# Parallelization strategy based on RenderScript reductions

Francisco Rafael Suárez Ruiz, Jesús Antonio Álvarez Cedillo and Alfonso Fernández Vázquez

Instituto Politécnico Nacional

franciscorafaelsr@gmail.com, jaalvarez@ipn.mx, alfonso.fernandez.v@gmail.com

**Abstract.** RenderScript is a set of tools designed by Google to support parallel processing on mobile devices with Android. This tools were designed to run on different processing components such as Central Processing Units (CPU), Digital Signal Processors (DSP) and Graphics Processing Units (GPU) and it allows portability between mobile electronics devices such as Tablets and Smartphones. RenderScript has a runtime that decides where and how to execute commands list in parallel, it differs in coding and abstraction problem from others platforms used as Open Computing Language (OpenCL) and Compute Unified Device Architecture (CUDA). However, in this new parallelization paradigm kernel is not optimized for a specific architecture. There are not clear strategies for reduction algorithms implementation. For this reason this paper proposes several strategies for reduction algorithms implementation between vectors using RenderScript.

## 1 Introduction

A vector processor is a processor that can compute an integer value in an instruction, usually a vector instruction is equivalent to execution of a complete instruction at a loop, where each iteration works on each individual component of the vector. Vector memory operations are better than scalar operations because:

1. Each result is independent.
2. A single vector instruction replaces many scalar instructions.
3. They require a memory access pattern with a fixed access (adjacent).
4. In the problem is avoided a jumping in control loop.
5. Operations running more faster.

In this paper we explain two strategies for running a binary vector reduction for mobile devices (Smartphones and Tablets) using RenderScript.

### 1.1 RenderScript

Some applications can be developed using personal purpose processing such as computer vision and image processing. The applications that use RenderScript running inside the virtual machine Android, so Java application programming

interface is used to management resources and regulates the control life of programs of the kernel [4].

RenderScript represents a set of tools and a high level language for parallel computation intensive processing on Android devices, combined with sequential programming generates profits and optimize processes running in the execution model.

RenderScript makes work distribution across different threads and it optimizes use of all available processors on the device, it focus on the high performance applications on GPU, DSP, CPU and multiple cores. The RenderScript algorithms are perfectly balanced when it are processing, so any general-purpose application may be made and implemented widely using the maximal of resources on mobile device.

RenderScript Programming is based on next stages:

1. High performance kernels, all source code is written with a source code supported with C99 language.
2. An interface application (API) developed with Java,

### 1.2 FilterScript

FilterScript is a subset of RenderScript which contains restrictions to work in a variety of processors. In FilterScript the pointers are not allowed, so you can not read the memory directly and have to use API access functions of RenderScript[1].

### 1.3 Execution Model

The mobile device architecture has major differences with other traditional parallel systems such as a desktop system or a system for high availability.

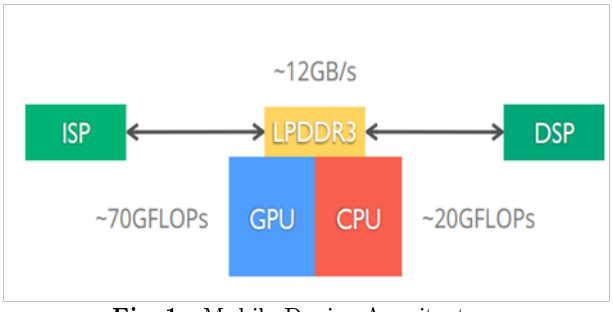


**Fig. 1.** Mobile Device Arquitecture

Communication between the CPU and the GPU is done through the DDR, LPDDR or MDDR memory (it's a type of synchronous DRAM special double precision for mobile devices) at 12Gb/s and an instruction set supports up to 70 GFLOPS on the GPU and up to 20 GFLOPS on the CPU. See Figure 1.

RenderScript was developed on the Low Level Virtual Manchine paradigm (LLVM). The execution model of an application in FilterScript or RenderScript shown in Figure 2.

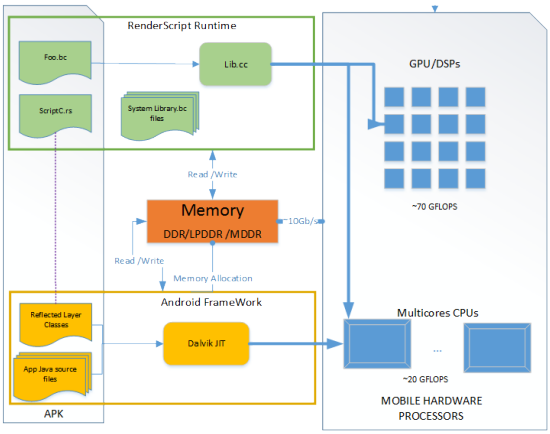


**Fig. 2.** Execution Model RenderScript based in [2]

This operation is as follows:

1. The execution program list is made in a language based on c99, is compiled with an intermediate format.
   (a) Encapsulated source code or KERNEL, is processed and creates a binary list not architecture dependent.
   (b) A special routine to compile binary list for one or more processors.
   (c) Special encapsulates are integrated according to the programmer needs for some common operations.
2. JAVA classes are created automatically by Development Kit.
3. Administration and enforcement resources are controlled by Java application interfaces.
4. They are integrate with the virtual machine, where are adding the executable binary and checklist as Java application.

## 2 Related work

In the literature, there are some proposals of how to code in RenderScript using other platforms of heterogeneous computing, however, not directly pose a programming strategy where are take advantage of all RenderScript tools, is the

case of [1], where the authors propose to use Heterogeneous Image Processing Acceleration (HIPA) to generate a image processing Kernel and to adapting to the RenderScript structure. The second proposal in [3], it shown the coding of a software implementation for translate OpenCl commands from to RenderScript.

After of the review of previous works that use applications and RenderScript code generation or strategies, are show that is not possible automatically export all the characteristics of a heterogeneous platforms and RenderScript computing resources are not optimally utilized; in [4] shown an alternative to programming high performance applications on android devices and show the Android architecture operation in APARAPI GPU, the authors focused their research on a performance study of the CPU and GPU using OpenCL on a GPU. the experiments were carried out in a Nexus 10 A15 CPU Dual-core, Quad core Mali-T604 GPU, and 2GB RAM.

Some algorithms require from an input data to define the number of elements of array reduced output, examples of these operations are: average cumulative sum, maximum, minimum, etc. A such operations within the parallel processing are called reduction operations.

There is a related work to the performance tests introduced by [5], they shown a the code that work on a HTC Desire smartphone. Dalvik Java source code was tested, the test consisted of implementation a set of twelve programs for Android platform using native code and Java, to evaluate the algorithm they found that only three applications run faster on Dalvik Java code and found performance issues.

In FilterScript and RenderScript, reduction operations used is not easy because a kernel implementation requires that length of input data should be equal to output data. Another important limitation is that is not possible to control over the numbers of threads that were invoked and this can only be changed by varying length characteristic of data input and output. Another paper presented by [7] performed additional performance tests where are comparing Android and Windows Mobile vs Java ME. They demonstrated where performance is better in Android. By other side, in [6] demonstrated a work where the native code is most efficient to develop high performance applications.

Analyzing existing studies have highlighted the need to attract new programming strategies for mobile parallel computing platform that supports general purpose programming; there are very few studies that show use of FilterScript or RenderScript, much less computation strategies vector on a platform of limited storing on Android Smartphones or Tablets, for this reason in this paper we focus to show two strategies for computing vector reductions in RenderScript implementations.

```
intsize;
void root(int32_t *v_out, uint32_t x,uint32_t y) {
if (x < divisor)
v_out[y] += v_out[sizey];
}
```

**Listing 1.1.** RenderScript kernel source code

```
intsize;
rs_allocation input;
float __attribute__((kernel)) root(uint32_t x) {
if(x < size)
return rsGetElementAt_float(input,x)+rsGetElementAt_float(
    input,size - x) ;

else
return 0;

}
```

**Listing 1.2.** Filterscript kernel source code

## 3 Solution proposal

RenderScript Lists exemplify programming strategies designed to use application interface for versions Android 4.1 in onwards. To solve reduction operations in RenderScript we have two possible strategies, these are exemplified with the cumulative sum of a vector of floating values on RenderScript and FilterScript version 2.2

### 3.1 First Strategy: RenderScript

The first strategy is to limit memory locations on threads running through identification number of thread.

Advantages:

1. Were performed on the same memory locations.
2. Ease of abstraction
3. Kernel in RenderScript be called recursively until the stop condition.

Disadvantages:

1. Threads are wasted.

The Kernel in RenderScript shown in Listing 1.2.

The Kernel in FilterScript shown in Listing 1.3.

```
1 float __attribute__((kernel)) root(const float2 v_in,uint32_t
    x) {
2 return v_in[0]+v_in[1];
```

**Listing 1.3.** Behavior of FilterScript and RenderScript kernel

### 3.2 Second Strategy: RenderScript and FilterScript

The second strategy is to use data structures provided by FilterScript and RenderScript. Is allow that input and output structures are the same size but can be of different type.

Advantages:

1. Invoked all threads are used.
2. Entire allocated memory is occupied.

Disadvantages:

1. Is necessary to change memory size for each reduction.
2. Algorithm must be adapted to the data structures.

In this case Listing 1.3 shows the behavior of FilterScript and RenderScript kernel

## 4 Results

With strategies proposed in the previous section, Mercator Series (equation 1) was implemented to calculate natural logarithm. This example was chosen because computation powers in the GPU have high computational complexity.

$$ln(1+x) = \sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n} x^n \tag{1}$$

This strategies was proved on a Tablet device with follow specifications:

| | |
|---|---|
| Device: | Tablet Acer Iconia A1 |
| RAM | Memory: 1GB RAM DDR3 |
| Internal | Memory: 16 GB |
| Chipset: | MediaTek MT8125 |
| CPU: | quad-core 1.2GHz |
| OS: | Android OS, v 4.2 Jelly Bean |
| GPU: | PowerVR SGX544MP3 |

Listing 1.4 shows the Kernel where is calculating the summation values.

For design of this kernel is used the second strategy proposed for implementation. Table 1 shows the average results.

It shows E1 as Strategy 1 and E2 as Strategy 2.

```
float2 __attribute__((kernel)) root(uint32_t x) {
float2 result;
int id = x+1;
result[0] = alter(id)*(pown(n,id)/id);
id = totalx;
result[1] = alter(id)*(pown(n,id)/id);
return result;
```

**Listing 1.4.** Mercator Kernel

**Table 1.** Average time results of running.

| DATA | Dalvik | E1 | E2 |
|---|---|---|---|
| **10** | 7.106161038 | 9 | 8 |
| **100** | 50.49752469 | 45 | 30 |
| **1000** | 2550 | 200 | 307 |
| **100000** | 6502500 | 20000 | 10000 |
| **1000000** | 4.22825E+13 | 15000 | 8000 |
| **1000000** | 1.78781E+27 | 12000 | 11000 |

Figure 4 shows time results, Figure 3 shows values obtained in the approximation obtained by Mercator series.

Energy consumption on device maintains a direct relationship with use and utilization rate of both CPU and GPU.

Table 2 shows resource usage percentage on device, table shows use of a device with a single core and a PowerVR GPU.

To calculate application consumption, when it is direct current (DC) electric power output at a certain moment by the product of the potential difference and

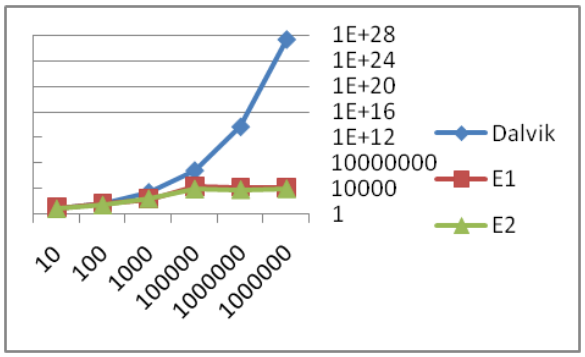


**Fig. 3.** Obtained values by Mercator series

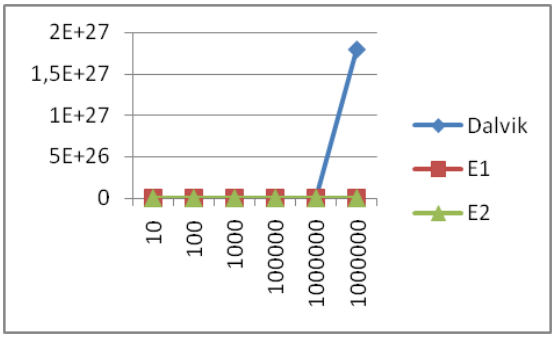


**Fig. 4.** Approximation obtained by Mercator series

**Table 2.** Resource usage percentage on device.

| | Dalvik | | E1 | | E2 | |
|---|---|---|---|---|---|---|
| | CPU | GPU | CPU | GPU | CPU | GPU |
| **10** | 40 | 2 | 10 | 40 | 22 | 42 |
| **100** | 60 | 4 | 12 | 50 | 24 | 45 |
| **1000** | 98 | 4 | 12 | 52 | 23 | 50 |
| **100000** | 100 | 4 | 15 | 55 | 25 | 48 |
| **1000000** | 100 | 4 | 15 | 55 | 27 | 50 |
| **1000000** | 100 | 4 | 16 | 70 | 28 | 60 |

the intensity of current passing through the device. For this reason the power is proportional to the current and voltage.

Equation 2 shows the corresponding expression

$$P = \frac{dw}{dt} = \frac{dw}{dq}.\frac{dq}{dt} = V.I \tag{2}$$

Where I is the instantaneous value of the current and V is the instantaneous voltage value.

If I is expressed in amperes and V in volts, P will be expressed in watts (W). The same definition applies when considering average values for I, V and P.

When the device is a resistor of value R or can calculate the equivalent resistance of the device, power can also be calculated using Equation 3

$$P = R.I^2 = \frac{V^2}{R} \tag{3}$$

DC voltage measured in normal operation is 4.2 Volts, When the calculation is performed at 100 percent the voltage in inductors (Vdrop) 14mVolts. Energy consumption by modifying Equation 3 is created equation 4

$$P = \frac{(Vbat)(Vdrop)}{R.} \tag{4}$$

Where Vbat is the nominal battery voltage 4.7 volts, Drop is voltage drop, where the voltage at 100% processor usage was 14 Volts in CPU and 12 GPU and mVolts. R represents the measurement resistance of 0.041 Ohms. Table 3 shows the final values of the power in Watts.

**Table 3.** Final values of the power in Watts

| | DALVIK | | E1 | | E2 |
|---|---|---|---|---|---|
| | cpu | gpu | cpu | gpu | cpu |
| **10** | 0,641951 | 0,027512 | 0,160488 | 0,550244 | 0,353073 |
| **100** | 0,962927 | 0,055024 | 0,192585 | 0,687805 | 0,385171 |
| **1000** | 1,57278 | 0,055024 | 0,192585 | 0,715317 | 0,369122 |
| **100000** | 1,604878 | 0,055024 | 0,240732 | 0,756585 | 0,40122 |
| **1000000** | 1,604878 | 0,055024 | 0,240732 | 0,756585 | 0,433317 |
| **1000000** | 1,604878 | 0,055024 | 0,25678 | 0,962927 | 0,449366 |

## 5 Conclusion

RenderScript, Google's parallel computing platform, is relatively new compared to other platforms and it continues with constant changes, which has led to its documentation is not quite complete. In RenderScript new strategies are needed for the implementation, since there is no control on number of threads to run like in others tools that have parallel computing platforms. . However, Google's platform has premise of being compatible with any number of devices and run the instruction list the most optimal manner in hardware available on your phone, also it can get excellent performance comparable to other platforms with the right programming strategies.

As reviewed in this paper, using RenderScript you can perform a lot of calculations in a short time on mobile devices, this opens the door for algorithms implementation that were previously inaccessible to these devices, for this reason scientific computing has a tool that can be of vital importance for algorithms validation.

## References

1. Membarth, R., Reiche, O., Hannig, F., Teich, J., Code generation for embedded heterogeneous architectures on android, in Design, Automation and Test in Europe Conference and Exhibition (DATE), pp.1, 6, 2428, March 2014.
2. Pavlov, S., Kazantsev, R., Getting Started with RenderScript on Intel® Architecture running the Android* OS. Internet: https://software.intel.com/en-us/android/articles/getting-started-with-renderscript-on-intel-architecture-running-the-android-os, Mar 25, 2014 [July 26, 2014].

3. ChengYan Yang, Yijui Wu, Liao, S., O2render: An OpenCL to RenderScript translator for porting across various GPU or CPU, in 2012 IEEE 10th Symposium on Embedded Systems for Realtime Multimedia (ESTIMedia), pp.67, 74, 1112, Oct. 2012.
4. HungShuen Chen, JrYuan Chiou, ChengYan Yang, Yijui Wu, Weichung Hwang, HaoChien Hung, ShihWei Liao, Design and implementation of high level compute on Android systems, in 2013 IEEE 11th Symposium on Embedded Systems for Realtime Multimedia (ESTIMedia), pp.96,104, 34, Oct. 2013.
5. C. M. Lin, J. H. Lin, C. R. Dow and C. M. Wen, Benchmark Dalvik and Native Code for Android System, in 2011 Second International Conference on Innovations in Bio inspired Computing and Applications (IBICA), Shenzhen, Guangdong, China, 2011.
6. Y. H. Lee, P. Chandrian and B. Li, Efficient Java Native Interface for Android Based Mobile Devices, in 2011 IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), Changsha, Hunan Provinc e, P. R. China, 2011.
7. T. M. Grønli, J. Hansen and G. Ghinea, Android vs Windows Mobile vs Java ME: a comparative study of mobile development environments, in Proceedings of the 3rd International Conference on PErvasive Technologies Related to Assistive Environments, Samos, Greece, 2010.