Part I: Choose the most accurate correct answer. (0.5 mark each)) [6 mark

1. In ________ we partition the tasks used in solving the problem among the cores, and each co carries out different operations on its part of the data.

   A. data-parallelism C. multithreading parallelism
   B. task-parallelism D. hybrid of task- and data-parallelism

2. In java, ________ is recommended to manage a small number of tasks executing concurrently.
   A. Thread Class
   B. Executers Class
   C. Runnable Class
   D. Concurrent Class

3. In MPI, a communicator is ________

   A. a collection of processes that can share the same memory.
   B. a collection of functions that can be invoked in parallel.
   C. a collection of processes that can send messages to each other.
   D. a collection of threads that can be controlled by the main thread.

4. The fairness policy that guarantees no starvation is ________

   A. ReentrantLock (true)
   B. ReentrantLock (false)
   C. ReentrantLock ()
   D. ReentrantLock (synchronize)

5. In single-processor systems, the multiple threads share CPU time known as ....
   A. Task sharing.
   B. Distributed memory.
   C. Multi-processor.
   D. Time sharing.

6. A program that may need to cooperate with other programs on a network to solve a problem is referred to as ....
   A. serial computing
   B. concurrent computing
   C. parallel computing
   D. distributed computing

7. MPI is suitable for a .... memory system.
   A. distributed
   B. shared
   C. hybrid
   D. serial

Part 1: Choose the most accurate correct answer. (0.5 mark each) [6 mar

1. In …………, we partition the tasks used in solving the problem among the cores, and each carries out different operations on its part of the data.

   A. data-parallelism C. multithreading parallelism
   B. task-parallelism D. hybrid of task- and data-parallelism

2. In java, ………….. is recommended to manage a small number of tasks executing concurrently.
   A. Thread Class
   B. Executers Class
   C. Runnable Class
   D. Concurrent Class

3. In MPI, a communicator is …………………

   A. a collection of processes that can share the same memory.
   B. a collection of functions that can be invoked in parallel.
   C. a collection of processes that can send messages to each other.
   D. a collection of threads that can be controlled by the main thread.

4. The fairness policy that guarantees no starvation is ________

   A. ReentrantLock (true)
   B. ReentrantLock (false)
   C. ReentrantLock ()
   D. ReentrantLock (synchronize)

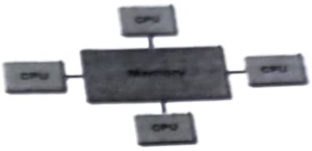
5. In single-processor systems, the multiple threads share CPU time known as ………
   A. Task sharing.
   B. Distributed memory.
   C. Multi-processor.
   D. Time sharing.

6. A program that may need to cooperate with other programs on a network to solve a problem is referred to as ………………
   A. serial computing
   B. concurrent computing
   C. parallel computing
   D. distributed computing

7. MPI is suitable for a ………………. memory system.
   A. distributed
   B. shared
   C. hybrid
   D. serial

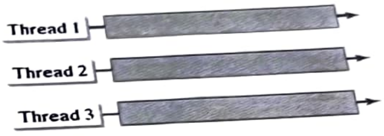
8. Given a parallel architecture (figure on the right). The kind of coordination that may be required includes...........

A. synchronization
B. pipelining
C. communication
D. A and B.



9. The figure (on the right) represents ....

A. serial execution of threads in a multiprocessor system.
B. execution of threads simultaneously in a multiprocessor system.
C. execution of threads concurrently in a single-core system.
D. execution of threads simultaneously in a single-core system.

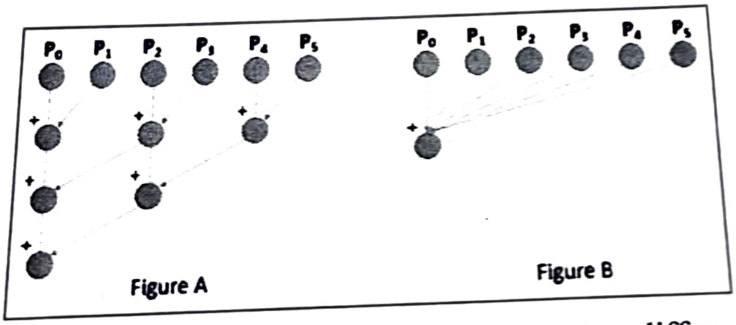


10. Coordination among multiple cores in a shared-memory system is achieved through which of the following?

A. Examining and updating shared memory locations.
B. Exchanging messages across a network.
C. Sending messages via pipelines.
D. B and C.

11. Identify an example of a shared-memory machine among the following.

A. Single-core desktop.
B. Multicore laptop.
C. Cluster of computers.
D. Cloud of servers.



12. Given Figures A and B representing two different approaches of global summation, wh them performs faster for large number of processes?

A. Figure A
B. Figure B
C. Both perform the same
D. The number of processes has no effect on performance.

Part 2: **Answer the following four questions.** (1 mark each) [4 marks]

Question 1) Based on MPI implementations, specify which process(es) is (are) granted access to standard input and output?

Question 2) To restrict the number of threads that access a shared resource, java provides different ways. List at least two.

Question 3) Mention two reasons that make a java thread enters the **Blocked** state.

Question 4) Why cannot we write programs that convert (translate) serial programs into parallel programs? Mention one reason.

Part 3: Determine the **output** of the following code. **[2 marks]**

```
int number;

if (process_rank == 0) {

    number = 1;

    MPI_Send(&number, 1, MPI_INT, 2, 0, MPI_COMM_WORLD);

} else if (process_rank == 2) {

    MPI_Recv(&number, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNOR
E);

    printf("Process %d received number %d from process 0\n", process_ran
k, number);

}
```

1

Part 4: Answer the following **two** questions. [3 marks]

Question 1) Write the code segment that creates two threads **exactly** using thread pool. The first thread runs `BigTask` and the second `SmallTask`. (2 marks)

```
public static void main(String[] args) {
        ____________________________________________
        ____________________________________________
        ____________________________________________
        ____________________________________________
        ____________________________________________
        ____________________________________________
        ____________________________________________

    }

public static class BigTask implements Runnable{   //do big task}
public static class SmallTask implements Runnable{   //do small task}
```

Question 2) *Identify the critical region in the following code fragment, and apply a suitable asynchronization method* (1 mark)

```
public void run() {
              threadFact(threadRank);
        }

        void threadFact(int threadRank) {
            double my_fact=1;
            double Total_fact=1;
            int my_n = n / THREAD_COUNT;
            int my_first_i = my_n * threadRank;
            int my_last_i = my_first_i + my_n;
            double my_fact = 0.0;

            for (int i = my_first_i; i < my_last_i; i++) {
                my_fact = num*(num-1);
            }
            Total_fact*=my_fact;
        }
}
```