计算机组织与体系结构实习报告 Lab3.1

学号：1500012863

姓名：杨程旭

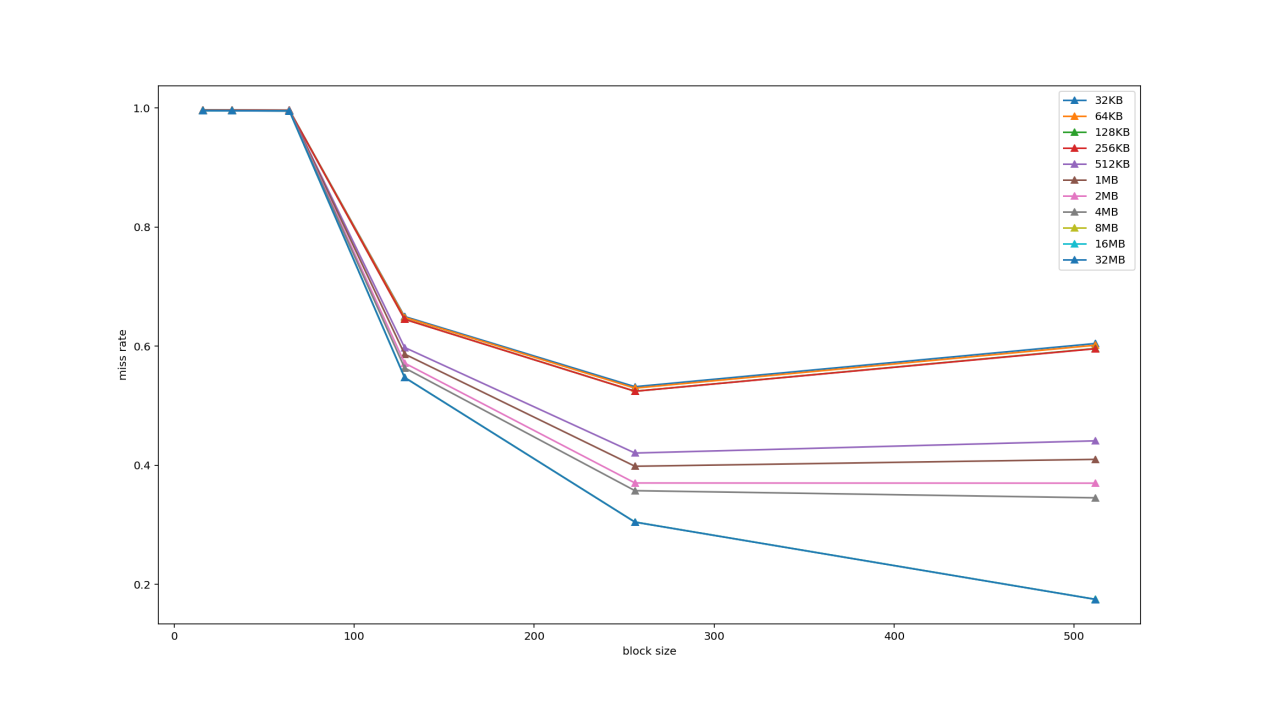
大班教师：程旭

PART 1、单层Cache模拟

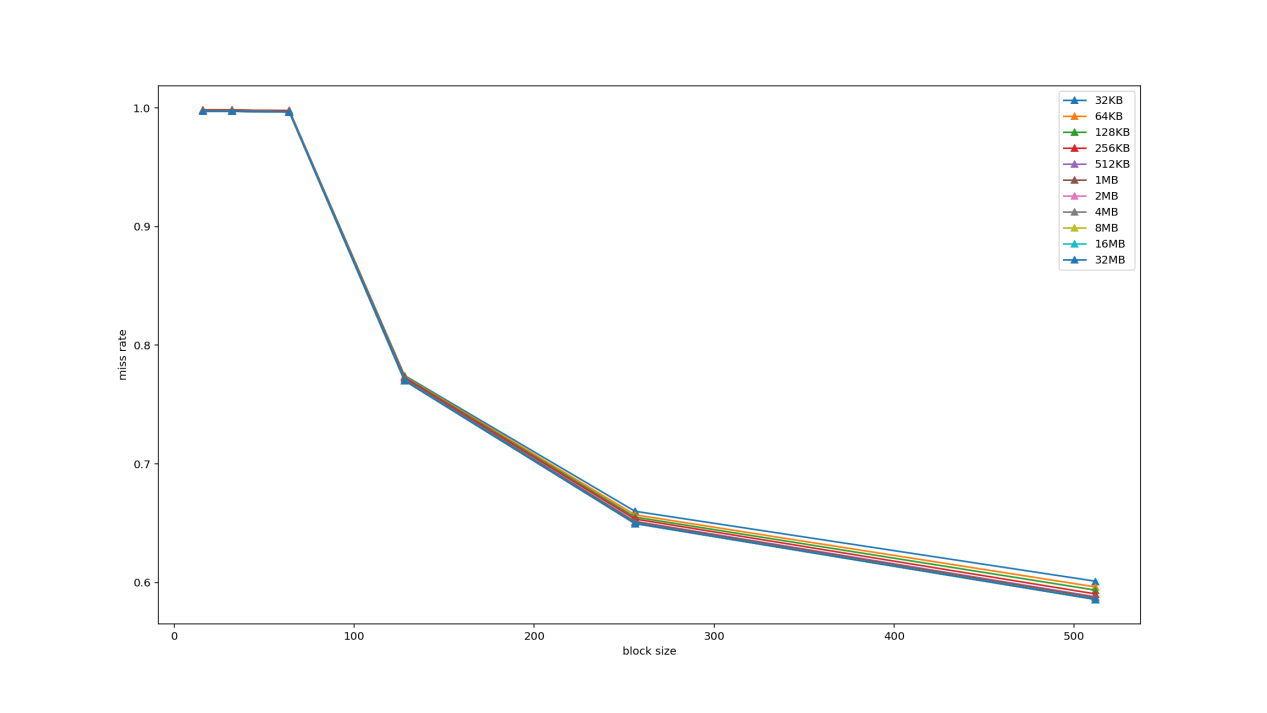
使用附件中所给模拟器框架，使用给定的测试trace，完成单层cache的模拟。要求：

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。（20分）

1.trace

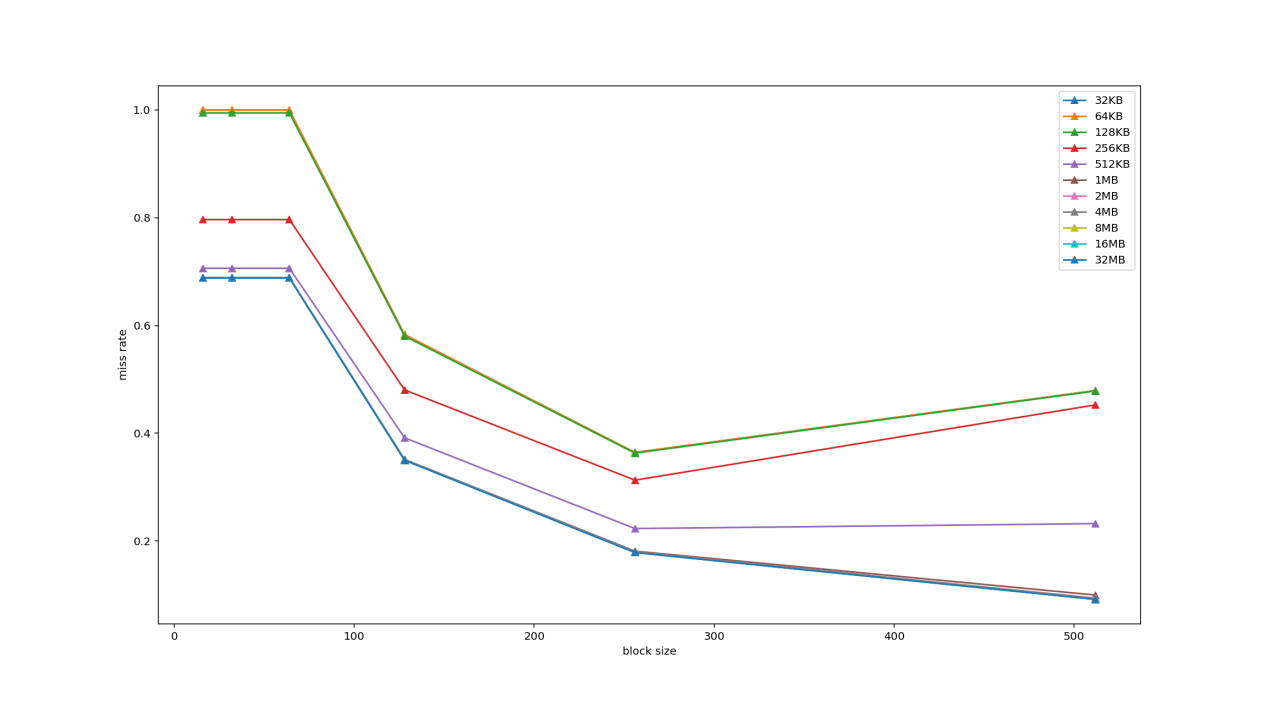


write back + write allocate

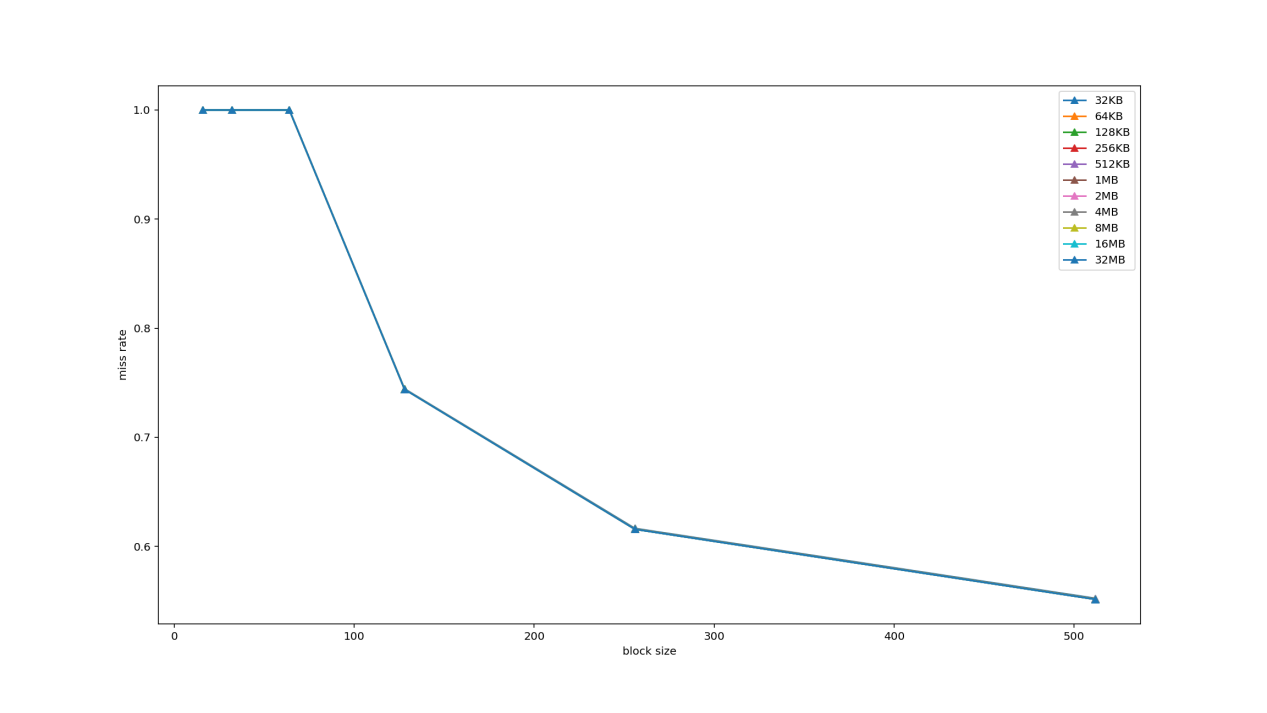


write through + no-write allocate

2.trace



write back + write allocate



write through + no-write allocate

分析：

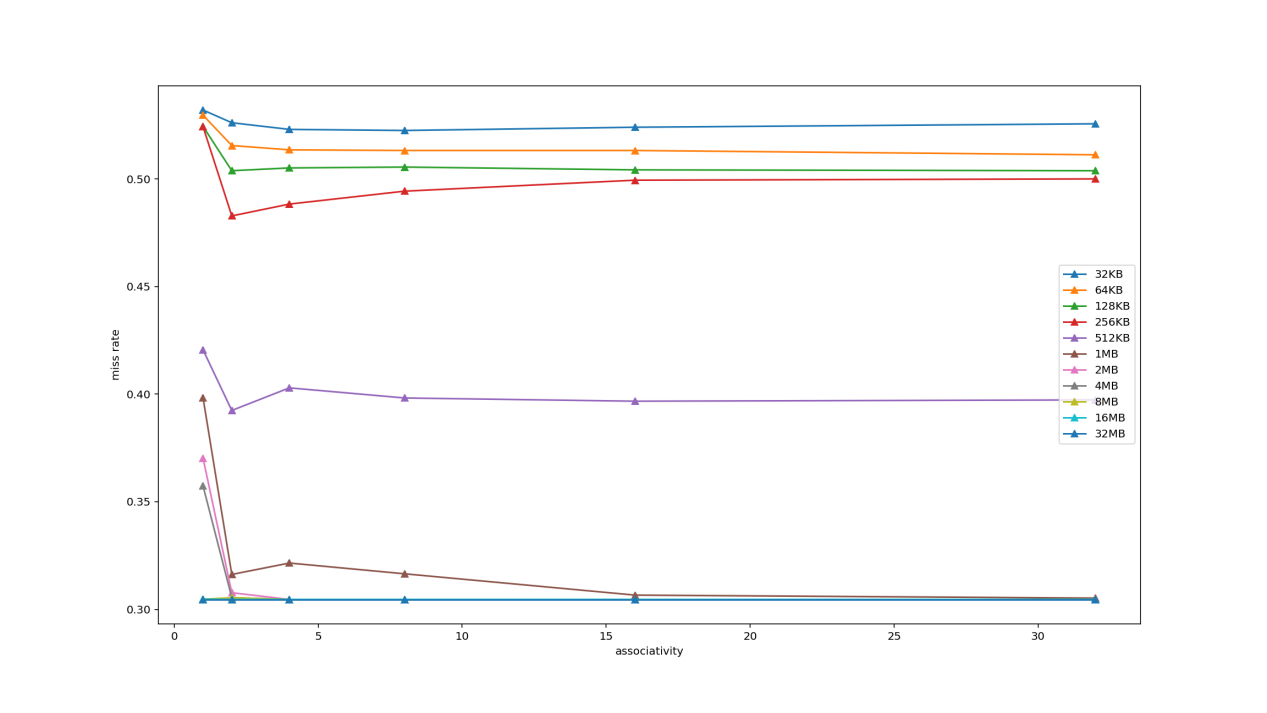
这一部分的测试相联度为1，cache size和block size变化。这一部分暂时不对写回和直写的策略进行分析（留在第3部分）。

从cache size的角度：可以看到，cache size较大的线总是在cache size较小的线的下方，这是很直观的，相同的block size下，cache size增加一定不会使miss rate上升。唯一例外的是第4张图，第4张图表示的是2.trace,使用直写加写不分配的策略，在不同的cache size下miss rate随block size的变化。可以看到，所有的线几乎重合在了一起，也就是说cache size影响非常小。这是因为，2.trace的数据非常的紧凑，在block size和相联度固定的情况下，cache size影响的是set number，从而只影响地址set index长度（位宽），在数据非常紧凑的情况下，几乎不产生影响。可以看到，第二张图和第四张图也非常的相似，几条线几乎重合，也是因为同样的原因。

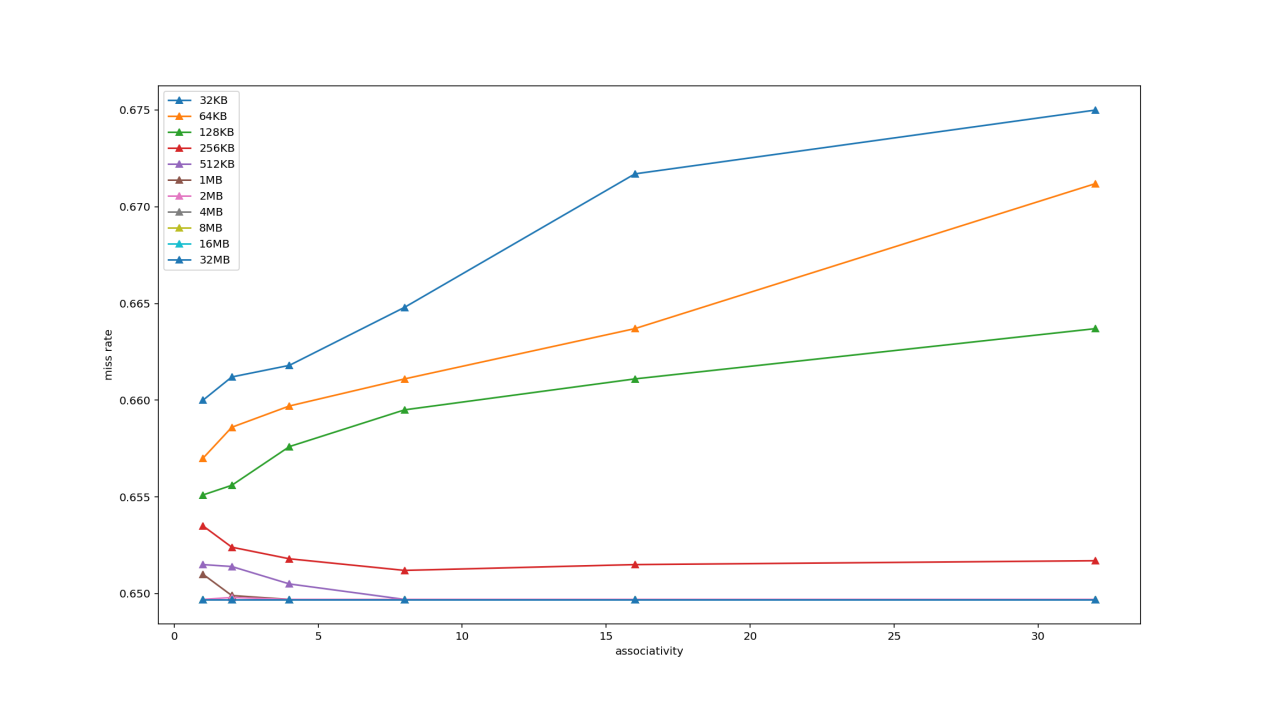
从block size的角度：固定cache size，当block size很小的时候（小于等于64k），miss rate很高，这是因为相联度为1，block size很小，很容易产生冲突不命中。随着block size的增长，miss rate总体是下降的趋势。但这种下降只是在一定范围内的，因为cache size固定的情况下，block size的增大意味着set number的减少，总有一个miss rate最低的点，这个点并不一定是block size最大的点，这也解释了为什么第一张图的尾部略有上升。

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。（20分）

1.trace

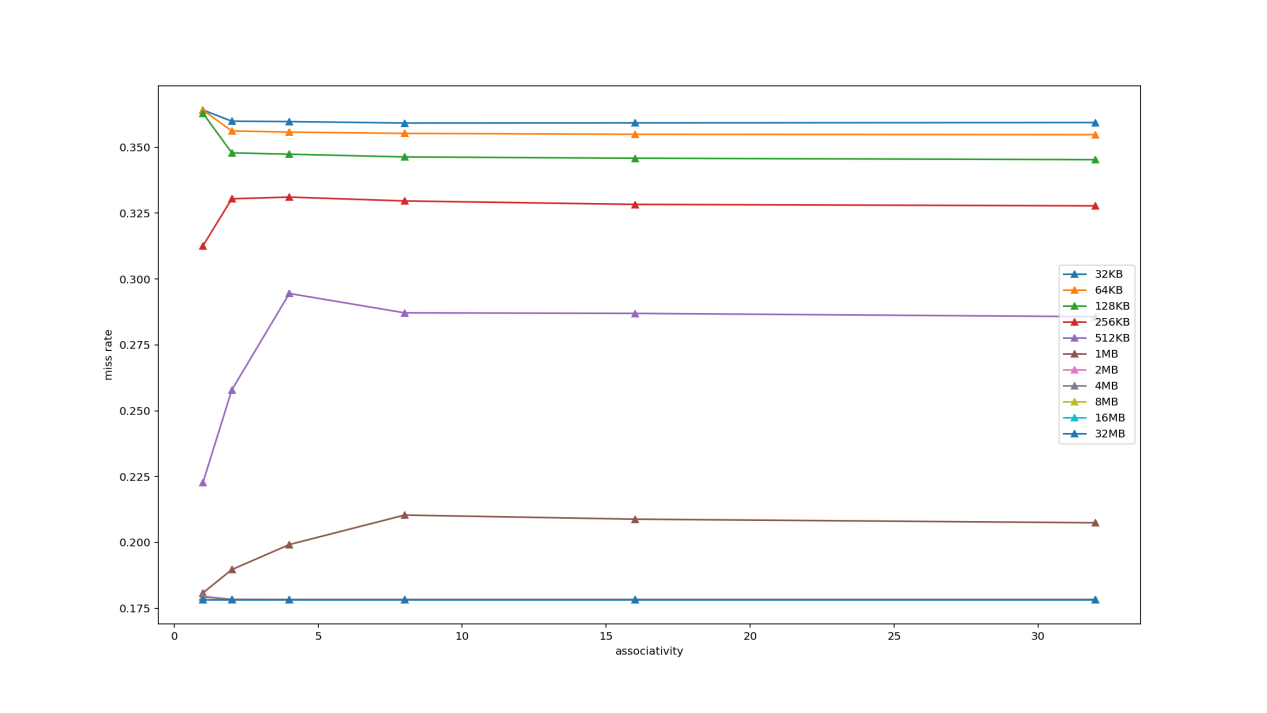


write back + write allocate

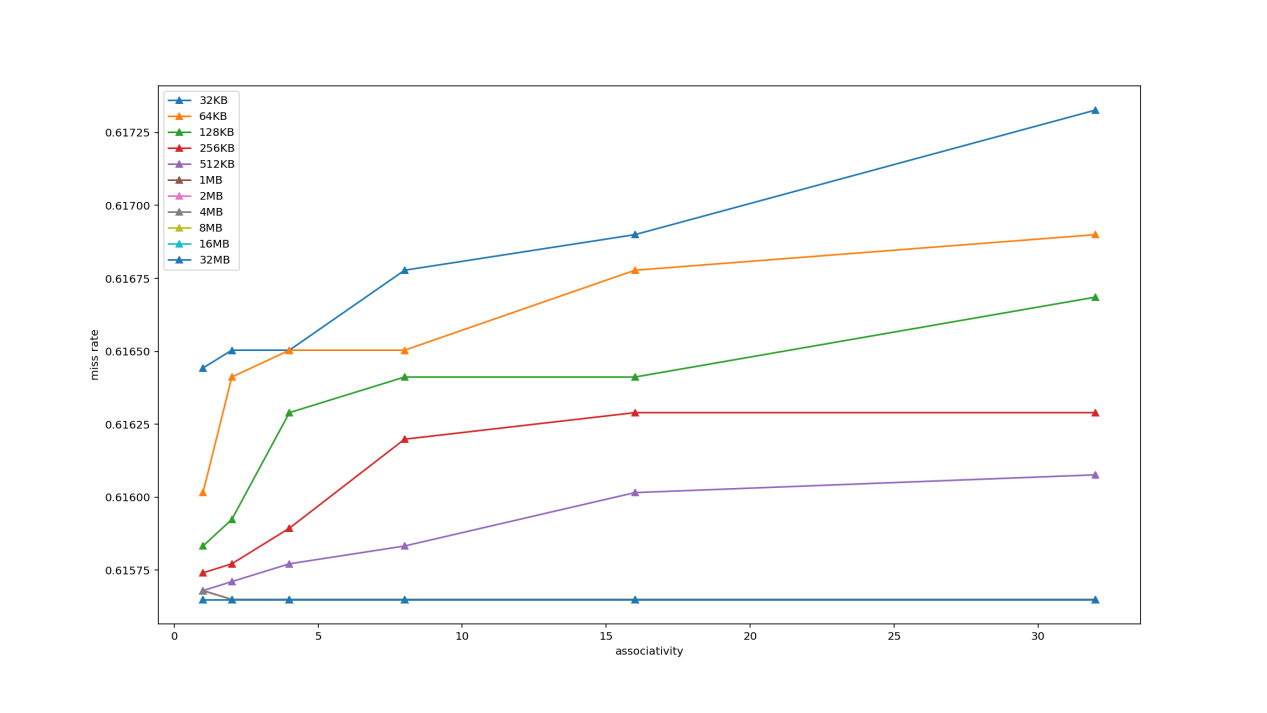


write through + no-write allocate

1. Trace



write back + write allocate



write through + no-write allocate

分析：

这一部分block size固定为256B，在不同的cache size下，miss rate随Associativity变化的趋势。

从cache size的角度：这一角度其实和上一部分的cache size的角度相同，分析是完全相同的，同时，从图2、图4也可以看出miss rate其实并不是完全重合，只是非常的接近，因为纵坐标的间隔非常的小，差距在小数点后两位。

从Associativity的角度：

1. 比较 Write Through 和 Write Back、 Write Allocate 和 No-write allocate 的总访问延时的差异。（10分）

|  |  |  |
| --- | --- | --- |
| 文件 | 写策略 | 延时平均值 |
| 1.trace | 写回+写分配 | 870784.3 |
| 直写+写不分配 | 866582.1 |
| 2.trace | 写回+写分配 | 2427228.2 |
| 直写+写不分配 | 2782870.2 |

分析：

写回+写分配（下称策略1）和直写+写不分配（下称策略2）在不同的情况下各有利弊，考虑如下不同情况：

情况1、连续写2次同一块地方，使用策略1的话第一次高速缓存会miss，第二次会命中miss rate为0.5，并且少了一次对内存的访问，使用策略2的话，两次都会miss，在这种情况下，是策略1优于策略2。

情况2、第一次写一个块，两个策略都会miss，策略1会在高速缓存缓存该块并设置dirty位，而策略2并不会，下一次要读另一个与该块冲突的块，策略1下先要把缓存的块写回到内存，然后再从内存读需要的块，这种情况下策略1会多一次内存的访问，策略2优于策略1。

PART 2. 与lab2中的处理器性能模拟器联调。

Lab 2中的流水线模拟器联调，运行测试程序。

1. 测试程序自选，测试结果正确，并打印动态执行的指令数和CPI。（40分）

- 能够与流水线模拟器协同工作。（10分）

- 能够模拟多层次cache。（10分）

- 测试程序运行正确，并打印出动态执行指令数，以及相应的CPI。（4分\*5）

|  |  |  |  |
| --- | --- | --- | --- |
| 测试程序 | 动态指令数 | 周期数 | CPI |
| Test1 | 29 | 423 | 14.5862 |
| Test2 | 29 | 423 | 14.5862 |
| Test3 | 118 | 406 | 3.4407 |
| Test4 | 163 | 717 | 4.3988 |
| Test5 | 119 | 407 | 3.4202 |
| Test6 | 69 | 347 | 5.0290 |
| Test7 | 133 | 554 | 4.1654 |
| Test8 | 61 | 459 | 7.5246 |
| Test9 | 34 | 298 | 8.7647 |
| Test10 | 24 | 288 | 12.0000 |

1. 对比lab2中的流水线模拟器，分析CPI的变化原因。（10分）

可以看到，当加入cache模拟后，CPI有了非常明显的提升，提升的原因主要是在访存阶段并不一定能在一个周期内取到数据。除此之外，还有一点需要注意的是，具有较好的空间局部性的程序（test3、5等带有循环的测试程序）CPI都相对较低，这是因为cache的作用在局部性较好的情况下较大，正所谓“又是局部性拯救了我们”。（出自ics课本）