Design Project: 3-Level Cache Implementation Using SimpleScalar

Zhe Chen, Yu Song
Department of Electrical and Computer Engineering
Tennessee Technological University
Cookeville, TN38505
12/04/2008
Goal

- To implement a 3-level cache (L1, L2, L3) using SimpleScalar with inclusion property
- To use at least 3 benchmarks to collect the cache hit/miss information for the following two configurations:
  - Configuration 1:
    - L1 D-cache: 16KB, L1 I-Cache: 16KB, direct mapped, block size: 32B
    - L2 cache: 512KB, 4-way, block size: 32B
    - L3 cache: 16MB, 8-way, block size: 32B
  - Configuration 2:
    - L1 D-cache: 32KB, L1 I-Cache: 32KB, 2-way, block size: 32B
    - L2 cache: 512KB, 4-way, block size: 128B
    - L3 cache: 4MB, 8-way, block size: 128B
Project Overview

- SimpleScalar (revision 3.0d)
  - http://simplescalar.com/
  - Sim-cache

- Development environments
  - Cygwin, for software development and benchmarking
  - Redhat Linux 9.0 + VirtualBox, for benchmarking

- Benchmarks
  - SPECint2000
  - SPECfp2000
  - X Benchmarks
  - Instructor benchmarks
Source Code Modification

- Source files
  - Sim-cache.c
  - Cache.c
  - Cache.h

- Modifications
  - Defined a macro (#define ECE7130_MODIFIED) for switching between the original code and the modified code
  - Created L3 cache contexts: cache structure pointers, cache option pointers, register cache stats, system call handler
  - Added command line parser supports for L3 cache
  - Added conditional L3-D and L3-I creation code
  - Applied cache inclusion property to L2 and L3
  - Integrated cache inclusion property verification code
Scripts

- Wrote shell scripts for facilitating the simulations
  - “testconfig” - call `sim-cache` with parameters for one configuration
  - “test” - call `testconfig` to run a single benchmark program for multiple configurations
  - “testbatch” - call `test` to run multiple benchmark programs for multiple configurations

- Add benchmark program file names in “testbatch”, run “testbatch” script, and wait for results. Results are saved in a series of text files with names from the benchmark programs.
Scripts

Testing codes format

- `#!/bin/bash`  
  `date`  
- `./test result_bzip200 ..../benchmarks/bzip200.outorderO2.gcc.100M.ss`  
  `date`

- `#!/bin/bash`  
  `date`  
- `./test result_anagram ..../benchmarks/anagram.alpha words anagram.in OUT`  
  `Date`
Simulations – SPECint2000

- SPECint2000 benchmarks
  - bzip200
  - gcc00
  - parser00
  - vortex
  - vpr00

- Configurations
  - Configuration 1: the same as the given configuration 1
  - Configuration 2: the same as the given configuration 2
  - Configuration 3: almost the same as the given configuration 2, except for smaller L2 cache size (256K) and block size (64B).
  - Configuration 4: almost the same as the given configuration 1, except for a bigger L3 cache block size (64B).
  - Configuration 3 and configuration 4 are just used for generating some comparison results
Miss Rates for Each Benchmark (1/2)
Miss Rates for Each Benchmark (2/2)

- Configuration 4 gets slightly lower miss rates than configuration 1 in most cases
  - Configuration 4 has a bigger L3 cache block size
- Configuration 3 gets higher miss rates than configuration 2 in all the cases
  - Configuration 3 has a smaller L2 cache size
Miss Rates for Each Configuration

Configuration 1

Configuration 2

Configuration 2 gets lower cache miss rate in most cases
Simulations – SPECfp2000

- SPECfp2000 benchmarks
  - Mesa00
  - art00

- Configurations
  - Configuration 1: the same as the given configuration 1
  - Configuration 2: the same as the given configuration 2
  - Configuration 3: almost the same as the given configuration 2, except for smaller L2 cache size (256K) and block size (64B).
  - Configuration 4: almost the same as the given configuration 1, except for a bigger L3 cache block size (64B).
  - Configuration 3 and configuration 4 are just used for generating some comparison results
Miss Rates for Each Benchmark

### mesa00

<table>
<thead>
<tr>
<th>Config</th>
<th>L1-I</th>
<th>L1-D</th>
<th>L2</th>
<th>L3</th>
</tr>
</thead>
<tbody>
<tr>
<td>config1</td>
<td>0.0027</td>
<td>0.1335</td>
<td>0.3849</td>
<td>0.5745</td>
</tr>
<tr>
<td>config2</td>
<td>0.0021</td>
<td>0.0954</td>
<td>0.1282</td>
<td>0.5728</td>
</tr>
<tr>
<td>config3</td>
<td>0.0013</td>
<td>0.0229</td>
<td>0.2575</td>
<td>0.2643</td>
</tr>
<tr>
<td>config4</td>
<td>0.0027</td>
<td>0.1335</td>
<td>0.3049</td>
<td>0.5745</td>
</tr>
</tbody>
</table>

### art00

<table>
<thead>
<tr>
<th>Config</th>
<th>L1-I</th>
<th>L1-D</th>
<th>L2</th>
<th>L3</th>
</tr>
</thead>
<tbody>
<tr>
<td>config1</td>
<td>0.0646</td>
<td>0.1023</td>
<td>0.3085</td>
<td>1</td>
</tr>
<tr>
<td>config2</td>
<td>0.0545</td>
<td>0.1001</td>
<td>0.2467</td>
<td>1</td>
</tr>
<tr>
<td>config3</td>
<td>0.0545</td>
<td>0.1001</td>
<td>0.5499</td>
<td>0.5609</td>
</tr>
<tr>
<td>config4</td>
<td>0.0646</td>
<td>0.1023</td>
<td>0.4502</td>
<td>1</td>
</tr>
</tbody>
</table>

### Configuration 1

<table>
<thead>
<tr>
<th>Miss Rate</th>
<th>L1-I</th>
<th>L1-D</th>
<th>L2</th>
<th>L3</th>
</tr>
</thead>
<tbody>
<tr>
<td>config1</td>
<td>0.0646</td>
<td>0.1023</td>
<td>0.3085</td>
<td>1</td>
</tr>
<tr>
<td>config1</td>
<td>0.0545</td>
<td>0.1001</td>
<td>0.1467</td>
<td>1</td>
</tr>
<tr>
<td>config2</td>
<td>0.0027</td>
<td>0.1335</td>
<td>0.3849</td>
<td>0.5745</td>
</tr>
<tr>
<td>config2</td>
<td>0.0021</td>
<td>0.0954</td>
<td>0.1202</td>
<td>0.5728</td>
</tr>
</tbody>
</table>

### Configuration 2
Simulations – X Benchmarks

- Xbenchmarks
  - xanim

- Configurations
  - Configuration 1: the same as the given configuration 1
  - Configuration 2: the same as the given configuration 2
  - Configuration 3: almost the same as the given configuration 2, except for smaller L2 cache size (256K) and block size (64B).
  - Configuration 4: almost the same as the given configuration 1, except for a bigger L3 cache block size (64B).

Configuration 3 and configuration 4 are just used for generating some comparison results.
Miss Rates for Each Benchmark

<table>
<thead>
<tr>
<th>config</th>
<th>L1-I</th>
<th>L1-D</th>
<th>L2</th>
<th>L3</th>
</tr>
</thead>
<tbody>
<tr>
<td>config1</td>
<td>0.0115</td>
<td>0.0626</td>
<td>0.7062</td>
<td>1</td>
</tr>
<tr>
<td>config2</td>
<td>0.011</td>
<td>0.0487</td>
<td>0.3159</td>
<td>1</td>
</tr>
<tr>
<td>config3</td>
<td>0.0011</td>
<td>0.0487</td>
<td>0.5299</td>
<td>0.5963</td>
</tr>
<tr>
<td>config4</td>
<td>0.0115</td>
<td>0.0626</td>
<td>0.7062</td>
<td>1</td>
</tr>
</tbody>
</table>
Simulations – Instructor Benchmarks

- Instructor benchmarks
  - anagram.alpha

- Configurations
  - Configuration 1: the same as the given configuration 1
  - Configuration 2: the same as the given configuration 2
  - Configuration 3: almost the same as the given configuration 2, except for smaller L2 cache size (256K) and block size (64B).
  - Configuration 4: almost the same as the given configuration 1, except for a bigger L3 cache block size (64B).
  - Configuration 3 and configuration 4 are just used for generating some comparison results
Miss Rates for Each Benchmark

<table>
<thead>
<tr>
<th>Configuration</th>
<th>L1 I</th>
<th>L1 D</th>
<th>L2</th>
<th>L3</th>
</tr>
</thead>
<tbody>
<tr>
<td>config 1</td>
<td>0.0895</td>
<td>0.1508</td>
<td>0.8811</td>
<td>1</td>
</tr>
<tr>
<td>config 2</td>
<td>0.0837</td>
<td>0.1414</td>
<td>0.3556</td>
<td>1</td>
</tr>
<tr>
<td>config 3</td>
<td>0.0837</td>
<td>0.1414</td>
<td>0.5907</td>
<td>0.602</td>
</tr>
<tr>
<td>config 4</td>
<td>0.0895</td>
<td>0.1508</td>
<td>0.8811</td>
<td>1</td>
</tr>
</tbody>
</table>
For miss rate, maybe some of the exact number are not right, but the trend is right as the diagrams shows, the higher level the cache, the higher miss rate it shows.

Configuration 2 is the best among all the configuration regarding to the miss rates, it is not surprising since it has has larger L1 cache and more ways.

For each benchmark suite, we tested at least one program.