**《计算机原理与体系结构》实验报告**

**学号：16300240023 姓名：杨芗琳 学院：计算机科学技术学院 完成时间：2018.6.14**

1. 实验名称

Cache性能实验

1. 实验目的

更好地了解Cache的工作原理，包括

映像规则（直接映像、全相连映像，n路组相连映像）；

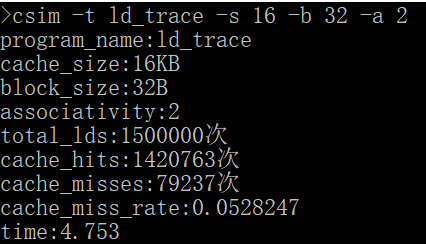
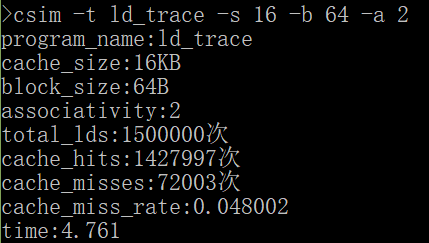
查找算法（如何查找数据是否在Cache中，需要什么样的数据结构支持）

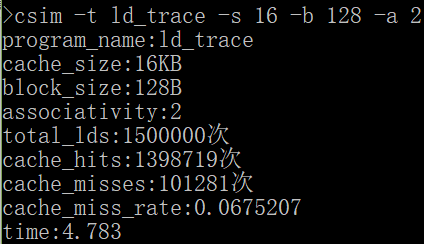
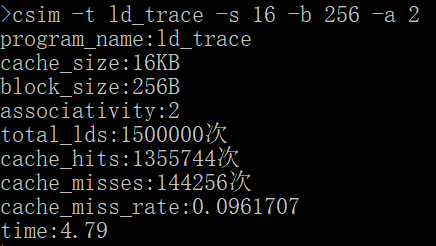
替换算法（LRU算法，随机算法）；

写策略（当进行写访问时，如何进行）。

1. 实验内容
2. 相同的Cache大小，不同的块大小情况下的cache命中率；

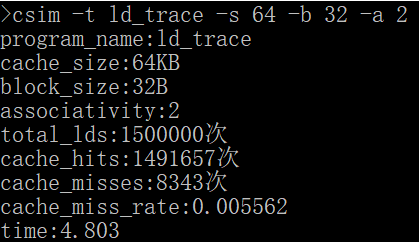
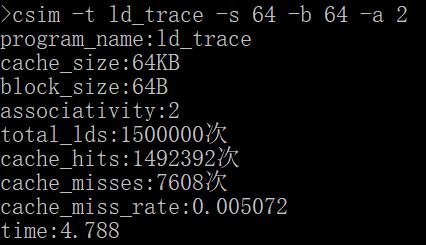
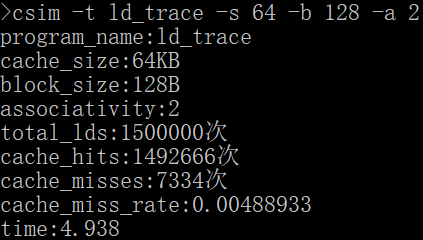
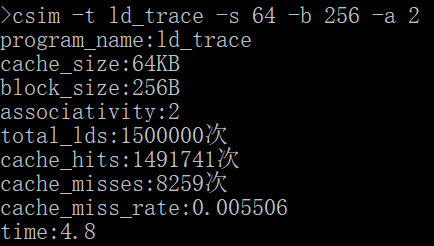
S=16KB，a=2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Cache块大小/B | 32 | 64 | 128 | 256 |
| Cache不命中率 | 0.053 | 0.048 | 0.068 | 0.096 |

S=64KB,a=2

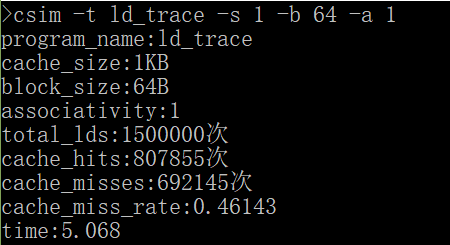
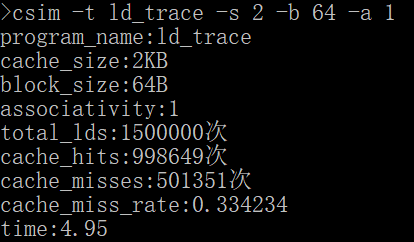
   

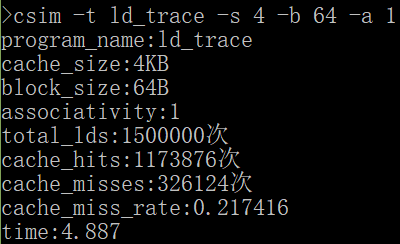
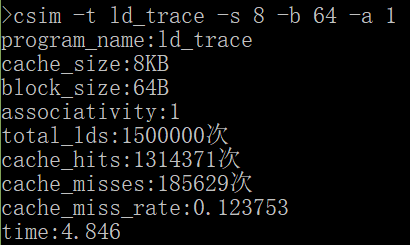
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Cache块大小 | 32 | 64 | 128 | 256 |
| Cache不命中率 | 0.0056 | 0.0051 | 0.0049 | 0.0055 |

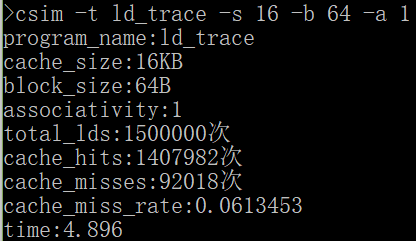
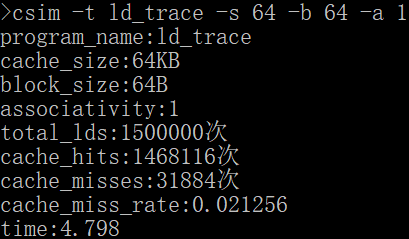
在Cache容量为16KB，两路组相联条件下，随着Cache块大小的增加，Cache不命中率先减少后增加。可以解释为1）增加了空间局部性，减少了强制性不命中2）减少了Cache中块的数目，所以有可能增加冲突不命中。在块大小比较小的情况下，第一种作用超过第二种，从而使不命中率下降。等到块大小较大时，第二种作用就超过了第一种，不命中率反而上升。

Cache容量越大，使不命中率达到最低的块大小也越大。在实验中，16KB、64KB的Cache，使不命中率达到最低的块大小分别是64B和128B。

1. 不同的Cache大小，相同的块大小情况下的cache命中率；

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Cache容量 | 1 | 2 | 4 | 8 | 16 | 64 |
| Cache命中率 | 0.46 | 0.33 | 0.22 | 0.12 | 0.06 | 0.02 |

降低Cache不命中率的直接方法使增加Cache容量，从实验数据很清楚看出，随着Cache容量的增加，不命中率大幅度减少。

1. 每次实验完成的时间：即csim程序执行的时间。

见截图

1. 编译运行环境

VS2017，Debug

1. Cache编写思路
2. 总体思路

模拟Cache的具体工作过程。对于每个地址，先除去块内位移（address/block\_size），然后得到索引（index\_temp = tmp % (size \* 1024 / block\_size / associativity);//计算当前索引）标识tag（tag\_temp = tmp/ (size \* 1024 / block\_size / associativity);//当前tag），最后查表，如果在数组contents（模拟目录表）中找到该项，则Cache\_hits++，否则，Cache\_misses++，并且按照随机替换法则，将新的块调入Cache中。

1. 实现

一个Cache类、一个item结构（模拟目录表中项）

|  |
| --- |
| class Cache {  public:  Cache() {  size = 0;  associativity = 0;  block\_size = 0;  cache\_hit = 0;  cache\_miss = 0;  total\_lds = 0;  cache\_miss\_rate = 0;  time = 0;  }  ~Cache() {}  void create(int s, int a, int b, string n);  void clear();  int read\_from\_file(string name);  void print\_reslut();  private:  int size;//Cache size,1KB~64KB(1024B~64\*1024B)  int associativity;//1,2,...,all  int block\_size;//32B,64B,128B,256B  int content;//表的项数 size\*1024/block\_size 4~2048  long cache\_hit;  long cache\_miss;  long total\_lds;  double cache\_miss\_rate;  double time;  string name;  item \*contents; //目录表  }; |
| struct item {  long tag = 0;  int in\_use = 0;  }; |

1. 具体函数

//具体实现见源代码

|  |
| --- |
| //公共函数  int random(int i)；//随机数生成函数  //Cache类成员函数  void create(int s, int a, int b, string n);  void clear();  int read\_from\_file(string name);  void print\_reslut(); |

1. 思考
2. 如何生成一个trace文件？

Rand()

单纯的rand()会返回一个0至RAND\_MAX之间的随机数值，而RAND\_MAX的值与int位数有关，最小是32767。不过rand()是一次性的，因为系统默认的随机数种子为1，只要随机数种子不变，其生成的随机数序列就不会改变。

Srand()

srand()可用来设置rand()产生随机数时的随机数种子。通过设置不同的种子，我们可以获取不同的随机数序列。可以利用srand((unsigned int)(time(NULL))的方法，利用系统时钟，产生不同的随机数种子。不过要调用time()，需要加入头文件< ctime >。

利用随机数生成一个trace文件

1. 如果把内存写(store指令)也考虑在内，应该如何修改程序？

应该在item类中添加一项Dirty，以表示数据是否被修改；同时添加一个硬件，批量写入修改的值。

1. 源代码

见附件：

Project1，VS2017，Debug

C++编写