

Term Project Description

EEC 485 High Performance Architecture, Fall 2007

Title: Simulator for a MIPS processor

Goal: This project is designed to help understand the memory hierarchy and their operation principles of modern microprocessors.

Overview: This project is on cache and virtual memory (VM). It focuses not only on implementation but also on research. This project consists of four parts. Project #1 is to develop cache simulator and Project #2 is to investigate the effect of various cache parameters on cache performance. Project #3 is to expand the cache simulator to include VM and Project #4 is to investigate the effect of various VM parameters on VM performance.

Inputs to the cache/VM simulator are (i) a list of memory addresses accessed and (ii) cache/VM parameters. Outputs from the cache/VM simulator are cache/VM performance metrics such as cache hit ratio, page fault ratio and TLB misses.

You can use any programming language of your own choice (C, C++, Java, Excel, etc.) but make sure that your simulator runs on a Windows XP-based PC.

Project #1: Implementation of cache simulator, Due Sep. 24 (M)

- This project is to develop a cache simulator and to measure the cache performance (hit ratio, etc.) with different programs. Turn in 3~5-page project report and a cache simulator.

Detailed description:

- The cache simulator will run based on a memory trace file that is a list of memory addresses referenced during the execution of a program. The cache simulator should output cache statistics such as hit ratio, etc.
- Assume the following cache parameters.
 - 64KB cache and 64B cache block
 - Single-level cache (see pages 505-507)
 - Direct-mapped cache (see pages 496-504, 538-541)
 - Write-through policy (see pages 483-485, 541-542)
 - LRU replacement policy (see pages 504, 541)
 - Data/instruction integrated cache
- Your job is to build a simulator that reads a memory trace and simulates the action of a cache. Your simulator should keep track of which memory blocks are loaded into cache. As it processes each memory reference from the trace, it should check to see if the corresponding memory address is in the cache or not.
- Of course, this is just a simulation of cache, so you do not actually need to read and write data from memory. Just keep track of which blocks are in the cache.
- Write-through policy is assumed in this project. Since we do not actually read or write, write policy does not affect the cache performance such as hit ratio. However, as we will see in Project #2, the number of memory write events will be different depending on the write policy employed.

Memory trace file

- As an input to the cache simulator, you will use four memory traces. You can access the four trace files from <ftp://tracebase.nmsu.edu/pub/tracebase4/r3000/din>
 - 022.li.din.Z
 - 085.gcc.din.Z
 - 047.tomcatv.din.Z
 - 078.swm256.din.Z

- Each trace is a real recording of a running program, taken from the SPEC benchmarks. Each trace only consists of one million memory accesses taken from the beginning of each program. Each trace is compressed and must be uncompressed before use.
- Each trace is a series of lines, each listing three fields per line: “accesstype address size/data.” Address and size/data are hexadecimal. Accesstype is
 - 0 read**
 - 1 write**
 - 2 instruction fetch**
- Addresses are byte addresses. In other words, memory address 0000000c mean the 13th byte from the top of the memory. Every memory reference accesses a word data (4 bytes) regardless the size/data field. For example, “1 000000c 0” denotes that a word data is written on memory address 0000000c. In other words, data at memory address 0000000c, 0000000d, 0000000e and 0000000f will be written. For this reason, all memory addresses in the trace files are word-aligned.

Simulator requirements:

- You are free to structure and write your simulator in any reasonable manner. However, you need to make your simulator flexible enough to accommodate input parameters. For example, it may be good idea for your simulator to take the following arguments:
 - your_simulator_name <tracefile> ... <quite|debug>
- The first argument gives the name of the memory trace file. If the last argument is "quiet", then the simulator should run silently with no output until the very end. If it is "debug", then the simulator should print out messages displaying the details of each event in the trace.
- At the end of the simulation, the cache simulator should print out a few simple statistics like this:
 - Memory trace: 085.gcc.din
 - Total memory references in trace: 1002050 (read: 400000, write: 300000, fetch: 302050)
 - <Cache parameters>
 - Cache size: 64KB
 - Cache block size: 64 bytes
 - <Cache statistics>
 - Number of memory write events: 300000
 - Number of block placements: 1751
 - Number of block replacements: 1751
 -
 - Cache hit ratio: 95.5%

Turn in:

Project report (3~5 pages) should include

- Design and implementation of your cache simulator
- Instruction how to use your cache simulator
- Results and discussion of cache performance
- Additional features

Executable program and source files

- Submit your executable as well as source files in a 3.5-inch floppy diskette or a USB stick
- Make sure your executable runs on Windows XP platform.

Grading:

- 40 points, Correctness
- 10 points, Documentation
- 10 points, Bonus points (You need to clearly list additional features in your document)

Project #2: Research on cache organization, Due Oct. 12 (F)

- This project is to expand the cache simulator in Project #1 and to compare the cache performance (hit ratio, etc.) with different cache parameters such as cache size and placement policy. (Consider to answer Exercise 7.21 in the textbook.) Turn in 5~7-page project report and a cache simulator. Each student makes a powerpoint presentation on Oct. 12 (F) to discuss their findings regarding cache performance.

Detailed description:

- The cache simulator in Project #2 supports various cache configurations including the followings
 - Any cache size and any block size
 - Cache level (Single-level or Two-level; see pages 505-507),
 - Cache placement policy (Direct, 2-way set associative, 8-way set associative, Fully associative; see pages 496-504, 538-541),
 - Write policy (Write-back and Write-through; pages 483-485, 541-542),
 - Replacement policy (Random or LRU; see pages 504, 541),
 - Data/instruction integrated or separate cache
- As in Project #1, your job is to build a simulator that reads a memory trace and simulates the action of a cache. Your simulator should keep track of which memory blocks are loaded into cache. As it processes each memory reference from the trace, it should check to see if the corresponding memory address is in the cache or not.
- As in Project #1, different write policies do not produce different cache performance such as hit ratio. However, the number of memory write events will be much higher with write-through than write-back policy while each write event involves less number of bytes. In other words, write-through policy typically writes a word data per write event but write-back policy writes a block of data per write event.
- You are supposed to use four memory trace files as in Project #1 to obtain cache performance.
- Final report includes figures such as Fig. 7.8, 30, and 31 and the corresponding discussion.

Simulator requirements:

- You are free to structure and write your simulator in any reasonable manner. However, you need to make your simulator flexible enough to accommodate various cache parameters. For example, it may be good idea for your simulator to take the following arguments:
your_simulator_name <tracefile> <cachesize> <blocksize> ... <quite|debug>
- The first and the last argument specify the trace file and the debug mode, respectively, as in Project #1. The second argument gives the cache size in the simulated CPU. The third argument gives the cache block size, etc.
- At the end of the simulation, the cache simulator should print out statistics like this:
Memory trace: 085.gcc.din
Total memory references in trace: 1002050 (read: 400000, write: 300000, fetch: 302050)
<Cache parameters>
Cache size: 64KB
Cache block size: 64 bytes
Cache level: Single
Cache placement policy: 2-way set associative
Cache replacement policy: LRU
Cache write policy: Write-back
.....
<Cache statistics>
Number of memory write events: 300000
Number of block placements: 1751
Number of block replacements: 1751
.....
Cache hit ratio: 95.5%

Turn in:

Project report (5~7 pages) should include

- Design and implementation of your cache simulator
- Instruction how to use your cache simulator
- Results and discussion of cache performance with different cache parameters
- Additional features

Executable program and source files

- Submit your executable as well as source files in a 3.5-inch floppy diskette or a USB stick
- Make sure your executable runs on Windows XP platform.

Powerpoint presentation

- There will be 15-minute power presentation
- Be sure to bring a powerpoint file on a USB memory stick

Grading:

20 points, Support for variety of cache organization

20 points, Results and discussion on cache performance in the report

10 points, Presentation

10 points, Bonus points (You need to clearly list additional features in your document)

Project #3: Implementation of virtual memory (VM) simulator, due Nov. 7 (W)

- This project is to expand the cache simulator developed in Project #2 to include virtual memory (VM) system. Turn in 5~7-page project report and a VM simulator.

Detailed description:

- The VM simulator will run based on a memory trace file that is a list of memory addresses referenced during the execution of a program. The VM simulator should output VM performance such as page fault ratio and TLB misses as well as cache statistics such as hit ratio.
- Assume the following VM parameters.
 - 4MB memory and 4KB page
 - Single-level page table (see pages 515, 521)
 - LRU page replacement policy (see pages 516-519)
 - Fully-associative TLB with 16 entries (see pages 521-524, 531)
- Assume the following cache parameters.
 - 64KB cache and 64B cache block
 - Single-level cache (see pages 505-507)
 - Direct-mapped cache (see pages 496-504, 538-541)
 - Write through policy (see pages 483-485, 541-542)
 - LRU replacement policy (see pages 504, 541)
 - Data/instruction integrated cache
- Your job is to build a VM that reads a memory trace and simulates the action of a VM and a cache. Your simulator should keep track of which pages are loaded into memory as well as which memory blocks are loaded into cache. As it processes each memory reference from the trace, it should check to see if the corresponding memory address is loaded. If not, it should choose a page to remove from memory. Of course, if the page to be replaced is dirty, it must be saved to disk. Finally, the new page is to be loaded into memory from disk, and the page table is updated.
- Of course, this is just a simulation of the page table, so you do not actually need to read and write data from disk. Just keep track of what pages are loaded. When a simulated disk read or write must occur, simply increment a counter to keep track of disk reads and writes, respectively.
- To see how VM and cache work together as a hierarchy, please refer pages 524-528 in the textbook.
- You are supposed to use four memory trace files as in Project #1 and #2 to obtain VM/cache performance. Assume that addresses in the memory trace files are virtual addresses.

Simulator requirements:

- You are free to structure and write your simulator in any reasonable manner. However, you need to make your simulator flexible enough to accommodate input parameters. For example, it may be good idea for your simulator to take the following arguments:
your_simulator_name <tracefile> ... <quit|debug>
- The first and the last argument specify the trace file and the debug mode, respectively, as in Project #1.
- At the end of the simulation, the VM simulator should print out a few simple statistics like this:
Memory trace: 085.gcc.din
Total memory references in trace: 1002050 (read: 400000, write: 300000, fetch: 302050)
<VM/cache parameters>
Memory size: 4MB
Page size: 4KB
Cache size: 64KB
Cache block size: 64 bytes
<VM statistics>
Number of disk reads: 2155
Number of disk writes: 934
TLB hit ratio: 78.3%
Page fault ratio: 12.2%
...
<Cache statistics>

Number of memory write events: 300000
Number of block placements: 1751
Number of block replacements: 1751
.....
Cache hit ratio: 95.5%

Turn in:

Project report (5~7 pages) should include

- Design and implementation of your VM simulator
- Instruction how to use your VM simulator
- Results and discussion of VM performance
- Additional features

Executable program and source files

- Submit your executable as well as source files in a 3.5-inch floppy diskette or a USB stick
- Make sure your executable runs on Windows XP platform.

Grading:

40 points, Correctness

10 points, Documentation

10 points, Bonus points (You need to clearly list additional features in your document)

Project #4: Research on VM organization, due Nov. 30 (F)

- This project is to expand the VM simulator developed in Project #3 to assess the VM performance with different VM parameters such as total memory size, page table structure and TLB structure. (Consider to answer Exercise 7.38 in the textbook.) Turn in 7~9-page project report and a VM simulator. Each student makes a powerpoint presentation on Nov. 30 (F) to discuss their findings regarding VM performance.

Detailed description:

- The VM simulator in Project #4 supports various VM configurations including the followings
 - Any memory size and any page size
 - Page table structure (single-level or two-level; see pages 515, 521)
 - Page table replace policy (LRU or random; see pages 516-519)
 - TLB structure (number of entries, fully or set associative; see pages 521-524, 531)
- It also supports various cache configurations including the followings
 - Any cache size and any block size
 - Cache level (Single-level or Two-level; see pages 505-507),
 - Cache placement policy (Direct, 2-way set associative, 8-way set associative, Fully associative; see pages 496-504, 538-541),
 - Write policy (Write back and Write through; pages 483-485, 541-542),
 - Replacement policy (Random or LRU; see pages 504, 541),
 - Data/instruction integrated or separate cache
- As in Project #3, your job is to build a simulator that reads a memory trace and simulates the action of a VM and a cache. Your simulator should keep track of which pages are loaded into memory as well as which memory blocks are loaded into cache. As it processes each memory reference from the trace, it should check to see if the corresponding memory address is loaded. If not, it should choose a page to remove from memory.
- To see how VM and cache work together as a hierarchy, please refer pages 524-528 in the textbook.
- You are supposed to use four memory trace files as in Project #3 to obtain VM/cache performance. Assume that addresses in the memory trace files are virtual addresses.

Simulator requirements:

- You are free to structure and write your simulator in any reasonable manner. However, you need to make your simulator flexible enough to accommodate input parameters. For example, it may be good idea for your simulator to take the following arguments:
your_simulator_name <tracefile> <memorysize> <pagesize> <rand|lru> <quite|debug>
- The first and the last argument specify the trace file and the debug mode, respectively, as in Project #3. The second argument gives the total memory size in the simulated computer. The third argument gives the page size. The fourth argument gives page replacement policy, which is either random or LRU, etc.
- At the end of the simulation, the VM simulator should print out a few simple statistics like this:
Memory trace: 085.gc.din
Total memory references in trace: 1002050 (read: 400000, write: 300000, fetch: 302050)
<VM parameters>
Memory size: 4MB
Page size: 4KB
TLB organization: fully-associative
TLB size: 16 entries
Page table organization: single-level
Page replacement policy: LRU
.....
<Cache parameters>
Cache size: 64KB
Cache block size: 64 bytes
Cache level: Single
Cache placement policy: 2-way set associative

Cache replacement policy: LRU
Cache write policy: Write-back
.....
<VM statistics>
Number of disk reads: 2155
Number of disk writes: 934
TLB hit ratio: 78.3%
Page fault ratio: 12.2%
...
<Cache statistics>
Number of memory write events: 300000
Number of block placements: 1751
Number of block replacements: 1751
.....
Cache hit ratio: 95.5%

Turn in:

Project report (7~9 pages) should include

- Design and implementation of your VM simulator
- Instruction how to use your VM simulator
- Results and discussion of VM performance with different VM parameters
- Additional features

Executable program and source files

- Submit your executable as well as source files in a 3.5-inch floppy diskette or a USB stick
- Make sure your executable runs on Windows XP platform.

Powerpoint presentation

- There will be 15-minute power presentation
- Be sure to bring a powerpoint file on a USB memory stick

Grading:

- 20 points, Support for variety of VM/cache organization
- 20 points, Results and discussion on VM/cache performance in the report
- 10 points, Presentation
- 10 points, Bonus points (You need to clearly list additional features in your document)