Assignment 3 - Phase 2: Using gem5 for estimating the performance of different application mappings onto different multi-core processors

Snorri Steffanson, Filippo Bernardi, *Master students, TU Eindhoven*

***Abstract*—**

***Index Terms*—ARM a9, ARM a15, gem5, multi threads**

I. Introduction

THIS paper shows performance evaluation of the same application, a jpeg encoder onto two different processors. Those two processors are composed differently. The first is a 3 ARM a9 core CPU. The second has one ARM a15 core and an ARM a9 core on it. The first step done it was to analyze the given application. For this task, Valgrind program has been used. The results is shown in figure 1

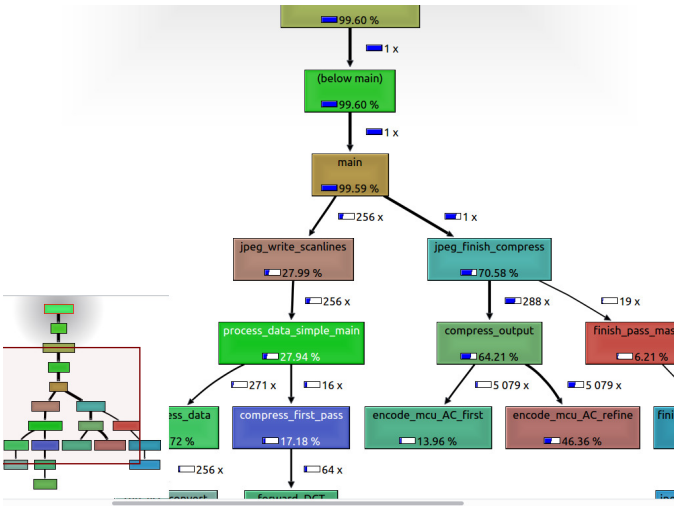

Fig. 1. Program calls in Valgrind

This results gives the hint of divide the code inside the CPUs onto different cores, after the main function. This paper is divided onto different section, the first part concern the Gem5 simulator and Pthread library. The other section explain and evaluate different application results.

January 27, 2017

II. Gem5 and Pthread library

A. Gem5 simulator

Gem5 is the simulator used for testing the effectiveness and the results of the changes made on the code. The metric for evaluate the performances are the "tick". Tick are the results that gem5 displayed after every simulation and represent the total execution time in picoseconds. The simulator can both work in Full System mode and Syscall Emulation mode $SEmode$. One one hand, in Full System mode the system it is much more accurate but it require also an higher amount of time for run the simulation, in this case it is possible to boot an entire OS from scratch. On the other hand, in $SEmode$ the system is less accurate but is much faster. In this paper it is used only the $SEmode$ because it is of interest evaluate an approximate results of the improved performance instead of an accurate analysis.

B. Pthread library

In order to divide the code into diferent threads, the Posix Thread library has been used. The library is composed mainly by this three functions:

- pthread_create

```
int pthread_create(pthread_t * thread,
        const pthread_attr_t * attr,
        void * (*start_routine)(void *),
        void *arg);
```

The function is needed to create new thread. The variable $thread$ returns the thread ID. $Attr$ it is possible to gives different attribute to the thread, set to NULL for default. $*start\_routine$ is the pointer to a function to be threaded. $*arg$ is the argument to pass to the function. In order to pass multiple arguments it is needed to create a struct and pass the struct as it has been done later on the report.

- pthread_join

```
int pthread_join(pthread_t th,
        void **thread_return);
```

In this function a thread is suspended till th terminates. $thread\_return$ return from the function. This function it is used for waiting the end of a thread.

- pthread_exit

```
void pthread_exit(void *retval);
```

This terminate a thread, where $*retval$ return the value of the function.

This description has been taken from
$http://www.yolinux.com/$

III. Function on thread

What as been done firstly is try to run on the two architecture without change any code, the results has been the following: 23714598000 number of tick for the 3 a9 cores and 18786430000. Now, the jpeg finish compression has been placed in another thread, for doing so the following tutorial has been used: http://timmurphy.org/2010/05/04/pthreads-in-c-a-minimal-working-example/

On one thred the jpeg_finish_compress there are the same results for 3 a9 23641697000 and 21533576000 for a15-a9

IV. New application

Due to the complexity of the jpeg compressing function we have decided to change to the same type of application but exploited in a simple manner. The new Program calls tree it can be seen in Figure 2

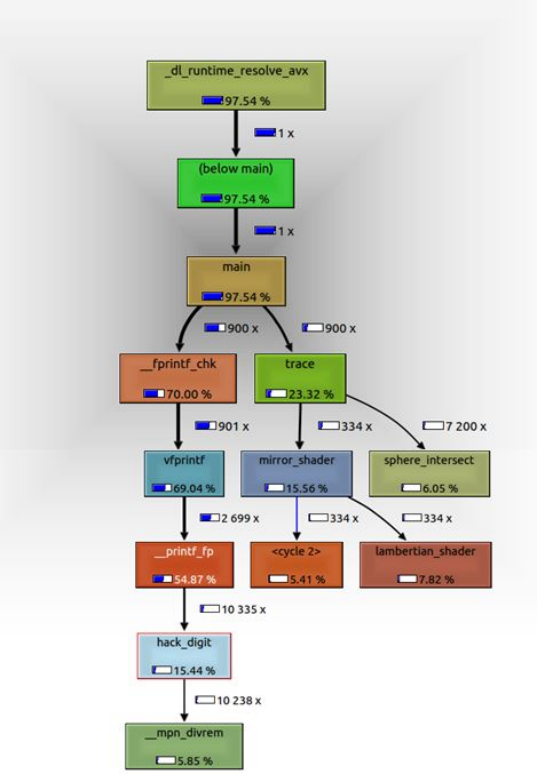

Fig. 2. New jpeg application calls in Valgrind

V. Pixels line on thread

The final application threaten in this report has been inspired directly from the application itself. Moreover, instead of using Valgrind for dividing the code, the authors have questioned what the Jpeg compression was doing. On behalf of this, the code was actually possible to divide among the all pixels line that the function was making as output. Therefore, the line of the picture where divided into different thread. Multiple discussion has been created and reported on this section. Firstly, a thread for each pixels line has been created. The idea was to created multiple thread and then let the compiler divide them among all the cores. In order to do so, the function:

However, because of gem5 was running in SE mode, it was not possible to create more thread that the cores available and then a different type of code has been created, tailored with the used CPU.

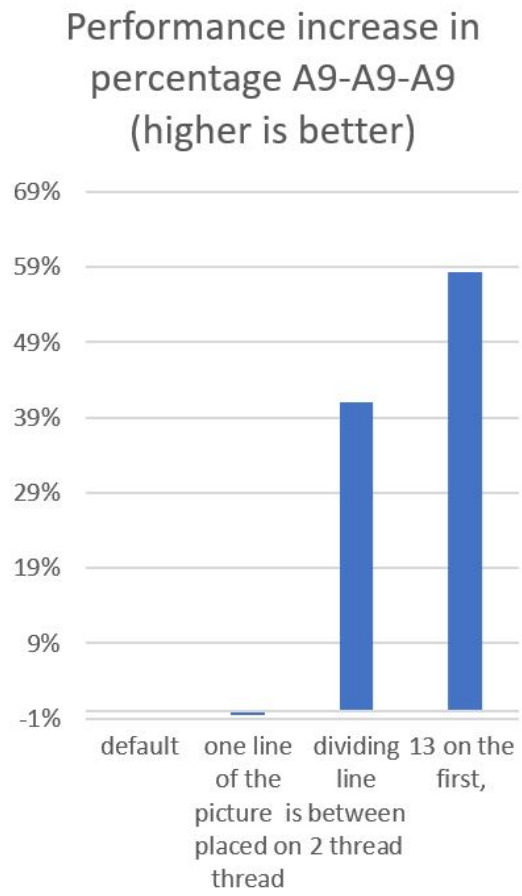

Fig. 3. Performances on a9 a9 a9 processor

VI. Appendix

A. scripts

```
#!/bin/sh
cd /home/$USER/srt-clean
sudo rm callgrind.out.*
valgrind --tool=callgrind ./srt
kcachegrind callgrind.out.*
cd /

#/bin/sh!
cd /home/$USER/srt-clean
make clean
make
cd /

#/bin/sh!
cd /home/$USER/srt-clean
```

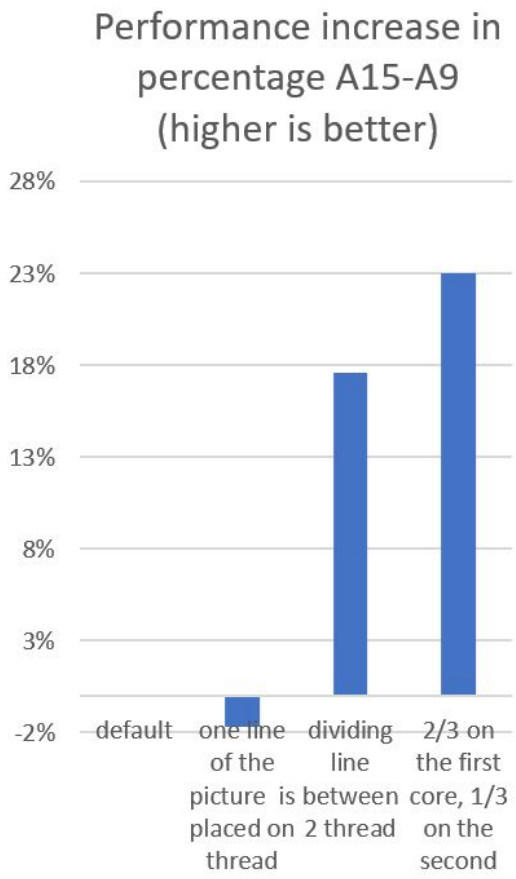

Fig. 4. Performances on a15 a9 processor

```
make clean
make -f Makefile.arm
/home/$USER/gem5/build/ARM/gem5.opt
/home/$USER/gem5/configs/example/arm-multic
        ore-A15-A9.py -c
        /home/$USER/srt-clean/srt
cd /

#/bin/sh!
cd /home/$USER/jpeg/jpeg-6a/
make clean
make -f Makefile.arm
/home/$USER/gem5/build/ARM/gem5.opt
         /home/$USER/gem5/configs/example/
        arm-multicore-A9-A9-A9.py -c
        /home/$USER/srt-clean/srt
cd /

#/bin/sh!
cd /home/$USER/srt-clean
make clean
make -f Makefile.arm
cd /
```

VII. Conclusion

Conclusion