

## ECE 463/521: SPRING 2017: PROJECT 1: CACHE HIERARCHY DESIGN

Due date: Tuesday, Feb 28th, 11:59 PM

Submission: Electronic submission via course page in Moodle

### 1. Ground rules

- This is an individual project.
- You can discuss the basic concepts with other students, however sharing code is strictly prohibited. A TA will scan source code from current and past semesters through automated tools that can detect code reuse. Source code that is flagged by these tools will receive appropriate action in accordance with university policy and referral to the Office of Student Conduct.
- Use the tag Project1 in the Discussion Forum for questions/comments/replies.
- Don't submit any code in the Discussion Forum.
- You must use C/C++ programming languages. You may also use Java but if you want to use Java make sure to inform the TA upfront.
- Compatibility with EOS Linux environment is required.

### 2. Project Summary

In this project, you will implement a cache hierarchy simulator. Your cache simulator must estimate the cache miss rates for different cache configurations when running different workloads. Different applications have different memory behavior. Accordingly, you will be provided with memory access traces derived from various benchmark applications. The rest of this document describes precisely what you will implement (Specification), how you will test your simulator (Validation runs), how you will communicate your findings (Report) and what you will submit (Report, simulator source code, etc.).

*Note: There are differences in requirements for ECE463 and ECE521 as detailed in the project description below. The project scope is significantly reduced for ECE463 students.*

### 3. Specification

Model a cache with parameterized geometry, replacement policy and inclusion property. For simplicity, assume that the cache access is atomic; that is, no new cache accesses will occur until the current access is served. Also, assume that write policy in effect is the Write-Back Write Allocate (WBWA) policy.

*ECE521 students are required to implement a two-level cache hierarchy (i.e., L1 and L2). ECE463 are required to implement only one-level cache (i.e., only L1).*

#### 3.1.1. Replacement Policy

Model a generic cache hierarchy with the following replacement policies supported. For all of these policies assume that when invalid cache blocks (aka, cache lines) are present they are selected as replacement candidates (for consistency in results just stick to picking the left most invalid cache line,

i.e. the lowest numbered way with the invalid cache line). If invalid cache lines are not present, use the replacement policy in effect.

- **LRU**: In the LRU replacement policy the least recently used block is evicted in order to make room for an access that misses the cache. A block that is accessed on a hit or installed on a miss becomes the most recently used. The LRU ranking of all the blocks is adjusted.
- **FIFO**: In the FIFO replacement policy the block which is the oldest to have been inserted into the set from amongst all the available blocks is the one that is evicted.

### 3.1.2. Inclusion Property

*This is only applicable to ECE521 students. ECE521 students should additionally implement inclusive, non-inclusive and exclusive L2 caches per the following description:*

- **Inclusive L2 caches**: These caches maintain the invariant that any block that exists in the L1 cache be also present in the L2 cache. This leads to back-invalidations. Back-invalidations are needed when a block is evicted from L2 Cache and there is still a copy of that block in the L1 Cache. The block at L1 will be invalidated to ensure that the L1 never has any block that the L2 does not have. To keep it simple it is OK to send a back invalidation to the L1 upon every L2 eviction (without having to explicitly track at the L2 cache whether or not a block is in the L1 cache).
- **Non-Inclusive L2 caches**: This is an intermediate policy which does not require a block in the L1 be also cache present in the L2 cache, although it may often do so. In that regard, a non-inclusive L2 cache behaves like an inclusive L2 cache but without back invalidation.
- **Exclusive L2 caches**: These caches maintain the invariant that a block will never reside in both the L1 and the L2. Whenever an access misses in L1 cache, the L2 cache is looked up. If it is a miss in the L2 the block is brought into the L1 from main memory without installing it in the L2. If the access were a hit in the L2 the block is brought into the L1 from the L2 and is invalidated in the L2. Note that the only way the L2 gets populated in this case is when the L1 evicts a block (whether clean or dirty), effectively making the L2 a victim cache. We will still use the term writeback only for the blocks that are dirty when evicted. However, the total writes to L2 will be the sum of writebacks and clean evictions from the L1.

## 4. Simulator Input

The simulator reads in a trace file in the following format:

```
r|w <hex address>
r|w <hex address>
...
```

The first argument is the operation. The character “r” means it is a read operation and the character “w” indicates a write operation. The second argument is the accessed address in hexadecimal format. The memory is byte addressable and the addresses are 48 bits long. To obtain the block address, block offset, tag and index you need to properly mask the address. Here is a sample trace:

```
r ae87c3c54d00
w ae87d0746340
r ae87d0746340
r ae9f7c125440
```

## 5. Simulator Output

Your simulator should output the following metrics at the end of a run. *See validation runs for the exact format.*

1. Memory hierarchy configuration and trace filename.
2. The following measurements:
  - a. Number of L1 reads

- b. Number of L1 read misses
- c. Number of L1 writes
- d. Number of L1 write misses
- e. L1 miss rate  

$$= (L1 \text{ read misses} + L1 \text{ write misses}) / (L1 \text{ reads} + L1 \text{ writes})$$
- f. Number of writebacks from L1  
 dirty evictions from the L1 (with an Exclusive L2, clean L1 evictions are also written to L2 but don't count those here)
- g. Number of L2 reads  

$$= L1 \text{ read misses} + L1 \text{ write misses}$$
- h. Number of L2 read misses
- i. Number of L2 writes  

$$= \text{writebacks from L1, in case of an Inclusive or Non-inclusive L2}$$

$$= \text{writebacks from L1} + \text{clean L1 evictions, in case of an Exclusive L2}$$
- j. Number of L2 write misses
- k. L2 miss rate  

$$= (L2 \text{ read misses} + L2 \text{ write misses}) / (L2 \text{ reads} + L2 \text{ writes})$$
- l. Number of writebacks from L2 to memory
- m. Total memory traffic (or the number of blocks transferred to/from memory)  
 Assuming the presence of an L2, this should match L2 read misses + L2 write misses + writebacks from L2 to memory, in case of a Non-inclusive or Exclusive L2 cache. In case of an Inclusive L2, if the blocks evicted from the L1 as the result of the back invalidation happen to be dirty, those should also be taken into account as writes to the memory (since they hold more recent data than what the memory contains).

## 6. Validation

### 6.1. Validation requirements

Sample simulation outputs are provided on the website. Each validation output provided includes:

- o The cache configuration to use
- o All measurements described in Section 5

You must run your simulator and debug it until it matches these provided *validation outputs*. Your simulator must print outputs to the console (i.e., to the screen). Your output *must* match both numerically and in terms of formatting because the TA will literally “diff” your output with the correct output. Therefore, redirect the console output of your simulator to a temporary file. This can be achieved by placing

```
> <your_output_file>
```

after the simulator command. And then make sure you test whether or not your outputs match the expected output, by running this unix command:

```
diff -iw <your_output_file> <posted_validation_file>
```

The `-iw` flag tells diff to ignore case (uppercase vs. lowercase) and whitespace. Therefore, just make sure there is some whitespace (such as a tab) where you see whitespace in the validation output. If the above command returns without printing anything to the screen, your validation was successful. Do this for each validation output provided.

### 6.2. Compiling and Running the Simulator

Part of what you need to hand in is source code. The TA will compile and run your simulator (i.e., source code). As such, you must meet the following strict requirements. Failure to meet these requirements will result in point deductions (see Section 10).

1. You *must* be able to compile and run your simulator on Eos Linux machines. This is required so that the TA can compile and run your simulator. If you are logging into an Eos machine remotely and do not know whether or not it is Linux (as opposed to SunOS), use the “uname” command to determine the operating system.
2. Along with your source code, you must provide a Makefile that automatically compiles the simulator. This Makefile must create a simulator named *sim\_cache*. The TA should be able to type only *make* (in the directory containing your sources and Makefile) and the simulator should compile successfully. The TA should be able to type only *make clean* to automatically remove object files and the simulator executable. An example Makefile will be posted on the web page, which you can copy and modify for your needs.
3. Your simulator must accept exactly 8 command-line arguments in the following order:  
`sim_cache <BLOCKSIZE> <L1_SIZE> <L1_ASSOC> <L2_SIZE> <L2_ASSOC> <REPL_POLICY> <INCLUSION> <TRACE_FILE>`
  - o BLOCKSIZE: Positive int. Block size in bytes (assumed to be same for all caches)
  - o L1\_SIZE: Positive int. L1 cache size in bytes.
  - o L1\_ASSOC: Positive int. L1 set-associativity (1 is direct-mapped).
  - o L2\_SIZE: Positive int. L2 cache size in bytes; 0 signifies that there is no L2 cache.
  - o L2\_ASSOC: Positive int. L2 set-associativity (1 is direct-mapped).
  - o REPL\_POLICY: Positive int. 0 for LRU, 1 for FIFO.
  - o INCLUSION: Positive int. 0 for non-inclusive, 1 for inclusive and 2 for exclusive.
  - o TRACE\_FILE: Character string. Full name of trace file including any extensions.

*Note to ECE463 students. Since your simulator does not support L2 caches, it must accept a 0 input for the L2\_SIZE, L2\_ASSOC, and INCLUSION arguments above.*

## 7. Experiments and Report

You will be provided with synthetic traces that are generated from several SPEC CPU2006 benchmarks (MCF, GCC and LBM). The traces are provided to you on the project webpage. For your report, plot the required figures as detailed below for each of the following traces: MCF, GCC and LBM. You will have to use a combination of your simulator output as well as latency reports from a cache simulator called CACTI in order to generate a set of plots and discuss insights from your findings. CACTI results are summarized and provided to you as a spreadsheet on the project webpage; you will not be required to run CACTI yourself. The metrics of interest are the miss rates of L1 and, if applicable, the L2 cache, and the Average Memory Access Time (AMAT) for a given cache hierarchy.

The simulator output is set up to directly provide the miss rates.

The hit times for AMAT calculation are provided in the CACTI spreadsheet. The hit times in the CACTI spreadsheet should be used for both L1 and L2 caches.

The miss penalty for one-level or two-level cache hierarchy should be calculated using the following formula:

Miss Penalty = MemoryLatency + Blocksize/MemoryBandwidth

where:

MemoryLatency = 25ns

MemoryBandwidth = 16GB/s

The following formulas should be used to calculate the AMAT.

For a one-level cache hierarchy:

AMAT = Hit Time + Miss Rate × Miss Penalty

For a two-level cache hierarchy:

AMAT = Hit Time<sub>L1</sub> + Miss Rate<sub>L1</sub> × (Hit Time<sub>L2</sub> + Miss Rate<sub>L2</sub> × Miss Penalty)

### 7.1. Exploring the L1 Cache Design

Assume that there is no L2 cache. Assume that LRU replacement policy is used for the L1 cache.

#### Plot 1

Fix the L1 block size at 64B and the L1 associativity at 4. Vary the cache size between 8KiB, 16KiB, 32KiB and 64KiB. Plot the L1 cache miss rates. The plot should have 4 data points per benchmark.

#### Plot 2

Fix the L1 block size at 64B and the L1 cache size at 32KiB. Vary the associativity between 1, 2, 4 and 8. Plot the L1 cache miss rates. The plot should have 4 data points per benchmark.

#### Further analysis and discussion based on Plots 1 and 2

Plot the Average Memory Access Time (AMAT) for the configurations from Plots 1 and 2 (you can plot the AMATs on a secondary y-axis within plots 1 and 2 or you may choose separate plots called 1a and 2a). Discuss how the cache size and associativity affect the AMAT. Does a lower L1 miss rate always result in a lower AMAT? Explain your answer. Try to find the best design that achieves a low AMAT.

### 7.2. Exploring the Replacement Policy

Assume that there is no L2 Cache.

#### Plot 3

Fix the L1 block size at 64B and the L1 associativity at 4. Vary cache size between 8KiB, 16KiB, 32KiB and 64KiB. For each configuration vary the replacement policy between LRU and FIFO. Plot the L1 Cache miss rates for each replacement policy. Note that for each benchmark this plot will have 2 data series (one per replacement policy) and each data series will have 4 data points (one per cache size). That is, 8 data points per benchmark.

#### Further analysis and discussion based on Plot 3

Discuss any interesting trends you see – make recommendations between the various replacement policies based on their complexity and the resulting miss rate.

### 7.3. Exploring the L2 Cache Design (This is only applicable to ECE521 students)

Fix block size at 64B, L1 cache size at 16KiB, L1 associativity at 4 and the replacement policy to be LRU. Assume that the L2 Cache is always non-inclusive.

#### Plot 4

Fix L2 associativity at 8. Vary the L2 cache size between 128KiB, 256KiB, 512KiB and 1MB. Plot the L2 cache miss rates. The plot should have 4 data points per benchmark.

#### Plot 5

Fix the L2 cache size at 128KiB. Vary L2 associativity between 1, 2, 4 and 8. Plot the L2 cache miss rates. The plot should have 4 data points per benchmark.

#### Further analysis and discussion based on Plots 4 and 5

Plot the Average Memory Access Time (AMAT) for the configurations from Plots 4 and 5. Plot, discuss and make recommendations similar to how you did for Plots 1 and 2.

### 7.4. Exploring the Inclusion Property choices (This is only applicable to ECE521 students)

Fix the block size at 64B, the L1 associativity at 4 and the L2 associativity at 8. Vary L1 cache size between 8KiB, 16KiB, 32KiB and 64KiB. For each resulting configuration, vary the L2 cache size between 128KiB, 256KiB, 512KiB and 1MB. There are a total of 16 configurations.

#### Plot 6

Plot the number of cache misses resulting in a memory read for each configuration for each inclusion policy (inclusive, exclusive and non-inclusive). Note that for each benchmark this plot will have 3 data series (one per inclusion policy) and each data series will have 16 points (one per configuration).

#### Plot 7

Plot the Average Memory Access Time (AMAT) for when using non-inclusive, inclusive and exclusive L2 caches for each configuration. The number of data points will be similar to Plot 6.

#### Further analysis and discussion based on Plots 6 and 7

Discuss interesting trends and observations. When do you think an exclusive cache can be better than a non-inclusive cache? When could a non-inclusive cache significantly outperform an inclusive cache? For this part, no design recommendations are required.

## 8. What to Submit via Wolfware

You must hand in a single zip file called **proj1.zip** that is no more than 1MB in size. Please respect the size limit on behalf of fellow students as the Wolfware/moodle submission space is limited. Notify the TA beforehand if you have special space requirements. However, a zip file of 1MB should be sufficient. Below is an example showing how to create proj1.zip from an Eos Linux machine. Suppose you have a bunch of **source code files** (\*.cc, \*.h), the **Makefile**, and your project report (**report.pdf**). Run “zip proj1 \*.cc \*.h Makefile report.pdf”

proj1.zip must only contain the following (any deviation may result in point deductions):

- Project report: This must be a single PDF document named report.pdf (or a single MS Word document called report.doc). The report must include a cover page. A sample cover page will be provided and will contain the project title, an honor pledge, and space for writing your full name as electronic signature of the honor pledge. See Section 7 for the content required in the report.
- Source code: Commented simulator source code; use any number of .cc/.h files, .c/.h files, etc.
- Makefile: A sample Makefile will be posted on the project webpage. See section 6.2 for what is expected of the makefile.

Note: Zip only the *files* listed above, *not* the directory containing these files. Do not use tar or tar.gz or anything else. It has to be a zip archive.

## 9. Grading

The following is the high-level break-up of the points awarded

- 50 points for substantial programming effort
- 30 points for correct validation
- 20 points for experiments and report

Validation will be scored as follows:

Item		Points for ECE463	Points for ECE521
L1 with LRU	Validation #0	5	3
	Validation #1	5	3
	Mystery A	5	3
L1 with FIFO	Validation #2	5	3
	Validation #3	5	3
	Mystery B	5	3
L1, L2 work	Validation #4	NA	2
	Validation #5	NA	2
	Mystery C	NA	2
L1, L2 inclusion property	Validation #6	NA	2
	Validation #7	NA	2
	Mystery D	NA	2

Experiments and reports will be scored as follows:

Item		Points for ECE463	Points for ECE521
Experiments and Report	Plot 1 + Discussion	6	2
	Plot 2 + Discussion	6	3
	Plot 3 + Discussion	8	3

	<a href="#">Plot 4 + Discussion</a>	NA	3
	<a href="#">Plot 5 + Discussion</a>	NA	3
	<a href="#">Plot 6 + Discussion</a>	NA	3
	<a href="#">Plot 7 + Discussion</a>	NA	3

Note: Mystery runs are validation configurations/runs, in addition to the sample validation runs which are provided to you, that a TA will use to test your simulator for correctness.

## 10. Deductions

- Late submissions are subject to points deduction in accordance with the “late assignment” policy detailed in the course syllabus.
- Up to -20 points for not complying with specific procedures. Follow all procedures very carefully to avoid penalties.

## 11. Other Recommendations

Please follow the proposed simulator design style and speed guidelines to make it easier for the TA and the instructor to help you when you have bugs in implementation.

### 11.1. Keep your design modular

Building a simulator in a modular fashion is a good programming practice. Modular design means that you implement different functions for different tasks rather than implementing everything in one long function. It also means you group or organize related data and methods into *structures* or *classes* and capture the behavior of similarly-behaving components with little code duplication.

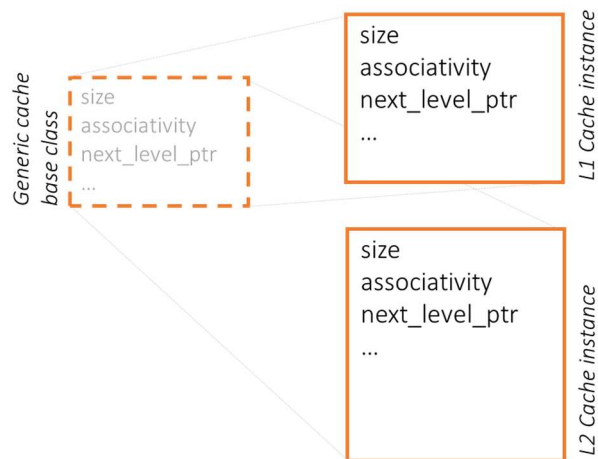


Figure 1: Sample Cache Hierarchy High Level Design

For example, different levels of the cache hierarchy may have different geometries and control policies, but at a high level they all are caches and to that extent may have the same high level methods and structural components. As a more specific example, notice that the L1 cache should have a function to locate the cache block to be evicted using a specific replacement policy. But every other cache level also needs to have a replacement policy, albeit a different policy. Further, whenever there is a hit at a cache level, the replacement policy related meta-data is modified, and whenever there is a miss at a cache level, the access is forwarded to the next level of cache.

It is strongly recommended that you observe these patterns of similarity (while being aware of the differences) and modularize the data structures that represents a generic cache. Use that as base class and allocate different instances (or derived class instances) of that class for each of the cache levels. [Figure Figure](#) shows an example of such design.

### 11.2. Keep your simulator parametric

Always assume that parameters are changeable. As an example, you should not assume a specific cache size or block size. Such parameters should be parsed from the input of the simulator and then used to instantiate an instance of the generic class.

### 11.3. Keep backups

It is good practice to frequently make backups of all your project files, including source code, your report, etc. You can backup files to another hard drive (Eos account vs. home PC vs. laptop vs. USB memory stick etc.). Just keep consistent copies in multiple places.

### 11.4. Make your simulator fast

Correctness of your simulator is of paramount importance. That said, making your simulator *efficient* is also important for a couple of reasons. First, the TA needs to test every student's simulator. Therefore, we are placing the constraint that your simulator must finish a single run in 2 minutes or less. If your simulator takes longer than 2 minutes to finish a single run, please see the TA. Second, you will be running many experiments: many cache configurations and multiple traces. Therefore, you will benefit from implementing a simulator that is reasonably fast. One simple thing you can do to make your simulator run faster is to compile it with a high optimization level. The example Makefile posted on the web page includes the `-O3` optimization flag. Note that when you are debugging your simulator in a debugger (such as `gdb`), it is recommended that you compile without `-O3` and, instead, compile with `-g`. Optimization includes register allocation and register-allocated variables are often not displayed properly in debuggers, which is why you want to disable optimization when using a debugger. The `-g` flag tells the compiler to include symbols (variable names, etc.) in the compiled binary. The debugger needs this information to recognize variable names, function names, line numbers in the source code, etc. When you are done with debugging, recompile with `-O3` and without `-g`, to get the most efficient simulator again.

### 11.5. Use the VCL

In addition to using the `grendel` cluster, and other Eos linux machines, NCSU's Virtual Computing Lab. (VCL) allows you to reserve virtual linux machines. Go to <http://vcl.ncsu.edu/>

1. select "Reservation > New Reservation",
2. you'll see a popup asking for environment and duration,
3. for the environment, select "Linux Lab Machine (Realm RHEnterprise 6)"
4. you may choose to reserve a shell for up to 4 hours beginning now or later (with the possibility to extend reservation before it expires),
5. click "Create Reservation" button,
6. wait until the screen updates with a reservation and click "Connect!",
7. you will be provided an internet address (IP) that you can use to access a machine through `putty` or `x-win` as you would usually do to securely login to other linux machines. (Note that VPN has become mandatory, so before invoking `putty` or `x-win`, make sure your VPN connection is up and running.).