Implementation of Two-level On-chip Caching and Study of the Effect of Various Parameters

Zezhou Zhang, Pengyu Zhang, and Tongqing Yu. Wonder Team

***Abstract*— In our project, we use two different benchmarks and found the effect of parameters on the performance of caches, which is reflected by the change of miss rate. We demonstrate that miss rate decreases generally when we have larger block size, more cache sets, and higher associativity. These results can be considered correct since it fits in the real world situation. At the same time, our results are reasonable because the properties keep the same after running through two benchmarks.**

I. Introduction

In this project, we use two functional cache simulators to get access to the configurations of each level of cache. We examine the performance of caches by changing the parameters under different benchmarks. Our project mainly involves with the number of sets, block size and associativity. In the first part, we use test-math as benchmark. Under each experience, we change one of the three parameters with two other parameters are fixed. Then, we can observe the effect of the parameter on miss rate, which is significant for caches. In the second part, we use anagram as benchmark. We find that caches own same properties under different situation..

II. Implementation

Our project should run on SimpleScalar under the Linux environment. The two simulators we used are **sim-cache** and **sim-cheetah** to check the performance of two-level cache based on different configurations. The results are coming from two different benchmarks in order to double-check the results that are correct for analysis.

A. Motivations

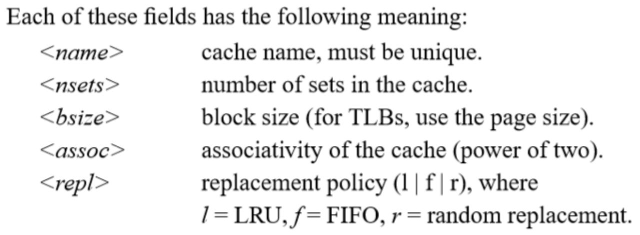
The purpose of caching is to store program instructions and data that are reused during program operations or to store information that the CPU may need for next fetching. In this way, the computer processor can quickly access information from the cache, rather than from the computer's main memory. Having faster speed to get access to these instructions increases the overall speed of a program. Various parameters can affect the performance of a cache. Our project intends to study the influence of different parameters on miss rate of the cache. The processor can have a better computation speed when it finds a memory location in cache, which means more cache hits can bring more benefits. Finding appropriate parameters for cache design optimization is nowadays significant to computer industry.

B. Simulators

The two simulators we used for this project are **sim-cache** and **sim-cheetah**. They both use the cache.c file for fast simulation of caches. The **sim-cache** can put emphasis on different parameters each time to simulate.

<name>:<nsets>:<bsize>:<assoc>:<repl>

Thus, the cache size can be calculated as the product of <nets>, <bsize>, and <assoc>. The advantage of **sim-cache** is that it can configure each level with different configurations. For example, l1 cache and l2 cache can be set as different cache sizes at one time.

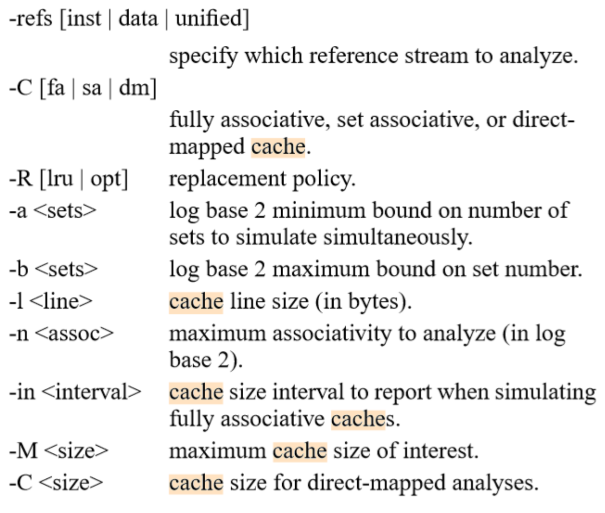


Each of these fields has the following meaning:

<name>	cache name, must be unique.
<nsets>	number of sets in the cache.
<bsize>	block size (for TLBs, use the page size).
<assoc>	associativity of the cache (power of two).
<repl>	replacement policy (l | f | r), where *l* = LRU, *f* = FIFO, *r* = random replacement.

Fig.1 The sim-cache Cache Configurations [1]

The **sim-cheetah** can run the simulation very efficiently **by** using high-speed Cheetah engine. It can also accept the different cache configurations for command-line arguments as the sim-cache, such as replacement policy, cache line size, and number of sets. The advantage of sim-cheetah is that it can generate ranges of simulation results for multiple cache configurations within a single simulation. [1]



-refs [inst | data | unified]	
	specify which reference stream to analyze.
-C [fa | sa | dm]	
	fully associative, set associative, or direct-mapped cache.
-R [lru | opt]	replacement policy.
-a <sets>	log base 2 minimum bound on number of sets to simulate simultaneously.
-b <sets>	log base 2 maximum bound on set number.
-l <line>	cache line size (in bytes).
-n <assoc>	maximum associativity to analyze (in log base 2).
-in <interval>	cache size interval to report when simulating fully associative caches.
-M <size>	maximum cache size of interest.
-C <size>	cache size for direct-mapped analyses.

Fig.2 Command-line Arguments in sim-cheetah [1]

Note that: 'line size' is the same as block size. Most of parameters are given as log base 2 of the number.

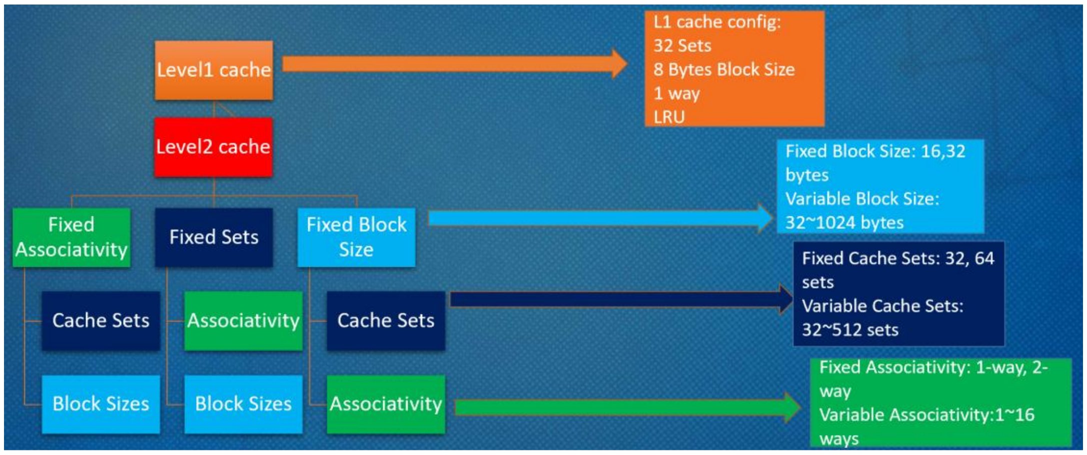

Level1 cache
L1 cache config:
32 Sets
8 Bytes Block Size
1 way
LRU
Level2 cache
Fixed Block Size: 16,32 bytes
Variable Block Size: 32~1024 bytes
Fixed Associativity
Fixed Sets
Fixed Block Size
Cache Sets
Associativity
Cache Sets
Fixed Cache Sets: 32, 64 sets
Variable Cache Sets: 32~512 sets
Block Sizes
Block Sizes
Associativity
Fixed Associativity: 1-way, 2-way
Variable Associativity:1~16 ways

Fig. 3 The Flow Chart of Experiments with Method of Control Variates

Both simulators are well suited for performing high-level cache studies that do not take cache access time into account. Because we put emphasis on the miss rates rather than the actual program execution time, these two simulators are ideal for this work.

C. *Benchmarks*

Benchmarking is an essential part of the design process because different architecture configurations can affect the results. The benchmarks we used in this project are **test-math** and **anagram**, so we can evaluate the outputs twice to see if the results have the same trend, such as a down-slope line of miss rates with respect to cache size.

The **test-math** performs various math computations in both integer and floating-point instructions, and it can generate a list of arithmetic operations as output. Also, it does not need input to display the result.

The **anagram** is also a test benchmark comes with SimpleScalar. It takes a dictionary file and a set of strings as inputs and creates anagrams of those strings based on the words in the dictionary [2]. In this benchmark, it can make the results less predictable; however, it can prove the performance based on previous benchmark if it can result in a similar trend.

The two benchmarks are used for validation of the actual outputs in order to make sure the data used to analysis are correct.

III. EXPERIMENT

We designed multiple experiments by implementing Control Variates and plotted curves based on the result of our experiments. All of our experiments are based on two-level on-chip cache as well as two different benchmarks, so we choose to fix the parameters of L1 cache. Then, we adjust parameters of L2 cache to obtain the effect of the number of ways, block size, and number of sets of L2 cache on overall miss rate. The flow chart of this experiment can be seen on Fig. 3. To study how L1 cache affects on miss rate, we also designed another experiment by only choosing another bunch of parameters of L1 cache.

A. *Parameters of Level 1 cache*

We choose 32 as the number of sets, 8 as the number of block sizes and 1-way associativity as parameters of L1 cache in our first configuration of L1 cache. The second configuration of L1 cache is implemented by changing the number of block sizes to 16.

B. *Experiment based on Sim-cache on the condition of the first configuration of L1 and Test-math as benchmark*

For each configuration of L1 cache, we still make use of the Control Variate method, which specifically means only changing one parameter of the three while keeping the other two parameters fixed.

In the first experiment, we fixed the number of sets and number of block sizes, and then varyed number of ways associativity from 1 to 2 at the step of 1. We designed 5 different numbers of sets which varies from 32 to 512 and 6 different numbers of the block size. After we got 5*6 which is 30 results of miss rate, we plotted the curve of “Miss rate” to “Number of sets of L2 cache”. The plot can be seen in Figure 4.

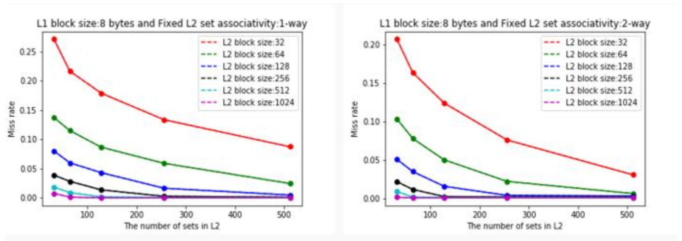


Fig. 4 L2:sets:block size:1-way:LRU VS. L2:sets:block size:2-way:LRU.

In the second experiment, we keep block size and number of sets fixed then varying number of sets from 32 to 64. The range of block size and associativities remain the same as we designed in the first experiment. The plot can be seen in Figure 5.

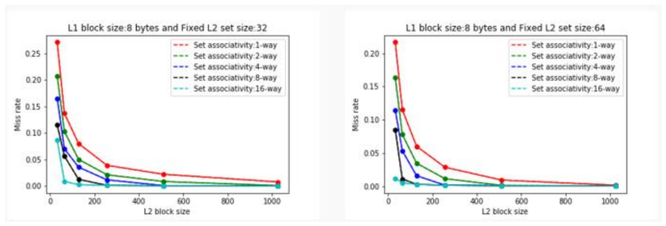


Fig. 5. L2:32 sets:block size:way:LRU VS. L2:64 sets:block size:way:LRU

C. *Experiment based on Sim-cache on the condition of the first configuration of L1 and Anagram as benchmark*

We take advantage of Anagram as new benchmark to implement our experiments under the circumstance of the same configuration in B. The plot can be seen in Figure 6.

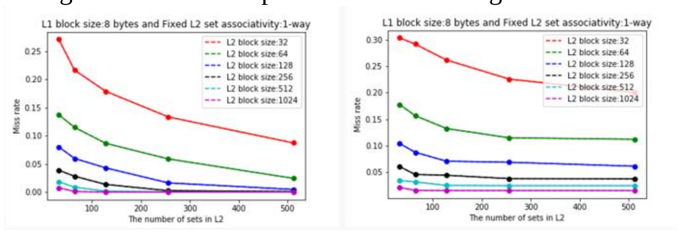


Fig. 6 L2: sets:block size:1-way:LRU(test_math) VS. L2:64 sets:block size:1-way:LRU(anagram)

D. *Experiment based on Sim-cheetah on the condition of the second configuration of L1 and Anagram as benchmark*

In this experiment, what we have completed is keeping the number of sets and number of block size fixed, then varying the number of associativities. The plot can be seen in Figure 7.

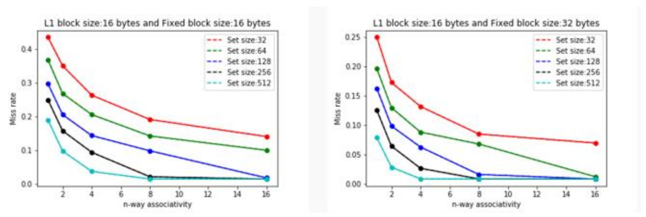


Fig. 7 L2:sets:16 block size:way:LRU VS. L2:sets:32 block size:way:LRU

E. *Experiment based on Sim-cheetah on the condition of the second configuration of L1 and Anagram as benchmark*

As the comparison to D, we keep the basic configuration the same as that in D. Then only doing our test by implementing Anagram as benchmark The plot can be seen in Figure 8.

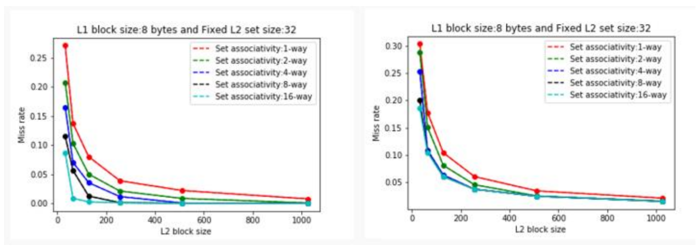


Fig. 8 L2:32 sets:block size:way:LRU (test_math)VS. L2:32 sets:block size:way:LRU(anagram)

IV. CONCLUSION

A. *The most commonly used cache size of L1 cache and L2 cache in Personal Computer.*

In the Personal Computer (PC) field, the most commonly used range of size of L1 cache is from 8KB to 64KB. L1 cache is made of Static RAM (SRAM) whose basic component is transistor. Most of SRAM uses 6 or 8 transistors per bit, therefore an L1 cache with large cache size need a huge number of transistors to build which will directly cause using out of circuit space. From the perspective of power-consuming, a large cache size will also cause high energy-consuming.

Besides, processors designers need to take hit rate as well as latency into account. Because the L1 cache is an on-board cache with zero wait delay, engineers need to carefully choose its size. With large cache size, the cache latency will increase because there is going to take lots of die area due to the incomplete usage of the cache.

There is a smaller number of choices compared with that of L1 cache size when designing L2 cache for PCs CPU. L2 cache typically varies from 256KB to 512KB. The goal of L2 cache is to provide more storage space to the processors and decreasing wait delay and it's also an on-board cache. As we have already mentioned above, the larger the cache size is, the more space it will need on the circuit board. And the larger space the cache takes, the slower the cache will be. Therefore, this is also a trade-off when implementing L2 cache size because we need to choose more than twice the original cache size if we just expect to get double performance in our new L2 cache.

B. *Effect of various parameters on cache miss rate.*

By using different benchmarks, we get the same trend of how parameters of cache affect miss rate.

Larger block size can reduce compulsory misses because it can take advantage of spatial locality. Smaller blocks do not take maximum advantage of spatial locality But if blocks are too large, there are fewer blocks available, and more potential conflicts misses.

Highly-associative cache can have a lower miss rate. Each set has more blocks, so there's less chance of a conflict between two addresses. Overall, this will reduce AMAT and memory stall cycles.

Larger cache sets can lead to cache size. Cache size also has a significant impact on performance. In a larger cache there's less chance that there will be of a conflict. Again this means the miss rate decreases, so the AMAT and number of memory stall cycles also decrease.

REFERENCES

[1] T. M. Austin *et al*., "The SimpleScalar Tool Set," *SimpleScalar.* SimpleScalar LLC 2395 Timbercrest Court, [Online]. Available: http://www.simplescalar.com

[2] D. Burger *et al*., "SimpleScalar Tutorial," *SimpleScalar*, Computer Sciences Department University of Wisconsin-Madison, [Online]. Available: http://www.simplescalar.com