLength; j++) {
            inputArray1[j] = (float)ranf.NextDouble() * 10;
            inputArray2[j] = (float)ranf.NextDouble() * 10;
        }

        FPA fpInput1 = new FPA(inputArray1);
        FPA fpInput2 = new FPA(inputArray2);

        var s1 = Stopwatch.StartNew();
        FPA fpOutput = PA.Add(fpInput1, fpInput2);
        evalTarget.ToArray(fpOutput, out outputArray);
        s1.Stop();

        Console.WriteLine(((double)(s1.Elapsed.TotalMilliseconds * 1000000) /
_arrayLength).ToString("0.00 ns"));
        Console.Read();
    }
}
```

Both of these programs were written using the C# programming language, this bares a lot of resemblance to Java in terms of syntax and some of the concepts required to use it.
The first program creates two floating point arrays of one thousand elements, fills them with random numbers between 1 and 10, and then adds them together whilst display the result of each addition to the console window.
This operation is being timed using the stopwatch functions, which is called either side of the operation (like pressing the start and stop button of a stop watch); once the operation is done the execution time in nanoseconds is displayed to the console window. The faster the execution time the higher the performance.
The second piece of code does exactly the same but using Microsoft Accelerator, the extra code that requires the use of Accelerator is highlighted in **BOLD** type(as shown in Appendix 1) to illustrate how easy it is to optimise such a program.
The first difference is the extra "using" statements (using statements are like the import statements in Java or #include statements in C/C++ where a file of the required functions is selected):

```
using Microsoft.ParallelArrays;
using FPA = Microsoft.ParallelArrays.FloatParallelArray;
using PA = Microsoft.ParallelArrays.ParallelArrays;
```

This just tells the program to use the parallel arrays functions within Microsoft Accelerator and creating two new objects from the subclasses FloatParallelArray and ParallelArrays. They are required to create an array that the Accelerator functions can interpret and to invoke the addition function. The next line of code to note is this one:

```
DX9Target evalTarget = new DX9Target();
```

This creates a new object from the DX9Target class, which means the program will be using the features built into DirextX9 to run the operation on the GPU. To make sure the user created data can interact with the Accelerator functions, it needs to be converted to a FloatParallelArray:

```
FPA fpInput1 = new FPA(inputArray1);
FPA fpInput2 = new FPA(inputArray2);
```

This puts the data into two objects that can interact with the Microsoft Accelerator function. Now to invoke the desired operation:

```
FPA fpOutput = PA.Add(fpInput1, fpInput2);
```

This creates a new array called fpOutput which is the result of the parallelised addition of the two input arrays. The Add function is called by using the ParallelArrays object PA created earlier. And finally to return the result to the program so the user can continue operating on the result array if they wish:

```
evalTarget.ToArray(fpOutput, out outputArray);
```

This will output the data in FloatParallelArray object fpOutput to a simple array named outputArray, which can now be operated on with simple C# functions.
This just goes to show how little effort it requires to optimise a program to implement the more modern practice of data parallel programming using a few simple functions included in Microsoft Accelerator. It also gives something to aim for when creating an Android Accelerator API in terms of ease of use if someone wishes to use this to optimise their own programs.
The important thing to note with using Microsoft Accelerator is that in order to implement it efficiently, it is recommended that it performs a large number of operations to offset the time it takes to transfer the data from system to memory to GPU memory and back again.[15]
Two screenshots of this program running on two different systems is given in Appendix 14, along with the specifications of the GPUs that it has run on.
As can be seen there execution times are different, the NVidia GTX 460 ran slower than the NVidia GT 520 MX. There are a few reasons for this disparity, although the 460 has more CUDA cores than the 520, and therefore can make more threads, the 520 has a higher clock speed so the processing time of each thread is faster. Also because the GTX 460 is a desktop GPU, connected to the computers motherboard, it takes time to transfer the information to and from GPU memory. Whereas with the GT 520, the laptop motherboard is more compact and all of the hardware is much more local, hence has a faster transit time to and from GPU memory. This reinforces the point made that using Accelerator needs to be carefully considered, as there is a trade-off for the performance gain; can this performance gain offset the trade off? The same will be true of an Accelerator like API built for Android.

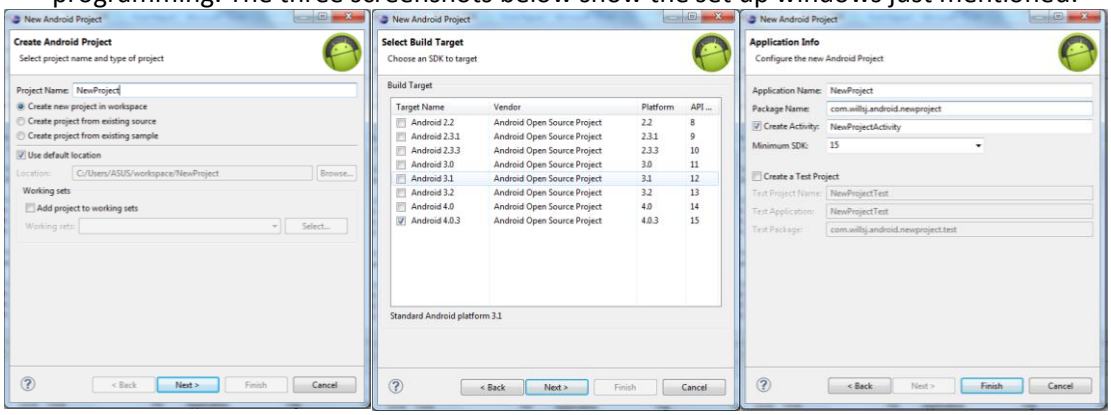
Appendix 2: Basic Android Application Development Guide

In order to program for Android devices, a set of tools is required, but in recent months these have become very easy to acquire from NVidia's Developer site[16]. There is a link to get the entire development kit required to recreate any code put together in this project and it all gets installed automatically, including the assignment of environment variables etc. This makes it possible to start writing programs for Android devices in a very short space of time and is incredibly user friendly.
For any of those who have programmed Java applications before on a PC, Android development is relatively intuitive. But for those that don't know how to get started on Android development, here is a simple step by step guide in building a basic Android application, like the one that was used in this project.

1. Creating the project: Android development is done in an Integrated Development Environment (IDE), in the case of this particular project; Eclipse was the IDE of choice. So from the top menu bar select, **File→New→Android Project**. A window will open prompting the user to enter a project name; another window will prompt a selection of Android operating system. The Software Development Kit (SDK) target for this project was Android 4.03 (Ice Cream Sandwich). Finally the user will be prompted to name the Java package and

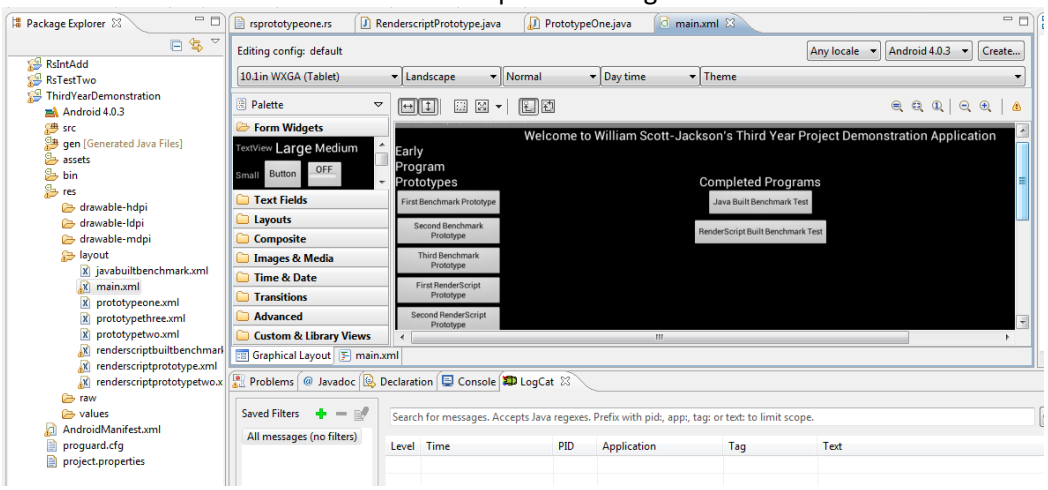
[15] http://research.microsoft.com/en-us/projects/accelerator/accelerator_intro.docx page 27
[16] http://developer.nvidia.com/tegra-android-development-pack

their first Activity, clicking **Finish** will generate some code so the user can start programming. The three screenshots below show the set up windows just mentioned:



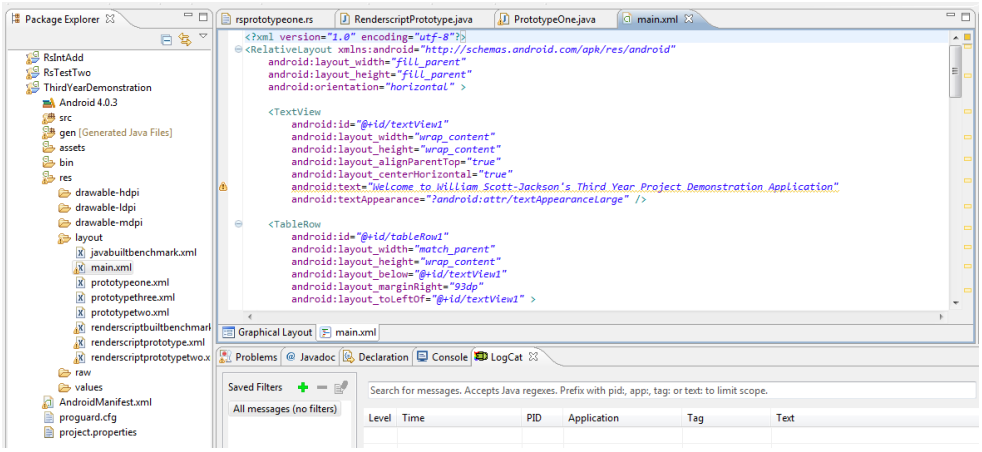
2. Create the layout[17]: One of the files generated in the project folder is the layout Extensible Mark-up Language (XML) file. This can be edited using a combination of a graphical editing tool and a text editor, as shown in the screens below. The graphical editing tool can be used for picking and placing elements of the user interface such as buttons that react to screen touches and bodies of text. The text editor can be used to assign identities to these elements so that the user can interact with them in their code, but also label them with text and specify their position with a simple coordinate type system as opposed to using a mouse to drag and drop them.

XML Graphical Editing Tool



XML Text Editing Tool

[17] Android 3.0 Application Development Cookbook (Kindle Edition), Packt Publishing, Kyle Merrifield Mew, Location 793 of 4119 (Page 33 of 261)



3. Write the code: When writing applications for Android, it's important to note that the way they interact with the user and how they are displayed are slightly different to what people may be used to already. Android works in Activities[18] which is essentially their equivalent of the windows found in most modern PC operating systems such as Windows, Linux and Mac. An activity is a way of presenting information to the user and gives them a means of interacting with the application. When the project is created, a simple piece of code (shown below) is generated that the programmer can start working from:

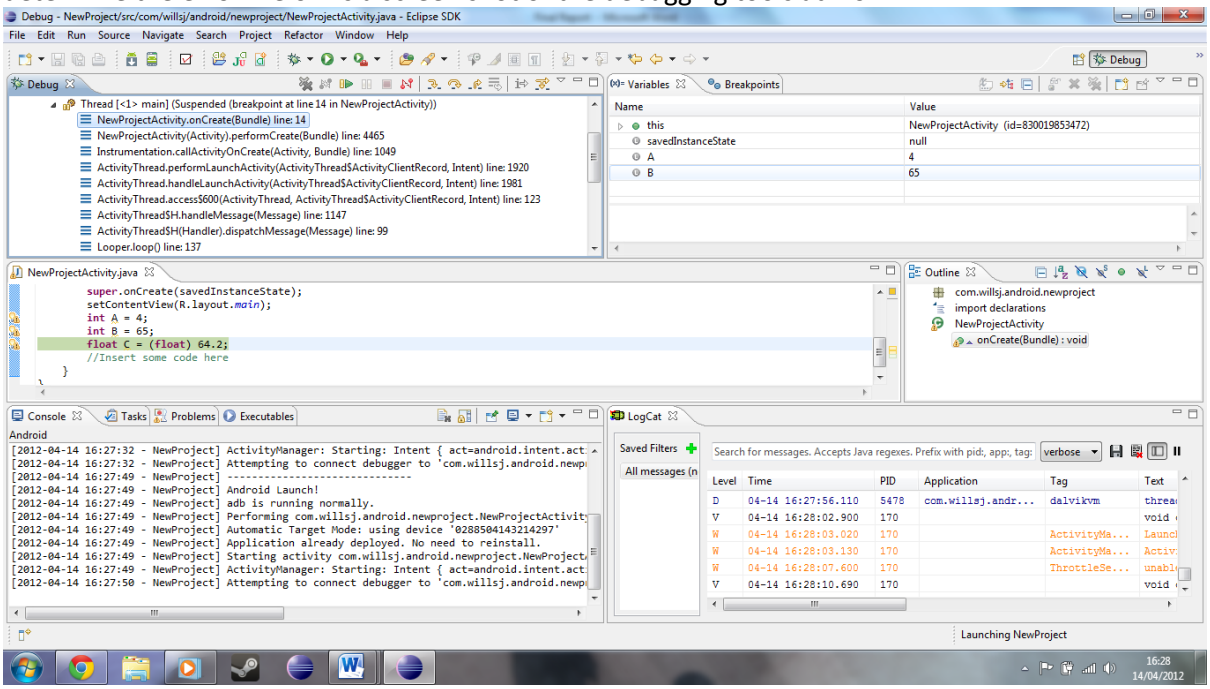
```
public class NewProjectActivity extends Activity {
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      //Insert some code here
      }
}
```

What is generated is the onCreate() method which contains the instructions to execute once the activity is generated, i.e. when the application fires up or when switching activities. Code can be written outside of this in either an onStart() method which is what happens once the Activity has been generated and the initial operations in the onCreate() method have finished, or an onClick() method which listens for an on screen button press. Examples of these can both be found within the test application.

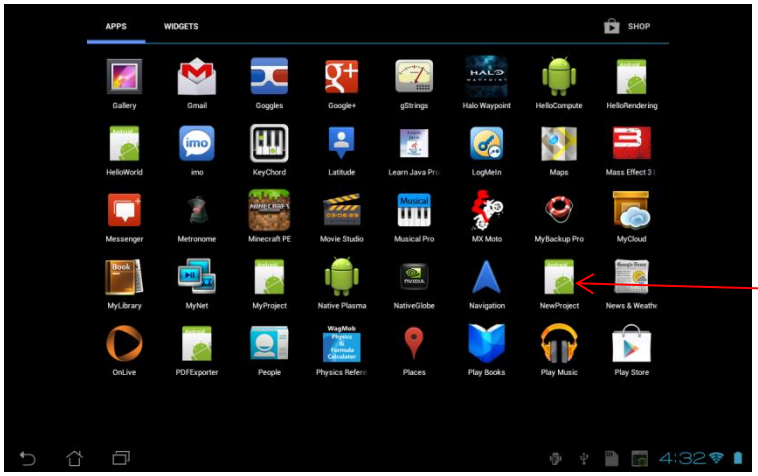
4. Debug the code: The Eclipse IDE has its own set of debugging tools for Android applications, if there are any errors beyond simple syntax errors then it's possible to try and find them using the debugger. The user can set up break points where they suspect there might be an error in the logic of their code and inspect the individual elements of the program to try and

[18] Android 3.0 Application Development Cookbook (Kindle Edition), Packt Publishing, Kyle Merrifield Mew, Location 417 of 4119 (Page 5 of 261)

determine the error. Below is a screen shot of the debugging tools at work:



5. Run the code: Once any errors have been removed then the user just needs to run the application, if this is the first successful build then it needs to be run from the Eclipse IDE by clicking the **Run** button in the menu and select the option to have it run as an Android Application. This installs the application onto the Android device; if the build and install is successful then the user can run the application from the Android device by clicking on the icon from the Android's Application menu:



Here is the icon for the test application. Just click it to run

It's easy to see that Android development is incredibly simple and helps emphasise the need to create an API that can optimise an application for someone who perhaps isn't as experienced with using performance enhancing programming techniques. It is more likely that entry level programmers may start mobile development with Android given the accessibility of the tools and relative ease of writing simple programs.

To run this project on an Android device, follow the first step but when prompted to create a new project, select the option **Create from Existing Code** and select the project directory on the attached CD. This will only work on an Android device and not the built in emulator as some of the features used are not supported by the emulator.

Appendix 3: Basic Android Native Development Guide

The Native Development Kit (NDK) is just one of the tools available to developers to help them write high performance applications for Android devices. Any performance critical operations should be written in C++ and get called from the Java layer when required. This method of development greatly increases the complexity, below is a step by step guide of how to build a very simple application using the NDK. The following example is taken from Android NDK: Beginner's Guide[19]:

1. Create the layout and Java source code: Following the steps from Appendix Two up to step three. Create the layout to suit the required application and write any non-performance critical code in Java.
2. Create the Java Native interface subdirectory: The next step is creating a subdirectory in the project folder called "jni". JNI stands for Java Native Interface, this is an interface that allows a Java program to call low level high performance code written in languages such as C++ and Assembly; in this case the language being called is C++. In this subdirectory create a file called Android.mk, there is no need to specify the file type as the tools well spot the file extension and generate the appropriate file type. Input the following code into the file Android.mk:

```
1. LOCAL_PATH := $(call my-dir)
2.
3. include $(CLEAR_VARS)
4.
5. LOCAL_MODULE     := mylib
6. LOCAL_SRC_FILES := mylib.c
7.
8. include $(BUILD_SHARED_LIBRARY)
```

   This file is known as a Makefile[20] and is an important component of the NDK build process; it defines what to compile and how to compile it.

3. Write the C++ source code: Now is a good time to write the critical performance critical operations using C++. Below is some sample code from the aforementioned text[21]:

```
#include <string.h>
#include <jni.h>
jstring Java_com_myproject_MyProject_getMyData(JNIEnv* pEnv, jobject pThis) {
      return (*pEnv)->NewStringUTF(pEnv, "My native project talks C++");
}
```

   This code will return a simple string to the Java layer from where it was called.

4. Integrate the Java code and C++ code: Now it's time to put it all together to have a program that performs non-performance critical, layout, user interface code in Java and performance critical operations in C++. So the Java code of this example is as shown below:

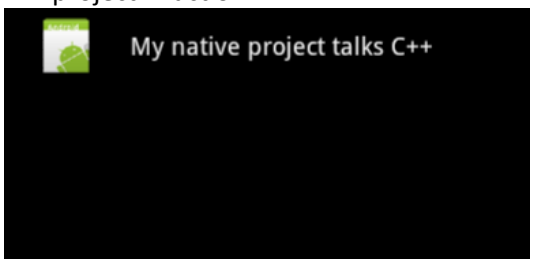
```
package com.myproject;
import android.app.Activity;
import android.os.Bundle;

public class MyProject extends Activity {
      public native String getMyData();
      static {
            System.loadLibrary("mylib");
      }
```

[19] Android NDK: Beginner's Guide Kindle Edition, Packt Publishing, Sylvian Ratabouil, Location 1205 of 6290
[20] Android NDK: Beginner's Guide Kindle Edition, Packt Publishing, Sylvian Ratabouil, Location 1282 of 6290
[21] Android NDK: Beginner's Guide Kindle Edition, Packt Publishing, Sylvian Ratabouil, Location 1242 of 6290

```
	public void onCreate(Bundle savedInstanceState) {
	super.onCreate(savedInstanceState);
		setTitle(getMyData());
		setContentView(R.layout.main);
	}
}
```

This simple piece of Java code[22] sets the layout to that defined in the XML file and sets the title to the string contained in the C++ by calling the native method defined just above the code. The method above the onCreate() method has the "native" identifier to denote that it is going to interface with some native code (e.g. C++) and the "String" identifier denotes that it returns a string. The command inside this method loads the library named "mylib" which is defined in the Makefile defined in step 2.

5. The project needs to be converted to a C++ project in eclipse, this can be done by right clicking the project folder in the IDE then **Other→Convert to C++ Project**. Then some final adjustments need to be made to some of the environment variables, and some build commands need to be established, but some of this will depend on the operating system and the development environment the programmer is using.
6. Run the project: This can be done in the same way as other Android projects by clicking on the run icon in the tool bar towards the top of the screen. Below is a screen shot of the project in action.



All this application does is sets the title of the project to that defined in the C++ code.

Appendix 4: Using RenderScript in Android Applications

The advantage that RenderScript possesses of the Native Development Kit is that in terms of development complexity it is the lesser of two evils. RenderScript is more complicated than just Java development but it's easy interfacing the two codes together and there is less time spent working on project settings. Also there is less time spent making sure the development environment it set up properly, the NDK requires the programmer to use a Bash Shell (either using the Bash command line or establishing build commands in the IDE) which can be incredibly daunting for a novice programmer, so it hasn't been covered in this report. So here is a step by step guide on how to write a basic Android application and including some RenderScript code to improve performance.

1. Create the layout and Java code: Again like NDK development, the first port of call is to create a simple user interface and some simple non-performance critical Java code.
2. Write the RenderScript code: This is the more complex part of the development process, RenderScript is essentially the C (C99 standard) programming language with functions that are all tailored to be executed on an Android device.
   The first thing that needs to be done by the programmer is define the version of RenderScript that they are working with; this is done with a simple #pragma statement. A #pragma statement is a way of telling the IDE or compiler if there are any hardware and operating system specific features the programmer needs[23]:

```
#pragma version(1)
```

[22] Android NDK: Beginner's Guide Kindle Edition, Packt Publishing, Sylvian Ratabouil, Location 1209 of 6290
[23] http://msdn.microsoft.com/en-us/library/d9x1s805(v=vs.71).aspx

And another #pragma statement that tells the system what Java package the RenderScript code is contained in:

```
#pragma rs java_package_name(com.android.willsj.example)
```

By convention the RenderScript system looks out for potentially two blocks of code, root() and init(). A root() function is a piece of code that is called repeatedly at a set interval of time defined by the programmer, and the init() function is for initialising variables. For anyone familiar with programming for an Arduino microcontroller it is similar to the setup() and loop() functions. The structure of a root()[24] function is given here:

```
int root() {
     //Do some operation here
     return 10;
     }
```

This root() function executes repeatedly at an interval defined in the return statement. The return statement tells the system the running interval in milliseconds. So in this example here, return 10; means that the function should run once every 10 milliseconds. If the device is too weighed down with processor intensive operations, it will run as fast as it can. Returning 0 will mean the operation will run only once.

A typical init() function is shown here:

```
void init() {
     int A = 12;
     float B = 2.4;
     double C = 23.23;
}
```

This is the first function that is called in RenderScript; it's the ideal place to initialise variables and parameters[25].

Alternatively it is possible to declare functions of your own such as:

```
void add() {
       int A = 2;
       int B = 2;
       int C = A + B;
}
```

The important thing to note about RenderScript is that whenever code is written in the source file, some Java code is automatically generated[26] by the Android tools so that the programmer can interact with the RenderScript code. A Java class filed will be given the name ScriptC_rsfilenamehere.java, and the user can use functions such as invoke_add() to call their RenderScript functions. If a user includes a struct in their program, then another class file is generated for that. But that will be discussed in the results as this project uses structs.

3. Write some Java code to interact with RenderScript: As just mentioned, when a RenderScript source file is written, some Java code is generated. As a simple example, if the add function

[24] http://mobile.tutsplus.com/tutorials/android/getting-started-with-renderscript-on-android/
[25] http://mobile.tutsplus.com/tutorials/android/getting-started-with-renderscript-on-android/
[26] http://developer.android.com/guide/topics/renderscript/index.html#func

shown above is written and the programmer wants to call if from Java, the relevant block of code will look like this.

```
RenderScript rs = RenderScript.create(this);
ScriptC_rsexample mScript = new ScriptC_rsexample(rs, getResources(),
R.raw.rsexample);
mScript.invoke_Add();
```

The above code will create a new RenderScript context, a context is a class that allows access to various application specific resources, in the case of Android for example, the use of their notifications and GPS location services. But in this case the specific resources are those offered by RenderScript. The second line of code creates an object from the generated class to give access to the functions within. The functions within this generated class depend on what was written in the RenderScript file. If variable declarations were included, then it's possible to set the values of those variables from the Java layer (again this is used and will be elaborated in the results section).

4. Run the code: Applications like these can be run in exactly the same way as regular Android applications, by just clicking on the **Run** button from the menu bar.
   As was demonstrated here, RenderScript is a much more viable solution for application optimisation, as although it is more complicated the extra code can added when it is needed rather than having to put more time and thought into the structure and code of the application as is required of the NDK. Hence this is why RenderScript was chosen for the implementation of this project.

Appendix 5: First Android Benchmark Prototype Development

The full source code plus code annotations can be found in the attached CD. It is located at **ThirdYearProjectDemonstrationTwo/src/com/willsj/android/thirdyeardemo/PrototypeOne.java**
The XML file of the layout of this part of the program can be found at **ThirdYearProjectDemonstrationTwo/res/layout/prototypeone.xml**
A screenshot of this code in action can be found in Appendix 15.

```
Calendar Start = Calendar.getInstance();
int startMinute = Start.get(Calendar.MINUTE);
int startSecond = Start.get(Calendar.SECOND);
int startMillisecond = Start.get(Calendar.MILLISECOND);
Start.roll(Calendar.MINUTE, true);
int endMinute = Start.get(Calendar.MINUTE);
int endSecond = Start.get(Calendar.SECOND);
int endMillisecond = Start.get(Calendar.MILLISECOND);
int loopCounter = 0;
while (true) {
	float A = (float) Math.sqrt(Math.random());
	GregorianCalendar now = new GregorianCalendar();
	if (now.get(Calendar.MINUTE) >= endMinute) {
		if (now.get(Calendar.SECOND)>= endSecond) {
			if (now.get(Calendar.MILLISECOND) >= endMillisecond) {
				break;
				}
			}
		}
	loopCounter++;
}
Description.setText("");
notifyUser.append("\n Test Complete");
notifyUser.append("\n The test did " + loopCounter + " loops in one minute");
```

In order to provide some sort of benchmark to aim for, a basic performance test was put together using Java. In theory, if the project went as planned then the equivalent RenderScript layer operation will run faster than the Java layer. An extract of the first prototype that was inspired by code found in Sam's Teach Yourself Java in 24 Hours[27] is shown above. The first thing this program does is finds the current time using the Calendar[28] functions built Java:

```
Calendar Start = Calendar.getInstance();
int startMinute = Start.get(Calendar.MINUTE);
int startSecond = Start.get(Calendar.SECOND);
int startMillisecond = Start.get(Calendar.MILLISECOND);
```

This gets the components of the current time up to millisecond resolution and stores each of these components in variables. In order to make the test run for one whole minute, the time variable is rolled ahead by one whole minute so that the program knows what time stamp to look for:

```
Start.roll(Calendar.MINUTE, true);
```

Then the components are stored into variables:

```
int endMinute = Start.get(Calendar.MINUTE);
int endSecond = Start.get(Calendar.SECOND);
int endMillisecond = Start.get(Calendar.MILLISECOND);
```

Then the program will run within a while loop until the minute is up and print the number of times the loop ran to the display. Naturally the higher the number of iterations, the faster the program has run. There are a few problems with this program though, the first problem is that the resolution of the test is in milliseconds, these days modern computers even tablet computers run in GigaHertz. So individual instructions can execute in the tiniest fractions of a second and millisecond resolution just won't be accurate enough to measure with. Secondly the program appears to lock up for the minute that it is running, so some sort of visual feedback is required to assure any user that the program hasn't crashed. And finally the if statements within the while loop outweigh the simple mathematical operation in terms of processing power, so if it ever came around to calculating some value of operations per second, then this test is not practical

Appendix 6: Second Android Benchmark Prototype Development

The full source code plus code annotations can be found in the attached CD. It is located at **ThirdYearProjectDemonstrationTwo/src/com/willsj/android/thirdyeardemo/PrototypeTwo.java**
The XML file of the layout of this part of the program can be found at **ThirdYearProjectDemonstrationTwo/res/layout/prototypetwo.xml**
A screenshot of this code in action can be found in Appendix 15.

```
Float[] A = new float[3000000];
float[] B = new float[3000000];
float[] result = new float[3000000];
int counter = 0;
for (int i = 0; i < 3000000; i++) {
      A[i] = (float) Math.random();
      B[i] = (float) Math.random();
}
Calendar Start = Calendar.getInstance();
int startMinute = Start.get(Calendar.MINUTE);
int startSecond = Start.get(Calendar.SECOND);
```

[27] Sam's Teach Yourself Java in 24 Hours Kindle Edition, Sam's, Rogers Cadenhead, location 2089 of 9011
[28] Java In a Nutshell, O'Reilly, David Flanagan, page 763

```
int startMillisecond = Start.get(Calendar.MILLISECOND);
for (int i =0; i < 3000000; i++) {
      result[i] = A[i] + B[i];
      counter++;
}
Calendar now = Calendar.getInstance();
int nextMinute = now.get(Calendar.MINUTE);
int nextSecond = now.get(Calendar.SECOND);
int nextMillisecond = now.get(Calendar.MILLISECOND);
int minutesTaken = nextMinute – startMinute;
int secondsTaken = nextSecond – startSecond;
int millisecondsTaken = nextMillisecond – startMillisecond;
```

It was worth developing another test, enter the second benchmark prototype, an extract of this is shown above. What this second test does is gets the current time, stores it in a variable:

```
int startMinute = Start.get(Calendar.MINUTE);
int startSecond = Start.get(Calendar.SECOND);
int startMillisecond = Start.get(Calendar.MILLISECOND);
```

Instead of running for a set amount of time, runs for a set amount of array element operations e.g. three million:

```
for (int i =0; i < 3000000; i++) {
      result[i] = A[i] + B[i];
      counter++;
}
```

Once the test has done all of the operations, gets the current time and stores that into a set of variables:

```
int nextMinute = now.get(Calendar.MINUTE);
int nextSecond = now.get(Calendar.SECOND);
int nextMillisecond = now.get(Calendar.MILLISECOND);
```

The difference between the start time and end time will tell the user the time that the program has taken to run:

```
int minutesTaken = nextMinute – startMinute;
int secondsTaken = nextSecond – startSecond;
int millisecondsTaken = nextMillisecond – startMillisecond;
```

This test runs a lot better than the first one, although the resolution is still in milliseconds and there is still a need for some user feedback, for example a progress or spinner.

Appendix 7: Third Android Benchmark Prototype Development

The full source code plus code annotations can be found in the attached CD. It is located at **ThirdYearProjectDemonstrationTwo/src/com/willsj/android/thirdyeardemo/PrototypeThree.java**
The XML file of the layout of this part of the program can be found at **ThirdYearProjectDemonstrationTwo/res/layout/prototypethree.xml**
A screenshot of this code In action can be found in Appendix 15.

```
Int MAX_PRIORITY = 1;
int MIN_PRIORITY = 2;
Handler handler = new Handler() {
      @Override
      public void handleMessage (Message msg) {
            String valueReturned = (String)msg.obj;
            if (testCountdown == true) {
                  testProgress.setVisibility(View.INVISIBLE);
                  testRunning.setText("" + valueReturned);
```

```
            }
            if (testStarted == true) {
                  testProgress.setVisibility(View.VISIBLE);
                  testRunning.setText("" + valueReturned);
            }
      }
};
public void onStart() {
      super.onStart();
      Thread background = new Thread(new Runnable() {
            public void run() {
                  testCountdown = true;
                  testStarted = false;
                  testFinished = false;
                  TimerFunctions sleep = new TimerFunctions();
                  for (int j = 5; j>-1; j--) {
                        String output = "The test will start in " + j + "
seconds";
                        sleep.timerInSeconds(1);
                        Message msg = handler.obtainMessage(1, output);
                        handler.sendMessage(msg);
                  }
                  testCountdown = false;
                  testStarted = true;
                  String output = "Test running... Please wait";
                  Message msg = handler.obtainMessage(1, output);
                  handler.sendMessage(msg);
                  try {
                        /*Perform the same test as demonstrated in Appendix
six*/
output = "The test is finished. The time taken is " + minutesTaken + " minutes, "
+ secondsTaken + " seconds and "
+ millisecondsTaken + " milliseconds.";
                        msg = handler.obtainMessage(1, output);
                        testFinished = true;
                        handler.sendMessage(msg);
                  } catch (Throwable t) {
                        output = "Error detected, ending the thread";
                        msg = handler.obtainMessage(1, output);
                        handler.sendMessage(msg);
                        }
                  }
            });
            background.setPriority(MAX_PRIORITY);
            background.start();
      }
```

Now looking at the third benchmark prototype given above, this implements the use of a thread to run the processor heavy performance test in the background, whilst displaying to the user feedback in the form of a progress spinner. A five second countdown as added too to show the user the test is about to start. Also to make sure that the time taken for the test taken to execute is representative of the performance possible from the processor, the thread was given maximum operating system priority. This means that the instructions will execute as fast as possible rather than letting the operating system prioritise other tasks first. So in theory the code will execute faster than in the previous sets of code and will provide the user with some feedback. All that was left to finish at this point was to increase the resolution of the timer then implement the test for different mathematical operations and data types.

Appendix 8: Final Java Benchmark Test Program

The full source code plus code annotations can be found in the attached CD. It is located at **ThirdYearProjectDemonstrationTwo/src/com/willsj/android/thirdyeardemo/JavaBuiltBenchmark.java**
ThirdYearProjectDemonstrationTwo/src/com/willsj/android/thirdyeardemo/IntergerMathTest.java
ThirdYearProjectDemonstrationTwo/src/com/willsj/android/thirdyeardemo/FloatingPointMathtest.java
The XML file of the layout of this part of the program can be found at **ThirdYearProjectDemonstrationTwo/res/layout/javabuiltbenchmark.xml**
A screenshot of this code in action can be found in Appendix 15

```
IntegerMathTest intTest = new IntegerMathTest();
FloatingPointMathTest floatTest = new FloatingPointMathTest();
int MAX_PRIORITY = 1;
int MIN_PRIORITY = 2;
public void onStart() {
	super.onStart();
	Thread background = new Thread(new Runnable() {
		public void run() {
			int testsToDo = 10000;
			try {
				countdown();
				String output =  intTest.IntegerAdditonTest(testsToDo);
				sendMessage(output);
				countdown();
				output = intTest.IntegerSubtractionTest(testsToDo);
				sendMessage(output);
				countdown();
				output = intTest.IntegerMultiplicationTest(testsToDo);
				sendMessage(output);
				countdown();
				output = floatTest.FloatingPointAdditionTest(testsToDo);
				sendMessage(output);
				countdown();
				output =
floatTest.FloatingPointSubtractionTest(testsToDo);
				sendMessage(output);
				countdown();
				output =
floatTest.FloatingPointMultiplicationTest(testsToDo);
				sendMessage(output);
				countdown();
				output = floatTest.FloatingPointDivisionTest(testsToDo);
				sendMessage(output);
			} catch (Throwable t) {
				String output = "Error Detected, ending the thread";
				Message msg = handler.obtainMessage(1, output);
				handler.sendMessage(msg);
			}
		}
	});
	background.setPriority(MAX_PRIORITY);
	background.start();
```

This works in a similar way to that shown in Appendix 7, although some improvements have been made. First of all some of the user feedback functionality like the countdown between tests has

been put into its own function countdown(), so the program will countdown to each test and show the user the test is still running. Secondly the actual test and others like it have been consolidated into their own class files **IntegerMathTest. Java** and **FloatingPointMathTest.java**
Although the most significant improvement of this code is the increased resolution of the timer, it now gives the result of the time taken in microseconds. Thanks to the System.nanoTime()[29] function, this provides the current time stamp in nanosecond resolution. Although in the implementation of the code; only the more significant component of the figure was taken i.e. numbers more than one thousand nanoseconds. This gave microsecond resolution as for the number of operations being performed; microsecond resolution was all that was needed to see the difference in execution times.

Appendix 9: Basic RenderScript Code Development

Below is an extract of some code in Java that interacts with RenderScript and the actual RenderScript code. The full source code can be found in the attached CD at
ThirdYearProjectDemonstrationTwo/src/com/willsj/android/thirdyeardemo/RenderScriptPrototype.java
And
ThirdYearProjetDemonstrationTwo/src/com/willsj/android/thirdyeardemo/rsprototypeone.rs
The XML layout file can be found at
ThirdYearProjectDemonstratioTwo/res/layout/renderscriptprototype.xml
A Screenshot of this code in action can be found in Appendix 15.

Java Layer Code

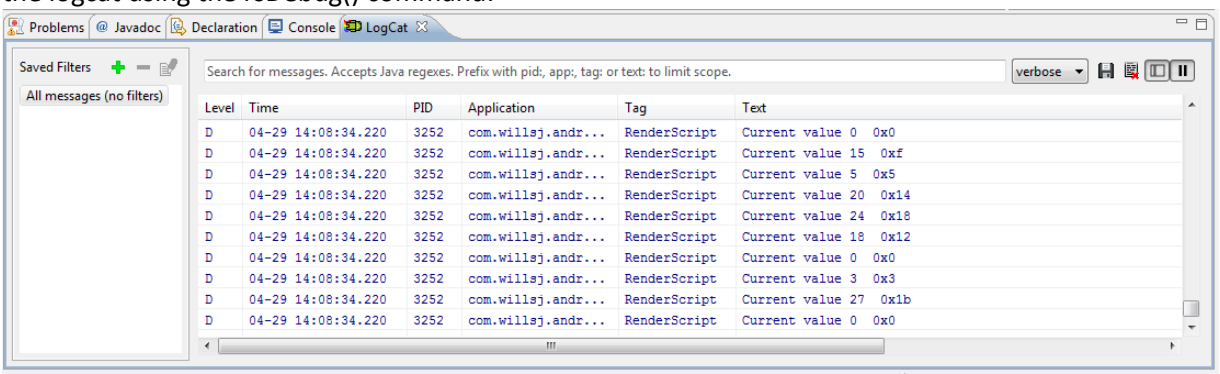
```
private void intAdd(int[] A, int[] B) {
	RenderScript rs = RenderScript.create(this);
	 ScriptC_rsprototypeone intaddscript = new ScriptC_rsprototypeone(rs,
getResources(), R.raw.rsprototypeone);
	 mScript = intaddscript;
	for(int i = 0; i < A.length; i++) {
		setNewValues(mScript, A[i], B[i]);
		intaddscript.invoke_intAdd();
		int C = getResult(mScript);
		notifyUser.append(“ “ + C);
	}
}

public void setNewValues(Script script, int A, int B) {
	mScript.set_numberA(A);
	mScript.set_numberB(B);
}

public int getResult(Script script) {
int C = mScript.get_numberC();
return C;
}
```

RenderScript Layer Code

```
void intAdd() {
	29umber = numberA + 29umber;
	rsDebug(“Current value”, 29umber);
}
```

[29] Java in a Nutshell, O’Reilly, David Flanagan, page 501

This is the first prototype program written using RenderScript to perform mathematical operations. The Java code is a function that responds to a user inputting two integer arrays A and B. The code does the necessary operations such as creating the RenderScript context and creating an object from the class generated in response to the RenderScript code. It then sets the values of the variables numberA and numberB to those pointed to by the index within the for loop. The code calls the RenderScript function; the two numbers are added together and stored in the variable numberC. This number is sent to the Eclipse Logcat so that it's possible to debug the program and ensure it works properly before further developing the code. Below is a screenshot of the data sent back to the logcat using the rsDebug() command:



There are a few problems inherent in this code, the first problem is that the getResult() function does not work as RenderScript is asynchronous, given the nature of the program the Java code will have to wait until that section of code is run before trying to retrieve the value. This would be particularly inefficient doing repeated calls of code and having to reset the values repeatedly. A more efficient method would be to save all of the values to memory, operate on them and retrieve all of them at once.

Appendix 10: An Improved RenderScript Code

Below is another extract of RenderScript code and the Java code required to interact with it. The full source code plus annotations can be found on the attached CD in the directory:
ThirdYearProjectDemonstrationTwo/src/com/willsj/android/thirdyeardemo/RenderscriptPrototypeTwo.java
And
ThirdYearProjectDemonstrationTwo/src/com/willsj/android/thirdyeardemo/rsprototypetwo.rs
The XML file for the layout can be found at:
ThirdYearProjectDemonstratioTwo/res/layout/renderscriptprototypetwo.xml
A screenshot of this code in action can be found in Appendix 15.

Java Layer Code

```
public int[] intAdd(int[] A, int[] B, RenderScript mRs, ScriptC_rsprototypetwo
mScript) {
	int[] D = new int[A.length];
	if (A.length > 1000) {
		int[] E = new int[A.length];
		IntSplitArrays intSplit = new IntSplitArrays();
		E = intSplit.addSplitArrays(A, B, mRs, mScript);
		return E;
	} else {
		D = messageHandler(A.length, mRs);
		saveToMemory(A, B, mRs, mScript);
		mScript.invoke_intAdd();
		sleep();
		return D;
		}
	}
```

```
public void saveToMemory(int[] A, int[] B, RenderScript mRs,
ScriptC_rsprototypetwo mScript) {
		ScriptField_Values Arrays = new ScriptField_Values(mRs, A.length,
Allocation.USAGE_GRAPHICS_VERTEX);
		mScript.bind_values(Arrays);
		Item i = new ScriptField_Values.Item();
		for (int j = 0; j < A.length; j++) {
			i.A = A[j];
			i.B = B[j];
			Arrays.set(i, j, true);
		}
	}

public int[] messageHandler(int length, RenderScript mRs) {
	final int[] D = new int[length];
	mRs.setMessageHandler(new RenderScript.RSMessageHandler(){
		@Override
		public void run() {
			Log.d(" ", String.valueOf(this.mID)+ "  " + mData + "  " +
mLength);
			D[mLength] = mID;
			}
		});
		return D;
	}
```

RenderScript Layer Code

```
typedef struct __attribute__((packed, aligned(4))) Values {
	int A;
	int B;
	int C;
} Values_t;
Values_t *values;

void intAdd() {
	Values_t *pValues = values;
	int index = rsAllocationGetDimX(rsGetAllocation(values));
	for (int i = 0; i < index; i++) {
		pValues->C = pValues->A + pValues->B;
		rsSendToClientBlocking(pValues->C, (const void *) &pValues->C, i);
		pValues++;
	}
}
```

Appendix 11: RenderScript Code Integer Implementation

This is an extract of Java code that divides arrays up into multiples of one thousand (providing that the array is of length greater than one thousand), and sends them to be processed one thousand at a time then reassembled into an array to be sent back to where the user called it from. The full source code plus annotations can be found on the attached CD at:
ThirdYearProjectDemonstrationTwo/src/com/willsj/android/thirdyeardemo/IntSplitArrays.java

```
public int[] addSplitArrays(int[] A, int[] B, RenderScript mRs,
ScriptC_rsprototypetwo mScript) {
```

```
	final int timesSplit = (int) Math.floor((double) (A.length/1000));
	int remainder = A.length - (timesSplit * 1000);
	final int[] E = new int[1000];
	final int[] F = new int[1000];
	final int[] J = new int[remainder];
	final int[] K = new int[remainder];
	int[] G = new int[1000];
	final int[] H = new int[A.length];
	for (int i = 0; i < timesSplit; i++) {
		for (int j = 0; j < 1000; j++) {
			E[j] = A[(i * 1000) + j];
			F[j] = B[(i * 1000) + j];
		}
		G = intAdd(E, F, mRs, mScript);
		for(int k = 0; k < G.length; k++) {
			H[(i * 1000) + k] = G[k];
		}
	}
	if (remainder == 0) {
		return H;
	} else {
		mRs.setMessageHandler(new RenderScript.RSMessageHandler(){
			@Override
			public void run() {
				Log.d(" ", String.valueOf(this.mID)+ " " + mData + " "
+ mLength);
				H[(timesSplit * 1000) + mLength] = mID;
				}
		});
		for (int l = 0; l < remainder; l++) {
			J[l] = A[(timesSplit * 1000) + l];
			K[l] = B[(timesSplit * 1000) + l];
		}
		saveToMemory(J, K, mRs, mScript);
		mScript.invoke_intAdd();
		return H;
		}
	}
```

Appendix 12: RenderScript Code Floating Point Implementation

This is an extract of the RenderScript API for the floating point data type. The full source code plus annotations can be found on the attached CD at:

ThirdYearProjectDemonstrationTwo/src/com/willsj/android/thirdyeardemo/floatOperationsRS.java
ThirdYearProjectDemonstrationTwo/src/com/willsj/android/thirdyeardemo/floatSplitArrays.java
ThirdYearProjectDemonstrationTwo/src/com/willsj/android/thirdyeardemo/rsprototypetwo.rs

Java Layer Code

```
public float[] floatAdd(float[] A, float[] B, RenderScript mRs,
ScriptC_rsprototypetwo mScript) {
	float[] D = new float[A.length];
	if (A.length > 1000) {
		float[] E = new float[A.length];
		floatSplitArrays floatSplit = new floatSplitArrays();
		E = floatSplit.addSplitArrays(A, B, mRs, mScript);
		return E;
	} else {
		D = messageHandler(A.length, mRs);
		saveToMemory(A, B, mRs, mScript);
```

```
            mScript.invoke_floatAdd();
            sleep();
            return D;
      }
}

public float[] addSplitArrays(float[] A, float[] B, RenderScript mRs,
ScriptC_rsprototypetwo mScript) {
      final int timesSplit = (int) Math.floor((double) (A.length/1000));
      int remainder = A.length - (timesSplit * 1000);
      final float[] E = new float[1000];
      final float[] F = new float[1000];
      final float[] J = new float[remainder];
      final float[] K = new float[remainder];
      float[] G = new float[1000];
      final float[] H = new float[A.length];
      for (int i = 0; i < timesSplit; i++) {
            for (int j = 0; j < 1000; j++) {
                  E[j] = A[(i * 1000) + j];
                  F[j] = B[(i * 1000) + j];
                  }
            G = floatAdd(E, F, mRs, mScript);
            for(int k = 0; k < G.length; k++) {
                  H[(i * 1000) + k] = G[k];
                  }
            }
      if (remainder == 0) {
            return H;
      } else {
            mRs.setMessageHandler(new RenderScript.RSMessageHandler(){
                  @Override
                  public void run() {
                        Log.d(" ", String.valueOf(this.mID)+ " " + mData + " "
+ mLength);
                        H[(timesSplit * 1000) + mLength] = mID;
                  }
            });
            for (int l = 0; l < remainder; l++) {
                  J[l] = A[(timesSplit * 1000) + l];
                  K[l] = B[(timesSplit * 1000) + l];
                  }
            saveToMemory(J, K, mRs, mScript);
            mScript.invoke_floatAdd();
            return H;
            }
}
```

RenderScript Layer Code

```
typedef struct __attribute__((packed, aligned(4))) floatValues {
      float A;
      float B;
      float C;
} floatValues_t;
floatValues_t *floatvalues;

void floatAdd() {
      floatValues_t *pfloatValues = floatvalues;
      int index = rsAllocationGetDimX(rsGetAllocation(floatvalues));
```

```
	for (int i = 0; i < index; i++) {
		pfloatValues->C = pfloatValues->A + pfloatValues->B;
		rsSendToClientBlocking(pfloatValues->C, (const void *)
&pfloatValues->C, i);
		pfloatValues++;
	}
}
```

Appendix 13: RenderScript Benchmark Test

The following code is an extract from the RenderScript benchmark test, the full source code plus annotations can be found on the attached CD at:

ThirdYearProjectDemonstrationTwo/src/com/willsj/android/RenderScriptBuiltBenchmark.java
ThirdYearProjectDemonstrationTwo/src/com/willsj/android/intOperationsRs/java
ThirdYearProjectDemonstrationTwo/src/com/willsj/android/rsprototype.rs

The XML file for the layout for this test can be found at:

ThirdYearProjectDemonstrationTwo/res/layout/renderscriptbuiltbenchmark.xml

A screenshot of this code in action can be found in Appendix 15.

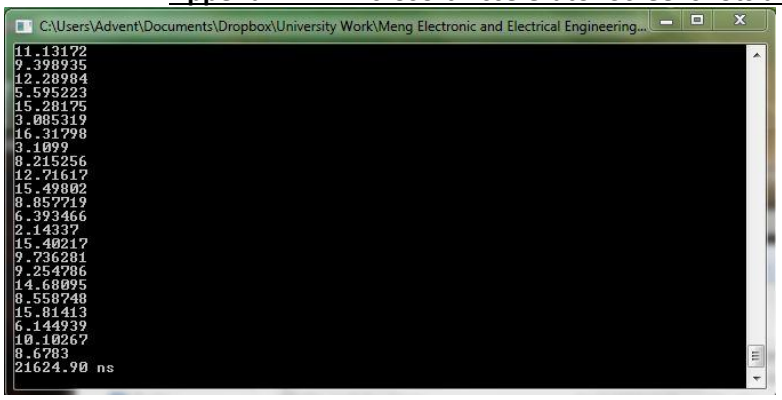
Java Layer Code

```
public int intAddBenchmark(int[] A, int[] B, RenderScript mRs,
ScriptC_rsprototypetwo mScript) {
	int[] t = messageHandler(1, mRs);
	saveToMemory(A, B, mRs, mScript);
	mScript.invoke_intAddBenchmark();
	TimerFunctions wait = new TimerFunctions();
	wait.timerInMilliseconds(10);
	return t[0];
}
```

RenderScript Layer Code

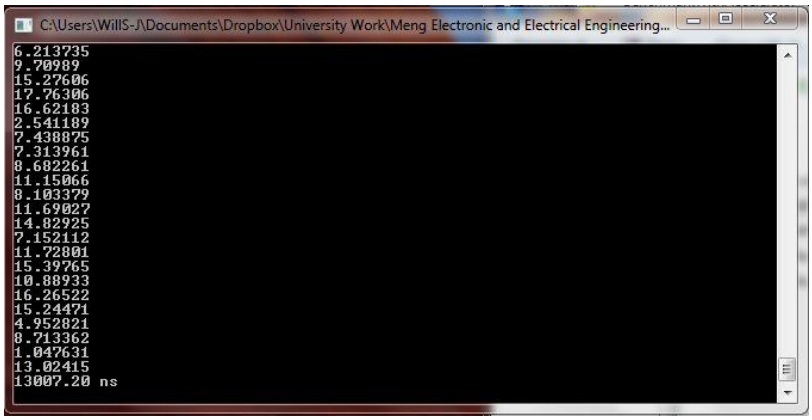
```
void intAddBenchmark() {
	Values_t *pValues = values;
	int index = rsAllocationGetDimX(rsGetAllocation(values));
	int64_t startTime = rsUptimeNanos();
	for (int i = 0; i < index; i++) {
		pValues->C = pValues->A + pValues->B;
		/*Move onto the next struct*/
		pValues++;
	}
	int64_t endTime = rsUptimeNanos();
	int timeTaken = endTime – startTime;
	rsDebug(“Time taken = “ ,timeTaken);
	rsSendToClientBlocking(timeTaken, (const void *) 0, 0);
}
```

Appendix 14: Microsoft Accelerator Screenshots and GPU Specifications



This is a screenshot of the program running on a desktop with an NVidia GTX 460 GPU.

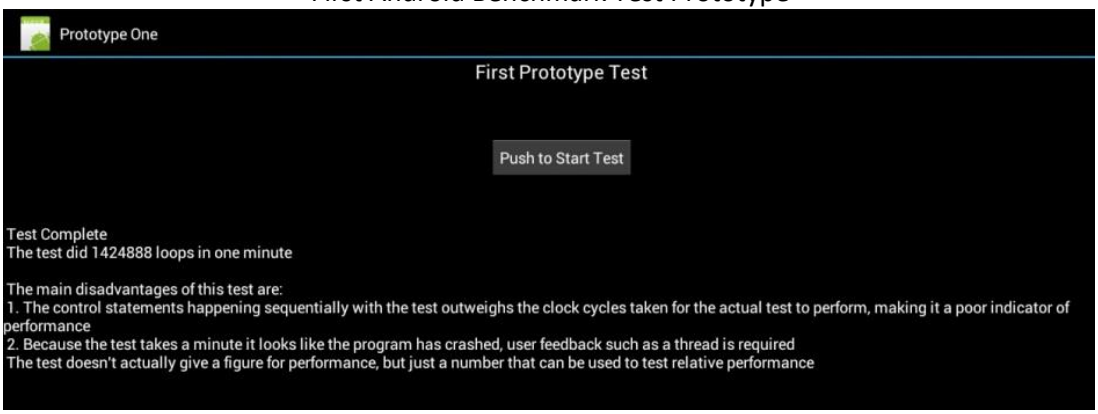
This is a screenshot of the program running on a laptop with an Nvidia GT 520MX GPU.



GPU	Nvidia GTX 460[30]	Nvidia GeForce GT 520MX[31]
CUDA Cores	336	48
Graphics Clock (MHz)	675	900
Processor Clock (MHz)	1350	1800

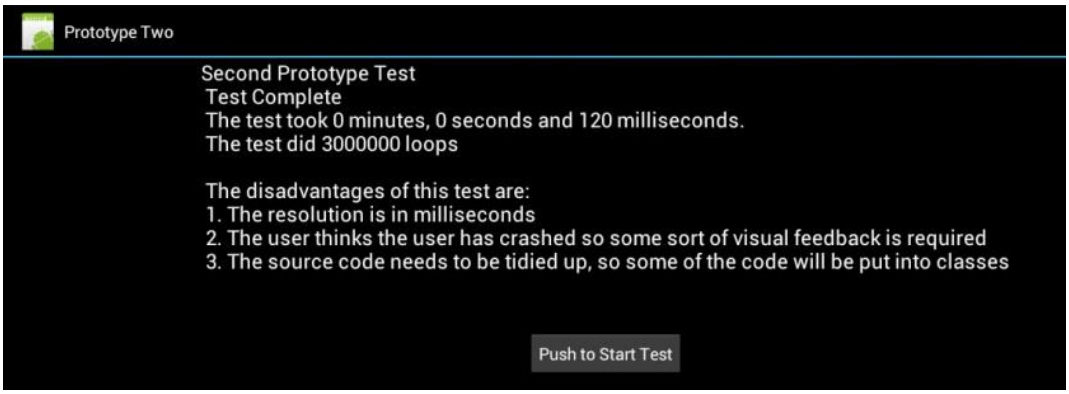
Appendix 15: Test Application Screenshots

First Android Benchmark Test Prototype

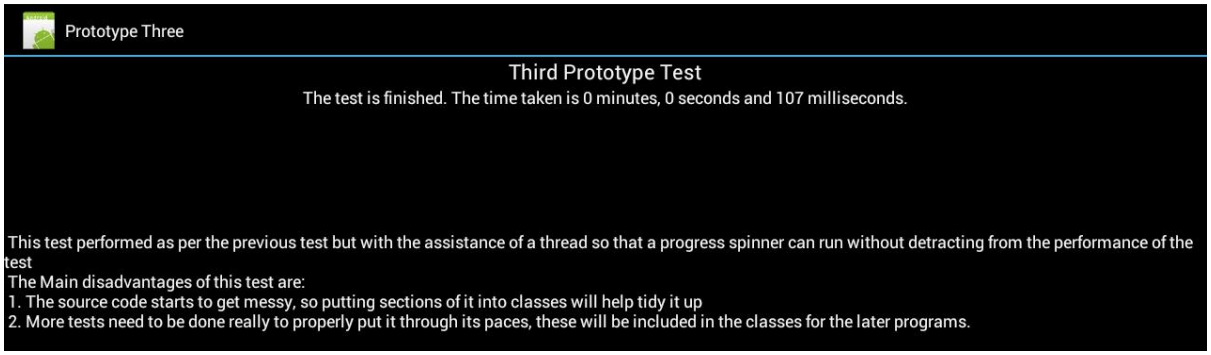


[30] http://www.nvidia.co.uk/object/product-geforce-gtx-460-uk.html
[31] http://www.notebookcheck.net/NVIDIA-GeForce-GT-520MX.54717.0.html

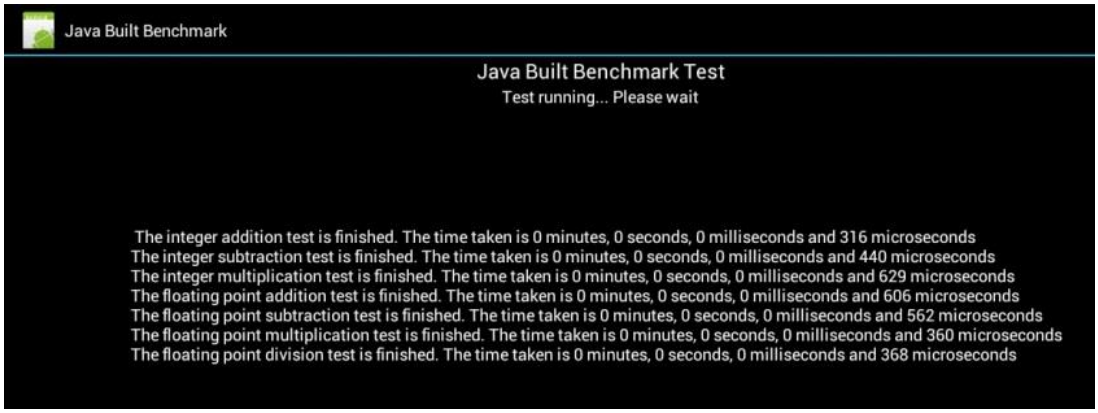
Second Android Benchmark Test Prototype



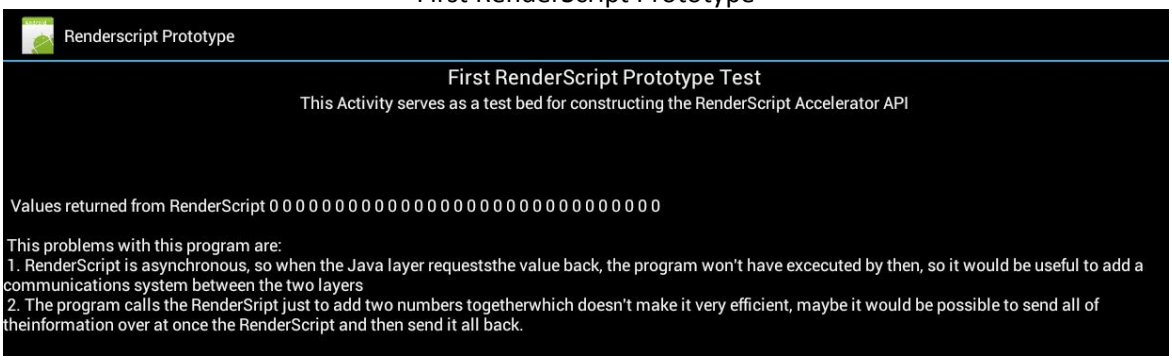
Third Android Benchmark Test Prototype



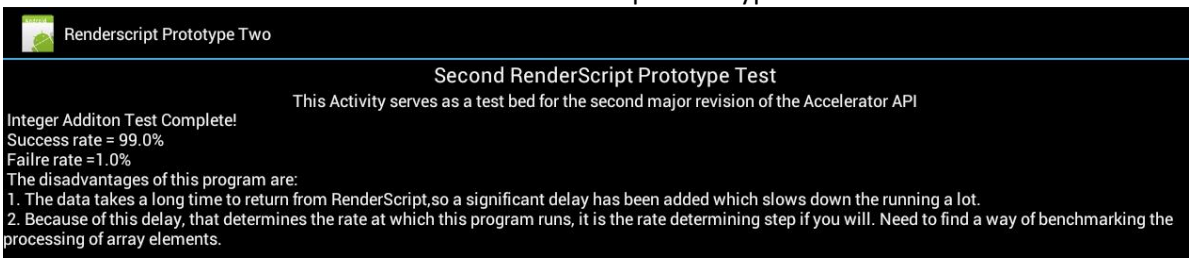
Finished Java Benchmark Test



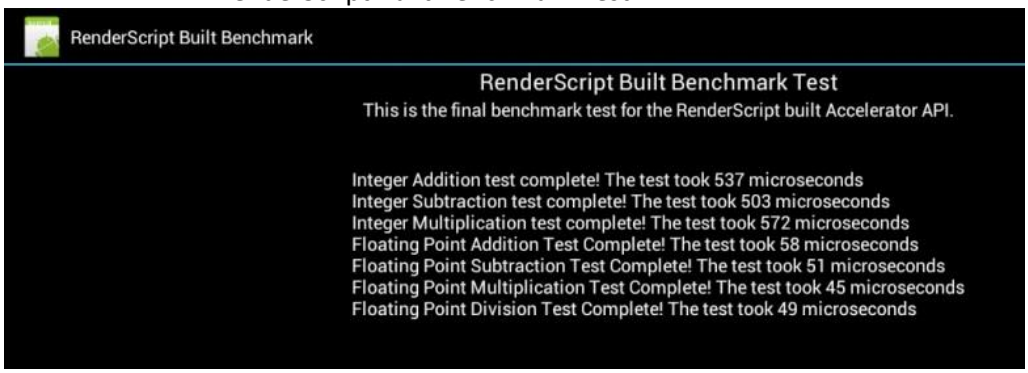
First RenderScript Prototype



Second RenderScript Prototype



RenderScript Built Benchmark Test



Appendix 16: Results of Benchmark Tests

Below are the results of a series of benchmark tests using the test application developed during the course of this project.

Java test

Execution time (microseconds)

Number of Array Elements	Int Addition	Int Subtraction	Int Multiplication	FP Addition	FP Subtraction	FP Multiplication	FP Division
1000	101	76	79	108	106	89	107
2000	155	107	137	120	117	118	119
3000	165	149	155	173	159	165	172
4000	189	185	192	217	214	247	208
5000	230	222	221	271	253	243	251
6000	295	250	256	296	305	264	297
7000	294	342	342	322	321	381	368
8000	320	305	375	370	372	376	336
9000	352	319	342	402	395	387	409

10000	349	356	349	404	430	434	402
20000		649	656	687	729		804
30000	1061	1017	1123	1829	1196	1162	1099
40000	1245	1756	1322	1439	1430	1467	1560
50000	1555	1452	1612	1249	1071	1108	1041
60000	2089	1171	1154	2305	2652	2264	2691
70000	2428	2552	2460	3411	3111	3025	3081
80000	3482	3415	3832	3505	3381	3093	3530
90000	3342	2438	2359	3236	3791	3199	3057
100000	3130	3137	3067	3200	3163	3352	3592

RenderScript Test

Execution time (microseconds)

Number of Array Elements	Int Addition	Int Subtraction	Int Multiplication	FP Addition	FP Subtraction	FP Multiplication	FP Division
1000	39	46	43	48	43	59	42
2000	109	85	88	87	91	90	87
3000	160	154	146	177	148	134	137
4000	182	181	192	184	180	184	179
5000	237	218	283	257	235	232	264
6000	273	277	276	282	291	278	275
7000	315	332	326	333	337	326	329
8000	385	389	388	431	391	392	380
9000	435	412	443	451	434	447	444
10000	484	498	492	488	486	489	492
20000	1002	1050	1151	1007	972	963	947
30000	1783	1634	1617	1624	1736	1507	1722
40000	2061	2073	1967	2000	1998	1967	2269
50000	2603	2662	2861	2851	2537	2555	2798
60000	3162	3016	3569	3075	3084	3138	3468
70000	3901	3616	4305	3954	3666	3718	4257
80000	4279	4111	4051	4885	4167	4592	4234
90000	5071	5338	4787	5338	5274	4959	5185
100000	5346	5996	5157	5296	5337	5363	6004

Appendix 17: Instructions on How to Use the API

Because of the varied source file types (Java source file and C source file), and the programs dependency on generated Java code, it was not plausible to package everything into a Java package to be used with a simple import statement. So in order to use this API it is necessary to import each if the source files yourself and in the right order. The source files that constitute the API are:

IntOperationsRS.java
IntSplitArrays.java
FloatOperationsRS.java
floatSplitArrays.java
rsprototypetwo.rs

The first step to including these files into a program is to import **rsprototypetwo.rs** into an application, change the package name (which is the second line of code) to that of the current Android application. Save it, upon saving the code should be generated so that it can be interacted with by Java code. The next step is to import the other Java classes and changing their package names appropriately, which can be done on the very first line of code on the class files. Then to use them input the following lines of code in your Activity class:

```
RenderScript mRs = RenderScript.create(this);
            ScriptC_rsprototypetwo mScript = new ScriptC_rsprototypetwo(mRs,
getResources(), R.raw.rsprototypetwo);
```

And finally import one of these two lines of code:

```
intOperationsRS intOp = new intOperationsRS();
floatOperationsRS floatOp = new floatOperationsRS();
```

This will allow the use of the functions in the API, then to call them as shown here:

```
C = intOp.intAdd(A, B, mRs, mScript);
D = floatOp.floatAdd(A, B, mRs, mScript);
```

Where A, B are the users input arrays, and the other two arguments are the other objects created in the above lines of code. The functions available are:
intAdd()
intSubtract()
intMultiply()
intAddBenchmark()
intSubtractBenchmark()
intMultiplyBenchmark()
floatAdd()
floatSubtract()
floatMultiply()
floatDivide()
floatAddBenchmark()
floatSubtractBenchmark()
floatMultiplyBenchmark()
floatDivideBenchmark()
All of which require the same input arguments, all of the functions except the “Benchmark” functions return an array that contains the result of that particular operation for that particular data type. The “Benchmark” functions return the execution time in microseconds as an integer variable. Although using this API is not as intuitive as desired, it still is relatively easy to use, the user isn’t exposed to anything that they shouldn’t be familiar with when it comes to Java programming and the extra lines of code required are comparable to that of Microsoft Accelerator. But in fact it is easier to import the files into an Android application than Microsoft Accelerator is to a C# and most certainly easier than it is importing to a C++ application.

Appendix 18: Original Project Proposal

The Aim of this project is to find a way to use techniques such as data parallel program to utilise the Graphics Processing Unit (GPU) of an Nvidia Tegra 2 powered Android Tablet PC.
Over the last few decades, the central processing units of computers have sped up at an alarming pace, but over the last few years the rate of increase in processor performance has decreased. This is called for a new design in processor technology, namely the multicore processor. Adding more processor cores to a computer though does present a challenge to the software developers working on computer systems. Now any program that needs to be run on a multicore system has to tell the

computer which processor to do what calculation, which means a lot of the time a program may be written for one system and then has to be rewritten for another system.
The same is true for the current Android powered smart phones and tablet PCs. In the last year dual core processors are being added to Android tablets namely the Nvidia Tegra 2 processor. This consists of two processor cores and 8 GPU cores, but the problem is that Android developers have to write specifically for this processor, and will have to do the same when the next model makes a debut.
In response to this problem, Microsoft Research developed Microsoft Accelerator, this is a library which allows a user to optimise their program for a multicore system, or a system with a powerful GPU without having to completely rewrite their code. This library can either be used as a library of header files for a C++ application, or a file of .dll file which is inserted into a projects debug bin in C#. And this is the type of solution I wish to develop for Android powered devices.
In order to tackle this challenge, I intend to learn how to use Microsoft Accelerator in order to ascertain how it works on a low level when it comes to utilising the DirectX features built into most modern GPUs. Then I hope to do some extended research on how to program for Android, particularly on a low level and specifically trying to utilise the GPU. I hope to find that in the Android software development kit, there is a library or libraries that are meant specifically for utilising the GPU, but the chances are that they are meant for graphical purposes. Such as generating a user interface or for graphical effects in Android market games, especially for the most recent models of tablet PCs running on a Honeycomb Operating System. It may be that a similar library to these can be used but adapted so that the GPU can used to perform mathematical operation or process. Such as a matrix multiplication. Once I have got this far, it will be time to implement some form of library that should resemble Microsoft Accelerator which can be readily embedded into an Android application in order to optimise it quickly. Once this library or section of a library is complete, I will need to test it, and in order to do this I plan to make a test application to demonstrate how it works. The type of app I want to write will either be a mathematical operation such as a simple matrix multiplication, or time permitting something considerably more ambitious such as signal processing and/or simulation.
This project should hopefully be tackled using the Android Software Development Kit (SDK) either using Java, which most applications are developed in, or alternatively using the Native Development Kit (NDK). The NDK requires the use to program in C++, but is more suitable for writing applications that don't require much memory when running, such as mathematical problems like simulations or signal processing.
Because Android is open source, then all of the software required is free to download and readily available and doesn't require a developers licence as the Apple IOS software does. So hopefully the budget for the project shall be minimal, although some reading material may be required as part of the research.

Appendix 19: Projected Project Timeline

Semester 1 Plan

Activities	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11
Initial Research											
Hands on time with Microsoft Accelerator											
Research into Android GPU Libraries											
Isolate any useful classes that may be of use and research further											
Learn to develop software in Android (especially with native code)											
Extended Research on Data Parallel Programming etc.											
Start Writing a Basic Program using Data Parallel Programming											
Report Writing											

Semester 2 Plan

Activities	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9
Start writing the library									
Write Benchmark programs to test library									
Write some demonstration applications									
Final Testing and Debugging									
Report Writing									

Bibliography

Published Resources

Publication Title	Publisher	Publication Author(s)	Page(s) Referenced	Data Published	ISBN Number
CUDA: By Example (Kindle edition)	Addison Wesley	Jason Sanders and Edward Kandrot	(Kindle Edition) Location 351 of 4339	July 2010	978-0-13-138768-3
Sam's Teach Yourself Java in 24 Hours (Kindle Edition)	Sam's	Rogers Cadenhead	(Kindle Edition) Location 2089 of 9011 Location 308 of 9011	October 2010	978-0-672-33076-8
Android 3.0 Application Development Cookbook (Kindle Edition)	Packt Publishing	Kyle Merrifield Mew	(Kindle Edition) Location 417 of 4119 (Page 5 of 261) Location 793 of 4119 (Page 33 of 261)	July 2011	978-1-849512-94-7
Android NDK: Beginners Guide (Kindle Edition)	Packt Publishing	Sylvian Ratabouil	(Kindle Edition) Location 1209 of 6290, Location 1205 of 6290, Location 1282 of 6290, Location 1242 of 6290	January 2012	978-1-849691-52-9
Pro Android Apps Performance Optimisation (Kindle Edition)	APress	Hervé Guihot	(Kindle Edition) Location 2614 of 5885, Location 5047 of 5885	January 2012	978-1-4302-4000-6
Java in a Nutshell, Fifth Edition	O'Reilly	David Flanagan	Page(s) 501, 763	March 2005	978-0-596-00773-7
C in a Nutshell	O' Reilly	Peter Prinz & Tony Crawford	Page 145	December 2005	978-0-596-00697-6

Internet Resources

Website Link (in order of which they are referred to)	Webpage Description/Information Accessed	Date Accessed
http://www.intel.com/content/www/us/en/silicon-innovations/moores-law-technology.html	Intel's official website that contains information on Moore's Law and the implications behind it.	21/12/11
http://research.microsoft.com/en-us/projects/accelerator/	The webpage that contains some brief information on Microsoft Accelerator, links to download the Accelerator library and documentation.	05/10/11
http://www.nvidia.com/object/tegra-3-processor.html	Information on the NVidia Tegra 3 processor, including a technical specification	10/02/12
http://www.nvidia.co.uk/object/tegra-2.html	Information on the NVidia Tegra 2 processor and a technical specification	17/10/11
http://research.microsoft.com/en-us/projects/accelerator/accelerator_intro.docx	Word document introducing Microsoft Accelerator and a brief development guide.	05/10/11
http://developer.android.com/guide/topics/renderscript/index.html	Android developer's website introducing RenderScript.	21/02/12
http://blogs.arm.com/multimedia/617-gpu-computing-in-android-with-arm-mali-t604-renderscript-compute-you-can/	Software Developers blog describing how RenderScript works and other information on Android GPGPU programming.	24/02/12
http://developer.android.com/guide/topics/renderscript/index.html#struct	Information on how to use structures in RenderScript.	03/03/12
http://developer.nvidia.com/tegra-android-development-pack	Webpage that provides a link to	07/10/11

	download the NVidia Android Development kit	
http://msdn.microsoft.com/en-us/library/d9x1s805(v=vs.71).aspx	Information on how a pragma statement works	01/03/12
http://mobile.tutsplus.com/tutorials/android/getting-started-with-renderscript-on-android/	A tutorial on how to develop using RenderScript and how some of the constituent components work (in the case of this reference the root() and init() functions)/	15/02/12
http://developer.android.com/guide/topics/renderscript/index.html#func	Information on how to create RenderScript functions and interact with them using Java	01/03/12
http://www.nvidia.co.uk/object/product-geforce-gtx-460-uk.html	Technical specification of the NVidia GTX 460 consumer desktop GPU	10/01/12
http://www.notebookcheck.net/NVIDIA-GeForce-GT-520MX.54717.0.html	Technical specification of the NVidia GT 520 MX consumer notebook GPU.	10/01/12