Analyzing Different Cache Structures Performance Base on Trace-Driven Simulation

Chu Seong Jae

School of Undergraduate Studies, DGIST

Abstract

Previous research was conducted about cache structures and cache performance. Analyzing cache read-write miss rate based on trace-driven simulation is proposed to measure cache performance. In this paper, we focused on the performance gap between cache structures with different cache parameters and different cache eviction policies. We measured the cache performance of different cache structures with a trace-driven cache simulation program. We found that the choice of cache eviction policy could increase the total performance of cache. We found that cache parameters should be composed to be associated with the program's characteristics to increase cache performance. We expanded cache performance analysis with various cache parameters that previous work neglected.

Analyzing Different Cache Structures Performance Base on Trace-Driven Simulation

Semiconductor technology evolves over several years, and the CPU runs faster than memory (Randal & David, 2016, p. 49). This can cause a bottleneck between the CPU and memory as the CPU can process instructions before fetching instructions from memory. This bottleneck problem can exist between memory and disk, CPU, and external network.

Computer system designers place cache memory between the CPU and memory to solve the CPU-memory bottleneck (Randal & David, 2016, p.49). Computer system designers first implement small SRAM (Static Random Access Memory) cache memory termed L1 cache between CPU and memory; as the CPU-memory bottleneck increases, computer system designers place another SRAM termed L2 cache between L1 cache and memory (Randal & David, 2016, p.651). The purpose of cache memory is based on temporal locality and spatial locality. “In a program with good temporal locality, a memory location referenced once is likely to be referenced again multiple times in the near future. In a program with good spatial locality, if a memory location is referenced once, then the program is likely to reference a nearby memory location in the near future” (Randal & David, 2016, p. 641). Cache memory is designed in several structures, such as a fully associative cache, a direct mapping cache, a set associative cache, and a sector cache (Rao, 1978, p.378). Much work has been conducted on caches, including cache management techniques in real-time embedded systems (Giovani et al., 2015).

As cache memory performs an essential role in computer architecture, measuring cache memory performance becomes critical. Cache performance analysis of a two-level demanded paged memory system is conducted in previous work (Rao, 1978, p.378). Peir et al. (1998) suggest that cache performance can be determined by access time (refers to the time to fetch data from the cache) and hit ratio (refers to the section of memory references that the cache can give) (p.1). Peir et al. (1998) also suggest that “Trace-driven simulation is

the standard methodology for the study and evaluation of cache memory design.” (p.1). Trace-driven simulation consists of a simulation program that mimics the cache-memory system to analyze and address traces (sequence of memory reference), which is applied to the simulation program (Richard & Trevor, 1997, p. 129).

In this paper, we made a cache simulation program that can execute trace files and analyze cache performance with the hit ratio metric. We made a cache with different structures inside the simulation program and executed six different trace files at each cache. Then, we explained the hit ratio changes from the output. With this work, we expected to check the usefulness of Trace-driven simulation methods in cache performance analysis and analyze details in cache structure.

Methodology

The cache simulation program was designed to have an L1 cache and an L2 cache. L2 cache is made with inclusive cache. Random policy and LRU (Least Recently Used) policy are implemented for cache block replacement policy. Write-back and write-allocate are used in the L1 and L2 cache. Cache parameters (capacity, associativity, block size) can be modified before executing the simulation. Address trace is obtained from the SPEC (Standard Performance Evaluation Corporation) benchmark. The cache simulation program is written in C++. The cache simulation program is executed on Linux.

To analyze performance effects based on cache parameters, a simulation was conducted with three parts. In the first simulation, the cache block size is fixed at 32 bytes, cache associativity is fixed at 4, while cache capacity is changed by 4, 32, 256, and 1024KB. In the second simulation, the cache block size is fixed at 32 bytes, cache capacity is fixed at 256KB, while cache associativity is changed by 1, 4, 8, and 16. The last simulation is performed with cache associativity fixed with 4, cache capacity is fixed with 256KB, while cache block size is changed by 16, 32, 64, and 128KB. During three simulations, read miss

rate and write miss rate based on cache block replacement policy was obtained.

Result

Tracefies are associated with the following names. Tracefile1 is 400_perlbench, Tracefile2 is 450_soplex, Tracefile3 is 453_povray, Tracefile4 is 462_libquantum, Tracefile5 is 473_astar, and Tracefile6 is 483_xalancbmk. We measured read-write miss in different eviction policies (random, LRU) at every measurement.

Figure 1

Cache read, write miss rate changed by cache capacity size in different tracefiles

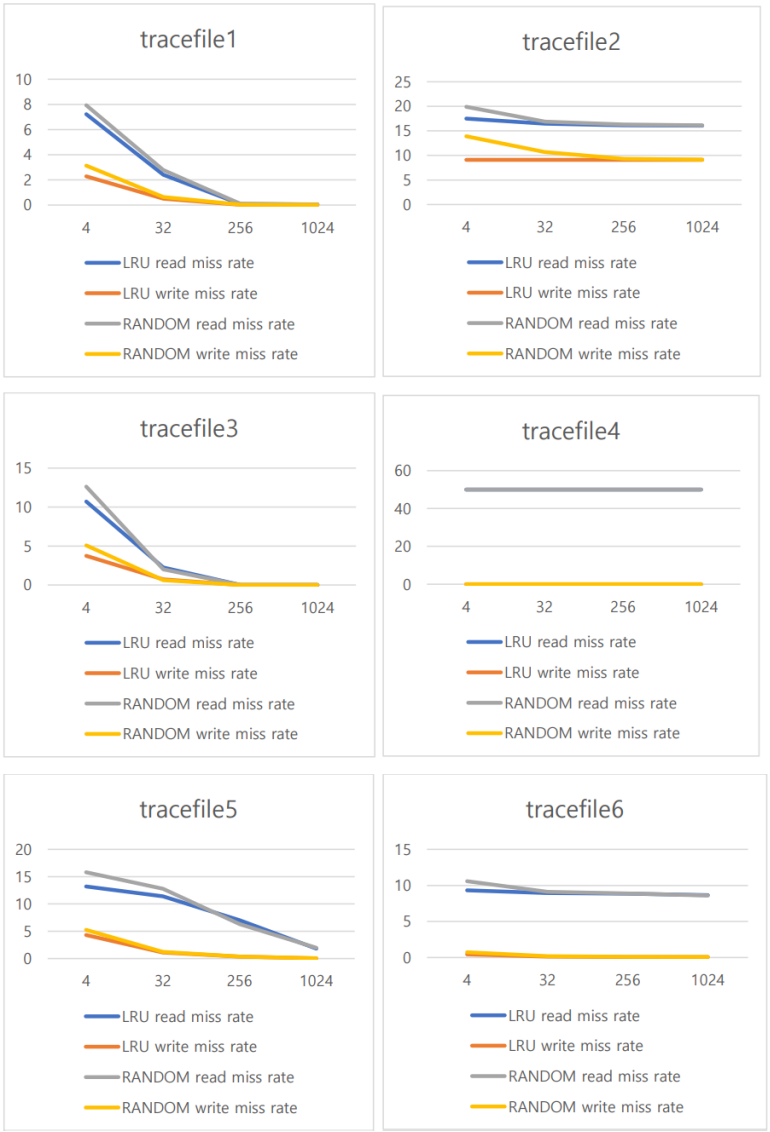

Figure 1 shows the cache read-write miss rate changed by cache capacity size in different trace files. Cache associativity is fixed with 4, and cache block size is fixed with 32B. In Figure 1, without tracefile4, the cache with LRU eviction policy showed a better read-write miss rate than the cache with random eviction policy. Also, tracefile1, tracefile3, and tracefile5 showed that with a larger cache capacity, the miss rate is decreasing. Tracefile2, tracefile4, and tracefile6 have a minor change of miss rate by cache capacity size.

Figure 2

Cache read, write miss rate changed by cache associativity in different tracefiles

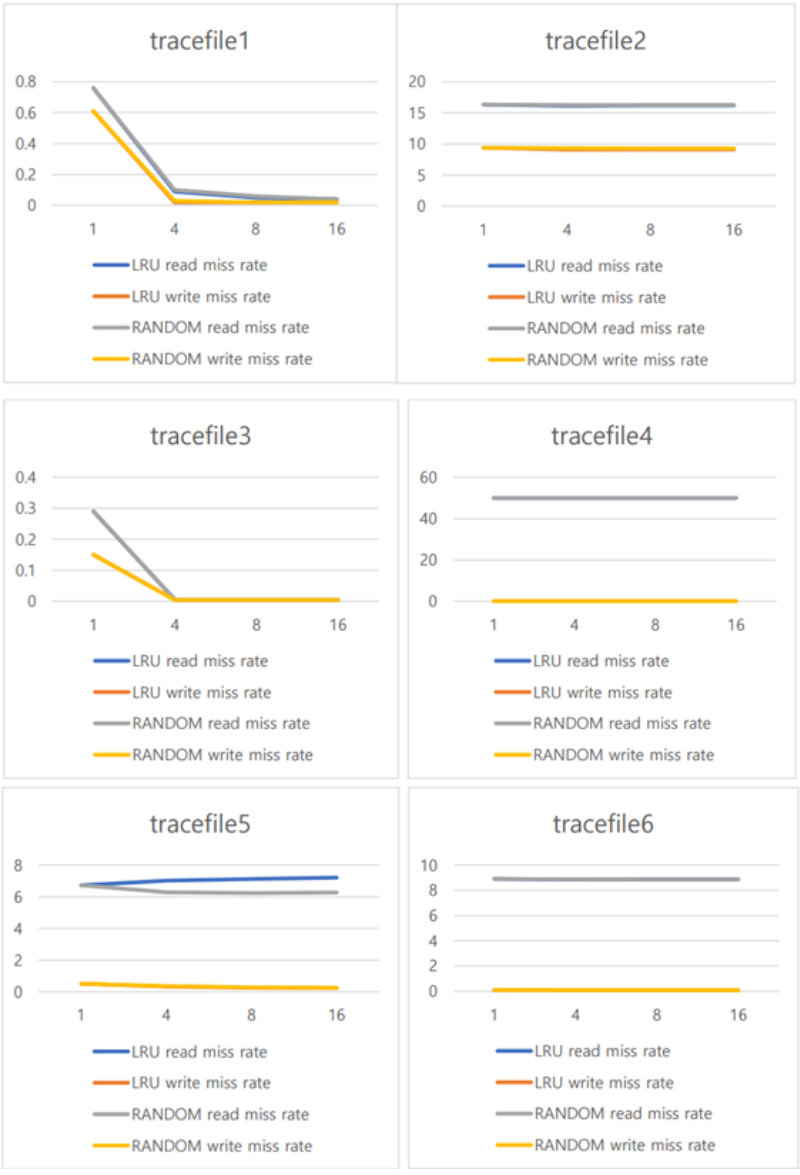

Figure 2 shows the cache read-write miss rate changed by cache associativity in different trace files. The cache capacity is fixed at 256KB, and the cache block size is fixed at 32B. In Figure 2, the miss rate difference between the LRU eviction policy and the random eviction policy is slight. Only tracefile1 and tracefile3's miss rate is improved by increasing cache associativity.

Figure 3

Cache read, write miss rate changed by cache block size in different tracefiles

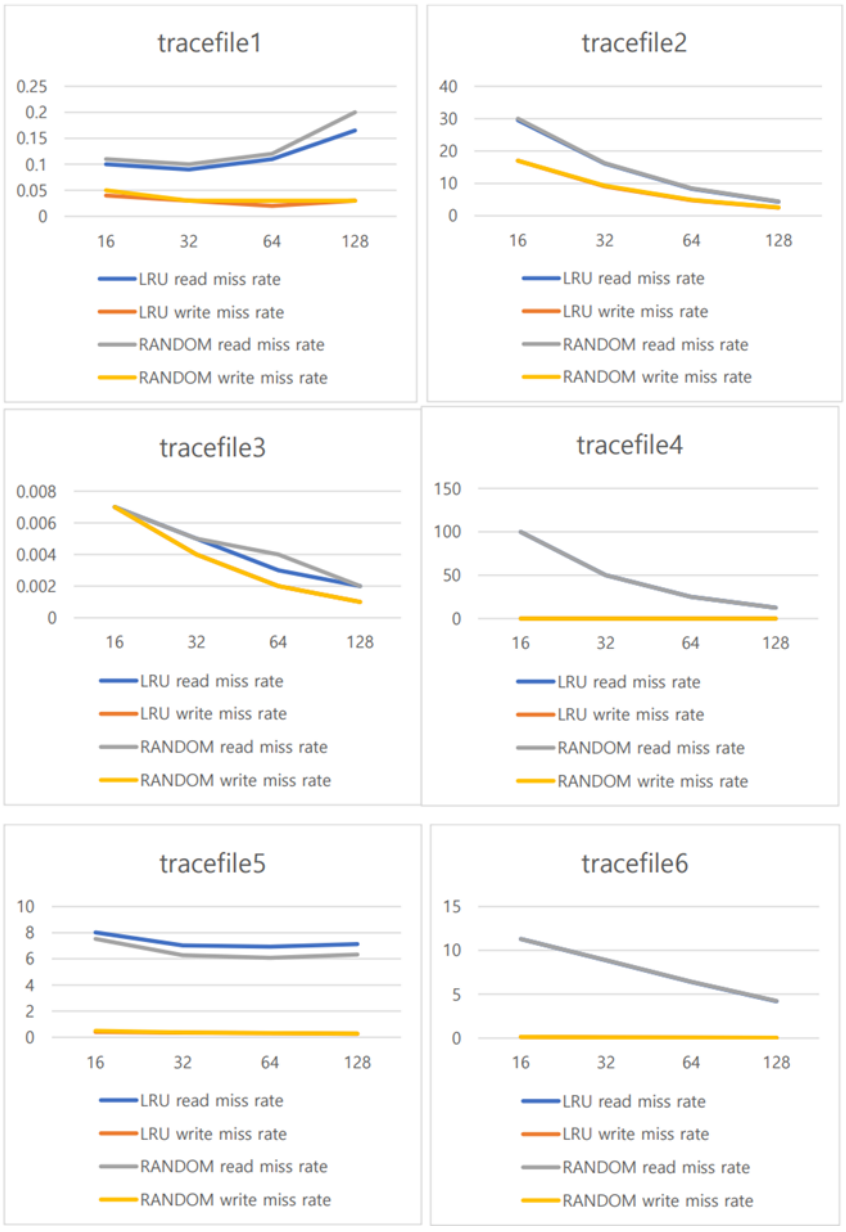

Figure 3 shows the cache read-write miss rate changed by cache block size in different trace files. Cache capacity is fixed at 256KB, and cache associativity is fixed at 4. In Figure 3, the difference in miss rate by eviction policy can be seen in tracefile1, tracefile3, and tracefile5. Unlike Figure 1 and Figure 2, the miss rate of tracefile2, tracefile4, and tracefile6 noticeably decreases by increasing cache block size.

Discussion

Previous work (Rao, 1978) analyzed the performance of four different cache structures (fully associative cache, direct mapping cache, set associative cache, and sector cache). In this work, we expand cache structures by three cache component (cache capacity, cache associativity, and cache block size) and analyze the performance of the cache with miss rate change when cache component size increase. Previous work (Rao, 1978) also analyzed two different cache eviction policies called RR (Random Replacement) and FIFO (First In First Out). In this work, we examine the miss rate difference of RR and LRU.

The result of the simulation explains that increasing cache component size does not always decline the miss rate of cache. Instead, a lower miss rate of cache depends on combinations of cache component size and trace file's characteristics. This expresses that cache structure should be designed to optimize the program runs on a system. In most cases, LRU shows the same or better performance than RR. This means that a change in eviction policy could increase the overall performance of the cache.

Our research used cache performance measurement proposed in (Rao, 1978) and expanded the experiment with trace-driven based simulations. Our work's weakness is that we only measure the performance of previously proposed cache structures, not suggesting new cache structures or new eviction policies. Future work should suggest new cache structures or eviction policies with trace-driven simulation.

References

Giovani, G., Ahmed, A., Renato, M., Antonio, A.F., Rodolfo, P. (2015). A Survey on Cache Management Mechanisms for Real-Time Embedded Systems, *ACM Computing Surveys, 48*(2), 1-36. https://doi.org/10.1145/2830555

Peir, J. K., Hsu, W. W., & Smith, A. J. (1998). *Implementation issues in modern cache memory*. University of California, Berkeley, Computer Science Division. https://www2.eecs.berkeley.edu/Pubs/TechRpts/1998/CSD-98-1023.pdf

Randal, E.B. & David R. O. (2016). *Computer Systems: A Programmer's Perspective*. PEARSON.

Rao, G. S. (1978). Performance analysis of cache memories. *Journal of the ACM (JACM), 25*(3), 378–395. https://doi.org/10.1145/322077.322081

Richard, A. U., & Trevor, N. M. (1997). Trace-driven memory simulation: a survey, *ACM Computing Surveys, 29*(2), 128 – 170. https://doi.org/10.1145/254180.254184