**Analysis of Direct Mapped Write-Back Cache Performance Using Trace Driven Simulation**

**Authors:**

Tyler Neal

CS 4431

Professor Soner Onder

4/18/2024

# **Abstract**

Theis report presents an investigation into differences on the performance characteristics of various cache configurations for a direct-mapped, write-back cache, using a trace-driven simulation with traces derived from Spec95 benchmarks. I have built a simulator that takes different cache attributes and simulates the cache with different positions in the hierarchy. The intent was to compare the miss rates for instruction, data, and unified caches and the effective memory access times for multiple cache hierarchies. I ran six benchmarks with a cache across different memories ranging from 4 KB to 1024 KB and a line varying from 4 and 8 words per line, moving from one, two, and three layers of cache hierarchy. The results show the performance difference between architectures, showing the cache-size-versus-speed trade-off.

# **Table of Contents**

[**Abstract 2**](#_a59yrwrkp17f)

[**Table of Contents 3**](#_hwvjrsx2pkdw)

[**Implementation 5**](#_lhg5vxchuhsr)

[Cache Configuration 5](#_w03blgvflmlv)

[Memory Allocation and Initialization 5](#_hj6jf7ysje15)

[Address Decomposition 5](#_m4xcgnug31s4)

[Cache Operations 6](#_aaj2rl3q0es9)

[Parsing Traces 6](#_usdnpq5aktpd)

[Performance Metrics 6](#_3tdt8s2hvs0v)

[Helpers 6](#_djj5bd93z7b1)

[**Configurations 7**](#_gtjn89aduv17)

[Trace Files 7](#_crli07yts9by)

[Phase 1: Single-Layer Cache Testing 7](#_tn4zra3czdjk)

[Phase 2: Multi-Layer Cache Testing 7](#_dp9kk44ma5to)

[Cache Timing Assumptions 8](#_33rp5e5lvx60)

[**Results 9**](#_t0fhs9g07uqh)

[Phase 1 Results 9](#_43ponbgsr3up)

[126.gcc129.compress 10](#_xby7rlhgol07)

[132.ijpeg 11](#_6lcpuxyar61g)

[134.perl 11](#_841b0c2e6qkb)

[099.go 12](#_xzxl6x41hpa0)

[124.m88ksim 12](#_qpwngzeyeiam)

[Phase 2 Results 13](#_g2xnn4m8yjh4)

[126.gcc 13](#_smzya6auhorb)

[129.compress 15](#_dh9orbbqvnv5)

[132.ijpeg 17](#_co1i6nuup8dm)

[134.perl 19](#_4tmkiknzvi2k)

[099.go 21](#_873chidlrej0)

[124.m88ksim 23](#_5zuo6fafjr7u)

[**Discussion 25**](#_avo5jgliqatf)

[Phase 1 Discussion 25](#_xxjo941akstt)

[Impact of Cache Size: 25](#_7r2titi4imuh)

[Impact of Line Size: 25](#_8w8tlqhyfmmz)

[Comparison Across Benchmarks: 25](#_4gxxok4skicq)

[Phase 2 Discussion 26](#_wgcm2txr0hl)

[Impact of Multi-Layering on AMAT: 26](#_3eb4jukhiqub)

[Impact of Increasing Cache Sizes within Layers: 26](#_wd1lti46xcel)

[Configurations with and without L3 Cache: 26](#_618cbwbkfwpl)

[Comparison Across Benchmarks: 26](#_pps6rkpvxbnz)

[**Conclusion 27**](#_e0u2perb6cov)

[Designing for Application-Specific Needs: 27](#_mf4qup8sj6c7)

[Balancing Cache Layers and Sizes: 27](#_jf7n02j7v7ui)

[Practical Implications for Simulation-Based Design: 27](#_2kdybhomyx8f)

[**Appendices 28**](#_j46c1ohzf3vj)

[Layer 1 Statistics 28](#_a75ctwd23w46)

[Layer 2 Statistics 32](#_nofibmv0sswp)

[Layer 3 Statistics 36](#_hp3hrkc00fvc)

[main.c 40](#_hr6etmesui6c)

[cache.h 50](#_7jfdzpuv6por)

[Makefile 53](#_u6awh2u6miu3)

[run1.sh 54](#_lxq9it5ls0al)

[run2.sh 56](#_ufgg1ly8mn8t)

# 

# **Implementation**

The simulation of the direct-mapped write-back cache is implemented in C and involves detailed mechanisms to simulate real-world caching operations at various layers (L1, L2, L3). Here, I’ll provide an in-depth explanation of the cache setup, operation, and the methods I’ve employed to assess its performance using a trace-driven approach.

## **Cache Configuration**

The cache configuration is established using command-line arguments, which specify the cache type, line size, number of cache layers, and individual sizes for the L1, L2, and L3 caches. The system supports up to three cache layers. If fewer than three layers are specified, the configuration for the remaining layers (second and/or third) is disregarded. Each cache layer can be individually configured to have different sizes and line sizes.

The caches are implemented as arrays of line structures, where each line can store a tag and a dirty bit. The dirty bit indicates whether the line has been modified, indicating if a write back is necessary. The tag, combined with the line's index, is used to determine cache hits or misses. A typical cache implementation also contains a valid bit in each line. However, for our purposes, this bit is unnecessary, and has been excluded as a result.

## **Memory Allocation and Initialization**

Each cache's memory is allocated dynamically based on the specified cache size and line size. The number of lines in a cache is calculated as the total cache size divided by the line size. Each line within the cache is initialized with a default 'uninitialized' tag and a clean (not dirty) status. A request struct, containing information about a specific trace, such as hex address, tag, index, and offset, is also allocated once during the program. This request is reused for each request the trace files generate in order to save memory.

## **Address Decomposition**

For each memory request processed by the cache, the address is decomposed into tag, index, and offset based on the configuration of the cache layer being requested. This decomposition must be done on the request’s address for each layer it traverses, as the size of the address fields is specific to the cache and line size.

#### 

## **Cache Operations**

The main operations supported are read and write. On a cache hit, if the operation is a read, the data is simply accessed. During a write, the dirty bit is also set. On a cache miss, the corresponding line is updated with the new tag, and the dirty bit is set or reset depending on whether the previous data contained in the line had been modified. If a line being replaced is dirty, this triggers a write-back operation.

## **Parsing Traces**

The simulation operates by processing a trace file that contains memory access requests formatted as @<I/D><R/W><address>. In this format, @ indicates the beginning of a trace entry, I or D specifies whether the access is for an instruction or data reference, R or W denotes a read or write operation, and the address is represented as an eight-character hexadecimal value. Each request is parsed, and the appropriate cache operation is performed. Traces are parsed until the end of the trace file is read.

## **Performance Metrics**

Various performance metrics are captured for each cache, including the number of requests processed, hits, misses, hit rate, miss rate, reads causing writes, writes causing write-backs. Additionally, the average memory access time is derived and printed once the program finishes running.

## **Helpers**

The cache simulation project uses a few key helper functions to manage and interpret data. The itob function converts integers to binary strings, helping to break down memory addresses into components like tag, index, and offset. The opposite is done by the btoi function, which turns binary strings back into integers for use in the cache's operations. There's also a printCacheStats function that shows important statistics like the number of requests, hits, and misses. This function can adjust its output style based on what’s being tested for, expanding or condensing the printed metrics.

# 

# **Configurations**

This section details the setup and parameters used for simulating cache performance, and is split into two distinct phases of testing according to the testing guidelines. Each phase will focus on different aspects of cache architecture, ranging in cache size, line size, and number of cache levels involved.

## **Trace Files**

The trace files used in this simulation range in size from about 25 KB to 6,500,000 KB. The files are '126.gcc', '129.compress', '132.ijpeg', '134.perl', '099.go', and '124.m88ksim'. These files represent a variety of program behaviors and are used to test the cache performance under different conditions. Each file is processed in ascending order to evaluate how the cache handles different types of data access patterns.

## **Phase 1: Single-Layer Cache Testing**

The first phase of testing focuses on single-layer cache configurations with variations in cache size and line size:

**Cache Sizes Tested**: 8 KB, 16 KB

**Line Sizes Tested**: 4 words, 8 words

The aim was to determine miss rates for an instruction cache, data cache, and a unified cache under each configuration.

## **Phase 2: Multi-Layer Cache Testing**

The second phase incorporates multiple cache layers with varying sizes, while focusing on a fixed line size.

**Fixed Line Size**: 8 words per line (consistent for all configurations to isolate the effect of cache size and layering)

**Cache Configurations Tested**:

* **L1 Cache:** 4 KB, 16 KB
* **L2 Cache:** 32 KB, 64 KB
* **L3 Cache:** 256 KB, 1024 KB

This phase was designed to explore the impact of multi-layer cache hierarchies on cache performance, specifically looking at miss rates and effective memory access times (AdMAT).

## **Cache Timing Assumptions**

For both phases, the following cache timing assumptions were made to calculate EMAT:

* **L1 Cache Hit Time**: 1 cycle
* **L2 Cache Hit Time**: 16 cycles
* **L3 Cache Hit Time**: 64 cycles
* **Main Memory Latency**: 100 cycles

Data transfers between cache levels and from the last cache level to main memory were assumed to complete in one cycle per line upon a hit.

# 

# **Results**

## **Phase 1 Results**

| **Comparison of Miss Rates for Direct-Mapped, Write-Back Cache Configurations: Data, Instruction, and Unified Caches Across Multiple Benchmarks** | | | | |
| --- | --- | --- | --- | --- |
|
|
| **Benchmark** | **Configuration** | **Unified Miss Rate** | **Instruction Miss Rate** | **Data Miss Rate** |
| 126.gcc | 8KB, 4 word/line | 4.48% | 6.13% | 7.97% |
|  | 8KB, 8 word/line | 3.03% | 4.16% | 6.43% |
|  | 16KB, 4 word/line | 2.74% | 3.75% | 5.01% |
|  | 16KB, 8 word/line | 1.86% | 2.55% | 4.09% |
| 129.compress | 8KB, 4 word/line | 0.22% | 0.35% | 0.25% |
|  | 8KB, 8 word/line | 0.14% | 0.23% | 0.21% |
|  | 16KB, 4 word/line | 0.11% | 0.18% | 0.13% |
|  | 16KB, 8 word/line | 0.07% | 0.12% | 0.10% |
| 132.ijpeg | 8KB, 4 word/line | 1.38% | 1.80% | 3.85% |
|  | 8KB, 8 word/line | 0.91% | 1.19% | 3.82% |
|  | 16KB, 4 word/line | 1.35% | 1.76% | 3.81% |
|  | 16KB, 8 word/line | 0.89% | 1.17% | 3.77% |
| 134.perl | 8KB, 4 word/line | 5.31% | 7.43% | 8.26% |
|  | 8KB, 8 word/line | 3.69% | 5.17% | 6.98% |
|  | 16KB, 4 word/line | 2.82% | 3.95% | 3.62% |
|  | 16KB, 8 word/line | 2.04% | 2.85% | 3.38% |
| 099.go | 8KB, 4 word/line | 4.82% | 6.12% | 13.08% |
|  | 8KB, 8 word/line | 2.75% | 3.48% | 9.96% |
|  | 16KB, 4 word/line | 3.77% | 4.78% | 10.07% |
|  | 16KB, 8 word/line | 2.12% | 2.70% | 7.73% |
| 124.m88ksim | 8KB, 4 word/line | 3.72% | 4.64% | 6.74% |
|  | 8KB, 8 word/line | 2.64% | 3.29% | 6.60% |
|  | 16KB, 4 word/line | 3.15% | 3.94% | 5.96% |
|  | 16KB, 8 word/line | 2.08% | 2.59% | 5.86% |

### **126.gcc**Chart**129.compress**Chart

### **132.ijpeg**Chart

### **134.perl**Chart

### **099.go**Chart

### **124.m88ksim**Chart

## **Phase 2 Results**

### **126.gcc**

| **Average Memory Access Time (AMAT) for Different Cache Configurations** | | | |
| --- | --- | --- | --- |
| **Benchmark: 126.gcc** | | | |
| **L1 Size (KB)** | **L2 Size (KB)** | **L3 Size (KB)** | **AMAT (cycles)** |
| 4 | **-** | **-** | 11.09 |
| 16 | **-** | **-** | 5.09 |
| 4 | 32 | **-** | 5.22 |
| 4 | 64 | **-** | 4.37 |
| 16 | 32 | **-** | 4.26 |
| 16 | 64 | **-** | 3.41 |
| 4 | 32 | 256 | 8.05 |
| 4 | 32 | 1024 | 7.46 |
| 4 | 64 | 256 | 9.34 |
| 4 | 64 | 1024 | 8.46 |
| 16 | 32 | 256 | 4.85 |
| 16 | 32 | 1024 | 4.61 |
| 16 | 64 | 256 | 5.04 |
| 16 | 64 | 1024 | 4.69 |

# Chart

### **129.compress**

| **Average Memory Access Time (AMAT) for Different Cache Configurations** | | | |
| --- | --- | --- | --- |
| **Benchmark: 129.compress** | | | |
| **L1 Size (KB)** | **L2 Size (KB)** | **L3 Size (KB)** | **AMAT (cycles)** |
| 4 | **-** | **-** | 1.38 |
| 16 | **-** | **-** | 1.10 |
| 4 | 32 | **-** | 1.11 |
| 4 | 64 | **-** | 1.10 |
| 16 | 32 | **-** | 1.07 |
| 16 | 64 | **-** | 1.06 |
| 4 | 32 | 256 | 1.32 |
| 4 | 32 | 1024 | 1.32 |
| 4 | 64 | 256 | 1.37 |
| 4 | 64 | 1024 | 1.37 |
| 16 | 32 | 256 | 1.11 |
| 16 | 32 | 1024 | 1.11 |
| 16 | 64 | 256 | 1.12 |
| 16 | 64 | 1024 | 1.12 |

# Chart

### **132.ijpeg**

| **Average Memory Access Time (AMAT) for Different Cache Configurations** | | | |
| --- | --- | --- | --- |
| **Benchmark: 132.ijpeg** | | | |
| **L1 Size (KB)** | **L2 Size (KB)** | **L3 Size (KB)** | **AMAT (cycles)** |
| 4 | **-** | **-** | 4.87 |
| 16 | **-** | **-** | 4.77 |
| 4 | 32 | **-** | 5.38 |
| 4 | 64 | **-** | 5.37 |
| 16 | 32 | **-** | 5.36 |
| 16 | 64 | **-** | 5.36 |
| 4 | 32 | 256 | 4.08 |
| 4 | 32 | 1024 | 4.08 |
| 4 | 64 | 256 | 4.07 |
| 4 | 64 | 1024 | 4.07 |
| 16 | 32 | 256 | 4.06 |
| 16 | 32 | 1024 | 4.06 |
| 16 | 64 | 256 | 4.06 |
| 16 | 64 | 1024 | 4.06 |

# Chart

### **134.perl**

| **Average Memory Access Time (AMAT) for Different Cache Configurations** | | | |
| --- | --- | --- | --- |
| **Benchmark: 134.perl** | | | |
| **L1 Size (KB)** | **L2 Size (KB)** | **L3 Size (KB)** | **AMAT (cycles)** |
| 4 | **-** | **-** | 12.35 |
| 16 | **-** | **-** | 4.38 |
| 4 | 32 | **-** | 4.77 |
| 4 | 64 | **-** | 3.80 |
| 16 | 32 | **-** | 3.49 |
| 16 | 64 | **-** | 2.53 |
| 4 | 32 | 256 | 4.34 |
| 4 | 32 | 1024 | 4.30 |
| 4 | 64 | 256 | 3.99 |
| 4 | 64 | 1024 | 3.91 |
| 16 | 32 | 256 | 2.87 |
| 16 | 32 | 1024 | 2.86 |
| 16 | 64 | 256 | 2.33 |
| 16 | 64 | 1024 | 2.31 |

# Chart

### **099.go**

| **Average Memory Access Time (AMAT) for Different Cache Configurations** | | | |
| --- | --- | --- | --- |
| **Benchmark: 099.go** | | | |
| **L1 Size (KB)** | **L2 Size (KB)** | **L3 Size (KB)** | **AMAT (cycles)** |
| 4 | **-** | **-** | 14.72 |
| 16 | **-** | **-** | 8.73 |
| 4 | 32 | **-** | 8.63 |
| 4 | 64 | **-** | 7.52 |
| 16 | 32 | **-** | 7.67 |
| 16 | 64 | **-** | 6.56 |
| 4 | 32 | 256 | 6.85 |
| 4 | 32 | 1024 | 6.76 |
| 4 | 64 | 256 | 6.18 |
| 4 | 64 | 1024 | 6.07 |
| 16 | 32 | 256 | 5.81 |
| 16 | 32 | 1024 | 5.76 |
| 16 | 64 | 256 | 5.13 |
| 16 | 64 | 1024 | 5.07 |

# Chart

### **124.m88ksim**

| **Average Memory Access Time (AMAT) for Different Cache Configurations** | | | |
| --- | --- | --- | --- |
| **Benchmark: 124.m88ksim** | | | |
| **L1 Size (KB)** | **L2 Size (KB)** | **L3 Size (KB)** | **AMAT (cycles)** |
| 4 | **-** | **-** | 12.91 |
| 16 | **-** | **-** | 6.86 |
| 4 | 32 | **-** | 5.75 |
| 4 | 64 | **-** | 5.48 |
| 16 | 32 | **-** | 4.79 |
| 16 | 64 | **-** | 4.51 |
| 4 | 32 | 256 | 4.74 |
| 4 | 32 | 1024 | 4.74 |
| 4 | 64 | 256 | 4.56 |
| 4 | 64 | 1024 | 4.56 |
| 16 | 32 | 256 | 3.76 |
| 16 | 32 | 1024 | 3.76 |
| 16 | 64 | 256 | 3.59 |
| 16 | 64 | 1024 | 3.59 |

# Chart

# **Discussion**

## **Phase 1 Discussion**

### **Impact of Cache Size:**

The simulation data clearly shows that increasing the cache size leads to a decrease in miss rates across all types of caches: instruction, data, and unified. For example, in the '126.gcc' benchmark, the unified miss rate decreases from 4.48% to 2.74% when the cache size is doubled from 8 KB to 16 KB, while maintaining a line size of 4 words. This shows that larger caches can reduce the frequency of cache misses.

### **Impact of Line Size:**

Increasing the line size generally reduces miss rates, which can be attributed to spatial locality. Larger line sizes suggest that more data is considered accessed upon a hit, which naturally decreases the number of cache misses. This can be seen in the data; for instance, increasing the line size in the 099.go benchmark results in a significant reduction in data miss rates. However, larger lines can potentially increase the chance of loading unneeded data into the cache, there needs to be balance in line size.

### **Comparison Across Benchmarks:**

The variability in miss rates between benchmarks such as ‘129.compress’ and ‘134.perl’ is likely due to differences in the characteristics of the trace files. ‘129.compress’ contains much lower miss rates, which might indicate more local access patterns. On the contrary, ‘134.perl’ suggests that it contains more dispersed access patterns.

## 

## **Phase 2 Discussion**

### **Impact of Multi-Layering on AMAT:**

The data gathered demonstrates a general trend, where adding more cache layers and increasing their size, consistently reduces AMAT across benchmarks. This effect becomes much more noticeable when comparing configurations that progressively increase space of the L2 and L3 caches.

For example, in the '134.perl' benchmark, the AMAT decreases from 4.38 cycles with only a 16 KB L1 cache to 2.53 cycles when an 64 KB L2 is added, and even further to 2.31 when a 1024 KB L3 layer is added.   
  
Another trend that stands out is that the largest AMAT is most often found in configurations with a single L1 layer of the smallest size. In fact, 5 out of the 6 benchmarks tested follow this pattern, with the only exclusion being 132.ijpeg.

### **Impact of Increasing Cache Sizes within Layers:**

Larger caches within each layer configuration generally lead to improved AMAT. This trend is consistent across different benchmarks. For instance, in the ‘134.perl’ benchmark, increasing the L1 cache size from 4KB to 16KB drastically reduces the AMAT from 12.35 cycles to 4.38 cycles in configurations excluding the L2 and L3 caches. This is likely due to an increased miss rate due to the cache size, paired with frequent time-intensive accesses to main memory.

### **Configurations with and without L3 Cache:**

The addition of an L3 cache does not always result in lower AMAT, which can be seen in some benchmarks where the inclusion of an L3 cache of 256 KB or 1024 KB increases AMAT compared to setups without L3. This implies that certain workloads (traces) might be most optimized when less complex cache structures.

### **Comparison Across Benchmarks:**

Different benchmarks react differently to changes in cache configuration. For example, '129.compress' shows relatively small changes in AMAT across various configurations, indicating that it might have a smaller working set or highly efficient cache usage. In contrast, '099.go' shows significant improvements with each addition of cache layers and increases in cache size, reflecting a workload that benefits more substantially from larger, multi-layer caches.

# **Conclusion**

The main findings of the project do highlight the size of the cache and line size, how caches are contained hierarchically layered in order to get an improved up-rated performance of the system. Below is the summary of the main findings.

### **Designing for Application-Specific Needs:**

The high variability of performance across the benchmarks would recommend that the cache setting should be tailored to the given application. It could be possible for systems to be optimized through typical access pattern analysis, then the cache size and line sizes get adjusted appropriately.

### **Balancing Cache Layers and Sizes:**

On the one hand, while larger caches and multilayer caches improve performance, a balance is to be struck between them, depending upon specific workload characteristics. In both, the latter need to take note of the case of diminishing returns when the cache is too large, or on the contrary, when the number of layers in the cache is overdone and only lead to increased complexity and cost without proportionate benefits.

### **Practical Implications for Simulation-Based Design:**

The purely statistical approach applied in this simulation underscores the importance of trace-driven simulations in making useful predictions for real-world cache performance. The approach presents a very cost-effective and insightful tool for cache memory research and development.

# **Appendices**

## **Layer 1 Statistics**

| Recorded Statistics for Different Configurations for Layer (1) | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Benchmark | Config (L1 size, L2 size, L3 size) | | | Requests | Hits | Misses | Reads to Writes | Writes to Writes |
| 126.gcc | 4 | - | - | 639380 | 574858 | 64522 | 20069 | 8852 |
|  | 16 | - | - | 639380 | 613261 | 26119 | 7672 | 3347 |
|  | 4 | 32 | - | 639380 | 574858 | 64522 | 20069 | 8852 |
|  | 4 | 64 | - | 639380 | 574858 | 64522 | 20069 | 8852 |
|  | 16 | 32 | - | 639380 | 613261 | 26119 | 7672 | 3347 |
|  | 16 | 64 | - | 639380 | 613261 | 26119 | 7672 | 3347 |
|  | 4 | 32 | 256 | 639380 | 574858 | 64522 | 20069 | 8852 |
|  | 4 | 32 | 1024 | 639380 | 574858 | 64522 | 20069 | 8852 |
|  | 4 | 64 | 256 | 639380 | 574858 | 64522 | 20069 | 8852 |
|  | 4 | 64 | 1024 | 639380 | 574858 | 64522 | 20069 | 8852 |
|  | 16 | 32 | 256 | 639380 | 613261 | 26119 | 7672 | 3347 |
|  | 16 | 32 | 1024 | 639380 | 613261 | 26119 | 7672 | 3347 |
|  | 16 | 64 | 256 | 639380 | 613261 | 26119 | 7672 | 3347 |
|  | 16 | 64 | 1024 | 639380 | 613261 | 26119 | 7672 | 3347 |

# 

# 

| 129.compress | 4 | - | - | 2213696 | 2205349 | 8347 | 3035 | 1191 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 16 | - | - | 2213696 | 2211485 | 2211 | 882 | 205 |
|  | 4 | 32 | - | 2213696 | 2205349 | 8347 | 3035 | 1191 |
|  | 4 | 64 | - | 2213696 | 2205349 | 8347 | 3035 | 1191 |
|  | 16 | 32 | - | 2213696 | 2211485 | 2211 | 882 | 205 |
|  | 16 | 64 | - | 2213696 | 2211485 | 2211 | 882 | 205 |
|  | 4 | 32 | 256 | 2213696 | 2205349 | 8347 | 3035 | 1191 |
|  | 4 | 32 | 1024 | 2213696 | 2205349 | 8347 | 3035 | 1191 |
|  | 4 | 64 | 256 | 2213696 | 2205349 | 8347 | 3035 | 1191 |
|  | 4 | 64 | 1024 | 2213696 | 2205349 | 8347 | 3035 | 1191 |
|  | 16 | 32 | 256 | 2213696 | 2211485 | 2211 | 882 | 205 |
|  | 16 | 32 | 1024 | 2213696 | 2211485 | 2211 | 882 | 205 |
|  | 16 | 64 | 256 | 2213696 | 2211485 | 2211 | 882 | 205 |
|  | 16 | 64 | 1024 | 2213696 | 2211485 | 2211 | 882 | 205 |

# 

| 132.ijpeg | 4 | - | - | 3571104 | 3432740 | 138364 | 68014 | 34271 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 16 | - | - | 3571104 | 3436345 | 134759 | 66792 | 33455 |
|  | 4 | 32 | - | 3571104 | 3432740 | 138364 | 68014 | 34271 |
|  | 4 | 64 | - | 3571104 | 3432740 | 138364 | 68014 | 34271 |
|  | 16 | 32 | - | 3571104 | 3436345 | 134759 | 66792 | 33455 |
|  | 16 | 64 | - | 3571104 | 3436345 | 134759 | 66792 | 33455 |
|  | 4 | 32 | 256 | 3571104 | 3432740 | 138364 | 68014 | 34271 |
|  | 4 | 32 | 1024 | 3571104 | 3432740 | 138364 | 68014 | 34271 |
|  | 4 | 64 | 256 | 3571104 | 3432740 | 138364 | 68014 | 34271 |
|  | 4 | 64 | 1024 | 3571104 | 3432740 | 138364 | 68014 | 34271 |
|  | 16 | 32 | 256 | 3571104 | 3436345 | 134759 | 66792 | 33455 |
|  | 16 | 32 | 1024 | 3571104 | 3436345 | 134759 | 66792 | 33455 |
|  | 16 | 64 | 256 | 3571104 | 3436345 | 134759 | 66792 | 33455 |
|  | 16 | 64 | 1024 | 3571104 | 3436345 | 134759 | 66792 | 33455 |

# 

| 134.perl | 4 | - | - | 5362238 | 4753717 | 608521 | 199571 | 130945 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 16 | - | - | 5362238 | 5180919 | 181319 | 63875 | 36558 |
|  | 4 | 32 | - | 5362238 | 4753717 | 608521 | 199571 | 130945 |
|  | 4 | 64 | - | 5362238 | 4753717 | 608521 | 199571 | 130945 |
|  | 16 | 32 | - | 5362238 | 5180919 | 181319 | 63875 | 36558 |
|  | 16 | 64 | - | 5362238 | 5180919 | 181319 | 63875 | 36558 |
|  | 4 | 32 | 256 | 5362238 | 4753717 | 608521 | 199571 | 130945 |
|  | 4 | 32 | 1024 | 5362238 | 4753717 | 608521 | 199571 | 130945 |
|  | 4 | 64 | 256 | 5362238 | 4753717 | 608521 | 199571 | 130945 |
|  | 4 | 64 | 1024 | 5362238 | 4753717 | 608521 | 199571 | 130945 |
|  | 16 | 32 | 256 | 5362238 | 5180919 | 181319 | 63875 | 36558 |
|  | 16 | 32 | 1024 | 5362238 | 5180919 | 181319 | 63875 | 36558 |
|  | 16 | 64 | 256 | 5362238 | 5180919 | 181319 | 63875 | 36558 |
|  | 16 | 64 | 1024 | 5362238 | 5180919 | 181319 | 63875 | 36558 |

# 

| 099.go | 4 | - | - | 19982647 | 17241315 | 2741332 | 876318 | 295659 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 16 | - | - | 19982647 | 18437893 | 18437893 | 482565 | 168189 |
|  | 4 | 32 | - | 19982647 | 17241315 | 2741332 | 876318 | 295659 |
|  | 4 | 64 | - | 19982647 | 17241315 | 2741332 | 876318 | 295659 |
|  | 16 | 32 | - | 19982647 | 18437893 | 18437893 | 482565 | 168189 |
|  | 16 | 64 | - | 19982647 | 18437893 | 18437893 | 482565 | 168189 |
|  | 4 | 32 | 256 | 19982647 | 17241315 | 2741332 | 876318 | 295659 |
|  | 4 | 32 | 1024 | 19982647 | 17241315 | 2741332 | 876318 | 295659 |
|  | 4 | 64 | 256 | 19982647 | 17241315 | 2741332 | 876318 | 295659 |
|  | 4 | 64 | 1024 | 19982647 | 17241315 | 2741332 | 876318 | 295659 |
|  | 16 | 32 | 256 | 19982647 | 18437893 | 18437893 | 482565 | 168189 |
|  | 16 | 32 | 1024 | 19982647 | 18437893 | 18437893 | 482565 | 168189 |
|  | 16 | 64 | 256 | 19982647 | 18437893 | 18437893 | 482565 | 168189 |
|  | 16 | 64 | 1024 | 19982647 | 18437893 | 18437893 | 482565 | 168189 |

# 

| 124.m88ksim | 4 | - | - | 119078668 | 104891343 | 14187325 | 4405375 | 943322 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 16 | - | - | 119078668 | 112095001 | 6983667 | 1962207 | 369498 |
|  | 4 | 32 | - | 119078668 | 104891343 | 14187325 | 4405375 | 943322 |
|  | 4 | 64 | - | 119078668 | 104891343 | 14187325 | 4405375 | 943322 |
|  | 16 | 32 | - | 119078668 | 112095001 | 6983667 | 1962207 | 369498 |
|  | 16 | 64 | - | 119078668 | 112095001 | 6983667 | 1962207 | 369498 |
|  | 4 | 32 | 256 | 119078668 | 104891343 | 14187325 | 4405375 | 943322 |
|  | 4 | 32 | 1024 | 119078668 | 104891343 | 14187325 | 4405375 | 943322 |
|  | 4 | 64 | 256 | 119078668 | 104891343 | 14187325 | 4405375 | 943322 |
|  | 4 | 64 | 1024 | 119078668 | 104891343 | 14187325 | 4405375 | 943322 |
|  | 16 | 32 | 256 | 119078668 | 112095001 | 6983667 | 1962207 | 369498 |
|  | 16 | 32 | 1024 | 119078668 | 112095001 | 6983667 | 1962207 | 369498 |
|  | 16 | 64 | 256 | 119078668 | 112095001 | 6983667 | 1962207 | 369498 |
|  | 16 | 64 | 1024 | 119078668 | 112095001 | 6983667 | 1962207 | 369498 |

# 

## **Layer 2 Statistics**

| Recorded Statistics for Different Configurations for Layer (2) | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Benchmark | Config (L1 size, L2 size, L3 size) | | | Requests | Hits | Misses | Reads to Writes | Writes to Writes |
| 126.gcc | 4 | - | - | - | - | - | - | - |
|  | 16 | - | - | - | - | - | - | - |
|  | 4 | 32 | - | 64522 | 47867 | 16655 | 3624 | 1656 |
|  | 4 | 64 | - | 64522 | 53316 | 11206 | 2261 | 1049 |
|  | 16 | 32 | - | 26119 | 9464 | 16655 | 3595 | 3595 |
|  | 16 | 64 | - | 26119 | 14913 | 11206 | 2238 | 1038 |
|  | 4 | 32 | 256 | 64522 | 47867 | 16655 | 3624 | 1656 |
|  | 4 | 32 | 1024 | 64522 | 47867 | 16655 | 3624 | 1656 |
|  | 4 | 64 | 256 | 64522 | 53316 | 11206 | 2261 | 1049 |
|  | 4 | 64 | 1024 | 64522 | 53316 | 11206 | 2261 | 1049 |
|  | 16 | 32 | 256 | 26119 | 9464 | 16655 | 3595 | 3595 |
|  | 16 | 32 | 1024 | 26119 | 9464 | 16655 | 3595 | 3595 |
|  | 16 | 64 | 256 | 26119 | 14913 | 11206 | 2238 | 1038 |
|  | 16 | 64 | 1024 | 26119 | 14913 | 11206 | 2238 | 1038 |

# 

# 

| 129.compress | 4 | - | - | - | - | - | - | - |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 16 | - | - | - | - | - | - | - |
|  | 4 | 32 | - | 8347 | 7218 | 1129 | 226 | 33 |
|  | 4 | 64 | - | 8347 | 7421 | 926 | 114 | 20 |
|  | 16 | 32 | - | 2211 | 1082 | 1129 | 226 | 33 |
|  | 16 | 64 | - | 2211 | 1285 | 926 | 114 | 20 |
|  | 4 | 32 | 256 | 8347 | 7218 | 1129 | 226 | 33 |
|  | 4 | 32 | 1024 | 8347 | 7218 | 1129 | 226 | 33 |
|  | 4 | 64 | 256 | 8347 | 7421 | 926 | 114 | 20 |
|  | 4 | 64 | 1024 | 8347 | 7421 | 926 | 114 | 20 |
|  | 16 | 32 | 256 | 2211 | 1082 | 1129 | 226 | 33 |
|  | 16 | 32 | 1024 | 2211 | 1082 | 1129 | 226 | 33 |
|  | 16 | 64 | 256 | 2211 | 1285 | 926 | 114 | 20 |
|  | 16 | 64 | 1024 | 2211 | 1285 | 926 | 114 | 20 |

# 

| 132.ijpeg | 4 | - | - | - | - | - | - | - |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 16 | - | - | - | - | - | - | - |
|  | 4 | 32 | - | 138364 | 4075 | 134289 | 33352 | 228 |
|  | 4 | 64 | - | 138364 | 4303 | 134061 | 33194 | 112 |
|  | 16 | 32 | - | 134759 | 470 | 134289 | 33349 | 223 |
|  | 16 | 64 | - | 134759 | 698 | 134061 | 33193 | 109 |
|  | 4 | 32 | 256 | 138364 | 4075 | 134289 | 33352 | 228 |
|  | 4 | 32 | 1024 | 138364 | 4075 | 134289 | 33352 | 228 |
|  | 4 | 64 | 256 | 138364 | 4303 | 134061 | 33194 | 112 |
|  | 4 | 64 | 1024 | 138364 | 4303 | 134061 | 33194 | 112 |
|  | 16 | 32 | 256 | 134759 | 470 | 134289 | 33349 | 223 |
|  | 16 | 32 | 1024 | 134759 | 470 | 134289 | 33349 | 223 |
|  | 16 | 64 | 256 | 134759 | 698 | 134061 | 33193 | 109 |
|  | 16 | 64 | 1024 | 134759 | 698 | 134061 | 33193 | 109 |

# 

| 134.perl | 4 | - | - | - | - | - | - | - |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 16 | - | - | - | - | - | - | - |
|  | 4 | 32 | - | 608521 | 503813 | 104708 | 18675 | 24689 |
|  | 4 | 64 | - | 608521 | 555591 | 52930 | 10806 | 10595 |
|  | 16 | 32 | - | 181319 | 76611 | 104708 | 18647 | 24646 |
|  | 16 | 64 | - | 181319 | 128389 | 52930 | 10781 | 10555 |
|  | 4 | 32 | 256 | 608521 | 503813 | 104708 | 18675 | 24689 |
|  | 4 | 32 | 1024 | 608521 | 503813 | 104708 | 18675 | 24689 |
|  | 4 | 64 | 256 | 608521 | 555591 | 52930 | 10806 | 10595 |
|  | 4 | 64 | 1024 | 608521 | 555591 | 52930 | 10806 | 10595 |
|  | 16 | 32 | 256 | 181319 | 76611 | 104708 | 18647 | 24646 |
|  | 16 | 32 | 1024 | 181319 | 76611 | 104708 | 18647 | 24646 |
|  | 16 | 64 | 256 | 181319 | 128389 | 52930 | 10781 | 10555 |
|  | 16 | 64 | 1024 | 181319 | 128389 | 52930 | 10781 | 10555 |

# 

| 099.go | 4 | - | - | - | - | - | - | - |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 16 | - | - | - | - | - | - | - |
|  | 4 | 32 | - | 2741332 | 1655140 | 1086192 | 213278 | 78190 |
|  | 4 | 64 | - | 2741332 | 1876726 | 864606 | 171857 | 62131 |
|  | 16 | 32 | - | 1544754 | 458562 | 1086192 | 209208 | 76624 |
|  | 16 | 64 | - | 1544754 | 680148 | 864606 | 168893 | 60975 |
|  | 4 | 32 | 256 | 2741332 | 1655140 | 1086192 | 213278 | 78190 |
|  | 4 | 32 | 1024 | 2741332 | 1655140 | 1086192 | 213278 | 78190 |
|  | 4 | 64 | 256 | 2741332 | 1876726 | 864606 | 171857 | 62131 |
|  | 4 | 64 | 1024 | 2741332 | 1876726 | 864606 | 171857 | 62131 |
|  | 16 | 32 | 256 | 1544754 | 458562 | 1086192 | 209208 | 76624 |
|  | 16 | 32 | 1024 | 1544754 | 458562 | 1086192 | 209208 | 76624 |
|  | 16 | 64 | 256 | 1544754 | 680148 | 864606 | 168893 | 60975 |
|  | 16 | 64 | 1024 | 1544754 | 680148 | 864606 | 168893 | 60975 |

# 

| 124.m88ksim | 4 | - | - | - | - | - | - | - |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 16 | - | - | - | - | - | - | - |
|  | 4 | 32 | - | 14187325 | 10795197 | 3392128 | 563854 | 383 |
|  | 4 | 64 | - | 14187325 | 11126514 | 3060811 | 486387 | 189 |
|  | 16 | 32 | - | 6983667 | 3591539 | 3392128 | 563849 | 379 |
|  | 16 | 64 | - | 6983667 | 3922856 | 3060811 | 486387 | 188 |
|  | 4 | 32 | 256 | 14187325 | 10795197 | 3392128 | 563854 | 383 |
|  | 4 | 32 | 1024 | 14187325 | 10795197 | 3392128 | 563854 | 383 |
|  | 4 | 64 | 256 | 14187325 | 11126514 | 3060811 | 486387 | 189 |
|  | 4 | 64 | 1024 | 14187325 | 11126514 | 3060811 | 486387 | 189 |
|  | 16 | 32 | 256 | 6983667 | 3591539 | 3392128 | 563849 | 379 |
|  | 16 | 32 | 1024 | 6983667 | 3591539 | 3392128 | 563849 | 379 |
|  | 16 | 64 | 256 | 6983667 | 3922856 | 3060811 | 486387 | 188 |
|  | 16 | 64 | 1024 | 6983667 | 3922856 | 3060811 | 486387 | 188 |

# 

## **Layer 3 Statistics**

| Recorded Statistics for Different Configurations for Layer (3) | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Benchmark | Config (L1 size, L2 size, L3 size) | | | Requests | Hits | Misses | Reads to Writes | Writes to Writes |
| 126.gcc | 4 | - | - | - | - | - | - | - |
|  | 16 | - | - | - | - | - | - | - |
|  | 4 | 32 | - | - | - | - | - | - |
|  | 4 | 64 | - | - | - | - | - | - |
|  | 16 | 32 | - | - | - | - | - | - |
|  | 16 | 64 | - | - | - | - | - | - |
|  | 4 | 32 | 256 | 16655 | 10432 | 6223 | 512 | 256 |
|  | 4 | 32 | 1024 | 16655 | 11415 | 5240 | 32 | 18 |
|  | 4 | 64 | 256 | 11206 | 4983 | 6223 | 511 | 256 |
|  | 4 | 64 | 1024 | 11206 | 5966 | 5240 | 32 | 18 |
|  | 16 | 32 | 256 | 16655 | 10432 | 6223 | 512 | 256 |
|  | 16 | 32 | 1024 | 16655 | 11415 | 5240 | 32 | 18 |
|  | 16 | 64 | 256 | 11206 | 4983 | 6223 | 511 | 256 |
|  | 16 | 64 | 1024 | 11206 | 5966 | 5240 | 32 | 18 |

# 

# 

| 129.compress | 4 | - | - | - | - | - | - | - |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 16 | - | - | - | - | - | - | - |
|  | 4 | 32 | - | - | - | - | - | - |
|  | 4 | 64 | - | - | - | - | - | - |
|  | 16 | 32 | - | - | - | - | - | - |
|  | 16 | 64 | - | - | - | - | - | - |
|  | 4 | 32 | 256 | 1129 | 439 | 690 | 1 | 0 |
|  | 4 | 32 | 1024 | 1129 | 440 | 689 | 0 | 0 |
|  | 4 | 64 | 256 | 926 | 236 | 690 | 1 | 0 |
|  | 4 | 64 | 1024 | 926 | 237 | 689 | 0 | 0 |
|  | 16 | 32 | 256 | 1129 | 439 | 690 | 1 | 0 |
|  | 16 | 32 | 1024 | 1129 | 440 | 689 | 0 | 0 |
|  | 16 | 64 | 256 | 926 | 236 | 690 | 1 | 0 |
|  | 16 | 64 | 1024 | 926 | 237 | 689 | 0 | 0 |

# 

| 132.ijpeg | 4 | - | - | - | - | - | - | - |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 16 | - | - | - | - | - | - | - |
|  | 4 | 32 | - | - | - | - | - | - |
|  | 4 | 64 | - | - | - | - | - | - |
|  | 16 | 32 | - | - | - | - | - | - |
|  | 16 | 64 | - | - | - | - | - | - |
|  | 4 | 32 | 256 | 134289 | 132497 | 1792 | 8 | 4 |
|  | 4 | 32 | 1024 | 134289 | 132499 | 1790 | 2 | 2 |
|  | 4 | 64 | 256 | 134061 | 132269 | 1792 | 8 | 4 |
|  | 4 | 64 | 1024 | 134061 | 132271 | 1790 | 2 | 2 |
|  | 16 | 32 | 256 | 134289 | 132497 | 1792 | 8 | 4 |
|  | 16 | 32 | 1024 | 134289 | 132499 | 1790 | 2 | 2 |
|  | 16 | 64 | 256 | 134061 | 132269 | 1792 | 8 | 4 |
|  | 16 | 64 | 1024 | 134061 | 132271 | 1790 | 2 | 2 |

# 

| 134.perl | 4 | - | - | - | - | - | - | - |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 16 | - | - | - | - | - | - | - |
|  | 4 | 32 | - | - | - | - | - | - |
|  | 4 | 64 | - | - | - | - | - | - |
|  | 16 | 32 | - | - | - | - | - | - |
|  | 16 | 64 | - | - | - | - | - | - |
|  | 4 | 32 | 256 | 104708 | 102170 | 2538 | 135 | 37 |
|  | 4 | 32 | 1024 | 104708 | 102536 | 2172 | 11 | 10 |
|  | 4 | 64 | 256 | 52930 | 50392 | 2538 | 135 | 37 |
|  | 4 | 64 | 1024 | 52930 | 50758 | 2172 | 11 | 10 |
|  | 16 | 32 | 256 | 104708 | 102170 | 2538 | 135 | 37 |
|  | 16 | 32 | 1024 | 104708 | 102536 | 2172 | 11 | 10 |
|  | 16 | 64 | 256 | 52930 | 50392 | 2538 | 135 | 37 |
|  | 16 | 64 | 1024 | 52930 | 50758 | 2172 | 11 | 10 |

# 

| 099.go | 4 | - | - | - | - | - | - | - |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 16 | - | - | - | - | - | - | - |
|  | 4 | 32 | - | - | - | - | - | - |
|  | 4 | 64 | - | - | - | - | - | - |
|  | 16 | 32 | - | - | - | - | - | - |
|  | 16 | 64 | - | - | - | - | - | - |
|  | 4 | 32 | 256 | 1086192 | 1072390 | 13802 | 1773 | 857 |
|  | 4 | 32 | 1024 | 1086192 | 1079354 | 6838 | 70 | 6 |
|  | 4 | 64 | 256 | 864606 | 850804 | 13802 | 1769 | 847 |
|  | 4 | 64 | 1024 | 864606 | 857768 | 6838 | 70 | 6 |
|  | 16 | 32 | 256 | 1086192 | 1072390 | 13802 | 1773 | 857 |
|  | 16 | 32 | 1024 | 1086192 | 1079354 | 6838 | 70 | 6 |
|  | 16 | 64 | 256 | 864606 | 850804 | 13802 | 1769 | 847 |
|  | 16 | 64 | 1024 | 864606 | 857768 | 6838 | 70 | 6 |

# 

| 124.m88ksim | 4 | - | - | - | - | - | - | - |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 16 | - | - | - | - | - | - | - |
|  | 4 | 32 | - | - | - | - | - | - |
|  | 4 | 64 | - | - | - | - | - | - |
|  | 16 | 32 | - | - | - | - | - | - |
|  | 16 | 64 | - | - | - | - | - | - |
|  | 4 | 32 | 256 | 3392128 | 3390189 | 1939 | 60 | 22 |
|  | 4 | 32 | 1024 | 3392128 | 3390270 | 1858 | 15 | 0 |
|  | 4 | 64 | 256 | 3060811 | 3058872 | 1939 | 60 | 22 |
|  | 4 | 64 | 1024 | 3060811 | 3058953 | 1858 | 15 | 0 |
|  | 16 | 32 | 256 | 3392128 | 3390189 | 1939 | 60 | 22 |
|  | 16 | 32 | 1024 | 3392128 | 3390270 | 1858 | 15 | 0 |
|  | 16 | 64 | 256 | 3060811 | 3058872 | 1939 | 60 | 22 |
|  | 16 | 64 | 1024 | 3060811 | 3058953 | 1858 | 15 | 0 |

# 

# 

# 

## main.c

/\*

===========================================================================

PROJECT: Direct-Mapped Write-Back Cache [Trace Driven Simulation]

===========================================================================

NAME: Tyler Neal

USER ID: tpneal

DATE: 04/17/2024

FILE NAME: cache.c

PROGRAM PURPOSE:

This file contains the initialization logic for the various structures

required to model a direct-mapped write-back cache. This includes

structures for cache blocks, sets, and the complete cache.

===========================================================================

\*/

#include "cache.h"

int cache\_layers = 0;

/\*\*

\* @brief Simulates a (1-3) layer direct-mapped write-back cache and prints statistics given inputed tracer files

\*

\* @param argc

\* @param argv

\* @return int

\*/

int main(int argc, char \*argv[]) {

clock\_t start\_time, end\_time; // Declare variables for timing

double elapsed\_time;

start\_time = clock(); // Record the start time

// Verify command-line arguments

if (argc < 8) {

fprintf(stderr, "\nUsage: %s <cache\_type> <line\_size> <cache\_layers> <L1\_size> <L2\_size> <L3\_size> <print\_style>\n", argv[0]);

exit(EXIT\_FAILURE);

}

// Parse command-line arguments

char cache\_type = \*(char\*)argv[1];

int line\_size = atoi(argv[2]) \* 4;

cache\_layers = atoi(argv[3]);

int print\_style = atoi(argv[7]);

// Initialize cache layers

Cache\* cache[3];

Cache \*L1, \*L2, \*L3;

int L1\_size = atoi(argv[4]) \* 1024; // Total size of L1 cache in bytes

int L2\_size = atoi(argv[5]) \* 1024; // Total size of L2 cache in bytes

int L3\_size = atoi(argv[6]) \* 1024; // Total size of L3 cache in bytes

// Setup L1

L1 = setupCache(1, L1\_size, line\_size);

cache[0] = L1;

// Setup L2

if (cache\_layers > 1) {

L2 = setupCache(2, L2\_size, line\_size);

cache[1] = L2;

}

// Setup L3

if (cache\_layers > 2) {

L3 = setupCache(3, L3\_size, line\_size);

cache[2] = L3;

}

// Process requests until end of file

char ch;

bool data\_found = false;

char buffer[11];

buffer[10] = '\0';

Request\* request = createRequest();

while ((ch = getchar()) != EOF) {

if (ch == '@') { // Tracer found

fgets(buffer, sizeof(buffer), stdin); // Reads trace format: @<I/D><R/W><hex-address>

for (int i = 0; i < cache\_layers; i++) { // Iterate layers until hit

formatRequest(request, cache[i], buffer);

if (request->ref\_type == cache\_type || cache\_type == 'U') {

cache[i]->requests++;

processRequest(request, cache[i], &data\_found); // Process request in appropriate cache

if (data\_found) break; // Break if hit was found

}

}

data\_found = false;

}

}

// Calculate miss rates

float miss\_rates[3];

for (int i = 0; i < cache\_layers; i++) {

miss\_rates[i] = ((float)cache[i]->misses / (float)cache[i]->requests) ;

}

// Print cache statistics

for (int i = 0; i < cache\_layers; i++) {

printCacheStats(cache[i], print\_style);

}

printf("------------------------------------------------------------\n");

if (cache\_layers == 1) {

printf("AMAT: %.2f\n", (HIT\_TIME\_L1 + (miss\_rates[0] \* MEM\_ACCESS\_TIME)));

}

if (cache\_layers == 2) {

printf("AMAT: %.2f\n", (HIT\_TIME\_L1 + (miss\_rates[0] \* (HIT\_TIME\_L2 + (miss\_rates[1] \* MEM\_ACCESS\_TIME)))));

}

if (cache\_layers == 3) {

printf("AMAT: %.2f\n", (HIT\_TIME\_L1 + (miss\_rates[0] \* (HIT\_TIME\_L2 + (miss\_rates[1] \* HIT\_TIME\_L3 + (miss\_rates[2] \* MEM\_ACCESS\_TIME))))));

}

printf("------------------------------------------------------------\n");

// Clean up memory

destroyRequest(request);

for (int i = 0; i < cache\_layers; i++) {

destroyCache(cache[i]);

}

end\_time = clock(); // Record the end time

elapsed\_time = ((double)(end\_time - start\_time)) / CLOCKS\_PER\_SEC; // Calculate elapsed time in seconds

printf("Total Elapsed Time: %.2f seconds\n", elapsed\_time);

return 0; // End of program

}

/\*\*

\* @brief Allocates memory for a cache and returns a pointer to it

\*

\* @param cache\_size

\* @param line\_size

\* @param layer

\* @return Cache\*

\*/

Cache\* constructCache(int cache\_size, int line\_size, int layer) {

// Variables

Cache\* cache;

// Allocate space for cache

cache = (Cache\*)malloc(sizeof(Cache));

if (cache == NULL) {

fprintf(stderr, "Failed to allocate memory for cache.\n");

exit(EXIT\_FAILURE);

}

// Initialize cache struct values

cache->cache\_size = cache\_size;

cache->line\_size = line\_size;

cache->num\_lines = (cache\_size / line\_size);

cache->layer = layer;

cache->requests = 0;

cache->hits = 0;

cache->misses = 0;

cache->read\_to\_write = 0;

cache->write\_to\_write = 0;

// Memory allocation for cache lines

cache->lines = (Line\*)malloc(cache->num\_lines \* sizeof(Line));

if (cache->lines == NULL) {

fprintf(stderr, "Failed to allocate memory for cache lines.\n");

exit(EXIT\_FAILURE);

}

for (int i = 0; i < cache->num\_lines; i++) {

// Setting initial values for each cache line

cache->lines[i].dirty = 0; // Marks line as initially clean (not modified)

cache->lines[i].tag[0] = 'x'; // Initial tag set to 'x' to represent uninitialized

}

if (DEBUG) {

printf("Cache Created\n..............\nSize: %d bytes\nLine Count: %d\nLine Size: %d bytes\n",

cache->cache\_size, cache->num\_lines, cache->line\_size);

}

return cache;

}

/\*\*

\* @brief Deallocates memory for a cache given its pointer

\*

\* @param cache

\*/

void destroyCache(Cache\* cache) {

if (cache != NULL) {

if (cache->lines != NULL) {

free(cache->lines);

}

free(cache);

}

if (DEBUG) {

printf("Cache successfully deleted\n");

}

}

/\*\*

\* @brief Cacluates and stores the address field sizes for a given cache

\*

\* @param layer

\* @param cache\_size

\* @param line\_size

\* @return Cache\*

\*/

Cache\* setupCache(int layer, int cache\_size, int line\_size) {

Cache\* cache = constructCache(cache\_size, line\_size, layer);

// Calculate Address Field Sizes

cache->offset\_size = (int)ceil(log2(line\_size));

cache->index\_size = (int)ceil(log2(cache\_size / line\_size));

cache->tag\_size = INSTRUCTION\_SIZE - cache->index\_size - cache->offset\_size;

// Debug Printing

if (DEBUG) {

printf("\nCache Size: %d\nLine Size: %d\ncache->tag\_size: %d\ncache->index\_size: %d\ncache->offset\_size: %d\n\n",

cache\_size, line\_size, cache->tag\_size, cache->index\_size, cache->offset\_size);

}

return cache;

}

/\*\*

\* @brief Allocates memory for a memory request

\*

\* @return Request\*

\*/

Request\* createRequest() {

// Allocate memory for Request

Request\* request = (Request\*)malloc(sizeof(Request));

if (request == NULL) {

fprintf(stderr, "Failed to allocate memory for request.\n");

exit(EXIT\_FAILURE);

}

return request;

}

/\*\*

\* @brief Deallocates memory for a request given its pointer

\*

\* @param request

\*/

void destroyRequest(Request\* request) {

if (request != NULL) {

free(request);

}

}

/\*\*

\* @brief Formats the tag index and offset of a request given the cache it will query

\*

\* @param request

\* @param cache

\* @param buffer

\*/

void formatRequest(Request\* request, Cache\* cache, char\* buffer) {

/\* Assign Characteristics Based on Tracer \*/

if (buffer[0] == 'I') request->ref\_type = 'I';

else if (buffer[0] == 'D') request->ref\_type = 'D';

else {

printf("Error: invalid request reference type of %c.\n", buffer[0]);

exit(EXIT\_FAILURE);

}

if (buffer[1] == 'R') request->access\_type = 'R';

else if (buffer[1] == 'W') request->access\_type = 'W';

else {

printf("Error: invalid request access type of %c.\n", buffer[1]);

exit(EXIT\_FAILURE);

}

// Fill in hex address

sscanf(buffer + 2, "%x", &request->address);

// Convert address to binary representation (placeholder function itob)

char binary[33];

char\* tmp = itob(request->address);

strcpy(binary, tmp);

free(tmp);

// Retrieve and copy tag, index, and offset

strncpy(request->tag, binary, cache->tag\_size);

request->tag[cache->tag\_size] = '\0';

strncpy(request->index, binary + cache->tag\_size, cache->index\_size);

request->index[cache->index\_size] = '\0';

strncpy(request->offset, binary + cache->tag\_size + cache->index\_size, cache->offset\_size);

request->offset[cache->offset\_size] = '\0';

}

/\*\*

\* @brief Takes a request and sends the read / write request to the supplied cache

\*

\* @param request

\* @param cache

\* @param data\_found

\*/

void processRequest(Request\* request, Cache\* cache, bool\* data\_found) {

if (cache == NULL || request == NULL) {

printf("Error: must operate on a valid cache and request.\n");

exit(EXIT\_FAILURE);

}

if (request->access\_type == 'R') {

readData(cache, request, data\_found);

} else if (request->access\_type == 'W') {

writeData(cache, request, data\_found);

} else {

printf("Error: invalid request access type during processRequest().\n");

exit(EXIT\_FAILURE);

}

}

/\*\*

\* @brief Reads data from a cache at the address specified in the request

\*

\* @param cache

\* @param request

\* @param data\_found

\*/

void readData(Cache\* cache, Request\* request, bool\* data\_found) {

int index = btoi(request->index);

if (index >= 0 && index <= cache->num\_lines) {

Line\* line = &cache->lines[index];

// Check if the tag matches

if (strcmp(line->tag, request->tag) == 0) {

cache->hits++;

\*data\_found = true;

} else {

cache->misses++;

if (line->dirty == 1) {

cache->read\_to\_write++;

}

// Load the new tag, mark as clean

strcpy(line->tag, request->tag);

line->dirty = 0;

}

}

}

/\*\*

\* @brief Writes data from a cache at the address specified in the request

\*

\* @param cache

\* @param request

\* @param data\_found

\*/

void writeData(Cache\* cache, Request\* request, bool\* data\_found) {

int index = btoi(request->index);

if (index >= 0 && index <= cache->num\_lines) {

Line\* line = &cache->lines[index];

// If line found in cache

if (strcmp(line->tag, request->tag) == 0) {

cache->hits++;

line->dirty = 1; // Data is now modified

\*data\_found = true;

} else {

cache->misses++;

if (line->dirty == 1) {

cache->write\_to\_write++;

}

// Load the new tag, mark as clean

strcpy(line->tag, request->tag);

line->dirty = 1;

}

}

}

/\*\*

\* @brief Prints cache statistics (style = 1: print for project part 1) (style = 1: print for project part 2)

\*

\* @param cache

\* @param style

\*/

void printCacheStats(Cache\* cache, int style) {

if (cache == NULL) {

printf("Error: must operate on a valid cache\n");

exit(EXIT\_FAILURE);

}

if (style == 1) {

// Output for Part 1

printf("Total Requests: %d\n", cache->requests);

printf(" Miss Rate: %.2f%%\n", ((float)cache->misses / (float)cache->requests) \* 100);

printf("------------------------------------------------------------\n");

} else if (style == 2) {

// Output for Part 2

printf("Cache Layer: L%d\n", cache->layer);

printf("----------------\n");

printf("Configuration:\n");

printf(" Size: %d bytes\n", cache->cache\_size);

printf(" Line Size: %d bytes\n", cache->line\_size);

printf(" Line Count: %d\n", cache->num\_lines);

printf("Performance Metrics:\n");

printf(" Total Requests: %d\n", cache->requests);

printf(" Hits: %d\n", cache->hits);

printf(" Misses: %d\n", cache->misses);

printf(" Hit Rate: %.2f%%\n", ((float)cache->hits / (float)cache->requests) \* 100);

printf(" Miss Rate: %.2f%%\n", ((float)cache->misses / (float)cache->requests) \* 100);

printf(" Read to Write Ratio: %d\n", cache->read\_to\_write);

printf(" Write to Write Ratio: %d\n", cache->write\_to\_write);

}

}

/\*\*

\* @brief Converts an integer to binary

\*

\* @param num

\* @return char\*

\*/

char\* itob(int num) {

size\_t numBits = sizeof(int) \* 8;

char\* binaryStr = (char\*)malloc(numBits + 1);

if (binaryStr == NULL) {

fprintf(stderr, "Memory allocation failed.\n");

return NULL;

}

binaryStr[numBits] = '\0'; // Null terminator

for (size\_t i = 0; i < numBits; ++i) {

binaryStr[numBits - 1 - i] = (num & (1 << i)) ? '1' : '0';

}

return binaryStr;

}

/\*\*

\* @brief Converts binary to an integer

\*

\* @param binary

\* @return int

\*/

int btoi(char\* binary) {

int value = 0;

size\_t len = strlen(binary); // Get the length of the binary string

for (size\_t i = 0; i < len; ++i) {

value <<= 1; // Shift the current value to the left by one bit

if (binary[i] == '1') {

value += 1; // Add 1 if the current binary digit is 1

} else if (binary[i] != '0') {

// Handle invalid characters in the binary string

fprintf(stderr, "Invalid character '%c' in binary string.\n", binary[i]);

return 0; // Return 0 or an appropriate error value

}

}

return value;

}

## cache.h

/\*

===========================================================================

PROJECT: Direct-Mapped Write-Back Cache [Trace Driven Simulation]

===========================================================================

NAME : Tyler Neal

USER ID : tpneal

DATE : 03/25/2024

FILE NAME : cache.h

PROGRAM PURPOSE:

This header file declares the structures necessary to model a direct-mapped

write-back cache. It provides declarations for cache blocks, sets, and the

entire cache structure.

PSEUDO :

1. Process relevant simulation information such as line\_size, cache\_size,

ect.

2. Instantiate cache, as well as its set of lines.

3. Parse stdin for address references.

4. Decode address reference into request.

5. Process request as a read or write operation

===========================================================================

\*/

#ifndef CACHE\_H

#define CACHE\_H

#include <math.h>

#include <stdbool.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

// Memory request times according to project specs

#define HIT\_TIME\_L1 1

#define HIT\_TIME\_L2 16

#define HIT\_TIME\_L3 64

#define MEM\_ACCESS\_TIME 100

// According to our implementation, these values wont exceed their defined max

#define INSTRUCTION\_SIZE 32

#define MAX\_TAG\_SIZE 21

#define MAX\_INDEX\_SIZE 16

#define MAX\_OFFSET\_SIZE 5

// Used for debug printing

#define DEBUG false

typedef struct Line {

int dirty;

char tag[MAX\_TAG\_SIZE];

} Line;

typedef struct Cache {

// Cache Details

int cache\_size;

int line\_size;

int num\_lines;

int layer;

Line \*lines;

// Feild Sizes

int tag\_size;

int index\_size;

int offset\_size;

// Recorded Metrics

int requests;

int hits;

int misses;

int read\_to\_write;

int write\_to\_write;

} Cache;

typedef struct Request {

char ref\_type;

char access\_type;

unsigned int address; // Hex address formatted as int

char tag[MAX\_TAG\_SIZE];

char index[MAX\_INDEX\_SIZE];

char offset[MAX\_OFFSET\_SIZE];

} Request;

// Function prototypes

Cache\* constructCache(int cache\_size, int line\_size, int layer);

void destroyCache(Cache \*cache);

Cache\* setupCache(int layer, int cache\_size, int line\_size);

Request\* createRequest();

void destroyRequest(Request \*request);

void formatRequest(Request\* request, Cache\* cache, char\* buffer);

void processRequest(Request \*request, Cache \*cache, bool \*data\_found);

void readData(Cache \*cache, Request \*request, bool \*data\_found);

void writeData(Cache \*cache, Request \*request, bool \*data\_found);

void printCacheStats(Cache \*cache, int style);

char\* itob(int num);

int btoi(char \*binary);

#endif // CACHE\_H

## Makefile

CC = gcc

FLAGS = -Wall -Wextra

OBJ\_FILES = main.o

EXE\_FILE = cache\_exec

.PHONY: all clean

all: $(EXE\_FILE)

$(EXE\_FILE): $(OBJ\_FILES)

$(CC) $(FLAGS) -o $@ $^ -lm

main.o: main.c cache.h

$(CC) $(FLAGS) -c $< -o $@

clean:

rm -f $(OBJ\_FILES) $(EXE\_FILE)

## 

## run1.sh

**(NOTE: Assumes that parent folder contains child folder “traces” containing tracers)**

# PART 1 -------------------------------------

# Clean up and compile environment

echo "Cleaning up environment and compiling..."

make clean

make all

clear

# Fixed arguments

cache\_layers='1'

empty\_layer='0'

print\_style='1'

# Trace file setup

tracer=('126.gcc' '129.compress' '132.ijpeg' '134.perl' '099.go' '124.m88ksim')

# Array loops

cache\_types=('U' 'I' 'D')

cache\_sizes=('8' '16')

line\_sizes=('4' '8')

echo "Starting cache configuration tests..."

# Loop over every permutation of tracer

for tracer in "${tracer[@]}"; do

tracer\_path="../traces/$tracer" # Concatenate the path to the tracer file

echo -e "\nXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n"

echo -e "\t\t\tTesting trace $tracer..."

echo -e "\nXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n"

for type in "${cache\_types[@]}"; do

echo -e "================================================================\n"

for size in "${cache\_sizes[@]}"; do

for line in "${line\_sizes[@]}"; do

echo -e "------------------------------------------------------------"

echo -e "Configuration: Type $type, Cache Size ${size}kb, Line Size ${line} words"

echo -e "------------------------------------------------------------"

./cache\_exec $type $line $cache\_layers $size $empty\_layer $empty\_layer $print\_style < $tracer\_path

echo -e "------------------------------------------------------------\n"

done

done

done

done

echo "============================================================"

echo -e "\n\t\t\tALL FINISHED!\n"

echo -e "============================================================\n"

## run2.sh

**(NOTE: Assumes that parent folder contains child folder “traces” containing tracers)**

# PART 2 -------------------------------------

# Clean up and compile environment

echo "Cleaning up environment and compiling..."

make clean

make all

clear

# Fixed arguments

cache\_type='D'

line\_size='8'

print\_style='2'

# Trace file setup

tracer=('126.gcc' '129.compress' '132.ijpeg' '134.perl' '099.go' '124.m88ksim')

# Layer counts

layers=('1' '2' '3')

# Cache sizes

L1\_sizes=('4' '16')

L2\_sizes=('32' '64')

L3\_sizes=('256' '1024')

# Counter for configuration number

config\_count=0

config\_count\_1=0

echo "Starting cache configuration tests..."

# Loop over every permutation of layer, cache size as needed

for tracer in "${tracer[@]}"; do

config\_count=0

config\_count\_1=$((config\_count\_1 + 1))

tracer\_path="../traces/$tracer" # Concatenate the path to the trace file

echo -e "\nXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n"

echo -e "\t\tTesting trace $tracer..."

echo -e "\nXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"

for layer in "${layers[@]}"; do

echo -e "\n====================================================================="

echo "Testing configurations for Layer $layer..."

echo -e "=====================================================================\n"

if [[ "$layer" == "1" ]]; then

for L1\_size in "${L1\_sizes[@]}"; do

config\_count=$((config\_count + 1))

echo -e "------------------------------------------------------------"

echo -e "Configuration[$config\_count\_1-$config\_count]: Layers $layer, L1 size ${L1\_size}kb"

echo -e "------------------------------------------------------------"

./cache\_exec $cache\_type $line\_size $layer $L1\_size 0 0 $print\_style < "$tracer\_path"

echo -e "------------------------------------------------------------\n"

done

elif [[ "$layer" == "2" ]]; then

for L1\_size in "${L1\_sizes[@]}"; do

for L2\_size in "${L2\_sizes[@]}"; do

config\_count=$((config\_count + 1))

echo -e "------------------------------------------------------------"

echo -e "Configuration[$config\_count\_1-$config\_count]: Layers $layer, L1 size ${L1\_size}kb, L2 size ${L2\_size}kb"

echo -e "------------------------------------------------------------"

./cache\_exec $cache\_type $line\_size $layer $L1\_size $L2\_size 0 $print\_style < "$tracer\_path"

echo -e "------------------------------------------------------------\n"

done

done

elif [[ "$layer" == "3" ]]; then

for L1\_size in "${L1\_sizes[@]}"; do

for L2\_size in "${L2\_sizes[@]}"; do

for L3\_size in "${L3\_sizes[@]}"; do

config\_count=$((config\_count + 1))

echo -e "------------------------------------------------------------"

echo -e "Configuration[$config\_count\_1-$config\_count]: Layers $layer, L1 size ${L1\_size}kb, L2 size ${L2\_size}kb, L3 size ${L3\_size}kb"

echo -e "------------------------------------------------------------"

./cache\_exec $cache\_type $line\_size $layer $L1\_size $L2\_size $L3\_size $print\_style < "$tracer\_path"

echo -e "------------------------------------------------------------\n"

done

done

done

fi

done

done

echo "============================================================"

echo -e "\n\t\t\tALL FINISHED!\n"

echo -e "============================================================\n"