332 Advanced Computer Architecture

Exercise (ASSESSED): Microarchitecture issues in a Sum of Absolute Differences benchmark

This is the first of two equally-weighted assessed coursework exercises. Working individually, do the exercise and write up a short report presenting and explaining your results. Submit your work in a pdf file electronically via CATE.[1] The CATE system will also indicate the deadline for this exercise.

Background

"Sum of Absolute Differences" (SAD) is a simple video quality metric, which is used in motion estimation for video compression. In this exercise, you will use an SAD implementation from the Parboil benchmark suite,[2] which is based on the full-pixel motion estimation algorithm found in the JM reference H.264 video encoder.

Running the SAD benchmark

Getting the benchmark code

Copy the benchmark code to your own directory, e.g.

```
prompt> mkdir /homes/yourid/ACA09
prompt> cd !$
prompt> cp -r /homes/phjk/ToyPrograms/ACA09/SAD ./
```

(The `./` above is the destination of the copy – your current working directory). List the contents of the benchmark directory:

```
prompt> cd SAD
prompt> ls
benchmarks common DIRECTORIES driver LICENSE parboil
README.benchmarks README.suite
```

Working with the Parboil suite

Make the Parboil suite's test harness:

```
prompt> cd common/src
prompt> make PARBOIL_ROOT=/homes/yourid/ACA09/SAD
prompt> cd ../..
```

See a description of the SAD benchmark:

[1] https://cate.doc.ic.ac.uk/

[2] http://impact.crhc.illinois.edu/parboil.php

```
prompt> ./parboil describe sad
Parboil parallel benchmark suite, version 0.1

  sad
    A "sum of absolute differences" benchmark.  This benchmark
    is based on the full-pixel motion estimation algorithm found
    in the JM reference H.264 video encoder.  Motion estimation
    searches for blocks in one image that approximately match blocks
    in another image.  This benchmark computes SADs for pairs of
    blocks, where an SAD is one metric for how closely two images
    match.

    There are three kernels.  One kernel computes SADs for 4-by-4
    blocks.  The next kernel consumes the first kernel's results to
    compute SADs for larger blocks, up to 8-by-8.  The last kernel
    computes SADs  for blocks up to 16-by-16.  Each kernel uses
    the previous kernel's output.

    Versions: cpu_ss cuda_base base cuda cpu_wattch cpu
    Data sets: default 64x64 16x16 32x32
```

In this exercise, we will simulate `cpu_ss` (SimpleScalar) and `cpu_wattch` (Wattch) versions. (Note that the paragraph about three kernels relates to the GPU versions.)

Compile and run the base CPU version:

```
prompt> ./parboil compile sad base
prompt> ./parboil run sad base default
```

`default` is the default data set, which is too large for simulation.[3]

Compile and run the SimpleScalar version withe the 32×32 data set:

```
prompt> ./parboil compile sad cpu_ss
prompt> ./parboil run sad cpu_ss 32x32
```

To clean previously compiled files type:

```
prompt> ./parboil clean sad base
```

Studying microarchitecture effects

Choose a Linux machine on the DoC network.[4]

Study the effect of various architectural features on the performance of the SAD benchmark.

To run the SAD benchmark in SimpleScalar type:

[3]It takes about 10 minutes on a 2.4GHz Core 2 Duo.
[4]See `https://www.doc.ic.ac.uk/csg/computers/`.

```
prompt> ./parboil run sad cpu_ss 32x32
```

To run the SAD benchmark in Wattch type:

```
prompt> ./parboil run sad cpu_wattch 32x32
```

To pass flags to SimpleScalar and Wattch, set the environment variable `SSFLAGS`:

```
prompt> setenv SSFLAGS "-ruu:size 64 -lsq:size 16"
```

Vary the RUU size between 2 and 256 (only powers of two are valid). (You may with to use a script `varyarch`.) Plot a graph showing your results. Explain what you see.

Vary the other microarchitecture parameters (leave the cache parameters unchanged). Where is the bottleneck (when running this application) in the default simulated architecture? Justify your answer.

Can you find the "sweet spot" architecture delivering best performance per unit power?

Write your results up in a short report (less than four pages including graphs and discussion).

Tools and tips

Plotting a graph

Try using the gnuplot program. Run the script above, and save the output in a file `table`. Type `gnuplot`. Then, at its prompt type:

```
set logscale x 2
plot [][0:2] 'table2' using 1:3 with linespoints
```

To save the plot as a postscript file, try:

```
set term postscript eps
set output "psfile.ps"
plot [][0:2] 'table2' using 1:3 with linespoints
```

Try `help postscript`, `help plot` etc for further details.

Appendix

If you have an NVIDIA GPU and CUDA installed, you can also compile and run CUDA versions: `cuda_base` and `cuda`, e.g.

```
prompt> ./parboil compile sad cuda
prompt> ./parboil run sad cuda default -S
```

Paul H.J. Kelly & Anton Lokhmotov, Imperial College London, 2009