

OS HW2

Multi-Threading Programming

Operating System 111 Fall

Professor: W.J. TSAI

TA. 王菱君 王麗婷 黃逸弘 余孟倫

APIs

- Thread management: `<pthread.h>`
 - `pthread_create`
 - `pthread_join`
 - `pthread_exit`
- Reference:

[POSIX Threads Programming | LLNL HPC Tutorials](#)

Exercise - Hello Thread

```
1  #include <iostream>
2  #include <thread>
3  #include <unistd.h>
4  using namespace std;
5
6  // child threading function
7  void* child_thread(void* data){
8      sleep(1);
9      cout << "Child Pthread ID - " << pthread_self() << endl;
10     char *str = (char*) data; // get data "Child"
11     for(int i = 0 ; i < 3 ; i++){
12         sleep(1);
13         cout << str << endl; // output every second
14     }
15     pthread_exit(NULL); // exit child thread
16 }
17
18 // main function
19 int main(void){
20     pthread_t t; // define thread
21     pthread_create(&t,NULL,child_thread,(void *)"Child"); // create a child thread
22     sleep(1);
23     cout << "Master Pthread ID - " << pthread_self() << endl; // output master thread's ID
24     // pthread_join(t,NULL); // wait for child threading finished, then output "Master"
25     for(int i = 0 ; i < 3 ; i++){
26         sleep(1);
27         cout << "Master" << endl;
28     }
29 }
30 // pthread_join(t,NULL); // output "Master" during child thread outputting "Child"
31 return 0;
32 }
```

<- hello_thread.cpp

```
$ g++ hello_thread.cpp -lpthread -o hello_thread
$ ./hello_thread
```

output:

```
Child Pthread ID - 0x16b93f000
Master Pthread ID - 0x104814580
Master
Child
Master
Child
Master
Child
```

Exercise - Single-threading

```
1  # include <iostream>
2  # include <thread>
3  # include <unistd.h>
4  using namespace std;
5
6  # define MAX 500
7
8  int matA[MAX][MAX];
9  int matB[MAX][MAX];
10 int matC[MAX][MAX]; // Result of Addition
11 int matD[MAX][MAX]; // Result of Multiplication
12
13 void Addition(){
14     // Addition -> matC
15     for (int i = 0; i < MAX; i++) {
16         for (int j = 0; j < MAX; j++) {
17             matC[i][j] = matA[i][j] + matB[i][j];
18         }
19     }
20 }
21 void Multiplication(){
22     // Multiplication -> matD
23     for (int i = 0; i < MAX; i++) {
24         for (int j = 0; j < MAX; j++) {
25             matD[i][j] = 0;
26             for (int k = 0; k < MAX; k++) {
27                 matD[i][j] += matA[i][k] * matB[k][j];
28             }
29         }
30     }
31 }
32
33 int main()
```

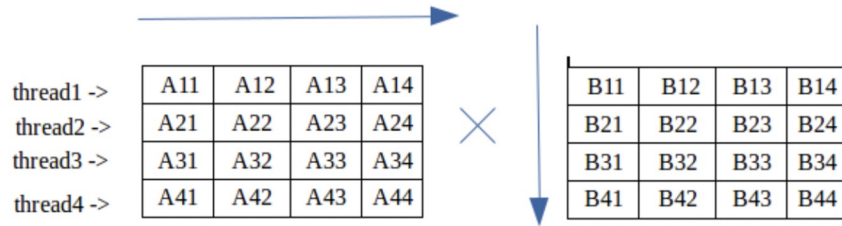
```
$ g++ single_thread.cpp -lpthread -o
single_thread
$ ./hello_thread < input.txt
```

output:

```
2248968
2528950360
```

Homework - Multithreading

- Use multithreading to do matrix calculation problems
- Find the best thread quantity between 2 ~ 20
- Output the **sum of every element** of your matrices



You should implement:

1. STDIN(e.g. scanf, cin)
2. Multiplication/addition function
3. Thread management
4. STDOUT(e.g. printf, cout)

DO NOT USE FILE I/O !

Compile & Run Commands

- **Compile:**

(single-thread) `$ g++ -o filename filename.cpp`

(multi-thread) `$ g++ -o filename filename.cpp -lpthread`

- **Run:**

`$./filename < input.txt`

- **Environment:**

You should run your code on the multiple CPU

Compile & Run Commands

- Performance between single-thread and multi-thread:

use “\$ time ./filename” to check the execution time

```
• (base) jamie@jamie-System-Product-Name:~/Documents$ time ./ST >STout.txt
```

real	0m0.721s
user	0m0.709s
sys	0m0.012s

```
• (base) jamie@jamie-System-Product-Name:~/Documents$ time ./MT1 >MT1out.txt
```

real	0m0.335s
user	0m1.074s
sys	0m0.047s

Speed up

- Compare the **real** time between single thread and the multi thread

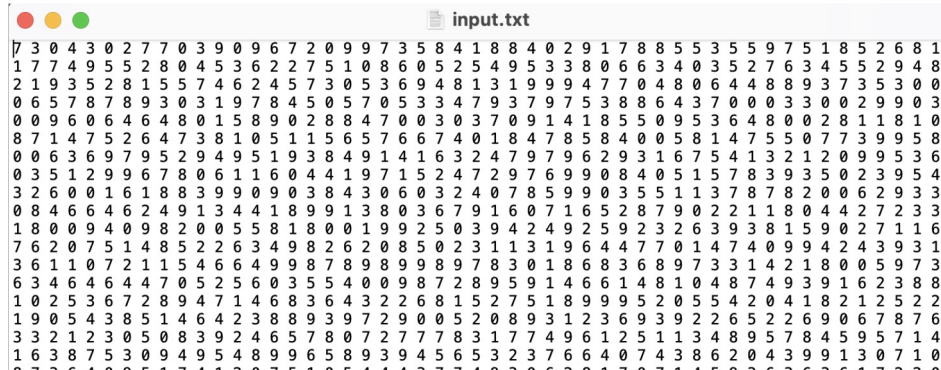
Speedup =

$$\text{real time of single thread} / \text{real time of multiple threads}$$

- Compare with your classmates (bigger speed-up gets higher score)
 - Top 1/5 get 20 points, second 1/5 get 16 points, etc.

Input format

- Input size : 500 * 500 matrix
- All elements are seperated by space (without newline)
- Matrix A start from line1, matrix B start from line 2
- Integer value random from 0~9



```
7 3 0 4 3 0 2 7 7 0 3 9 0 9 6 7 2 0 9 9 7 3 5 8 4 1 8 8 4 0 2 9 1 7 8 8 5 5 3 5 5 9 7 5 1 8 5 2 6 8 1
1 7 7 4 9 5 5 2 8 0 4 5 3 6 2 2 7 5 1 0 8 6 0 5 2 5 4 9 5 3 3 8 0 6 6 3 4 0 3 5 2 7 6 3 4 5 5 2 9 4 8
2 1 9 3 5 2 8 1 5 5 7 4 6 2 4 5 7 3 0 5 3 6 9 4 8 1 3 1 9 9 9 4 7 7 0 4 8 0 6 4 4 8 8 9 3 7 3 5 3 0 0
0 6 5 7 8 7 8 9 3 0 3 1 9 7 8 4 5 0 5 7 0 5 3 3 4 7 9 3 7 9 7 5 3 8 8 6 4 3 7 0 0 0 3 3 0 0 2 9 9 0 3
0 0 9 6 0 6 4 6 4 8 0 1 5 8 9 0 2 8 8 4 7 0 0 3 0 3 7 0 9 1 4 1 8 5 5 0 9 5 3 6 4 8 0 0 2 8 1 1 8 1 0
8 7 1 4 7 5 2 6 4 7 3 8 1 0 5 1 1 5 6 5 7 6 6 7 4 0 1 8 4 7 8 5 8 4 0 0 5 8 1 4 7 5 5 0 7 7 3 9 9 5 8
0 0 6 3 6 9 7 9 5 2 9 4 9 5 1 9 3 8 4 9 1 4 1 6 3 2 4 7 9 7 9 6 2 9 3 1 6 7 5 4 1 3 2 1 2 0 9 9 5 3 6
0 3 5 1 2 9 9 6 7 8 0 6 1 1 6 0 4 4 1 9 7 1 5 2 4 7 2 9 7 6 9 9 0 8 4 0 5 1 5 7 8 3 9 3 5 0 2 3 9 5 4
3 2 6 0 0 1 6 1 8 8 3 9 9 0 9 0 3 8 4 3 0 6 0 3 2 4 0 7 8 5 9 9 0 3 5 5 1 1 3 7 8 7 8 2 0 0 6 2 9 3 3
0 8 4 6 6 4 6 2 4 9 1 3 4 4 1 8 9 9 1 3 8 0 3 6 7 9 1 6 0 7 1 6 5 2 8 7 9 0 2 2 1 1 8 0 4 4 2 7 2 3 3
1 8 0 0 9 4 0 9 8 2 0 0 5 5 8 1 8 0 0 1 9 9 2 5 0 3 9 4 2 4 9 2 5 9 2 3 2 6 3 9 3 8 1 5 9 0 2 7 1 1 6
7 6 2 0 7 5 1 4 8 5 2 2 6 3 4 9 8 2 6 2 0 8 5 0 2 3 1 1 3 1 9 6 4 4 7 7 0 1 4 7 4 0 9 9 4 2 4 3 9 3 1
3 6 1 1 0 7 2 1 1 5 4 6 6 4 9 9 8 7 8 9 8 9 9 8 9 7 8 3 0 1 8 6 8 3 6 8 9 7 3 3 1 4 2 1 8 0 0 5 9 7 3
6 3 4 6 4 6 4 4 7 0 5 2 5 6 0 3 5 5 4 0 0 9 8 7 2 8 9 5 9 1 4 6 6 1 4 8 1 0 4 8 7 4 9 3 9 1 6 2 3 8 8
1 0 2 5 3 6 7 2 8 9 4 7 1 4 6 8 3 6 4 3 2 2 6 8 1 5 2 7 5 1 8 9 9 9 5 2 0 5 5 4 2 0 4 1 8 2 1 2 5 2 2
1 9 0 5 4 3 8 5 1 4 6 4 2 3 8 8 9 3 9 7 2 9 0 0 5 2 0 8 9 3 1 2 3 6 9 3 9 2 2 6 5 2 2 6 9 0 6 7 8 7 6
3 3 2 1 2 3 0 5 0 8 3 9 2 4 6 5 7 8 0 7 2 7 7 7 8 3 1 7 7 4 9 6 1 2 5 1 1 3 4 8 9 5 7 8 4 5 9 5 7 1 4
1 6 3 8 7 5 3 0 9 4 9 5 4 8 9 9 6 5 8 9 3 9 4 5 6 5 3 2 3 7 6 6 4 0 7 4 3 8 6 2 0 4 3 9 9 1 3 0 7 1 0
```

Output format

- Output the sum of every element of the Addition matrix FIRST
- Then output the sum of the Multiplication matrix NEXT LINE

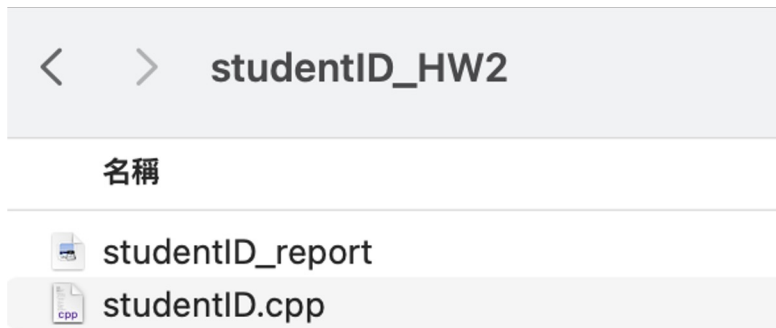
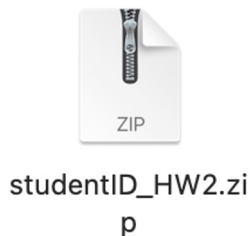
(Hint: If the sum is too big, try use long long int)

Requirements

- Multi-threading method should be much faster than Single-threading, and their results must be exactly the same.
- Write your codes in C/C++
- You need to hand in one multi_thread versions and a report. Put `studentID.cpp` and `studentID_report.pdf` into the same compressed file without input, output data or any folder
- The type of compressed file must be “`studentID_HW2.zip`”
- Use Ubuntu or NYCU work station as your environment.

Requirements

- The compressed file needs to be as follow:



Grading

- Total score : 100 pts. **COPY WILL GET 0 POINT!**
- Speedup : 20 pts
- Multi-thread programs : 50 pts (correctness)
- Report : 30 pts
- Incorrect file format : -10 pts
- Use FILE I/O : -5 pts
- Deadline : 2022/10/30 (SUN) 23:59
- **Late submission will get a -20% point per day.**