Assignment - A3

Problem Statement: Parallel Sorting Algorithms: For bubble sort and merge sort based on existing sequential algorithms.

Objectives:

- To understand parallel execution & decomposition of tasks.
- To understand basics of parallel execution using OpenMP.

Outcomes:

- Understood parallel execution of decomposed tasks for given sorting algorithm
- Understood and implemented concepts of OpenMP.

Software and Hardware Requirements:

Google ~~cla~~ colab, compatible web browser, computer peripherals.

Date of Completion: 21/08/20

Theory:

- OpenMP: Refers to Open Multiprocessing. It is a shared memory programming model that is supported by multiple popular compilers such as GNU, Gcc, Intel icc so ~~and~~ on and so forth.
- It is widely used as the syntax of the program remains similar to the sequential execution of the task, with only specific parts of the code augmented with computer directives to specify parallelism.
- For a program that induces parallelism using OpenMP, there are specific regions in the source code wherein the code is executed simultaneously on multiple threads.

Every thread has its own program counter and executes one instruction at a time, similar to sequential execution. All such executions are then synchronized and aggregated to obtain a solution to a given problem.

Syntax: #pragma omp construct [clause[clause]...]

/* code to be executed in parallel goes here */

- OpenMP applies parallelism to structured blocks only and requires that its header file be included.
- Header file: #include <omp.h>
- The OpenMP program starts with one thread which is referred to as the master thread, executing the program in a sequential manner. When the compiler directive for parallel execution is encountered, threads are created for the same.
- These threads are referred to as the slave threads and are used to execute the decomposed task in parallel.
- All slave threads execute the code specified in the structured block of the compiler directive simultaneously. After that the execution of individual threads is synchronized, i.e. the work of slave threads is merged with that of master thread.
- Master thread may also execute the aforementioned structured block.

It is necessary that all slave threads complete execution before control is passed to the master thread.

```
#include <omp.h>
#include <iostream>
int main()
{
    #pragma omp parallel {
        int tid = omp_get_thread_num();
        cout<<"Hello" <<tid;
  }  }
```

The above example is a sample OpenMP program in which every thread has a unique ID with master thread having ID = 0.

- omp_get_thread_num(): returns the total number of threads.
- omp_set_thread_num(X): specifies x threads to be used for the parallel execution of structured block.

→ Data Environment for OpenMP programs:

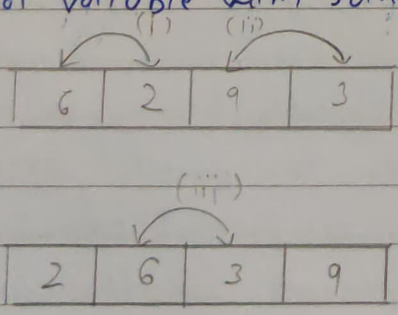
- Global variables (declared outside the scope of a parallel region) are shared among threads unless explicitly made private.
- Automatic variables declared within parallel region scope are private.
- Stack variables declared in functions called from within a parallel regions are private.

# pragma omp parallel private (x)

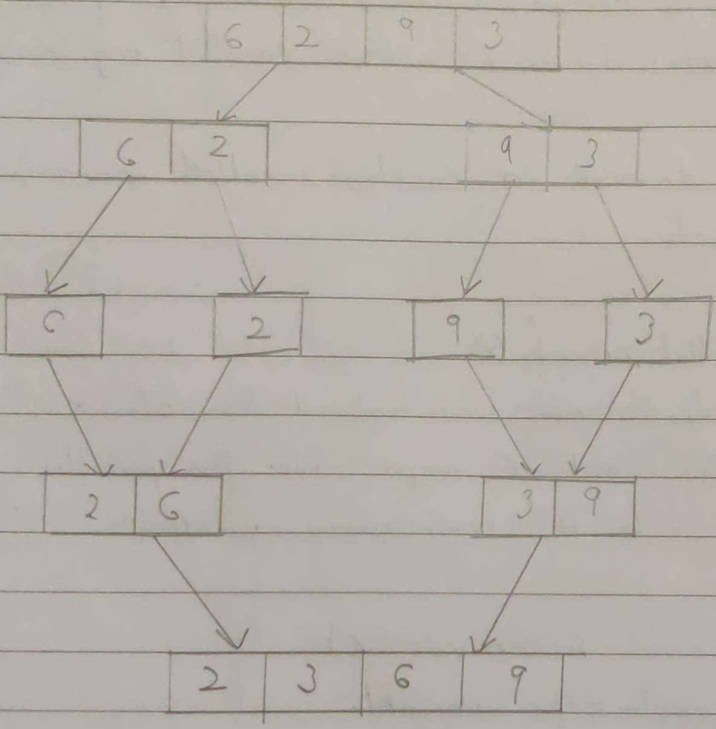
- each thread receives its own uninitialized variable x.
- the variable x falls out of scope after the parallel region.
- a global variable with the same name is unaffected (3.0 and later)

# pragma omp parallel firstprivate (x)

- x must be a global-scope variable.
- each thread receives a by-value copy of x.
- the local x's fall out-of scope after the parallel region.
- the base global variable with same name is unaffected.



- In bubble sort parallel implementation, each bubble will be executed simultaneously, i.e. in the given example 6 and 2 will be compared along with 9 and 3 in parallel.
- In merge sort parallel implementation, the array will be continuously divided until it cannot be divided further and every comparison will be made simultaneously by all threads. Here 6 and 2 along with 9 and 3 will be compared and shuffled simultaneously. Similarly, all other comparisons are made.



→ Algorithm:

↳ Bubble Sort:

- Include omp.h header file.
- Initialize vector/array of elements, keeping the number of elements large.
- Initialize number of threads for parallel execution.
- Specific compiler directive for parallel execution. In this case we will perform each comparison simultaneously, that is comparison of 1st and 2nd element.

3rd and 4th element so on & so forth will be done simultaneously

→ Merge sort:
- Include omp.h header file.
- Initialize number of threads for parallel execution.
- Specific compiler directive for parallel execution. In this case every half of the divided array will be sorted in parallel as for merge sort we continue to divide the array until it cannot be further divided and then make comparisons. All such comparisons will be done by threads simultaneously.

Test Cases

Operation	Input:	Output
Bubble (Par)	240, 301, 479, 886,	13, 18, 33, 37, ... time = 0.00239
Bubble (Seq)	856, 623, 905,	time = 5.4437e-05
Merge (Par)	270, 981, 371, 180,	0.000418623
Merge (Seq)	828, 597, 747, 690, ...	4.0197e-05

Conclusion: Successfully executed bubble sort and merge sort using OpenMP.