计算机系统结构 lab1

Cache Analysis - report

梅瑞虎 - 2016011335 - 计64

2020 年 4 月 8 日

1 Cache 模拟总体框架

本项目包含以下主要内容:

```
code
|----CMakeList.txt              # cmake 配置文件
|----include
|    |----MappingPolicy/        # 映射方式实现
|    |----ReplacementPolicy/    # 替换策略实现
|    |----Handler.h             # Cache 模拟类
|    |----BlockPolicy.h         # 块配置策略类
|    |----WritemissPolicy.h     # 写策略
|    |----Utils.h               # 辅助函数
|    |----Reader.h              # 读取 trace
|    |----Writer.h              # 结果输出
|    |----...
|----src
|    |----Handler.cpp           # 模拟类的部分成员函数实现
|    |----CacheSim.cpp          # 函数入口
|    |----Utils.cpp             # 辅助函数实现
\----trace/                     # trace 文件
```

2 编译运行

需要 g++ make 以及 3.10 以上版本的 cmake. 如下可正常编译运行:

```
$ mkdir build && cd build
$ cmake .. && make
$ ../bin/CAHCE
```

默认块大小 8B, 8 路组相联组织 Cache, 写回写分配, 使用 LRU 替换算法. 输出到 log 文件夹. 同时控制台内输出 Cache 综述, 以及元数据所占物理位数和真实使用位数. 输出最终的 miss rate (百分数表示, 保留 3 位小数).

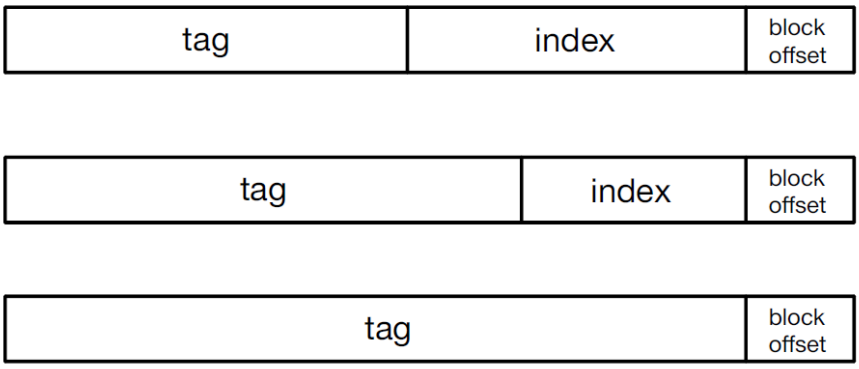
3 实现细节

Cache 写策略实现比较简单, 其中写回和写直达在本次实验中并没有体现出区别, 两者主要是在速度和一致性上有区别, 写回速度快, 写直达一致性好, 对于缺失率并无影响 (单 CPU 单 Cache). 同时写回造成元数据多 1 位 (dirty 位).

块大小主要影响了元数据的空间大小.

3.1 映射方式

对于 64 位地址, 不同映射对应的地址划分如下图, 分别对应直接映射, 组相联映射和全相联映射.



元数据包含三部分: valid(1), dirty(1/0) 和 tag(x). 由于没有真实存取数据, 三种映射方式本质上都是对元数据的修改和读取, 所以抽象出了接口, 根据元数据建立时的位宽和

tag 位宽模拟 Cache 的实现. 而 LRU 和 PLRU (二叉树) 替换策略会用到这里 index=n 的位宽, 定义直接映射 n=1, 全相联 n=N, 其中 N 为 Cache 总块数.

在空间使用上, 根据一组元数据实际使用的位数向上取 8 的倍数, 即使用最少 Byte 来存每一组元数据.

另外由于使用 OOP, 在行为上与真实硬件有差别.

3.2 替换策略

替换策略也进行了接口抽离.

实现的替换策略均指可用 Cache 已满条件下的替换, 未满则按顺序填入可用 Cache 块.

随机替换最为简单, 不再赘述.

LRU 替换策略中, 对每一组 (n 块) Cache, 需要 n*log2(n) 空间记录. 与映射方式中实现相同, 使用最少 Byte 进行实际存储.

PLRU 替换策略中, 每一组需要 n-1 位记录. 与之前不同的是, 一组 Cache 在 n 较小时, 使用最少 Byte 单独存一组记录值会浪费大量空间, n=4 时, 空间利用率仅为 37.5%.

于是计算总空间需求大小, 用最少 Byte 进行记录. 缺点是涉及更多位运算, 易出错且调试麻烦.

4 简要分析

详见 res 目录下的图表对比和 log_output 目录下模拟结果.

末尾标志: B 表示写回, T 表示写直达; A 表示写分配, N 表示写不分配.

4.1 块大小

在块大小大到一定程度后, 块越大, 缺失率越大. 极端情况下, 仅有 1 块, 基本只能利用空间局部性, 时间局部性利用不充分, 缺失率极大.

在块大小小到一定程度后, 块越小, 缺失率越大. 极端情况下, 1 块仅能存储 1 Byte, 只能利用时间局部性, 无法利用空间局部性, 缺失率极大.

在本次实验中, 8B 比较合适, 而 32B 64B 较大, 在结果上体现为块越大缺失率越大 (多数 trace).

4.2 映射方式

根据一组 Cache 的块数 n, n 越大, 缺失率相对越低.

但是相对的, n 越大查找效率和替换效率越低, 如全相联仅含 1 组, n 最大, 效率最低. 而其余三种映射方式在所有 trace 上速度基本不可辨.

4.3 替换方式

在实际实验中, 随机替换和理论上的最优策略 LRU 相差不大, 而 PLRU (二叉树) 替换策略最差.

4.4 写策略

本实验中写回写直达无区别, 写分配的缺失率总体上要比写不分配低, 与此同时写分配效率要比写不分配低 (也是有利有弊).

写策略可能要根据应用场景来选择.

5 实验感想

强烈希望老师和助教在之后的实验中提供一份验证集(小规模)答案, 或者大致范围 (比如缺失率在 60%~80%).

软件模拟硬件, 在尽可能贴近硬件行为时调试会极为麻烦, 而且一个小 bug 可能不影响正确执行, 但是对结果影响极大. 比如我在实现全相联映射时, 发现了之前在直接映射实现中留下的 bug (位操作相关), 但是可以正常跑出结果 (错误结果).

另外希望之后的实验可以有一个框架, 或者删去核心内容的 step by step 的文档. 便于助教老师评判, 也可以让学生专注于学习内容相关的代码工作.

以上. 感谢老师的悉心指导和实验过程中助教的耐心解答!