332 Advanced Computer Architecture

Assessed exercise 1: instruction-level-parallelism in computing contour trees

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

Consider the height contours of an island with hills. If we draw a series of contours at 1-meter intervals, we find that higher contours are enclosed in lower contours. Each contour has exactly one parent contour, but may have more than one child. Thus, they form a tree, the "contour tree". A minimal contour tree branches only where contours split. This exercise is about an algorithm for computing minimal contour trees fast - much faster than the strategy of simply plotting many contours. A nice introduction to the problem and to the algorithmic strategies involved is here,

`http://web.cse.ohio-state.edu/~hwshen/788/Site/Slides_files/contourTree.pdf`.

We will be working with a contour tree library called "tourtre" (`https://github.com/sedillard/libtourtre`), and a sample application that uses it called tltree.

Your task is to find a configuration for a processor core that runs this benchmark using the minimum total amount of energy. For simplicity we count the entire benchmark run, which includes reading in the 3D dataset.

Running tltree natively

Copy the exercise's directory tree to your own directory:

```
cd
cp -r /homes/phjk/ToyPrograms/ACA17/tourtre-exp ./
cd tourtre-exp
```

Now (in this directory) you can run the program:

```
./tltree 32 32 32 bucky-32x32x32.raw
```

The parameters provide a sample 3D dataset as input and specify its size. It takes a fraction of a second. The program simply prints the number of nodes in the contour tree that it has calculated - in this case, 803.

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

Running tltree under the Simplescalar-Wattch simulator

You can now run the program using the simulator as follows:

```
./run-wattch
```

Here we use a much-reduced-size dataset as the simulator is rather slow - this takes 30-40 seconds. There is a danger that using such a small size might lead to misleading results, so some care is needed. Some larger datasets are provided.

For your convenience the `tourtre-exp`directory contains some example scripts for running the benchmark — see `varyarch` and `varyarch-energy`, explained below.

Studying microarchitecture effects

Choose an idle Linux machine on the DoC network.[2]

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

Vary the RUU size between 2 and 256 (only powers of two are valid). (Use a script `varyarch`.) Plot a graph showing your results (see below). Explain what you see. Now do the same but look at the total energy consumed — use the script `varyarch-energy`.

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 that runs this program with optimum energy efficiency? To be precise: find the simple-scalar configuration which finishes the computation in the minimum total energy. See the section below on how SimpleScalar's Wattch extension estimates the energy utilisation.

Write your results up in a short report (not more than four pages including graphs and discussion). The best solutions will be the ones which report a systematic strategy to find the optimum implementation, and which offer some insight and analysis of the results that you observe.

Tools and tips

The first output to look at from the simulator is "sim_cycle" - the total number of cycles to complete the run. It's often also useful to look at "sim_IPC", the instructions per cycle - provided you always execute the same number of instructions. The time taken to perform the simulation "sim_elapsed_time" simply tells you how long the simulator took.

Other outputs from the simulator can be helpful in guiding your search - eg "ruu_full", the proportion of cycles when the RUU is full.

[2] See `https://www.doc.ic.ac.uk/csg/facilities/lab/workstations`.

How wattch reports energy utilisation

Wattch reports an estimate of the total energy (roughly in nanojoules) required for the computation, tagged with "total_power_cycle_cc1" (the comment on this line is misleading - it *is* energy, not power).

Wattch is documented in this article,

`http://www.eecs.harvard.edu/~dbrooks/isca2000.pdf`

Figure 5 in the article shows the impact of Wattch's three different clock gating models - "cc1", "cc2" and "cc3". For this exercise you are invited to use "cc1" (this is what the script "varyarch-energy" reports). "cc2" and "cc3" are approximate models reflecting more optimistic/ambitious circuit-level implementations of power optimisation. "cc1" assumes that each unit is is fully on if any of its ports are accessed in that cycle. "cc2" assumes power scales linearly with port usage. "cc3" assumes ideal clock gating where the power scales linearly with port usage as in "cc2", but disabled units are entirely shut off.

Parameters that should not be changed

To limit the scale of the exercise you are asked not to change the cache configuration. Note that the perfect branch predictor is not a real thing it is a simulation trick to help you understand the scope for performance improvement through better branch prediction. Similarly, the -issue:wrongpath option must be true setting it false is not realistic Note that some simplescalar parameters make arbitrary changes to assumptions about the latency of operations they are not architectural choices eg "-fetch:speed" (speed of front-end of machine relative to execution), and "-fetch:mplat" (extra branch mis-prediction latency).

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 [][] 'table' using 1:3 with linespoints
```

To save the plot as a postscript file, try:

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

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

Source code

If you are interested, the source code for `tltree` is available here:

```
/homes/phjk/ToyPrograms/ACA17/tourtre/libtourtre-master
```

To compile the code you need to make the library and then make the application:

```
make clean
make
cd examples/trilinear
make
```

Paul Kelly, Imperial College London, 2017