Georgia Institute of Technology

CS4290/CS6290/ECE4100/ECE6100

Assignment #2
Program due: Tuesday (2/11) 11:50 pm T-square
Prof. Moinuddin Qureshi, Instructor

This is an individual assignment. You can discuss this assignment with other classmates but you should do your assignment individually. Please follow the submission instructions. If you do not follow the submission file names, you will not receive the full credit. Please check the class homepage to see the latest update. Your code must run on the Shuttle cluster with g++4.1. This assignment is sufficiently longer than the first assignment and requires lots of C++ programming skills. Please start early!

Part 1: Simulator (80%): Complete the memory system

You will extend your Lab #1 pipeline design. Please download the new simulator frame. From sim.cpp/sim.h, you copy all the lines that have "NEW-LAB2" to your lab1 sim.cpp/sim.h. You have to extend your sim.cpp. In the new frame, knob related files and makefile are changed. Cache and memory related files are also added to help your assignment. Please look at memory.h file very carefully and read this assignment description and faq carefully and understand them.

Step 1:

- You need to complete the dcache_access function in this assignment. I-cache is still a perfect cache (i.e., always cache hit) for this assignment

  To activate your dcache structure, you must turn off KNOB_PERFECT_DCACHE.
  e.g.) sim --perfect_dcache=0

- Note that KNOB_PERFECT_DCACHE should work even after you implement a data cache. Hence, when KNOB_PERFECT_DCACHE value is 1, regardless of data cache size, all the cache access should be cache hits.

- The cache has a 64B block size, true LRU and write-back policy. D-cache access latency is set by KNOB_DCACHE_LATENCY. KNOB_DCACHE_SIZE, and KNOB_DCACHE_WAY set the cache configurations. Cache size should use a K-Byte unit
  e.g) sim --perfect_dcache=0 --dcache_size=1 --dcache_way=4
  i.e.) cache size = 1KB, 1024/4/64=4 sets

- Note that, a load/store instruction still takes load/store instruction latency cycles inside the execution stage just like the first assignment. You can assume that load/store latency-cycle is required to compute memory addresses.

- You implement the write-allocate policy, so for both store and load misses, you bring the entire cache block.
- To simplify your simulator, store instructions can be retired before memory requests are serviced. Load instructions have to wait until the memory request comes back from the DRAM. This means, the processor allows an out of order retirement.

- We are implementing a **non-blocking cache**. Even if an instruction generates a cache miss, the pipeline continues to execute if there are ready instructions.

- Inside the simulator, the simulator calls the dcache_access function at the last cycle of dcache access latency to mimic the actual hardware more closely. In other words, if KNOB_DCACHE_LATENCY is 3, and a memory op starts to access the dcache at cycle 4 in the simulator. (i.e., the op is latched into the EXE_latch at the end of cycle 3). The simulator calls dcache_access at cycle 6, so that the LRU replacement policy and/or an insertion into mshr_entry are done at cycle 6.

Step 2: Summary of how/when to send an op to MSHR.
You need to send op into the MSHR. The size of MSHR is determined by KNOB_MSHR_SIZE.

- [1]: In MEM_stage, the instruction checks the dcache if mem_type is greater than 1. The result of dcache hit or miss is available after KNOB_DCACHE_LATENCY. (in your simulator, the memory instruction could just wait in the EXE_latch for the KNOB_DCACHE_LATENCY cycles and then calls dcache_access function at the last cycle).
- [2](cache hit): If a cache hit, the instruction will be moved to the WB stage.
- [2](cache miss): If a cache miss, go to 3
  [3]
  - The instruction searches MSHR using the memory address (cache block address). If there is a match (i.e., the memory address is the same), the processor records the instruction id into the MSHR entry. (check_piggyback() function is provided for this purpose in memory.cpp) To simplify a simulator, when a new memory op is merged into a MSHR entry, the simulator does not update the memory_request information in the MSHR_entry. i.e.) For example, if a load op creates a MSHR entry and later a store op is merged (piggyback) into the MSHR. The simulator does not change the m_mem_req information in the corresponding MSHR entry.
  - If an old memory request is a store instruction and the new memory request is a load instruction, and their addresses are overlapped, the new load instruction can get the value directly from the store instruction. (store-load forwarding). In this case, it works like cache hit. The new load instruction can be move to the WB stage after KNOB_DCACHE_LATENCY. When the simulator checks MSHR, it checks cache block address not the actual memory address. Because we always bring the entire cache block.
- [4]: If there is no match in MSHR, the processor checks the size of MSHR. If there is no space in the MSHR, the processor stalls. When there is a space in the MSHR, the processor can put a request into the MSHR. The mem stage can process a new instruction from the following cycle. If there is space in the MSHR, the processor creates an entry in the MSHR.

Step 3: Modeling a DRAM (main_mem->run_a_cycle())

- mem->run_a_cycle is called before WB_stage();
- The mem->run_a_cycle function inserts a memory request into the dram_in_queue, send a request from dram_in_queue to dram bank scheduling buffer, accesses the DRAM bank, and moves to dram_out_queue and finally inserts the request into the dcache and MSHR.
- There is no size limitation for dram_in_queue, dram_out_queue and dram_bank_sch_queue
  Every cycle,

(1) send_bus_in_queue(): traverse mshr and generate a new memory request
(2) push_dram_sch_queue();
(3) dram_schedule(): dram scheduling
(4) send_bus_out_queue()
(5) fill_queue()

- A memory request can move only one function to the other function at one cycle. (i.e., if a memory request is inserted into the dram_in_queue the request can be sent to the dram_sch_queue at the following cycle.)

- (1) send_bus_in_queue(): Traverse all MSHR entries. If there is a new MSHR entry (i.e., m_stat == MEM_NEW), and insert the new memory request into **dram_in_queue.** (Caution!: Do not delete the MSHR entry. The simulator should keep the MSHR entry.) At one cycle, we only insert only one memory request into the dram_in_queue.

- (2) push_dram_sch_queue(): From dram_in_queue, move a memory request into the dram_bank_sch_queue
  Each dram bank has its own queue
- (3) dram_schedule(): Traverse all entries in dram_bank_sch_queue and if there is an unscheduled memory request, check the corresponding DRAM bank and see whether the DRAM bank is available.
  (if (dram_bank_rdy_cycle[bank_id] < cycle_count), the bank is available)

  bank_id = (addr/(KNONB_DRAM_PAGE_SIZE*1024))%(KNOB_DRAM_BANK_NUM)

  - If the corresponding bank is idle, check the last row buffer number.
  - If the memory request has the same row id, you set req->m_rdy_cycle = cycle_count + KNOB_MEM_LATENCY_ROW_HIT
  - If the row buffer number of the corresponding dram bank is different from the row id of memory request, you set req->m_rdy_cycle = cycle_count + KNOB_MEM_LATENCY_ROW_MISS, and also set row buffer id (addr/(KNOB_DRAM_PAGE_SIZE*1024)) as the row id of the memory request. You also set dram_bank_rdy_cycle = req->m_rdy_cycle;

- (4) send_bus_out_queue(): check dram_bank_sch_queue and see whether memory request is ready or not. (i.e. req->m_rdy_cycle < cycle_count). When the memory request is ready, the memory request is moved into the dram_out_queue.
  If multiple requests are ready at the same cycle, we still move all the requests into the fill_queue

- (5)fill_queue(): Pop the oldest request in the dram_out_queue and insert the request into the dcache and also free the corresponding MSHR entry. Only one memory request is popped from dram_out_queue at one cycle. After we free the MSHR entry, all the waiting instructions can be latched to the WB stage and can be moved to the WB stage at the following cycle. To simplify the problem, all the waiting instructions can be moved to the WB stage together and all of them can retire together. The freed MSHR entry is ready to be used at the same cycle. (The code to free mshr entry is already provided)

- Relevant Op structures
  op->ld_vaddr: load address
  op->st_vaddr: store address
  op->mem_read_size: load memory size (for unaligned accesses, we only consider the first cache block in this assignment)

op->mem_write_size: store memory size (for unaligned accesses, we only consider the first cache block in this assignment)

Knobs related to this assignment

KNOB_DCACHE_SIZE: data cache size (kbytes) (default value: 512 i.e., 512KB)
KNOB_DCACHE_WAY: N-way set associative data cache (default value: 4)
KNOB_DCACHE_LATENCY: cache latency when a cache hit (default value: 5)
KNOB_MEM_LATENCY_ROW_HIT: DRAM access latency when row buffer hit. (default value: 100)
KNOB_MEM_LATENCY_ROW_MISS: DRAM access latency when row buffer miss (default value: 200)
KNOB_DCACHE_FIXED_PENALTY: dcache miss has the fixed penalty (default value: 200)
KNOB_MSHR_SIZE: the number of entries in the MSHR (default value is 4)
KNOB_DRAM_BANK_NUM: The number of DRAM banks (default value is 4
KNOB_DRAM_PAGE_SIZE: the size of DRAM banks (unit: KB) (default value is 2 i.e., 2KB)
KNOB_PRINT_MEM_DEBUG: print out memory debug information

You have to update dcache_hit_count, dcache_miss_count accordingly.

Grading Policy

- IPC and cache hit/miss ratio check when cache size is varied (20%)
- IPC and other DRAM related counters check when DRAM ROW HIT latency is varied (15%)
- IPC and other DRAM related counters check when DRAM ROW MISS latency is varied (15%)
- IPC and other DRAM related counters check when MSHR size is varied (10%)
- IPC difference within 1% is acceptable for above cases.
- Using very short traces test the following items (40%)
    - check how MSHR full is handled
    - check how non-blocking is handled
    - check how store-load forwarding is handled
    - check how requests are merged

Partial Grading
If you implement a blocking cache with a fixed memory latency (use KNOB_DCACHE_FIXED_PENALTY) you will get 20%.
If you implement a non-blocking cache with with a fixed memory latency, you will get 50%.

Late Policy
Given that school was close for two days due to bad weather, the submission deadline for this assignment is pushed to Tuesday night, instead of Sunday night. We will not be accepting a late submission for this assignment as this may adversely impact your preparation for the first midterm (scheduled to be on Feb 18).

Submission Guide
Please do not turn in pzip files(trace files). Trace file sizes are quite huge and they will cause problems with the disk quota of

T-square.
(Tar the lab2 directory. Gzip the tarfile and submit lab2.tar.gz file at T-square)
cd lab2
make clean
rm *.pzip
cd ..
tar cvf lab2.tar lab2/*.h lab2/*.cpp
gzip lab2.tar

Part 2: Report (20%)
Include your simulation results in a report. Please note that there are many simulation cases so it will take several hours to simulate all of them.

The default configuration is

MSHR size: 4
cache size, way: 512KB 4 way
Dcache latency: 5 cycles
DRAM row buffer hit latency: 100 cycles
DRAM row buffer miss latency: 200 cycles
MSHR size:4, DRAM_BANK_NUM: 4, DRAM_PAGE_SIZE: 2KB
Use 2 traces: mem-trace.pzip (memory intensive trace), non-mem-trace.pzip (non-memory intensive traces)

- (10 points) Vary the cache size (128KB, 512KB, 1MB, and 2MB) and measure IPC and cache hit ratio. Discuss the performance improvements. Estimate a working set size based on the evaluated four points.
  Where you see a big cache hit ratio increment when you increase the cache size is the estimated working set size.

- (10 points) Vary the set associativity from (1, 2, 4, 8, 16 ways) and measure IPC and cache hit ratio. Discuss the performance improvements.

Submission Guide
Your submission file name MUST be report.pdf and you must include your name and T-square account name . If you fail to put your name and your file name is different than "report.pdf", your report won't be graded .
