Improvements of Memory Hierarchy

Xiaoxu Na

Abstract – Several attempts have been implemented to improve the performance of the processor. The experiments are simulated by SimpleScalar 3.0, with the SPEC 2000.
Index Term – SimpleScalar, Performance Improvement

1 Introduction

It was correctly predicted that unlimited of fast memory would be deadly needed. An economical solution to that desire is a memory hierarchy, which takes advantages of locality and trade-offs in the cost performance of memory technologies. [1]

In order to reproduce the behavior of a processor, an architectural simulator is used, thus SimpleScalar is chosen among those popular simulators due to its stronger performance and open source which makes it of high expansibility. SimpleScalar was originally developed by Todd M. Austin, PhD, and the tool suite is currently maintained by SimpleScalar LLC.[2] In order to be fair, SPEC CPU 2000 is chosen to simulate performances. SPEC 2000 is the next generation industry-standardized CPU intensive benchmark suit.[3] Is it designed to provide a comparative measure of compute intesive performance across the widest practical range of hardware, which is exactly why it is necessarily implemented.

2 Backgrounds

2.1 IPC (instructions per clock cycle)

In industry, IPC is commonly used to measure the processor's performance. It is the multiplicative inverse of clock cycles per instruction. Instead of using CPU execution time which is the product of the clock cycle time and the sum of CPU clock cycles and memory stall cycles, we used IPC as a measurement of a processor since the labs will be done to the same amount of instructions, 100 million instructions, and based on the same clock cycle rate. While this only completes one portion of CPU execution time, we used miss rate to measure the most impact element that influences the memory stall cycles in later sections.

2.2 Miss Rate

Since fast memory is expensive, it is very important to have a memory hierarchy which is organized into several levels, with the principle that the nearer to the processor the faster and smaller the cache is. When a word is not found (missed) in the upper level cache, it must be fetched from the lower level in the hierarchy and the corresponding block is placed in the upper level to continue. Miss rate describes misses per reference, however there is another way to indicate miss rate that is misses per instruction. Here, we use misses per reference, which is consistent with what is used in SimpleScalar as well. This is an important issue to reflect how the performance of the computing system is.

3 Experiment Results

3.1 Experimental Setup

In the lad, we stimulated a three level memory hierarchy, with the first level distributed into instruction cache and data cache, and the second level as a unified cache. Using the sim-outorder, we fast run the first 100 million instructions as to warm up the entire processor for each simulation since there are many cold misses at the beginning. The results are counted on the 100 million instructions followed. There are many ways for optimizations and the

following discusses one after another with experiment analysis and comparison.

3.2 SimpleScalar Structure

Before the lab, we did go deeper to dig the structure of SimpleScalar 3.0 itself. Take the LRU (least recently used) replacement policy as an example. Each cache set is built with a data structure of hash table chain. Thus, every hit make the head node pointed to the block just hit. Consequently, the tail block becomes the least recently used one, which is replaced while this set is chosen with a miss. This is quite efficient as for the insertion the time complexity is $\Theta(1)$, and the access time it short enough for frequent accesses without necessary search.

3.3 Larger Block Size

There are some basic ways to reduce the miss rate and we started from the simplest one, larger block size.

Taking advantage of spatial locality, larger block size straightly increases the block size, which reduce compulsory misses, but on the other hand, it increases conflict misses. Each time there is a miss, the processor need to fetch a block in the lower memory, larger block size consequently increases the miss penalty.

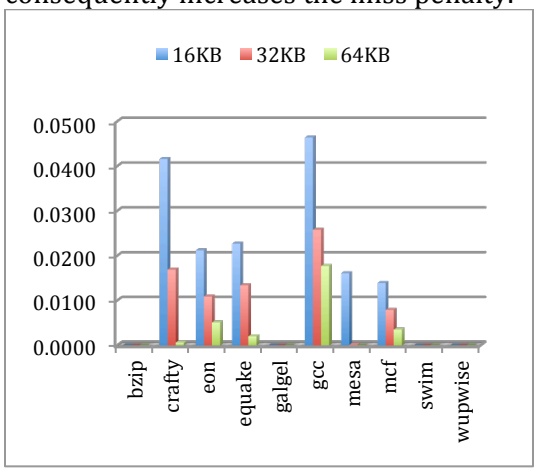

Fig. 1. Level 1 Instruction Cache Miss Rate vs. Block Size

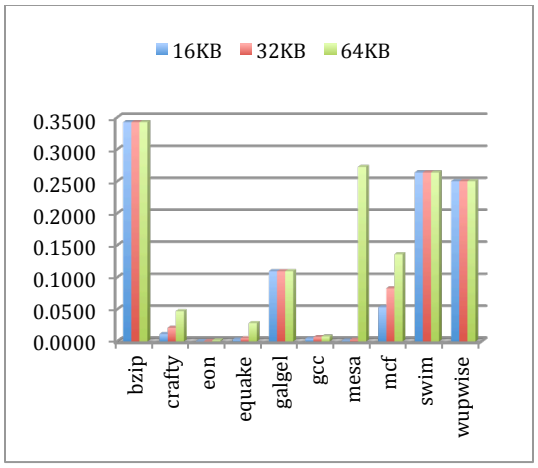

Fig. 2. Level 2 Cache Miss Rate vs. Block Size

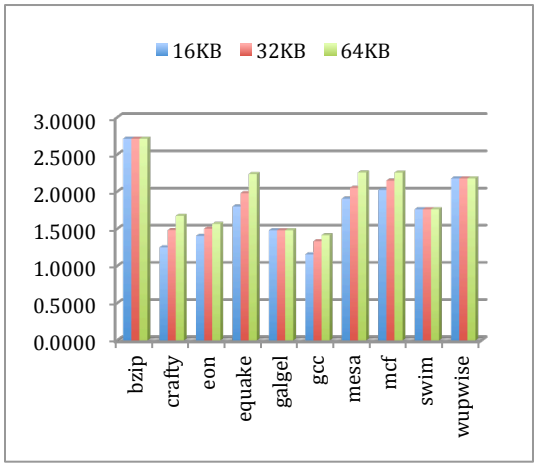

Fig. 3. Instructions per Cycle vs. Block Size

As shown in the Fig. 1 and Fig. 2, larger block size did reduce the miss rate of the level on which it was. However, it increased the lower level miss rate, especially for mesa and mcf. After all, it did excellent work on overall IPC as shown in Fig.3 except bzip, swim and wupwise that made no differences.

3.4 Higher Associativity

Obviously, increasing associativity reduces conflict misses, however it would absolutely come at the cost of more hit time since there are more sets to be tag compared.

As it has been shown in the experiments, 8-way associativity cache has almost the same miss rate as fully associative cache, we did the following labs up to 8-way only.

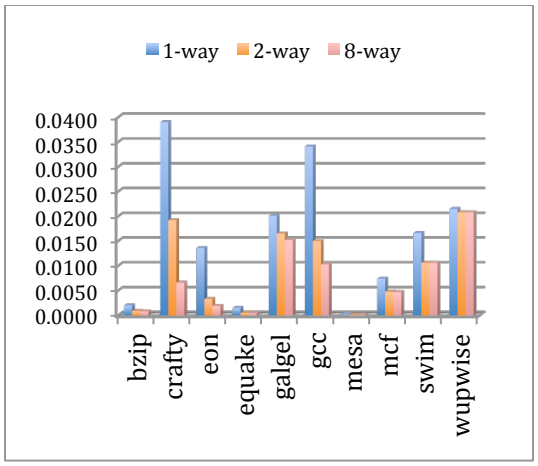

Fig. 4. Level 1 Instruction Cache Miss Rate vs. Associativity

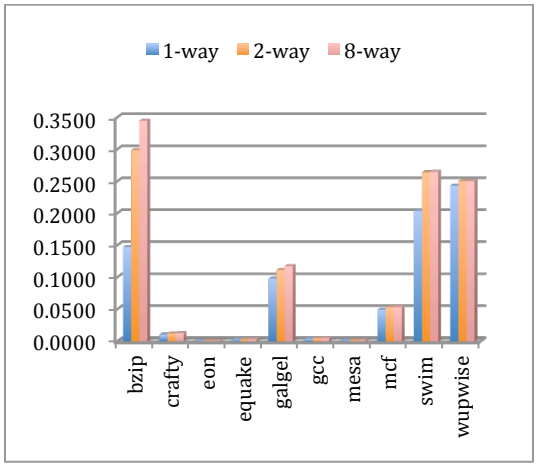

Fig. 5. Level 2 Cache Miss Rate vs. Associativity

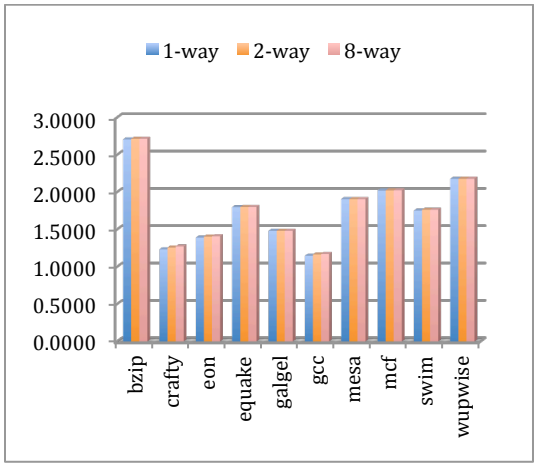

Fig. 6. Instructions per Cycle vs. Associativity

As shown in the Fig. 3 and Fig. 4, higher associativity did quite well on reducing the miss rate of the level on which it was for almost all benchmark we tried. However, it dramatically increased the lower level miss rate, which made not much improvement for IPC as displayed in Fig.6. For SPEC 2000, it is not worthy maintaining higher associativity.

3.5 Enlarging Cache Capacity

The straight way to reduce capacity misses is to increase cache capacity. Although larger cache slightly increases the hit time, it is one of the easiest ways to cut down the overall average memory access time perfectly. At the same time, it lessens both the capacity misses and conflict misses. The most downside it is not the cost efficient.

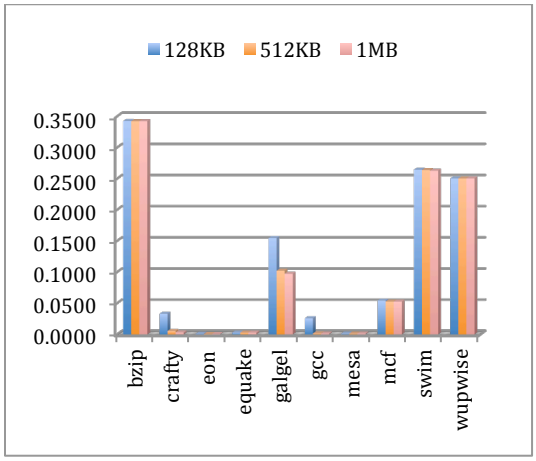

Fig. 7. Level 2 Cache Miss Rate vs. Cache Capacity

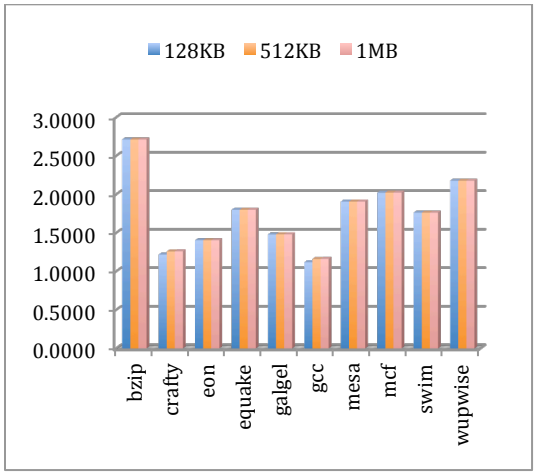

Fig. 8. Instructions per Cycle vs. Cache Capacity

From Fig. 7 and Fig. 8, the only impacts are on crafty and galgel of whom IPCs are slightly improved. Suppose that the benchmark do not occupy much space, which lower the influence of the cache capacity. Thus, it is always a trade-

of, not as easy as the larger the better, since it may be a waste for a smaller program.

3.6 Increasing TLB (translation lookaside buffer)

There is a TLB for level one data and instruction caches respectively. A TLB is a memory cache that is used to reduce the time taken to access a user memory location.[4] Further more, the TLB and cache access might be paralleled if the bits of the page offset is more than the sum of the bits of index and block offset, which helps to reduce the miss penalty.

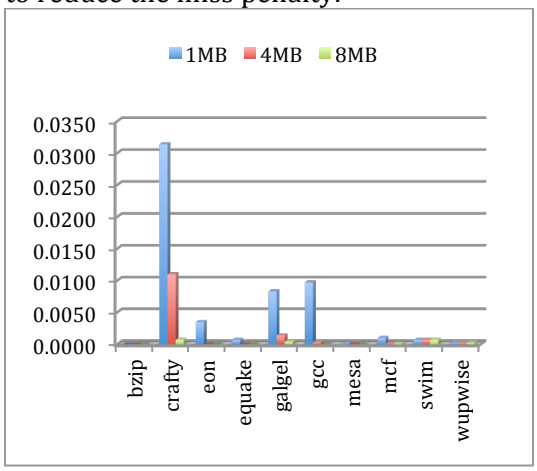

Fig. 9. DTLB Miss Rate vs. DTLB Size

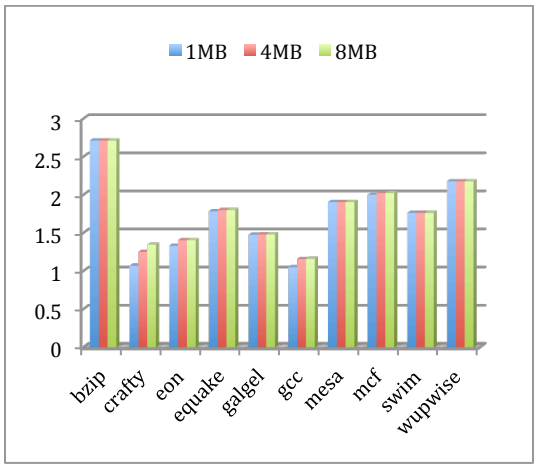

Fig. 10. . Instructions per Cycle vs. DTLB Size

Fig.9 indicated evident improvement on crafty, gagel and gcc. However, bigger TLB only helped a bit on IPC of crafty, eon, and gcc.

3.7 Bigger LSQ (load/store queue)

Many processors take advantage of out-of-order instruction execution to hide their memory access latency. Simplescalar uses a LSQ to store the instructions and traces instructions to make sure the sequential
writes to secure the memory consistency.

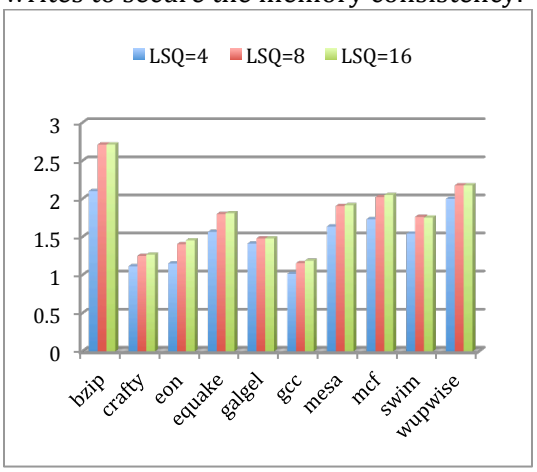

Fig. 11. . Instructions per Cycle vs. LSQ Size

It is markedly demonstrated in Fig. 11 that bigger LSQ helped all benchmark to run faster. However, as the disparity between processor and memory speeds increases, delays in the load-store queue become bottleneck.[5] Many researches have been implemented in this area, adding a load wait buffer (LWB) to temporary remove the long-latency loads, as well as waiting instructions buffer (WIB) proposed by R. Lebeck[6], indicating a remarkable improvement on IPC by 303%.

4 Further Work

There is always a trade-off as we discover the better way to improve the performance of the processor. A combination of several impact factors may be studied and lead to a better solution.

Acknowledgements

The author would like to express the gratitude to Dr. Xin Fu.

References

[1] John L. Hennessy, David A. Patterson. *Computer Architecture A Quantitative Approach, Fifth Edition*, Waltham, MA, 2012, 72.
[2] Todd M. Austin, Ph.D. and SimpleScalar, LLC. "SimpleScalar License Agreement" *http://www.simplescalar.com/*, 1994-2003.
[3] Standard Performance Evaluation Corporation. "SPEC CPU 2000 V1.3" https://www.spec.org/cpu2000/, 2007.
[4] "*A Survey of Techniques for Architecting TLBs*", Concurrency and Computation: Practice and Experience, 2016.
[5] Shelley Chen, Jennifer Morris, "*Out-of-Order Memory Accesses Using a Load Wait Buffer*", http://www.ece.cmu.edu/
[6] R. Lebeck, J. J. Koppanalil, T. Li, J. Patwardhan, and E. Rotenberg. "*A Large, Fast Instruction Window for Tolerating Cache Misses*", 29th International Symposium on Computer Architecture, May 2002.