CS-473:
System programming
for
Systems on Chip

Practical work 3

Profiling and memory distance

Version:
1.0

Contents

1 Introduction

1.1 Prerequisites

On moodle you find the file `mandelbrot_fxpt.zip`; it contains the complete source code of the fixed-point version of the Mandelbrot set. Download this file, and unzip it in the directory `programms/mandelbrot`, where you also have the floating point version of the last PW. Furthermore, you find the file `support.zip`, which contains support files for profiling and configuring the caches. Download this file and unzip it in the `support` directory (please overwrite `or32Print.c` with the version provided in the `support.zip`-file).

For this PW, please make sure that you have the latest version of the virtual-prototype on your GECKO-board.

1.2 Memories

In the theory session of today we have seen that SDRAM's have some particularities, like the burst mode and the CPU-Memory distance. In this practical work we are going to see which influences this can have on your program. Please note that the VGA-controller *fetches* each line in a burst of `nrOfPixelsPerLine/2` 32-bit words.

2 Exercises

2.1 Prerequisites

Familiarize yourself with the fixed-point version of the Mandelbrot set. Compile the program by:

```
cd mandelbrot/sandbox
../compile_fxpt.sh
```

Make sure that you do not have any errors. Then, download the `fractal_fxpt.mem` file to your virtual prototype and see the result.

Of course you can also use your own fixed-point version of the Mandelbrot set. Please make sure that you insert the commands required for profiling and memory distance in your `main_fxpt.c` and add `../../support/cache.c` and `../../support/profile.c` to your compile script.

2.2 Profiling

In the provided fixed point version of the mandelbrot algorithm, already two profiling counters are defined as:

```
setProfilingCounterMask(PROFILING_COUNTER_0, STALL_CYCLES_MASK |
                                             I_CACHE_NOP_INSERTION_MASK);
setProfilingCounterMask(PROFILING_COUNTER_1, BUS_IDLE_MASK);
```

The profiling counter 0 counts the stall cycles. To understand why also the `I_CACHE_NOP_INSERTION_MASK` is used, the behavior of the fetch-stage needs to be known. In case the fetch-stage has not yet a new instruction, it will not stall the CPU, but it will insert `l.nop` instructions. This is counted with the given mask.

The profiling is started, respectively stopped with the macros:

```
startProfiling();
stopProfiling();
```

After stopping the profiling, please wait at least 5 CPU-cycles, as the profiling module is pipe-lined.

The results of the profiling counters can than be shown with:

```
printProfilingTime( PROFILING_COUNTER_0 , "Stall time    " );
printProfilingTime( PROFILING_COUNTER_1 , "Bus idle time " );
printProfilingTime( RUN_TIME_COUNTER , "Runtime time  " );
```

There exists also another support function that prints the results in hexadecimal format:

```
printProfilingCycles( RUN_TIME_COUNTER , "Runtime cycles  " );
```

In this exercise we are going to add a two profiling counters.

Add and print out the number of:

- Instruction cache fetches.
- Instruction cache misses.

What can you observe?

2.3 Memory distance

Your virtual prototype has the ability to *emulate* a speed difference between the CPU and the external SDRAM. This emulation consists of *slowing down* the SDRAM-accesses. The macro used for this purpose is `setMemoryDistance( val )` that is defined in `profiling.h`. The value-range of `val` is 0 (the SDRAM is running at the same frequency as the CPU) up to 63 (one SDRAM clock cycle equals to 63 CPU-clock cycles).

In this exercise we are going to *play* with the memory distance. Up to this moment we executed the mandelbrot algorithm with a memory distance value 0.

Execute the mandelbrot algorithm with the memory-distance values 1,2,5, and 25.
What can you observe, and how can you explain it?

2.4 Fractal size

The size of the fractal is defined by:

```
const int SCREEN_WIDTH = 512;   //!< screen width
const int SCREEN_HEIGHT = 512;  //!< screen height
```

We are going to play with these sizes, please make sure that for the following experiments the memory-distance is 0.

Execute the mandelbrot algorithm for the resolutions 256x256, 128x128, and 300x300.
What can you observe? Is my software bogus?

We are now changing the framebuffer by replacing:

```
rgb565 frameBuffer[SCREEN_WIDTH * SCREEN_HEIGHT];
```

with:

```
rgb565 *frameBuffer= (rgb565 *) 0x10000;
```

Execute the mandelbrot algorithm for the resolutions 256x256, 128x128, and 300x300.
What can you observe? Why is there a difference with the previous experiment.

For the resolutions 256x256 and 128x128, execute them with the memory distance values 1,2,5, and 25.