ECE3056 Assignment 6 - Memory Hierarchy Design

Due Friday, 30 December, 2012

1 Summary

For this assignment you will design a memory hierarchy, optimizing for energy, with up to 1.5 megabits of SRAM and up to two levels of unified cache.

2 Introduction

Memory system design is crucial to the performance of modern computing systems in every application area. Speed, die area, and energy usage are all important considerations made when any microcomputer system is designed.

In the previous assignment, you built a cache simulator and used it to evaluate a set of unified L1 cache designs based on miss rate. In this follow-up assignment, you will use the results from this simulator along with a simple power model to design a multi-level memory hierarchy.

This assignment is intended to be open-ended. You are asked to design a cache that satisfies a set of constraints: Use 1.5 megabits of SRAM and up to two levels of unified cache. Any memory hierarchy fitting these constraints will be accepted, as long as it can be demonstrated that an effort was made to minimize energy according to the given model.

3 Deliverables

For this assignment, you will turn in a report describing:

- The design chosen.
- Alternative designs explored.
- Experiments performed to arrive at the design.
- Data collected in these experiments.

and also any program code (scripts, makefiles, spreadsheets) generated for the assignment.

4 Modeling Multi-Level Caches

The cache simulator from the previous assignment has been modified to model arbitrary multi-level cache hierarchies. The baseline program.s usage message now reads:

```
./cachesim <trace> <<block size> <cache size> <ways>>xN
```

By specifying block size, cache size, and associativity for each level (of N) of cache, multi-level cache hierarchies can be built. Unlike the previous assignment, no script is provided to automate the search.

5 Energy and Timing Models

Design space exploration is typically performed using simplified models. The following represents a simplified energy and timing model for a hypothetical memory system.

5.1 Energy Model

There are two basic components to energy expenditure in digital electronics: static and dynamic energy. Static energy is dissipated at a constant rate by each SRAM cell and logic element on chip whether it is switching or not, and dynamic energy is energy that depends on circuit activity.

We will model energy expended over a span of time with duration T as:

$$\sum_{l \in \text{levels}} E_{\text{l}}$$

Where E_{l} is the energy expended at cache level l:

$$E_{\text{l}} = P_{\text{leakage}} T + \vec{c} \cdot \vec{D}$$

where:

- P_{leakage} is $2N$nW, N being be number of total SRAM cells (bits).
- $\vec{c}$ is the set of counter values (output of cache sim: $\begin{pmatrix} \text{accesses} \\ \text{misses} \\ \text{writebacks} \end{pmatrix}$), reset at the beginning of the period.
- $\vec{D}$ is the dynamic power vector, $\begin{pmatrix} 4 \\ 70 \\ 90 \end{pmatrix}$ pJ, doubled for each successive level of cache. The exception to this rule is the last level. DRAM accesses (reads or writes) cost 100pJ per 8 bytes.

Be sure to include tag, dirty, and valid bits in your calculations of the number of SRAM cells.

5.2 Timing Model

Although the goal is to optimize for energy, because of leakage, the execution time for applications is important. Assume that applications execute at a baseline 1CPI with a clock rate of 2 GHz. Access times for each level of cache are

$\max\{\lceil \log_2(\frac{\text{cache\_size}}{4\text{kB}}) \rceil + 1, 1\}$ cycles. Main memory requires $20 + 2 \cdot$block_size cycles. A one-cycle access time for L1 is hidden by the pipeline (for both instruction fetches and data accesses), but each additional cycle beyond 1 must be added to the execution time. There is an additional one-cycle penalty each doubling of associativity above 2.

How many instructions are in each application? Accesses marked with an `i` are instruction fetches. Counting the number of lines beginning with i in each of these traces (and assuming 6 instructions per access in the remaining traces) gives us the following for the application traces:

Trace	Instructions
bubble	4 503 249
merge	5 058 837
shuf	11 108 665
sieve	17 953 033
random64k	1 572 864
stream1M	1 572 864

6 Example Simulation Run and Energy Calculation

For this example, we will run the power and timing models for three cache hierarchy designs and a single application (bubble sort):

- 128kB 32-way L1 with 32-byte lines
- 96kB 6-way L2 with 1024-byte lines, 32kB 2-way L1 with 64-byte lines
- 64kB 32-way L2 with 64-byte lines, 8kB direct mapped L1 with 64-byte lines

6.1 Running the Simulation

The commands to run the simulator for these configurations (and the outputs given by the simulator) are:

```
$ ./cachesim trace.bubble 32 131072 32
6322343, 15368, 4551
$ ./cachesim trace.bubble 64 32768 2 1024 98304 6
6322343, 18384, 5207, 23591, 6029, 1599
$ ./cachesim trace.bubble 64 8192 1 64 65536 32
6322343, 84130, 24101, 108231, 11684, 3739
```

6.2 Modelling Execution Time

Bubble sort requires 4,503,249 cycles at the baseline 1CPI (based on the instruction count table in Section 5.2), plus:

- 6 + 4 - 1 cycles for accesses to the 128kB L1
  - 6-cycle access time.
  - 4 cycles due to associativity.
  - Minus 1 cycle due to latency hiding in the pipeline.
- 4 - 1 cycles for accesses to the 32kB L1
- 2 - 1 cycles for accesses to the 8kB L1
- 6 + 1 cycles for accesses to the 96kB L2
  - Note that an "additional cycle for each doubling" implies a floor function, so a 6-way cache has the same latency as a 4-way cache and a 3-way cache has the same latency as a 2-way cache.
- 5 + 4 cycles for accesses to the 64kB L2

This leads to the following execution times for our three cache configurations (remember the clock rate is given as 2 GHz):

- $4503249 + 9{\cdot}6322343 + (15368 + 4551){\cdot}(20 + 2{\cdot}32)$
  $\Rightarrow 63077532\text{cyc} \Rightarrow 31.54\text{ms}$
- $4503249 + 3{\cdot}6322343 + 7{\cdot}23591 + (6029 + 1599){\cdot}(20 + 1024{\cdot}2)$
  $\Rightarrow 39410119\text{cyc} \Rightarrow 19.71\text{ms}$
- $4503249 + 1{\cdot}6322343 + 9{\cdot}108231 + (11684 + 3739){\cdot}(20 + 64{\cdot}2)$
  $\Rightarrow 14082275\text{cyc} \Rightarrow 7.04\text{ms}$

6.3 Modelling Energy

To estimate leakage power, we must first compute the total number of bits of SRAM. Assuming 32-bit physical addresses, we get:

Cache	Blocks	Sets	Tag Bits	Tag Store Bits	Total Bits
128kB L1	4096	128	20	90 112	1 138 688
32kB L1	512	256	18	10 240	272 384
8kB L1	128	128	19	2 688	68 224
96kB L2	96	16	18	1 920	788 352
64kB L2	1024	32	21	23 552	547 840

Given 2nW per bit, the cache hierarchies have the following energy dissipation due to leakage power:

- $31.54\text{ms}{\cdot}2\frac{\text{nW}}{\text{bit}}{\cdot}1138688\text{bit} \Rightarrow 71.83\mu\text{J}$
- $19.71\text{ms}{\cdot}2\frac{\text{nW}}{\text{bit}}{\cdot}(272384\text{bit} + 788352\text{bit}) \Rightarrow 41.81\mu\text{J}$
- $7.04\text{ms}{\cdot}2\frac{\text{nW}}{\text{bit}}{\cdot}(68224\text{bit} + 547840\text{bit}) \Rightarrow 8.674\mu\text{J}$

These values must then be added to the dynamic power, computed by multiplying counter values by coefficients and summing the results. The coefficient vectors for each level are determined using the method given in Section 5.2:

- $\begin{pmatrix} 4 \\ 100\frac{\text{block\_size}}{8} \\ 100\frac{\text{block\_size}}{8} \end{pmatrix}$ pJ in an L1 cache with no L2.
- $\begin{pmatrix} 4 \\ 70 \\ 90 \end{pmatrix}$ pJ in an L1 cache when an L2 is present.
- $\begin{pmatrix} 8 \\ 100\frac{\text{block\_size}}{8} \\ 100\frac{\text{block\_size}}{8} \end{pmatrix}$ pJ in the L2 cache.

This makes the dynamic power for our three sample caches:

- $\begin{pmatrix} 6322343 \\ 15368 \\ 4551 \end{pmatrix} \cdot \begin{pmatrix} 4 \\ 400 \\ 400 \end{pmatrix}$ pJ $\Rightarrow 33.26\mu$J
- $\begin{pmatrix} 6322343 \\ 18384 \\ 5207 \end{pmatrix} \cdot \begin{pmatrix} 4 \\ 70 \\ 90 \end{pmatrix}$ pJ $+ \begin{pmatrix} 23591 \\ 6029 \\ 1599 \end{pmatrix} \cdot \begin{pmatrix} 8 \\ 12800 \\ 12800 \end{pmatrix}$ pJ $\Rightarrow 124.9\mu$J
- $\begin{pmatrix} 6322343 \\ 84130 \\ 24101 \end{pmatrix} \cdot \begin{pmatrix} 4 \\ 70 \\ 90 \end{pmatrix}$ pJ $+ \begin{pmatrix} 108231 \\ 11684 \\ 3739 \end{pmatrix} \cdot \begin{pmatrix} 8 \\ 800 \\ 800 \end{pmatrix}$ pJ $\Rightarrow 46.55\mu$J

Making the total energy:

- $71.83\mu\text{J} + 33.26\mu\text{J} \Rightarrow 105.1\mu\text{J}$
- $41.81\mu\text{J} + 124.9\mu\text{J} \Rightarrow 166.7\mu\text{J}$
- $8.674\mu\text{J} + 46.55\mu\text{J} \Rightarrow 55.22\mu\text{J}$

Clearly the third design is the best, if we look only at this application.

7 Advice

This is an open-ended project and there are many ways to reach its goals. What follows is a bulleted list of techniques and considerations for this project.

- MATLAB and the workalike open source product GNU Octave provide an easy way to quickly generate plots and analyses of data. The builtin `csvread()` function is extremely useful.

- Both Octave and Python make handy calculators (and Python is arbitrary-precision) for doing simple things like figuring out the exact number of bytes in a 3MB cache.
- Another possible way to generate plots and do the energy calculations is with a spreadsheet.
- If you are not comfortably proficient at creating shell scripts or batch files, there is always the option of renaming the `main()` function and writing a new `main()` function that calls it multiple times.
- In fact, while you're modifying the simulator, consider building in the energy and timing models.
- All of the traces take a while to run through the simulator. Consider using the `head` and `tail` commands to cut them into smaller chunks for testing your caches.
- How do you use `head` and `tail`? The most important command is `man` (as in "manual"). Try `man head`, `man tail`.
- One aspect of this assignment left open-ended is which metric of central tendency to use when comparing designs over several applications. Arithmetic mean (or, in a similar vein, just the sum of the energy from several traces) makes a lot of sense here but it's certainly not the only option.