Final Research Proposal

Final Project EECE7352 Computer Architecture

By Anna DeVries, Gautham Ranganathan, Shrikanth Showri Rajan

Research Problem

Analysis and implementation of different cache optimization techniques.

Background

The gap in performance between processor speed and memory continues to widen; new architectures are released every year with increased capabilities but memory systems remain stalemate, unable to cope with new demands. In this regard, it has become increasingly important to exploit cache memory efficiently.

Project Description

This project studies the performance improvements between four unique optimization techniques and three different simple cache design configurations. Each configuration and its optimization's performance on a standard benchmark are compared between implementations. This comparison provides an indication of which technique is the most effective.

Literature Review

Cache Optimization Techniques

The first article, "Cache Memory: An Analysis on Optimization Techniques," discusses five different cache optimization categories - replacement algorithms, cache algorithms, design based optimizations, compiler and prediction based optimization, and web based optimization [1]. The articles continues to layout a comparison of 16 specific optimization methods, comparing miss rate, miss penalty, hit time, hit rate, power consummation, access time, cost, complexity, and cycle time [1]. This article provides the team with guidance on important factors to measure during benchmark testing, such that an informed comparison can be created between the optimizations. Additionally, the team is implementing three of the optimizations discussed in the article - changing cache size, associativity and block size.

Cache Line Coloring Optimization

This paper discusses a link time procedure mapping algorithm which works by creating an improved program layout by color mapping considering the procedure size, cache size, cache line size and call graph. This algorithm reduces the miss rate by 40% from the original mapping. The algorithm intelligently places procedures in the cache layout by preserving color dependencies with a procedure's parents and children in the call graph, resulting in fewer instruction cache conflicts [2]. This paper implements the cache colouring algorithm and uses the ATOM tool for simulations. The paper uses programs like espresso, Perl, li, bison and gzip. The

team will carry out a similar study for different workloads on a x86 machine using PIN tool by intel for generating trace.

Methodology

This project consists of three main tasks: architecture design feature implementation, cache design based optimization, and program to memory layout optimization. The team will use C++ for all architecture and optimization designs; benchmarks are in C. The project will compare the various architecture optimizations with three different benchmarks (chosen from either CacheBench, STREAM[3], Linpack, gcc, Whetstone, or other SPEC benchmarks). The team will collect 3 result parameters for each optimization: miss rate (MR), hit time (HT), and hit rate (HR) [1]. Each optimization will be implemented individually and results compared with configuration base case.

First step, the team will implement a simple cache. The simple cache, or base case, will consist of the following architecture specifications: 64 KB cache size (32 KB instruction cache and 32 KB data cache), 32-byte block size with 32-bit addresses (x86 represented). The base case with utilize a first-in-first-out (FIFO) replacement algorithm and consist of three configurations (direct mapped, 2-way set associative and 4-way set associative).

Second step, the team will implement 3 different cache design based optimization techniques on each simple cache configuration. The different optimization techniques include improving the replacement algorithm (least recently used LRU and least frequently used LFU), increasing cache size, and increasing block size. The team expects a higher level of associativity in the cache configurations to reduce capacity and conflict misses, an increase in cache size to reduce miss rates but increase access time, and an increase in block size to decrease compulsory misses while increasing conflict misses [1]. The team also expects LRU to decrease conflict misses by using a more advanced algorithm to remove memory in cache.

Third step, the team will implement cache coloring as a program to memory cache optimization. We will simulate the three-base case configurations in C++. All benchmarks will be in compiled from C with GCC. We will create an instruction trace of each benchmark using SimpleScalar and/or Pin. An example of the results table the team will create for each cache configuration is provided below.

Table 1: Expected result table.

Direct-Mapped Cache			
	Total Demand Fetches	Miss Rate	Hit Rate
Base Case (no optimizations)			
LFU Replacement Algorithms			
LRU Replacement Algorithm			
Larger Cache (128 KB cache size)			
Larger Block Size (128-byte block size)			

Cache Coloring			

Expectations

The expected relation between gathered results and associated grade is shown below in the table. The team will work together to complete every task but each member is assigned a primary and secondary role of responsibility for one aspect of the project: architecture feature design, design based optimizations, and program-memory layout optimization.

The primary method for the team's communication is in-person, secondary through Slack messaging, and tertiary through email. The team's version control system (VCS) will be Git, hosted on GitHub.com. The team's research and analysis notation tool will be LaTeX, hosted on Overleaf.com. In general, the team will follow a waterfall type methodology for completing the project - requirements, design, implementation, and verification.

Individual members' tasks are provided below in the responsibility matrix. The project is split into three parts. The first task includes implementing a simple cache for each configuration (direct-mapped, two-way, and four-way set associative). The second task includes creating and implementing three optimization features (increasing cache size, increasing block size and improving the replacement algorithm). Additionally, this task is responsible for creating a program to gather all necessary data from benchmark tests and running all benchmark experiments. The third task includes implementing cache coloring.

Table 2: Results with expected associated grade.

A	Implemented three simple cache configurations, implemented four optimization features, and tested three benchmarks.
A-	Implemented three simple cache configurations, implemented three optimization features, and tested three benchmarks.
B+	Implemented two simple cache configurations, implemented three optimization features, and tested two benchmarks.
B	Implemented two simple cache configurations, implemented two optimization features, and tested two benchmarks.
B-	Implemented two simple cache configurations, implemented one optimization feature, and tested two benchmarks.
C+	Implemented one simple cache configuration, implemented one optimization feature, and tested one benchmark.
C	Implemented one simple cache configuration, and tested one benchmark.

Table 3: Responsibility Matrix

	Primary Responsibility	Secondary Responsibility
Architecture Design Feature	Anna DeVries	Gautham Ranganathan

(simple cache implementation for direct-mapped, 2-way and 4-way set associative)		
Design Based Optimizations, Data Gathering Script and Benchmark Testing (increase cache size, increased block size, improved replacement algorithm)	Gautham Ranganathan	Shrikanth Showri Rajan
Program-Memory Layout Optimization (cache coloring)	Shrikanth Showri Rajan	Anna DeVries

References

[1] M. W. Ahmed and M. A. Shah, "Cache Memory: An analysis on Optimization Techniques," *International Journal of Computer and Information Technology*, vol. 4, no. 2, pp. 414 - 418, March 2015.

[2] A. H. Hashemi, D. R. Kaeli, and B. Calder, "Efficient Procedure Mapping Using Cache Line Coloring," *ACM SIGPLAN Conference on Programming Language Design and Implementation*, June 1997.

[3] https://asc.llnl.gov/computing_resources/purple/archive/benchmarks/memory/membench_bm readme.html#CB%20Required%20Problems.