Multi-Core Processor Lab Report 1

Maximilian Kamps (jjb140)

11.02.2025

1 Introduction

This document serves as a report for the Lab project in the Multi-Core Processor Systems course at the University of Amsterdam. It documents the behavior of various levels of cache complexity simulated using SystemC.

2 Assignment 1: Single processor cache

Assignment 1 involves simulating a single 32kiB, 8-way set-associative L1 D-cache with a 32-Byte line size. Consequently, the cache has 128 8-way sets. The cache employs a least-recently-used write-back replacement strategy and an allocate-on-write policy. Cache access is modeled with a 1-cycle delay, while main memory access incurs a 100-cycle delay. Section 2.1 details the abstraction of the cache for simulation, and Section 2.2 evaluates its behavior across various memory access traces.

2.1 Design

Figure 1 illustrates the design of an L1 cache for a single processor. To implement cache behavior, all wires from the CPU to memory are rerouted through the cache, effectively placing it between the two. An additional wire connects the newly added *Status* and *CacheStatus* ports, allowing the cache to inform the CPU whether a requested read or write resulted in a hit or miss. Access to main memory during a miss is modeled implicitly; whenever the cache needs to read or write to main memory, it waits for 100 cycles.

When the cache receives a new address from the CPU, it first computes the block address by dividing the address by the line size (32). The block address is then used to find the set where the address should be inserted by taking its modulo with the cache associativity (8). Finally, the address is either inserted into a new cache line (in case of a miss), which may cause eviction of a different cache line, or, in case of a hit, the existing cache line gets updated. A cache line consists of a dirty bit (set when the cache line has been written to), a valid bit (indicating if the cache line is valid), a timestamp using the simulation time (for LRU eviction), and a tag corresponding to the block address.

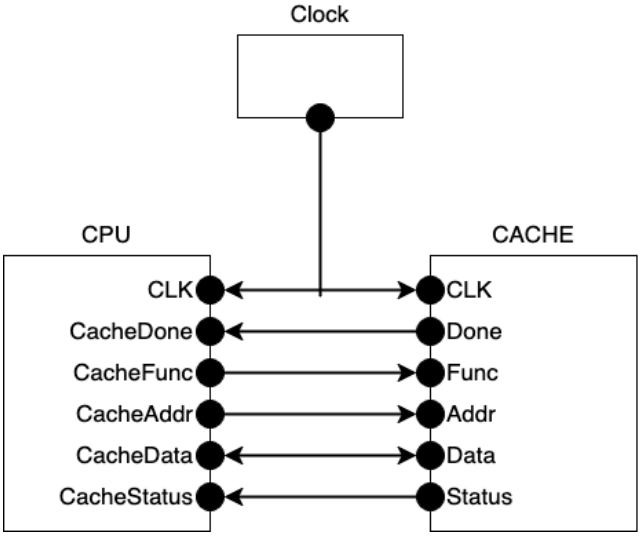

Figure 1: Design of a single processor cache

2.1.1 CLI commands

The command line structure of the assignment has been slightly modified. After the trace file, the user can specify the verbosity (0 for no logging and 1 for logging).

2.2 Simulation

This section presents the results of the cache simulation, evaluating its performance in various scenarios. It compares cache access times, hit rates, and memory delay reductions under different conditions, such as optimization levels and cache sizes. The experiments focus on understanding the impact of cache memory on system performance and highlight the key metrics that define its efficiency.

2.2.1 Cache vs direct memory access

This section explores the reduction in memory delay, when introducing a cache between the main memory and the CPU. Figure 2 compares the memory delay reduction (%) when introducing a cache for different trace files, with the *-O2* compiler optimization enabled and disabled. Using a cache significantly reduces memory delay, achieving reductions of approximately 98.0%-95.3% when *-O2* is disabled compared to direct main memory access without a cache. With *-O2* enabled, the reduction ranges from approximately 98.0%-82.0%. Consequently, utilizing a cache greatly reduced memory delay.

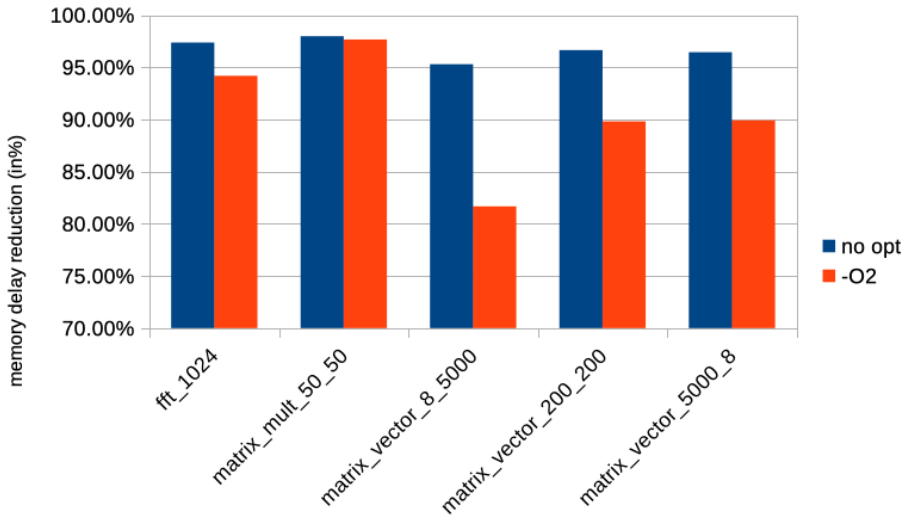

Figure 2: Memory delay reduction by using a Cache (in %) for different traces and optimization levels

2.2.2 Cache hit rate

Figure 3 shows the combined read/write cache hit rate (%) for different trace files, with and without the *-O2* compiler optimization. The chart indicates that enabling *-O2* reduces the hit rate across all traces by approximately 3.2% to 13.6%. Section 2.2.4 investigates the reduction in cache hits caused by the optimization.

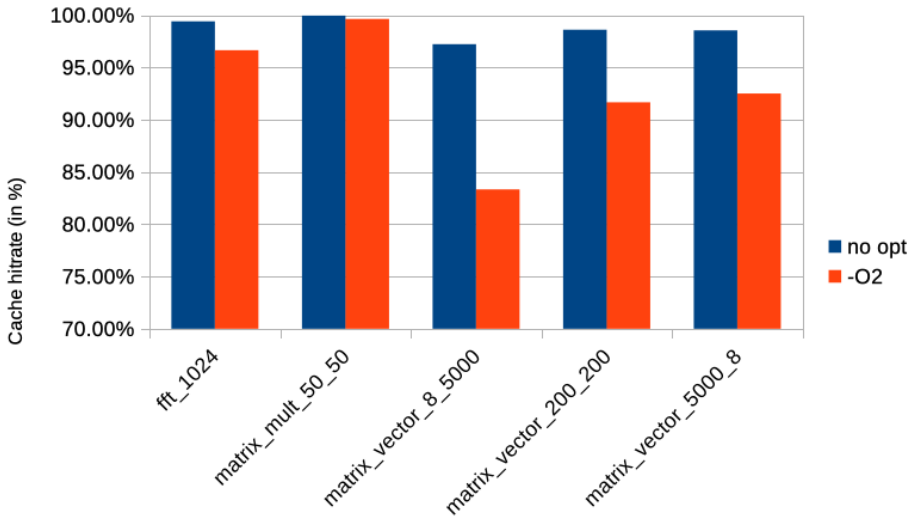

Figure 3: Cache hit rate (in %) for different traces and optimization levels

2.2.3 Absolute memory delay of the Cache

Figure 4 illustrates the total memory delay (in ns). Enabling *-O2* optimizations significantly reduces memory delay across all traces, with reductions ranging from 35% to 84%. The most substantial reduction is observed in the matrix multiplication trace (*matrix_mult_50_50*). Section 2.2.4 investigates the reduction in memory delay caused by the optimization.

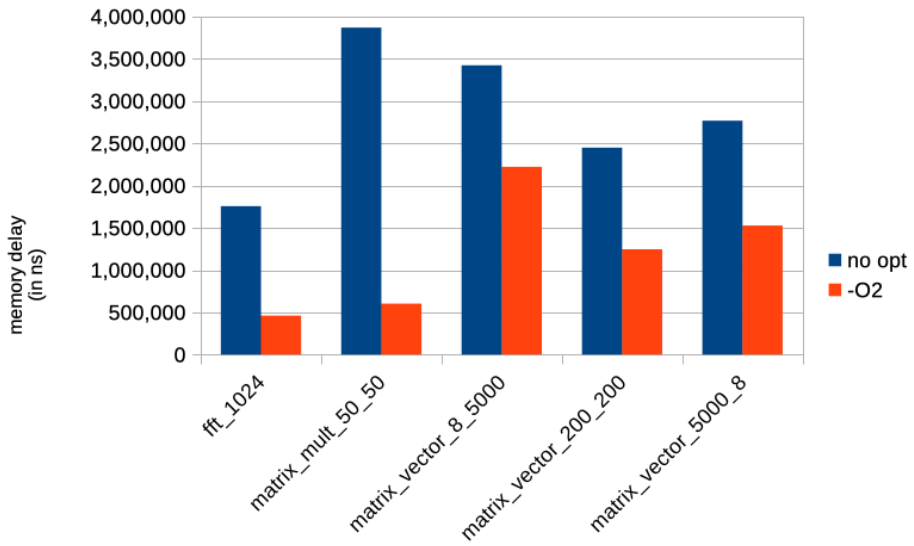

Figure 4: Memory delay (in ns) for different traces and optimization levels

2.2.4 Consequences of optimizing with -O2

To investigate the improved memory delay when enabling *-O2*, Figure 5 shows the number of reads and writes in the trace files with and without *-O2* enabled.

Trace files without optimization exhibit significantly more reads than writes. Comparing across optimizations, the *-O2* files contain substantially fewer reads and writes. The most notable difference is observed in the trace (*matrix_mult_50_50*), where the optimized file contains only 15% of the reads and 1% of the writes compared to its non-optimized counterpart. Consequently, the gap in memory delay shown in Figure 4 is explained by simulations using *-O2* traces, which perform significantly fewer reads and writes while maintaining a similar cache hit rate, as shown in Figure 3. In conclusion, *-O2* reduces the number of memory operations required while trying to preserve caching accuracy.

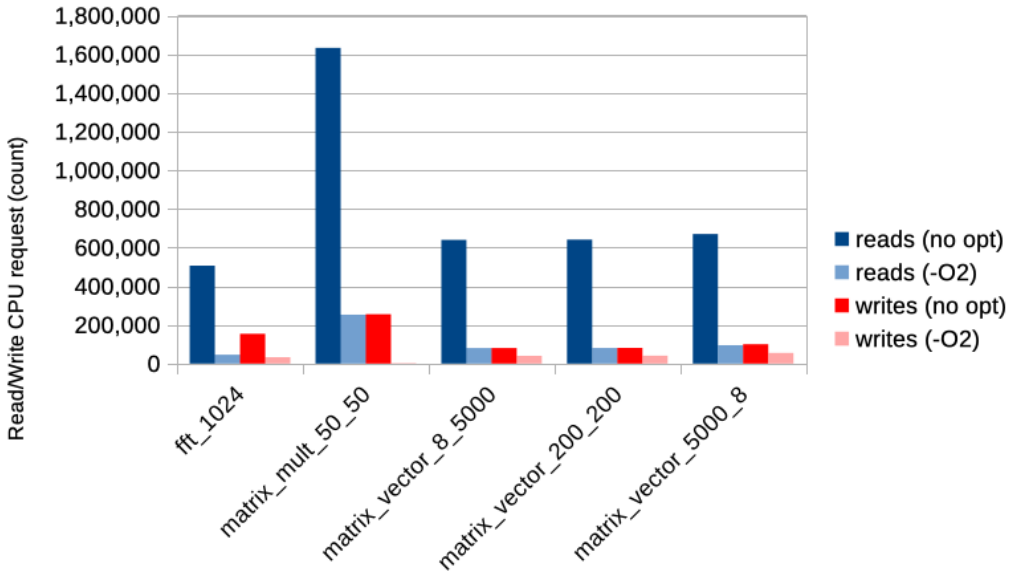

Figure 5: Number of reads and writes in different traces with different optimization levels

2.2.5 Cache size vs hit rate

Figure 6 plots the effect of different cache sizes (in B) on the cache hit rate (in %) for the fast furrier transformation and matrix multiplication traces, with the `-O2` optimization enabled and disabled. It is expected that increasing the cache size should also increase

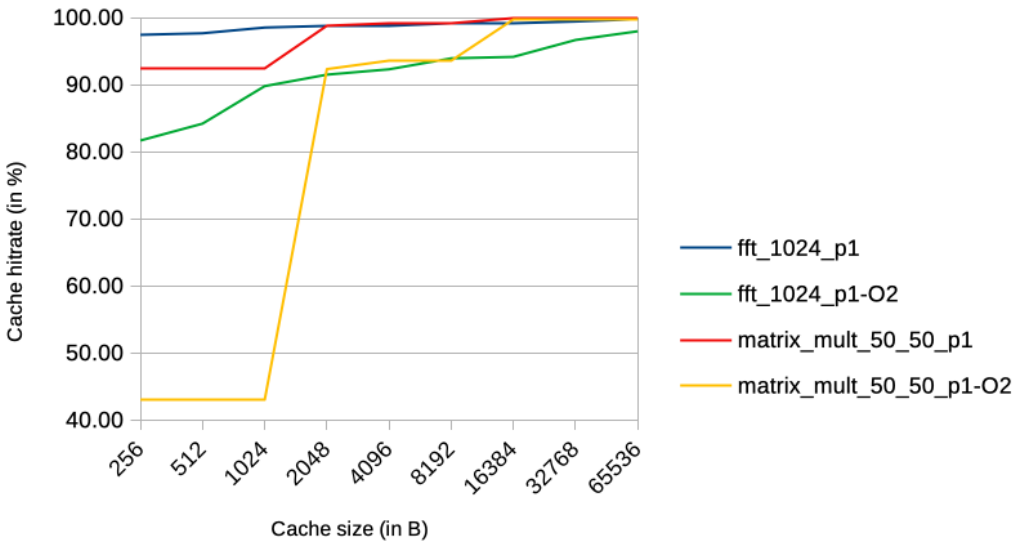

Figure 6: Cache hits (in %) for different traces and cache sizes (in B)

the hit rate. This is because the cache evicts lines less frequently, leading to fewer costly memory swaps between main memory and the cache. The plot supports this hypothesis. For the matrix multiplication transformation without optimizations, increasing the cache size

causes a gradual rise in cache hits from around 81.66% to 97.94%. When considering the same algorithm with optimizations enabled, the increase is far more dramatic. Here, the cache registers only 43.05% hits before increasing the cache from 1024B to 2048B, which causes a spike to 92.3%. Further increasing the cache size gradually raises the accuracy to 99.63%. The difference caused by optimization levels is likely due to memory access, with `-O2` being far less predictable, leading to a significantly higher number of evictions compared to its unoptimized counterpart. Consequently, cache hit rates remain relatively stable for the unoptimized trace, even for small caches. In conclusion, when running an algorithm that requires substantial memory on a very small cache, it can be beneficial to disable optimizations.