Multithreaded Programming and Synchronization

Programming #2

Task

Write a program that uses multiple threads to speed up the samplesort algorithm

Multithreaded program

- You can either use
  - *Semaphores for synchronization and mutual exclusion, or*
  - *Mutex locks for mutual exclusion and condition variables for synchronization*

Objectives

An assessment task related to ILO4 [Practicability]

A learning activity related to ILO 2

To learn how to use POSIX pthreads and semaphore libraries

- have hands-on practice in designing and developing multithreading programs
- create, manage, and coordinate multiple threads in a shared memory environment
- design and implement synchronization schemes for multithreaded processes using semaphores, mutex locks and condition variables.

Sequential Samplesort

1. Choose ***b*** samples from the data sequence.

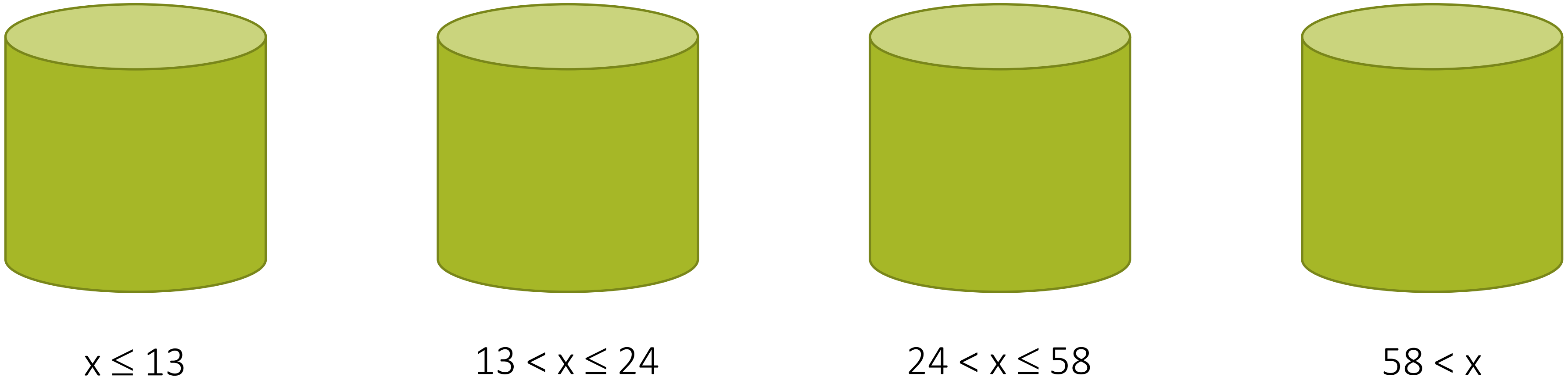
2. Sort the *b* samples such that $s_1 < s_2 < s_3 < \cdots < s_b$.

3. Partition the data sequence into ***b*** + 1 subsequences (i.e. buckets) based on the samples, such that every element in the j^{th} subsequence is greater than s_{j-1} and smaller than or equal to s_j . We assume $s_0 = -\infty$ and $s_{b+1} = +\infty$.

4. Sort each subsequence.

5. Concatenate the sorted subsequences to form the sorted data sequence.

42	98	2	31	86	87	5	**13**	99	44	67	37	17	7	87	3	96	71	40	19	**58**	13	61	77	11	13	6	81	76	18	**24**	14	63	59	99	17	36	84	1	48



1	2	3	5	6	7	11	13	13	13

14	17	17	18	19	24

31	36	37	40	42	44	48	58

59	61	63	67	71	76	77	81	84	86	87	87	96	98	99	99

How many buckets to use?

How to select samples?

Parallelizing Samplesort

Samplesort is very well suited and intuitive for parallelization and scaling

The number of buckets equals the number of threads (p)

Assume there are n numbers to be sorted and n >> p

Selection of p-1 samples using oversampling in a distributed manner

Parallel Samplesort - 5 phases

Phase 1

Each worker thread gets $^{n}/_{p}$ numbers

Uses sequential quicksort to sort its 'local' sequence

Each worker selects p samples from its 'local' sequence at indices

$0,\ ^{n}/_{p^2},\ ^{2n}/_{p^2},\ \cdots\ ,\ ^{(p-1)n}/_{p^2}$

Concurrently

42	98	2	31	86	87	5	13	99	44	67	37	17	7	87	3	96	71	40	19	58	13	61	77	11	13	6	81	76	18	24	14	63	59	99	17	36	84	1	48

Thr0

42	98	2	31	86	87	5	13	99	44
2	5	**13**	31	42	**44**	86	**87**	98	99

Thr1

67	37	17	7	87	3	96	71	40	19
3	7	**17**	19	37	**40**	67	**71**	87	96

Thr2

58	13	61	77	11	13	6	81	76	18
6	11	**13**	13	18	**58**	61	**76**	77	81

Thr3

24	14	63	59	99	17	36	84	1	48
1	14	**17**	24	36	**48**	59	**63**	84	99

Parallel Samplesort - 5 phases

Phase 2

Main thread gathers all p × p samples and sorts them

Selects p-1 pivots from the sorted samples at indices: $p + \lfloor p/2 \rfloor - 1,\ 2p + \lfloor p/2 \rfloor - 1,\ \cdots\ ,\ (p-1)p + \lfloor p/2 \rfloor - 1$

2	13	44	87	3	17	40	71	6	13	58	76	1	17	48	63

1	2	3	6	13	13	17	17	40	44	48	58	63	71	76	87

1	2	3	6	13	13	17	17	40	44	48	58	63	71	76	87

Parallel Samplesort - 5 phases

Phase 3

Each worker gets the p-1 pivot values

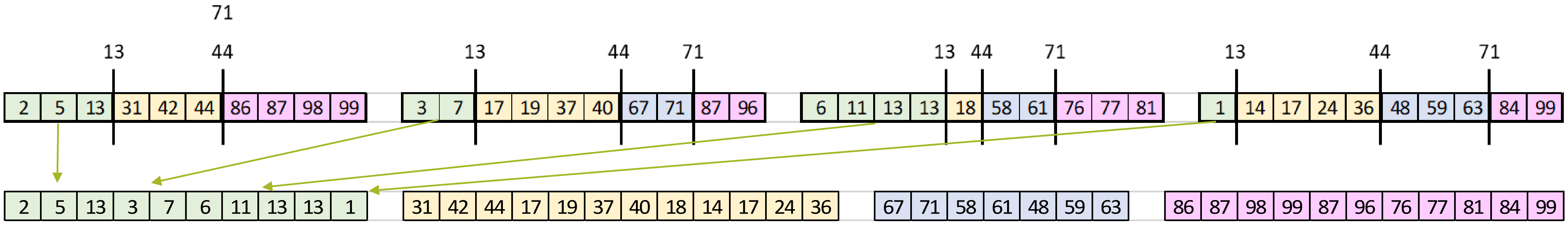
Partitions its 'local' sorted numbers into p disjoint pieces

Concurrently

Phase 4

Worker ***i*** keeps its ***i****th* partition and collects from other threads their ***i****th* partitions and merges them into a single sequence

Concurrently



Parallel Samplesort - 5 phases

Phase 5

- Each worker sorts its sequence
- Main thread gets the final sorted sequence

Concurrently

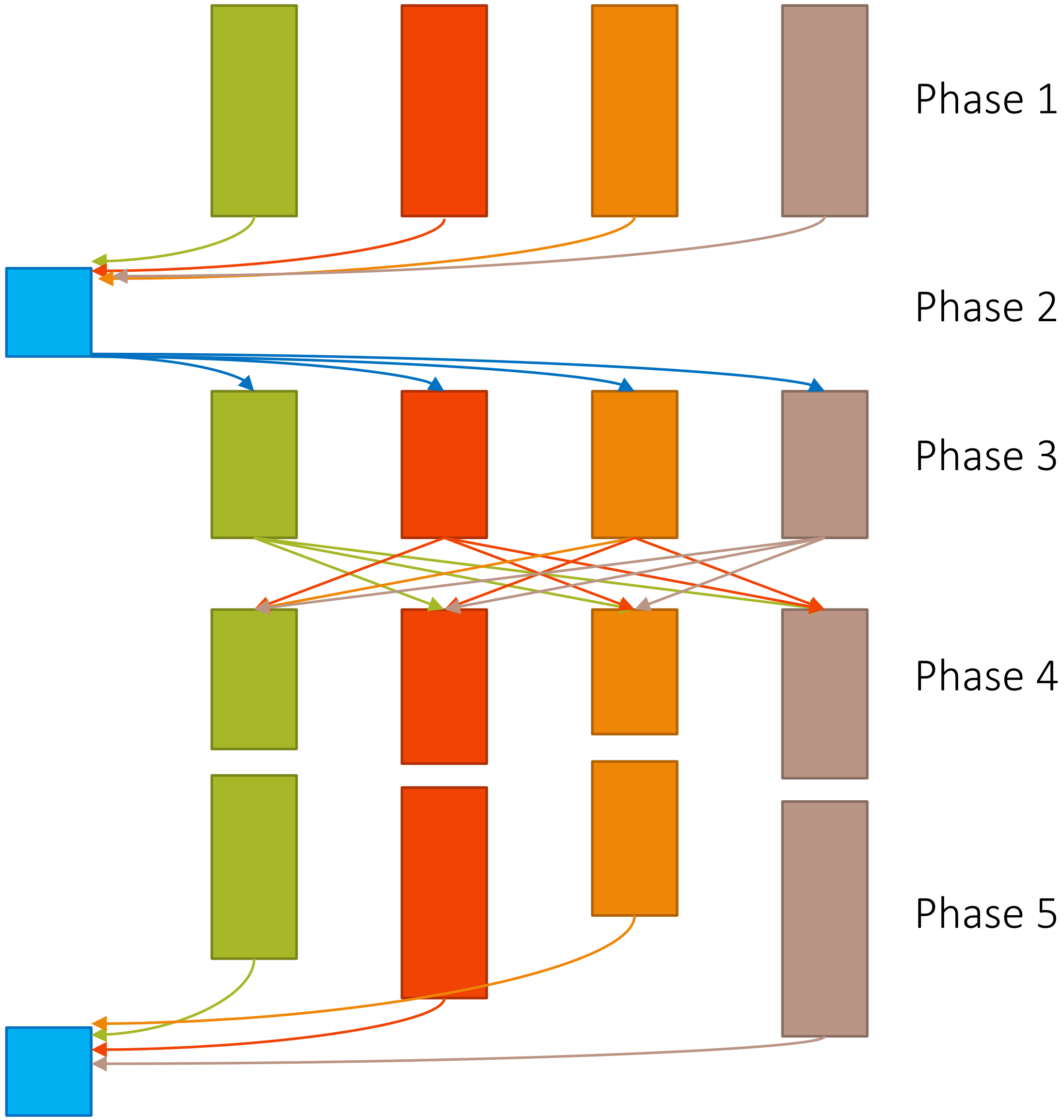
1	2	3	5	6	7	11	13	13	13

14	17	17	18	19	24	31	36	37	40	42	44

48	58	59	61	63	67	71

76	77	81	84	86	87	87	96	98	99	99

1	2	3	5	6	7	11	13	13	13	14	17	17	18	19	24	31	36	37	40	42	44	48	58	59	61	63	67	71	76	77	81	84	86	87	87	96	98	99	99



Summary

seqsort.c

```c
int main (int argc, char **argv)
{
  long i, j;
  struct timeval start, end;

  if ((argc != 2))
  {
    printf("Usage: seq_sort <number>\n");
    exit(0);
  }

  size = atol(argv[1]);
  intarr = (unsigned int *)malloc(size*sizeof(unsigned int));
  if (intarr == NULL) {perror("malloc"); exit(0); }

  // set the random seed for generating a fixed random
  // sequence across different runs
  char * env = getenv("RANNUM");  //get the env variable
  if (!env)                       //if not exists
    srandom(3230);
  else
    srandom(atol(env));

  for (i=0; i<size; i++) {
    intarr[i] = random();
  }

  // measure the start time
  gettimeofday(&start, NULL);

  // just call the qsort library
  // replace qsort by your parallel sorting algorithm
using pthread
  qsort(intarr, size, sizeof(unsigned int), compare);

  // measure the end time
  gettimeofday(&end, NULL);

  if (!checking(intarr, size)) {
    printf("The array is not in sorted order!!\n");
  }

  printf("Total elapsed time: %.4f s\n", (end.tv_sec -
start.tv_sec)*1.0 + (end.tv_usec -
start.tv_usec)/1000000.0);

  free(intarr);
  return 0;
}
```

```
root@1a8d95e9d93a:/home/c3230# gcc seqsort.c -o seqsort
root@1a8d95e9d93a:/home/c3230# ./seqsort 100000
First : 29844
At 25%: 537515580
At 50%: 1074491439
At 75%: 1610064405
Last  : 2147482304
Total elapsed time: 0.0223 s
root@1a8d95e9d93a:/home/c3230# ./seqsort 100000
First : 29844
At 25%: 537515580
At 50%: 1074491439
At 75%: 1610064405
Last  : 2147482304
Total elapsed time: 0.0228 s
root@1a8d95e9d93a:/home/c3230# export RANNUM='9876'
root@1a8d95e9d93a:/home/c3230# ./seqsort 100000
First : 2351
At 25%: 538105243
At 50%: 1074266763
At 75%: 1611575065
Last  : 2147455648
Total elapsed time: 0.0235 s
root@1a8d95e9d93a:/home/c3230# export RANNUM='3109'
root@1a8d95e9d93a:/home/c3230# ./seqsort 100000
First : 19740
At 25%: 537365138
At 50%: 1075734945
At 75%: 1607485034
Last  : 2147464298
Total elapsed time: 0.0226 s
root@1a8d95e9d93a:/home/c3230# unset RANNUM
root@1a8d95e9d93a:/home/c3230# ./seqsort 100000
First : 29844
At 25%: 537515580
At 50%: 1074491439
At 75%: 1610064405
Last  : 2147482304
Total elapsed time: 0.0238 s
root@1a8d95e9d93a:/home/c3230#
```

Your task

1. Duplicate a copy of the seqsort.c program and change its name to psort.c
2. Accept one ***optional*** input argument
   - ./psort <number> **[<no_of_workers>]**
   - default value of no_of_workers is 4
3. Implement the parallel Samplesort algorithm by using multiple threads

```
// measure the start time
gettimeofday(&start, NULL);

// add the parallel implementation of the samplesort algorithm here

// measure the end time
gettimeofday(&end, NULL);
```

4. The random sequence is accessible via the pointer ***intarr*** before the sorting and we expect you to use the same pointer ***intarr*** to point to the final sorted sequence

Measure the multithreaded program

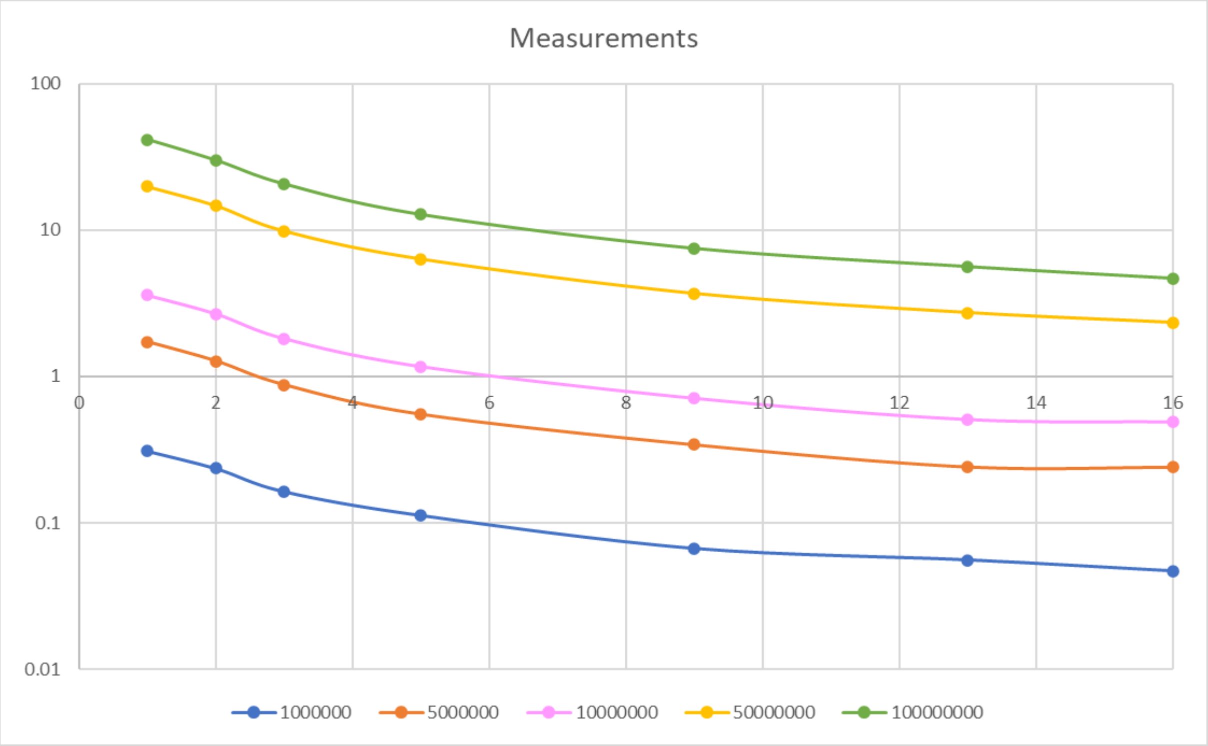
Run the programs on Workbench2 Linux server.

	No. of worker threads	Data size
seqsort	NA	1,000,000, 5,000,000, 10,000,000, 50,000,000, 100,000,000
psort	2, 3, 5, 9, 13, 16	1,000,000, 5,000,000, 10,000,000, 50,000,000, 100,000,000

Complete and submit the measure.xlsx file.

Sample Measurements

	1000000	5000000	10000000	50000000	100000000
1	0.3087	1.7186	3.5992	19.8209	41.5514
2	0.2349	1.2732	2.6839	14.679	30.0242
3	0.1633	0.8765	1.8161	9.8536	20.6842
5	0.1123	0.5514	1.1674	6.3372	12.7964
9	0.0668	0.3415	0.7075	3.6856	7.5123
13	0.0557	0.241	0.506	2.7264	5.6416
16	0.0469	0.2395	0.4865	2.3305	4.6927



Computer platform to use

You can develop and test your program on any Linux/Mac/WSL platform.

Finally test and benchmark the programs on Workbench2 server

Your program should be written in C and successfully compiled with gcc

Your submission will be primarily tested under workbench2 server. Make sure that your program can be compiled *without any error*. Otherwise, we have no way to test your submission and thus you may lose *all* the marks

Submission of Assignment

You should name your program to "psort_StudentNumber.c" (replace StudentNumber with your HKU student number) and the measurement excel file to measure_StudentNumber.xlsx

Submit your program and excel file to the course's Moodle web site (to Programming #2 submission page)

Add your signature in the header of the submitted program and make clear documentation

```
/*************************************************************
* Filename: psort_3015234567.c
* Student name and Number: Harry Potter 3015234567
* Development platform: WSL2
* Remark: Complete all features
*************************************************************/
```

Grading Criteria

You can use either semaphores or (mutex locks and condition variables) for concurrency and synchronization control. They will be graded without a difference.

Please read the assignment document

Documentation (1 point)	Include necessary documentation to clearly indicate the logic of the program Include required student's info at the beginning of the program
Measurement (1 point)	The performance of the seqsort and psort programs on the Workbench2 server.
psort.c (8 points)	Workers execute in parallel and with any number of worker threads and data sizes
	Correct result
	Threads terminate successfully and allow main thread to collect all sorted sequences
	Main thread wait for all workers to terminate before displaying the results