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

**System Design Report**

Honghao Gan & Zhijia Chen

## Introduction to 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 1. 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 Design and Configuration

We are going to use SimpleScalar to implement a 3-level cache (L1, L2 and L3). L1 cache consists of separate I and D caches, both L2 and L3 are unified caches. We will modify SimpleScalar to make the cache display 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. And we will also modify SimpleScalar to support multicore. The cache will be evaluated against two configurations, one for single core processor and the other for dual core processor.

### Set replacement policy

SimpleScalar is shipped with three set replacement polices: LRU, FIFO and Random. We will use LFU and Random replacement polices in this project. Since LFU is not supported, we need to modify source codes to implement the policy.

### Configuration 1: Single Core Processor

The cache is configured as following:

* 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

Figure 2 shows the designed cache structure for configuration 1, and Figure 3 to Figure 5 depict the logic layout of the three levels of cache respectively.



Figure . Cache structure for configuration 1

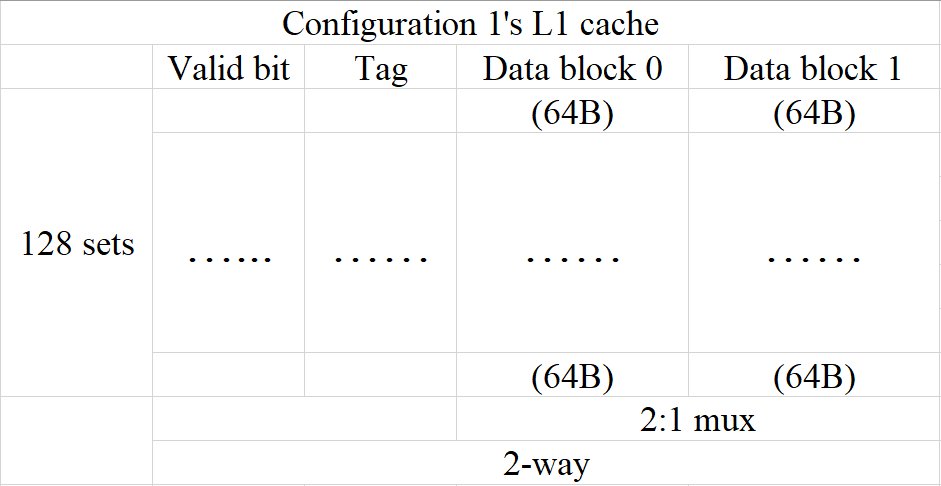


Figure . L1 cache design for configuration 1

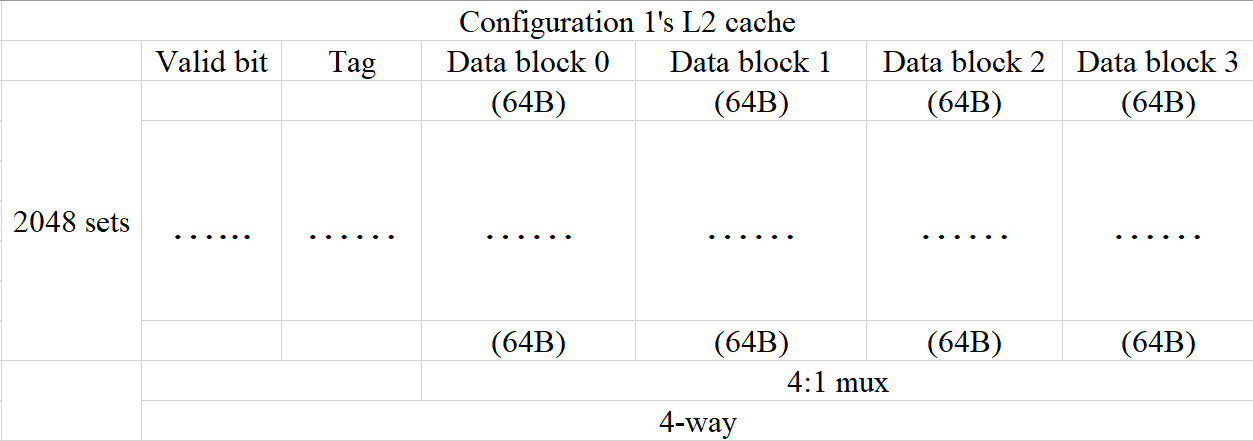


Figure . L2 cache design for configuration 1

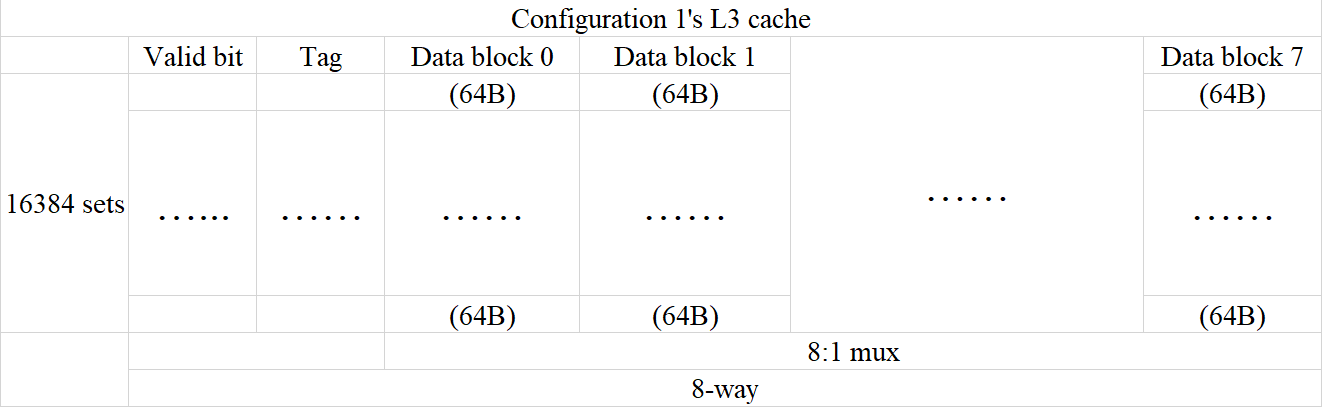


Figure . L3 cache design for configuration 1

### Configuration 2: Dual Core Processor

We will study the performance of the cache with dual core processor in this configuration. Each core has its own L1 and L2 cache and two cores share the L3 cache. The detailed configurations are listed below.

* 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

Figure 6 shows the designed cache structure for configuration 2, and Figure 7 to Figure 9 depict the logic layout of the three levels of cache respectively.



Figure . Cache structure for configuration 2

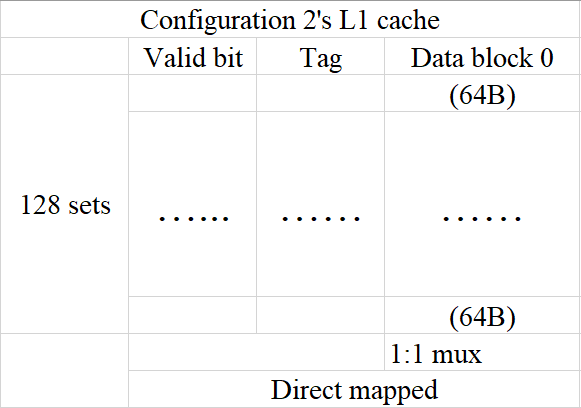


Figure . L1 cache design for configuration 2

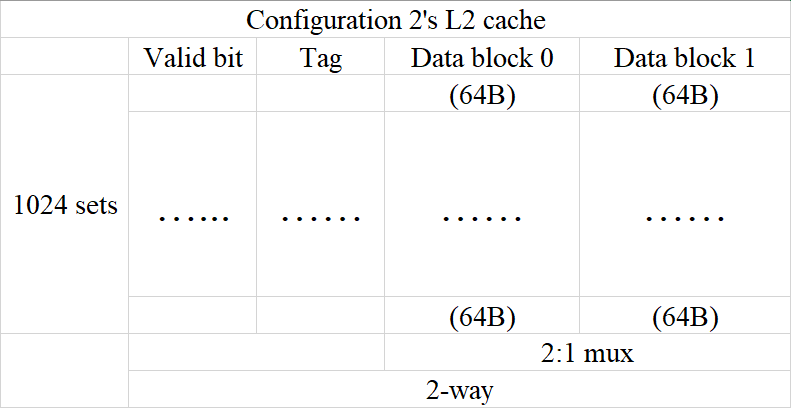


Figure . L2 cache design for configuration 2

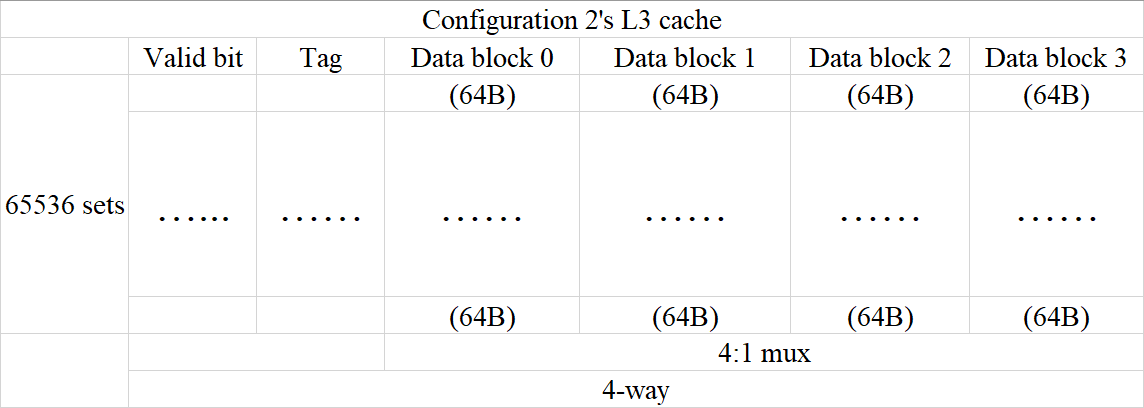


Figure . L2 cache design for configuration 2

## Testing Plan

We will study how different memory reference pattern of the testing programs affect the performance. To test the cache, we will need to write a program that references memory sequentially, and a program that references memory randomly. For the two configurations, we will run the cache with the two programs respectively and collect the cache hit/miss rates.

## Task Allocation

Honghao Gan:

* Modify the SimpleScalar to add LFU set replacement policy.
* Write the testing programs.
* Run benchmarks and collect data.

Zhijia Chen:

* Modify the SimpleScalar to support inclusion property and 3-level cache.
* Modify the SimpleScalar to support multicore.