15-213/18-213, Fall 2012

Cache Lab: Understanding Cache Memories

Assigned: Tuesday, October 2, 2012

Due: Thursday, October 11, 11:59PM

Last Possible Time to Turn in: Sunday, October 14, 11:59PM

**1 Logistics**

This is an individual project. You must run this lab on a 64-bit x86-64 machine.

**SITE-SPECIFIC: Insert any other logistical items here, such as how to ask for help.**

这是一个单独的项目。 您必须在64位x86-64计算机上运行此实验。

特定地点：在此处插入其他任何后勤物品，例如如何寻求帮助。

**2 Overview**

This lab will help you understand the impact that cache memories can have on the performance of your C programs.

The lab consists of two parts. In the first part you will write a small C program (about 200-300 lines) that simulates the behavior of a cache memory. In the second part, you will optimize a small matrix transpose function, with the goal of minimizing the number of cache misses.

本实验将帮助您了解缓存对C程序性能的影响。  
  
实验室由两部分组成。 在第一部分中，您将编写一个小的C程序（大约200-300行），该程序模拟高速缓存的行为。 在第二部分中，您将优化一个小型矩阵转置函数，以最大程度地减少高速缓存未命中的次数。

**3 Downloading the assignment**

**SITE-SPECIFIC: Insert a paragraph here that explains how the instructor will hand out the** cachelab-handout.tar **file to the students.**

Start by copying cachelab-handout.tar to a protected Linux directory in which you plan to do your work. Then give the command

现场说明：在此处插入一段，说明教师如何将cachelab-handout.tar文件分发给学生。

首先将cachelab-handout.tar复制到计划在其中进行工作的受保护Linux目录。 然后发出命令

linux> tar xvf cachelab-handout.tar

This will create a directory called cachelab-handout that contains a number of files. You will be modifying two files: csim.c and trans.c. To compile these files, type:

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：

1

linux> make clean

linux> make

**WARNING:** Do not let the Windows WinZip program open up your.tarfile (many Web browsers are setto do this automatically). Instead, save the file to your Linu x directory and use the Linux tar program to extract the files. In general, for this class you should NEVER use any platform other than Linux to modify your files. Doing so can cause loss of data (and important work !).

警告：不要让Windows WinZip程序打开您的.tar文件（许多Web浏览器设置为自动执行此操作）。 而是将文件保存到Linu x目录中，并使用Linux tar程序提取文件。 通常，对于此类，您永远不要使用Linux以外的任何平台来修改文件。 这样做可能会导致数据丢失（和重要的工作！）。

**4 Description**

The lab has two parts. In Part A you will implement a cache simulator. In Part B you will write a matrix transpose function that is optimized for cache performance.

实验室分为两个部分。 在A部分中，您将实现一个缓存模拟器。 在B部分中，您将编写一个针对高速缓存性能优化的矩阵转置函数。

**4.1** **Reference Trace Files**

The traces subdirectory of the handout directory contains a collection of *reference trace files* that we will use to evaluate the correctness of the cache simulator you write in Part A. The trace files are generated by a Linux program called valgrind. For example, typing

270/5000

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

on the command line runs the executable program “ ls -l”, captures a trace of each of its memory accesses in the order they occur, and prints them on stdout.

157/5000

在命令行上运行可执行程序“ ls -l”，按其发生的顺序捕获其每个内存访问的痕迹，并在stdout上打印它们。

Valgrind memory traces have the following form:

Valgrind内存跟踪具有以下形式：

1. 0400d7d4,8

M 0421c7f0,4 L 04f6b868,8 S 7ff0005c8,8

Each line denotes one or two memory accesses. The format of each line is

每行表示一个或两个内存访问。 每行的格式是

[space]operation address,size

The *operation* field denotes the type of memory access: “I” denotes an instru ction load, “L” a data load, “S” a data store, and “M” a data modify (i.e., a data load follo wed by a data store). There is never a space before each “I”. There is always a space before each “M”, “L”, and “S”. The *address* field specifies a 64-bit hexadecimal memory address. The *size* field specifies the number of bytes accessed by the operation.

操作字段表示内存访问的类型：“ 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，按a表示的数据加载） 数据存储）。 每个“ I”之前都没有空格。 每个“ M”，“ L”和“ S”之前总是有一个空格。 地址字段指定64位十六进制内存地址。 size字段指定操作访问的字节数。

**4.2** **Part A: Writing a Cache Simulator**

In Part A you will write a cache simulator in csim.c that takes a valgrind memory trace as input, simulates the hit/miss behavior of a cache memory on this trace, and outputs the total number of hits, misses, and evictions.

223/5000

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

2

We have provided you with the binary executable of a *reference cache simulator*, called csim-ref, that simulates the behavior of a cache with arbitrary size and associativity on a valgrind trace file. It uses the LRU (least-recently used) replacement policy when choosing which cache line to evict.

我们为您提供了参考高速缓存模拟器的二进制可执行文件，称为csim-ref，它可在valgrind跟踪文件上模拟具有任意大小和关联性的高速缓存的行为。 在选择逐出哪个缓存行时，它使用LRU（最近使用）替换策略。

The reference simulator takes the following command-line arguments:

参考模拟器采用以下命令行参数：

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

* -h: Optional help flag that prints usage info可选的帮助标志，用于打印使用情况信息
* -v: Optional verbose flag that displays trace info显示跟踪信息的可选详细标志
* -s <s>: Number of set index bits (S = 2s is the number of sets)设置的索引位数（S = 2s是设置的数量）
* -E <E>: Associativity (number of lines per set)关联性（每组行数）
* -b <b>: Number of block bits (B = 2b is the block size)块位数（B = 2b是块大小）
* -t <tracefile>: Name of the valgrind trace to replay要重播的valgrind跟踪的名称

The command-line arguments are based on the notation (S, E, and B) from page 597 of the CS:APP2e textbook. For example:

119/5000

命令行参数基于CS：APP2e教科书中第597页的符号（S，E和B）。 例如：

linux> ./csim-ref -s 4 -E 1 -b 4 -t traces/yi.trace

hits:4 misses:5 evictions:3

The same example in verbose mode:

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 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

Your job for Part A is to fill in the produces the identical output as the You'll need to write it from scratch.

csim.c file so that it takes the same command line arguments and reference simulator. Notice that this file is almost completely empty.

对于A部分，您的工作是填写与产生相同的输出，您将需要从头开始编写它。

csim.c文件，以便它使用相同的命令行参数和引用模拟器。 请注意，该文件几乎完全为空。

**Programming Rules for Part A**

* Include your name and loginID in the header comment for csim.c.

65/5000

•在csim.c的标题注释中包含您的名称和loginID。

3

* Your csim.c file must compile without warnings in order to receive credit .

•您的csim.c文件必须在没有警告的情况下进行编译才能获得信誉。

* Your simulator must work correctly for arbitrary S, E, and B. This means that you will need to allocate storage for your simulator's data structures using the malloc function. Type “man malloc” for information about this function.

•您的模拟器必须能够对任意的S，E和B正常工作。这意味着您将需要使用malloc函数为模拟器的数据结构分配存储空间。 键入“ man malloc”以获取有关此功能的信息。

* For this lab, we are interested only in data cache performance, so your simulator should ignore all instruction cache accesses (lines starting with “I”). Reca ll that valgrind always puts “I” in the first

column (with no preceding space), and “M”, “L”, and “S” in the second column (with a preceding space). This may help you parse the trace.

•对于本实验，我们仅对数据高速缓存性能感兴趣，因此您的模拟器应忽略所有指令高速缓存访问（以“ I”开头的行）。 考虑到valgrind始终将“ I”放在首位

列（不带空格），第二列中的“ M”，“ L”和“ S”（带空格）。 这可以帮助您解析跟踪。

* To receive credit for Part A, you must call the function printSummary, with the total number of hits, misses, and evictions, at the end of your main function:

159/5000

•要获得A部分的学分，您必须在主函数末尾调用函数printSummary，其中包含命中，未命中和逐出的总数：

printSummary(hit\_count, miss\_count, eviction\_count);

* For this this lab, you should assume that memory accesses are aligned properly, such that a single memory access never crosses block boundaries. By making this assumption, you can ignore the request sizes in the valgrind traces.

230/5000

•对于本练习，您应该假设内存访问已正确对齐，以使单个内存访问永远不会越过块边界。 通过进行此假设，您可以忽略valgrind跟踪中的请求大小。

**4.3** **Part B: Optimizing Matrix Transpose**

In Part B you will write a transpose function in trans.c that causes as few cache misses as possible.

Let A denote a matrix, and Aij denote the component on the ith row and jth column. The *transpose* of A, denoted AT , is a matrix such that Aij = ATji.

To help you get started, we have given you an example transpose function in trans.c that computes the transpose of N × M matrix A and stores the results in M × N matrix B:

在B部分中，您将在trans.c中编写一个转置函数，该函数将导致尽可能少的高速缓存未命中。

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

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，该函数计算N×M矩阵A的转置并将结果存储在M×N矩阵B中：

char trans\_desc[] = "Simple row-wise scan transpose"; void trans(int M, int N, int A[N][M], int B[M][N])

The example transpose function is correct, but it is ineffici ent because the access pattern results in relatively many cache misses.

Your job in Part B is to write a similar function, called transpose\_submit, that minimizes the number of cache misses across different sized matrices:

示例的转置函数是正确的，但是它效率低下，因为访问模式会导致相对较多的缓存未命中。

在B部分中，您的工作是编写一个类似的函数，称为transpose\_submit，该函数可最大程度地减少不同大小的矩阵之间的高速缓存未命中数：

char transpose\_submit\_desc[] = "Transpose submission";

void transpose\_submit(int M, int N, int A[N][M], int B[M][N]);

Do *not* change the description string (“ Transpose submission”) for your transpose\_submit function. The autograder searches for this string to determine which transpose function to evaluate for credit.

请勿更改您的transpose\_submit函数的描述字符串（“ Transpose提交”）。 自动分频器搜索此字符串，以确定要评估信用的转置函数。

4

**Programming Rules for Part B**

* Include your name and loginID in the header comment for trans.c.
* Your code in trans.c must compile without warnings to receive credit.
* You are allowed to define at most 12 local variables of type int per transpose function.1
* You are not allowed to side-step the previous rule by using any variables of type long or by using any bit tricks to store more than one value to a single variable.
* Your transpose function may not use recursion.

446/5000

•在trans.c的标题注释中包括您的姓名和loginID。  
  
•您在trans.c中的代码必须在没有警告的情况下编译才能获得信誉。  
  
•每个转置函数最多可以定义12个int类型的局部变量。  
  
•不允许您使用long类型的任何变量或使用任何位技巧将多个以上的值存储到单个变量中来规避上一条规则。  
  
•您的转置功能可能不使用递归。

* If you choose to use helper functions, you may not have more than 12 local variables on the stack at a time between your helper functions and your top level transpose function. For example, if your transpose declares 8 variables, and then you call a function which uses 4 variables, which calls another function which uses 2, you will have 14 variables on the stack, and you will be in violation of the rule.
* Your transpose function may not modify array A. You may, however, do whatever you want with the contents of array B.
* You are NOT allowed to define any arrays in your code or to use an y variant of malloc.

•如果选择使用辅助函数，则在辅助函数和顶级转置函数之间，一次最多只能有12个局部变量在堆栈上。 例如，如果您的转置声明了8个变量，然后调用了一个使用4个变量的函数，又调用了一个使用2个变量的函数，则堆栈中将有14个变量，这将违反该规则。

•您的转置函数可能不会修改数组A。但是，您可以对数组B的内容做任何想做的事情。

•不允许在代码中定义任何数组或使用malloc的y变体。

**5 Evaluation**

This section describes how your work will be evaluated. The full score for this lab is 60 points:

* Part A: 27 Points
* Part B: 26 Points
* Style: 7 Points

**5.1** **Evaluation for Part A**

For Part A, we will run your cache simulator using different cache parameters and traces. There are eight test cases, each worth 3 points, except for the last case, which is worth 6 points:

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8个测试用例，每个用3分，最后一个用例除外，用6点：

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

1. The reason for this restriction is that our testing code is not able to count references to the stack. We want you to limit your references to the stack and focus on the access patterns of the source and destination arrays.

5

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

You can use the reference simulator csim-ref to obtain the correct answer for each of these test cases.

During debugging, use the -v option for a detailed record of each hit and miss.

For each test case, outputting the correct number of cache hits, misses and evictions will give you full credit for that test case. Each of your reported number of hits, misses and evictions is worth 1/3 of the credit for that test case. That is, if a particular test case is worth 3 points, and your simulator outputs the correct number of hits and misses, but reports the wrong number of evictions, then you will earn 2 points.

您可以使用参考模拟器csim-ref为每个测试用例获得正确的答案。

在调试期间，请使用-v选项详细记录每次命中和未命中。

对于每个测试用例，输出正确数量的缓存命中，未命中和逐出将使您对该测试用例有充分的了解。 您报告的命中，未命中和逐出的数量中的每一个都占该测试用例的1/3功劳。 也就是说，如果特定测试用例价值3分，并且您的模拟器输出正确的命中率和未命中数，但是报告了错误的驱逐数，那么您将获得2分。

**5.2** **Evaluation for Part B**

For Part B, we will evaluate the correctness and performance of your transpose\_submit function on three different-sized output matrices:

* 32 × 32 (M = 32, N = 32)
* 64 × 64 (M = 64, N = 64)
* 61 × 67 (M = 61, N = 67)

**5.2.1** **Performance (26 pts)**

For each matrix size, the performance of your transpose\_submit function is evaluated by using valgrind to extract the address trace for your function, and then using the reference simulator to replay this trace on a cache with parameters (S = 5, E = 1, B = 5).

Your performance score for each matrix size scales linearly with the number of misses, M, up to some threshold:

* 32 × 32: 8 points if M < 300, 0 points if M > 600
* 64 × 64: 8 points if M < 1, 300, 0 points if M > 2, 000
* 61 × 67: 10 points if M < 2, 000, 0 points if M > 3, 000

Your code must be correct to receive any performance points for a particular size. Your code only needs to be correct for these three cases and you can optimize it specifically for these three cases. In particular, it is perfectly OK for your function to explicitly check for the input sizes and implement separate code optimized for each case.

6

**5.3** **Evaluation for Style**

There are 7 points for coding style. These will be assigned manually by the course staff. Style guidelines can be found on the course website.

The course staff will inspect your code in Part B for illegal arrays and excessive local variables.

编码风格有7分。 这些将由课程人员手动分配。 风格指南可在课程网站上找到。

课程人员将检查B部分中的代码是否存在非法数组和过多的局部变量。

**6 Working on the Lab**

**6.1** **Working on Part A**

We have provided you with an autograding program, called test-csim, that tests the correctness of your cache simulator on the reference traces. Be sure to compile your simulator before running the test:

我们为您提供了一个称为test-csim的自动分级程序，该程序可在参考迹线上测试缓存模拟器的正确性。 在运行测试之前，请确保编译模拟器：

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| linux> | make |  |  |  |  |  |  |  |
| linux> ./test-csim | | |  |  |  |  |  |  |
|  |  |  | Your simulator | | Reference simulator | | |  |
| Points (s,E,b) | | Hits | Misses | Evicts | Hits | Misses | Evicts |  |
| 3 | (1,1,1) | 9 | 8 | 6 | 9 | 8 | 6 | traces/yi2.trace |
| 3 | (4,2,4) | 4 | 5 | 2 | 4 | 5 | 2 | traces/yi.trace |
| 3 | (2,1,4) | 2 | 3 | 1 | 2 | 3 | 1 | traces/dave.trace |
| 3 | (2,1,3) | 167 | 71 | 67 | 167 | 71 | 67 | traces/trans.trace |
| 3 | (2,2,3) | 201 | 37 | 29 | 201 | 37 | 29 | traces/trans.trace |
| 3 | (2,4,3) | 212 | 26 | 10 | 212 | 26 | 10 | traces/trans.trace |
| 3 | (5,1,5) | 231 | 7 | 0 | 231 | 7 | 0 | traces/trans.trace |
| 6 | (5,1,5) | 265189 | 21775 | 21743 | 265189 | 21775 | 21743 | traces/long.trace |
| 27 |  |  |  |  |  |  |  |  |

For each test, it shows the number of points you earned, the cache parameters, the input trace file, and a comparison of the results from your simulator and the reference simulator.

对于每个测试，它都会显示您获得的积分数，缓存参数，输入的跟踪文件，以及模拟器和参考模拟器的结果比较。

Here are some hints and suggestions for working on Part A:

以下是有关A部分工作的一些提示和建议：

* Do your initial debugging on the small traces, such as traces/dave.trace.
* The reference simulator takes an optional -v argument that enables verbose output, displaying the hits, misses, and evictions that occur as a result of each memory access. You are not required to implement this feature in your csim.c code, but we strongly recommend that you do so. It will help you debug by allowing you to directly compare the behavior of your simulator with the reference simulator on the reference trace files.

•对小轨迹（例如traces / dave.trace）进行初始调试。

•参考模拟器采用可选的-v参数，该参数启用详细输出，显示由于每次内存访问而发生的命中，遗漏和逐出。 您不需要在csim.c代码中实现此功能，但是我们强烈建议您这样做。 通过允许您直接将模拟器与参考跟踪文件上的参考模拟器的行为进行比较，它将帮助您进行调试。

* We recommend that you use the getopt function to parse your command line arguments. You'll need the following header files:

•我们建议您使用getopt函数来解析命令行参数。 您将需要以下头文件：

#include <getopt.h>

#include <stdlib.h>

#include <unistd.h>

7

See “ man 3 getopt” for details.

* Each data load (L) or store (S) operation can cause at most one cache miss. The data modify operation
  1. is treated as a load followed by a store to the same address. Thus, an M operation can result in two cache hits, or a miss and a hit plus a possible eviction.
* If you would like to use C0-style contracts from 15-122, you can include contracts.h, which we have provided in the handout directory for your convenience.

有关详细信息，请参见“ man 3 getopt”。

•每个数据加载（L）或存储（S）操作最多可能导致一个高速缓存未命中。 数据修改操作

（M）被视为负载，然后将其存储到相同的地址。 因此，M操作可能导致两个高速缓存命中，或者未命中和命中再加上可能的逐出。

•如果您想使用15-122之间的C0样式的合同，则可以包括为方便起见，我们在讲义目录中提供的contract.h。

**6.2** **Working on Part B**

We have provided you with an autograding program, called test-trans.c, that tests the correctness and performance of each of the transpose functions that you have registered with the autograder.

You can register up to 100 versions of the transpose function in your trans.c file. Each transpose version has the following form:

/\* Header comment \*/

char trans\_simple\_desc[] = "A simple transpose";

void trans\_simple(int M, int N, int A[N][M], int B[M][N])

{

/\* your transpose code here \*/

}

Register a particular transpose function with the autograder by making a call of the form:

registerTransFunction(trans\_simple, trans\_simple\_desc);

in the registerFunctions routine in trans.c. At runtime, the autograder will evaluate each reg-istered transpose function and print the results. Of course, one of the registered functions must be the transpose\_submit function that you are submitting for credit:

registerTransFunction(transpose\_submit, transpose\_submit\_desc);

See the default trans.c function for an example of how this works.

The autograder takes the matrix size as input. It uses valgrind to generate a trace of each registered trans-pose function. It then evaluates each trace by running the reference simulator on a cache with parameters (S = 5, E = 1, B = 5).

For example, to test your registered transpose functions on a 32 × 32 matrix, rebuild test-trans, and then run it with the appropriate values for M and N :

linux> make

linux> ./test-trans -M 32 -N 32

Step 1: Evaluating registered transpose funcs for correctness:

func 0 (Transpose submission): correctness: 1

8

func 1 (Simple row-wise scan transpose): correctness: 1

func 2 (column-wise scan transpose): correctness: 1

func 3 (using a zig-zag access pattern): correctness: 1

Step 2: Generating memory traces for registered transpose funcs.

Step 3: Evaluating performance of registered transpose funcs (s=5, E=1, b=5)

func 0 (Transpose submission): hits:1766, misses:287, evictions:255

func 1 (Simple row-wise scan transpose): hits:870, misses:1183, evictions:1151

func 2 (column-wise scan transpose): hits:870, misses:1183, evictions:1151

func 3 (using a zig-zag access pattern): hits:1076, misses:977, evictions:945

Summary for official submission (func 0): correctness=1 misses=287

In this example, we have registered four different transpose functions in trans.c. The test-trans program tests each of the registered functions, displays the results for each, and extracts the results for the official submission.

Here are some hints and suggestions for working on Part B.

* The test-trans program saves the trace for function I in file trace.fI.2 These trace files are invaluable debugging tools that can help you understand exactly where the hits and misses for each transpose function are coming from. To debug a particular function, simply run its trace through the reference simulator with the verbose option:

linux> ./csim-ref -v -s 5 -E 1 -b 5 -t trace.f0 S 68312c,1 miss

L 683140,8 miss L 683124,4 hit L 683120,4 hit

L 603124,4 miss eviction S 6431a0,4 miss

...

* Since your transpose function is being evaluated on a direct-mapped cache, conflict misses are a potential problem. Think about the potential for conflict mi sses in your code, especially along the diagonal. Try to think of access patterns that will decrease the number of these conflict misses.
* Blocking is a useful technique for reducing cache misses. See

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

for more information.

1. Because valgrind introduces many stack accesses that have nothing to do with your code, we have filtered out all stack accesses from the trace. This is why we have banned local arrays and placed limits on the number of local variables.

9

**6.3** **Putting it all Together**

We have provided you with a *driver program*, called ./driver.py, that performs a complete evaluation of your simulator and transpose code. This is the same program your instructor uses to evaluate your handins. The driver uses test-csim to evaluate your simulator, and it uses test-trans to evaluate your submitted transpose function on the three matrix sizes. Then it prints a summary of your results and the points you have earned.

To run the driver, type:

linux> ./driver.py

**7 Handing in Your Work**

Each time you type make in the cachelab-handout directory, the Makefile creates a tarball, called userid-handin.tar, that contains your current csim.c and trans.c files.

**SITE-SPECIFIC: Insert text here that tells each student how to hand in their** userid-handin.tar **file at your school.**

**IMPORTANT:** Do not create the handin tarball on a Windows or Mac machine, and do not handin files inany other archive format, such as .zip, .gzip, or .tgz files.

10