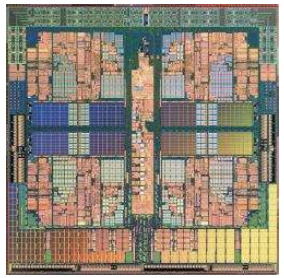
KAIST



Computer Architecture Lab.

Lab 4
Cachelab

Contents

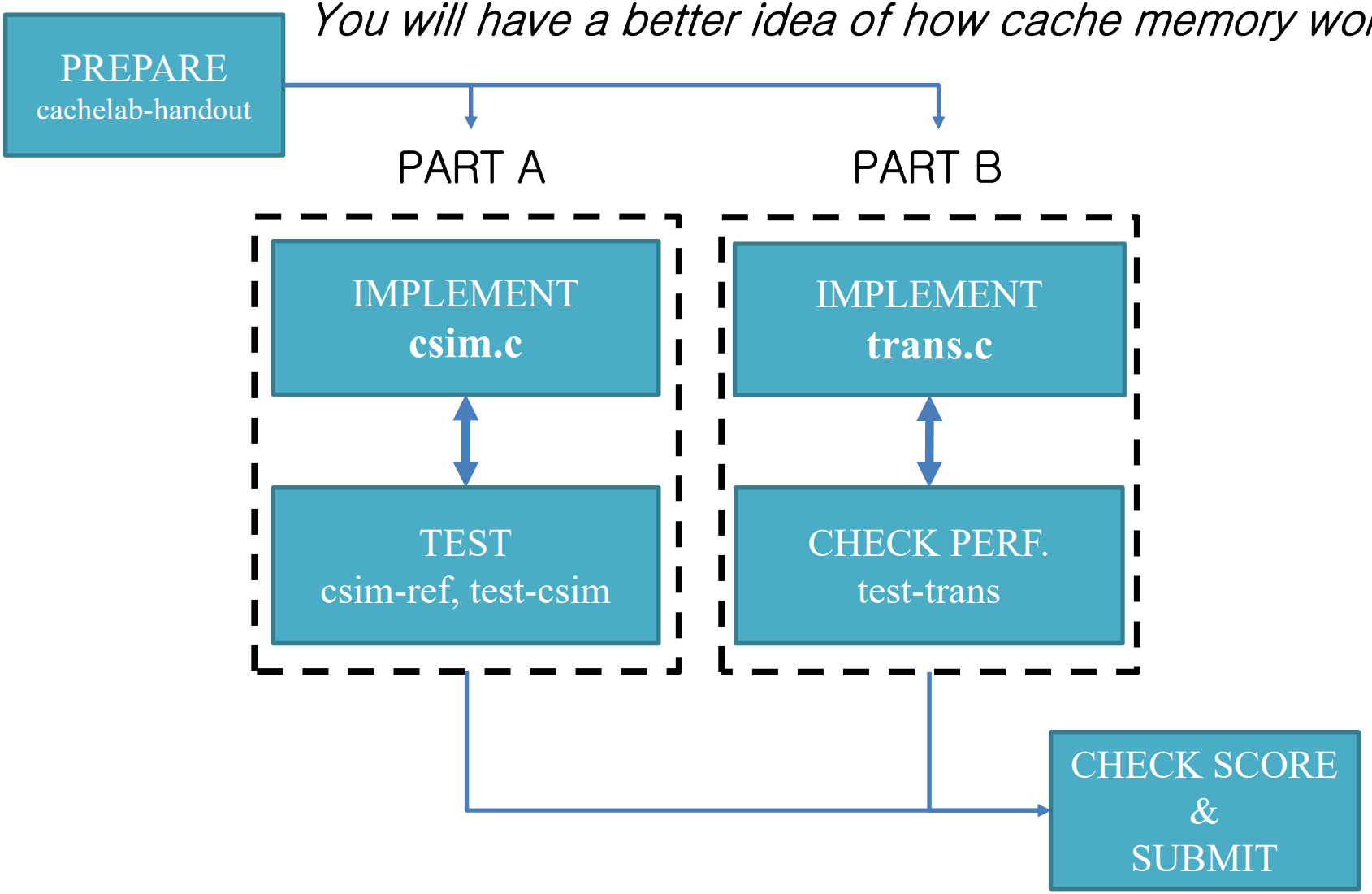
- Introduction
- Instructions
- Grading Policy
- Administration

Introduction

- Main purpose of this lab : Understand the impact that cache memories have on the performance of your program.
- You will be implementing two features for this lab in C programming language.
- Part A : Cache simulator(maybe 200 ~ 300 lines)
  - Goal : Properly simulate the behavior of caches
- Part B : Matrix transpose function
  - Goal : minimize number of cache miss

Introduction - Workflow

We recommend that you solve 'A 'first.
You will have a better idea of how cache memory works



Instructions – Preparing

- **Login to your assigned canis server**
  - **Use the same server for datalab**
  - **Yon can check which server you have been assigned on KLMS**
- **Copy and decompress**
  - **Linux> cp /home/lab/cachelab-handout.tar ~**
  - **Linux> tar xvf cachelab-handout.tar**
  - **You will see the following directories and files**

```
cs230_ta@canis01:~$ ls
cachelab-handout  cachelab-handout.tar
cs230_ta@canis01:~$ cd cachelab-handout/
cs230_ta@canis01:~/cachelab-handout$ ls
Makefile  README  cachelab.c  cachelab.h  csim-ref  csim.c  driver.py  test-csim  test-trans.c  trace.tmp  tracegen.c  traces  trans.c
cs230_ta@canis01:~/cachelab-handout$
```

Instructions – Cache Sim(Part A)

- **Overview of Cache simulator**
  - **Input : Trace files (described in the next slide)**
  - **Output : the number of cache hits, misses, evictions**
- **Goal of Cache simulator**
  - **Produce the same output of csim-ref**

```
sjna@cluster:~/cachelab/src$ ./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
```

What you should do on Part A

- You have to implement csim.c

```
sjna@cluster:~/cachelab-handout$ ls
Makefile  cachelab.c  csim-ref  driver.py  test-trans.c  traces
README    cachelab.h  csim.c    test-csim  tracegen.c    trans.c
```

- csim.c - Implement from scratch

```c
#include "cachelab.h"

int main()
{
    printSummary(0, 0, 0);
    return 0;
}
```

printSummary function is defined in “cachelab.c”

```c
void printSummary(int hits, int misses, int evictions)
{
    printf("hits:%d misses:%d evictions:%d\n", hits, misses, evictions);
    FILE* output_fp = fopen(".csim_results", "w");
    assert(output_fp);
    fprintf(output_fp, "%d %d %d\n", hits, misses, evictions);
    fclose(output_fp);
}
```

Instructions – Trace files

- **Traces**
  - **Input of cache simulator (located in subdirectory "traces")**

```
sjna@cluster:~/cachelab-handout$ ls
Makefile  cachelab.c  csim-ref  driver.py  test-trans.c  traces
README    cachelab.h  csim.c    test-csim  tracegen.c    trans.c
```

  - **Trace file format : [space] operation address, size**
    - **Operation : "I" instruction load , "S" : data store**
      "M" : data modify , "L" : data load
    - **Space : There is never a space before each "I" operation**
    - **Address : 64-bit hexadecimal memory address**
    - **Size : the number of bytes accessed by the operation**

Example of trace file

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

Tools for you on Part A

- **csim-ref**
  - baseline program of cache simulator
  - You can check details using verbose option

- **Autograding program test-csim**
  - Check your results for each testing case (total 8 cases)
  - 7 cases give 3 points, last case gives 6 points
  - Total point is 27

Usage csim-ref

- **csim-ref is baseline program**
  - **Your program should produce the same result of this program**
- **In the manual, usage is as following:**

```
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

- **Output**

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

*** The output of your program should be same ! ***

csim-ref – debug option

- You can apply verbose option to csim-ref
  - The same example in verbose mode

```
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
```

- You can check details for each line
  - If you implement –v option (it is optional) , you can compare the behavior of your program with that of csim-ref
  - It will be helpful for debugging

Autograding program test-csim

- **After writing code, you can check your score**
  - **Before executing test-csim, you have to do 'make'**

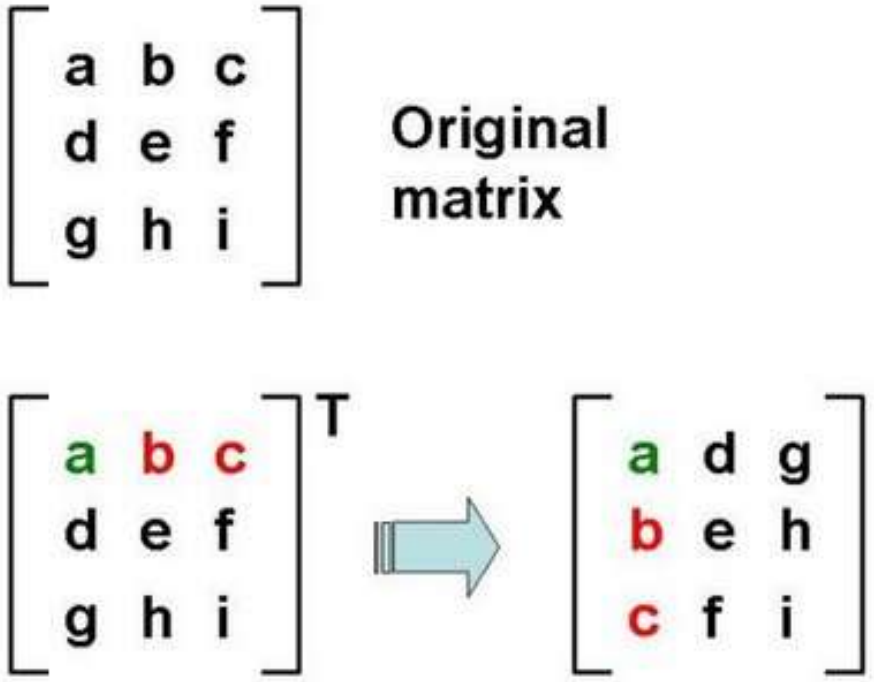
```
sjna@cluster:~/cachelab-handout$ make
gcc -g -Wall -Werror -std=c99 -m64 -o csim csim.c cachelab.c -lm
gcc -g -Wall -Werror -std=c99 -m64 -O0 -c trans.c
gcc -g -Wall -Werror -std=c99 -m64 -o test-trans test-trans.c cachelab.c trans.o
gcc -g -Wall -Werror -std=c99 -m64 -O0 -o tracegen tracegen.c trans.o cachelab.c
# Generate a handin tar file each time you compile
tar -cvf sjna-handin.tar  csim.c trans.c
csim.c
```

```
sjna@cluster:~/cachelab/src$ ./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
```

Instructions – Transpose(Part B)

- **Part B : Optimizing Matrix Transpose**
  - **Goal: Minimize the number of cache misses**

- **Matrix Transpose**



Instructions – Write Function

- **Write your functions in trans.c**
- **Each function must follow the format below**

```
/* 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 */
}
```

Store the result
No return value!

- **Example given in trans.c**

```
/*
 * trans - A simple baseline transpose function, not optimized for the cache.
 */
char trans_desc[] = "Simple row-wise scan transpose";
void trans(int M, int N, int A[N][M], int B[M][N])
{
    int i, j, tmp;

    for (i = 0; i < N; i++) {
        for (j = 0; j < M; j++) {
            tmp = A[i][j];
            B[j][i] = tmp;
        }
    }

}
```

Instructions - Register

- Add your transpose function in trans.c
- Use the function below in 'registerFunctions()'
  - registerTransFunction(trans_simple, trans_simple_desc);

```
void registerFunctions()
{
    /* Register your solution function */
    registerTransFunction(transpose_submit, transpose_submit_desc);

    /* Register any additional transpose functions */
    registerTransFunction(trans, trans_desc);

}
```

Your transpose function for submission and grades

Additional transpose function for testing

You may add more functions by using 'registerTransFunction'

You can register up to 100 versions

Instructions – Test Tool

- **test-trans : auto testing program for your function**

- **Example**

```
linux> make
linux> ./test-trans -M 32 -N 32
Step 1: Evaluating registered transpose funcs for correctness:
func 0 (Transpose submission): correctness: 1
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
```

Tip for Part B

- **Note that the test cases are fixed!**
  - **You may write optimized code for certain cases**
- **Define transpose submit as below**

```
void transpose_32x32(int M, int N, int A[N][M], int B[M][N]);
void transpose_64x64(int M, int N, int A[N][M], int B[M][N]);
void transpose_61x67(int M, int N, int A[N][M], int B[M][N]);

char transpose_submit_desc[] = "Transpose submission";
void transpose_submit(int M, int N, int A[N][M], int B[M][N])
{
  if(N == 32)
    transpose_32x32(M, N, A, B);

  else if(N == 64)
    transpose_64x64(M, N, A, B);

  else
    transpose_61x67(M, N, A, B);

}
```

Do NOT edit this string!

Autograder searches this string for your grades

- **Define each function separately**
  - **transpose_32x32, transpose_64x64, transpose_61x67**

Grading Policy – Part A

- **Part A : total 27 points**
- **Compare the following 8 cases with csim-ref**
  - ./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
- **Give 3 points for each match, 6 points for last case**

Grading Policy – Part B

- **Part B : total 26 points**

- **We will be grading the following cases**
  - 32 × 32 (M = 32, N = 32)
  - 64 × 64 (M = 64, N = 64)
  - 61 × 67 (M = 61, N = 67)

- **Apply the following conditions for grading**
  - 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

Grading Tool

- **driver.py : Tool for grading both part A and part B**
  - **After compiling your code(by 'make'), type './driver.py'**

```
Part A: Testing cache simulator
Running ./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


Part B: Testing transpose function
Running ./test-trans -M 32 -N 32
Running ./test-trans -M 64 -N 64
Running ./test-trans -M 61 -N 67

Cache Lab summary:
                        Points   Max pts      Misses
Csim correctness          27.0        27
Trans perf 32x32           6.9         8         343
Trans perf 64x64           1.8         8        1843
Trans perf 61x67           8.8        10        2118
      Total points        44.5        53
```

Submit

- After finishing the assignment you must submit your files to the server

1) Generate handin tar file by typing ‘make’

```
cs230_ta@canis01:~/cachelab-handout$ make
gcc -g -Wall -Werror -std=c99 -m64 -o csim csim.c cachelab.c -lm
gcc -g -Wall -Werror -std=c99 -m64 -O0 -c trans.c
gcc -g -Wall -Werror -std=c99 -m64 -o test-trans test-trans.c cachelab.c trans.o
gcc -g -Wall -Werror -std=c99 -m64 -O0 -o tracegen tracegen.c trans.o cachelab.c
# Generate a handin tar file each time you compile
tar -cvf cs230_ta-handin.tar  csim.c trans.c
csim.c
trans.c
cs230_ta@canis01:~/cachelab-handout$ ls
Makefile  cachelab.c  cs230_ta-handin.tar  csim-ref  driver.py  test-trans    trace.tmp  tracegen.c  trans.c
README    cachelab.h  csim                 csim.c    test-csim  test-trans.c  tracegen   traces      trans.o
```

2) Type ‘submit Lab4 userid-handin.tar’

```
cs230_ta@canis01:~/cachelab-handout$ submit Lab4 cs230_ta-handin.tar

[CS230] System Programming Submit tool
================================================
 * Your ID :  cs230_ta
 * Lab Number :  Lab4
 * Making first submission for Lab4
 * Upload Status : Success
 * Upload Time :  2016-10-26 19:42:24
 * Upload Count (submission version) : 1

If there seems to be a problem with your submission
please email cs230_ta@calab.kaist.ac.kr.
================================================
```

Check Submission

- **You can check your submission status**

1) Type ‘submit check Lab4’

2) You should see something similar to the following messages

```
[CS230] System Programming Submit tool
======================================================
 * Your ID : cs230_ta
 * Lab Number : Lab4
 * Upload Time(Latest) : 10-26_19:42:24

If there seems to be a problem with your submission
please email cs230_ta@calab.kaist.ac.kr.
======================================================

cs230_ta@canis01:~/cachelab-handout$
```

Important Notes

- Read the documentation cachelab.pdf carefully
  - There are important tips and warnings
- Please do your work on a 64-bit Linux machine
  - Avoid unwanted surprises when decompressing on Windows or MAC
- Do not forget to submit
- Copying is strictly forbidden

Administration(1)

- **Due Date : ~ 2016/11/8, 23:59**
- **Only Electronic handins (no handwritten reports)**
- **Results are graded on submitted code**
  - **If you do not submit, no grade!**
  - **Will be graded on the most recently submitted code**

Administration(2)

- **Late Policy:**
  - Accept submits until 3 days over due
  - Receive 15% penalty per day over due work
- **Definition of 'late':**
  - The most recent submission is over due
  - Be cautious when submitting! TAs will not accept complains such as "I accidently submitted", "I forgot to submit"
- **TA's E-mail Address : cs230_ta@calab.kaist.ac.kr**

Thank You!!