IIT Madras
Department of Computer Science and Engineering

CS6600: July-Nov '24
Project 1: Cache and Memory Hierarchy Design
Due: Friday, September 27 at 11:59 PM

1. Ground rules

1. All students must work alone for this project.
2. Sharing of code between students is considered cheating. The TAs will scan source code through various tools available to us for detecting cheating. Source code that is flagged by these tools will receive the following actions:
   - Zero credits for the project.
   - Final grade will be two grades lower than the actual grade obtained by the student.
3. Students are encouraged to engage in white board discussions, which is an essential aspect of the project.
4. It is recommended that you do all your work in the C or C++ languages. Exceptions (in rare cases) must be approved by the faculty.
5. Students must get the code to work on the provided Docker.

2. Project Description

In this project, you will implement a flexible cache and memory hierarchy simulator and use it to compare the performance, area, and energy of different memory hierarchy configurations, using a subset of the SPEC-2000 benchmark suite.

3. Specification of Memory Hierarchy

Design a generic cache module that can be used at any level in a memory hierarchy. For example, this cache module can be instantiated as an L1 cache, an L2 cache, or a victim cache. Since it can be used at any level of the memory hierarchy, it will be referred to as CACHE throughout this specification.

3.1 Configurable Parameters

CACHE should be configurable in terms of supporting any cache size, associativity, and block size, which are specified at the beginning of simulation:

- SIZE : Total bytes of data storage.
- ASSOC : The associativity of the cache.
- BLOCKSIZE : The number of bytes in a block.

The following constraints are imposed on the above parameters:

- BLOCKSIZE is a power of two.
- The number of sets is a power of two.
- Note that ASSOC (and, therefore, SIZE) need not be a power of two.

3.2 Replacement Policy

CACHE should use the LRU (least-recently-used) replacement policy.

Note that when printing out the final contents of CACHE, you must print the blocks within a set based on their recency of access, i.e., print out the most recently used (MRU) block first, the next most recently used block second, and so on, and the least recently used (LRU) block last. This is required so that your output is consistent with the output of the TAs' reference simulator.

3.3 Write Policy

CACHE should support the WBWA (write-back + write-allocate) write policy.

- *Write-allocate:* A write that misses in CACHE will cause a block to be allocated in CACHE.
- *Write-back:* A write updates the corresponding block in CACHE, making the block dirty. It does not update the next level in the memory hierarchy (cache or memory). If a dirty block is evicted from CACHE, a writeback will be sent to the next level in the memory hierarchy.

3.4 Block Allocation: Sending requests to next level in the memory hierarchy

Your simulator must be capable of modeling one or more instances of CACHE to form an overall memory hierarchy, as shown in Fig. 1.

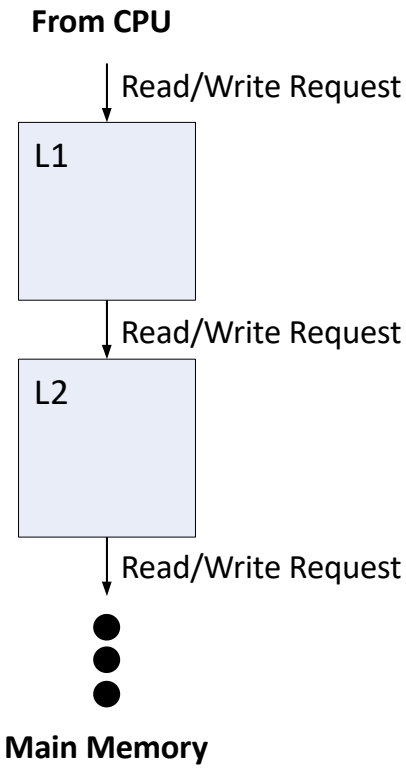

Fig. 1. General memory hierarchy consisting of multilevel caches followed by main memory.

CACHE receives a read or write request from whatever is above it in the memory hierarchy (either the CPU or another cache). The only scenario where the CACHE must interact with the next level below it (either another CACHE or main memory) is when the read or write request misses in CACHE. When the read or write request misses in CACHE, it must allocate the requested block so that the read or write operation can be performed.

3.5 State Updates

After servicing a read or write request, whether the corresponding block was in the cache already (hit) or had just been allocated (miss), remember to update the state information associated with the set/ block, as discussed in the lecture.

4. Augment CACHE with a Victim Cache

Students must additionally augment CACHE with a Victim Cache (VC). In this project, consider the VC to be an extension implemented within CACHE. This preserves the abstraction of one or more instances of CACHE interacting in an overall memory hierarchy, as shown in Fig. 1), where each CACHE may have a VC within it.

4.1 Victim Cache Enable/Disable Config

Your simulator should be able to specify, for each CACHE, whether or not its VC is enabled.

4.2 Victim Cache Parameters

The VC is fully-associative and must use the LRU replacement policy. The number of blocks in the VC should be configurable. A CACHE and its VC have the same BLOCKSIZE.

4.3 Interaction between CACHE and its VC

The only time CACHE and its VC may interact is when a read or write request misses in CACHE. In this case, the requested block X was not found in CACHE. If the corresponding set has at least one invalid block (*i.e., the set is not full*), then this set has never transferred a victim block to VC, and therefore, the VC cannot possibly have the requested block X and need not be searched. In this special case, CACHE need not interact with VC and instead goes directly to the next level. However, if the set does NOT have any invalid blocks (*i.e., the set is full*), VC needs to be searched for the requested block X. In the event that VC has the block X, the following actions need to be performed:

- A victim block V is singled out for eviction from CACHE.
- CACHE issues a “swap request [X, V]” to its VC, as discussed in the lecture.

On the other hand, if VC does not have the block X, the following steps need to be performed.

- VC must make space for block V from CACHE.
- CACHE issues a read of requested block X to the next level in the memory hierarchy.

5. Memory Hierarchies to be Investigated

You will only study the memory hierarchy configurations shown in Fig. 2. Also, these are the only configurations the TAs will test. For this project, all CACHEs in the memory hierarchy will have the same BLOCKSIZE.

You will be implementing non-inclusive policy for L2 CACHE, as described below:

- A block that misses in both L1 and L2 will be allocated in L2 as well as L1.
- When a block is evicted from L2, there is no need to back invalidate the block from L1, since inclusion need not be guaranteed.

It should be straightforward to support the non-inclusive policy since you only need to correctly implement block allocation (described in section 3.4) within CACHE. If block allocation is implemented correctly, the memory hierarchy will automatically handle cascaded requests globally.

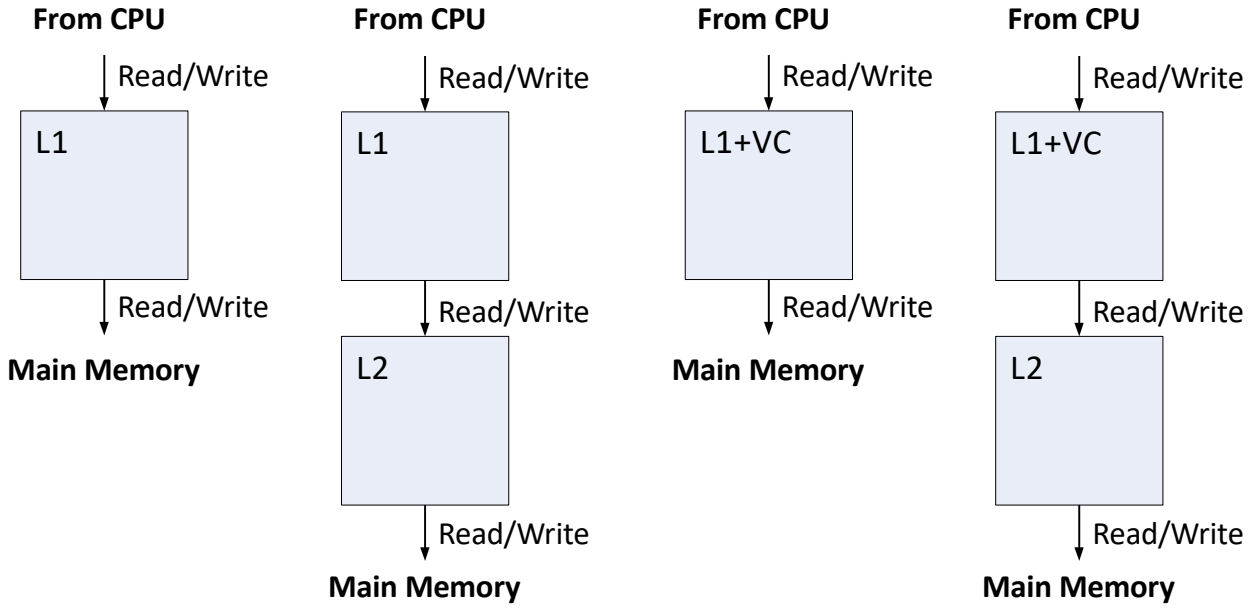

Fig. 2. Memory hierarchies to be investigated for the project.

6. Inputs to Simulator

The simulator reads a trace file in the following format:

r | w <hex address>
r | w <hex address>
…

Example trace:

r adedebef
w dead0f00
w dead0a00
r deed0a00

7. Outputs from Simulator

7.1 Raw Statistics

The simulator should print the following statistics after completion of the run:

a. number of L1 reads
b. number of L1 read misses
c. number of L1 writes
d. number of L1 write misses
e. number of swap requests from L1 to its VC (*0 if VC is disabled*)
f. swap request rate = SRR = (swap requests)/(L1 reads + L1 writes)
g. number of swaps between L1 and its VC (*0 if VC is disabled*)
h. combined L1+VC miss rate = MR_{L1+VC} =
(L1 read misses + L1 write misses – swaps)/(L1 reads + L1 writes)
i. number of writebacks from L1 or its VC (if enabled), to next level
j. number of L2 reads
k. number of L2 read misses
l. number of L2 writes
m. number of L2 write misses
n. L2 miss rate (from standpoint of stalling the CPU) = MR_{L2} = (L2 read misses)/(L2 reads)
o. number of writebacks from L2 to memory
p. total memory traffic = number of blocks transferred to/from memory
(*with L2, should match: L2 read misses + L2 write misses + writebacks from L2*)
(*w/o L2, should match : L1 read misses + L1 write misses – swaps + writebacks from L1/VC*)

7.2 Performance Metric: Average Access Time (AAT)

The formula to compute the Average Access Time (AAT) of L1, with and without L2, has been covered in the lecture. The following table (Table 1) describes how to obtain the latency parameters needed to computer AAT.

Parameters	**Description**	**How to get parameter**
HT_{L1}	Hit time of L1	Access time (in ns) can be obtained from CACTI tool. The CACTI executable is provided to you along with sample code that shows how to call the executable from your simulator and obtain the metric. Note that some VC configurations may be too small for CACTI to analyze. For the VC configurations that CACTI fail, use HT_{VC} = 0.2 ns.
HT_{L2}	Hit time of L2	
HT_{VC}	Hit time of VC	
Miss_Penalty	Time to fetch a block from main memory	Use Memory_Access_Latency = 20 ns and Peak Bandwidth = 16GB/s for computing the miss penalty.

Table 1. Access time parameters, description, and how to obtain them.

7.3 Performance and Power Metric: Energy Delay Product (EDP)

The energy-delay product is a combined metric that is used to account for both factors together, namely performance and power consumption. A combined metric is better because performance and power consumption are often at odds. Optimizing a combined metric leads to a balanced trade-off between performance and power consumption.

Energy-delay product is energy multiplied by time (delay). Since, ideally, we want the lowest possible energy consumed as well as the lowest possible execution time, the goal is to find a design that minimizes the energy-delay product.

For memory hierarchy without L2 cache:

Total energy =

(L1 reads + L1 writes)	$\times E_{L1}$ +
(L1 read misses + L1 write misses)	$\times E_{L1}$ +
2 × (swap requests)	$\times E_{VC}$ +
(L1read misses + L1write misses - swaps)	$\times E_{mem}$ +
(writebacks from L1/VC)	$\times E_{mem}$

Explanation of above expression:

- The first term accounts for accessing the L1 cache for all reads and writes.
- The second term accounts for putting a new block in the L1 for all L1 read and write misses.
- The third term accounts for the fact that all swap requests cause the VC to be accessed twice:
    - First, to search for the requested block X.
    - Second, to put the victim block V into the VC.
- The fourth term accounts for fetching blocks from main memory.
- The fifth term accounts for writing blocks to main memory.

For memory hierarchy with L2 cache:

Total energy =

(L1reads + L1writes)	$\times E_{L1}$ +
(L1read misses + L1write misses)	$\times E_{L1}$ +
2 × (swap requests)	$\times E_{VC}$ +
(L2 reads + L2 writes)	$\times E_{L2}$ +
(L2 read misses + L2 write misses)	$\times E_{L2}$ +
(L2 read misses + L2 write misses)	$\times E_{mem}$ +
(writebacks from L2)	$\times E_{mem}$

Explanation of above expression:

- The first term accounts for accessing the L1 cache for all reads and writes.
- The second term accounts for putting a new block in the L1 for all L1 read and write misses.
- The third term accounts for the fact that all swap requests cause the VC to be accessed twice:

    - First, to search for the requested block X
    - Second, to put the victim block V into the VC.
- The fourth term accounts for accessing the L2 cache for all L2 reads and writes.
- The fifth term accounts for putting a new block in the L2 for all L2 read and write misses.
- The sixth term accounts for fetching blocks from main memory.
- The seventh term accounts for writing blocks to main memory.

Finally, the expression for the Energy-Delay Product (EDP) is as follows. The unit of EDP is joules·seconds.
*EDP = (Total energy) × **(Total access time)***

Parameters	**Description**	**How to get parameter**
E_{L1}	Energy for single access to L1	"Total dynamic read energy per access (nJ)" from CACTI tool. The CACTI executable is provided to you along with sample code that shows how to call the executable from your simulator and obtain the metric.
E_{L2}	Energy for single access to L2	
E_{VC}	Energy for single access to VC	
E_{mem}	Energy for single access to main memory	Use E_{mem} = 0.05 nJ.

Table 2. EDP parameters, description, and how to obtain them.

7.4 Area Metric

For the memory hierarchy configurations investigated in this project:
Area = $Area_{L1}$ + $Area_{L2}$ (if L2 is enabled) + $Area_{VC}$ (if VC is enabled)

Area, for a given cache configuration, needs to be obtained from CACTI.

8. Validation Requirements

Sample simulation outputs will be provided to validate the correctness of your simulator. These are called validation runs. You must run your simulator and debug it until it matches passes all the validation runs.

Each validation run includes:

1. The memory hierarchy configuration.
2. The final contents of all caches in the memory hierarchy.
3. All measurements described in Section 7 (raw statistics, AAT, EDP, and Area).

Your simulator output must match both numerically and in terms of formatting, because the TAs will only *diff* your simulator's output with the correct output. You must confirm the correctness of your simulator by following these two steps for each validation run:

1. Redirect the console output of your simulator to a temporary file. This can be achieved by placing "> *your_output_file*" after the simulator command.
2. Test whether or not your outputs match properly, by running this unix command:
   diff –iw <your_output_file> <provided_validation_output_file>

The *–iw* flags tell *diff* to treat upper-case and lower-case as equivalent and to ignore the amount of whitespace between words. Therefore, do not need to worry about the exact number of spaces or tabs as long as there is some whitespace where the validation runs have whitespace.

8.1 Simulator Compilation and Execution Requirements

You will hand in source code, and the TAs will compile and run your simulator. You must meet the following requirements without fail:

1. You must be able to compile and run your simulator on the provided Docker.
2. You must provide a *Makefile* that automatically compiles the simulator. This *Makefile* must create a simulator named "*cache_sim*". The TAs should be able to type only "*make*" and the simulator will successfully compile. An example *Makefile* will be provided to you. Please feel free to modify it based on your needs.
3. Your simulator must accept the following command-line arguments in the specified order:

   cache_sim <*L1_SIZE*> <*L1_ASSOC*> <*L1_BLOCKSIZE*>
   <*VC_NUM_BLOCKS*>
   <*L2_SIZE*> <*L2_ASSOC*>
   <*trace_file*>

   - L1_SIZE : L1 cache size in Bytes.
   - L1_ASSOC : L1 set-associativity.
   - L1_BLOCKSIZE : L1 block size in Bytes.
   - VC_NUM_BLOCKS : Number of blocks in the Victim Cache (0 if no VC).
   - L2_SIZE : L2 cache size in bytes. (0 if no L2)
   - L2_ASSOC : L2 set-associativity.
   - trace_file : Character string specifying the full name of trace file.

   Example: If you want to simulate a memory subsystem configuration with L1-cache of size 1024 Bytes, associativity of 2, block size 16 Bytes, and L2-cache of size 8192 Bytes, associativity of 4, and Victim cache for L1 of size 16 blocks, then the command would be

   $ ./cache_sim 1024 2 16 16 8192 4 gcc_trace.txt

8.2 Compiling and Running the Simulator on the Docker

The following are the steps to be followed to compile and run the simulator on Docker.

Step 1 : Install Docker. For example, if you are installing on Ubuntu based machine, you can follow this: https://docs.docker.com/engine/install/ubuntu/. Docker desktop can also be used; in the latest version they come with integrated terminal in which the following commands can be run.

Step 2 : Build the docker image. For this, go to the folder containing Dockerfile and run:

$ docker build -t assign1 .
Where, assign1 is the name for the docker image.

Step 3 : Create a container and run. The command for it is

$ docker run -it assign1 /bin/bash

Step 4 : Now we will get a shell inside the docker container. If you run make, it should build the simulation executable.

$ cd Assignment_files
$ make
$ mkdir outputs
$ ./cache_sim 1024 2 16 0 0 0 gcc_trace.txt > outputs/gcc.output0.txt

Note: Docker container is a temporary instance, so any changes made is not saved.

9. Project Experiments and Report

You are required to run all the experiments on the (provided) GCC benchmark.

9.1 L1 Cache Investigation: SIZE and ASSOC

Plot #1

For this experiment, use BLOCKSIZE = 32, and assume that there are no Victim and L2 caches.

Plot L1 miss rate on the y-axis versus log2(SIZE) on the x-axis, for ten different cache sizes: SIZE = 2KB, ... , 1MB. The graph should contain five separate curves, one for each of the following ASSOCs: direct-mapped, 2-way set-associative, 4-way set-associative, 8-way set-associative, and fully-associative.

Discussion to include in your report:

1. Discuss trends in the graph.
2. Estimate the *compulsory miss rate* from the graph.
3. For each associativity, estimate the *conflict miss rate* from the graph.

Plot #2

Similar plot #1, but plot AAT on the y-axis instead of L1 miss rate.

Discussion to include in your report:

1. For a memory hierarchy with only an L1 cache and BLOCKSIZE = 32, which configuration yields the best AAT?

Plot #3

Carry out a similar experiment, as described for plot #2, with the following changes:

- Add the following L2 cache to the memory hierarchy: 256KB, 8-way set-associative, and the same block size as L1 cache.
- Vary the L1 cache size between 2KB and 128KB.

Discussion to include in your report:

- With the L2 cache added to the system, which L1 cache configuration yields the best AAT? What is the %improvement in this optimal AAT compared to the optimal AAT in plot #2?
- Compare the *EDP* and *total area* for the optimal-AAT configuration with L2 cache (plot #3) versus without L2 cache (plot #2).

9.2 L1 Cache Investigation: SIZE and BLOCKSIZE

Plot #4:

For this experiment, use ASSOC = 4 and vary both the SIZE and the BLOCKSIZE. Assume that there is no L2 cache.

Plot L1 miss rate on the y-axis versus log2(BLOCKSIZE) on the x-axis, for four different block sizes: BLOCKSIZE = 16, 32, 64, and 128. The plot should contain six separate curves, one for each of the following L1 cache sizes: SIZE = 1KB, 2KB, …, 32KB.

Discussions to include in your report:

- As block size is increased from 16 to 128, is the tradeoff between exploiting more spatial locality versus increasing cache pollution evident in the graph? Does the spatial locality-cache pollution trade-off depend on the cache size?
- What is the optimal BLOCKSIZE for a 4-way set-associative 8KB L1 cache?

9.3 L1 + L2 Co-optimization

Use the following configuration for this experiment:

- L1 cache: SIZE is varied, BLOCKSIZE = 32, ASSOC = 4.
- L2 cache: SIZE is varied, BLOCKSIZE = 32, ASSOC = 8.

Plot #5:

Create a 3D "surface" plot, with AAT (plotted on the z-axis) as a function of log2(L1 cache size) (plotted on the x-axis) and log2(L2 cache size) (plotted on the y-axis). Vary the L1 cache size between 1KB and 64KB, and L2 cache size between 32KB and 1MB. Make sure to only plot points for which the L1 cache is smaller than the L2 cache.

Discussions to include in your report:

- Which memory hierarchy configuration yields the best AAT?
- Can you propose a memory hierarchy configuration that has smaller total area, but provides an AAT within 5% of the best AAT?

Plot #6:

Create a plot similar to plot #5, but with EDP plotted on the z-axis instead of AAT.

Discussions to the include in the report:

- Which memory hierarchy configuration yields the best EDP?

9.4 Victim Cache Investigation

Use the following configuration for this experiment:

- L1 cache : SIZE is varied, ASSOC is varied, and BLOCKSIZE = 32.
- Victim cache : #entries is varied.
- L2 cache : SIZE = 256 KB, ASSOC = 8, and BLOCKSIZE = 32.

Plot #7:
Plot AAT on the y-axis versus log2(L1 SIZE) on the x-axis, for six different L1 cache sizes: SIZE = 1KB, 2KB, …, 32KB. The plot should contain seven separate curves, i.e. one for each of the following configurations:

- Direct-mapped L1 cache with no Victim Cache.
- Direct-mapped L1 cache with 4-entry Victim Cache.
- Direct-mapped L1 cache with 8-entry Victim Cache.
- Direct-mapped L1 cache with 16-entry Victim Cache.
- 2-way set-associative L1 cache with no Victim Cache.
- 4-way set-associative L1 cache with no Victim Cache.

Discussions to include in the report:

- Does adding a Victim Cache to a direct-mapped L1 cache yield performance comparable to a 2-way set-associative L1 cache of the same size?
- Which memory hierarchy configuration yields the best AAT?
- Can you propose a memory hierarchy configuration that has a smaller total area, but yields an AAT that is within 5% of the best AAT reported above?

10. Submission Requirements

You must submit a single zip file, named <StudentID>_assignment1.zip, which should contain the following:

- **Source code.** You must include all the source code files (.cc/.h or .c/.h)
- **Makefile.** The Makefile needs to be updated to compile all your source code files.
- **Project report.** This should be a single *report.doc* or *report.pdf* file, which contains the plots and the discussions.

11. Grading Policy

The grading policy for this project is described in the following table (Table 3).

Description	Points
Substantial programming effort that results in a simulator that builds, but reports inaccurate statistics or cache contents.	30
Working simulator that reports correct statistics and cache contents for the validation runs (provided *a priori* to the students) and the mystery runs (created by the TAs).	40 (10 points for L1 10 points for L1 and L2 10 points for L1 + VC 10 points for L1 + VC and L2)
Project report	30

Table 3. Grading policy for the cache and memory hierarchy project.