**Implementing a 3-Level Cache Using SimpleScalar**

**System Design Report**

Honghao Gan & Zhijia Chen

## SimpleScalar background

SimpleScalar is a toolset that models a virtual computer system with CPU, cache and memory hierarchy. It allows users to build modeling applications that simulate real programs running on a range of modern processors and systems. The toolset includes simulators ranging from a fast functional simulator to a detailed, dynamically scheduled processor model that supports non-blocking caches, speculative execution, and branch prediction. Figure. 1 gives an overview of the SimpleScalar toolset.

A close up of a logo

Description automatically generated

Figure . SimpleScalar toolset overview

### Supported Instruction Set Architecture (ISA)

SimpleScalar can simulate Alpha and PISA (Portable ISA). The PISA instruction set is a simple MIPS-like instruction set maintained primarily for instructional use. The tool set takes binaries compiled for the SimpleScalar architecture and simulates their execution on one of several provided processor simulators. The machine running SimpleScalar is called the Host machine or Host while the ISA that one is targeting such as Alpha or PISA is called Target. Gcc cross-compiler for PISA is available on the internet. We will use gcc cross-compiler in this project.

### Available Simulators

* The toolset provides a collection of microarchitecture simulators that emulate the microprocessor at different levels of details, as listed following.
* **sim-fast**: fast instruction interpreter, optimized for speed. This simulator does not account for the behavior of pipelines, caches, or any other part of the microarchitecture. It performs only functional simulation using in-order execution of the instructions.
* **sim-safe**: slightly slower instruction interpreter, as it checks for memory alignment and memory access permission on all memory operations.
* **sim-profile**: instruction interpreter and profiler. This simulator keeps track of and reports dynamic instruction counts, instruction class counts, usage of address modes, and profiles of the text and data segments.
* **sim-cache**: memory system simulator. This simulator can emulate a system with multiple levels of instruction and data caches, each of which can be configured for different sizes and organizations. Since we are implementing 3-level cache, we are going to use this simulator as it provides sufficient simulation details for studying the cache but still has relatively fast performance.
* **sim-bpred**: branch predictor simulator. This tool can simulate difference branch prediction schemes and reports results such as prediction hit and miss rates. Like sim-cache, this does not simulate accurately the effect of branch prediction on execution time.
* **sim-outorder**: detailed microarchitectural simulator. This tool models in detail and out-of-order microprocessor with all of the bells and whistles, including branch prediction, caches, and external memory. This simulator is highly parameterized and can emulate machines of varying numbers of execution units.

**Cache Configuration & Basic Design**

In this project, you will implement a 3-level cache (L1, L2, L3) using simplescalar (or other similar tool approved by the instructor). L1 cache consists of separate I and D caches, both L2 and L3 are unified caches. They demonstrate inclusion property, i.e, all data in the L1 D-cache are present in L2, and all data in L2 are present in L3 cache.

Your tasks include:

1) Get familiar with Simplescalar. The simulator and its related documents can be downloaded at <http://www.simplescalar.com/>.

2) Modify the Simplescalar to support inclusion property and 3-level cache.

3) Modify the Simplescalar to support multicore.

4) Perform experiments to evaluate the cache design. You need use at least 2 benchmarks to collect the cache hit/miss information for the following configurations:

Configuration 1: single core processor

L1 D-cache: 16KB, L1 I- Cache: 16KB, 2-way, block size: 64B

L2 cache: 512KB, 4-way, block size: 64B

L3 cache: 8MB, 8-way, block size: 64B

Configuration 2: Dual core processor, each core has its own L1 and L2 cache and two cores share the L3 cache.

L1 D-cache: 8KB, L1 I- Cache: 8KB, direct mapped, block size: 64B

L2 cache: 128KB, 2-way, block size: 64B,

L3 cache: 16MB, 4-way, block size: 256B

**Schedule**

* 11/26: System Design Report Due
* 12/10: Project Demo/Presentation
* 12/14: Final report due.

**Requirements and Constraints**

* Work in groups of up to 2 students each
* Students select your group members
* Each member must design part of the project and must write his/her **OWN** part of the final report.
* Essential to help each other **WITHIN** teams!

**Phase 1 - System Design Report**

* This document reports the basic design including the design of each cache level, the design of the cache replacement algorithm (LRU or LFU), your understanding on SimpleScalar, and your plan to do the testing, etc. You should also indicate the responsibility of each team member in this report. It counts for 20% of your project grade.

**Phase 2 - Presentation and Demo**

* Each team will give a short presentation (approximately 15 minutes) about your project. Each member will orally talk about your design part.
* Demonstrate the simulation and testing results to the class.

**Phase 3 - Final Report**

* A final report contains two parts. The first part is a team-based report that includes the title of your project, a list of project team members and description of each member's contribution, a detailed description of your design, a discussion of how you test your design, and a discussion of what does not work correctly in your final design. You should also turn in the source code for your design and any relevant test benches in electronic format.
* The second part of the final report is an individual report. Each member will write a short individual report to state:

                1) What's your contribution on the team project?  
                2) What have you learned from this project?  
                3) Evaluate your team member's work.  
                4) Any comments on this project.

**Grade**

* Projects will be graded on how well the design works, the complexity of the design, optimizations made to the design, the thoroughness of the testing methodology, and the overall quality of the reports, presentation and demo.
  + Grade = System Design Report (20%) +Project functionality (25%) + Presentation (15%)+Final Report (40%)
  + Deduct 5 points per day late on each phase