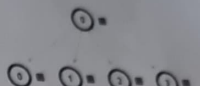
Part 1: Choose the most accurate correct answer.

1. Which of the following MPI functions does the operation in the figure → [6 marks]



   A. MPI_Reduce
   B. MPI_Allreduce
   C. MPI_Scatter
   D. MPI_Bcast

2. Suppose there are 3 variables x, y, and z stored in the memory; Determine the starting address of y and z, if the values of the displacement are:

```
array_of_displacement[ ] = {0, 50, 130}
```

Variable	Type	Starting Address
x	Integer	100
y	Integer	?
z	double	?

   A. 50, 130
   B. 150, 280
   C. 150, 230
   D. 130, 50

3. The code in an OpenMP program that is covered by a pragma is executed by _____.
   A. All threads in thread_count
   B. Slaves thread
   C. The master thread
   D. OpenMP programs don't use pragma

4. Vector processors is considered _____.
   A. SISD
   B. SIMD
   C. MIMD
   D. MISD

5. Suppose a Single Instruction Multiple Data (SIMD) hardware consisting of 400 threads is meant to process an array of 4800 elements, how many iterations will the hardware take to complete processing the array?
   A. 4610
   B. 11
   C. 12
   D. 400

6. The bisection width of a three-dimensional hypercube is ..... Links
   A. 8
   B. 3
   C. 2
   D. 1

Part 2: Answer the following: **[2 marks]**

Question 1:
OpenMP provides several mechanisms for preventing the race condition problem. List at least three.

.....
.....
.....
.....

Question 2: Fill in the blank:
A program that relies on MPI-provided buffering is said to be _____

.....
.....

Part 3: **[2 marks]**

Question 1:
Name two main approaches of Instruction Level Parallelism (ILP).

.....
.....
.....

Question 2:
Explain the difference between UMA and NUMA multicore systems.

.....
.....
.....
.....

Part 4: **[3 marks]**

Question 1: (1 mark)
Answer the following two questions:

a) Can the following loop be parallelized using OpenMP parallel for?

```
for(int i = 1; i < 100; i++) {
        array[i] = array[i] - 5;
    }
```

b) If the answer is Yes, write the code. If the answer is NO, explain why it cannot be parallelized.

Yes.
#pragma omp parallel for num_threads(thread_count)

Question 2: (1 mark)

Study the code below. If the thread_count = 3 and n = 12, the iterations assigned to thread1, thread2, and thread3 are _____ , _____, and _____ , respectively.

```
int n , A[n];
int i;

#pragma omp parallel for num_threads(thread_count) \
schedule <static, 2>
  for (i=0; i < n; i++)
      A[i] = i * 3;
```

أرقام الاتريتس اللي ختتخص لكل واحد فيهم

Note that the answer should follow the following.
thread1 (iteration#, iteration#, iteration#, iteration#)
thread2 (iteration#, iteration#, iteration#, iteration#)
thread3 (iteration#, iteration#, iteration#, iteration#)

Question 3: (1 mark)

Fill in the blanks (*A, an B*) in the following fragment of code to measure the elapsed time of execution of `Parallel_sum()` function.

```
int my_rank, comm_sz;
double start, finish, elapsed_time, max_elapsed_time;

MPI_Init(NULL, NULL);
MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);

//A- to ensure that all processes have been reached this point
.....(A).....
start = MPI_Wtime();
Parallel_sum();
finish = MPI_Wtime();
elapsed_time = finish - start;
MPI_Reduce(&.....(B)....., &max_elapsed_time,1, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD);

if (my_rank == 0) {
    printf("Elapsed time: %f", max_elapsed_time);
}
MPI_Finalize();
```

.....

.....

.....

.....

[2 marks]

Part 5:

Answer the following:

Given the following information (*run time of serial program = 80 seconds, run time of parallel program = 10 seconds, data size = 1000, number of processes = 4*), What is the **efficiency** of the parallel program?

$\frac{80}{4 \times 10} = 2$

.....

.....

.....

.....

Bonus Question:

Complete the output of the following MPI code when it is executed using three processes?

[1.5 marks]

```
void Trace() {
      int a = 2, b = 4, c = 0, d = 0;
      int comm_sz;
      int my_rank;
      MPI_Init(NULL, NULL);
      MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
      MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

            MPI_Reduce(&a, &c, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
            MPI_Reduce(&b, &d, 1, MPI_INT, MPI_SUM, 2, MPI_COMM_WORLD);


      printf("Process %d : c=%d and d=%d", my_rank, c, d);
      MPI_Finalize();
}
```

Output:

Process 0: c = ... and d = ...

Process 1: c = ... and d = ...

Process 2: c = ... and d =.....