15-213/18-213, Fall 2012
15-213/18-2132012 年秋

Cache Lab: Understanding Cache Memories
Cache Lab: Understanding Cache memory 缓存实验室: 理解缓存存储器

Assigned: Tuesday, October 2, 2012
分配时间: 2012 年 10 月 2 日，星期二

Due: Thursday, October 11, 11:59PM
截止日期: 10 月 11 日，星期四，11:59 PM

Last Possible Time to Turn in: Sunday, October 14, 11:59PM
最后上交时间: 10 月 14 日，星期天，11:59 PM

1 Logistics
1 物流

This is an individual project. You must run this lab on a 64-bit x86-64 machine.
这是一个单独的项目。您必须在 64 位 x86-64 机器上运行这个实验室。

> **SITE-SPECIFIC: Insert any other logistical items here, such as how to ask for help.**
> **SITE-SPECIFIC: 在这里插入任何其他物流项目，比如如何寻求帮助。**

2 Overview
2 概述

This lab will help you understand the impact that cache memories can have on the performance of your C programs.
这个实验室将帮助你理解缓存内存对 c 程序性能的影响。

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.

实验室由两部分组成。在第一部分中，你将编写一个小 c 程序(大约 200-300 行)来模拟缓存内存的行为。在第二部分，你将优化一个小的矩阵转置函数，目标是最小化缓存错过的数量。

3 Downloading the assignment
3 下载任务

SITE-SPECIFIC: Insert a paragraph here that explains how the instructor will hand out the `cachelab-handout.tar` file to the students.
SITE-SPECIFIC: 在这里插入一个段落，说明教师将如何向学生分发 cachelab-handout.tar 文件。

Start by copying `cachelab-handout.tar` to a protected Linux directory in which you plan to do your work. Then give the command
首先，将 cachelab-handout.tar 复制到一个受保护的 Linux 目录中，然后在这个目录中完成你的工作。然后给出命令

```
linux> tar xvf cachelab-handout.tar
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。你将修改两个文件: csim.c 和 trans.c。要编译这些文件，输入:

```
linux> make clean
Linux > make clean
linux> make
Linux > make
```

WARNING: Do not let the Windows WinZip program open up your `.tar` file (many Web browsers are set to 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 windowswinzip 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，将文件保存到你的 Linu x 目录中，然后使用 Linux tar 程序来解压缩文件。一般来说，对于这个类，你应该永远不要使用 Linux 以外的任何平台来修改你的文件。这样做会导致数据丢失(和重要的工作.

4 Description
描述

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.
实验室有两部分。在 a 部分，你将实现一个缓存模拟器。在 b 部分，你将编写一个矩阵转置函数，该函数为缓存性能进行了优化。

4.1 Reference Trace Files
4.1 参考跟踪文件

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
Handout 目录的 trace 子目录包含一组引用跟踪文件，我们将使用这些文件来评估您在 a 部分中编写的缓存模拟器的正确性。跟踪文件是由一个名为 valgrind 的 Linux 程序生成的。例如，输入

```
linux> valgrind --log-fd=1 --tool=lackey -v --trace-mem=yes ls -l
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`.
在命令行上运行可执行程序" ls-l"，按照每次访问的顺序捕获每次内存访问的跟踪，并在 stdout 上打印它们。

`Valgrind` memory traces have the following form:
`Valgrind` 内存跟踪的形式如下：

```
I 0400d7d4,8
0400d7d4,8
```

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

Each line denotes one or two memory accesses. The format of each line is
每一行表示一个或两个内存访问。每一行的格式为

```
[space]operation address,size
```
[空间]操作地址，大小

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(即数据加载后跟数据存储)。每个“ i”前面从来没有空格。每个“ m”，“ l”和“ s”前面都有一个空格。地址字段指定一个 64 位的十六进制内存地址。Size 字段指定操作访问的字节数。

4.2 Part A: Writing a Cache Simulator
4.2 Part a: Writing a Cache Simulator 4.2 a 部分: 写一个 Cache 模拟器

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.
在 a 部分中，您将在 csim.c 中编写一个缓存模拟器，该模拟器接受 valgrind 内存跟踪作为输入，模拟该跟踪上缓存内存的命中/错过行为，并输出命中、错过和驱逐的总数。

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.
我们为您提供了一个称为 csim-ref 的引用缓存模拟器的二进制可执行文件，该文件模拟了在 valgrind 跟踪文件上具有任意大小和关联性的缓存的行为。它使用 LRU (最近使用最少的)替换策略来选择驱逐哪个缓存行。

The reference simulator takes the following command-line arguments:
引用模拟器接受以下命令行参数:

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

- `-h`: Optional help flag that prints usage info
  `- h: 打印使用信息的可选帮助标志`
- `-v`: Optional verbose flag that displays trace info
  `V: 显示跟踪信息的可选详细标志`
- `-s <s>`: Number of set index bits ($S = 2^s$ is the number of sets)
  `- s < s > : 集合索引位的数目(s = 2s 是集合的数目)`
- `-E <E>`: Associativity (number of lines per set)
  `- e < e > : 关联性(每组的行数)`
- `-b <b>`: Number of block bits ($B = 2^b$ is the block size)
  `- b < b > : 块位数(b = 2b 是块大小)`
- `-t <tracefile>`: Name of the `valgrind` trace to replay
  `- t < 跟踪文件 > : 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:
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 hits:4 misses:5 evictions:3
Linux > ./csim-ref-s 4-e 1-b 4-t trace/yi.trace 命
中: 4 次未命中: 5 次驱逐: 3 次
```

The same example in verbose mode:
详细模式中的相同示例:

```
linux> ./csim-ref -v -s 4 -E 1 -b 4 -t
traces/yi.trace L 10,1 miss
Linux >/csim-ref-v-s 4-e 1-b 4-t trace/yi.trace l
10,1 miss
M 20,1 miss hit
M20，一枪未命中
```

```
L 22,1 hit
L22,1 命中
S 18,1 hit
S 18,1 命中
L 110,1 miss eviction
L110,1 没有被驱逐
L 210,1 miss eviction
L210，一人未被驱逐
M 12,1 miss eviction hit
M12,1 次驱逐失败
hits:4 misses:5 evictions:3
点击率： 4 次未击中： 5 次驱逐： 3 次
```

Your job for Part A is to fill in the `csim.c` file so that it takes the same command line arguments and produces the identical output as the reference simulator. Notice that this file is almost completely empty. You'll need to write it from scratch.
你在 a 部分的工作是填写与你需 C 文件，以便它采用相同的命令行参数和引用模拟器。注意这个文件几乎是空的。
要从头开始写的相同的输出结果。

Programming Rules for Part A
A 部分的编程规则

- Include your name and loginID in the header comment for `csim.c`.
在 csim.c 的头注释中包含您的名字和 loginID。

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

你的 csim.c 文件必须在没有警告的情况下编译才能获得信用。

- 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.

您的模拟器必须对任意的 s、 e 和 b 正确工作。这意味着你需要使用 malloc 函数为模拟器的数据结构分配存储空间。输入“ man malloc”获取关于这个函数的信息。

- 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

对于本实验室，我们只对数据缓存性能感兴趣，因此您的模拟器应该忽略所有指令缓存访问(以“ i”开头的行)。Reca ll that valgrind 总是把“ i”放在第一个

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.

列(前面没有空格) ，第二列中的“ m”、“ l”和“ s”(前面有空格)。这可能有助于你解析跟踪。

- 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:

为了获得 a 部分的学分，您必须在主函数的末尾调用 printSummary 函数，其中包含命中、未命中和驱逐的总数:

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

打印摘要(点击计数，未命中计数，驱逐计数) ;

- 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.

对于这个实验室，你应该假设内存访问是正确对齐的，这样一个单一的内存访问永远不会跨越块边界。通过这个假设，你可以忽略 valgrind 跟踪中的请求大小。

4.3 Part B: Optimizing Matrix Transpose
4.3 b 部分: 优化矩阵转置

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

在 b 部分，你将在 trans.c 中编写一个转置函数，尽可能减少缓存丢失。

Let A denote a matrix, and A_{ij} denote the component on the ith row and jth column. The *transpose* of A, denoted A^T , is a matrix such that $A_{ij} = A^T_{ji}$.

设 a 表示矩阵，Aij 表示第 i 行和第 j 列上的组件。A 的转置，表示 AT，是一个矩阵，使得 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:

为了帮助你入门，我们给你一个 trans.c 中的转置函数的例子，它计算 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])
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.
示例转置函数是正确的，但是它是低效的，因为访问模式导致相对多的缓存丢失。

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:
在 b 部分中，您的工作是编写一个类似的函数，称为 transpose _ submit，该函数将不同大小的矩阵之间的缓存丢失数量最小化:

```
char transpose_submit_desc[] = "Transpose submission";
Char Transpose _ submit _ desc [] = " Transpose submission";
void transpose_submit(int M, int N, int A[N][M], int B[M][N]);
(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.
不要更改转座 _ 提交函数的描述字符串("转座提交")。Autograder 会搜索这个字符串来决定哪个转置函数可以获得学分。

Programming Rules for Part B
B 部分程序设计规则

- Include your name and loginID in the header comment for `trans.c`.
在 trans.c 的头注释中包括你的名字和 loginID。

- Your code in `trans.c` must compile without warnings to receive credit.
C 中的代码必须在没有警告的情况下进行编译才能获得信用。

- You are allowed to define at most 12 local variables of type `int` per transpose function.[1]
每个转置函数最多允许定义 12 个 int 类型的局部变量

- 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.
不允许通过使用 long 类型的任何变量或通过使用任何位技巧将多个值存储到单个变量来绕过前面的规则。

- Your transpose function may not use recursion.
你的转置函数不能使用递归。

- 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.
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的堆栈上一次可能不会有超过 12 个本地变量。例如，如果你的转置声明 8 个变量，然后你调用一个使用 4 个变量的函数，它调用另一个使用 2 的函数，你将在堆栈上有 14 个变量，你将违反规则。

- Your transpose function may not modify array A. You may, however, do whatever you want with the contents of array B.
你的转置函数不能修改数组 a。然而，你可以对数组 b 的内容做任何你想做的事情。

- You are NOT allowed to define any arrays in your code or to use an y variant of `malloc`.
不允许在代码中定义任何数组或使用 malloc 的 y 变体。

5 Evaluation
5 求值

This section describes how your work will be evaluated. The full score for this lab is 60 points:
这一部分描述了如何评价你的工作。这个实验室的满分是 60 分:

- Part A: 27 Points
A 部分: 27 分

- Part B: 26 Points
B 部: 26 分

- Style: 7 Points

风格: 7 分

5.1 Evaluation for Part A
5.1 甲部的评估

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运行缓存模拟器。有 8 个测试用例，每个用例值 3 分，除了最后一个用例值 6 分:

```
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
Linux > ./csim-s 1-e 1-b 1-t trace/yi2.trace
linux > ./csim-s 4-e 2-b 4-t trace/yi.trace
linux > ./csim-s 2-e 1-b 4-t trace/dave.trace
linux > ./csim-s 2-e 1-b 3-t trace/trans.trace
linux > ./csim-s 2-e 2-b 3-t trace/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.
这个限制的原因是我们的测试代码不能计算对堆栈的引用。我们希望你限制你对堆栈的引用，并且关注源数组和目标数组的访问模式。

```
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
Linux  >  ./csim-s  2-e4-b  3-t  trace/trans.trace
linux  >  ./csim-s  5-e  1-b  5-t  trace/trans.trace
linux > ./csim-s 5-e 1-b 5-t trace/long.trace
```

You can use the reference simulator `csim-ref` to obtain the correct answer for each of these test cases.
你可以使用参考模拟器 csim-ref 来获得每个测试用例的正确答案。
During debugging, use the `-v` option for a detailed record of each hit and miss.
在调试过程中，使用 -v 选项来详细记录每次命中和未命中。

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.
对于每个测试用例，输出正确的缓存命中次数、未命中次数和驱逐次数将会给予你对该测试用例的完全信任。你报告的每一次命中、未命中和驱逐的次数都相当于这个测试用例的三分之一。也就是说，如果一个特定的测试用例值 3 分，并且您的模拟器输出正确的命中和未命中的次数，但是报告了错误的驱逐次数，那么您将获得 2 分。

5.2 Evaluation for Part B
5.2 b 部分的评估

For Part B, we will evaluate the correctness and performance of your `transpose_submit` function on three different-sized output matrices:
对于 b 部分，我们将在三个不同大小的输出矩阵上评估你的转置 _ 提交函数的正确性和性能:

- 32 × 32 (M = 32, N = 32)
32 × 32(m = 32，n = 32)

- 64 × 64 (M = 64, N = 64)
64 × 64(m = 64，n = 64)

- 61 × 67 (M = 61, N = 67)
61 × 67(m = 61，n = 67)

5.2.1 Performance (26 pts)
5.2.1 表现(26 分)

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).
对于每个矩阵大小，通过使用 valgrind 提取函数的地址跟踪，然后使用参考模拟器在带有参数(s = 5，e = 1，b = 5)的缓存上重放该跟踪，来评估转置 _ submit 函数的性能。

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

你对每个矩阵大小的性能得分与未命中数 m 呈线性关系，达到某个阈值:

- 32 × 32: 8 points if $m < 300$, 0 points if $m > 600$
32 × 32: $m < 300$ 得 8 分，$m > 600$ 得 0 分

- 64 × 64: 8 points if $m < 1,300$, 0 points if $m > 2,000$
64 × 64: $m < 1,300$ 时 8 分，$m > 2,000$ 时 0 分

- 61 × 67: 10 points if $m < 2,000$, 0 points if $m > 3,000$
61 × 67: $m < 2,000$ 得 10 分，$m > 3,000$ 得 0 分

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.
您的代码必须是正确的，以获得任何特定大小的性能点。你的代码只需要在这三种情况下是正确的，你可以针对这三种情况进行优化。特别是，对于你的函数来说，显式地检查输入大小和实现针对每种情况优化的独立代码是完全可以的。

5.3 Evaluation for Style
5.3 风格评估

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.
代码风格有 7 点。这些都将由课程人员手动指定。样式指南可以在课程网站上找到。

The course staff will inspect your code in Part B for illegal arrays and excessive local variables.
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
在实验室工作

6.1 Working on Part A
6.1 在 a 部工作

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:
我们已经为您提供了一个自动评分程序，叫做 test-csim，它可以测试您的缓存模拟器在参考跟踪上的正确性。在运行测试之前一定要编译你的模拟器:

```
linux>      make
Linux >  ：制造
linux> ./test-csim
Linux > ./test-csim
```

		Your simulator 你的模拟器			Reference simulator 参考模拟器			
Points 积分	(s,E,b) (s，e，b)	Hits 点击率	Misses 没打中	Evicts 驱逐	Hits 点击率	Misses 没打中	Evicts 驱逐	
	(1,1,1)							traces/yi2.trace
3	(1,1,1)	9	8	6	9	8	6	痕迹/yi2.trace
	(4,2,4)							traces/yi.trace
3	(4,2,4)	4	5	2	4	5	2	痕迹/yi.trace
	(2,1,4)							traces/dave.trace
3	(2,1,4)	2	3	1	2	3	1	Trace/dave.trace
	(2,1,3)							traces/trans.trace
3	(2,1,3)	167	71	67	167	71	67	Trace/trans.trace
	(2,2,3)							traces/trans.trace
3	(2,2,3)	201	37	29	201	37	29	Trace/trans.trace
	(2,4,3)							traces/trans.trace
3	(2,4,3)	212	26	10	212	26	10	Trace/trans.trace
	(5,1,5)							traces/trans.trace
3	(5,1,5)	231	7	0	231	7	0	Trace/trans.trace
	(5,1,5)							traces/long.trace
6	(5,1,5)	265189	21775	21743	265189	21775	21743	痕迹/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，它显示您获得的分数、缓存参数、输入跟踪文件以及模拟器和参考模拟器结果的比较。

Here are some hints and suggestions for working on Part A:
下面是一些关于 a 部分的提示和建议:

- Do your initial debugging on the small traces, such as `traces/dave.trace`.
  对小跟踪(如 trace/dave.trace)进行初始调试。

- 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.
  引用模拟器接受一个可选的 -v 参数，该参数启用详细输出，显示每次访问内存时发生的命中、未命中和驱逐。你不需要在你的 csim.c 代码中实现这个特性，但是我们强烈建议你这样做。它允许你直接比较模拟器和参考跟踪文件中的参考模拟器的行为，从而帮助你进行调试。

- 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:

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

See “ man 3 getopt” for details.
详情请参阅“ man3getopt”。

- Each data load (L) or store (S) operation can cause at most one cache miss. The data modify operation
每个数据加载(l)或存储(s)操作最多会导致一个缓存丢失。数据修改操作
(M) 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.
将被视为加载，然后存储到相同的地址。因此，一个 m 操作可以导致两个缓存命中，或者一个未命中，一个命中加上一个可能的驱逐。

- 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.
如果您想使用 15-122 中的 c0 风格的契约，可以包括 contracts.h，为了方便起见，我们在讲义目录中提供了它。

6.2 Working on Part B
6.2 在 b 部工作

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

You can register up to 100 versions of the transpose function in your `trans.c` file. Each transpose version has the following form:
您可以在 trans.c 文件中注册最多 100 个版本的转置函数。每个转座版本都有以下格式:

```
/* Header comment */
标题注释
char trans_simple_desc[] = "A simple transpose";
Char trans _ simple _ desc [] = ‟ a simple transpose”;
void trans_simple(int M, int N, int A[N][M], int B[M][N])
Void trans _ simple (int m, int n, int a [ n ][ m ] , int b
[ m ][ n ])
{
    /* your transpose code here */
    /* 你的转座码在这里 */
}
```

Register a particular transpose function with the autograder by making a call of the form:
通过调用以下表单，在自动校正器中注册一个特定的转位函数:

```
registerTransFunction(trans_simple, trans_simple_desc);
寄存器函数(trans _ simple, trans _ simple _ desc) ;
```

in the `registerFunctions` routine in `trans.c`. At runtime, the autograder will evaluate each registered 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:

在 trans.c 中的 register 函数例程中。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，其中一个注册函数必须是你提交的转位 _ 提交函数:

```
registerTransFunction(transpose_submit, transpose_submit_desc);
寄存器转换函数(转位 _ 提交，转位 _ 提交 _ desc) ;
```

See the default `trans.c` function for an example of how this works.
查看默认的 trans.c 函数，看看它是如何工作的。

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).
自动平地机采用矩阵大小作为输入。它使用 valgrind 生成每个已注册的反姿势函数的轨迹。然后通过在带有参数(s = 5，e = 1，b = 5)的缓存上运行参考模拟器来评估每个跟踪。

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 :
例如，要在 32 × 32 矩阵上测试已注册的转置函数，请重新构建 test-trans，然后使用 m 和 n 的适当值运行它:

```
linux> make
Linux > make
linux> ./test-trans -M 32 -N 32
Linux >/test-trans-m32-n32
Step 1: Evaluating registered transpose funcs for correctness:
第一步：评估已注册转位功能的正确性：
func 0 (Transpose submission): correctness: 1
Func 0(Transpose submission) : 正确性：1
```

```
func 1 (Simple row-wise scan transpose): correctness: 1
Func1(简单的逐行扫描转位） : 正确性: 1
func 2 (column-wise scan transpose): correctness: 1
Func 2(列式扫描转位） : 正确性: 1
func 3 (using a zig-zag access pattern): correctness: 1
Func 3(使用之字形访问模式） : 正确性: 1

Step 2: Generating memory traces for registered transpose funcs.
第二步: 为已注册的转座函数生成内存跟踪。

Step 3: Evaluating performance of registered transpose funcs (s=5, E=1, b=5)
步骤 3: 评价注册转座函数(s = 5, e = 1, b = 5)的性能
func 0 (Transpose submission): hits:1766, misses:287, evictions:255
Func 0(Transpose 提交） : 命中: 1766, 未命中: 287, 驱逐: 255
func 1 (Simple row-wise scan transpose): hits:870, misses:1183, evictions:1151
Func 1(Simple row-wise scan transpose) : 命中: 870, 未命中: 1183, 驱逐: 1151
func 2 (column-wise scan transpose): hits:870, misses:1183, evictions:1151
Func 2(column-wise scan transpose) : 命中: 870, 未命中: 1183, 驱逐: 1151
func 3 (using a zig-zag access pattern): hits:1076, misses:977, evictions:945
Func 3(使用之字形访问模式） : 命中: 1076, 未命中: 977, 驱逐: 945

Summary for official submission (func 0): correctness=1 misses=287
Summary for official submission (func 0) : 正确性 = 1 miss = 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.
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 程序测试每个已注册的函数，显示每个函数的结果，并提取结果以供正式提交。

Here are some hints and suggestions for working on Part B.
下面是一些关于第二部分的提示和建议。

- 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:

  Test-trans 程序将函数 i 的跟踪保存在 trace.fI 文件中。这些跟踪文件是非常有价值的调试工具，可以帮助您准确理解每个转位函数的命中和错过来自哪里。要调试一个特定的函数，只需使用详细选项在参考模拟器中运行它的跟踪即可:

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

```
1683140,8 击不中
1683124,4 击中
1683120,4 击中
L 603124,4 miss eviction
S 6431a0,4 miss
L 603124，驱逐未遂 s
6431a0,4 未遂
...
...
```

- 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.

由于您的转置函数是在直接映射的缓存上计算的，因此冲突丢失是一个潜在的问题。考虑一下代码中冲突遗漏的可能性，特别是沿着对角线。试着想想访问模式，这将减少这些冲突遗漏的数量。

- Blocking is a useful technique for reducing cache misses. See

阻塞是一种减少缓存丢失的有用技术

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

for more information.
了解更多资料。

[2]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.

因为 valgrind 引入了许多与你的代码无关的堆栈访问，我们已经从跟踪中过滤掉了所有的堆栈访问。这就是为什么我们禁止了本地数组，并且限制了本地变量的数量。

6.3 Putting it all Together
6.3 把它们放在一起

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.
我们为您提供了一个驱动程序，名为。Py，它可以对你的模拟器进行完整的评估，并转换代码。这是你的教练用来评估你的双手的程序。驱动程序使用 test-csim 来评估你的模拟器，它使用 test-trans 来评估你提交的三个矩阵大小的转置函数。然后它会打印出你的结果和你所获得的分数的总结。

To run the driver, type:
要运行驱动程序，请键入:

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

7 Handing in Your Work
交出你的作品

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.
每次在 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。

> **SITE-SPECIFIC: Insert text here that tells each student how to hand in their `userid-handin.tar` file at your school.**
> **SITE-SPECIFIC: 在这里插入文本，告诉每个学生如何在学校提交他们的 userid-handin.tar 文件。**

IMPORTANT: Do not create the handin tarball on a Windows or Mac machine, and do not handin files in any other archive format, such as `.zip`, `.gzip`, or `.tgz` files.
重要提示: 不要在 Windows 或 Mac 机器上创建 handin tarball，也不要以任何其他存档格式提交文件，例如。压缩。Gzip，或。Tgz 文件。

10