实验三：Cache设计

1. 题目分析

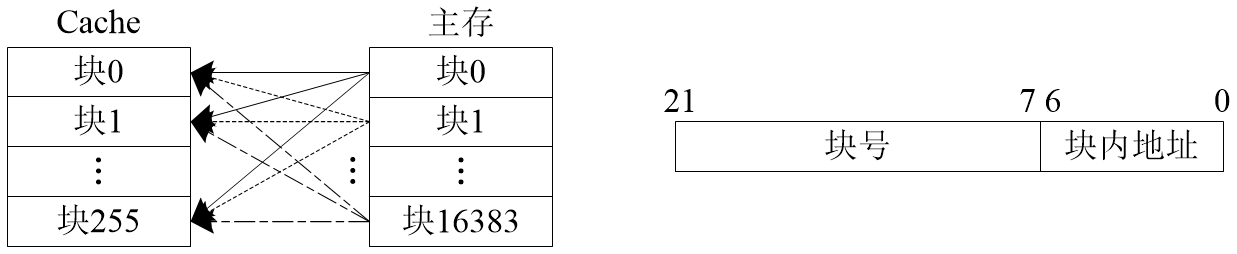
本次实验要求模拟实现Cache的设计运行，分别实现全相联、直接映射和组相联三种Cache，并运行插桩分析和运行程序测试效果，根据读写的命中率来衡量；

阅读程序，插桩代码是通过模拟cache来运行的，即当测试程序有读写请求时插桩代码会记录读写次数，通过插桩代码里实现的cache结构进行模拟cache访问，并记录命中的次数，最终输出命中率；

1. 设计思路
2. 全相联

首先是标志位的获取，全相联cache的地址是除去块偏移之后的位，因此直接将地址右移m\_blk\_log即可；

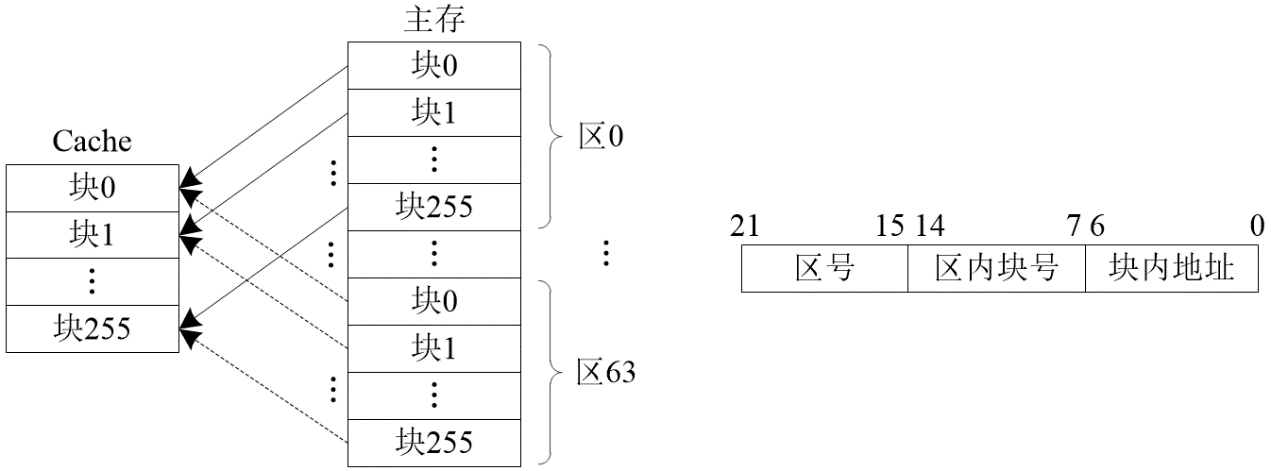
全相联是内存中的所有块都可以映射到cache中的任何一个块，也就是说访问cache时，需要遍历整个cache的所有块，寻找与访问地址标志位相同且有效的块，找到即为命中，否则不命中，执行LRU更换策略；



1. 直接映射

首先是标志位和块索引，直接映射时，主存每一个块映射到cache的一个块，cache的一个块映射到主存的多个块，因此需要解析出块号和区号，区号就是标志tag；

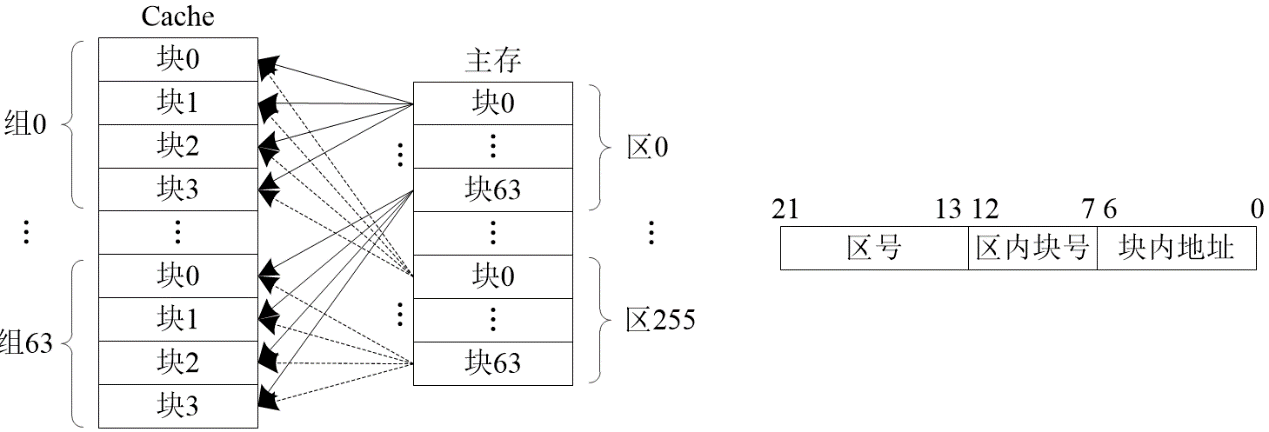
实现时，先解析地址，得到标志tag和块号，去cache寻找对应的块即可，查询是否标志相等且有效，找到则命中，否则不命中，直接映射没有更换策略，不命中时直接修改这个块的tag位和有效位即可；



1. 组相联

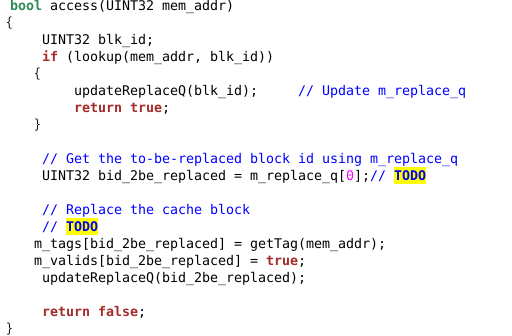
组相联解析出的区内块号是组索引，区号是标志tag；

实现时需要组索引来找到对应的组，然后遍历这个组的所有块，查找标志相同且有效的块，找到则命中，否则不命中，执行LRU更新策略；

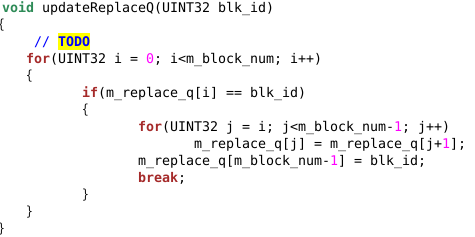


1. 关键实现
2. 全相联

首先时access函数，先调用lookup判断是否命中，命中则更新替换队列，否则不命中，取出替换队列第一个指向的块，修改标志位和有效位并函数返回false



更新替换队列函数采用LRU策略，即队列第一个元素是最不长访问的块，需要首先替换，队列最后是最近访问过的块，因此对于传入的id，其可能是刚命中或者刚被替换，都是刚被访问过，将其放到队列最后，其后面的元素依次前移；



1. 直接映射

直接映射主体和全相联类似，主要是地址解析的函数不同，这里tag位需要地址左移块数地log值和块大小的log值，块号是取中间一段，需要用到按位与：



除此之外，直接映射不需要更新替换队列

1. 组相联

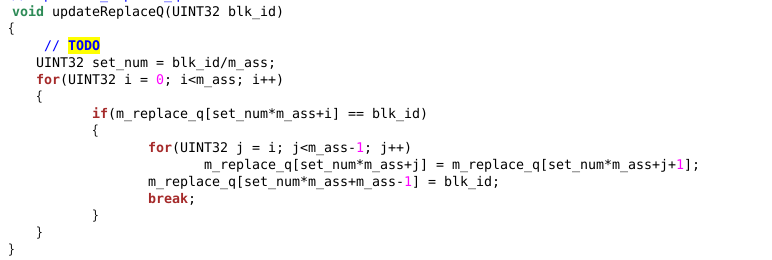


继承父类的前提下还要声明自己的属性来保存相联度和组个数、块大小的log值

解析出标志和组号



更新替换队列时要先算出所属的块号，然后乘以相联度来得到这一组的起始块号，更新策略和之前相同：



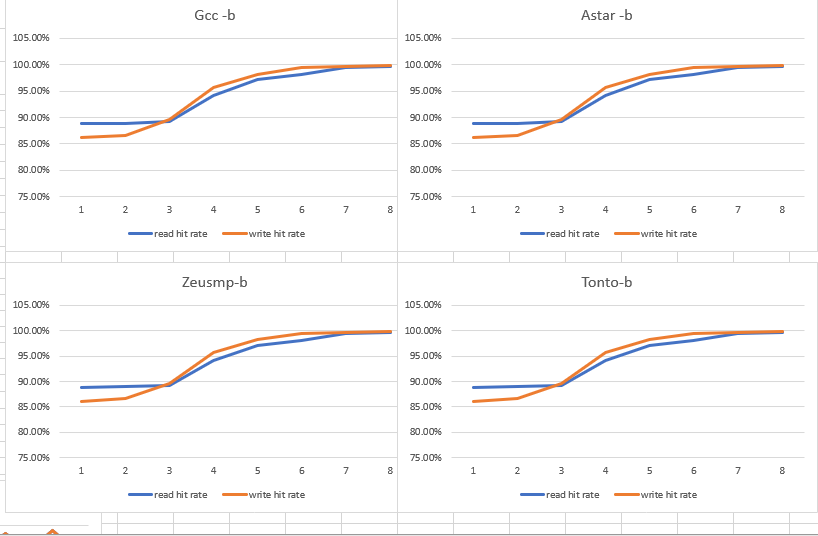
1. 测试结果及分析

1、块大小影响

表一：只改变块大小（块大小的log值）读写命中率的变化（相联度4，块个数512）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 块大小log值b | Gcc | Astar | Zeusmp | Tonto |
| 1 | read:88.69%  Write:86.16% | read:86.28%  Write:86.09% | read:86.98%  Write:86.01% | read:88.82%  Write:86.13% |
| 2 | read:88.89%  Write:86.44% | read:88.75%  Write:86.75% | read:88.63%  Write:86.36% | read:88.93%  Write:86.57% |
| 3 | read:88.58%  Write:89.74% | read:90.39%  Write:89.87% | read:88.25%  Write:89.41% | read:89.17%  Write:89.59% |
| 4 | read:93.69%  Write:95.73% | read:93.6.9%  Write:95.74% | read:94.47%  Write:95.80% | read:94.16%  Write:95.76% |
| 5 | read:97.25%  Write:98.17% | read:97.11%  Write:98.18% | read:97.14%  Write:98.16% | read:97.16%  Write:98.17% |
| 6 | read:98.67%  Write:99.20% | read:98.24%  Write:99.49% | read:98.09%  Write:99.47% | read:98.15%  Write:99.49% |
| 7 | read:99.46%  Write:99.64% | read:99.44%  Write:99.74% | read:99.49%  Write:99.64% | read:99.49%  Write:99.63% |
| 8 | read:99.78%  Write:99.83% | read:99.70%  Write:99.83% | read:99.77%  Write:99.83% | read:99.67%  Write:99.83% |

图一：只改变块大小（块大小的log值）读写命中率的变化（相联度4，块个数512）



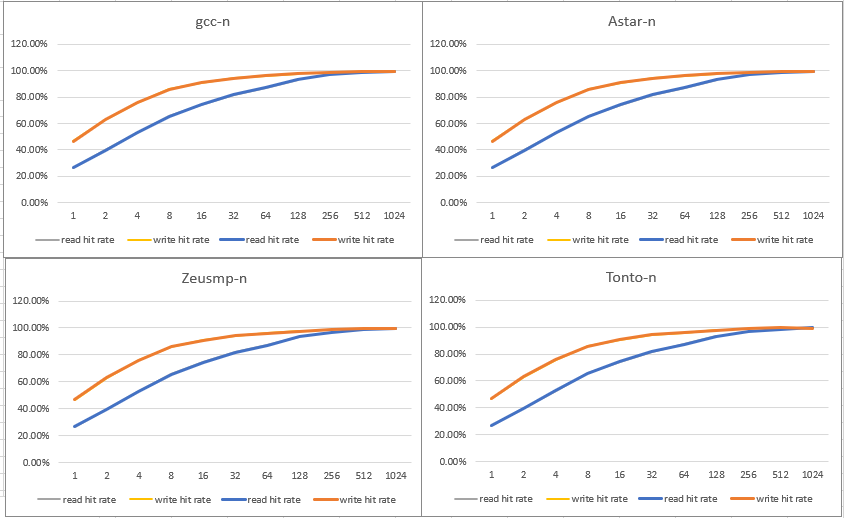
分析：块大小范围是1-8，通过观察四个测试程序的命中率之后发现，命中率随块大小的增大上升，且当大小达到2^7时命中率达到顶峰，不再有较大变化，事实上保持其他参数不变，块大小在增大的同时，cache的总容量是增加的，因此命中率会不断上上升，到2^7之后也基本上接近100%，不再上升；

2、块个数影响

表二：只改变块个数命中率的变化（相联度4，块大小2^6）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 块个数n | Gcc | Astar | Zeusmp | tonto |
| 1 | read:26.96%  Write:46.39% | read:26.43%  Write:46.76% | read:25.56%  Write:47.01% | read:26.66%  Write:46.78% |
| 2 | read:39.96%  Write:63.02% | read:39.68%  Write:63.03% | read:40.32%  Write:63.14% | read:39.71%  Write:63.02% |
| 4 | read:53.00%  Write:75.69% | read:53.59 %  Write:76.05% | read:53.13%  Write:76.08% | read:52.92%  Write:75.89% |
| 8 | read:65.50%  Write:85.91% | read:65.99%  Write:86.11% | read:65.30%  Write:85.86% | read:65.60%  Write:85.91% |
| 16 | read:74.46%  Write:90.72% | read:74.38%  Write:90.72% | read:74.36%  Write:90.83% | read:74.40%  Write:90.70% |
| 32 | read:81.77%  Write:94.08% | read:81.72%  Write:94.05% | read:81.79%  Write:94.18% | read:81.80%  Write:94.21% |
| 64 | read:87.31%  Write:96.04% | read:87.08%  Write:96.05% | read:87.06%  Write:95.78% | read:87.16%  Write:96.00% |
| 128 | read:93.03%  Write:97.50% | read:93.25%  Write:97.58% | read:93.13%  Write:97.53% | read:93.29%  Write:97.56% |
| 256 | read:97.02%  Write:98.71% | read:96.91%  Write:98.76% | read:96.94%  Write:98.75% | read:96.89%  Write:98.75% |
| 512 | read:98.60%  Write:99.55% | read:98.60%  Write:99.55% | read:98.60%  Write:99.55% | read:98.60%  Write:99.55% |
| 1024 | read:99.38%  Write:99.29% | read:99.41%  Write:99.35% | read:99.43%  Write:99.34% | read:99.40%  Write:99.33% |

图二：只改变块个数命中率的变化（相联度4，块大小2^6）



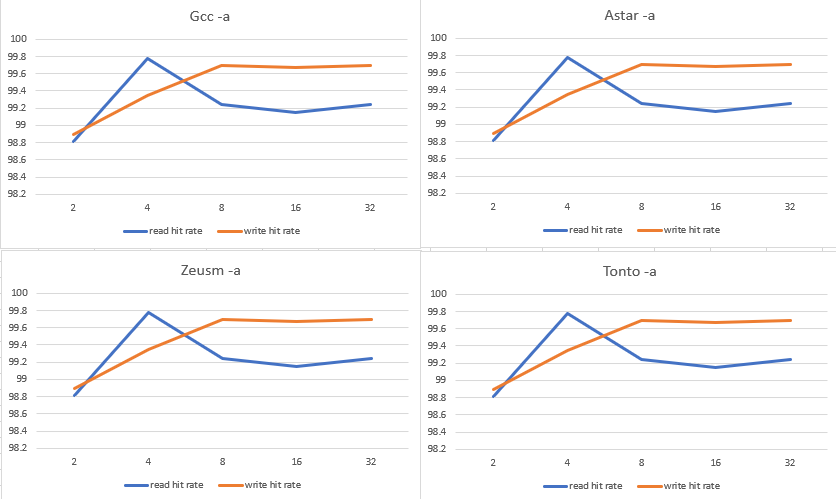
分析：同样，随着块个数的增加，cache容量增加，命中率不断上升，最后在512左右基本接近100%，不再变化；

3、相联度的影响

表三：只改变相联度，命中率的变化（块个数512，块大小2^6）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 相联度-a | Gcc | Astar | Zeusm | Tonto |
| 2 | Read:98.78% write:98.85% | Read:98.76% write:98.85% | Read:98.80% write:98.89% | Read:98.81% write:98.89% |
| 4 | Read:97.78% write:99.35% | Read:97.72% write:99.22% | Read:97.82% write:99.30% | Read:99.78% write:99.35% |
| 8 | Read:98.92% write:99.62% | Read:98.92% write:99.62% | Read:98.93% write:99.62% | Read:99.24% write:99.69% |
| 16 | Read:99.15% write:99.67% | Read:99.15% write:99.67% | Read:99.15% write:99.67% | Read:99.15% write:99.67% |
| 32 | Read:99.24% write:99.69% | Read:99.24% write:99.69% | Read:99.24% write:99.69% | Read:99.24% write:99.69% |

图三：只改变相联度，命中率的变化（块个数512，块大小2^6）



分析：在cache容量不变的情况下改变相联度对于读命中率来说，都是在相联度为4时达到顶峰，对于写命中率来说在相联度达到8之后基本不再变化，这和课上所讲大于8的相联度实际意义不大所符合，而都命中率在相联度大于8之后有所下降，但是幅度并不大；

表四：块数16，相联度为2，块大小为2^2时三种方式的命中率和平均读写时间：

|  |  |  |  |
| --- | --- | --- | --- |
|  | 全相联 | 直接映射 | 组相联 |
| Gcc | read:44.05% 0.53  Write:40.59% 1.04 | read:33.50% 0.55  Write:33.10% 1.07 | read:76.03% 0.52  Write:75.30% 1.01 |
| Astar | read:42.98% 0.54  Write:40.72% 1.04 | read:32.41% 0.55  Write:32.74% 1.06 | read:76.02% 0.52  Write:76.13% 1.00 |
| Zeusmp | read:41.17% 0.55  Write:40.51% 1.03 | read:30.40% 0.57  Write:31.63% 1.06 | read:74.65% 0.53  Write:74.07% 0.98 |
| Tonto | read:40.99% 0.53  Write:40.49% 0.98 | read:29.79% 0.52  Write:31.65% 0.97 | read:74.87% 0.49  Write:74.15% 0.90 |

分析：这一组参数是小容量cache，可以看出其命中率都较低，同时其平均运行速度相比其他两组cache也比较低，猜测原因是cache容量小，在遍历寻找标志位比较的时候运行次数较少，而三种组织方式相比，明显组相联的效果最好，直接相连最差，说明在小容量cache条件下应该尽量使用组相联。

表五：块数256，相联度为4，块大小为2^4时三种方式的命中率和平均读写时间：

|  |  |  |  |
| --- | --- | --- | --- |
|  | 全相联 | 直接映射 | 组相联 |
| Gcc | read:89.61% 0.84  Write:93.00% 1.56 | read:84.06% 0.56  Write:89.00% 1.04 | read:92.73% 0.52  Write:95.26% 0.57 |
| Astar | read:89.49% 0.83  Write:92.83% 1.54 | read:83.87% 0.55  Write:89.02% 1.02 | read:92.65% 0.54  Write:95.21% 1.00 |
| Zeusmp | read:89.15% 0.82  Write:93.43% 1.55 | read:84.24% 0.54  Write:89.57% 1.03 | read:92.08% 0.51  Write:95.72% 0.98 |
| Tonto | read:88.85% 0.84  Write:93.88% 1.63 | read:83.90% 0.55  Write:90.06% 1.07 | read:91.45% 0.53  Write:96.03% 1.03 |

分析：这一组cache容量较大，相联度和块大小也有了提高，因此整体命中率都不低，仍然是组相联的命中率最高，直接映射最低，因此组相联的运行时间也是最低的，可以发现，除非cache容量可以很大，否则不必要使用直接相连，组相联选定合适的相联度是最好的；

表六：块数1024，相联度为8，块大小为2^8时三种方式的命中率和平均读写时间：

|  |  |  |  |
| --- | --- | --- | --- |
|  | 全相联 | 直接映射 | 组相联 |
| Gcc | read:99.90% 1.39  Write:99.86% 2.46 | read:98.82% 0.48  Write:99.65% 0.86 | read:99.89% 0.42  Write:99.86% 0.75 |
| Astar | read:99.89% 1.50  Write:99.86% 2.70 | read:99.21% 0.46  Write:99.68% 0.83 | read:99.88% 0.46  Write:99.86% 0.83 |
| Zeusmp | read:99.89% 1.64  Write:99.86% 2.95 | read:98.05% 0.52  Write:99.42% 0.93 | read:99.88% 0.52  Write:99.86% 0.94 |
| Tonto | read:99.88% 1.47  Write:99.86% 2.60 | read:99.23% 0.48  Write:99.72% 0.86 | read:99.87% 0.49  Write:99.85% 0.87 |

分析：这一组cache容量最大，从命中率来看，三种方式都很接近，全相联和组相联的命中率几乎相等，这是由于相联度调到了8所导致的，这种情况下直接映射的平均时间较少，命中率和其他两种差不多，也是很好的选择；

以上三种方式当中，组相联的结构最复杂，其效果也最好，直接映射最简单，效果较差，需要较大的cache容量才能达到很好的效果，而对于全相联，其实现成本最高，而组相联设置相联度为8时的效果和全相联类似，因此相联度为8的组相联cache是很好的选择。

附录：

部分截图：

