ICS Lab4-Cachelab

> deadline 2023.11.29

实验简介

前言

CSAPP第6章配套实验。
本实验的目的是加深同学们对高速缓存cache认识。实验分为三个部分：

- part A：用c语言设计一个cache模拟器，它能读入特定格式的trace文件（trace文件中模拟了一系列的对存储器的读写操作），并且输出cache的命中、缺失、替换次数；我们会为你提供一部分代码
- part B：根据特定的cache参数设计一个矩阵转置的算法，使得矩阵转置运算中cache的miss次数尽可能低。
- part C（honor part）：分为两部分
  - 继续优化part B的矩阵转置算法
  - 同part B，不过设计一个矩阵乘法的算法

本次实验参考CMU CSAPP课程的Cache Lab。
考虑到pj将至，助教将本次lab的难度相较于原版调低了一些（除了honor-part，但honor-part的分数很少），而且本次实验全程用c语言（可以不用和抽象的汇编打交道了），所以大家不用过于担心~~~

分值分配

- part A: `40%`
- part B: `36%`
- part C (honor part): `9%`
- 实验报告+代码风格：`15%`

部署实验环境

（1）下载

从 `elearning` 下载 `cachelab-handout.tar` 。这是一个 `tar` 文件，需要对其解包。
（ubuntu 虚拟机 or WSL）打开终端，进入到上述文件对应的目录下，然后执行如下命令：

```
tar -xvf cachelab-handout.tar
```

会在当前目录解包出一个 `cachelab-handout` 文件夹，其中的内容就是本次实验用到的的文件了。

（2）准备工作

确保已安装了 gcc

在终端中检查是否安装了 gcc：

```
gcc -v
```

如果已安装，终端将会反馈版本信息，否则会反馈 `command not found` 。
如未安装，尝试执行以下命令进行安装：

```
sudo apt-get install gcc
```

确保已安装了 make

检查是否安装 make，在终端输入：

```
make -v
```

同理，如未安装，尝试以此执行以下命令：

```
sudo apt-get update
sudo apt-get install make
sudo apt-get install libc6 libc6-dev libc6-dev-i386
```

确保安装python

```
python --version
```

一般情况下系统是自带python的
如未安装，请自行上网搜索安装教程

安装valgrind

```
sudo apt-get install valgrind
```

part A

intro

设计一个cache模拟器，读入指定格式的trace文件，模拟cache的运行过程，然后输出cache的命中、缺失、替换次数
trace文件是通过`valgrind`的lackey工具生成的，它具有以下格式

```
I 0400d7d4,8
 M 0421c7f0,4
 L 04f6b868,8
 S 7ff0005c8,8
```

每行格式为

```
[space]operation address,size
```

其中`I`代表读指令操作，`L`代表读数据操作，`S`代表写数据操作，`M`代表修改数据操作（即读数据后写数据）。**除了`I`操作外，其他操作都会在开头都会有一个空格。**`address`为操作的地址，`size`为操作的大小（单位为字节）。

to-do

你的所有实现都在`csim.c`和`csim.h`中
你的全局变量和函数需要定义在`csim.h`中，你的函数实现需要在`csim.c`中
我们提供了一个`csim-ref`的文件，是一个参考实现，你可以通过它来检查你的实现是否正确，它的用法如下：

```
./csim-ref [-hv] -s <s> -E <E> -b <b> -t <tracefile>
```

- `-h`代表帮助
- `-v`代表verbose，即输出详细信息
- `-s`代表cache的set数
- `-E`代表每个set中的cache line数
- `-b`代表cache line的大小（单位为字节）
- `-t`代表trace文件的路径

`csim-ref`会输出cache的命中、缺失、替换次数，比如：

```
$ ./csim-ref -s 16 -E 1 -b 16 -t traces/yi.trace
hits:4 misses:5 evictions:3
```

verbose模式：

```
$ ./csim-ref -v -s 16 -E 1 -b 16 -t traces/yi.trace
L 10,1 miss
M 20,1 miss hit
L 22,1 hit
S 18,1 hit
L 110,1 miss eviction
L 210,1 miss eviction
```

```
M 12,1 miss eviction hit
hits:4 misses:5 evictions:3
```

你的实现需要具有和`csim-ref`相同的功能，包括verbose模式输出debug信息
在`csim.c`中，我们已经为你提供了基本的解析命令行参数的代码，你需要在此基础上进行实现
cache的替换策略为`LRU`算法

requirements

- 你的代码在编译时不能存在warning
- 你 **只能** 使用c语言来实现（助教看不懂c++和python）
- 虽然给了测试数据，但不允许面向数据编程，助教会做源码检查；不允许通过直接调用`csim-ref`来实现

evaluation

共有8项测试

```
./csim -s 1 -E 1 -b 1 -t traces/yi2.trace
./csim -s 4 -E 2 -b 4 -t traces/yi.trace
./csim -s 2 -E 1 -b 4 -t traces/dave.trace
./csim -s 2 -E 1 -b 3 -t traces/trans.trace
./csim -s 2 -E 2 -b 3 -t traces/trans.trace
./csim -s 2 -E 4 -b 3 -t traces/trans.trace
./csim -s 5 -E 1 -b 5 -t traces/trans.trace
./csim -s 5 -E 1 -b 5 -t traces/long.trace
```

原始分为：前7项每项`3`分，最后一项`6`分，共`27`分；对于每一项，`hit, miss, eviction`的正确性各占`1/3`的分数
原始分将会被乘以`40/27`得到最终的分数
最终的分数可以通过`./driver.py`来查看

hints

- 使用`malloc`和`free`来构造cache
- 你可以使用`csim-ref`来检查你的实现是否正确，通过开启verbose模式可以更好地debug
- `LRU`算法可以简单地使用计数器的实现方式
- 对于具体如何实现没有太多要求，大家八仙过海各显神通~~~

part B

intro

cache为何被称为“高速缓存”，是因为读取cache的速率远快于读取主存的速率（可能大概100倍），因此cache miss的次数往往决定了程序的运行速度。因此，我们需要尽可能设计cache-friendly的程序，使得cache miss的次数尽可能少。
在这部分的实验，你将对矩阵转置程序（一个非常容易cache miss的程序）进行优化，让cache miss的次数尽可能少。你的分数将由cache miss的次数决定

to-do

你的所有实现都将在`trans.c`中
你将设计这样的一个函数：它接收四个参数：M，N，一个N * M的矩阵A和一个M * N的矩阵B，你需要把A转置后的结果存入B中。

```
char trans_desc[] = "some description";
void trans(int M, int N, int A[N][M], int B[M][N])
{
    // your implementation here
}
```

每设计好一个这样的函数，你都可以在`registerFunctions()`中为其进行“注册”，只有“注册”了的函数才会被加入之后的评测中，你可以“注册”并评测多个函数；为上面的函数进行注册只需要将下面代码加入`registerFunctions()`中

```
registerTransFunction(trans, trans_desc);
```

我们提供了一个名为`trans()`的函数作为示例
你需要保证有一个且有唯一一个“注册”的函数用于最终提交，我们将靠“注册”时的description进行区分，请确保你的提交函数的description是“Transpose submission”，比如

```
char transpose_submit_desc[] = "Transpose submission";
void transpose_submit(int M, int N, int A[N][M], int B[M][N])
{
    // your implementation here
}
```

我们将使用特定形状的矩阵和特定参数的cache来进行评测，所以你 **可以** 针对这些特殊情况来编写代码

requirements

- 你的代码在编译时不能存在warning
- 在每个矩阵转置函数中，你至多能定义12个int类型的局部变量（不包括循环变量，但你不能将循环变量用作其他用途），且不能使用任何全局变量。你不能定义除int以外类型的变量。你不能使用malloc等方式申请内存块。你可以使用int数组，但等同于数组大小的数量的int类型变量也同样被计入
- 你不能使用递归
- 你只允许使用一个函数完成矩阵转置的功能，而不能在函数中调用任何辅助函数
- 你不能修改原始的矩阵A，但是你可以任意修改矩阵B
- 你可以定义宏

evaluation

我们将使用cache参数为：`s = 48, E = 1, b = 48`，即每个cache line大小为48字节，共有48个cache line，每个set中只有1个cache line。

我们将使用以下3种矩阵来进行评测

- 48 * 48的矩阵，分值12分，miss次数 < 500则满分，miss次数 > 800则0分，500~800将按miss次数获取一定比例的分数
- 96 * 96的矩阵，分值12分，miss次数 < 2200则满分，miss次数 > 3000则0分，2200~3000将按miss次数获取一定比例的分数
- 93 * 99的矩阵，分值12分，miss次数 < 3000则满分，miss次数 > 4000则0分，3000~4000将按miss次数获取一定比例的分数
- 荣誉分4分，将在荣誉部分介绍

我们只会针对这三种矩阵进行测试，所以你 **可以** 只考虑这三种情况

step 0

```
make clean && make
```

step 1

在测试之前，进行算法正确性的测试

```
./tracegen -M <row> -N <col>
```

比如对48 * 48转置函数进行测试

```
./tracegen -M 48 -N 48
```

你也可以对特定的函数进行测试，比如对第0个“注册”的函数

```
./tracegen -M 48 -N 48 -F 0
```

step 2

```
./test-trans -M <row> -N <col>
```

这个程序将使用valgrind工具生成trace文件，然后调用csim-ref程序获取cache命中、缺失、替换的次数

hints

- 在调用./test-trans之后，可以使用如下命令查看你的cache命中/缺失情况；你可以把f0替换为fi来查看第 i 个“注册”的函数带来的cache命中/缺失情况

```
./csim-ref -v -s 48 -E 1 -b 48 -t trace.f0 > result.txt
```

- 这篇文章可能对你有所启发
- cache的关联度为1，你可能需要考虑冲突带来的miss
- 脑测一下你的miss次数或许是一个很好的选择，你可以计算一下大概有多少比例的miss，然后乘以总的读写次数；你可以在上面生成的`result.txt`文件中验证你的想法
- 你可以认为A和B矩阵的起始地址位于某个cacheline的开始（即A和B二维数组的起始地址能被48整除）

part C --honor part

> warning: 本部分较难，可能花费比较多的时间，但是分值较低，请自行平衡付出时间的收益

1

- （2分）在part B中，将48 * 48的矩阵转置情况的cache miss次数优化到`450`次以下
- （2分）在part B中，将96 * 96的矩阵转置情况的cache miss次数优化到`1900`次以下

2

intro

同part B，但是需要实现一个矩阵乘法算法
cache参数：`s = 32, E = 1, b = 32`
评测矩阵：A：32 * 32；B：32 * 32

to-do

```
cd honor-part
```

你的所有实现都将在`mul.c`中
实现以下函数，将A * B的结果存入C中

```
char mul_desc[] = "some description";
void mul(int M, int N, int A[N][M], int B[M][N], int C[N][N])
{
    // your implementation here
}
```

并在`registerFunctions()`“注册”，步骤同part B

requirements

同part B，但是你可以定义至多16个int类型的局部变量（不包括循环变量，但你不能将循环变量用作其他用途）；你不能修改原始的矩阵A和B，但是你可以任意修改矩阵C

evaluation

你将获得附加分5分当你的cache miss次数 `< 4000`

step 0

```
make clean && make
```

step 1

在测试之前，进行算法正确性的测试

```
./tracegen -M 32 -N 32
```

step 2

```
./test-mul -M 32 -N 32
```

评分

在项目根目录下

```
./driver.py
```

注意请保证在项目根目录和`./honor-part`目录下都已经`make`过了

提交实验

（1）内容要求

你需要提交：

- csim.c
- csim.h
- trans.c
- mul.c（如果完成了的话）
- 一份实验报告

实验报告应该包含以下内容：

- 实验标题，你的姓名，学号。
- 你在终端中执行`./driver.py`后的截图。
- 描述你每个部分实现的思路，要求简洁清晰。
- 如果有，请务必在报告中列出引用的内容以及参考的资料。
- 对本实验的感受（可选）。
- 对助教们的建议（可选）。

（2）格式要求

可提交`.md`文件或者`.pdf`文件。不要提交`.doc`或`.docx`文件。
将所有代码文件和实验报告打包成`tar`文件。将其命名为`<学号>.tar`

参考资料

- 原版Cache Lab
- C语言处理参数的 getopt() 函数