计算机组织与体系结构实习 Lab 3

高速缓存的模拟、配置和优化

苏睿 1600017705

一、Cache 模拟器设计概述

1. 编译执行

模拟器开发环境为 Ubuntu 18.04.3 LTS，使用的编程语言为 C++，编译器为 g++7.4.0

模拟器是在课程所给的 cache 模拟器的基础上进行添加修改完成的，共包含 6 个文件 storage.h, cache.h, cache.cpp, memory.h, memory.cpp, main.cpp，附有 makefile 文件，使用 make 指令即可编译获得 cache 模拟器 sim。

模拟器执行方式

1) 单级 Cache 模拟 ./sim -t [trace_file] -c [numc] -b [numb] -a [numa]，其中 trace_file 为输入的 trace 文件，numc 为 log(cache size)，numb 为 log(block size)，numa 为 log(associativity)。

如 cache size = 32KB, block size = 64, 相联度为 8，模拟 1.trace 时

./sim -t 1.trace -c 15 -b 6 -a 3

模拟器支持修改 cache 的读写策略，默认情况下 cache 模拟器采用 write allocate + write through + LRU replace algorithm，如需更改，执行时添加

-no-allocate 写不分配

-through 直写

-random 随机替换算法

2) 由于 lab 中(三) cache 的策略优化要求使用两级 cache 并给定了 L1,L2 cache 的参数，因此 ./sim -t [trace_file] -l2 使用给定的 L1, L2 cache 参数执行 trace 文件。

（1.trace/2.trace 和 01.trace/02.trace 中地址分别由 10 进制和 16 进制给出，所以此部分只支持 01.trace/02.trace）

使用预取策略优化 cache，-prefetch

2. 模拟器实现

cache 的工作原理为将下一级存储的部分内容存放在 cache 中，提供更小的访问延迟，根据程序访问的局部性原理，cache 能够很大地提高程序执行的效率。

cache 首先划分为 S 个组，每个组有 E 行，每行保存 B 块，整个 cache 的大小 cache size = S * E * B，块大小 block size = B，相联度 associativity = E。

对于给定的地址 addr，将其切分为三部分，低 log(B)位为块号，中间 log(S)位为组号，高 64 - log(B) - log(S)位为 tag，查找时根据组号找到对应组，依次比较组中每行的 tag，若存在相同的，则发生 cache hit，取出该行对应块号的字节；若不存在相同的，则发生 cache miss，cache 需要从下一级存储中读取 tag 对应的行，替换掉原有的某行，再进行读取。

write back 与 write through 策略：写回表示当对 cache 中的内容修改后，不立刻写入下一级存储，而是等到该行被替换时再写入下一级存储，需要有 modify 位记录该行

是否被修改过；直写表示每次对 cache 内容的修改都直接同步到下一级存储中。

write allocate 与 no write allocate 策略：写分配表示当发生写操作 miss 时，先将下一级存储中的对应内容读取到 cache 中再进行写操作；写不分配表示当发生写 miss 时，直接修改下一级存储中的内容，不载入到当前缓存中。

cache 的替换策略：随机策略，每次发生 cache miss 时，若该组已满需要替换，随机选取一行进行替换；LRU 最近不常使用策略，选取最近不常使用的行替换，实现方式为每行保存一个计数器 cnt，当某行被访问时，其 cnt = 0，其余所有行的 cnt++，替换时选取 cnt 最大的行替换。

二、单级 cache 模拟

./sim -t [trace_file] -c [numc] -b [numb] -a [numa] [-no-allocate] [-through] [-random]

运行 1.trace 的结果如图

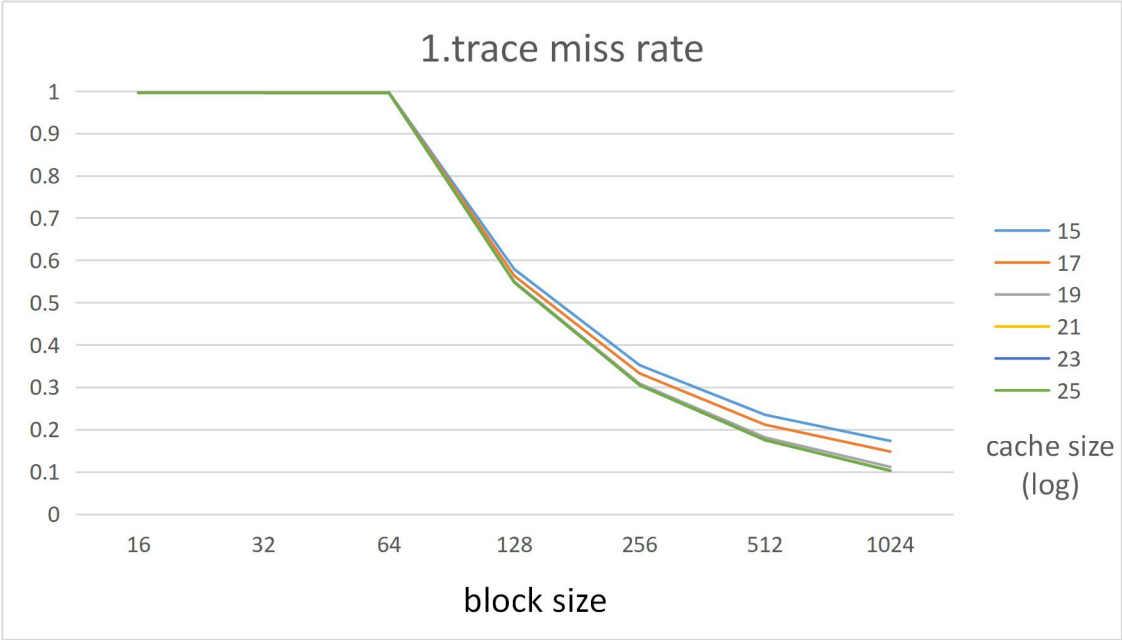
```
surui@surui:~/Documents/architecture/cache/cache$ ./sim -t ../trace/1.trace -c 15 -b 8 -a 3
 l1 cache total access count: 10001, total access time: 361801
                 hit count: 6483, miss count: 3518
          replace count: 3390, fetch lower layer count: 3518
          prefetch count: 0
 l1 cache miss rate : 0.351765
 l1 cache average access time : 36.176482
 memory total access count: 6688
```

1. 在不同的 Cache Size（ 32KB ~ 32MB）的条件下，Miss Rate 随 Block Size 变化的趋势，收集数据并绘制折线图。并说明变化原因。

log(Cache Size) 分别取 15(32KB), 17(128KB), 19(512KB), 21(2MB), 23(8MB), 25(32MB)，log(Block Size)分别取 4(16), 5(32), 6(64), 7(128), 8(256), 9(512), 10(1024)，相联度固定为 8，策略固定为 write back, write allocate，替换策略为 LRU 替换。

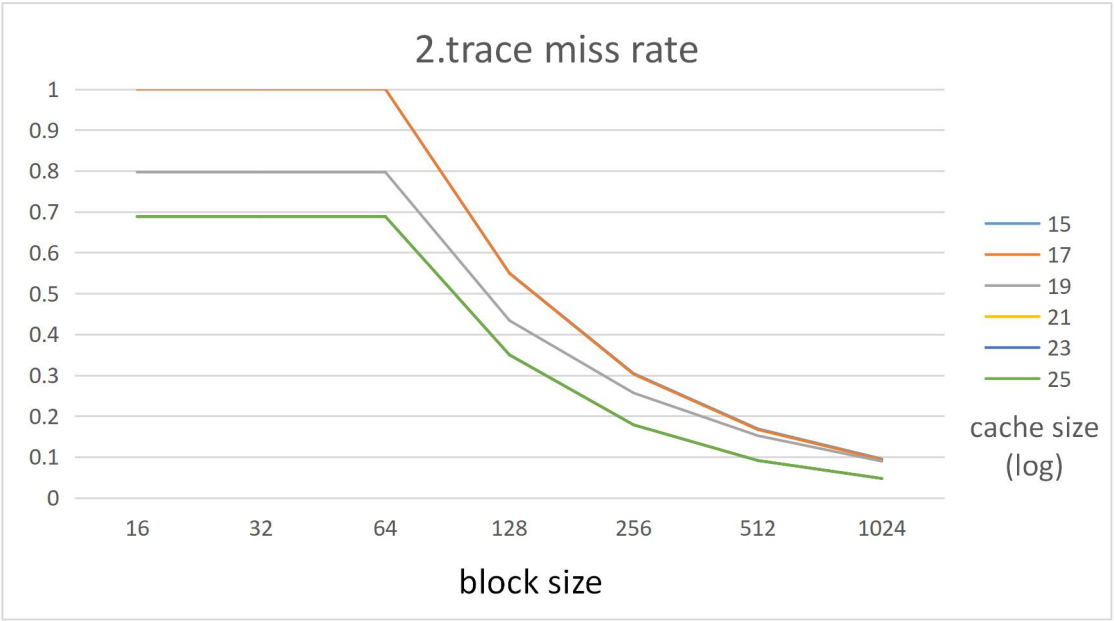
对 1.trace 测试的 miss rate：

cache size block size	15(32KB)	17(128KB)	19(512KB)	21(2MB)	23(8MB)	25(32MB)
16	0.9966	0.9965	0.9964	0.9954	0.9954	0.9954
32	0.9964	0.9963	0.9962	0.9952	0.9952	0.9952
64	0.9961	0.9960	0.9959	0.9949	0.9949	0.9949
128	0.5785	0.5628	0.5493	0.5474	0.5474	0.5474
256	0.3518	0.3323	0.3080	0.3046	0.3046	0.3046
512	0.2346	0.2112	0.1811	0.1748	0.1748	0.1748
1024	0.1728	0.1476	0.1114	0.1025	0.1025	0.1025



对 2.trace 测试的 miss rate：

cache size block size	15(32KB)	17(128KB)	19(512KB)	21(2MB)	23(8MB)	25(32MB)
16	1.0000	1.0000	0.7968	0.6882	0.6882	0.6882
32	1.0000	1.0000	0.7968	0.6882	0.6882	0.6882
64	1.0000	1.0000	0.7968	0.6882	0.6882	0.6882
128	0.5499	0.5488	0.4337	0.3496	0.3496	0.3496
256	0.3041	0.3023	0.2564	0.1784	0.1784	0.1784
512	0.1688	0.1668	0.1520	0.0912	0.0912	0.0912
1024	0.0951	0.0932	0.0892	0.0472	0.0472	0.0472



可以看到，在 cache size 和相联度都固定的情况下，miss rate 随 block size 的增大而减小，根据程序的局部性原理，相邻两次存储访问地址总是相近的，随着 block size 的增大，相邻两次访问落在相同 block 的概率也就增加，从而使 miss rate 减小。当 block size 小到一定程度 miss rate 就变得接近 100%。

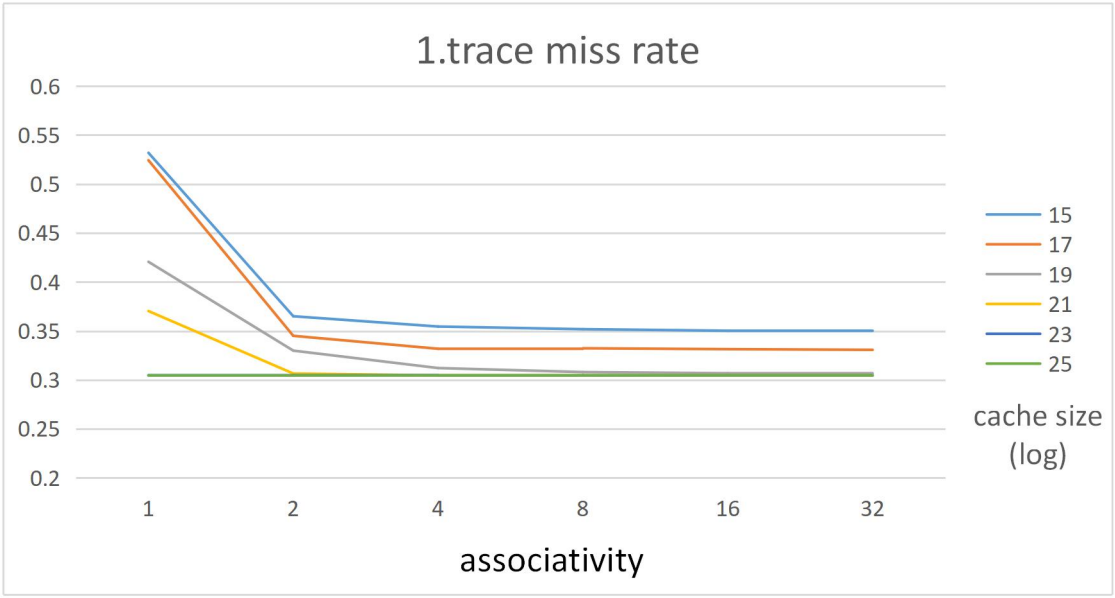
在 block size 和相联度都固定的情况下，miss rate 随 cache size 的增大而减小，这是很明显的，高速缓存中存储的内容增多自然会导致 miss rate 减小。当 cache size 增大到一定程度后，miss rate 不再继续减小，这可能是因为所给 trace 中数据访问的范围不那么大。

2. 在不同的 Cache Size 的条件下，Miss Rate 随 Associativity（1-32）变化的趋势，收集数据并绘制折线图。并说明变化原因。

这里采用固定 block size 的方式，log(Cache Size)分别取 15(32KB), 17(128KB), 19(512KB), 21(2MB), 23(8MB), 25(32MB)，log(associativity)分别取 0(1), 1(2), 2(4), 3(8), 4(16), 5(32)，block size 固定为 256，策略固定为 write back, write allocate，替换策略为 LRU 替换。

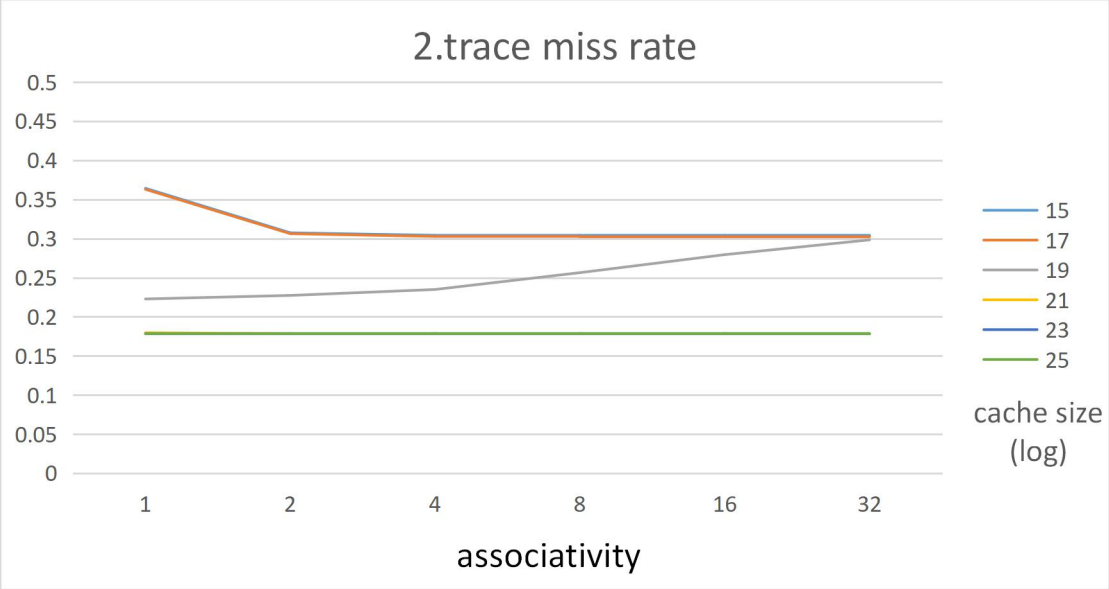
对 1.trace 测试的 miss rate：

cache size associativity	15(32KB)	17(128KB)	19(512KB)	21(2MB)	23(8MB)	25(32MB)
1	0.5319	0.5243	0.4206	0.3703	0.3047	0.3046
2	0.3650	0.3450	0.3299	0.3065	0.3048	0.3046
4	0.3545	0.3318	0.3121	0.3046	0.3046	0.3046
8	0.3518	0.3323	0.3080	0.3046	0.3046	0.3046
16	0.3502	0.3314	0.3068	0.3046	0.3046	0.3046
32	0.3499	0.3307	0.3069	0.3046	0.3046	0.3046



对 2.trace 测试的 miss rate：

cache size associativity	15(32KB)	17(128KB)	19(512KB)	21(2MB)	23(8MB)	25(32MB)
1	0.3642	0.3629	0.2228	0.1796	0.1784	0.1784
2	0.3073	0.3062	0.2273	0.1784	0.1784	0.1784
4	0.3041	0.3027	0.2349	0.1784	0.1784	0.1784
8	0.3041	0.3023	0.2564	0.1784	0.1784	0.1784
16	0.3041	0.3023	0.2794	0.1784	0.1784	0.1784
32	0.3041	0.3023	0.2983	0.1784	0.1784	0.1784



在 block size 和 cache size 都固定时，随着 associativity 的增加，1.trace 的 miss rate 减少，但 2.trace 的 miss rate 增加，这是因为随着 associativity 的增加，cache 的 set number 减少，从而导致 cache 中 tag 位变短，从而数据的访存顺序会产生影响，但整体的 miss rate 变化并不大。

在 block size 和 associativity 都固定时，随着 cache size 的增加，miss rate 不断减小，与之前的原因是一样的，高速缓存空间增大导致存储内容增大，命中率增加。

3. 比较 Write Through 和 Write Back、Write Allocate 和 No-write Allocate 的总访问延时的差异。

固定 cache size 为 32KB, block size 为 256，相联度为 8，替换策略采用 LRU。

假设 cache 的访问延迟为 1cycle，memory 的访问延迟为 100cycle。

对 1.trace 测试的 miss rate：

miss rate / average cycle	write back	write through
write allocate	0.3518 / 36.18	0.3518 / 67.12
no write allocate	0.6575 / 66.75	0.6575 / 66.90

对 2.trace 测试的 miss rate：

miss rate / average cycle	write back	write through
write allocate	0.3041 / 31.41	0.3041 / 62.66
no write allocate	0.6157 / 62.57	0.6157 / 62.57

在 write back + write allocate 情况下，average cycle = miss rate * (100+1) + (1-miss rate) *1，write through 在每次写操作时都会将修改直接写入 memory 中，因此会较 write back 有更大的延迟，但不会影响 miss rate；no write allocate 在每次写访问 miss 时会直接将修改写入 memory 中，但不将该行装载入 cache，由于程序访问的局部性原理，miss rate 会较 write allocate 更大。

因此，最优的策略应为 write back + write allocate。

三、与 lab2 中 CPU 模拟器联调

将 lab2 中的 riscv 模拟器修改，添加以下参数的三级 cache，设置 memory 的访问延迟为 100 cpu cycle，执行 ./simulator -P [input_file]。

Level	Capacity	Associativity	Line size(Bytes)	WriteUp Polity	Hit Latency
L1	32 KB	8 ways	64	write Back	1 cpu cycle
L2	256 KB	8 ways	64	write Back	8 cpu cycle
LLC	8 MB	8 ways	64	write Back	20 cpu cycle

```
surui@surui:~/Documents/architecture/simulator$ ./simulator -P bin_code/add
11 12 13 14 15 1 2 3 4 5
instuction number : 642, cycle number : 1122, condition branch error rate : 0.130
CPI : 1.747664
simulate over!
l1 cache status :
        access count: 932, miss count: 10
        replace count: 0, fetch lower layer count: 10
        miss rate: 0.010730
l2 cache status :
        access count: 10, miss count: 5
        replace count: 0, fetch lower layer count: 5
        miss rate: 0.500000
l1 cache status :
        access count: 5, miss count: 2
        replace count: 0, fetch lower layer count: 2
        miss rate: 0.400000
memory status :
       access count: 2
```

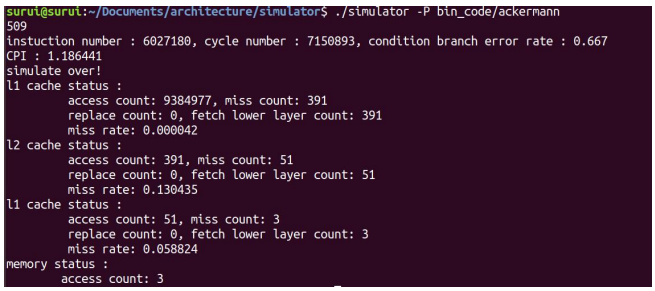
```
surui@surui:~/Documents/architecture/simulator$ ./simulator -P bin_code/mul-div
5 10 15 20 25 1 2 3 4 5
instuction number : 667, cycle number : 1147, condition branch error rate : 0.130
CPI : 1.719640
simulate over!
l1 cache status :
        access count: 957, miss count: 10
        replace count: 0, fetch lower layer count: 10
        miss rate: 0.010449
l2 cache status :
        access count: 10, miss count: 5
        replace count: 0, fetch lower layer count: 5
        miss rate: 0.500000
l1 cache status :
        access count: 5, miss count: 2
        replace count: 0, fetch lower layer count: 2
        miss rate: 0.400000
memory status :
       access count: 2
```

```
surui@surui:~/Documents/architecture/simulator$ ./simulator -P bin_code/n
3628800
instuction number : 283, cycle number : 713, condition branch error rate : 0.100
CPI : 2.519435
simulate over!
l1 cache status :
        access count: 404, miss count: 12
        replace count: 0, fetch lower layer count: 12
        miss rate: 0.029703
l2 cache status :
        access count: 12, miss count: 5
        replace count: 0, fetch lower layer count: 5
        miss rate: 0.416667
l1 cache status :
        access count: 5, miss count: 2
        replace count: 0, fetch lower layer count: 2
        miss rate: 0.400000
memory status :
       access count: 2
```

```
surui@surui:~/Documents/architecture/simulator$ ./simulator -P bin_code/qsort
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
instuction number : 19546, cycle number : 24567, condition branch error rate : 0.474
CPI : 1.256881
simulate over!
l1 cache status :
        access count: 32108, miss count: 46
        replace count: 0, fetch lower layer count: 46
        miss rate: 0.001433
l2 cache status :
        access count: 46, miss count: 9
        replace count: 0, fetch lower layer count: 9
        miss rate: 0.195652
l1 cache status :
        access count: 9, miss count: 2
        replace count: 0, fetch lower layer count: 2
        miss rate: 0.222222
memory status :
       access count: 2
```

```
surui@surui:~/Documents/architecture/simulator$ ./simulator -P bin_code/simple-function
11 12 13 14 15 1 2 3 4 5
instuction number : 651, cycle number : 1111, condition branch error rate : 0.130
CPI : 1.706605
simulate over!
l1 cache status :
        access count: 943, miss count: 10
        replace count: 0, fetch lower layer count: 10
        miss rate: 0.010604
l2 cache status :
        access count: 10, miss count: 4
        replace count: 0, fetch lower layer count: 4
        miss rate: 0.400000
l1 cache status :
        access count: 4, miss count: 2
        replace count: 0, fetch lower layer count: 2
        miss rate: 0.500000
memory status :
       access count: 2
```

```
surui@surui:~/Documents/architecture/simulator$ ./simulator -P bin_code/mat-mul
1 2 3 4 5
0 1 0 0 0
0 0 1 0 0
0 0 0 1 0
0 0 0 0 1
instuction number : 8492, cycle number : 10084, condition branch error rate : 0.167
CPI : 1.187471
simulate over!
l1 cache status :
        access count: 12085, miss count: 18
        replace count: 0, fetch lower layer count: 18
        miss rate: 0.001489
l2 cache status :
        access count: 18, miss count: 5
        replace count: 0, fetch lower layer count: 5
        miss rate: 0.277778
l1 cache status :
        access count: 5, miss count: 2
        replace count: 0, fetch lower layer count: 2
        miss rate: 0.400000
memory status :
       access count: 2
```



（哇，图中第三个 status 应该是 LLC cache，搞错懒得改了....）

所得结果记录如下表：

	L1 miss rate	L2 miss rate	LLC miss rate	指令数	周期数	CPI	原 CPI
add	0.0107	0.5000	0.4000	642	1122	1.75	1.16
mul-div	0.0104	0.5000	0.4000	667	1147	1.72	1.15
n	0.0297	0.4167	0.4000	283	713	2.52	1.12
qsort	0.0014	0.1957	0.2222	19546	24567	1.26	1.22
simple-function	0.0106	0.4000	0.5000	651	1111	1.71	1.15
mat-mul	0.0015	0.2778	0.4000	8492	10084	1.19	1.14
ackermann	0.0000	0.1304	0.0588	6027180	7150893	1.19	1.19

由上表可以看出，L1 cache 的 miss rate 均较小，L2, L3 cache 由于访问次数很少的原因 miss rate 较大（ackerann 的 L2, L3 miss rate 仍很小），说明 cache 模拟性能还是很好的。

简单分析以上数据，add 中 L2 访问次数 10，LLC 访问次数 5，memory 访问次数 2，额外的周期数为 10 * 8 + 5 * 20 + 2 * 100 = 380，原 lab 中 add 的周期数为 742，恰好满足 742 + 380 = 1122，与结果吻合。

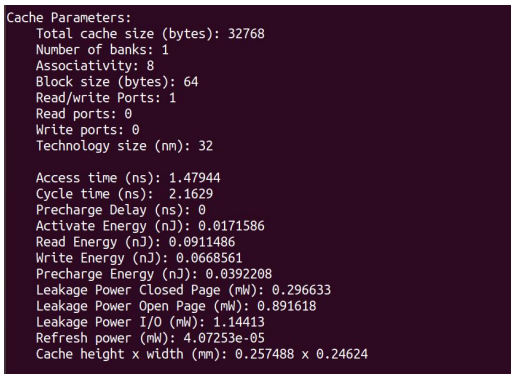
在加入 cache 模拟后，CPI 比原 CPI 增大了，但我们发现，指令数越少，CPI 增大的就越多，指令数很大的如 qsort, mat-mul, ackermann，CPI 较原 CPI 增加很少，这是由于 LLC, memory 的访问次数都很少，额外的周期数在几百的数量级，指令数越多，这些额外的周期数影响就越小。

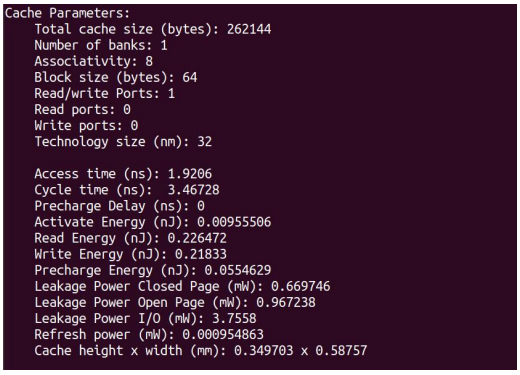
四、高速缓存管理策略优化

Level	Capacity	Associativity	Line Size(Bytes)	WriteUp Policy	Hit Latency
L1	32KB	8 ways	64	write back	1 cpu cycle
L2	256KB	8 ways	64	write back	2 cpu cycle

1. 使用 cacti65 存储模型计算给定参数的 cache 延迟。

下载 cacti65；安装依赖库 g++-multilib；修改 cache.cfg 中的 cache 参数为 L1 和 L2；分别执行./cacti -infile cache.cfg，所得结果如下：





得到 L1 cache 的 Hit Latency = 1.48ns；L2 cache 的 Hit Latency = 1.92ns。

假设 1 cpu cycle = 1.48ns（即 cpu 频率为 676MHz），此时 L1 cache 的访问延迟为 1 cpu cycle，L2 cache 的访问延迟为 2 cpu cycle。

2. 执行 trace 文件 01-mcf-gem5-xcg, 02-stream-gem5-xaa

./sim -t [trace_file] -l2

cache 访问策略为 write back + write allocate，替换策略为 LRU，由于不存在随机性，因此以下结果只运行 1 遍即可。

AMAT 通过公式 AMAT = hit time + miss rate * penalty 来计算，这里的 penalty 为 L1 cache miss 后的额外需要的周期数，使用 L2 的 average access time 计算。

01-mcf-gem5-xcg

```
surui@surui:~/Documents/architecture/cache/cache$ ./sim -t ../trace/01-mcf-gem5-xcg.trace -l2
 l1 cache total access count: 232612, total access time: 2558368
                 hit count: 185984, miss count: 46628
          replace count: 46116, fetch lower layer count: 46628
 l1 cache miss rate : 0.200454
 l1 cache average access time : 10.998435

 l2 cache total access count: 89662, total access time: 2411824
                 hit count: 67337, miss count: 22325
          replace count: 18229, fetch lower layer count: 22325
 l2 cache miss rate : 0.248991
 l2 cache average access time : 26.899065

 memory total access count: 39305
```

- L1 Cache： 平均 Miss Rate = 0.2005
- L2 Cache： 平均 Miss Rate = 0.2490
- AMAT =hit time + miss rate * penalty = 1 + 0.2005 * 26.90 = 6.39

02-stream-gem5-xaa

```
surui@surui:~/Documents/architecture/cache/cache$ ./sim -t ../trace/02-stream-gem5-xaa.trace -l2
 l1 cache total access count: 162897, total access time: 2047143
                 hit count: 144424, miss count: 18473
          replace count: 17961, fetch lower layer count: 18473
 l1 cache miss rate : 0.113403
 l1 cache average access time : 12.567101

 l2 cache total access count: 35926, total access time: 1919152
                 hit count: 17453, miss count: 18473
          replace count: 14377, fetch lower layer count: 18473
 l2 cache miss rate : 0.514196
 l2 cache average access time : 53.419585

 memory total access count: 32342
```

- L1 Cache： 平均 Miss Rate = 0.1134

- L2 Cache： 平均 Miss Rate = 0.5142

- AMAT =hit time + miss rate * penalty = 1 + 0.1134 * 53.42 = 7.06

3. 优化 cache

采用预取策略，每次发生访问 miss 时，取出访问行的同时一并取出下一行。

由于程序访问的局部性原理，若访问发生 miss 时，其下一行在近期被访问的概率是很大的，采用此策略可以有效降低 miss rate。

将本策略应用到 L1 和 L2 cache 上，其余策略不变。

./sim -t [trace_file] -l2 -prefetch

01-mcf-gem5-xcg

```
surui@surui:~/Documents/architecture/cache/cache$ ./sim -t ../trace/01-mcf-gem5-xcg.trace -l2 -prefetch
 l1 cache total access count: 232612, total access time: 2658406
                 hit count: 191015, miss count: 41597
         replace count: 41320, fetch lower layer count: 41597
         prefetch count: 30040
 l1 cache miss rate : 0.178826
 l1 cache average access time : 11.428499

 l2 cache total access count: 91557, total access time: 2548414
                 hit count: 67904, miss count: 23653
         replace count: 21767, fetch lower layer count: 23653
         prefetch count: 14061
 l2 cache miss rate : 0.258342
 l2 cache average access time : 27.834180

 memory total access count: 50962
```

- L1 Cache： 平均 Miss Rate = 0.1788

- L2 Cache： 平均 Miss Rate = 0.2583

- AMAT =hit time + miss rate * penalty = 1 + 0.1788 * 27.83 = 5.98

02-stream-gem5-xaa

```
surui@surui:~/Documents/architecture/cache/cache$ ./sim -t ../trace/02-stream-gem5-xaa.trace -l2 -prefetch
 l1 cache total access count: 162897, total access time: 1105377
                 hit count: 153657, miss count: 9240
         replace count: 8981, fetch lower layer count: 9240
         prefetch count: 9236
 l1 cache miss rate : 0.056723
 l1 cache average access time : 6.785742

 l2 cache total access count: 30197, total access time: 984394
                 hit count: 20957, miss count: 9240
         replace count: 7189, fetch lower layer count: 9240
         prefetch count: 9236
 l2 cache miss rate : 0.305991
 l2 cache average access time : 32.599066

 memory total access count: 27722
```

- L1 Cache： 平均 Miss Rate = 0.0567

- L2 Cache： 平均 Miss Rate = 0.3060

- AMAT =hit time + miss rate * penalty = 1 + 0.0567 * 32.60 = 2.85

可以发现，L1 cache 的 miss rate 和 AMAT 相较未使用预取策略时均有所降低。

五、收获与总结

复习了关于高速缓存的内容，自己动手实现了一个 cache 模拟器，能够支持 write back/write through, write allocate/no write allocate, random replace/LRU replace, 以及预取策略，对高速缓存有了更深的理解。