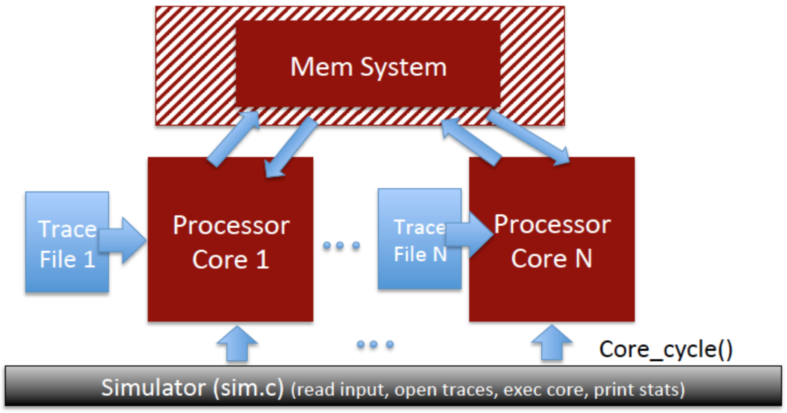
ECE 4100 / ECE 6100 / CS 4290 / CS 6290
Advanced Computer Architecture

Lab 4: CMP Memory System Design (10 Pts)

Parts A + B (+ C) Due: Friday, November 20, 2015 (11:55 pm)
Part D + E Due: Wednesday, November 25, 2015 (11:55 pm)

As part of this lab assignment you will build a multi-level cache simulator with DRAM based main memory. The system will then be extended to incorporate multiple cores, where each core has a private L1 (I and D) cache, and a shared L2 cache. Misses and writebacks from the shared L2 cache are serviced by a DRAM based main memory consisting of **16 banks** and per-bank row buffers.



We will build this simulator in two phases. First Phase (A, B, and C) are for single core system and the Second Phase (D, and E) are to extend the system to consist of multiple cores.

Part A: Design a Standalone Cache Module (2 points)

In this part, you will estimate the cache miss rate for the DCACHE. As we are only interested in cache hit/miss information we will not be storing data values in our cache. We will provide you with traces for 100M instructions from three SPEC2006 benchmarks: bzip2, lbm, and libq.

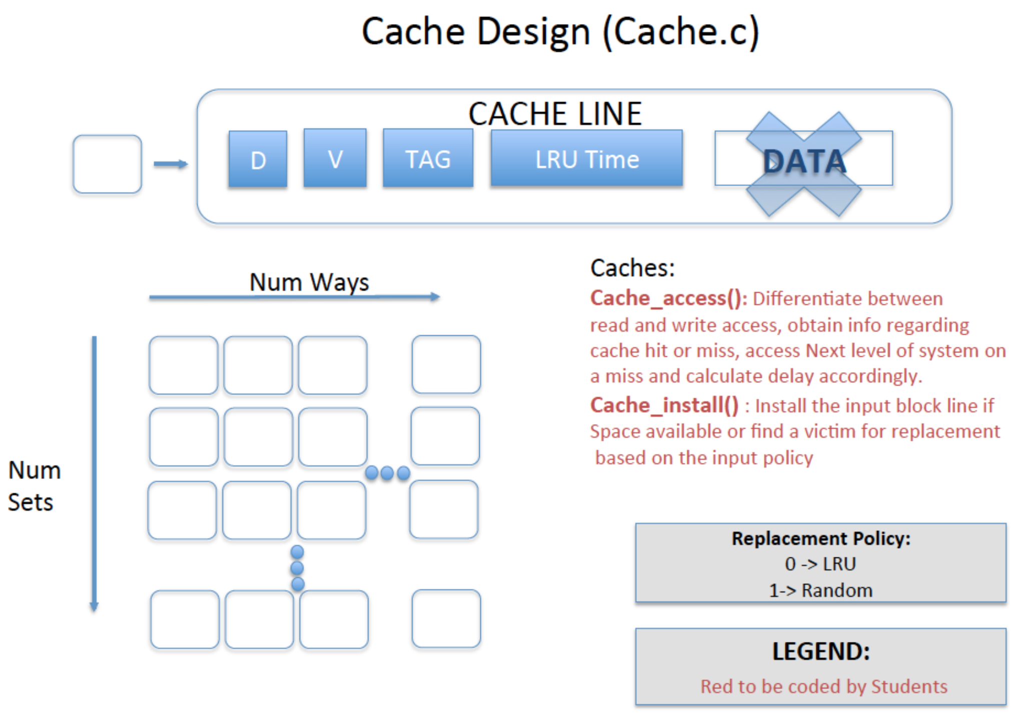
Each trace record consists of 4 bytes of instruction PC, 1 byte of instruction type (0: ALU 1:LOAD 2:STORE) and 4 bytes of virtual address for LD/ST instructions.

We will provide you with the trace reader and the ability to change cache parameters from command line. The tracer calls the memory system for LD ST instructions and the function in the memory system in turn calls the **cache_access** function for the DCACHE. If the line is not found in the DCACHE the memsys function calls the **cache_install** function for the DCACHE, which in turn calls **cache_find_victim**. At the end, the memory system also calls the cache_print_stats function for the DCACHE. The cache init function and cache print stats functions are already written for you.

Your job is to write three functions in cache.cpp

1. **cache_access** – If the line is present in the cache if yes return HIT
2. **cache_install** – install the line in the cache, and track evicted line
3. **find_cache_victim** – find the victim to be evicted

You will modify ONLY these three functions in cache.cpp and leave all other files untouched (except for run.sh). The cache functions must be written such that it will work for different associativity, linesize, cache size, and for different replacement policies (such as LRU and Random).


Cache Design (Cache.c)
CACHE LINE
D
V
TAG
LRU Time
DATA
Num Ways
Num Sets
Caches:
Cache_access(): Differentiate between read and write access, obtain info regarding cache hit or miss, access Next level of system on a miss and calculate delay accordingly.
Cache_install() : Install the input block line if Space available or find a victim for replacement based on the input policy
Replacement Policy:
0 -> LRU
1-> Random
LEGEND:
Red to be coded by Students

Part B: Multi-level Cache
(2 points for ECE 6100/ CS 6290, 3 points for ECE 4100/CS 4290)

You will implement an ICACHE, DCACHE, and connect it to a unified L2 cache and DRAM. You will also estimate timing for each request in this part.

Your job is to write two functions in memsys.cpp

1. **memsys_access_modeBC**, which returns the delay required to service a given request.
2. **memsys_L2_access,** which is called by memsys_access_modeBC and returns the delay for servicing a request that accesses the L2 cache.

Note that for the purpose of calculating delay, you can assume that writebacks are done off the critical path and hence do not account for the accumulation of delay for a read request.

NOTE: All caches are Write-back and Allocate-On-Write-Miss

Part C: Implementing Simple DRAM
(2 points for ECE 6100 / CS 6290, optional (extra credit) for ECE 4100 / CS 4290)

For Part C, you will model a simple row buffer for the DRAM memory system. You will assume that the DRAM memory has **16 banks** and the memory system employs an open page policy.

Your job is to write one function in dram.cpp
memsys_access_modeCDE which returns the dram delay. By default it returns a fixed value.

Part D: Making the system Multicore
(2 points for ECE 6100 / CS 6290, 3 points for ECE 4100 / CS 4290)

You will conduct the effectiveness of your memory system for a multicore processor. In particular, you will implement a two-core processor and evaluate your design for three mix workloads : Mix1 (libq-bzip2) Mix2(libq-lbm) and Mix3(bzip2-lbm). You will model a simple LRU based replacement in the shared L2 cache and report the weighted speedup under LRU replacement. Pay attention to the memory traffic and the memory row buffer hit rate for the mix workload compared to the isolated workloads from Part C.

Your job is to write two functions in memsys.cpp

1. **memsys_access_modeDE**, which returns the delay required to service a given request.
2. **memsys_L2_access_multicore,** which is called by memsys_access_modeDE and returns the delay for servicing a request that accesses the L2 cache.

Part E: Implement Static Way Partitioning (2 points)

Shared caches can cause a badly behaving application to completely consume the capacity of the shared cache, and cause significant performance degradation to the neighboring application. This can be mitigated with way partitioning, whereby each core can be given a certain number of ways per set as their quota. In this part you will implement static way partitioning for a system consisting of two cores. We will provide "SWP_core0ways" as the quota for core 0 (the remaining N-SWP_core0ways becomes the quota of core 1).

Your job is to update **cache_find_victim** in cache.cpp to handle SWP.

Part F: To be announced soon (3 points Extra Credit for everyone)
We will announce this part later.

How to get run the simulator

1. Download the tarball “Lab_4.tar.gz”
2. $tar –xvzf Lab_4.tar.gz
3. cd Lab_4
4. cd src
5. make
6. ./sim –h (to see the simulator options)
7. ./sim mode 1 ../traces/*.mtr.gz   (to test the default configuration)
8. ../scripts/runall.sh runs all configurations for all traces

WHAT TO SUBMIT FOR Parts A+B+C:

For Phase 1 (parts A, B,C) you will submit 2 folders:

1. src.tar.gz with updated cache.cpp, memsys.cpp and dram.cpp
2. results.tar.gz (which is a tarball of your results for the three parts, for each of the three trace files, so nine .res files)

WHAT TO SUBMIT FOR Parts D + E (and later F)

For Phase 2 (parts D, E, F) you will submit 2 folders:

1. src.tar.gz with updated cache.cpp, memsys.cpp and dram.cpp
2. results.tar.gz (which is a tarball of your results for the three parts, for each of the three trace files, so nine .res files)

FAQ:

1. You can use the timestamp method to implement LRU replacement. We have provided cycle_count as a means for tracking the time.

2. You will need the last_evicted line for Part B, when you will need to schedule a writeback from the Dcache to the L2cache, and from the L2cache to DRAM.

3. For RANDOM replacement, you may get a minor change in the miss rate compared to what is shown in the report.