Lab04 实验报告

姓名： 何伟　　**学号：** 171240537

2018 年 12 月 12 日

摘要

实现了 cache 的模拟, 多次随机测试通过

1 实验进度

1.1 init_cache

用结构体组织 cache 的行, 组, 和 cache. 在 init 函数中申请空间, 并将 valid_bit 设置为 0;

1.2 cache_read

根据组相联映射 cache 的访问过程. 想判断对应的组中是否有对应的 tag 和有效的 valid_bit, 命中则直接根据块内地址从 cache 中读出信息. 没有命中在分两种情况, 如果当前组内有空行, 则读出主存块到这个空行, 否则, 随机选一组替换.

1.3 cache_write

与 read 类似, 先判断是否命中, 命中则写进 cache. 不命中, 则, 分配一行, 将主存块读入这一行在进行写的操作, 若没有空闲行则需要进行替换. 随机替换, 并判断修改为是否为 1, 如果是 1, 要将替换行的内容先写回主存块, 在进行 cache_write.
最后随机种子测试均通过.

```
allcnt: 9888798
 maruko@me  ~/lab4  master + ● ?  ./a.out
random seed = 1544511337
Random test pass!
cycle_cnt: 56634615
miss_cnt: 1524045
allcnt: 9887883
 maruko@me  ~/lab4  master + ● ?  ./a.out
random seed = 1544511339
Random test pass!
cycle_cnt: 56634827
miss_cnt: 1524053
allcnt: 9888976
 maruko@me  ~/lab4  master + ● ?  ./a.out
random seed = 1544511340
Random test pass!
cycle_cnt: 56634757
miss_cnt: 1524049
allcnt: 9888393
 maruko@me  ~/lab4  master + ● ?  ./a.out microbench-test.log.bz2
random seed = 1544511343
cycle_cnt: 61012591
miss_cnt: 1604788
allcnt: 3411581
```

2 思考题

64B 的 cache,4 路组相联测得了随机序列和 microbench_test 的命中率如下:

```
maruko@me ~/lab4 master ● ./a.out
random seed = 1544515050
Random test pass!
cycle_cnt: 56634251
miss_cnt: 1524023   allcnt: 9889276   RATE: 0.845891
maruko@me ~/lab4 master ●
```

```
maruko@me ~/lab4 master + ./a.out microbench-test.log.bz2
random seed = 1544541147
cycle_cnt: 2836758  misscnt: 27951  allcnt: 1807431  0.984536
```

建模, 考虑到访问 cache 时要在组内的行中进行搜索, 跟每组的行数有关, 于是将访问 cache 在 for 循环找到满足条件的行数过程中每次加 1, 在访问主存时会进行 memcpy 操作, 而代价与块大小有关, 建模成与 BLOCK_WIDTH 有关的函数, 随后修改 cache 的参数.

首先是在 cache 总大小为 64B, 组关联度为 4 下, 块大小的改变. 从宽度为 6 到 7, 再到 8, 命中率逐渐提高, 访问主存带来的代价虽然提高了, 但是次数少了, 总体的代价还是降低了.

```
maruko@me ~/lab4 master + ● ? ./a.out microbench-test.log.bz2
random seed = 1544544398
cycle_cnt: 12124578  misscnt: 27945  allcnt: 1807431  RATE: 0.984539
maruko@me ~/lab4 master + ● ? rm a.out
maruko@me ~/lab4 master + ● ? make
gcc -Wall -Werror -O2 -ggdb -o a.out main.c cpu.c cache.c mem.c
maruko@me ~/lab4 master + ● ? ./a.out microbench-test.log.bz2
random seed = 1544544440
cycle_cnt: 10844808  misscnt: 14096  allcnt: 1807431  RATE: 0.992201
maruko@me ~/lab4 master + ● ? rm a.out
maruko@me ~/lab4 master + ● ? amke
zsh: command not found: amke
maruko@me ~/lab4 master + ● ? make
gcc -Wall -Werror -O2 -ggdb -o a.out main.c cpu.c cache.c mem.c
maruko@me ~/lab4 master + ● ? ./a.out microbench-test.log.bz2
random seed = 1544544460
cycle_cnt: 10091371  misscnt: 7186  allcnt: 1807431  RATE: 0.996024
```

之后, 将块宽度设为 6, 改变 cache 总大小, 在达到 68B 之后命中率不会有较大的提高, 总体的代价随着 cache 总大小的增加缓慢降低.

之后考察关联度对 cache 代价的影响, 发现关联度变大了反而会增加开销, 打印出 microbench_test 中所有的访存序列, 发现地址较为连续, 因此最好使用直接映射的方式, 命中时间短.

对于 microbench_test, 采用直接映射设计的 cache 反而效率更高,cache 的总大小在 64B 较为合适, 而 BLOCK_WIDTH 提高也会对命中率有很大影响, 取 7 合适.

3 实验心得

实现之前要好好理解 cache 的工作原理–命中不命中怎么读写, 以及写回写分配, 随机替换等. 之后想清楚主存地址如何划分, 都准备好之后就可以开始写了, 由于事先写了 ICS 作业, 对 cache 有了初步的了解. 但写的时候还是找了很久的 bug. 第一个坑是写的

时候是低地址在地位, 一开始没有注意, 过不去. 后来把一个块的数据都打印出来发现了. 第二个是 cache_write 替换的时候直接把数据替换到了当前地址, 还没有反应过来, 找了不少时间的 bug. 第三个是读写的地址都要转化为 4 的整数倍. 还有一些小问题什么的也记不清了.

copy paste 是个不好的习惯, 经常挂在嘴边说, 但是写的时候还是没想起来, 读写时都有大量的代码重复段, 完成的时候都是直接 copy paste 了, 导致 debug 的时候要改很多东西. 后来才把重复的地方封装在函数中, 代码看起来也整洁多了.

写完之后运行出错, 不知道是读出错了还是写出错了. 可能在遥远的过去某次写就出了问题. 为了将 bug 及时的反映出来. 在 cpu_uncache_write 之后, 通过 cache_read 和 mem_uncache_read 对刚写的进行比对, 及时报错, 有利于快速 debug.