15-213/18-213,2012 年秋季

缓存实验室:了解缓存内存

分配时间:2012 年 10 月 2 日星期二

截止日期:10 月 11 日星期四晚上 11:59

最后提交时间:10 月 14 日星期日晚上 11:59

1 物流

这是一个个人项目。您必须在 64 位 x86-64 计算机上运行此实验。

特定于站点:在此处插入任何其他后勤项目,例如如何寻求帮助。

2 概述

本实验将帮助您了解高速缓存对 C 语言性能的影响
程式。

该实验室由两部分组成。在第一部分中,您将编写一个小型 C 程序（大约 200-300 行)来模拟高速缓存的行为。在第二部分中,您将优化一个小矩阵转置函数,目标是最大限度地减少缓存未命中次数。

3 下载作业

特定于站点:在此处插入一段内容,解释讲师如何将cachelab-handout.tar文件分发给学生。

首先将cachelab-handout.tar 复制到您计划在其中进行工作的受保护的Linux 目录。然后给出命令

```
linux> tar xvf cachelab-handout.tar
```

这将创建一个名为cachelab-handout 的目录,其中包含许多文件。您将修改两个文件:csim.c 和 trans.c。要编译这些文件,请键入:

```
Linux> 清理干净
linux > 制作
```

警告:不要让 Windows WinZip 程序打开您的 .tar 文件（许多 Web 浏览器都设置为自动执行此操作）。相反,请将文件保存到 Linux 目录并使用 Linux tar 程序提取文件。一般来说,对于此类,您不应该使用 Linux 以外的任何平台来修改您的文件。这样做可能会导致数据丢失（以及重要的工作！）。

4 说明

该实验室有两个部分。在 A 部分中,您将实现一个缓存模拟器。在 B 部分中,您将编写一个针对缓存性能进行优化的矩阵转置函数。

4.1 参考跟踪文件

讲义目录的 Traces 子目录包含参考跟踪文件的集合,我们将使用它们来评估您在 A 部分中编写的缓存模拟器的正确性。跟踪文件由名为 valgrind 的 Linux 程序生成。例如,输入

```
linux> valgrind --log-fd=1 --tool=lackey -v --trace-mem=yes ls -l
```

在命令行上运行可执行程序“ls -l”,按发生的顺序捕获每次内存访问的跟踪,并将它们打印在标准输出上。

Valgrind 内存痕迹具有以下形式：

```
我 0400d7d4,8
 中号 0421c7f0,4
 L 04f6b868,8
 S 7ff0005c8,8
```

每行表示一次或两次内存访问。每行的格式为

[空格]操作地址,大小

操作字段表示存储器访问的类型：“I”表示指令加载，“L”表示数据加载，“S”表示数据存储，“M”表示数据修改（即数据加载后跟随数据存储）。每个“我”之前都没有空格。每个“M”、“L”和“S”之前始终有一个空格。地址字段指定 64 位十六进制内存地址。大小字段指定操作访问的字节数。

4.2 A 部分:编写缓存模拟器

在 A 部分中,您将在 csim.c 中编写一个缓存模拟器,该模拟器将 valgrind 内存跟踪作为输入,模拟该跟踪上缓存的命中/未命中行为,并输出命中、未命中和逐出的总数。

我们为您提供了参考缓存模拟器的二进制可执行文件,称为 csim-ref,它模拟 valgrind 跟踪文件上具有任意大小和关联性的缓存的行为。在选择要逐出的缓存行时,它使用 LRU（最近最少使用)替换策略。

参考模拟器采用以下命令行参数：

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

- -h:打印使用信息的可选帮助标志
- -v:显示跟踪信息的可选详细标志
- -s <s>:集合索引位数（S = 2s是集合数）
- -E <E>:关联性（每组行数）
- -b <b>:块位数（B = 2b是块大小）
- -t <tracefile>:要重放的 valgrind 跟踪的名称

命令行参数基于CS:APP2e 教科书第 597 页中的符号（s、 E 和b）。例如：

```
linux> ./csim-ref -s 4 -E 1 -b 4 -t traces/yi.trace 命中:4 未命中:5 驱逐:3
```

详细模式下的相同示例：

```
linux> ./csim-ref -v -s 4 -E 1 -b 4 -t traces/yi.trace L 10,1 错过 M 20,1 错过命中 L 22,1 命中 S 18,1
命中 L 110,1未命中
驱逐 L 210,1 未命中驱逐 M
12,1 未命中驱逐
命中次数:4 未命
中:5 驱逐:3
```

A 部分的工作是填写 csim.c 文件,以便它采用相同的命令行参数并生成与参考模拟器相同的输出。请注意,该文件几乎完全是空的。

您需要从头开始编写它。

A 部分的编程规则

- 在csim.c 的标题注释中包含您的姓名和登录ID。

- 您的csim.c 文件编译时必须不发出警告才能获得积分。
- 您的模拟器必须对任意s、 E 和b 正确工作。这意味着您需要使用 malloc 函数为模拟器的数据结构分配存储空间。键入“man malloc”以获取有关此函数的信息。
- 对于本实验,我们只对数据缓存性能感兴趣,因此您的模拟器应忽略所有指令缓存访问（以“I”开头的行）。回想一下,valgrind 总是将“I”放在第一列中（前面没有空格）,将“M”、“L”和“S”放在第二列中（前面有空格）。这可以帮助您解析跟踪。
- 要获得 A 部分的学分,您必须调用函数 printSummary,并提供 A 部分的总数。命中、未命中和驱逐,在主函数的末尾：

```
printSummary(hit_count, miss_count, eviction_count);
```

- 对于本实验,您应该假设内存访问已正确对齐,以便单个内存访问永远不会跨越块边界。通过做出此假设,您可以忽略 valgrind 跟踪中的请求大小。

4.3 B部分:优化矩阵转置

在 B 部分中,您将在 trans.c 中编写一个转置函数,该函数会导致尽可能少的缓存未命中。

设A表示矩阵， Aij表示第i行j列的分量。 A的转置,表示为AT，是一个矩阵,使得Aij = AT 来自。

为了帮助您入门,我们在 trans.c 中为您提供了一个示例转置函数,用于计算N × M矩阵A的转置并将结果存储在M × N矩阵B 中：

```
char trans_desc[] =  简单逐行扫描转置  ; void trans(int M, int N, int A[N][M], int B[M][N])
```

示例转置函数是正确的,但效率低下,因为访问模式会导致相对较多的缓存未命中。

您在 B 部分中的工作是编写一个类似的函数,称为 transpose_submit,它可以最大限度地减少数量不同大小矩阵的缓存未命中数：

```
char transpose_submit_desc[] =  转置提交  ; void transpose_submit(int M, int N, int A[N]
[M], int B[M][N]);
```

不要更改transpose_submit 函数的描述字符串（“转置提交”）。自动评分器搜索此字符串以确定要评估哪个转置函数以获得学分。

B 部分的编程规则

- 在trans.c 的标题注释中包含您的姓名和登录ID。
- 您在trans.c 中的代码必须在没有警告的情况下进行编译才能获得积分。
- 每个转置函数最多可以定义 12 个 int 类型的局部变量。1
- 不允许使用任何 long 类型的变量或使用
  将多个值存储到单个变量的任何位技巧。
- 您的转置函数可能不使用递归。
- 如果您选择使用辅助函数,则辅助函数和顶级转置函数之间堆栈上的局部变量不得同时超过 12 个。例如,如果您的转置声明了 8 个变量,然后您调用一个使用 4 个变量的函数,该函数调用另一个使用 2 个变量的函数,则堆栈上将有 14 个变量,并且您将违反规则。
- 您的转置函数可能不会修改数组 A。但是,您可以对
  数组 B 的内容。
- 您不得在代码中定义任何数组或使用malloc 的任何变体。

5 评价

本节描述如何评估您的工作。本实验满分为 60 分:

- A 部分:27 分
- B 部分:26 分
- 风格:7 分

5.1 A部分评估

对于 A 部分,我们将使用不同的缓存参数和跟踪来运行您的缓存模拟器。有八个测试用例,每个测试用例得分 3 分,除了最后一个测试用例得分 6 分:

```
linux> ./csim -s 1 -E 1 -b 1 -t 痕迹/yi2.trace linux> ./csim -s 4 -E 2 -b 4 -t 痕迹/yi.trace
linux> ./csim -s 2 -E 1 -b 4 -t 痕迹/dave.trace linux> ./csim -s 2 -E 1 -b 3 -t 痕迹/
trans.trace linux> ./csim -s 2 -E 2 -b 3 -t痕迹/trans.trace
```

1这个限制的原因是我们的测试代码无法计算对堆栈的引用。我们希望你限制你的
对堆栈的引用并重点关注源数组和目标数组的访问模式。

```
linux> ./csim -s 2 -E 4 -b 3 -t 痕迹/trans.trace linux> ./csim -s 5 -E 1 -b 5 -t 痕迹/
trans.trace linux> ./csim -s 5 -E 1 -b 5 -t 痕迹/long.trace
```

您可以使用参考模拟器 csim-ref 来获取每个测试用例的正确答案。

在调试过程中,使用 -v 选项详细记录每次命中和未命中的情况。

对于每个测试用例,输出正确的缓存命中、未命中和驱逐次数将为您提供该测试用例的满分。您报告的每个命中、未命中和逐出次数都相当于该测试用例的 1/3 功劳。也就是说,如果一个特定的测试用例值得 3 分,并且您的模拟器输出正确的命中和未命中次数,但报告错误的驱逐次数,那么您将获得 2 分。

5.2 B 部分的评估

对于 B 部分,我们将在三个不同大小的输出矩阵上评估 transpose_submit 函数的正确性和性能:

- 32 × 32 （中号= 32,中号= 32）
- 64 × 64 （中号= 64,中号= 64）
- 61 × 67 （中号= 61,中号= 67）

5.2.1 表现（26分）

对于每个矩阵大小,通过使用 valgrind 提取函数的地址跟踪来评估 transpose_submit 函数的性能,然后使用参考模拟器在带有参数(s = 5, E = 1, b = 5）。

每个矩阵大小的性能得分与未命中数m 线性缩放,直至达到某个阈值:

- 32 × 32:如果m < 300,则为 8 分;如果m > 600 ,则为 0 分
- 64 × 64:如果m < 1、300 ,则为 8 点;如果m > 2、000,则为0点
- 61 × 67:如果m < 2, 000,则为 10 分;如果m > 3, 000 ,则为 0 分

您的代码必须正确才能获得特定大小的任何性能点。您的代码只需要针对这三种情况是正确的,并且您可以专门针对这三种情况对其进行优化。特别是,您的函数完全可以显式检查输入大小并实现针对每种情况优化的单独代码。

5.3 风格评价

编码风格有 7 点。这些将由课程工作人员手动分配。风格指南
可以在课程网站上找到。

课程工作人员将检查 B 部分中的代码是否存在非法数组和过多的局部变量。

6 在实验室工作

6.1 处理 A 部分

我们为您提供了一个名为 test-csim 的自动评分程序，用于测试您的评分的正确性
缓存模拟器上的参考痕迹。请务必在运行测试之前编译模拟器：

```
linux > 制作
Linux> ./测试-csim
                                        你的模拟器                  参考模拟器
点 (s,E,b) 3 (1,1,1) 3 (4,2,4)          命中未驱逐                  命中未驱逐
      3 (2,1,4) 2 3                     9       8       6       9       8       6 条痕迹/yi2.trace
      (2,1,3) 167 3                     4       5       2       4       5       2 条迹线/do.trace
      (2,2,3) 201 3 (2,4,3) 212 3 (5,1,5) 231 6 (5,1,5) 265189 21775 21743 265189
                                                                                1 条痕迹/dave.trace
      21775 21743 迹线/long.trace       2       3       1       3       67 条迹线/事务迹线
                                        167     71      67      71      29 条迹线/事务迹线
                                        201     37      29      37      10 条迹线/事务迹线
                                        212 231 26 7    10 0    26 7    0 条迹线/事务迹线
```

27

对于每个测试，它都会显示您获得的分数、缓存参数、输入跟踪文件和
比较您的模拟器和参考模拟器的结果。

以下是有关 A 部分工作的一些提示和建议：

- 对小跟踪进行初始调试，例如traces/dave.trace。
- 参考模拟器采用可选的-v 参数来启用详细输出，显示
  由于每次内存访问而发生的命中、未命中和逐出。您不需要
  在您的 csim.c 代码中实现此功能，但我们强烈建议您这样做。它会
  通过允许您直接将模拟器的行为与参考进行比较来帮助您进行调试
  模拟器上的参考跟踪文件。
- 我们建议您使用getopt 函数来解析命令行参数。你会
  需要以下头文件：

```
#include <getopt.h>
#include <stdlib.h>
#include <unistd.h>
```

有关详细信息,请参阅“man 3 getopt”。

- 每个数据加载(L) 或存储(S) 操作最多可导致一次高速缓存未命中。数据修改操作 (M) 被视为加载,然后存储到同一地址。因此,M 操作可能导致两次高速缓存命中,或者一次未命中和一次命中加上可能的逐出。

- 如果您想使用15-122 的C0 风格合约,您可以包含contracts.h,我们将其包含在内为了您的方便,已在讲义目录中提供。

6.2 B 部分的工作

我们为您提供了一个名为 test-trans.c 的自动评分程序,用于测试您在自动评分器中注册的每个转置函数的正确性和性能。

您可以在 trans.c 文件中注册最多 100 个版本的转置函数。每个转置版本具有以下形式:

```
/* 标题注释 */ char
trans_simple_desc[] =    A simple transpose   ; void trans_simple(int M, int
N, int A[N][M], int B[M][N]) {

        /* 这里是你的转置代码 */
}
```

通过调用以下形式向自动评分器注册特定的转置函数:

```
registerTransFunction(trans_simple, trans_simple_desc);
```

在 trans.c 的 registerFunctions 例程中。在运行时,自动评分器将评估每个注册的转置函数并打印结果。当然,注册的函数之一必须是您要提交以获取学分的 transpose_submit 函数:

```
registerTransFunction(transpose_submit, transpose_submit_desc);
```

有关其工作原理的示例,请参阅默认的 trans.c 函数。

自动分级器将矩阵大小作为输入。它使用 valgrind 生成每个已注册转置函数的跟踪。然后,它通过使用参数(s = 5、 E = 1、 b = 5)在缓存上运行参考模拟器来评估每个跟踪。

例如,要在 32 × 32 矩阵上测试注册的转置函数,请重建 test-trans,然后使用适当的M和N 值运行它:

```
linux > 制作
linux> ./test-trans -M 32 -N 32 第 1 步:评估已注册转置函数
的正确性: func 0(转置提交):正确性:1
```

func 1（简单行扫描转置）：正确性：1 func 2（列扫描转置）：正确性：1 func 3（使用 zig-zag 访问模式）：
正确性：1

步骤 2：为注册的转置函数生成内存跟踪。

步骤 3：评估已注册转置函数的性能 (s=5、E=1、b=5) func 0（转置提交）：命中：1766、未命中：287、驱逐：255 func 1（简单逐行扫描转置）：命中：870，
未命中：1183，逐出：1151 func 2（按列扫描转置）：命中：870，未命中：1183，逐出：1151 func 3（使用锯齿形访问模式）：命中：1076，
未命中：977，驱逐：945

正式提交摘要（func 0）：正确性=1 错过=287

在这个例子中，我们在 trans.c 中注册了四个不同的转置函数。 test-trans 程序测试每个注册的函数，显示每个函数的结果，并提取结果以供正式提交。

以下是有关 B 部分工作的一些提示和建议。

- test-trans 程序将函数i的跟踪保存在文件trace.fi 中。[2] 这些跟踪文件是非常宝贵的调试工具，可以帮助您准确了解每个转置函数的命中和未命中来自何处。要调试特定函数，只需使用 verbose 选项通过参考模拟器运行其跟踪即可：

  linux> ./csim-ref -v -s 5 -E 1 -b 5 -t trace.f0 S 68312c,1 未命中 L 683140,8 未命中 L
  683124,4 命中 L 683120,4
  命中 L 603124,4 未命中 S
  6431a0,4 小姐

  ...

- 由于您的转置函数是在直接映射缓存上进行评估的，因此冲突未命中是一个潜在的问题。考虑一下代码中发生冲突遗漏的可能性，尤其是沿对角线。尝试考虑可以减少这些冲突未命中次数的访问模式。

- 阻塞是减少高速缓存未命中的有用技术。看

  http://csapp.cs.cmu.edu/public/waside/waside-blocking.pdf

  了解更多信息。

2因为valgrind引入了很多与你的代码无关的堆栈访问，所以我们过滤掉了所有堆栈
从跟踪访问。这就是为什么我们禁止局部数组并限制局部变量的数量。

6.3 综合起来

我们为您提供了一个名为 ./driver.py的驱动程序,它可以对您的模拟器和转置代码执行完整的评估。这与您的讲师用来评估您的交作业的程序相同。驱动程序使用 test-csim 来评估您的模拟器,并使用 test-trans 来评估您在三个矩阵大小上提交的转置函数。然后它会打印您的结果摘要和您获得的积分。

要运行驱动程序,请键入：

```
Linux> ./driver.py
```

7 交作业

每次您在cachelab-handout 目录中键入make 时,Makefile 都会创建一个名为userid-handin.tar 的tarball,其中包含当前的csim.c 和trans.c 文件。

站点特定:在此处插入文本,告诉每个学生如何在学校提交其userid-handin.tar文件。

重要提示:请勿在 Windows 或 Mac 计算机上创建 handin tarball,也不要以任何其他存档格式（例如 .zip、.gzip 或 .tgz 文件)提交文件。