Cache Lab

Instructors:

Sungyong Ahn

What is Cache LAB

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

- Part A: Writing a Cache Simulator
  - Write a small C program (about 200-300 lines) that simulates the behavior of a cache memory

- Part B: Optimizing Matrix Transpose
  - Optimize a small matrix transpose function, with the goal of minimizing the number of cache misses

Handout Instructions

- Lab materials are contained in a Unix tar file called *`cachelab-handout.tar`*
  - You can download from **PLATO**

- You have to extract the tar file in Linux
  - `$> tar xvf cachelab-handout.tar`

- You will be modifying two files: `csim.c` and `trans.c`.
- WARNING: Do not let the Windows WinZip program open up your .tar file

Reference Trace Files

- The `traces` subdirectory of the handout directory contains a collection of reference trace files generated by a Linux program called `valgrind`
- `[space]operation address,size`
- `64-bit hexadecimal memory address`

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

`I`: instruction load

`L`: data load

`S`: data store

`M`: data modify

NOTE: There is never a space before each “`I`”

Part A:Writing a Cache Simulator

- Write a cache simulator in `csim.c`
  - Input: a `valgrind` memory trace
  - Simulates the hit/miss behavior of a cache memory
  - Outputs: the total number of hits, misses, and evictions.

Part A:Writing a Cache Simulator

- Reference cache simulator (`csim-ref`)
    - Usage: `./csim-ref [-hv] -s <s> -E <E> -b <b> -t <tracefile>`
    - `-h`: Optional help flag that prints usage info
    - `-v`: Optional verbose flag that displays trace info
    - `-s <s>`: Number of set index bits (S = 2^s is the number of sets)
    - `-E <E>`: Associativity (number of lines per set)
    - `-b <b>`: Number of block bits (B = 2^b is the block size)
    - `-t <tracefile>`: Name of the `valgrind` trace to replay

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

Part A:Writing a Cache Simulator

```
/* Kildong Hong, 201912345 */
#include <getopt.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
```

- Programming Rules for Part A
  - Include your name and loginID in the header comment for `csim.c`.
  - Your `csim.c` file must compile without warnings in order to receive credit
  - Ignore all instruction cache accesses (lines starting with “`I`”)
  - The data modify operation (`M`) is treated as a load followed by a store to the same address.
  - Assume that memory accesses are aligned properly
    - you can ignore the request sizes in the `valgrind` traces

Part B: Optimizing Matrix Transpose

- Write a transpose function, called transpose_submit in trans.c that causes as few cache misses as possible
- The transpose of A, denoted A^T , is a matrix such that $A_{ij} = A^T_{ji}$
- Example transpose function

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

- Your job to write a similar function, called transpose_submit

```
char transpose_submit_desc[] = "Transpose submission";
void transpose_submit(int M, int N, int A[N][M], int B[M][N]);
```

Warning: Do not change the description string (“Transpose submission”)

Part B: Optimizing Matrix Transpose

- 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
  - Your transpose function may not use recursion
  - 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 any variant of malloc.

Evaluation (Part A: 27)

- Autograding program, called `test-csim`

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

TEST_CSIM_RESULTS=1
```

Evaluation (Part B: 26)

- Aautograding program, called `test-trans.c`
- You can register up to 100 versions of the transpose function in your `trans.c` file.

```
/*
 * You can define additional transpose functions below. We've defined
 * a simple one below to help you get started.
 */
char transpose_test_desc[] = "Transpose test";
void transpose_test(int M, int N, int A[N][M], int B[M][N])
{
    /* Insert matrix transpose code you want to test */
}

void registerFunctions()
{
    ...
    /* Register any additional transpose functions */
    registerTransFunction(transpose_test, transpose_test_desc);
    ...
}
```

Evaluation (Part B: 26)

- Aautograding program, called `test-trans.c`
- You can register up to 100 versions of the transpose function in your `trans.c` file.
- You must implement two versions of the transpose function.

```
linux> make
linux> ./test-trans -M 32 -N 32
Function 0 (4 total)
Step 1: Validating and generating memory traces
Step 2: Evaluating performance (s=5, E=1, b=5)
func 0 (Transpose submission): hits:1766, misses:287, evictions:255
Function 1 (4 total)
Step 1: Validating and generating memory traces
Step 2: Evaluating performance (s=5, E=1, b=5)
func 1 (Simple row-wise scan transpose): hits:870, misses:1183, evictions:1151

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

Evaluation (Putting it all Together)

- *driver program*, called `./driver.py`, that performs a complete evaluation of your simulator and transpose code.

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.
- You need to upload your `userid-handin.tar` file, to **PLATO**.

- Note that you must implement two transpose function including `transpose_submit` function

Submission

- Due to **6/14 (Mon.)** 23:59
  - 하루 딜레이 시 만점기준 25% 감점

- `userid-handin.tar` 파일 PLATO에 제출

- Please Read the Writup.