332 Advanced Computer Architecture

Exercise (ASSESSED): Microarchitecture issues in Three-Dimensional Modeling of RNA

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

This exercise is based on a small sample application which has been used for performance benchmarking in the functional programming community, but we will study a C implementation. The program generates 3D structures of RNA. An RNA molecule is specified by its nucleotide sequence (eg CCAG...). The program performs a backtracking search for a conformation that satisfies various constraints. The test problem we will use is an "anticodon loop", with the constraint that the loop joins us. The science behind the problem is described in this article:

> Marc Feeley, Marcel Turcotte, and Guy Lapalme. 1994. Using Multilisp for solving constraint satisfaction problems: an application to nucleic acid 3D structure determination. Lisp Symb. Comput. 7, 2-3 (December 1994), 231-247. DOI=10.1007/BF01018696 http://dx.doi.org/10.1007/BF01018696
>
> `http://www.iro.umontreal.ca/~feeley/papers/FeeleyTurcotteLapalmeLSC94.pdf`

Running the benchmark

Getting the benchmark code

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

```
prompt> mkdir /homes/yourid/ACA13
prompt> cd !$
prompt> cp -r /homes/phjk/ToyPrograms/ACA09/Pseudoknot ./
```

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

```
prompt> cd Pseudoknot
prompt> ls
init_nucleotides.c  nucleic-core.c~  nucleic.h~   varyarch
Makefile     nucleic-database.c  nucleic.scm  varyarch-energy
nucleic.c     nucleic-database.c~  run-wattch
nucleic-core.c     nucleic.h  run-wattch~
```

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

Build the code using Make:

```
prompt> make
```

(It is not necessary for this exercise to figure out what the code does or how it works, but if you wish to, you are advised to read "nucleic.scm" - a version of the code in the Scheme language - as this has the most useful comments. The code we actually run is derived from "nucleic.c", but split into three files to overcome some issues with the simplescalar compiler).

Run the program:

```
prompt> ./nucleic.x86 1
```

This runs extremely quickly[2]. Verify that the number of solutions found is 179. Run the SimpleScalar version:

```
prompt> /homes/phjk/simplesim/sim-outorder ./nucleic.ss 1
```

This might take a a few seconds to run. We will also be using an extension of simplescalar, called "wattch", that also computes the energy used by the simulated architecture:

```
prompt> /homes/phjk/simplesim-wattch/sim-outorder ./nucleic.ss 1
```

To clean previously compiled files type:

```
prompt> make clean
```

Studying microarchitecture effects

Choose a Linux machine on the DoC network.[3]

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

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. 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.

[2] Although not needed for this exercise, longer-running versions of the program can be executed using "./nucleic.x86 2" and "./nucleic.x86 3"; more solutions are found.

[3] See `https://www.doc.ic.ac.uk/csg/computers/`.

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.

0.0.1 How wattch reports energy utilisation

Wattch reports an estimate of the total energy 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.

Paul H.J. Kelly Imperial College London, 2013