Cache and Memory Performance Profiling Report

Yunhua Fang

September 2024

Introduction

In order to gain a deeper understanding of cache and memory hierarchy in modern computers. We design a set of experiments that will quantitatively reveal the following:

1. The read/write latency of cache and main memory when the queue length is zero (i.e., zero queuing delay)
2. The maximum bandwidth of the main memory under different data access granularity (i.e., 64B, 256B, 1024B) and different read vs. write intensity ratio (i.e., read-only, write-only, 70:30 ratio, 50:50 ratio)
3. The trade-off between read/write latency and throughput of the main memory to demonstrate what the queuing theory predicts
4. The impact of cache miss ratio on the software speed performance (the software is supposed to execute relatively light computations such as multiplication)
5. The impact of TLB table miss ratio on the software speed performance (again, the software is supposed to execute relatively light computations such as multiplication)

Read/Write Latency Check

To test the latency of cache and main memory with zero queueing delays, the environment cleanly runs tests that most of the applications are closed to ensure that no competition between processes. For main memory latency, Intel Memory Latency Checker is applied to test the latency in idle environment (sudo ./mlc −−idle_latency).

memory_latency_test.c respectively tests the read and write latency for different level caches and main memory. In the scripts, we design a long array to access and write it. By adjusting the size of the array, different memory levels can be simulated. L1 Cache size = 64 bytes; L2 Cache size = 512 bytes; L3 Cache size = 4096 bytes; Main Memory (> Last Level Cache Size) size = 32768 bytes. RDTSCP is an instruction used in x86 processors that reads the processor's time-stamp counter (TSC). It provides a high-resolution time measurement commonly used to measure code performance or track the time elapsed between different points in a program. Latency is calculated by Latency (ns) = (Cycles per Access) / (CPU Frequency in GHz). We apply it to measure the accurate latency. The final result seems counter-intuitive on main memory because write latency is smaller than read latency. The reason may be because the script does not prevent prefetching. The feature reduces the latency on write tasks. Table 1 records the read/write latency of different cache levels and main memory.

Memory	L1 Cache	L2 Cache	L3 Cache	Main memory
Read Latency (ns)	3.789	8.996	20.360	69.608
Write Latency (ns)	15.152	16.572	31.723	28.883

Table 1: Read/Write Latency with 0 Queueing Delay

Maximum Bandwidth

This section presents test results about the maximum bandwidth of the main memory under different data access granularity and different read/write workloads. The script maximum_bandwidth.c creates a large array to ensure that the data access can reach the main memory. This ensures that the CPU must access the main memory for these operations, effectively testing the memory subsystem's bandwidth. The memory channel is also saturated

based on the design to reach maximum bandwidth. The program calculates the bandwidth following the equation $Bandwidth = \frac{total\_data\_transferred}{time\_taken}$. Table 2 presents the maximum bandwidth under different data granularity with different read/write ratios. The unit of maximum bandwidth is **GB/s**.

r/w ratio | data granularity	64B	256B	1024B
read-only	26.063	48.676	174.338
write-only	18.195	32.912	100.685
7:3	22.077	39.351	144.509
5:5	21.053	37.186	108.178

Table 2: Maximum Bandwidth

Trade-off between R/W Latency and Throughput

Little's Theorem states that in a stable system, the average number of services L in a queuing system is equal to the average arrival rate λ multiplied by the average time W an service spends in the system: $L = \lambda W$. To see the effectiveness of Little's Law, the experiment involves running a custom C program (little_the.c) that performs a fixed number of read/write operations on a large memory buffer. By adjusting the number of concurrent threads, we simulate different levels of concurrency and measure the resulting throughput and latency. Memory indices in the workload are accessed randomly to prevent cache prefetching and optimize cache usage. Table 3 presents that the throughput increases with the coherence level increases. The average latency on operations also increases because threads need to compete. The number of services in the queue becomes more because of the expanded throughput and longer average latency, which follows Little's theorem.

Threads	Time Taken(s)	Throughput(ops/ns)	Avg Latency(ns)	L
1	6.332	0.157	46	7.296
2	5.045	0.198	46	9.068
4	2.784	0.359	50	18.083
8	1.502	0.666	51	33.892
16	1.094	0.914	53	48.354
32	0.819	1.221	53	64.334

Table 3: Little's Theorem

The Impact of Cache Miss Ratio

CPU has a fast processing speed, but the interaction speed between the CPU and the main memory cannot follow up the processing speed. Hence, the capability of the CPU cannot be fully utilized. Caches reduce the delay in the CPU's interaction with the main memory by storing some data close to the CPU. With a low cache miss rate, the CPU accesses more data from caches at lower latency, which accelerates the whole system. The experiment in this section reveals the impact of cache miss ratio in a simple software process (cache_miss.c). The system for the experiment has L1d cache 672 KiB, L2 cache 28 MiB, and L3 cache 33 MiB. By adjusting the array size in the program, we break the different cache-level limits to increase the cache miss ratio. From Table 4 we can observe that the time increases dramatically when the array size respectively exceeds the L1, L2, and L3 caches. It proves that the cache miss ratio plays an important role in process performance. The program runs faster with a higher cache hit rate.

Array Size	80,000	3,000,000	4,000,000	10,000,000
Time Taken (s)	0.000736	0.013436	0.017346	0.039305

Table 4: Cache Miss

The Impact of TLB Table Miss Ratio

The Translation Lookaside Buffer (TLB) is a special cache used by the CPU to reduce the time taken to access memory locations. It stores recent translations of virtual memory addresses to physical memory addresses. When a program accesses a memory address, the CPU first checks the TLB for the translation. If the translation is in the TLB (a TLB hit), the physical address is quickly retrieved. If not (a TLB miss), the CPU must walk the page table to find the translation, which is a slower process. TLB misses can significantly degrade performance because they introduce additional latency due to page table walks. This is especially noticeable in applications with irregular memory access patterns or large working sets that exceed the capacity of the TLB.

In the program (tlb.c), we control the TLB miss ratio by adjusting the stride in a large array size (64,000,000) through multiple pages. Small stride means accessing memory sequentially (stride of 1), resulting in fewer TLB misses. Large stride means accessing memory with larger strides, causing the program to skip over multiple pages, resulting in higher TLB misses. In Table 5, the time cost increases with the stride increases. The data shows that the TLB table has capability to influence the speed performance. However, the impact of TLB miss decreases when the stride becomes large enough. This is because memory accesses span more pages not currently in the TLB. After the limit of TLB is exceeded, the maximum impact of TLB miss is also reached.

Stride	1	512	1024	4096
Time Taken(s)	0.359726	2.336073	2.407489	2.428064

Table 5: Caption