Cache 替换策略设计与分析实验说明

（一）实验简介

本实验部分源于 “Cache Replacement Championship”（https://www.jilp.org/jwac-1/）希望同学们能够深入理解不同的 Cache 替换策略，动手实现自己的 Cache 替换策略，同时观测不同的替换策略对于程序运行性能的影响。

（二）需要提交的文件

1. CRC/src/LLCsim 目录下的两个文件：

replacement_state.cpp

replacement_state.h

2. 实验报告：

（1）说明不同 Cache 替换策略的核心思想和算法，包括前人提出的和你自己设计实现的。

（2）根据模拟器的实验结果报告在不同替换策略下，Benchmark trace 程序的运行时间、Cache 命中率等情况。

（3）分析自己设计实现的 Cache 替换策略在自己制作的以及 Benchmark 中给出的一系列特定 trace 下的性能表现情况，并讨论原因。

（4）鼓励同学们探索近年来新提出的 Cache 替换算法，制作有代表性的 trace，设计有创新性的替换策略，做到以上三点之一的同学可获得额外加分。

（三）实验流程

1. 实验环境配置：

本实验中的模拟器需要较老版本的 OS 内核，目前已知模拟器可以在 Ubuntu 10.04 运行，不能在 Ubuntu 14.04 运行，其它系统对模拟器的支持情况还不清楚。

考虑到 Ubuntu 10.04 的环境要求过于苛刻，我们推荐大家使用 Vagrant 这一环境配置工具，同时为大家准备了配置所需的 Vagrantfile，同学们只需下载 Virtual box（https://www.virtualbox.org）和 Vagrant（https://www.vagrantup.com）按照 Vagrant 官方网站的指导就可以通过实验目录下的 Vagrantfile 自动配置 Unbuntu 10.04 的 Virtual Box 虚拟机，具体流程如下：

1. 安装 Vagrant 之后进入到包含了我们提供的 Vagrantfile 的目录下执行 vagrant up（确保已安装了 Virtual box 或其他虚拟机软件）
2. Vagrantfile 脚本将自动下载 Ubuntu 10.04 镜像（这可能会消耗一部分你的流量），并在 Virtual box 中配置 Ubuntu 10.04
3. 在这个过程中可能会遇见“Implementation of the USB 2.0 controller not found!”的错误，只需在 VirtualBox settings -> ports -> USB 中禁用 USB 功能即可。

4. 进入 Virtual Box 软件，运行 Ubuntu 10.04 虚拟机，用户名密码均为：vagrant
5. 根目录下有 CRC_VAGRANT.tar.gz，解压后进入文件夹中，之后的操作就按照文件夹中的 README 的指示编译运行即可。

如果有些同学无法使用 Virtual box，Vagrant 也支持其他虚拟机软件，相关配置方法参见 Vagrant 官网上的配置文档。

2. 模拟器配置：

模拟器文件在 CRC_VAGRANT.tar.gz 压缩包下，解压后的文件夹内有 README 文件，对应在 Running the Simulator 部分说明了运行模拟器的具体方法。

例如，在 64 位系统上使用之前制作的 hello_world 程序的 trace（制作方法后面会介绍）运行模拟器，可以通过如下的方法：

```
../bin/CMPsim.usetrace.64 \
-threads 1 \
-t ../traces/hello_world.out.trace.gz \
-o hello_world.stats \
-cache UL3:1024:64:16 -LLCrepl 0
```

“-cache UL3:1024:64:16”中：

“L3”表示实验在第 3 级 Cache 上进行；“U”表示统一存储指令和数据；“1024”表示 Cache 大小为 1024KB；“64”表示 Cache 行大小为 64B；“16”表示 16 路组相连；“-LLCrepl 0”的取值与 Cache 替换算法实现的位置有关，更详细的信息请参见 README。

本次实验只要求测试 Cache 替换策略在单线程 benchmark 下的性能。所有实验参数和实验结果均记录在输出文件 *.stats.gz 中，请在实验报告中报告这些信息。

3. trace

trace 是指一段真实程序运行情况的记录，本实验中测试 Cache 替换策略性能的 trace 有以下两种类型：

（1）你自己制作的 trace：

任何能够在你的机器上运行的程序都可以用来制作 trace，README 的 Generating Traces to Simulate 部分说明了将可执行程序制作成 trace 的方法。例如，在 64 位系统上制作 hello_world 程序的 trace，可通过以下方法（确保 hello_world 的可执行文件在当前目录下）：

```
pinkit/pin-2.7-31933-gcc.3.4.6-ia32_intel64-linux/pin \
-t ./bin/CMPsim.gentrace.64 \
-threads 1 \
-o traces/hello_world.out \
```

-- ./hello_world

（2）统一的 benchmark trace：

下载链接：https://cloud.tsinghua.edu.cn/d/9099b07af8e74f25b86b、下载密码：goodluck

本次实验只要求使用单线程程序制作的 trace，你需要在实验报告中分别分析各类 Cache 替换策略在统一的 benchmark 下和至少一个你自己制作的 trace 下的性能表现。（注意：自己制作的 trace 应当有一定的访存量，否则将无法体现出不同 Cache 替换策略的性能差异）

4. 设计 Cache 替换策略

该部分是本次实验的重点，只需修改模拟器 CRC/src/LLCsim 目录下的 replacement_state.cpp 和 replacement_state.h 两个文件，其它文件请不要改动（具体编译方法请参考 README 中的“Writing Your Own Replacement Algorithm”部分）你需要完成如下两项工作：

（1）实现已经提出的 Cache 替换策略：

模拟器已经实现了一个 LRU 替换策略供大家参考，在此基础上请同学阅读相关论文，设计实现前人已经提出过的 Cache 替换策略。

可参考 https://www.jilp.org/jwac-1/JWAC-1%20Program.htm，但这些都是 2010 年提出的策略，年代已经相对久远。因此鼓励大家查找阅读近些年来新提出的讨论 Cache 替换策略的论文，并尝试实现。如果策略过于复杂超出了我们实验框架限制的范围，也请在实验报告中予以介绍。

（2）设计实现自己的 Cache 替换策略

设计实现自己的 Cache 替换策略，并在实验报告中分析自己所设计的 Cache 替换策略的优缺点，最好能结合一个你自己制作的比较有代表性 trace。