332 Advanced Computer Architecture

Exercise (ASSESSED): Microarchitecture issues in Local Assembly using the Finite Element Method for a Helmholtz problem

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

Firedrake is an automated system for the portable solution of partial differential equations using the finite element method (FEM). To solve a problem, an FEM splits the domain of the equation into a set of disjoint cells, leading to what is usually called a "mesh". The **local assembly** is a fundamental step of an FEM, in which the solution of the equation is sought within each cell in the mesh. This exercise is based on a small piece of code that has been automatically generated through Firedrake. It implements the local assembly phase for a Helmholtz equation, which is used, for example, to simulate the transmission of heat through an object. To simplify your study, the program performs local assembly on a "ficticious" mesh, in which cells coordinates are not read from memory, and all local assembly results are discarded once computed, rather than accumulated in a global matrix for the subsequent step of FEM.

For more information about Firedrake, visit

    http://www.firedrakeproject.org

Running the benchmark

Getting the benchmark code

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

```
prompt> mkdir /homes/yourid/ACA14
prompt> cd !$
prompt> cp -r /homes/phjk/ToyPrograms/ACA14/HelmholtzAssembly ./
```

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

```
prompt> cd HelmholtzAssembly
prompt> ls
gettimemicroseconds.c  gettimemicroseconds.h  Helmholtz_3Dmesh_poly3.c
Makefile  reference_element.h
```

[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).

Run the program:

```
prompt> ./helmholtz.x86
```

This runs extremely quickly[2]. Run the SimpleScalar version:

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

This might take 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 ./helhmoltz.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 Helmholtz 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.

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.

[2]Although not needed for this exercise, longer-running versions of the program can be executed changing the compile-time parameter *ITER* in the Makefile. Conceptually, a higher value of *ITER* means that the mesh contains more cells.

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

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.

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 and Fabio Luporini, Imperial College London, 2013