Project #1

(Report due at instructor's office or in class by end of lecture on Thursday, October 17th, 2013)

Project description: Cache design choices (i.e. # of levels, size, associativity, replacement policy etc.) affect the performance of a microprocessor. In this project, you are asked to fine-tune the cache hierarchy of an Alpha microprocessor for 4 individual benchmarks. The cache design parameters you can modify are:

- **Cache levels**: One or two levels, for data and instruction caches.
- **Unified caches**: Selection of separate vs. unified instruction/data caches. For example, you can have separate L1 caches and a unified L2 cache.
- **Size**: Cache size, one of the most important choices.
- **Associativity**: Selection of cache associativity (e.g. direct mapped, 2-way set associative, etc.).
- **Block size**: Block size of the cache, usually 64 or 32 bytes.
- **Block replacement policy**: Selection between FIFO, LRU and Random.

While larger caches generally mean better performance, they also come at a greater cost. Thus, sensible design choices and trade-offs are required. To this end, in this project you will also be asked to define a cost function and to use it in order to identify the optimal configuration.

Part 1: Preparation

Step 0: Find a teammate

Projects will be carried out in teams of two. You are asked to identify your teammate and send an email to the TA (tianyu.chen@utdallas.edu) with the names and UTD netIDs of the team members by Sep. 24th. If you cannot find a teammate, email the TA before the above date and one will be assigned to you. There are 68 students in the class, so we should end up with 34 teams.

Step 1: Prepare your account

A virtual server has been set up for the purpose of carrying out the projects of this class. The server name, accessible from on-campus machines or via UTD VPN from off-campus machines, is cs6304-32. To access it, use your UTD netid and password via ssh as follows:

ssh xxx000000@cs6304-32 (where xxx000000 is your netid)

Step 2: Simplescalar setup

Simplescalar is a functional simulator which we will use in this project. It supports various ISAs and comes in different flavors, according to simulation needs. Since we focus on caches on this project, we will use sim-cache. More information can be found on www.simplescalar.com.

1. Download Simplescalar 3.0 (www.simplescalar.com)

2. Extract it: `tar xvzf simplesim-3v0e.tgz`
3. Install Simplescalar:

   a. `cd simplesim-3.0`

   b. `make`

4. Run a sample benchmark and verify it works:

   `./sim-cache tests-alpha/bin/test-math`
   Your output should include the following line, verifying that everything works:
   `"-1e-17 == -1e-17 Worked!"`

Step 3: Benchmarks setup

1. Download benchmarks (distributed by Prof. Todd Austin at University of Michigan): http://www.eecs.umich.edu/~taustin/eecs573_public/instruct-progs.tar.gz
2. Extract them in the simplesim-3.0 directory (or anywhere, just be careful with the paths)
3. Test all benchmarks (information can be found in the benchmarks/README file). For example, the following commands test the GCC benchmark:

   ```
   ./sim-cache benchmarks/cc1.alpha -O ./benchmarks/1stmt.i
   diff benchmarks/1stmt.s benchmarks/1stmt.s.ref
   ```

   Diff should return nothing, meaning that everything worked fine.

   Note: Benchmark "compress95" does not run on the server. Please use the other ones.

Part 2: Find CPI

In this part, we will calculate the CPI for the four individual benchmarks. Our baseline configuration will be the Alpha 21264 EV6 configuration:

- **Cache levels**: Two levels.
- **Unified caches**: Separate L1 data and instruction cache, unified L2 cache.
- **Size**: 64K Separate L1 data and instruction caches, 1MB unified L2 cache.
- **Associativity**: Two-way set-associative L1 caches, Direct-mapped L2 cache.
- **Block size**: 64 bytes.
- **Block replacement policy**: FIFO.

In order to simulate the EV6 configuration, the appropriate parameters need to be supplied to simplescalar. Specifically:

```
Parameter       Default           Info
-cache:dl1      dl1:256:32:1:l  # l1 data cache config, i.e., {<config>|none}
-cache:dl2      ul2:1024:64:4:l # l2 data cache config, i.e., {<config>|none}
-cache:il1      il1:256:32:1:l  # l1 inst cache config, i.e., {<config>|dl1|dl2|none}
```

```
-cache:il2                dl2     #     l2     instruction     cache     config,     i.e.,
{<config>|dl2|none}
-tlb:itlb         itlb:16:4096:4:l # instruction TLB config, i.e., {<config>|none}
-tlb:dtlb         dtlb:32:4096:4:l # data TLB config, i.e., {<config>|none}
```

The cache config parameter <config> has the following format:

```
<name>:<nsets>:<bsize>:<assoc>:<repl>
<name>   - name of the cache being defined
<nsets>  - number of sets in the cache
<bsize>  - block size of the cache
<assoc>  - associativity of the cache
<repl>   - block replacement strategy, 'l'-LRU, 'f'-FIFO, 'r'-random
```

Examples: `-cache:dl1 dl1:4096:32:1:l`

Cache levels can be unified by pointing a level of the instruction cache hierarchy to the data cache hierarchy using the "dl1" and "dl2" cache configuration arguments. Most sensible combinations are supported, e.g.,

A unified L2 cache (il2 is pointed to dl2):

```
-cache:il1 il1:128:64:1:l -cache:il2 dl2
-cache:dl1 dl1:256:32:1:l -cache:dl2 ul2:1024:64:2:l
```

Or, a fully unified cache hierarchy (il1 pointed to dl1):

```
-cache:il1 dl1
-cache:dl1 ul1:256:32:1:l -cache:dl2 ul2:1024:64:2:l
```

Note: Configurations that do not make much sense are not supported. For example, it is not possible to run sim-cache with L1 cache unified and L2 separate. Indeed, when L1 is unified, L2 needs to be unified. But the following configurations are supported:

L1 separate L2 separate
L1 separate L2 unified
L1 unified L2 unified

However, there is parsing bug in simplescalar when both caches are unified. So, instead of:

-cache:il1 dl1 -cache:dl1 ul1:256:32:1:l -cache:dl2 ul2:1024:64:2:l

you need to use

-cache:il1 dl1 -cache:dl1 ul1:256:32:1:l -cache:dl2 ul2:1024:64:2:l -cache:il2 none

---

In order to execute the GCC benchmark using the EV6 configuration, we need to run the following command:

```
./sim-cache -cache:dl1 dl1:512:64:2:f -cache:il1 il1:512:64:2:f -cache:il2 dl2 -cache:dl2 dl2:16384:64:1:f -tlb:itlb none -tlb:dtlb none ./benchmarks/cc1.alpha -O ./benchmarks/1stmt.i
```

This defines a L1 Data cache, 2-way set-associative, with a 64-byte block, 512 sets (2*64*512=64KB). It also defines a similar Instruction cache, and a unified L2 1MB cache (16384 sets *64 block size, direct mapped). For this project, you can ignore the presence of TLBs.

The execution of this command provides the following output:

```
sim: ** simulation statistics **
sim_num_insn              337326966 # total number of instructions executed
sim_num_refs              121892633 # total number of loads and stores executed
sim_elapsed_time                 39 # total simulation time in seconds
sim_inst_rate          8649409.3846 # simulation speed (in insts/sec)
il1.accesses              337326966 # total number of accesses
il1.hits                  335739247 # total number of hits
il1.misses                  1587719 # total number of misses
il1.replacements            1586695 # total number of replacements
il1.writebacks                    0 # total number of writebacks
il1.invalidations                 0 # total number of invalidations
il1.miss_rate                0.0047 # miss rate (i.e., misses/ref)
il1.repl_rate                0.0047 # replacement rate (i.e., repls/ref)
il1.wb_rate                  0.0000 # writeback rate (i.e., wrbks/ref)
il1.inv_rate                 0.0000 # invalidation rate (i.e., invs/ref)
dl1.accesses              124102754 # total number of accesses
dl1.hits                  122784344 # total number of hits
dl1.misses                  1318410 # total number of misses
dl1.replacements            1317386 # total number of replacements
dl1.writebacks               424033 # total number of writebacks
dl1.invalidations                 0 # total number of invalidations
dl1.miss_rate                0.0106 # miss rate (i.e., misses/ref)
dl1.repl_rate                0.0106 # replacement rate (i.e., repls/ref)
dl1.wb_rate                  0.0034 # writeback rate (i.e., wrbks/ref)
dl1.inv_rate                 0.0000 # invalidation rate (i.e., invs/ref)
dl2.accesses                3330162 # total number of accesses
dl2.hits                    2893762 # total number of hits
dl2.misses                   436400 # total number of misses
dl2.replacements             420016 # total number of replacements
dl2.writebacks               141027 # total number of writebacks
dl2.invalidations                 0 # total number of invalidations
dl2.miss_rate                0.1310 # miss rate (i.e., misses/ref)
dl2.repl_rate                0.1261 # replacement rate (i.e., repls/ref)
dl2.wb_rate                  0.0423 # writeback rate (i.e., wrbks/ref)
dl2.inv_rate                 0.0000 # invalidation rate (i.e., invs/ref)
ld_text_base           0x0120000000 # program text (code) segment base
ld_text_size                1564672 # program text (code) size in bytes
ld_data_base           0x0140000000 # program initialized data segment base
ld_data_size                 277104 # program init'ed `.data' and uninit'ed `.bss'
size in bytes
ld_stack_base           0x011ff9b000 # program stack segment base (highest address in
stack)
ld_stack_size                 16384 # program initial stack size
ld_prog_entry          0x0120025f70 # program entry point (initial PC)
ld_environ_base        0x011ff97000 # program environment base address address
ld_target_big_endian              0 # target executable endian-ness, non-zero if big
endian
mem.page_count                  786 # total number of pages allocated
mem.page_mem                   6288k # total size of memory pages allocated
mem.ptab_misses                613757 # total first level page table misses
mem.ptab_accesses           926298919 # total page table accesses
mem.ptab_miss_rate             0.0007 # first level page table miss rate
```

So, the L1 ICache has a 0.47% miss rate, the Dcache 1.06%, and the unified L2 13.10%.

Deliverables: Given an L1 miss penalty of 5 cycles, L2 miss penalty of 40 cycles, and one cycle cache hit/instruction execution, calculate the CPI for each benchmark.

Part 3: Optimize CPI for each benchmark

If you repeat the previous experiment, this time separating the L2 cache into a 512KB L2 DCache and a 512KB L2 ICache, you should get miss rates of 12.72% and 15.84%, which implies worse performance than in the case of a unified L2 cache. This is an example of how a cache design choice affects the performance of the microprocessor.

Deliverables: Given a two-level cache hierarchy, 128KB available for L1 cache and 1MB available for L2 cache, identify the optimal configuration (in terms of achieved CPI) for each benchmark. You should decide between unified/separate caches, associativity, replacement policy etc. Explain the reasoning behind your design choices for each configuration. Present graphs showing the trade-offs between design choices. Note that writing scripts (python, shell) to automate the process will save you a lot of time.

Part 4: Define cost function

In this part, you need to use your intuition in order to define a cost function for the caches, in terms of area overhead and performance. Obviously, larger caches are more expensive, so size should be a key parameter of the cost function (don't just use the fixed value from part 3). Similarly, an LRU replacement policy is more expensive than FIFO, and associativity also increases the cost of the cache (by adding extra hardware). L2 caches are cheaper because they are much slower (a larger L1 cache may be a good idea, but no designer uses it because of cost).

Deliverables: Define a cost function, in arbitrary cost units, using any parameters you see fit. The cost function should accurately reflect the design choices. For example, change of replacement policy from FIFO to LRU could increase the cost by 5%, while doubling the cache sizes would double the cost.

Part 5: Optimize caches for performance/cost

Given the cost function you defined in the previous part, you can now accurately select an optimal cache configuration for each benchmark, as well as all benchmarks combined.

Deliverables: Identify the optimal cache configuration for each benchmark, and the optimal configuration for all benchmarks (in terms of the average CPI). Present graphs showing the trade-off between CPI and cost for different design choices.