Lab1
Implementation of a Cache Simulator

Yueqi Chen(121160005)
Nanjing University
Department of Computer Science and Technology
Yueqichen.0x0@gmail.com

Abstract

Typically called cache in computer architecture is used to bridge the gap between the processor and main memory. Parameters, such as cache size, cacheline size, associativity and replacement policy, could be set to make up different cache hierarchies.
In this experiment, I implemented a cache simulator in C language to compare different cache hierarchies' performance by running SPEC2000 traces on them respectively. Experiment's results indict that set-associativity cache trades off between hit rate and power/cost so as to achieve great performance. Based on the visual CPI comparison, the trend that multi-level cache hierarchy could ameliorate performance is definite. What's more, this experiment shows an affinity between Load instruction's hit rate and Store instruction's hit rate. According to the experiment, Victim cache's effect is not clear and in need of further investigation.

1 Introduction

Over recent years, processor speeds have increased at a faster rate than DRAM speeds. This trend is shown in Figure 1.

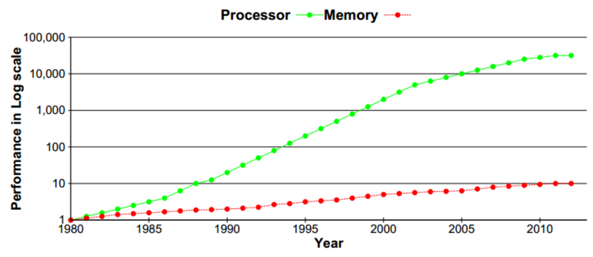


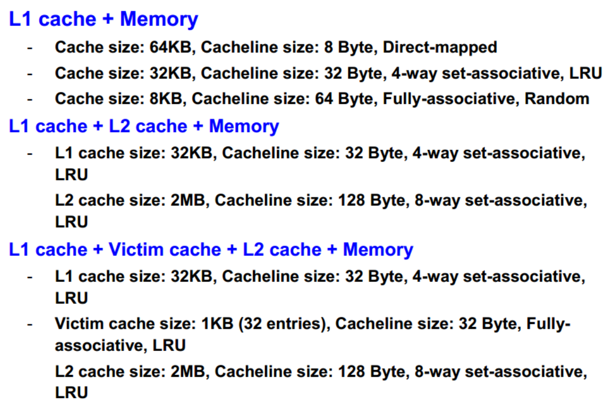
Figure 1: Gap in performance between the processor and memory

Current generation processors have main memory access latency of more than 300 processor cycles and projections show that this will increase in near future. The memory acts as a barrier/wall to achieve more performance. This is often referred as the Memory Wall.

Table 1: Access time assumption

Cache	Time(cycle)
L1 cache	1
Victim cache	1
L2 cache	10
Memory	100

To bridge the gap between the processor and main memory, caches are used to keep frequently-used data needed by the cores. Performance improvement is achieved by serving the request directly from caches, which are faster than DRAM.
Due to varying cache requirement of applications, designers use a hierarchy of caches: the caches closer to the processor are smaller and faster while caches further away are slower.
Parameters, such as cache size, cacheline size, associativity and replacement policy of caches from different level could finally greatly influence the performance, power consumed, reliability and many others facets of the cache hierarchy and even the overall processor.
In this experiment, I vary the aforementioned parameters in different cache hierarchies under the same traces (SPEC2000). Some statistical data, for example hit rate of each level cache, is thus collected or calculated to reflect the parameters' influences on the performance which is measured mainly by CPI(cycle per instruction).
Figure 2 shows the specific configurations of each cache hierarchies. Table 1 indicts the assumptions of access time in different cache level.
For the rest of this report, I describe the detailed design of cache simulator in Section 2. Section 3 gives out the whole results and discusses the primary facets of the



L1 cache + Memory

- Cache size: 64KB, Cacheline size: 8 Byte, Direct-mapped
- Cache size: 32KB, Cacheline size: 32 Byte, 4-way set-associative, LRU
- Cache size: 8KB, Cacheline size: 64 Byte, Fully-associative, Random

L1 cache + L2 cache + Memory

- L1 cache size: 32KB, Cacheline size: 32 Byte, 4-way set-associative, LRU
  L2 cache size: 2MB, Cacheline size: 128 Byte, 8-way set-associative, LRU

L1 cache + Victim cache + L2 cache + Memory

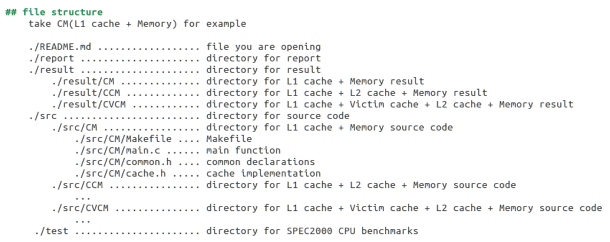
- L1 cache size: 32KB, Cacheline size: 32 Byte, 4-way set-associative, LRU
- Victim cache size: 1KB (32 entries), Cacheline size: 32 Byte, Fully-associative, LRU
  L2 cache size: 2MB, Cacheline size: 128 Byte, 8-way set-associative, LRU

Figure 2: Configurations

results. Section 4 concludes this report.

2 Detailed Design

In this section, I will describe the experiment setup, file structure, data collected during cache's running and access procedure of L1 Cache + Victim Cache + L2 Cache + Memory architecture.
I use GCC (4.8.4) and GNU Make (3.81) to compiler my source files. The executable file could be run in any Linux machine. The trace is 'SPEC2000 CPU' which includes gcc, gzip, mcf, swim, twolf traces.



```
## file structure
    take CM(L1 cache + Memory) for example

    ./README.md .................... file you are opening
    ./report ....................... directory for report
    ./result ....................... directory for result
        ./result/CM ................ directory for L1 cache + Memory result
        ./result/CCM ............... directory for L1 cache + L2 cache + Memory result
        ./result/CVCM .............. directory for L1 cache + Victim cache + L2 cache + Memory result
    ./src .......................... directory for source code
        ./src/CM ................... directory for L1 cache + Memory source code
            ./src/CM/Makefile .... Makefile
            ./src/CM/main.c ...... main function
            ./src/CM/common.h .... common declarations
            ./src/CM/cache.h ..... cache implementation
        ./src/CCM .................. directory for L1 cache + L2 cache + Memory source code
            ...
        ./src/CVCM ................. directory for L1 cache + Victim cache + L2 cache + Memory source code
            ...
     ./test ........................ directory for SPEC2000 CPU benchmarks
```

Figure 3: File Structure

Figure 3 displays the fire structure: directory 'src' is for source code which is further specified as three sub directories, directory 'test' is for trace files, directory 'result' has a similar structure as directory 'src' and it is for corresponding statistics data. More file structure information could be found in README.md.
During the cache's running, I will collect data including number of cycles, number of cache access(load and store instruction respectively), number of memory access(load and store instruction respectively), number of instructions, number of cycles used for execution instructions and so on.
Table 2 lists the symbols I used in access procedure as well as calculation formulas. Equations 1-4 are the formulas used to attain final results. Note that if 'X' is 1, then CaX_t is actually $Ca1_t$, which means the number of access to L1 cache. Since access procedure of Cache + Memory or L1 Cache + L2 Cache + Memory is similar to but simpler than L1 Cache + Victim Cache + L2 Cache + Memory, so I will only elaborate on the scenario of L1 Cache + Victim Cache + L2 Cache + Memory.

Figure 4 visualizes the access procedure: accessing L1 cache to Victim cache to L2 cache to memory if it keeps missing. If hitting at any level, it will fresh its upper level caches (except victim cache) and then read in the next trace record. In the scenario of victim cache, I will only change the access frequency number of the exact block, no fresh operation will be done in L1 cache. The procedure of counting cycles is not reflected in Figure 2 to make the overall procedure clear, since this could be done at the same time when accessing any level cache.

Table 2: Collected Data

Symbol	Meaning
Cy_t	number of cycles for specific trace
Cy_i	number of cycles used in executing instructions
CaX_t	number of access to L1/L2/Victim cache
CaX_l	number of access to L1/L2/Victim cache (load)
CaX_s	number of access to L1/L2/Victim cache (store)
$CaXH_t$	number of hit to L1/L2/Victim cache
$CaXH_s$	number of hit to L1/L2/Victim cache (load)
$CaXH_t$	number of hit to L1/L2/Victim cache (store)
Me_t	number of access to memory
Me_l	number of access to memory (load)
Me_s	number of access to memory (store)
N_i	number of instructions

$$Hit\ rate\ of\ CacheX = \frac{CaXH_t}{CaX_t} \tag{1}$$

$$Hit\ rate\ of\ CacheX\ for\ Load = \frac{CaXH_l}{CaX_l} \tag{2}$$

$$Hit\ rate\ of\ CacheX\ for\ Store = \frac{CaXH_s}{CaX_s} \tag{3}$$

$$CPI = \frac{N_i}{Cy_i} \tag{4}$$

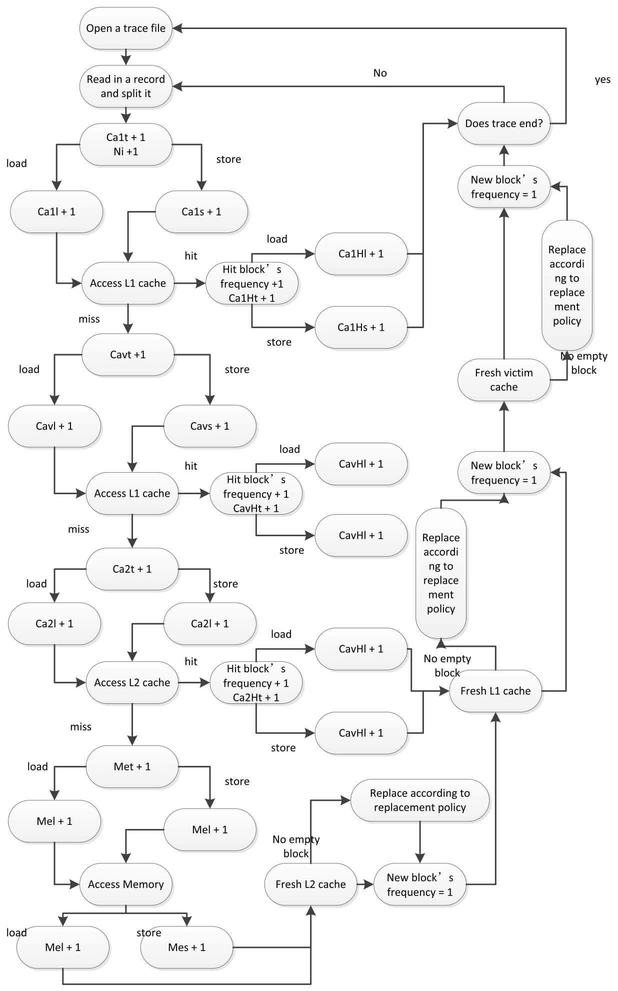


Figure 4: Access Procedure

Table 3: L1 cache + Memory swim.trace

cache size	cacheline size	associativity	output	result
64KB	8 bytes	direct mapped	number of cache access	303193
			number of cache access for load	220668
			number of cache access for store	82525
			number of memory access	20226
			number of memory access for load	1611
			number of memory access for store	18615
			cache hit rate	0.933290
			cache hit for load	0.992699
			cache hit for store	0.774432
			CPU time	3198802
			CPU time for load and store	2325793
			CPI	7.670999
32KB	32 bytes	4-way-set associative	number of cache access	303193
			number of cache access for load	220668
			number of cache access for store	82525
			number of memory access	6772
			number of memory access for load	886
			number of memory access for store	5886
			cache hit rate	0.977664
			cache hit for load	0.995985
			cache hit for store	0.928676
			CPU time	1853402
			CPU time for load and store	980393
			CPI	3.233561
8KB	64 bytes	fully associative	number of cache access	303193
			number of cache access for load	220668
			number of cache access for store	82525
			number of memory access	5905
			number of memory access for load	2336
			number of memory access for store	3569
			cache hit rate	0.980524
			cache hit for load	0.989414
			cache hit for store	0.956752
			CPU time	1766702
			CPU time for load and store	893693
			CPI	2.947604

Table 4: L1 cache + Memory mcf.trace

cache size	cacheline size	associativity	output	result
64KB	8 bytes	direct mapped	number of cache access	727230
			number of cache access for load	5972
			number of cache access for store	721258
			number of memory access	719494
			number of memory access for load	348
			number of memory access for store	719146
			cache hit rate	0.010638
			cache hit for load	0.941728
			cache hit for store	0.002928
			CPU time	72963108
			CPU time for load and store	72676630
			CPI	99.936241
32KB	32 bytes	4-way-set associative	number of cache access	727230
			number of cache access for load	5972
			number of cache access for store	721258
			number of memory access	179987
			number of memory access for load	191
			number of memory access for store	179796
			cache hit rate	0.752503
			cache hit for load	0.968017
			cache hit for store	0.750719
			CPU time	19012408
			CPU time for load and store	18725930
			CPI	25.749666
8KB	64 bytes	fully associative	number of cache access	727230
			number of cache access for load	5972
			number of cache access for store	721258
			number of memory access	90211
			number of memory access for load	246
			number of memory access for store	89965
			cache hit rate	0.875953
			cache hit for load	0.958808
			cache hit for store	0.875267
			CPU time	10034808
			CPU time for load and store	9748330
			CPI	13.404741

3 Results & Discussion

Table 5: L1 cache + Memory gzip.trace

cache size	cacheline size	associativity	output	result
64KB	8 bytes	direct mapped	number of cache access	481044
			number of cache access for load	320441
			number of cache access for store	160603
			number of memory access	160153
			number of memory access for load	159477
			number of memory access for store	676
			cache hit rate	0.667072
			cache hit for load	0.502320
			cache hit for store	0.995791
			CPU time	17098654
			CPU time for load and store	16496344
			CPI	34.292797
32KB	32 bytes	4-way-set associative	number of cache access	481044
			number of cache access for load	320441
			number of cache access for store	160603
			number of memory access	159577
			number of memory access for load	159410
			number of memory access for store	167
			cache hit rate	0.668269
			cache hit for load	0.502529
			cache hit for store	0.998960
			CPU time	17041054
			CPU time for load and store	16438744
			CPI	34.173058
8KB	64 bytes	fully associative	number of cache access	481044
			number of cache access for load	320441
			number of cache access for store	160603
			number of memory access	160855
			number of memory access for load	160730
			number of memory access for store	125
			cache hit rate	0.665613
			cache hit for load	0.498410
			cache hit for store	0.999222
			CPU time	17168854
			CPU time for load and store	16566544
			CPI	34.438728

Table 6: L1 cache + Memory twolf.trace

cache size	cacheline size	associativity	output	result
64KB	8 bytes	direct mapped	number of cache access	482824
			number of cache access for load	351403
			number of cache access for store	131421
			number of memory access	5578
			number of memory access for load	1168
			number of memory access for store	4410
			cache hit rate	0.988447
			cache hit for load	0.996676
			cache hit for store	0.966444
			CPU time	2008827
			CPU time for load and store	1040624
			CPI	2.155286
32KB	32 bytes	4-way-set associative	number of cache access	482824
			number of cache access for load	351403
			number of cache access for store	1205
			number of memory access	2263
			number of memory access for load	1058
			number of memory access for store	1205
			cache hit rate	0.995313
			cache hit for load	0.996989
			cache hit for store	0.990831
			CPU time	1677327
			CPU time for load and store	709124
			CPI	1.468701
8KB	64 bytes	fully associative	number of cache access	482824
			number of cache access for load	351403
			number of cache access for store	131421
			number of memory access	1978
			number of memory access for load	1185
			number of memory access for store	793
			cache hit rate	0.995903
			cache hit for load	0.996628
			cache hit for store	0.993966
			CPU time	1648827
			CPU time for load and store	680424
			CPI	1.409673

Table 7: L1 cache + Memory gcc.trace

cache size	cacheline size	associativity	output	result
64KB	8 bytes	direct mapped	number of cache access	515683
			number of cache access for load	318197
			number of cache access for store	197486
			number of memory access	22375
			number of memory access for load	4830
			number of memory access for store	17545
			cache hit rate	0.956611
			cache hit for load	0.984821
			cache hit for store	0.911158
			CPU time	377764
			CPU time for load and store	2753183
			CPI	5.338906
32KB	32 bytes	4-way-set associative	number of cache access	515683
			number of cache access for load	318197
			number of cache access for store	197486
			number of memory access	9444
			number of memory access for load	4194
			number of memory access for store	5250
			cache hit rate	0.981686
			cache hit for load	0.986820
			cache hit for store	0.973416
			CPU time	2484564
			CPU time for load and store	1460083
			CPI	2.831358
8KB	64 bytes	fully associative	number of cache access	515683
			number of cache access for load	318197
			number of cache access for store	197486
			number of memory access	7363
			number of memory access for load	4221
			number of memory access for store	3142
			cache hit rate	0.985722
			cache hit for load	0.986735
			cache hit for store	0.984090
			CPU time	2276464
			CPU time for load and store	1251983
			CPI	2.427815

Table 8: L1 cache + L2 cache + Memory

trace	output	result
swim.trace	number of L1 cache access	303193
	number of L2 cache access	6772
	number of Memory access	1826
	number of L1 cache access for load	220668
	number of L2 cache access for load	886
	number of Memory access for load	272
	number of L1 cache access for store	82525
	number of L2 cache access for store	5886
	number of Memory access for store	1554
	L1 cache hit rate	0.977664
	L2 cache hit rate	0.730360
	L1 cache hit rate for load	0.995985
	L2 cache hit rate for load	0.693002
	L1 cache hit rate for store	0.928676
	L2 cache hit rate for store	0.735984
	CPU time	1426522
	CPU time for load and store	553513
	CPI	1.825613
mcf.trace	number of L1 cache access	727230
	number of L2 cache access	179987
	number of Memory access	45025
	number of L1 cache access for load	5972
	number of L2 cache access for load	191
	number of Memory access for load	80
	number of L1 cache access for store	721258
	number of L2 cache access for store	179796
	number of Memory access for store	44945
	L1 cache hit rate	0.752503
	L2 cache hit rate	0.749843
	L1 cache hit rate for load	0.968017
	L2 cache hit rate for load	0.581152
	L1 cache hit rate for store	0.750719
	L2 cache hit rate for store	0.750022
	CPU time	7316078
	CPU time for load and store	7029600
	CPI	9.666268

Table 9: L1 cache + L2 cache + Memory

trace	output	result
gzip.trace	number of L1 cache access	481044
	number of L2 cache access	159577
	number of Memory access	157854
	number of L1 cache access for load	320441
	number of L2 cache access for load	159410
	number of Memory access for load	157836
	number of L1 cache access for store	160603
	number of L2 cache access for store	167
	number of Memory access for store	18
	L1 cache hit rate	0.668269
	L2 cache hit rate	0.010797
	L1 cache hit rate for load	0.502529
	L2 cache hit rate for load	0.009874
	L1 cache hit rate for store	0.998960
	L2 cache hit rate for store	0.892216
	CPU time	18464524
	CPU time for load and store	17862214
	CPI	37.132183
twolf.trace	number of L1 cache access	482824
	number of L2 cache access	2263
	number of Memory access	424
	number of L1 cache access for load	351403
	number of L2 cache access for load	1058
	number of Memory access for load	186
	number of L1 cache access for store	13421
	number of L2 cache access for store	1205
	number of Memory access for store	238
	L1 cache hit rate	0.995313
	L2 cache hit rate	0.812638
	L1 cache hit rate for load	0.996989
	L2 cache hit rate for load	0.824197
	L1 cache hit rate for store	0.990831
	L2 cache hit rate for store	0.802490
	CPU time	1516057
	CPU time for load and store	547854
	CPI	1.134687
gcc.trace	number of L1 cache access	515683
	number of L2 cache access	9444
	number of Memory access	443
	number of L1 cache access for load	318197
	number of L2 cache access for load	4194
	number of Memory access for load	323
	number of L1 cache access for store	197486
	number of L2 cache access for store	5250
	number of Memory access for store	120
	L1 cache hit rate	0.981686
	L2 cache hit rate	0.953092
	L1 cache hit rate for load	0.986820
	L2 cache hit rate for load	0.922985
	L1 cache hit rate for store	0.973416
	L2 cache hit rate for store	0.977143
	CPU time	1678904
	CPU time for load and store	654423
	CPI	1.269041

Table 10: L1 cache + Victim cache + L2 cache + Memory

trace	output	result
swim.trace	number of L1 cache access	303193
	number of Victim cache access	6772
	number of L2 cache access	6771
	number of Memory access	1826
	number of L1 cache access for load	220668
	number of Victim access for load	886
	number of L2 cache access for load	886
	number of Memory access for load	272
	number of L1 cache access for store	82525
	number of Victim access for store	5886
	number of L2 cache access for store	5885
	number of Memory access for store	1554
	L1 cache hit rate	0.977664
	Victim cache hit rate	0.000148
	L2 cache hit rate	0.730321
	L1 cache hit rate for load	0.995985
	Victim cache hit rate for load	0.000000
	L2 cache hit rate for load	0.693002
	L1 cache hit rate for store	0.928676
	Victim cache hit rate for store	0.000170
	L2 cache hit rate for store	0.735939
	CPU time	1433284
	CPU time for load and store	560275
	CPI	1.847915
mcf.trace	number of L1 cache access	727230
	number of Victim cache access	179987
	number of L2 cache access	179987
	number of Memory access	45025
	number of L1 cache access for load	5972
	number of Victim access for load	191
	number of L2 cache access for load	191
	number of Memory access for load	80
	number of L1 cache access for store	721258
	number of Victim access for store	179796
	number of L2 cache access for store	179796
	number of Memory access for store	44945
	L1 cache hit rate	0.752703
	Victim cache hit rate	0.000000
	L2 cache hit rate	0.749843
	L1 cache hit rate for load	0.968017
	Victim cache hit rate for load	0.000000
	L2 cache hit rate for load	0.581152
	L1 cache hit rate for store	0.750719
	Victim cache hit rate for store	0.000000
	L2 cache hit rate for store	0.750022
	CPU time	7496065
	CPU time for load and store	7209587
	CPI	9.913765

Table 11: L1 cache + Victim cache + L2 cache + Memory

trace	output	result
gzip.trace	number of L1 cache access	481044
	number of Victim cache access	159604
	number of L2 cache access	159576
	number of Memory access	157853
	number of L1 cache access for load	320441
	number of Victim access for load	159423
	number of L2 cache access for load	159410
	number of Memory access for load	157836
	number of L1 cache access for store	160603
	number of Victim access for store	181
	number of L2 cache access for store	166
	number of Memory access for store	17
	L1 cache hit rate	0.668213
	Victim cache hit rate	0.000175
	L2 cache hit rate	0.010797
	L1 cache hit rate for load	0.502489
	Victim cache hit rate for load	0.000082
	L2 cache hit rate for load	0.009874
	L1 cache hit rate for store	0.998873
	Victim cache hit rate for store	0.082873
	L2 cache hit rate for store	0.897590
	CPU time	18624018
	CPU time for load and store	18021708
	CPI	37.463741
twolf.trace	number of L1 cache access	482824
	number of Victim cache access	3198
	number of L2 cache access	2259
	number of Memory access	423
	number of L1 cache access for load	351403
	number of Victim access for load	1446
	number of L2 cache access for load	1055
	number of Memory access for load	186
	number of L1 cache access for store	131421
	number of Victim access for store	1752
	number of L2 cache access for store	1204
	number of Memory access for store	237
	L1 cache hit rate	0.993376
	Victim cache hit rate	0.293621
	L2 cache hit rate	0.812749
	L1 cache hit rate for load	0.995885
	Victim cache hit rate for load	0.270401
	L2 cache hit rate for load	0.823697
	L1 cache hit rate for store	0.986669
	Victim cache hit rate for store	0.312785
	L2 cache hit rate for store	0.803156
	CPU time	1519115
	CPU time for load and store	550912
	CPI	1.141020
gcc.trace	number of L1 cache access	515683
	number of Victim cache access	12012
	number of L2 cache access	9442
	number of Memory access	441
	number of L1 cache access for load	318197
	number of Victim access for load	5691
	number of L2 cache access for load	4194
	number of Memory access for load	323
	number of L1 cache access for store	197486
	number of Victim access for store	6321
	number of L2 cache access for store	5248
	number of Memory access for store	118
	L1 cache hit rate	0.976707
	Victim cache hit rate	0.213953
	L2 cache hit rate	0.953294
	L1 cache hit rate for load	0.982115
	Victim cache hit rate for load	0.263047
	L2 cache hit rate for load	0.922985
	L1 cache hit rate for store	0.967993
	Victim cache hit rate for store	0.169752
	L2 cache hit rate for store	0.977515
	CPU time	1690696
	CPU time for load and store	666215
	CPI	1.291908

Any required results have been list in Table 5-11. Since I use CPI to measure the performance of cache, I will extract the CPIs of each cache hierarchy of each trace file in Figure 5. To make it clear, I number the cache hierarchies :

1. 64KB cache + Memory
2. 32KB cache + Memory
3. 8KB cache + Memory
4. L1 cache + L2 cache + Memory
5. L1 cache + Victim cache + L2 cache + Memory

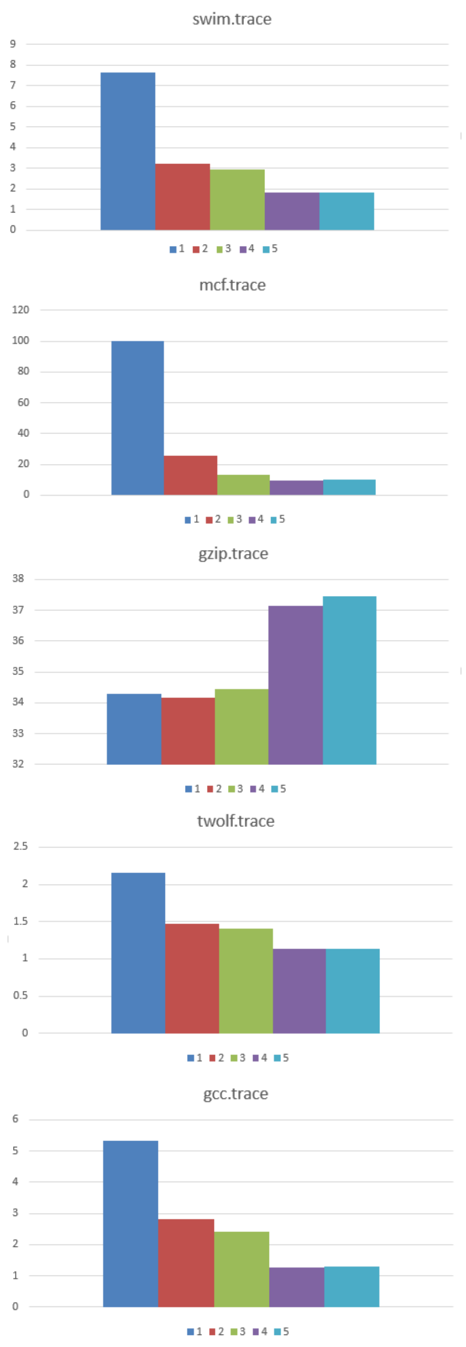


Figure 5: CPIs of different cache hierarchies of different traces

The general trend in Figure 5 is that as I add more levels of cache into the hierarchy, the performance is becoming better. However, when it comes to gzip.trace, cache hierarchy No.4 and No.5 have higher CPIs than No.1-3. The reason is that the hit rate of L2 cache is quite low according to the detailed data in Table 9 and Table 11 — accessing L2 cache with seldom hit but 10 extra cycles could inevitably cost more cycles than the scenario without L2 cache. However, the relative gap between different cache hierarchies is smaller enough to be ignored compared to other traces. What's more, gzip.trace's hit rate of L1 cache is relatively much lower than any other traces running in the same cache hierarchy. Therefore, I extrapolate that gzip.trace does not have a great temporal locality or spatial locality. More detailed analysis about gzip.trace could be done from the perspective of Load vs. Store, which is elaborated in another part of discussion.

Associativity also has a salient influence on performance. No.1-3 cache hierarchies have the same architecture (Cache + Memory) but different associativities. In the scenario of fully-associativity, an exact block could be mapped into any cachelize, which indicates that conflicts happened in fully-associativity are less than any other kinds of mapping methods. Therefore, it is not surprising at all that No.3 outperforms No.1 and No.2 with such absolute "freedom". However, as we all known that fully-associativity needs more cost of hardware and power than other mapping methods. Also the cycles needed to access a block in fully-associativity could increase dramatically due to the long tag checking procedure. Thus, fully-associativity is not preferred in reality. Direct associativity is not a great option neither. Hit rate in cache using direct associativity is low beyond imagination. This conclusion is conspicuously reflected in the lab results where the relevant cache's size(No.1) is twice bigger than No.2 and 8 times bigger than No.3. Good news is that by using set-associative cache, we could trade off between hit rate and power/cost — No.2 attains nearly the same CPI as No.3 and keeps appropriate size compared with No.1.

The design of Victim cache backfires when the CPI of No.5 is generally greater than No.4. It is plausible that if we reserve some L1 cache's recent discarded cache blocks, we could reduce the number of access to L2 cache. This speculation is based on the assumption that discarded cache blocks have a high probability to be reused in next several instructions. When I scrutinize the collected data about Victim cache, however, Victim cache seldom verifies the above assumption — Victim cache's hit number is too small to improve the overall performance. There are many explanations for this.

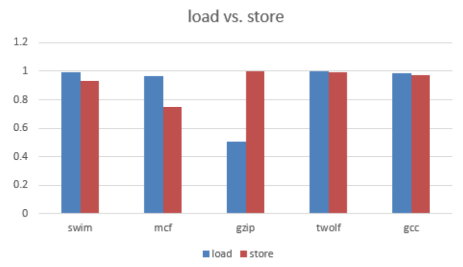


Figure 6: load hit rate vs. store hit rate in No.2 cache hierarchy

Firstly, The Victim cache's configuration, such as cache size or replacement policy may not be optimized. Secondly, I have only run five traces and this cannot rule out the Victim cache's possible significant efficacy under other circumstances. In a nutshell, more specific experiments concerning Victim cache should be done to draw a reliable conclusion.

Besides, I compared Load's hit rate with Store's hit rate in No.2 cache hierarchy in Figure 6. Analysis shows that there is no definite numerical relationship between Load's hit rate and Store's hit rate. What's more, in gzip.trace, Load's hit rate is drastically lower than Store's hit rate, which indicts that compulsory is the main cause to gzip.trace's abnormal low hit rate. This may due to gzip task's feature, which could be analysed in the future work.

4 Conclusion

In this experiment, I implemented a cache simulator to compare different cache hierarchies' performance by running SPEC2000 trace on them respectively. I find out that set-associativity cache trades off between hit rate and power/cost by comparing the statistical data of fully-associativity cache and direct-mapped cache. Based on the visual CPI comparison, the trend that multi-level cache hierarchy could ameliorate performance is definite. What's more, this experiment shows an affinity between Load instruction's hit rate and Store instruction's hit rate. As part of future experiment, I will investigate whether Victim cache really has a significant effect and how to optimize Victim cache's configuration for better performance. Also, I will make a scrutiny into gzip.trace's aberrant behavior. Another facet of cache hierarchy: power is an alternative direction of future experiment.