# 系统结构第一次实验报告

#### 实验环境为 linux

叶奕宁 2019013287 计97

#### 系统结构第一次实验报告

- 1.测量Cache Size
- 2. 测量 Cache Line size
- 3.测量L1D Cache相联度

算法

原理剖析

实验

- 4.矩阵乘优化
- 5.可选部分2

测量Cache 替换策略

测量Cache是否写直达

6.对本次实验的意见和建议。

## 1.测量Cache Size

#### 首先调用

```
lscpu | grep cache
```

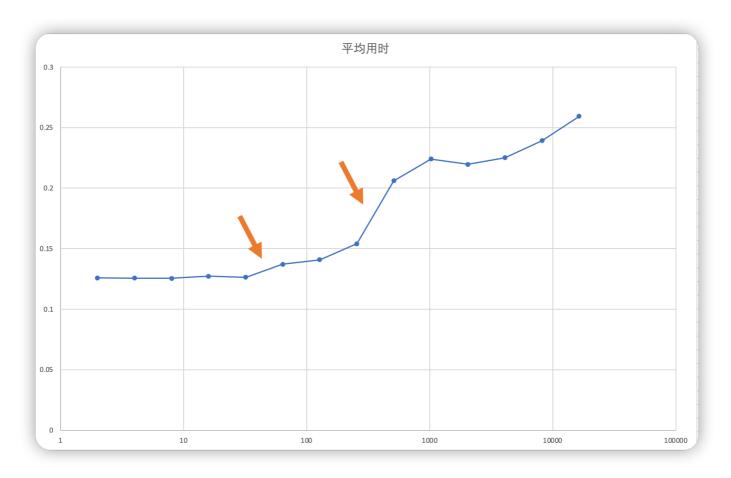
#### 看出系统的缓存大小为

```
L1d cache: 32K
L1i cache: 32K
L2 cache: 256K
L3 cache: 35840K
```

- 开一个数组
- 多次随机访问数组位置(用伪随机算法加速,提前算好随机数序列,千万别在循环内部生成随机数,要不然访存时间测不出来了...)
- 计算时间
- 改变数组大小多次尝试

#### 得到访问时间如下:

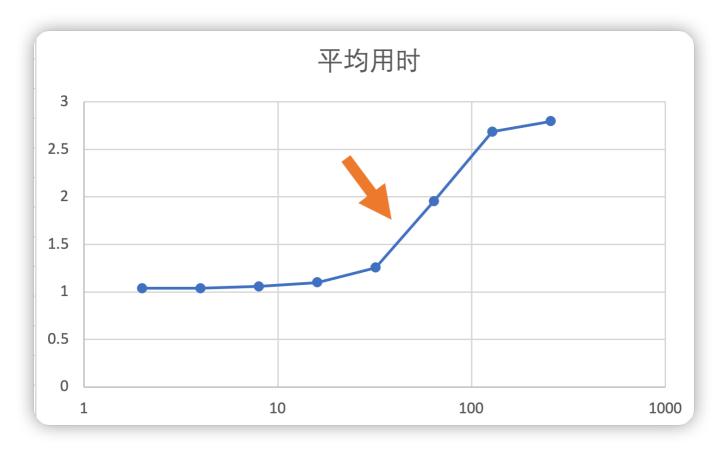
```
4 KB, 0.125688s, dummy:33554432
8 KB, 0.125575s, dummy:33554432
16 KB, 0.127201s, dummy:33554432
32 KB, 0.126343s, dummy:33554432
64 KB, 0.135202s, dummy:33554432
128 KB, 0.140837s, dummy:33554432
256 KB, 0.174021s, dummy:33554432
512 KB, 0.206156s, dummy:33554432
1024 KB, 0.224174s, dummy:33554432
2048 KB, 0.219786s, dummy:33554432
4096 KB, 0.222523s, dummy:33554432
8192 KB, 0.239338s, dummy:33554432
16384 KB, 0.259365s, dummy:33554432
32768 KB, 0.414883s, dummy:33554432
65536 KB, 0.664532s, dummy:33554432
131072 KR 0 754477s dummy 33554432
```



可以看出,当数组大小从 32KB->64KB,256KB->512KB 时,访存用时有一个跳变式的提升。这分别对应于L1缓存失效和L2缓存失效

## 2. 测量 Cache Line size

- 使用不同步长对大于 L1D Cache Size 的某个数组做相同次数的访问。
- 如果访问是间断的,对数组间隔顺序访问,命中率就会降低,平均访问延迟增大。当间隔达到一定的大小,即超过 Cache Line Size,将造成每次都缺失的最坏情况,平均访问延迟达到最大。



由图可以看出,当跨度 32B->64B 时,用时有一个明显的增大,cache缺失率变大,推测Cache Line大小为64B

### 3.测量L1D Cache相联度

#### 算法

使用如下算法测量相联度:

- 使用一个 2 倍 Cache Size 大小的数组
- 将数组分为  $2^n$  块,只访问其中的奇数块
- 逐渐增大 n 的取值,当某一次访问时间变慢时, $2^{n-2}$  就是相联度

#### 原理剖析

测量的原理是尝试制造"相联度冲突"强制触发CacheLine的驱逐,以此来推算出Cache的相联度。

- 假设缓存为空, 当访问一块连续的内存时, 会在缓存的同一个 way 中顺序加载。
- 由于一个大小等同于2倍 Cache Size 的数组被分为了 $2^n$ 块,因此若已知相联度为2的正整数次方,单个块的大小只可能是1个 Way 总大小的2的整数次方倍(可以是负整数,即可以是2倍、1倍、1/2倍、1/4倍等)。
- 当单个块的大小为一个 way 总大小的整数倍时,一个块会占满整数个 way 。其后面紧跟着的一个不访问的块同样也会映射满整数个 way ,再下一个需要访问的块同理,因此这样所有的奇数块会刚好"占满"缓存,不会浪费。
- 但只要n增大到恰好使得一个块的大小只相当于1/2个 way 的大小。假如第一个块映射到了 Set 3 和 Set 2,

那么下一个不访问的块会映射到 Set1 和 Set0, 再下一个访问的块又映射到了 Set 3 和 Set 2。

- .....
- 以此类推,所有访问的块都映射到 Set 3和Set 2,所有不访问的块都映射到 Set 1和 Set 0。因此有一半的缓存都没有使用,导致缺失和替换的次数显著增加,访问时间明显增加。
- ullet 此时,假设原数组分为了 $2^n$ 块,缓存大小为S,数组大小为2S,一个块的大小为 $\frac{S}{2^{n-1}}$ ,一个Way的大小为 $\frac{S}{2^{n-2}}$ ,因此相联度为 $2^{n-2}$ 。

#### 实验

带入 Cache Size = 32KB, 可以明显看出在分成32块时用时有一个明显的提升, 因此推测相联度为8

```
interval = 2, time = 1.35165s, temp = 1073741824
interval = 4, time = 1.32557s, temp = 1073741824
interval = 8, time = 1.31667s, temp = 1073741824
interval = 16, time = 1.31976s, temp = 1073741824
interval = 32, time = 1.41856s, temp = 1073741824
interval = 64, time = 1.43129s, temp = 1073741824
interval = 128, time = 1.40214s, temp = 1073741824
interval = 256, time = 1.49887s, temp = 1073741824
interval = 512, time = 1.71985s, temp = 1073741824
```

### 4.矩阵乘优化

```
time spent for original method: 6.47663 s
time spent for new method: 2.22641 s
time ratio of performance optimization: 2.909
score: 2
```

- 首先,将循环顺序改一下,让所有的调用对于最内层是顺序访问,发现速度提升2倍
- 接下来,对于b的调用提到外面,并且改成 register int ,发现变成了3倍
- 实际测试,用不同的 block size 分块计算好像并没有变得更好...不知道为啥

总体代码如下

```
register int ii, jj, kk, temp;
for (ii = 0; ii < MATRIX_SIZE; ii++) {
  for (kk = 0; kk < MATRIX_SIZE; kk++) {
    temp = a[ii][kk];
    for (jj = 0; jj < MATRIX_SIZE; jj++) {
      d[ii][jj] += temp * b[kk][jj];
    }
}</pre>
```

最牛的是开了-O3以后优化了10倍,还是O3厉害(

### 5.可选部分2

可选部分1没写......

### 测量Cache 替换策略

我们考虑这件事,内存的平均访问时间通过如下方法计算:

$$T = ext{hit} imes T_{ ext{cache}} + (1 - ext{hit}) imes T_{ ext{miss}} \ = ext{hit} imes A + B \ \propto ext{hit}$$

其中 $T_{\text{cache}}$ , $T_{\text{miss}}$ 对于一个芯片是常数。

这个道理告诉我们如何猜测替换策略:

- 对于一组数,不同的替换策略会导致不同的hit率
- 我们随机生成很多的数组,每一组都计算不同替换策略的hit率,同时统计用时
- 我们最终得到很多个hit率序列,还有一个时间序列
- 进行相关性分析,最相关的序列最可能就是我们猜测正确的hit率序列,对应的策略很可能就是CPU的cache替 换策略

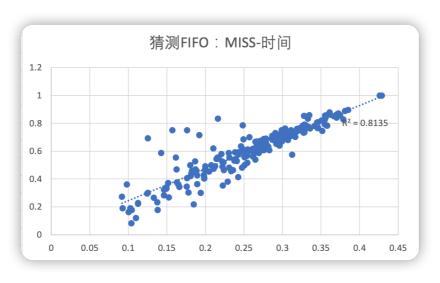
然而,我们该怎么精确控制cache的hit和miss呢?我们用 posix\_memalign 函数,该函数可以帮助申请一大堆对齐的内存.

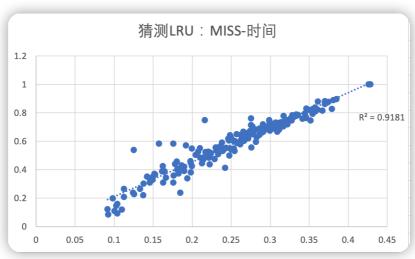
- 我们控制申请一大堆对齐为 4096KB 的内存,由于Cache里面的hash函数是低位地址,因此这些内存一定会对应到同一个 Cache Set 里
- 我们知道自己CPU的相联度为8,因此访问数组Arr中 index=0,1,2,3,4,5,6,7 的内容总能hit,而一旦访问别的,就会触发缺失,同时我们可以通过index的访问历史判断不同策略下具体会替换谁

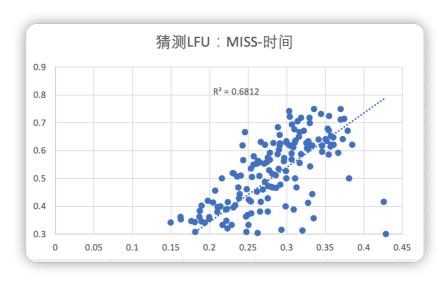
换句话说,我们用这种方法可以实时观测cache中的内容具体是什么。我们也可以用这种方法观察cache相联度

我们实现了FIFO类、LRU类、LFU类,对不同的序列模拟命中率,模型替换过程。

接下来我们进行随机模拟,得到猜测hit与实际时间的散点图,并进行拟合,结果如下:







#### 拟合的结果分析上看,LRU的结果最符合线性,推测替换策略为LRU

虽然拟合以后看起来策略的 $R^2$ 看起来都不高,但这是有道理的:

- 对于一个访问过程来说,每次访问时由于CPU占用等原因可能会有一个偏差P。因此整个序列都会有一个偏差,且每个点的偏差都不同,这取决于当时电脑的运行状况。最终带来拟合效果偏差
- 同时不知道CPU会不会有什么怪优化,怪缓存对结果造成影响...总之看不同策略对比还是挺有区别的......

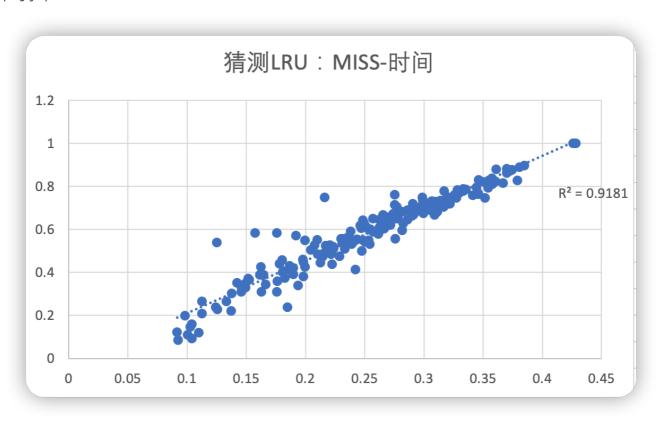
0   11707 2   3   407 2   507 6   607 0   70 707 4   807 2   879   10 10 7 970   11   12   13   14   13   15   15   10   12   12   12   12   12   13   14   13   13   13   13   13   13	Sequence		FIFO	LRU	MRU	MRU _N	NRU	PLRU	PLRUI	HOO	QLRU _H00 _M1 _R2 _U1	QLRU _H11 _M1 _R0 _U0	QLRU _H11 _M1 _R1 _U2	QLRU _H21 _M1 _R0 _U0 _UMO	SRRII
10.1117121717117107034567689010166712134151737576761617181920127212232242526172777671222727   15   15   18   18   18   18   18   18	0 ? 1 0 ? 0 ? 0 ? 0 ? 2 0 ? 3 4 2 ? 5 5 ? 5 ? 6 0 ? 7 8 9 10 0 ? 11 9 ? 12 10 ? 9 ? 13 14 15 16 0 ? 17 2 ? 0 ? 18 19 20 0 ? 21 22 0 ? 19 ? 23 5 ? 24 25 4 ? 2 ? 4 ? 26	19	18	19	19	19	19	19	19	19	20	19	18	19	20
1907  12 070 070 073 45 677 8 870 070 373 71 101 121 1387 14 1537 16 17 18 190 202 21 22 23 122 34 172 25 277 24 177 22 172 22 23 072 44   18			18	20	20	19	18	20	20	21	21	20	20	21	22
97 1 1 7 0 7 0 7 0 7 2 0 7 0 7 3 4 5 6 7 0 7 8 9 10 0 7 0 7 1 1 1 2 1 3 8 7 1 1 4 6 1 0 7 3 7 1 1 1 2 1 3 8 7 1 1 4 6 1 0 7 3 7 1 1 2 1 3 8 7 1 1 4 6 1 0 7 3 7 1 1 2 1 3 8 7 1 1 4 6 1 0 7 3 7 1 1 2 1 3 8 7 1 1 4 6 1 0 7 3 7 1 1 2 1 3 8 7 1 1 4 6 1 0 7 3 7 1 1 2 1 3 8 7 1 1 4 6 1 0 7 3 7 1 1 2 1 3 8 7 1 1 4 6 1 1 3 8 7 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1			14	15	15	15	15	15	15	15	15	15	15	15	18
107 2 3 37 4 5 6 37 7 8 9 37 10 77 37 11 12 77 37 13 67 07 14 15 37 47 16 17 18 19 67 47 20 207 21 37 167 22 23 24 227 37 25 37 37 25 72 61 17 27  107 107 12 3 4 5 6 7 3 7 8 9 37 10 77 37 11 12 77 37 13 67 07 14 15 37 47 16 17 18 19 67 47 20 207 21 37 167 22 23 24 22 50 72 61 37 227  108 10 12 3 40 7 5 60 7 7 8 07 27 20 10 10 70 11 12 13 41 51 60 72 70 18 19 22 20 37 21 22 23 24 25 50 72 61 37 227  109 10 12 3 4 5 6 7 3 7 8 9 10 11 07 07 12 13 41 51 60 72 71 71 18 19 20 12 20 37 21 22 23 24 25 26 37  100 12 23 4 5 6 7 8 7 8 9 10 11 8 7 07 67 8 12 9 10 11 8 7 07 6 16 71 18 19 20 21 07 07 37 97 22 23 07 107 167 107 22 27 8 7 24 25  100 11 107 17 12 17 23 40 7 5 07 07 6 17 7 27 17 8 9 10 11 8 7 07 6 17 17 18 19 20 12 22 34 25 26 37  107 12 17 12 17 12 17 12 17 13 17 14 15 16 17 18 19 10 17 18 18 19 10 17 18 18 19 10 18 20 11 17 18 18 19 10 18 20 11 17 18 18 19 10 18 20 11 17 18 18 19 10 18 20 11 17 18 18 19 10 18 20 11 17 18 18 19 10 18 20 11 17 18 18 19 10 18 20 11 17 18 18 19 10 18 20 11 17 18 18 19 10 18 20 11 17 18 18 19 10 18 20 11 17 18 18 19 10 18 20 11 17 18 18 19 10 18 20 11 17 18 18 19 10 18 20 11 18 18 18 18 18 18 18 18 18 18 18 18			13	15	15	14	14	15	15	15	15	15	14	15	15
11 20 707 17 3 37 4 07 5 607 7 8 07 27 07 9 10 27 37 11 07 107 12 07 57 07 13 37 107 14 15 16 17 97 18 19 22 10 37 21 12 22 32 42 50 72 61 137 227 13 4 15 10 107 12 13 14 15 16 07 227 17 17 18 19 20 10 17 27 37 97 12 22 32 42 50 67 38 17 17 17 18 14 15 17 18 19 18 19 18 19 18 19 18 19 18 18 18 18 18 18 18 18 18 18 18 18 18	0? 1 1? 0? 0? 0? 2 0? 0? 3 4 5 6 7 0? 8 9 10 0? 0? 3? 11 12 13 8? 1? 14 6? 0? 3? 15 0? 9? 16 0? 17 1? 18 19 14? 20 8? 0? 7? 21 21? 22 23 0? 24	18	16	16	16	16	16	16	18	16	18	16	16	17	18
100 1 2 3 4 5 6 7 3 7 5 7 1 7 8 9 5 7 3 7 10 1 7 0 7 4 7 11 1 2 1 3 1 4 1 5 1 6 0 7 2 7 1 7 1 7 18 19 20 15 7 2 1 2 7 17 2 2 17 2 3 9 7 2 4 197 2 5 2 6 7 19 3 7 2 1 7 1 7 7 15 15 17 15 17 15 16 16 17 17 17 18 18 19 20 1 2 10 2 10 2 10 2 10 2 10 10 10 10 10 10 10 10 10 10 10 10 10	10 ? 2 3 3? 4 5 6 3? 7 8 9 3? 10 7? 3? 11 12 7? 3? 13 6? 0? 14 15 3? 4? 16 17 18 19 6? 4? 20 20? 21 3? 16? 22 23 24 22? 3? 25 3? 3? 25? 26 11? 27	17	16	16	16	16	15	17	17	17	17	17	16	16	17
1000000000000000000000000000000000000	1 2 0? 0? 1? 3 3? 4 0? 5 6 0? 7 8 0? 2? 0? 9 10 2? 3? 11 0? 10? 12 0? 5? 0? 13 3? 10? 14 15 16 17 9? 18 19 2? 20 3? 21 22 23 24 25 0? 26 13? 22?	16	15	16	16	16	16	16	16	18	19	16	15	16	18
10   10   10   10   10   10   10   10	0? 1 2 3 4 5 6 7 3? 5? 1? 8 9 5? 3? 10 1? 0? 4? 11 12 13 14 15 16 0? 2? 17 1? 18 19 20 15? 21 2? 11? 22 1? 23 9? 24 19? 25 26 27 19? 3? 21? 1? 7?	13	11	10	9	9	9	11	13	11	11	10	10	11	13
11707234171727075671757891011870767871297913107141516137107171818719202122324256272897299717	07 07 07 12 3 4 37 27 5 6 7 27 8 67 9 77 97 47 17 10 11 12 13 47 14 37 15 137 07 16 17 18 19 20 21 07 07 37 97 22 23 07 107 167 107 227 87 24 25	17	15	17	16	16	17	17	17	16	14	15	15	16	15
11707172 17273 4075 07 07 6177 27178 9 10 17 11 17 12 13 97 17 14 87 37 15 16 17 17 17 18 17 157 19 177 20 1 22 23 24 67 57 17 17 17 18 17 157 19 17 17 18 17 157 19 177 20 1 22 23 24 67 57 17 17 17 18 17 157 19 17 17 18 17 157 19 17 17 20 1 12 23 24 67 57 17 17 17 18 17 157 18 17 157 19 17 17 18 17 157 19 17 17 20 1 12 23 24 17 15 27 15 17 17 17 18 17 157 19 17 17 20 1 12 23 24 27 50 17 27 18 19 10 11 12 17 13 14 15 16 17 18 19 10 17 20 17 21 19 97 97 57 22 27 97 17 23 24 17 17 25 17 17 27 18 17 17 18 18 18 17 17 18 18 18 17 17 18 18 18 18 18 18 18 18 17 17 18 18 18 18 18 18 18 18 18 18 18 18 18	0? 0? 1 2 0? 3 4 5 6 6? 5? 7 8 9 10 11 6? 5? 0? 12 0? 13 14 15 0? 14? 2? 16 17 13? 3? 13? 18 19 9? 0? 20 0? 6? 2? 0? 21 4? 0? 22 23 24 25 26 3?	14	15	15	15	15	16	15	14	15	17	17	15	16	17
1171723171741756577670787891011127713141516171819107201721197975722797172324177252173726197273177  12	1 1? 0? 2 3 4 1? 1? 2? 0? 5 6 7 1? 5? 8 9 10 11 8? 0? 6? 8? 12 9? 9? 13 10? 14 15 16 13? 10? 17 18 18? 19 20 21 22 23 24 25 26 27 28 9? 29 9? 1?	16	18	17	14	16	17	18	16	16	16	18	18	18	16
1 2 0 2 2 3 4 5 2 6 1 7 5 7 6 7 0 7 6 7 2 7 8 9 10 11 8 7 1 2 7 1 3 3 7 14 137 2 7 15 16 17 6 7 18 14 7 19 20 18 7 2 1 2 2 2 7 2 2 7 2 7 2 3 2 4 2 5 2 6 1 7 7 2 8 6 7	1 1? 0? 1? 2 1? 2? 3 4 0? 5 0? 0? 6 1? 7 2? 1? 8 9 10 1? 11 1? 12 13 9? 1? 14 8? 3? 15 16 1? 17 17? 18 1? 15? 19 17? 20 21 22 23 24 6? 5? 1? 17?	20	19	21	20	20	20	20	20	20	21	22	21	23	22
90 1 1 2 3 0 7 0 7 4 0 7 0 7 0 7 5 3 7 0 7 8 4 7 9 1 7 0 8 7 7 4 7 1 0 11 1 2 1 3 5 7 8 7 4 3 7 1 5 16 13 7 6 7 1 18 9 7 19 15 7 2 0 2 1 1 7 0 7 2 2 2 3 2 7 15 7 18 18 17 18 19 19 20 20 20 20 20 18 18 17 18 19 19 10 11 17 12 13 2 7 1 4 15 16 13 17 18 7 0 7 1 18 9 19 15 7 2 0 2 1 1 7 0 7 2 2 2 3 2 7 15 7 2 2 1 13 14 15 7 1 1 1 1 2 1 3 13 7 1 4 1 5 1 7 1 6 13 7 1 7 18 7 0 7 2 0 7 1 2 7 2 3 7 2 7 2 7 2 8 7 1 2 1 3 2 7 1 4 1 5 1 6 1 7 1 8 0 7 1 9 1 9 1 1 7 2 2 1 2 2 2 3 7 2 1 7 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2	1 1? 1? 2 3 1? 1? 4 1? 5 6 5? 7 6? 3? 8 9 10 11 12 7? 13 14 15 16 17 18 19 10? 20 1? 21 19? 9? 5? 22 2? 9? 1? 23 24 17? 25 21? 3? 26 19? 27 3? 1?	12	13	13	13	13	13	13	12	13	13	12	12	12	14
11   17   17   17   17   17   27   27	1 2 0? 2? 3 4 5 2? 6 1? 5? 7 6? 0? 6? 2? 8 9 10 11 8? 12 7? 13 3? 14 13? 2? 15 16 17 6? 18 14? 19 20 18? 21 22 2? 22? 9? 23 24 25 26 1? 27 28 6?	13	15	14	15	14	15	12	13	13	14	14	14	14	13
100   101   107	0?0?0?1230?0?40?0?0?563?0?784?91?0?8?9?4?101112135?8?143?151613?6?17189?1915?20211?0?222322?15?	19	17	20	18	17	18	19	19	20	20	20	18	19	18
10 1 2 3 4 5 6 7 8 9 8 7 4 7 10 11 12 12 7 13 1 4 0 7 1 5 6 7 16 16 7 17 18 6 7 19 07 137 20 207 47 21 87 17 22 37 20 7 24 25 227 26 7 28 87 17 29 30  10 1 2 3 3 4 5 6 7 8 9 8 7 4 7 10 11 12 12 7 13 1 4 10 7 16 27 16 12 7 18 10 7 14 07 15 27 1 6 27 5 1 7 18 7 7 19 20 21 6 3 7 3 9 7 22 22 7 8 7 23 24 25 26 27 5 7 28  11 1 33 11 13 13 13 13 12 11 10 10 10 10 10 10 10 10 10 10 10 10	0 ? 1 1 ? 0 ? 0 ? 0 ? 0 ? 2 ? 3 0 ? 4 5 0 ? 0 ? 0 ? 6 7 8 9 1 ? 5 ? 0 ? 1 1 12 13 13 ? 14 15 1 ? 16 13 ? 17 8 ? 0 ? 4 ? 18 19 0 ? 20 21 22 23 0 ? 1 ? 9 ? 24 25 13 ?	18	17	17	15	15	16	18	18	19	18	18	17	18	21
11 207 3 07 4 5 27 27 6 7 8 9 10 8 7 11 47 57 6 7 47 107 12 13 14 07 15 27 16 27 57 17 18 77 19 20 21 67 37 97 22 227 87 23 24 25 26 27 57 28  11 33 18 18 18 18 18 18 18 18 18 18 18 18 18	07 07 1 17 07 2 3 07 4 27 07 5 6 37 7 37 8 9 10 11 17 12 13 27 14 15 16 17 18 07 19 97 187 20 207 17 21 117 22 217 87 17 227 37 27 67 23 237 207 24	15	15	15	15	15	15	15	15	16	16	17	17	16	19
11	0? 1 2 3 4 5 6 7 8 9 8? 4? 10 11 12 1? 12? 13 14 0? 15 6? 16 16? 17 18 6? 19 0? 13? 20 20? 4? 21 8? 1? 22 3? 23 0? 24 25 22? 26 27 28 8? 1? 29 30	9	9	9	8	10	9	10	9	10	8	9	9	9	11
100 1 107 107 27 3 4 5 37 6 7 77 97 37 97 8 37 57 07 07 9 10 11 12 07 27 07 107 07 13 07 14 57 27 15 07 16 07 07 17 57 18 19 157 20 14 90 57 57 21 22  24 22 28 32 32 42 32 42 42 42 42 42 42 42 42 42 42 42 42 42	0 1 2 0? 3 0? 4 5 2? 2? 6 7 8 9 10 8? 11 4? 5? 6? 4? 10? 12 13 10? 14 0? 15 2? 16 2? 5? 17 18 7? 19 20 21 6? 3? 9? 22 22? 8? 23 24 25 26 27 5? 28	11	13	11	13	13	13	12	11	10	10	10	10	10	10
1 17 17 17 17 2 3 07 407 17 5 67 37 8 57 9 10 57 67 11 10 797 57 12 13 87 87 17 27 57 14 15 16 17 107 18 19 20 117 87 21 37 22 87 23 07 24 25  10 10 20 20 3 4 27 5 60 7 57 8 9 10 11 27 12 13 14 15 87 16 107 107 18 77 19 127 20 21 12 22 307 24 25 26 27 17 72 8 107 28 70	07 07 1 2 3 4 5 07 6 7 8 9 10 07 11 07 17 12 13 14 07 15 16 17 37 67 127 18 07 19 147 127 20 21 22 07 37 23 27 57 24 25 87 07 26 27 17 28 29 30	11	7	10	11	9	8	11	11	12	13	9	9	9	10
107 2 07 3 4 27 5 07 57 27 6 7 47 8 9 10 47 11 12 87 27 37 13 14 15 87 16 107 107 18 77 19 127 20 21 22 23 07 24 25 26 27 177 28 107 28 72 29 9 13 18 18 20 10 19 18 18 20 10 12 13 14 15 07 16 17 18 107 19 19 27 20 21 22 23 07 24 25 26 27 177 28 107 28 29 29 13 37 24 29 13 37 24 29 13 37 24 29 13 37 24 29 13 37 24 29 13 37 24 29 13 37 24 29 13 37 24 29 13 37 24 29 13 37 24 29 13 24 24 29 24 29 13 24 29 24	07 1 07 07 2 3 4 5 37 6 7 77 07 37 07 8 37 57 07 07 9 10 11 12 07 27 07 107 07 13 07 14 57 27 15 07 16 07 07 17 57 18 19 157 20 147 07 57 21 22	24	22	25	23	23	24	23	24	24	24	25	25	24	25
07 1 2 07 3 37 37 07 4 37 5 07 6 7 8 9 10 37 11 27 12 13 37 107 14 15 16 17 18 07 37 19 37 20 21 37 37 22 23 37 24 25 137 37 21 737 26 27 137 37    18	1 17 17 17 17 2 3 07 4 07 17 5 6 7 37 8 57 9 10 57 67 11 07 97 57 12 13 87 87 17 27 57 14 15 16 17 107 18 19 20 117 87 21 37 22 87 23 07 24 25	16	16	16	16	16	16	17	16	15	15	15	16	16	17
1 2 0 7 3 0 7 4 3 7 5 3 7 4 7 6 6 7 7 8 9 10 11 97 5 7 5 7 3 7 9 7 3 7 12 13 14 15 0 7 16 17 18 47 19 7 20 2 1 5 7 15 7 9 17 17 17 2 2 2 3 19 7 9 2 17 19 7 3 7 2 4 19 7 16 17 19 7 18 10 1 1 97 12 13 14 5 7 13 7 18 10 1 1 1 1 1 2 1 7 13 14 15 0 7 16 17 18 47 19 7 2 0 2 1 5 7 15 7 9 7 5 7 0 1 7 17 17 2 2 9 7 0 8 7 2 3 2 4 19 7 2 5 14 1	1072073427507572767478910471112872737131415871610710717071877191272021222307242526271772810728729	13	11	12	11	11	11	13	13	13	13	13	13	13	13
07 1 07 07 2 3 4 27 5 60 7 7 8 9 10 11 07 12 13 14 57 137 15 47 16 17 17 07 18 07 19 137 17 20 21 57 15 7 97 57 07 177 177 22 97 07 87 23 24 197 25  14	07   2 07 3 37 37 07 4 37 5 07 6 7 8 9 10 37 11 27 12 13 37 107 14 15 16 17 18 07 37 19 37 20 21 37 37 22 23 37 24 25 137 37 217 37 26 27 137 37	18	18	20	20	20	19	18	18	20	20	20	20	20	20
00000000000000000000000000000000000000	1 2 07 3 07 4 37 5 37 47 6 67 7 8 9 10 11 97 57 57 37 97 37 12 13 14 15 07 16 17 18 47 19 47 197 20 21 57 107 147 187 22 23 197 97 217 197 37 24 197	16	17	15	16	15	16	16	16	16	16	17	17	17	17
00000000000000000000000000000000000000	07 1 07 07 2 3 4 22 5 6 07 7 8 9 10 11 07 12 13 14 57 137 15 47 16 17 17 07 18 07 19 137 17 20 21 57 157 97 57 07 177 177 22 97 07 87 23 24 197 25	14	12	13	13	13	14	14	14	15	15	15	13	14	16
10721731737070727074507076707897787109707070711071213471413707371507171617071807190720770707			19	21	20	20	21	21	21	21	22	22	21	22	23
07 1 07 07 07 07 2 3 4 5 07 6 7 17 8 9 87 87 37 07 77 10 11 12 127 13 14 37 07 67 127 147 127 15 16 07 17 18 107 17 19 20 21 22 157 23 237 187 24 27 [19]			24	26	25	25	25	26	26	26	26	26	26	26	26
			17	19	20	20	19	18	19	17 _	21 _	18	18	18	18
	1 2 07 27 07 3 27 07 07 4 5 6 07 27 57 27 7 07 8 57 37 87 9 10 27 27 17 07 17 11 12 87 13 14 07 15 17 57 37 07 16 147 17 18 07 17 19 17 20 07	26	23	27	23	24	24	25	26	26	25	27	27	25	29
12 207 3 4 5 07 6 07 22 07 07 7 37 8 7 7 9 10 11 17 77 07 12 13 57 07 14 15 77 117 16 07 17 18 19 20 21 22 23 07 117 24 137 25 37 26 27 167 28 29			1	14	14	14	12	14	14	15	16	13	13	12	14

● 查了一下资料,Intel的CPU主要是用 Tree-PLRU策略替换,本身的序列和LRU也有所不同,大体上和LRU一致,这应该也能解释为什么拟合效果差一些……

### 测量Cache是否写直达

- 写直达策略在写命中和写不命中时的访问延迟相似
- 写回策略在写命中时的访问延迟明显小于写不命中

关键在于怎么控制写命中和写不命中,由上面的图我们可以很容易得到: MISS高与MISS低即为写命中和写不命中的平均水平



# 6.对本次实验的意见和建议。

无