2018.1 Human Media Multicore Computing
Midterm Exam (7pm-8:20pm, 4/23)

supervisor signature	

StudentID# : ( ) , Name : ( )

※ You may answer either in English or Korean language unless instructed to answer in English.

1. (22 points) Fill out the blanks (a)~(m) with **the most appropriate English words**. x_____ means the word starts with alphabet 'x'.

- Abbreviation 'GPGPU' stands for
  (a. G_____) (b. P_____) computing on Graphics Processing Units.

- The usual way to avoid a race condition in a multi-threaded program is (c.).

- Parallelism overheads mainly include
  - cost of (d.),
  - cost of (e.),
  - cost of (f.),
  - and extra (redundant) computation

- (g.) means federation of computer resources from multiple locations to reach a common goal. It tends to be more loosely coupled, (h.), and geographically dispersed.

- Currently, CPU clock speed is not increasing rapidly because (i.) is too high to be tolerated in high clock speed.

- A cluster computer contains multiple PCs that are connected together with (j.), generally.

- Race condition is a type of flaw in an electronic or software system where (k.) the sequence or timing of other uncontrollable events.

2. (12 points) Determine whether each of following statements is True or False. If you are not sure, you may not give your answer, because if your answer is wrong, penalty (score subtraction: -2 points) is applied.

(a) Each thread in one process shares global variables in a multi-threaded program. (True / False)

(b) One of the main disadvantages of shared memory parallel systems is the lack of scalability, meaning it is difficult to add large number of CPUs for performance enhancement. (True / False)

(c) Coarse-grain parallelism implies relatively high communication and synchronization overhead compared to fine-grain parallelism. (True / False)

(d) Moore's law states that potential program speedup is limited by the fraction of code that can be parallelized. (True / False)

3. (18 points) Answer to following questions.

(a) What is the meaning of "**Blocking**" in **ArrayBlockingQueue** which is available in Java Concurrency Utilities. Your answer should contain '**what happens when**'.
( )

(b) What is the main difference between **HashMap** and **ConcurrentHashMap** in Java. Explain with sufficient details.
( )

(c) What does "**X.join();**" do in JAVA code? Explain with sufficient details.
( )

(d) What does "**wait();**" do in JAVA code? Explain with sufficient details.
( )

(e) What does "**notify();**" do in JAVA code? Explain with sufficient details.
( )

(f) What is 'cache coherence' problem? Explain with sufficient details.
( )

5.(48 points) (a) Following multi-threaded java codes compute and display the sum of each element in an array with size **NUM_END** (**int_arr**) that is initialized as {1, 2, 3,…, **NUM_END**}. (Assume **NUM_END** is divisible by **NUM_THREAD**.) There can be two approaches, a naive approach and a divide-and-conquer approach, we learned in our lectures. In the naive approach, the entire array is divided into **NUM_THREAD** sub-arrays and each thread simultaneously computes the sum of its assigned sub-array. The main thread adds together each thread's answer for the final result. In the divide-and-conquer approach, a thread recursively creates two threads (left and right) to compute the sum of left half and right half of the array, respectively until reaching at some point. Fill out empty boxes below with appropriate JAVA codes for the naive approach and divide-and conquer approach..

----------< source code >----------

< Naive Approach>

```
// Naive Approach
class SumThread extends Thread {
  int lo; // fields for communicating inputs
  int hi;
  int[] arr;
  int ans = 0; // for communicating result
  SumThread(int[] a,int l,int h) { lo=l; hi=h; arr=a; }
  public void run() {
```

```
  }
}

class ex2 {
  private static int NUM_END=10000;
  private static int NUM_THREAD=4;

  public static void main(String[] args) {
    int[] int_arr = new int [NUM_END];
    int i,s;
    for (i=0;i<NUM_END;i++) int_arr[i]=i+1;
    s=sum(int_arr);
    System.out.println(s);
  }

  static int sum(int[] arr) {
```

```
    return ans;
  }
}
```

< Divide-and-Conquer (Recursive) Approach >

```
// Divide-and-Conquer (Recursive) Approach
class SumThread extends Thread {
  int lo; // fields for communicating inputs
  int hi;
  int[] arr;
  int ans = 0; // for communicating result
  private static int SEQUENTIAL_CUTOFF=100;
  SumThread(int[] a,int l,int h) { lo=l; hi=h; arr=a; }
  public void run() {
```

```
  }
}

class ex2 {
  private static int NUM_END=10000;

  public static void main(String[] args) {
    int[] int_arr = new int [NUM_END];
    int i,s;
    for (i=0;i<NUM_END;i++) int_arr[i]=i+1;
    s=sum(int_arr);
    System.out.println(s);
  }

  static int sum(int[] arr) {
```

```
    return ans;
  }
}
```

(b) What is the main problem (drawback) of the naive approach? Why does such problem occur?. Explain what and why.

( )

(c) What is the meaning of **SEQUENTIAL_CUTOFF** in the divide-and-conquer code? Explain with sufficient details.

( )

(d) Explain what problem may occur if **SEQUENTIAL_CUTOFF** is too small in the divide-and-conquer code?

( )