SWE3021 Multicore Programming - Midterm Exam Fall 2018

Student ID	Name

For Instructor/TA only,

Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Total

Academic Honor Pledge

I promise or affirm that I will not at any time be involved with cheating or plagiarism while enrolled as a student in Operating Systems class at Sungkyunkwan University. I understand that violation of this code will result in penalties as severe as Indefinite suspension from the University

Your Signature: ______________________

1. True or False: Choose the correct answer [30 pts]

		T	F
(a)	A parallel program is a concurrent program.	☑	☐
(b)	A concurrent program is a parallel program.	☐	☑
(c)	Strong scaling is defined as how the solution time varies with the number of processors for a fixed problem size per processor.	☐	☑
(d)	Thread-level parallelism exploit pipelining, superscalar capability, and branch prediction.	☐	☑
(e)	Exploratory decomposition is suitable for class divide-and-conquer problems.	☐	☑
(f)	The length of critical path is determined by the number of available processors.	☑	☐

2. The following code is designed to compute the sum of array elements in parallel. However, this code is not scalable because of two problems. What are the two problems? How can you resolve the problems? [10 pts]

```
int global; // global counter
pthread_mutex_t glock; // global lock
int local[MaxThreads]; // local counter per thread
int array[N];

void* thread_func(void* arg) {
    threadID = (int) arg;

    for(int i= threadID*K; i < (threadID+1)*K; i++){
        local[threadID] += array[i];
        if (local[threadID] >= THRESHOLD) { // transfer to global
            pthread_mutex_lock(&glock);
            global += local[threadID];
            pthread_mutex_unlock(&glock);
            local[threadID] = 0;
        }
    }
}
```

First problem:
synchronization inside loop

Second problem:
false sharing

3. Suppose a program is divided into 7 tasks. One task at the beginning and another task at the end cannot be concurrently executed with other tasks and they take 3 ms and 4 ms respectively. Between these two tasks, there are 5 tasks that can be executed concurrently and each of these tasks takes 16 ms. [20 pts]

(a) What is maximum speed-up according to Amdahls law?

Your answer:
Amdahl's Law: S=1/f - 87/7

(b) What is the largest number of processes we can use after which no speedup will be seen?

Your answer:
5 processors

(c) If we use that largest number of processes you calculated for (b), what is the speedup and what is the efficiency? Do not use Amdahl's law in your calculations.

Speedup:
87/23

Efficiency:
87/115

(d) Is there a difference between the speedup you calculated in (c) and (a)? If so, why is that.

Your answer:
Even if we use more processors, the execution time of 5 tasks is still 5 ms.

4. The following termination detection algorithm does not work under a particular circumstance. What is the condition? How can you fix it? [10 pts]

```
pid = get_my_pid();
start_termination_detection();
do{
  while(work_available()){
    work_a_bit();
    auto message = non_blocking_get_message_from( (pid-1+N)%N );
    if(message != NULL && message.type == TOKEN){
       // passes busy token
       send_message_to( (pid+1)%N, BUSY);
    }
    if(pending_work_request()){
        send_some_work((pid+1)%N);
    }
  }
  if(pid == 0) send_message_to(1, IDLE);

  auto message = blocking_get_message_from( (pid-1+N)%N );
  if(message.type == WORK){
    // received additional workload from previous process
    continue;
  }
  else if(message.type == TOKEN){
      if(message.token == IDLE && pid == 0) {
          // P0 received an idle token. Let's terminate.
          send_message_to( 1, TERMINATE);
          break;
      }
      else if(message.token == IDLE) send_message_to( (pid+1)%N, IDLE);
      else send_message_to( (pid+1)%N, BUSY);
  }
  else if(message.type == TERMINATE){
      send_message_to( (pid+1)%N, TERMINATE);
      break;
  }
} while(1);
```

Answer:
Fast token and slow work problem. We need FIFO message channels to fix it.

5. Given the following sequence of numbers:

   `<3, 7, 4, 8, 6, 2, 1, 5>`

   Show the steps of parallel bitonic sorting using 8 processors. [10pts]

   Answer:
   3 7 8 4 2 6 5 1
   3 4 7 8 6 5 2 1
   3 4 2 1 6 5 7 8
   2 1 3 4 6 5 7 8
   1 2 3 4 5 6 7 8

6. Given the following sequence of numbers:

```
Value:      7   2   4   1   6
Binary:   111 010 100 001 110
```

Show the steps (bits, histogram, prefix sum, index) of parallel radix sort to produce a sorted set of numbers. [10pts]

Answer:

```
     bit 0: 1 0 0 1 0
 histogram: 3 2
prefix sum: 0 3
    offset: 3 0 1 4 2
   results: 2 4 6 7 1

     bit 1: 1 0 1 1 0
 histogram: 2 3
prefix sum: 0 2
    offset: 2 0 3 4 1
   results: 4 1 2 6 7

     bit 2: 1 0 0 1 1
 histogram: 2 3
prefix sum: 0 2
    offset: 2 0 1 3 4
   results: 1 2 4 6 7
```

7. What is the average degree of concurrency for parallel shear sorting when N^2 processors sort N^2 numbers? [10 pts]

Answer:
Sequential sorting: $n^2 logn$ steps
parallel shear sorting: $nlogn$ steps
Average degree of concurrency: $n^2 logn/nlogn = n$

8. Write a parallel function `void reverse(int arr[], int size)` in pseudo code (e.g., parallel_for i in 1..N), Pthread, or OpenMP that reverses the order of an integer array of size N. Constraint 1: The size of integer array can be as large as the available free memory space. So, your function has to perform in-place updates, i.e., you are not allowed to use any other large array. Constraint 2: For performance reasons, you are not allowed to synchronize threads. Constraint 3: Your machine has P processors where P is much smaller than N (P << N). [10 pts]

Your answer:

```
#pragma omp parallel for private(temp)
  for(int i=0; i<N/2; i++){
      temp = a[N/2+i];
      a[N/2+i] = A[i];
      a[i] = temp;
  }
```