EEE5501-01 **Spring 2021**

Assignment 3: Multi-threading

Due: Sunday, May 16, 2021, 11:59PM

1 Introduction

- The objective of this assignment is to implement a multi-threaded program in C++.
- You have to write a parallel merge sort algorithm either using `pthread` or `std::thread` based on the lecture notes of `thread.pdf`. You are free to choose your preferred option of multi-threading implementation.
- In Ubuntu or Mac terminal, use `wget` to download a copy of skeleton code compressed into `thread.tar`.

```
$ wget https://icsl.yonsei.ac.kr/wp-content/uploads/thread.tar
```

- Decompress the `tar` file, and go to the `thread/` directory. Type `ls` to find the following list of files.

```
$ tar xf thread.tar
$ cd template/
$ ls
data    main.cc   sort.h       tar.sh
data.h  Makefile  stopwatch.h
```

- From the listed files, `sort.h` is the only file you will have to work on.
- You are allowed to change other files for your own testing and validation, but they will revert to the original state when your assignment is graded.
- To compile the code, you can simply type `make` in the terminal. `Makefile` that comes along with the skeleton code has automated all the compiling scripts for you.

```
$ make
g++ -Wall -Werror -g -std=c++11 -o main.o -c main.cc
g++ -o thread main.o -pthread
```

- A run command needs to specify the number of threads to engage in the program execution and optionally an input file containing data to sort. If the input file is not specified, it loads the `data` file in the directory by default. Four million random integer numbers in `data` are stored in binary format not in plain texts.

```
$ ./thread
Usage: ./thread <num_threads> <data_file>

$ ./thread 4
Elapsed time = 0 usec
Failed: array is not sorted
```

2 Implementation

- The following code box shows the contents of `main()`.
- `main()` restricts the number of threads to be less than 1024, and it has to be a power of two such as 1, 2, 4, 8, $\cdots$ to be compatible with bilateral divide-and-conquer strategy of merge sort algorithm.

- It loads data points through the `load()` function defined in `data.h`. When the function returns, `array` points to the data array, and `size` tells the total number of data in the array.
- The data array is supposed to be sorted by the `sort()` function. This function is surrounded by `stopwatch` to measure the runtime of it.
- Lastly, `fin()` checks if the array is sorted and then deallocates it. If the array is sorted, it displays a message of `Done: array is sorted`, or otherwise you will see `Failed: array is not sorted`.
- Four million data points are overwhelmingly large to use for validating an initial implementation of merge sort algorithm. You will have to modify `main()` to test your own implementation with a small, traceable array.

```
/* main.cc */

#include <cstdint>
#include <iostream>
#include "data.h"
#include "sort.h"
#include "stopwatch.h"

int main(int argc, char **argv) {
    // Run command message
    if((argc < 2) || (argc > 3)) {
        std::cerr << "Usage: " << argv[0]
                  << " <num_threads> <data_file>" << std::endl;
        exit(1);
    }

    // Get the number of threads and data file name
    unsigned num_threads  = (unsigned)std::stoi(argv[1]);
    const char *data_file = argc == 3 ? argv[2] : "data";

    // Data array and its size
    int *array = 0; uint64_t size = 0;

    // Max num_threads is 1024.
    unsigned n = num_threads;
    if(n > 1024) {
        std::cerr << "Error: too many threads" << std::endl;
        exit(1);
    }
    // num_threads must be power of two.
    while(!(n & (unsigned)1)) { n = n >> 1; }
    if(n != 1) {
        std::cerr << "Error: num_threads is not power of two" << std::endl;
        exit(1);
    }

    // Load data to the array.
    load(data_file, array, size);

    // Measure time spent on sorting.
    stopwatch_t stopwatch;
    stopwatch.start();
    // Sort the array.
    sort(array, size, num_threads);
    stopwatch.stop();
    stopwatch.display();
    // Finalize.
    fin(array, size);

    return 0;
}
```

- The next shows `sort.h` that you will have to implement a parallel merge sort algorithm in it. This file has nothing but an empty `sort()` function.
- You will probably have to define a few more functions in this file to implement the parallel merge sort and have them called from the `sort()` function.
- There are many different ways to implement the parallel merge sort. The goal of this assignment is to practice multi-threaded programming, not sorting algorithm itself. Thus, this assignment does not put any restrictions or constraints on the code except for the use of standard libraries such as `std::merge`, `std::inplace_merge`, `std::sort`, etc. You have to write your own implementation of the divide-and-merge algorithm.

```
/* sort.h */

#ifndef __SORT_H__
#define __SORT_H__

#include <algorithm>
#include <cstdlib>
#include <cstring>
#include <functional>
#include <new>
#include <pthread.h>
#include <thread>

template <typename T>
void sort(T *array, const size_t num_data, const unsigned num_threads) {
    /* Assignment */
}

#endif
```

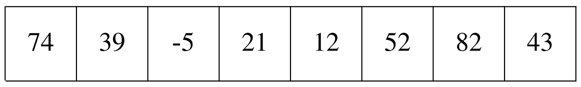
- When the parallel merge sort is implemented, executing the code will produce the following output, for example.

```
$ ./thread 1
Elapsed time = 537.591 msec
Done: array is sorted

$ ./thread 2
Elapsed time = 290.997 msec
Done: array is sorted

$ ./thread 4
Elapsed time = 171.846 msec
Done: array is sorted

$ ./thread 8
Elapsed time = 117.977 msec
Done: array is sorted

$ ./thread 16
Elapsed time = 103.167 msec
Done: array is sorted
```

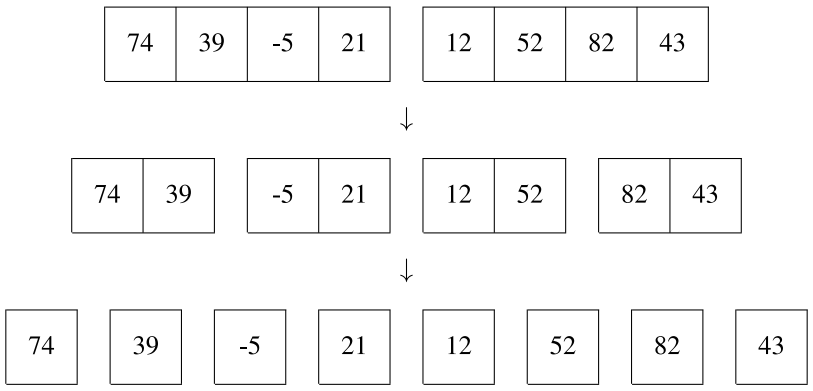
- Since the execution time of code strongly depends on the performance of executing hardware and number of cores, the absolute time in seconds is not a part of grading.
- However, your code must show improving performance for increasing the number of threads. The performance will most likely be saturated by the number of cores available in your CPU.

3 Merge Sort

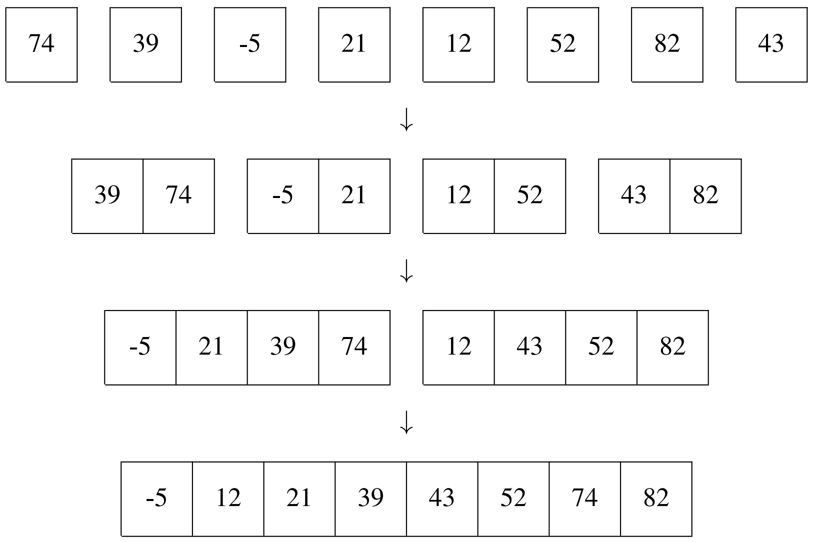
- This section briefly explains the baseline idea of merge sort algorithm.
- The merge sort achieves sorting by repeatedly dividing a data array into half and merging the divided sub-arrays. This approach is often called as a divide-and-conquer strategy.
- The following shows the concept of merge sort. Suppose the array has eight integer numbers randomly arranged.



- The merge sort begins with dividing the array into half. It keeps dividing the array until it is no more divisible.

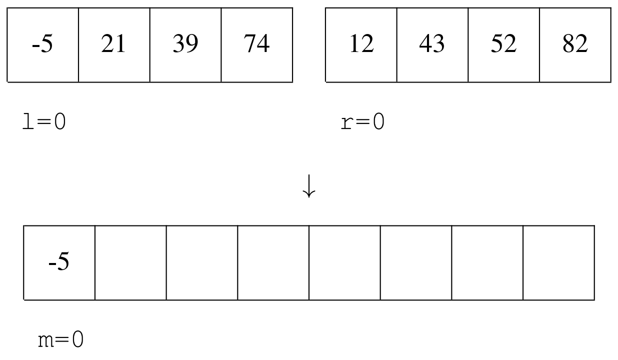


- Then, each of the next steps combines two sorted sub-arrays to form a larger sorted array.

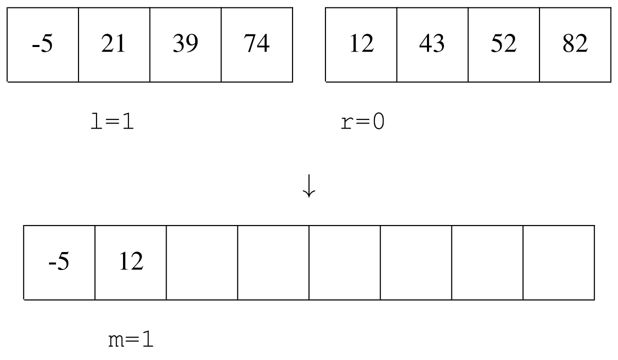


- The key of merge sort is that two sub-arrays to be merged are already sorted. Thus, simply scanning through the sub-arrays can form a merged, sorted array. The following draws a part of the last step in the previous example that combines two four-element sub-arrays.
- Three variables (or pointers) are introduced in the following illustration. `l` and `r` are the current positions of left and right sub-arrays, respectively. `m` is the current index of the merged array.

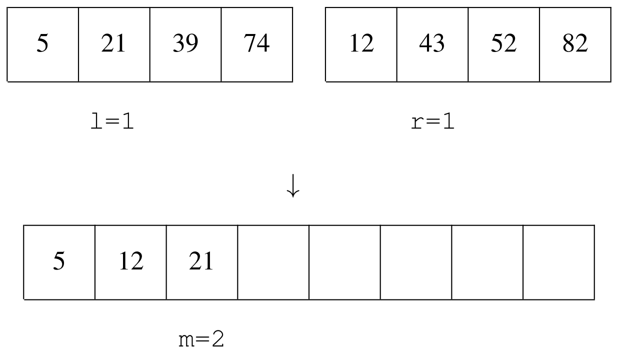
First merge:



Second merge:



Third merge:



4 Submission

- When the assignment is done, execute the `tar.sh` script in the `thread/` directory.
- It will compress the `thread/` directory but not including the 16MB `data` file into a `tar` file named after your student ID such as `2021310000.tar`.

```
$ ./tar.sh
rm -f main.o thread
thread/
thread/main.cc
thread/Makefile
thread/sort.h
thread/stopwatch.h
thread/data.h
thread/tar.sh

$ ls
2021310000.tar  data.h   Makefile  stopwatch.h
data            main.cc  sort.h    tar.sh
```

- Upload the tar file (e.g., `2021310000.tar`) on LearnUs. Do not rename the `tar` file or C++ files included in it.

5 Grading Rules

- The following is the general guideline for grading. 30-point scale will be used for this assignment. The minimum score is zero, and negative scores will not be given. Grading rules are subject to change, and the grader may add a few extra rules for fair evaluation of students' efforts.

  -3 points: A submitted `tar` file is renamed and includes redundant tags such as `project3`, student name, etc.

  -5 points: A program code does not have sufficient amount of comments. Comments in the skeleton code do not count. You must make an effort to clearly explain what each part of your code intends to do.

  -15 points: The code sorts the array, but increasing the number of threads does not decrease the execution time.

  -25 points: The code does not compile or fails to sort the array, but it shows substantial efforts to complete the assignment.

  -30 points: The code uses standard libraries such as `std::sort` and `std::merge` to implement the merge sort. Such an implementation will be regarded as lack of efforts.

  -30 points: No or late submission. The following cases will also be regarded as no submissions.

  ∗ Little to no efforts in the code (e.g., submitting nearly the same version of code as the skeleton code) will be regarded as no submission. Even if the code is incomplete, you must make substantial amount of efforts to earn partial credits.

  ∗ Fake codes will be regarded as cheating attempts and thus not graded. Examples of fake codes are i) hard-coding a program to print expected outputs to deceive the grader as if the program is correctly running, ii) copying and pasting random stuff found on the Internet to make the code look as if some efforts are made. More serious penalties may be considered if students abuse the grading rules.

  Final grade = F: The submitted code is copied from someone else. All students involved in the incident will be penalized and given F for the final grades irrespective of assignments, attendance, etc.

- Your teaching assistant (TA) will grade your assignments. If you think your assignment score is incorrect for any reasons, feel free to discuss your concerns with the TA. In case no agreement is made between you and the TA, elevate the case to the instructor to review your assignment. Refer to the course website for the contact information of TA and instructor: `https://icsl.yonsei.ac.kr/eee5501`
- Begging partial credits for no valid reasons will be treated as a cheating attempt, and such a student will lose all scores of the assignment.