

（深圳）

实验报告

开课学期： 2021秋季

课程名称： 计算机体系结构(实验)

实验名称： Cache设计

实验性质： 设计探索型

实验时间： 8 地点：

学生班级： 1901105

学生学号： 190110509

学生姓名： 王铭

评阅教师：

报告成绩：

实验与创新实践教育中心印制

2021年11月

# 1. 实验内容

本实验将基于Pin插桩分析的工作机理，使用C/C++实现三种映射方式的Cache软件模型，从而在进一步熟悉和掌握插桩工具使用的同时，加深对Cache基本结构和工作原理的理解。

  本实验的实验内容包括：

（1）利用Pin插桩工具，分别编写全相联、直接相联和组相联映射的Cache模型；

  （2）探讨命中率与Cache基本参数（如块数、块大小、相联度、替换算法等）的关系，绘制折线图并分析之；

  （3）通过测试，对比分析三种映射方式的区别和联系。

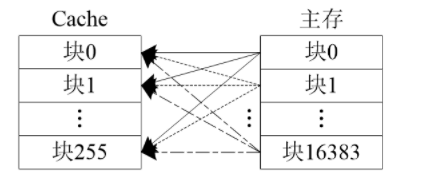
# 2. 设计与实现

## 2.1 题目分析

首先分别实现三种Cache：全相联Cache、直接相连Cache和组相联Cache即可。

**1.全相联Cache:**

如图所示，主存中的每一块都可以映射到Cache中任意一块。



可将地址分解为：

标记（Tag） | 块内地址

每次要访问主存时，先遍历Cache，查看要访问地址的Tag是否与Cache中的块标记位匹配，若匹配则命中，否则未命中，可以用LRU算法，选择Cache的一块，从主存中调入并更新Cache的Tag位。

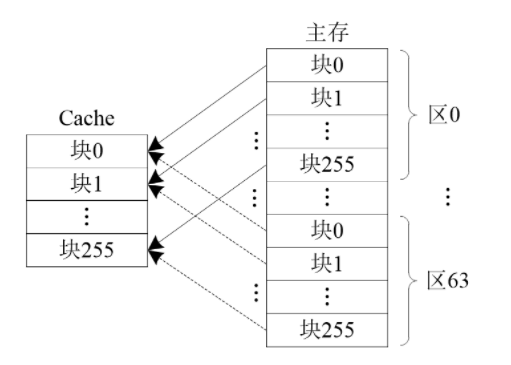
LRU算法实现：

由于是Cache采用的是全相联，主存的每一块都能映射到Cache的任意一块，故维护一个队列结构，该队列按照最近一次访问时间递减的顺序组织，队列头（最近一次访问时间距当前时间最长的块）为下一个要替换的块。

每次访问，若命中，则将当前队列中该块移动到队列尾部。若未命中，则取队列头进行替换，修改该块的Tag位即可。

**2.直接相连Cache:**

如图所示，将主存划分为若干个区，每个区所包含的块数与Cache块数相同，并且每个区中的第i块只能映射到Cache中的第i块。



因此可以将地址分解为：

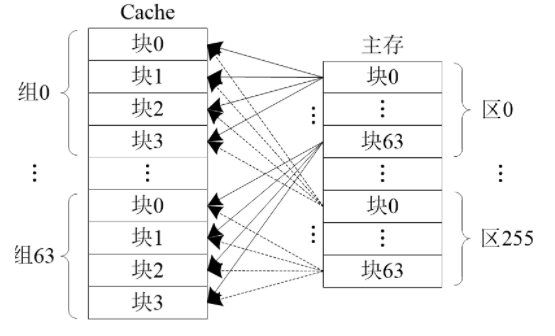
Tag | Cache索引 | 块内地址

每次要访问主存时，直接根据地址分解后的Cache索引访问对应的Cache块，查看要访问地址的Tag是否与Cache中的块标记位匹配，若匹配则命中，否则未命中。由于使用的是直接映射Cache，主存的每一块在Cache块中对应块是固定的，命中时无需更新，未命中时需要把Cache对应块的Tag修改为当前地址的Tag即可。

**3.组相联Cache:**

如图所示，主存被划分成若干个区，Cache被划分成若干个组，并且主存中每个区

包含数据块个数等于Cache的组数；Cache组间采用直接映射，组内采用全相联映射。



因此可以将地址分解为：

Tag | Cache索引 | 块内地址

每次要访问主存时，直接根据地址分解后的Cache索引访问对应的Cache组，由于采用的组相联，组内是全相联的，因而需要遍历该组，查看要访问地址的Tag是否与该组中的某一块标记位匹配，若匹配则命中，否则未命中。类似全相联映射，采用LRU算法维护一个最近访问时间由大到小的队列，不同的是，在Cache的每个组中均维护一个这样的队列。在Cache命中时，将当前Cache组内命中的块从队列中移动到队列尾部，在Cache未命中时，选择该组的队列头所对应的Cache块换入新的Tag即可。

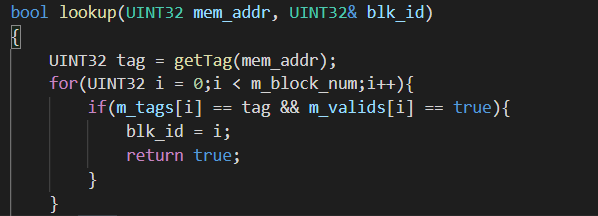
## 2.2 实验过程

**1.全相联Cache的代码实现**

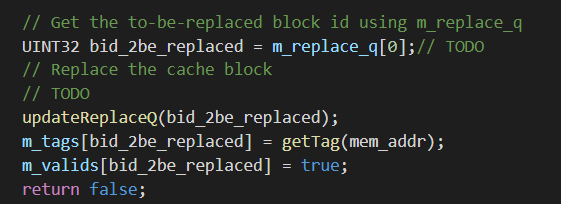
将上述分析转化为代码即可：

全相联根据传入地址获取tag，只需要让传入地址右移m\_blksz\_log位即可。其中m\_blksz\_log是块大小取对数。

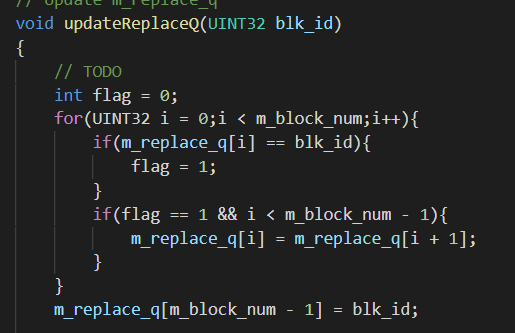
查询是否命中的函数，因为是全相联，需用for循环遍历，依次比较Tag位，命中则返回该块的块号：



最后是更新的实现，未命中时，取队列头作为要覆盖的块，将其Tag更换同时有效位置true即可：



更新策略，采用LRU算法即可，每次找到传入的blk号，然后将该块在队列的位置移动到队列末尾，维护好该队列即可：



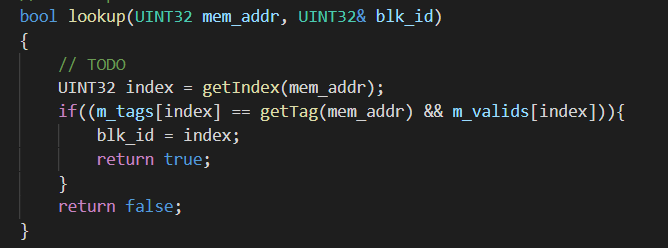
**2.直接相连Cache的代码实现**

与全相联类似，分解地址时，取tag只需让传入的地址右移(x + m\_blksz\_log)位，其中m\_blksz\_log是块大小取对数，x是cache块数取对数。

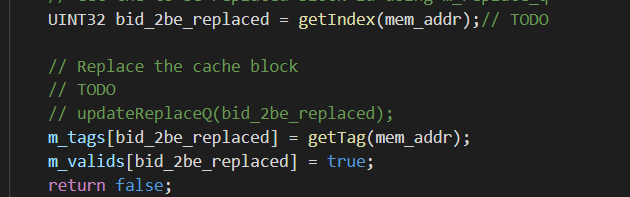
取Cache索引则需让传入的地址先右移m\_blksz\_log位，再与(1 << x – 1)与即可。

查询时，直接根据索引去查询是否与Cache块中保存的Tag匹配即可。直接相连无需维护队列，每次未命中直接覆盖其索引指向的Cache块即可。

其中查询代码如下：

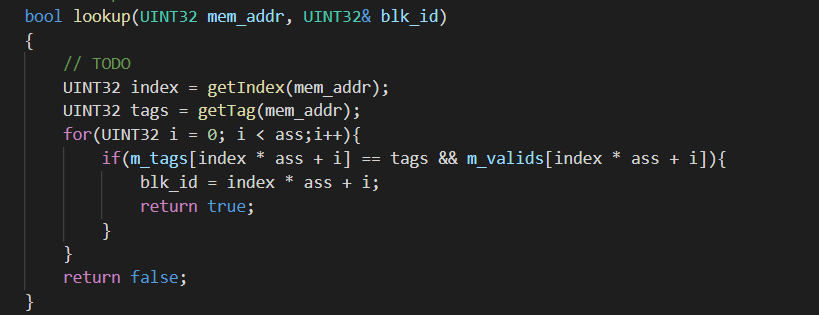


未命中时，根据索引更新：

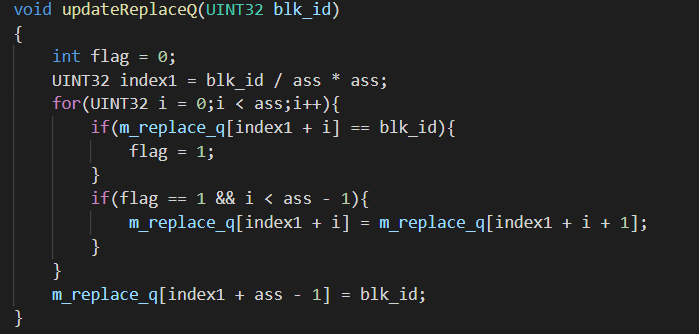


**3.组相联Cache的代码实现**

取Tag和Index的操作以及查询的操作与直接映射相似，其中查询操作需要先找到组索引，再在组内顺序查找。代码如下，其中ass是相联度：



组相联Cache需要维护队列，维护过程与全相联类似，但不同于全相联的是每个Cache组维护一个队列，代码如下，其中index1为该组队列的队列头：



更新操作类似全相联，未命中时取队列头更新即可，可参考全相联的更新实现。

## 2.3 实验结果及分析

**2.3.1基本测试**

默认Cache的容量为32KB，块数为512，块大小为64B。其中，组相联Cache默认为四路组相联，分别测试实验指导书中的四个测试集，所得测试结果如下表所示：

（1）gcc

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 平均读时间(us) | 平均写时间(us) | 读命中率 | 写命中率 |
| 全相联映射 | 1.50 | 2.94 | 98.34% | 99.53% |
| 直接映射 | 0.97 | 1.90 | 94.88% | 97.46% |
| 组相联映射 | 0.99 | 1.93 | 97.85% | 99.35% |

（2）astar

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 平均读时间(us) | 平均写时间(us) | 读命中率 | 写命中率 |
| 全相联映射 | 1.50 | 2.92 | 98.11% | 99.49% |
| 直接映射 | 0.98 | 1.92 | 95.54% | 98.40% |
| 组相联映射 | 0.99 | 1.93 | 97.80% | 99.33% |

（3）zeusmp

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 平均读时间(us) | 平均写时间(us) | 读命中率 | 写命中率 |
| 全相联映射 | 1.46 | 2.60 | 98.33% | 99.51% |
| 直接映射 | 0.91 | 1.63 | 95.12% | 98.44% |
| 组相联映射 | 0.94 | 1.68 | 97.69% | 99.30% |

（4）tonto

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 平均读时间(us) | 平均写时间(us) | 读命中率 | 写命中率 |
| 全相联映射 | 1.46 | 2.73 | 98.19% | 99.48% |
| 直接映射 | 0.95 | 1.85 | 95.74% | 98.35% |
| 组相联映射 | 0.96 | 1.86 | 97.79% | 99.33% |

**2.3.2探究基本参数对命中率的影响**

**①Cache块数对命中率的影响**

块大小均为64B，相联度为4

(1)gcc

|  |  |  |
| --- | --- | --- |
| Cache块数 | 读命中率 | 写命中率 |
| 512 | 97.85% | 99.35% |
| 256 | 96.10% | 98.86% |
| 128 | 93.23% | 97.87% |

(2)astar

|  |  |  |
| --- | --- | --- |
| Cache块数 | 读命中率 | 写命中率 |
| 512 | 97.80% | 99.33% |
| 256 | 95.98% | 98.90% |
| 128 | 92.99% | 97.87% |

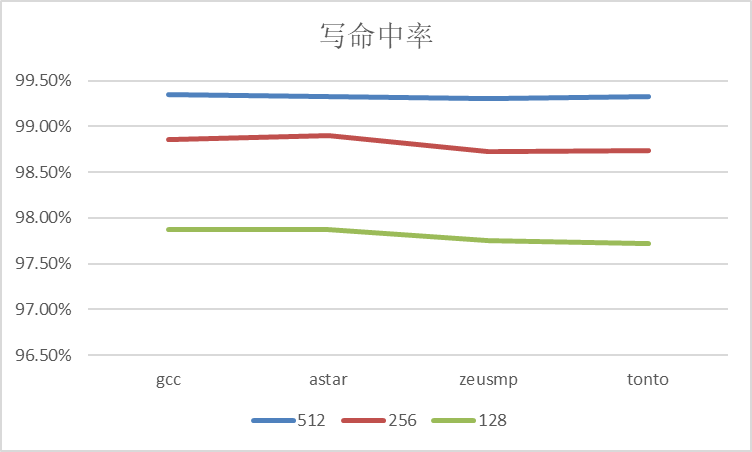
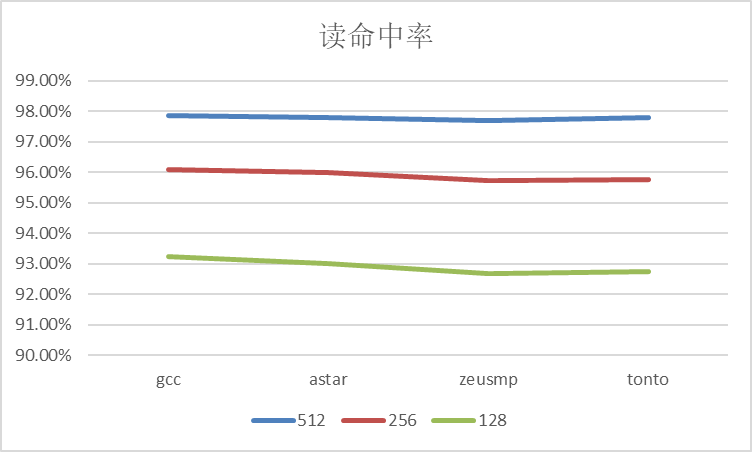
(3)zeusmp

|  |  |  |
| --- | --- | --- |
| Cache块数 | 读命中率 | 写命中率 |
| 512 | 97.69% | 99.30% |
| 256 | 95.73% | 98.73% |
| 128 | 92.67% | 97.75% |

(4)tonto

|  |  |  |
| --- | --- | --- |
| Cache块数 | 读命中率 | 写命中率 |
| 512 | 97.79% | 99.33% |
| 256 | 95.77% | 98.74% |
| 128 | 92.73% | 97.72% |

绘制折线图如下：



结论：由折线图可以看出，无论是读命中率还是写命中率，在其他参数相同的情况下，命中率均随着块数的增加而增加。

**②Cache块大小对命中率的影响**

块数均为512，相联度为4

(1)gcc

|  |  |  |
| --- | --- | --- |
| Cache块大小(B) | 读命中率 | 写命中率 |
| 64 | 97.85% | 99.35% |
| 32 | 96.00% | 98.48% |
| 16 | 93.28% | 96.87% |

(2)astar

|  |  |  |
| --- | --- | --- |
| Cache块大小(B) | 读命中率 | 写命中率 |
| 64 | 97.80% | 99.33% |
| 32 | 95.91% | 98.51% |
| 16 | 93.12% | 96.87% |

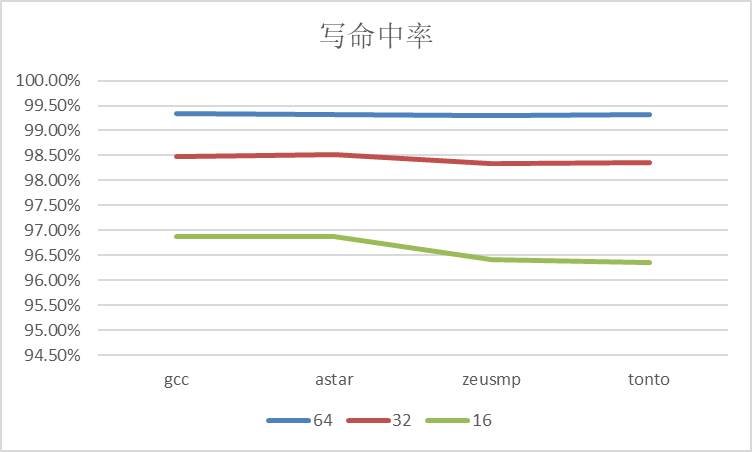
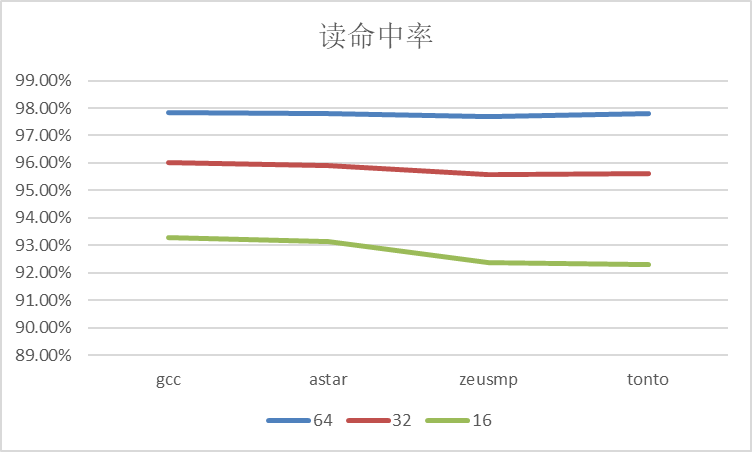
(3)zeusmp

|  |  |  |
| --- | --- | --- |
| Cache块大小(B) | 读命中率 | 写命中率 |
| 64 | 97.69% | 99.30% |
| 32 | 95.57% | 98.33% |
| 16 | 92.37% | 96.41% |

(4)tonto

|  |  |  |
| --- | --- | --- |
| Cache块大小(B) | 读命中率 | 写命中率 |
| 64 | 97.79% | 99.33% |
| 32 | 95.61% | 98.35% |
| 16 | 92.30% | 96.36% |

绘制折线图如下：



结论：由折线图可以看出，无论是读命中率还是写命中率，在其他参数相同的情况下，命中率均随着块大小的增加而增大

**③相联度对命中率的影响**

块数为512，块大小为64B

（1）gcc

|  |  |  |
| --- | --- | --- |
| 相联度 | 读命中率 | 写命中率 |
| 4 | 97.85% | 99.35% |
| 8 | 97.99% | 99.42% |
| 2 | 97.41% | 99.12% |

（2）astar

|  |  |  |
| --- | --- | --- |
| 相联度 | 读命中率 | 写命中率 |
| 4 | 97.80% | 99.33% |
| 8 | 97.90% | 99.40% |
| 2 | 97.23% | 99.21% |

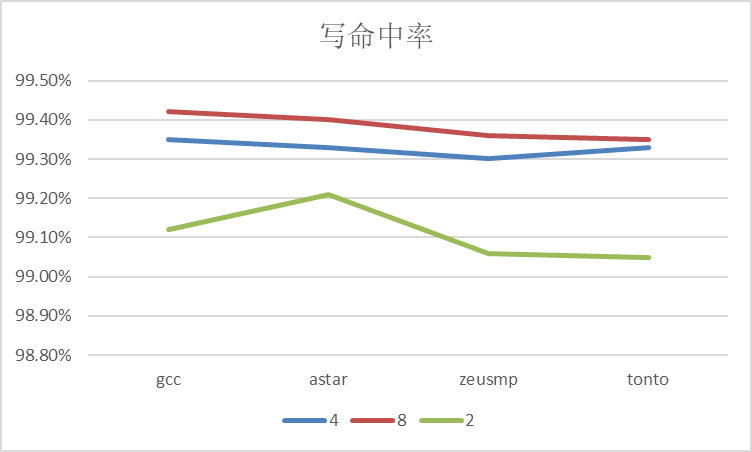
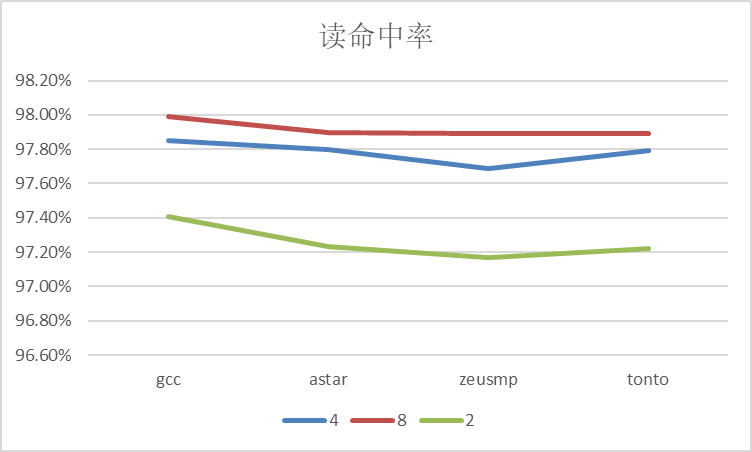
（3）zeusmp

|  |  |  |
| --- | --- | --- |
| 相联度 | 读命中率 | 写命中率 |
| 4 | 97.69% | 99.30% |
| 8 | 97.89% | 99.36% |
| 2 | 97.17% | 99.06% |

（4）tonto

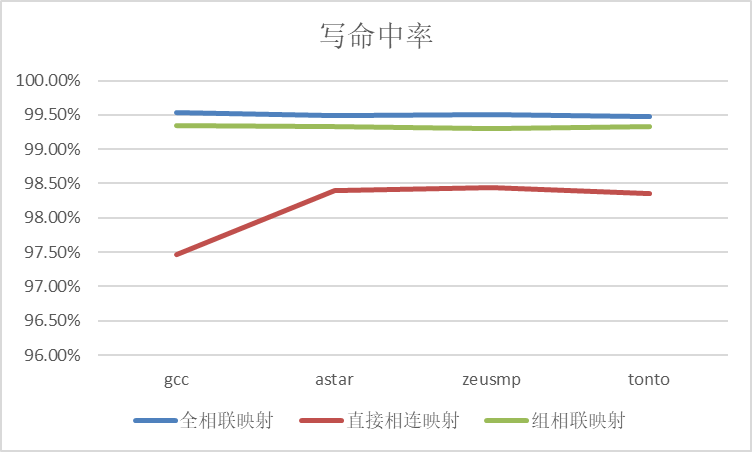
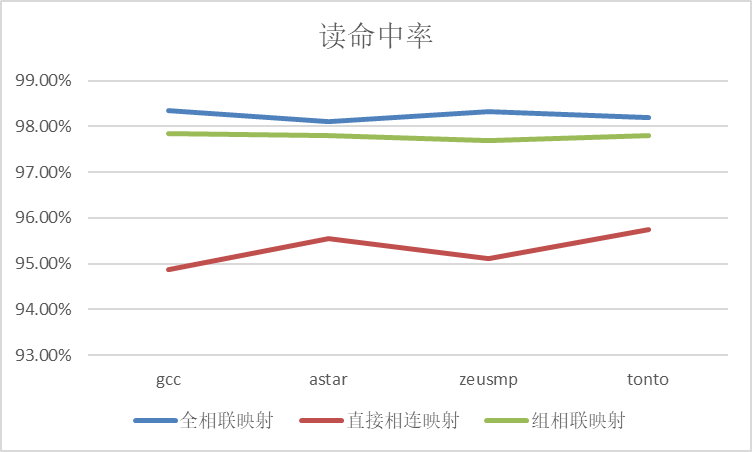
|  |  |  |
| --- | --- | --- |
| 相联度 | 读命中率 | 写命中率 |
| 4 | 97.79% | 99.33% |
| 8 | 97.89% | 99.35% |
| 2 | 97.22% | 99.05% |

绘制折线图如下：



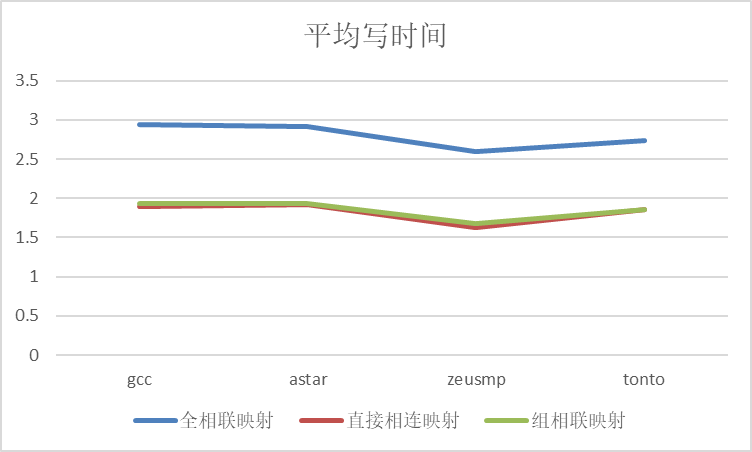
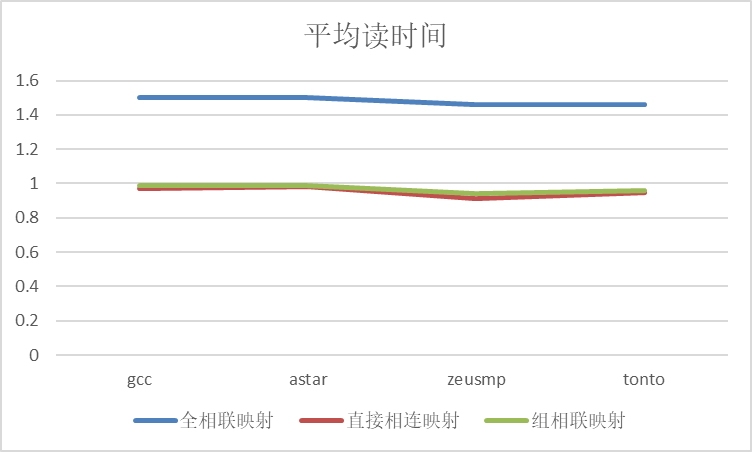
**2.3.3映射方式对比分析**

以2.3.1中记录的基础数据，在相同参数下，分析三种Cache模型运行上述4个benchmark的命中率、的读、写命中率的数据绘制成折线图如图所示：



结论：由图可以看出，无论是读命中率还是写命中率，其他条件相同的情况下，命中率从大到小排序为：全相联映射>组相联映射>直接映射

同样地，可以将平均读、写时间绘制成折线图如下：



结论：由图可以看出，无论是平均读时间还是平均写时间，其他条件相同的情况下，平均读写时间从大到小排序为：全相联映射>组相联映射>直接映射。

接着，将组相联cache的相联度提升为8，可以得到：

（1）gcc

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 平均读时间(us) | 平均写时间(us) | 读命中率 | 写命中率 |
| 全相联映射 | 1.38 | 2.48 | 98.34% | 99.53% |
| 直接映射 | 0.94 | 1.69 | 95.08% | 98.27% |
| 组相联映射 | 0.95 | 1.72 | 97.99% | 99.40% |

（2）astar

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 平均读时间(us) | 平均写时间(us) | 读命中率 | 写命中率 |
| 全相联映射 | 1.42 | 2.52 | 98.15% | 99.50% |
| 直接映射 | 0.96 | 1.69 | 94.91% | 97.98% |
| 组相联映射 | 0.98 | 1.73 | 97.93% | 99.42% |

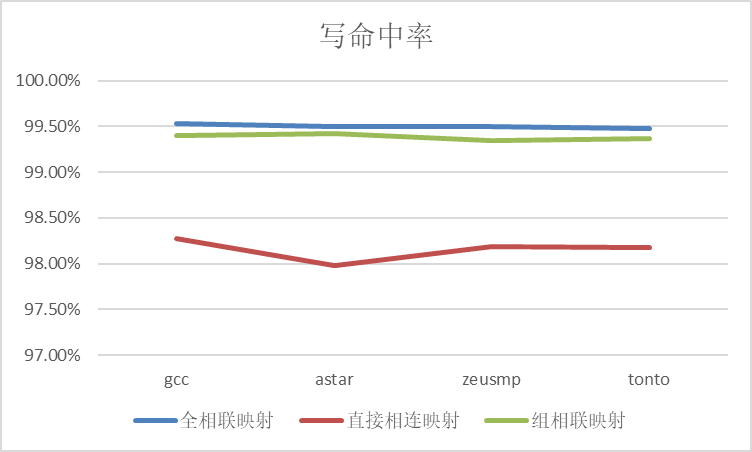
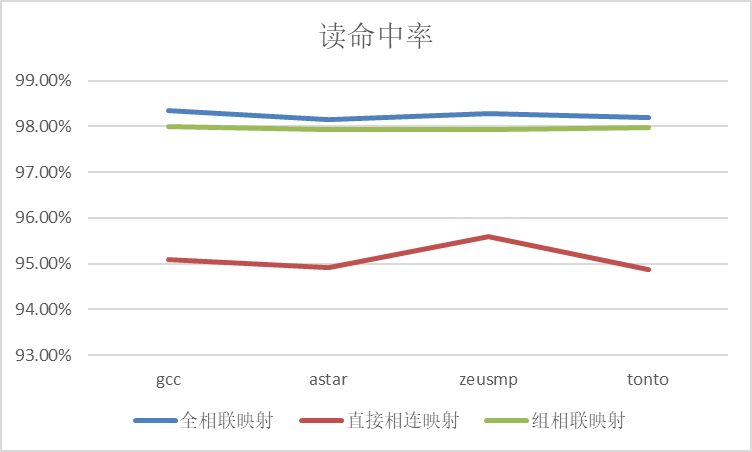
（3）zeusmp

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 平均读时间(us) | 平均写时间(us) | 读命中率 | 写命中率 |
| 全相联映射 | 1.47 | 2.59 | 98.29% | 99.50% |
| 直接映射 | 0.90 | 1.60 | 95.60% | 98.19% |
| 组相联映射 | 0.98 | 1.73 | 97.92% | 99.35% |

（4）tonto

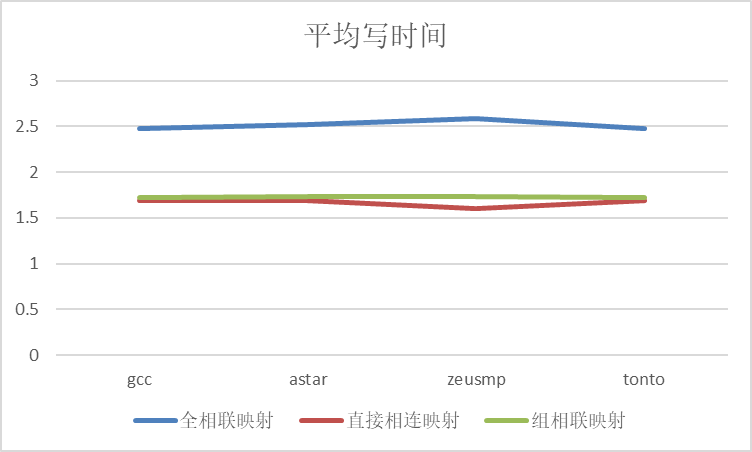
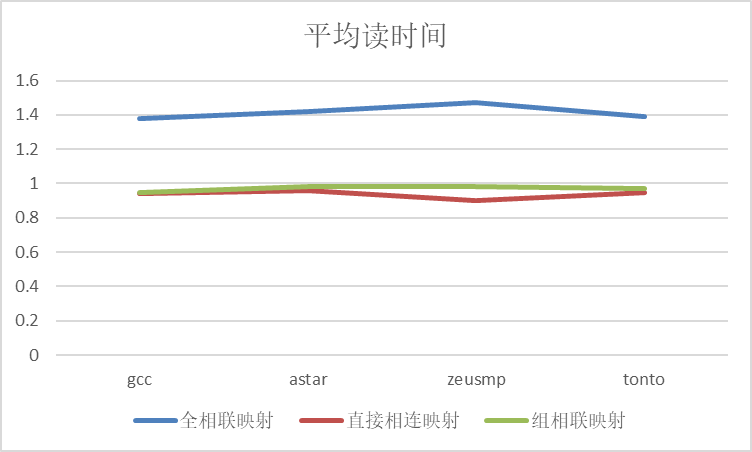
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 平均读时间(us) | 平均写时间(us) | 读命中率 | 写命中率 |
| 全相联映射 | 1.39 | 2.48 | 98.19% | 99.48% |
| 直接映射 | 0.95 | 1.69 | 94.87% | 98.18% |
| 组相联映射 | 0.97 | 1.72 | 97.97% | 99.37% |

画出读、写命中率的折线图如下：



由图可得，无论是读命中率还是写命中率，其他条件相同的情况下，命中率从大到小排序为：全相联映射>组相联映射>直接映射，且全相联和组相联区分度比相联度为4要高。

类似地，画出平均读、写时间的折线图：



可以发现，其他条件相同的情况下，提高相联度后，读写时间的次序排序仍为全相联映射>组相联映射>直接映射。且直接映射和组相联之间的区分度比相联度为4更好。

由上述分析可以得出结论：其他条件相同的情况下，三种映射的读、写命中率排序以及平均读、写时间大小排序均为：全相联映射>组相联映射>直接映射。

# 3. 总结和感想

通过本次实验，明确了三种映射方式Cache各自的特点，进行了性能的比较、分析，更好地理解了Cache的原理、性能以及作用，加深了对理论课上所学知识的理解。但本次实验测试较慢，导致效率较低。