Modern Computer Architecture Lab Report

Mick van Gelderen 4091566

Arian Stolwijk 4001079

November 2013

1 The kernel function

The concept of this lab is to take a piece of code that is executed frequently and run it on a reconfigurable co-processor. The extracted piece of code is called the kernel and the device is in our case the ρ-VEX: A reconfigurable and extensible VLIW processor.

The kernel needs to be compiled for the ρ-VEX so that we can inject the result, the bytecode, in to the co-processor. The rest of the code runs as usual on a regular processor.

2 Communication

We have access to several (but not enough during the lab) FPGAs which are configured as Microblaze processors and run Linux.

The ρ-VEX is a configured as a co-processor that can be controlled using a number of memory mapped files. The is a file for writing the instruction memory, one for reading and writing the data memory, one for reading the status and one for writing control commands.

We abstracted this away to a small interface with the following functionality:

rvexInit
Attempts to open the files.

rvexDispose
Closes all files opened by rvexInit.

rvexBytecode
Allows you to write bytecode to the instruction memory.

rvexWrite
Allows you to write to the data memory.

rvexRead
  Allows you to read from the data memory.

rvexSeek
  Allows you to jump to a given position in the data memory.

rvexGo
  Writes the clear and start commands to the control memory and blocks until the status reports that the operation was successful.

3 Profiling results

We compiled the unmodified x264 application with profiling support on a VM that allows us to compile microblaze and ρ-VEX applications. Unfortunately we cannot use gprof on the Microblaze machine itself so we ran gprof on the VM itself.

Using the input video ~/Videos/inputs/eledream_640x360_8.y4m we obtained the following profiling results.

```
gprof x264 | head -n 10
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 13.70      0.10     0.10  1599044     0.00     0.00  x264_pixel_satd_8x4
 13.70      0.20     0.10   570708     0.00     0.00  get_ref
  6.85      0.25     0.05    38770     0.00     0.00  x264_pixel_sad_x4_16x16
  5.48      0.29     0.04   460484     0.00     0.00  quant_4x4
  4.11      0.32     0.03   123076     0.00     0.00  sa8d_8x8
```

We see that the function `x264_pixel_satd_8x4` is called 1.6 million times during the video conversion. These calls together take up about 14% of the total time. An equal amount of time is spend in `x264_pixel_satd_8x4`.

The video eledream_640x360_128.y4m makes the application spend about 18% of the runtime in the pixel processing function and 16% in `get_ref`.

The bigger our input video the more time we will spend processing and the more effect an optimization will have if it targets part of the processing code.

Judging from the profiling information there are two potential places where optimization will be effective: `x264_pixel_satd_8x4` and `get_ref`.

When we looked at the source code of x264 we thought that the nature of `x264_pixel_satd_8x4` was more suitable for optimization because it had some loops in it. The `get_ref` function was a lot more irregular and also harder to understand. On the flip side, the overhead caused by communication between the microblaze processor and the ρ-VEX would be smaller for `get_ref` because it is called three times less than `x264_pixel_satd_8x4`.

In the end we chose to try and put the computation of `x264_pixel_satd_8x4` on the co-processor and let `get_ref` for what it was. The function `x264_pixel_satd_8x4` was easier to understand and making it work was more important to us than choosing the best optimization area right away.

4 Endianess

Since we had to compile for the Microblaze using the flag -DWORD-BIGENDIAN we figured that the Microblaze would be a big endian machine. You can always test it by writing a multibyte value like 0xDEADBEEF to memory and read the individual bytes. If you read 0xDE 0xAD 0xBE 0xEF you will know that you have a big endian machine. If you get that sequence but in reverse you know its a little endian machine.

5 Results of using the ρ-VEX

We modified the x264 application to log its processing time computed with clock_gettime. This required linking the rt library.

Then we tried to find an fpga that was not being used by anyone else and we ran the application.

- How much speedup did you obtain?
- Is this what you have expected?
- Give a theoretical calculation for the speedup you should have ex- pected and compare it to the practical result.

6 Additional assignment

- What were the results of the additional assignment? How did it affect the speed of the application?

Figure 1: The Universe

7 Conclusion

"I always thought something was fundamentally wrong with the universe" [1]

References

[1] D. Adams. *The Hitchhiker's Guide to the Galaxy.* San Val, 1995.