

实验报告

开课学期： 2022秋季

课程名称： 计算机体系结构(实验)

实验名称： 实验3：Cache设计

学生班级： 6

学生学号： 200110618

学生姓名： 邓皓元

实验与创新实践教育中心印制

2022年11月

# 1. 题目分析

*回顾课上讲解的几种不同地址索引方式的Cache，用自己的话描述其原理及工作过程，并对比分析各索引方式的优缺点。*

// TODO

VIVT Cache工作原理：

采用虚拟地址作为索引和块标记，可以直接接受虚拟地址访问。当CPU需要访问存储器时，将虚拟地址同时发给TLB和Cache，以同时进行Cache访问和虚实地址转换 —— Cache接受虚拟地址并判断当前访问是否命中，如果命中则直接从相应的Cache块中取出数据并返回给CPU；如果发生缺失，则使用TLB转换得到的物理地址访问主存，以取出数据块并装入Cache。

VIVT Cache工作过程：

1.将虚拟主存地址通过映射规则映射成Cache地址，分解成对应的虚拟tag标记和虚拟index索引

2.用该Cache地址(虚拟tag标记和虚拟index索引)查询Cache目录表，判断是否命中

3.若命中，则直接使用Cache目录表中该Cache地址存放的数据

4.若未命中，则需要使用TLB转换得到的物理地址访问主存，将对应的数据存入Cache目录表中(虚拟tag标记和虚拟index索引)，之后再次访问Cache得到正确的数据

PIPT Cache工作原理：

PIPT的Cache又称物理Cache，采用物理地址作为索引和块标记，接受物理地址访问。CPU在访问存储器时，首先需要将虚拟地址发给TLB进行虚实地址转换，然后使用得到的物理地址访存。当Cache命中时，从相应的Cache块中取出数据并返回给CPU；当Cache访问缺失时，使用物理地址访问主存，取出相应的数据块并装入Cache。

PIPT Cache工作过程：

1.将虚拟主存地址通过TLB进行虚实地址转换变为物理主存地址，将物理主存地址通过映射规则映射成Cache地址，分解成对应的物理tag标记和物理index索引

2.用该Cache地址(物理tag标记和物理index索引)查询Cache目录表，判断是否命中

3.若命中，则直接使用Cache目录表中该Cache地址存放的数据

4.若未命中，则需要使用TLB转换得到的物理地址访问主存，将对应的数据存入Cache目录表中(物理tag标记和物理index索引)，之后再次访问Cache得到正确的数据

VIPT Cache工作原理：

VIPT的Cache采用虚拟地址作为索引、物理地址作为块标记，接受虚拟地址访问。CPU访问VIPT Cache时，使用虚拟地址作为索引查Cache的元数据，同时将虚拟地址发给TLB进行虚实地址转换。理想情况下，虚拟索引在Cache中查到了相应的数据时，TLB的虚实地址转换也正好完成，此时即可进行物理标识的比对。

VIPT Cache工作过程：

1.将虚拟主存地址通过TLB进行虚实地址转换变为物理主存地址，将虚拟主存地址和物理主存地址通过映射规则映射成Cache地址，分解成对应的物理tag标记和虚拟index索引

2.用该Cache地址(物理tag标记和虚拟index索引)查询Cache目录表，判断是否命中

3.若命中，则直接使用Cache目录表中该虚拟Cache地址存放的数据

4.若未命中，则需要使用TLB转换得到的物理地址访问主存，将对应的数据存入Cache目录表中(物理tag标记和虚拟index索引)，之后再次访问Cache得到正确的数据

各索引方式的优缺点：

VIVT(虚拟索引-虚拟标识)：

优点：在VIVT Cache存储系统中，Cache访问和虚实地址转换是并行进行的，在Cache缺失时可以马上使用物理地址访问主存，因此VIVT Cache具有较低的访问延迟

缺点：由于不同的进程具有不同的虚拟地址空间，因此发生进程切换时，需要清空VIVT Cache，从而增加了进程切换的开销，也降低了整体的Cache命中率

PIPT(物理索引-物理标识)：

优点：发生进程切换时，不需要清空Cache，整体的Cache命中率较高。

缺点：PIPT Cache存储系统的Cache访问和虚实地址转换是串行进行的，因此具有较大的访问延迟

VIPT(虚拟索引-物理标识)：

优点：VIPT Cache采用了虚拟地址索引、物理地址标识，使得Cache访问和TLB虚实地址转换具有一定的并行度，从而同时具备了VIVT Cache并行访问的优点和PIPT Cache不受进程切换影响的优点。整体的Cache命中率较高且访问延迟较低

# 2. 设计与实现

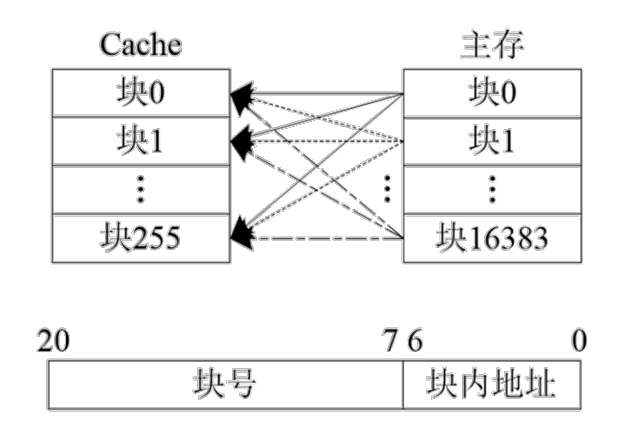
*\*注：若做了附加题，除了提交相应代码，还需按照指导书要求，在报告中添加相应的分析、设计过程和运行结果截图和对比等内容，否则不加分。*

## 2.1 方案设计

*详细阐述各个Cache模型的关键实现，建议结合算法流程图等图形化工具来辅助描述。*

// TODO

全相联Cache：结构：



在 Cache Model Base基础上增加的成员变量：

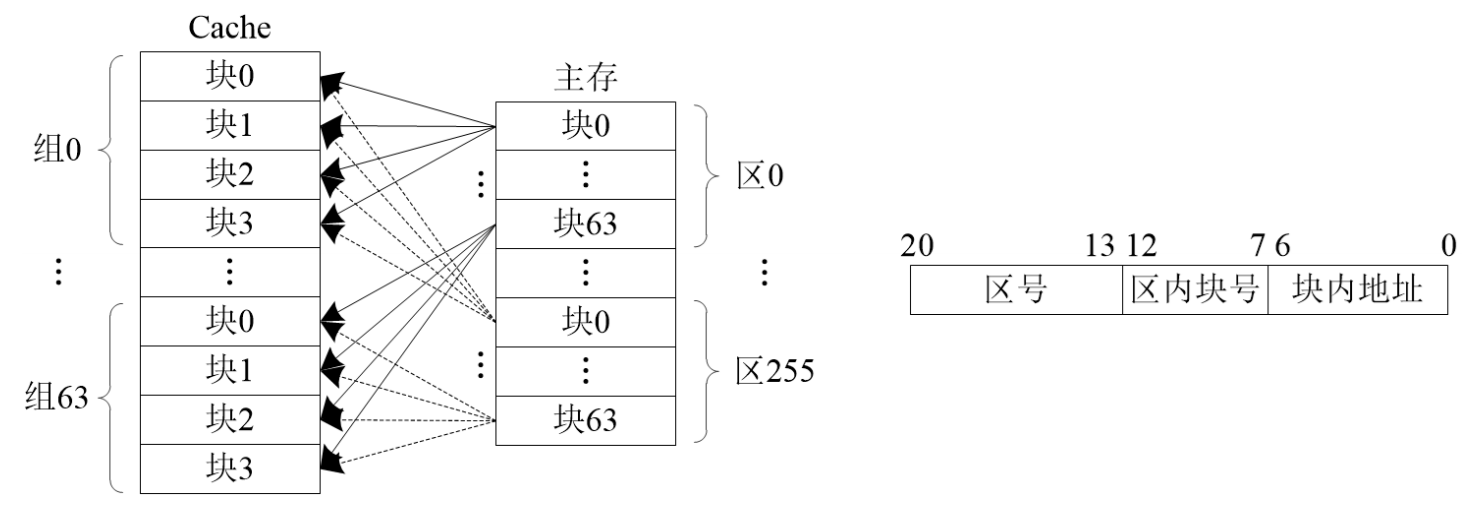
UINT32 cur = 0;

m\_replace\_q记录了每个目录项的活跃度

实现方法：从主存地址中分解出Cache地址，利用Cache地址在整个Cache空间中查找是否命中，若命中对应的m\_replace\_q[命中index]++，若未命中，就进行Cache块替换

替换方法：需要替换时，先在整个Cache空间中寻找空闲的Cache块(m\_valid[当前] == false)进行替换，若找不到空闲的Cache，再寻找最不活跃的Cache进行替换，设置一个cur指针(UINT32)用来遍历轮询m\_replace\_q数组，若是查到m\_replace\_q[cur]!=0则m\_replace\_q[cur]--，直到找到m\_replace\_q[cur]==0为止，将cur对应的Cache块替换(m\_tag[cur]=new tag(addr))，m\_replace\_q[cur]++，cur++)相当于一个全局LRU，cur每次到达最大块数则重置为0，每次找到最不活跃的Cache块之后，cur指向下一个Cache块

组相联Cache：结构：



在 Cache Model Base基础上增加的成员变量：

UINT32\* cur; // 指向每个组内块的LRU指针

UINT32 m\_sets\_log; // 组数量的对数

UINT32 m\_set\_block\_num; // 每个组中块的数量(相联度)

cur初始化为指向UINT32[1<<m\_sets\_log]，作为指向每个组内块的LRU指针，m\_sets\_log和m\_set\_block\_num则记录传递的参数sets\_log和set\_block\_num

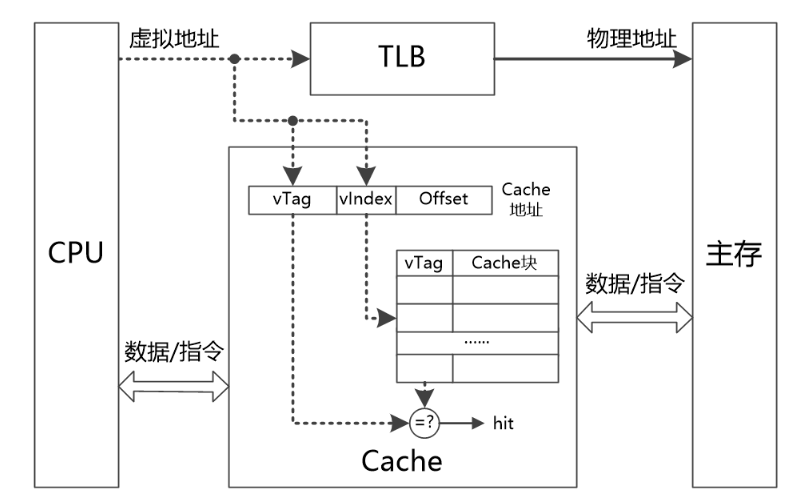
m\_replace\_q记录了每个目录项的活跃度

实现方法：从主存地址中分解出index地址和tag标记，查找对应的Cache组内是否有命中的Cache块，若命中对应的m\_replace\_q[当前]++，若未命中，就进行Cache块替换

替换方法：需要替换时，在对应的Cache组内中寻找空闲的Cache块(m\_valid[当前] == false)进行替换，若找不到空闲的Cache，再寻找最不活跃的Cache进行替换，利用当前组的cur[index地址]指针用来遍历轮询m\_replace\_q数组，若是查到m\_replace\_q[index地址\* m\_set\_block\_num + cur]!=0则m\_replace\_q[index地址 \* m\_set\_block\_num + cur]--

，直到找到m\_replace\_q[index地址 \* m\_set\_block\_num + cur]==0为止，将cur对应的Cache块替换(m\_tag[index地址 \* m\_set\_block\_num + cur]=tag标记)，m\_replace\_q[index地址 \* m\_set\_block\_num + cur]++，cur++)相当于组数量的局部LRU，cur每次到达最大组内块数则重置为0，每次找到组内最不活跃的Cache块之后，cur指向组内下一个Cache块

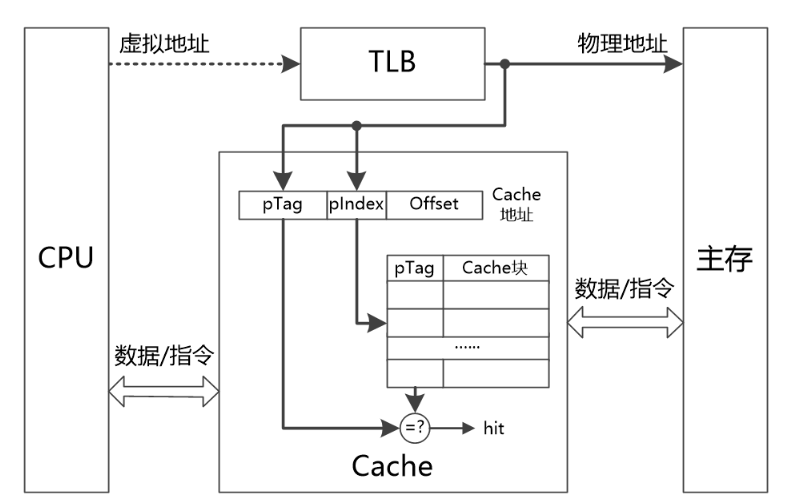
VIVT组相联Cache：结构：



在 Cache Model Base基础上增加的成员变量：与组相联Cache一样

实现方法和替换方法：与组相联Cache一样

PIPT组相联Cache：结构：



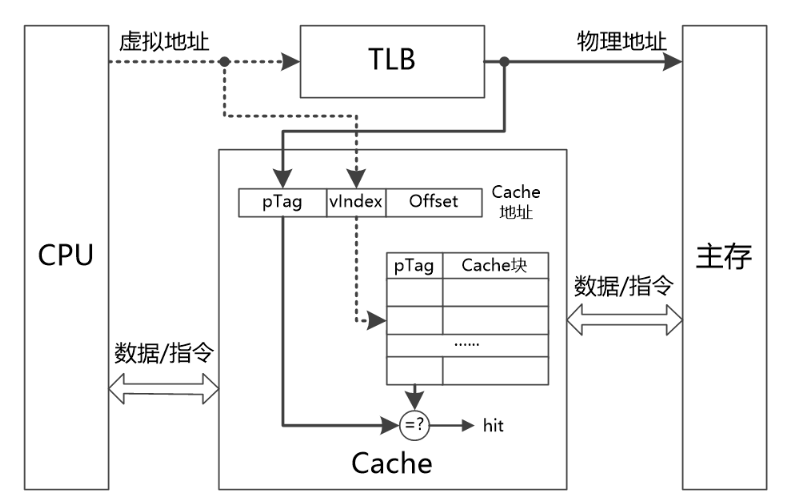
在 Cache Model Base基础上增加的成员变量：与组相联Cache一样

实现方法：利用TLB将虚拟主存地址转化为物理主存地址，从物理主存地址中分解出物理index地址和物理tag标记，查找对应的Cache组内是否有命中的Cache块，若命中对应的m\_replace\_q[当前]++，若未命中，就进行Cache块替换

替换方法：需要替换时，在对应的Cache组内中寻找空闲的Cache块(m\_valid[当前] == false)进行替换，若找不到空闲的Cache，再寻找最不活跃的Cache进行替换，利用当前组的cur[物理index地址]指针用来遍历轮询m\_replace\_q数组，若是查到m\_replace\_q[物理index地址 \* m\_set\_block\_num + cur]!=0则m\_replace\_q[物理index地址 \* m\_set\_block\_num + cur]--

，直到找到m\_replace\_q[物理index地址 \* m\_set\_block\_num + cur]==0为止，将cur对应的Cache块替换(m\_tag[物理index地址 \* m\_set\_block\_num + cur]=物理tag标记)，m\_replace\_q[物理index地址 \* m\_set\_block\_num + cur]++，cur++)相当于组数量的局部LRU，cur每次到达最大组内块数则重置为0，每次找到组内最不活跃的Cache块之后，cur指向组内下一个Cache块

VIPT组相联Cache：结构：



在 Cache Model Base基础上增加的成员变量：与组相联Cache一样

实现方法：利用TLB将虚拟主存地址转化为物理主存地址，从虚拟主存地址和物理主存地址中分解出虚拟index地址和物理tag标记，查找对应的Cache组内是否有命中的Cache块，若命中对应的m\_replace\_q[当前]++，若未命中，就进行Cache块替换

替换方法：需要替换时，在对应的Cache组内中寻找空闲的Cache块(m\_valid[当前] == false)进行替换，若找不到空闲的Cache，再寻找最不活跃的Cache进行替换，利用当前组的cur[虚拟index地址]指针用来遍历轮询m\_replace\_q数组，若是查到m\_replace\_q[虚拟index地址 \* m\_set\_block\_num + cur]!=0则m\_replace\_q[虚拟index地址 \* m\_set\_block\_num + cur]--

，直到找到m\_replace\_q[虚拟index地址 \* m\_set\_block\_num + cur]==0为止，将cur对应的Cache块替换(m\_tag[虚拟index地址 \* m\_set\_block\_num + cur]=物理tag标记)，m\_replace\_q[虚拟index地址 \* m\_set\_block\_num + cur]++，cur++)相当于组数量的局部LRU，cur每次到达最大组内块数则重置为0，每次找到组内最不活跃的Cache块之后，cur指向组内下一个Cache块

## 2.2 实验结果及分析

*（1）按照实验指导书要求，测试不同参数下的组相联Cache性能，将测试结果绘制成折线图，并分析实验结果；*

*（2）按照实验指导书要求，改变Cache参数，测量不同索引方式下的组相联Cache性能，绘制表格或折线图记录实验结果，并进行对比分析。*

*如果进行了优化，需给出优化前后的结果对比截图及文字分析（如果优化没有效果，也需分析原因）。*

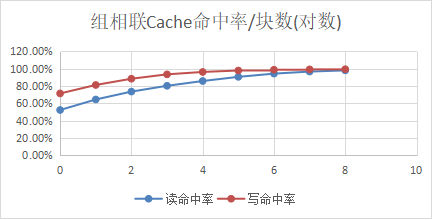
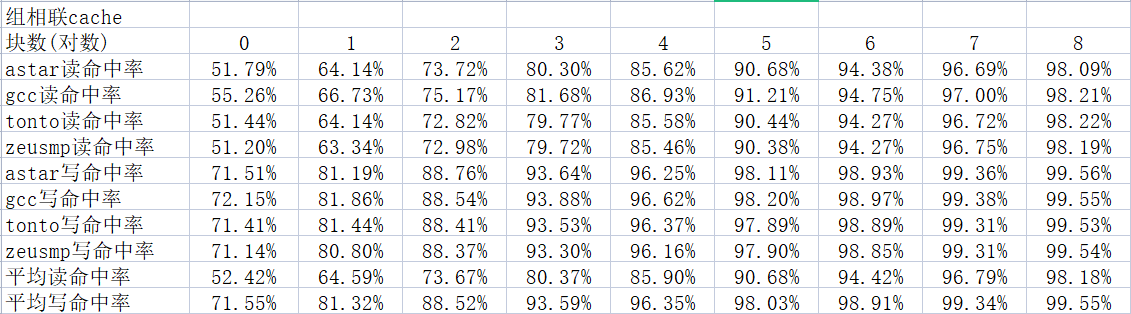
// TODO

做某个参数变化的测试时，其他参数保持默认值

默认值：块数(对数)为7、块大小(对数)为6、相联度为4

(1)

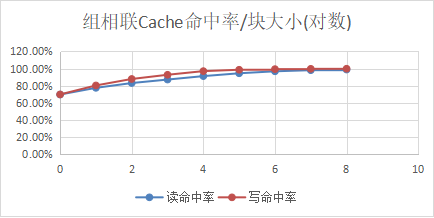
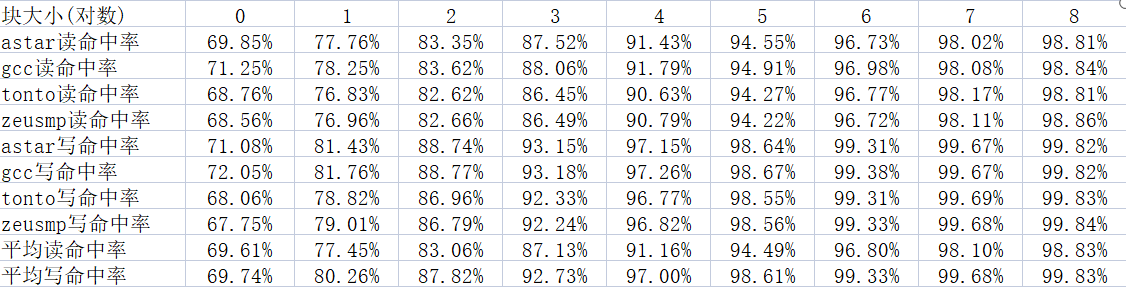
不同块数的组相联Cache性能(命中率)



分析测试结果可知，当其他参数不变时，随着块数的增加组相联Cache性能(命中率)逐渐增长速度较快，且块数影响组相联Cache性能(命中率)程度较大

且相同块数情况下组相联Cache写命中率高于读命中率较多

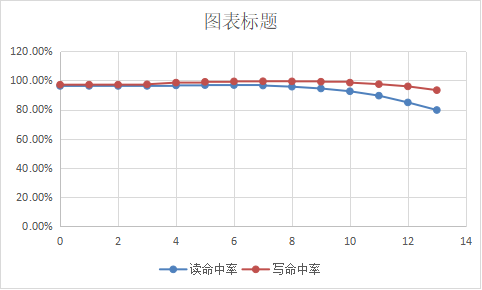
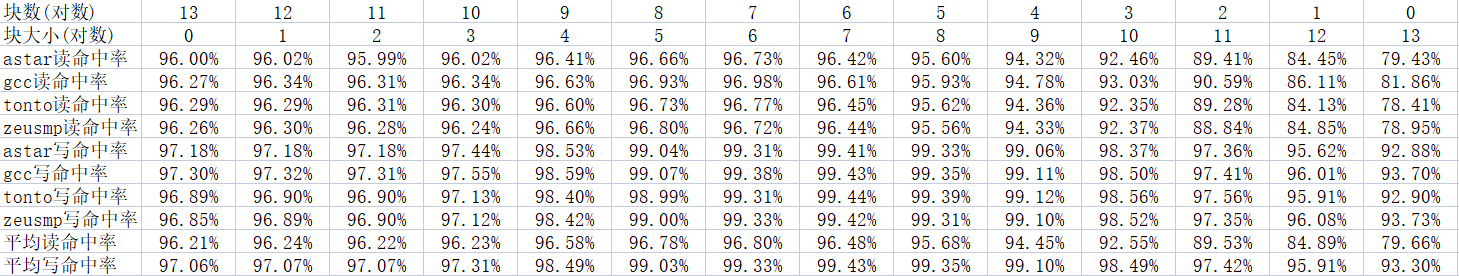
不同块大小的组相联Cache性能(命中率)(块数保持不变)



分析测试结果可知，当其他参数不变时，随着块大小的增加组相联Cache性能(命中率)逐渐增长且速度中等，且块数影响组相联Cache性能(命中率)程度中等

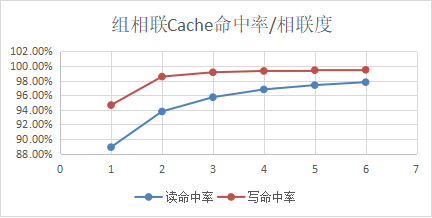
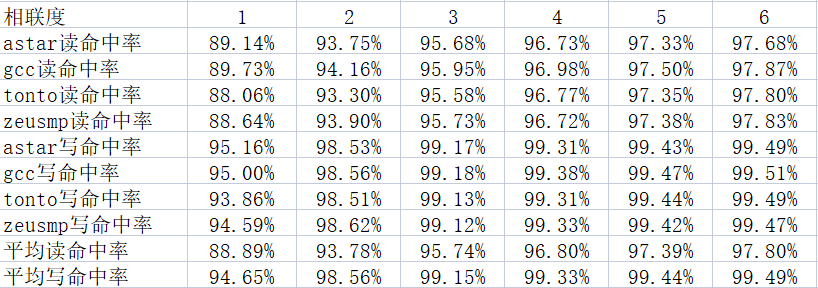
且相同块大小情况下组相联Cache写命中率高于读命中率较少

不同块大小的组相联Cache性能(命中率)(块容量保持不变)



但事实上，当Cache容量确定，而不是Cache块数确定的时候，块大小的增加会导致Cache块数的减少，所以一开始随着块大小增加的时候，命中率增加，但当块大小增加到一定大小时，块数逐渐减少，会导致Cache命中率降低，因此随着块大小增加Cache命中率先升高后降低(假设此时相联度为4，块数和块大小对数和为13)

不同相联度的组相联Cache性能(命中率)



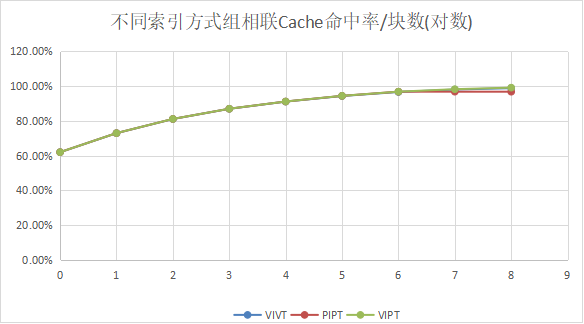
分析测试结果可知，当其他参数不变时，随着相联度的增加组相联Cache性能(命中率)逐渐增加速度较慢，且块数影响组相联Cache性能(命中率)程度较小

且相同相联度情况下组相联Cache写命中率高于读命中率中等

(2)

命中率=(读命中率+写命中率)/2

不同索引方式组相联Cache命中率/块数(对数)

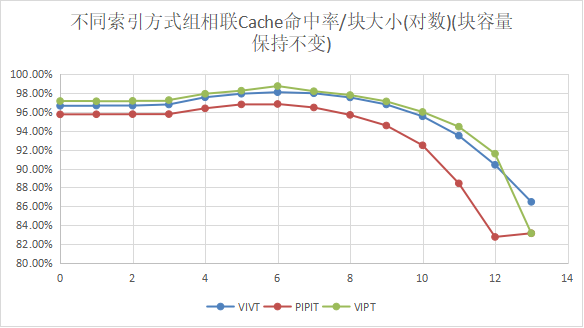


分析测试结果可知，当其他参数不变时，随着块数的增加组相联Cache性能(命中率)逐渐增长速度较快，且块数影响组相联Cache性能(命中率)程度较大

三种索引方式命中率接近

平均命中率：VIPT>VIVT>PIPT

不同索引方式组相联Cache命中率/块大小(对数)(块容量保持不变)

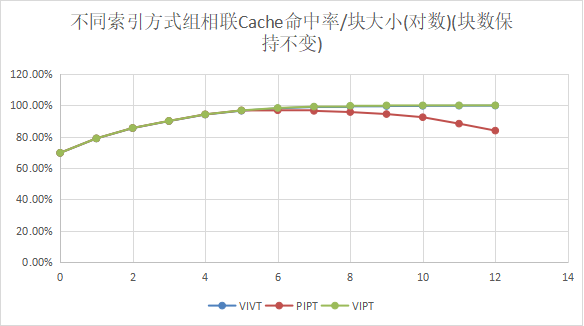


分析测试结果可知，当其他参数不变时，随着块大小的增加不同索引方式组相联Cache性能(命中率)先增长后降低，且当块大小较大的时候，块数较少导致命中率随着块大小增大降低速率越来越快

VIVT与VIPT索引方式命中率接近，PIPT命中率相对较低

平均命中率：VIPT>VIVT>PIPT

不同索引方式组相联Cache命中率/块大小(对数)(块容量保持不变)

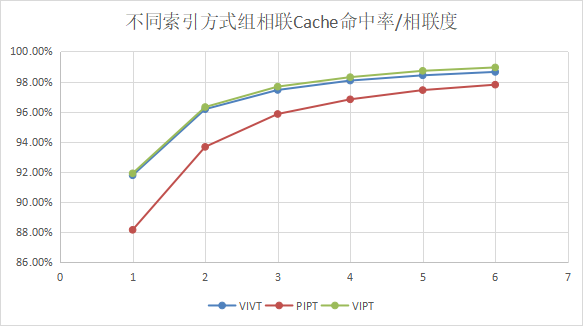


分析测试结果可知，当其他参数不变时，随着块大小的增加不同索引方式组相联Cache性能(命中率)逐渐增长，且块数影响组相联Cache性能(命中率)，

其中PIPT在块大小增大到一定程度时命中率开始降低，猜测是模拟TLB对于虚拟地址的转换效果不理想，导致模拟物理地址与真实物理地址相差较大，导致PIPT与VIVT、VIPT结果差距较大。

平均命中率：VIPT>VIVT>PIPT

不同索引方式组相联Cache命中率/相联度



分析测试结果可知，当其他参数不变时，随着相联度的增加组相联Cache性能(命中率)逐渐增长速度较慢，且相联度影响组相联Cache性能(命中率)程度较小

VIVT与VIPT索引方式命中率接近，PIPT命中率相对较低

平均命中率：VIPT>VIVT>PIPT