1. Which performance metric measures the total time taken to execute a program?
A) MIPS
B) FLOPS
C) Wall Clock time
D) CPU time

2. Amdahl's Law is used to:
A) Measure the speed of a processor
B) Predict the maximum achievable speedup using parallel computing
C) Calculate the memory bandwidth
D) Estimate the number of cache misses

3. According to Amdahl's Law, if 20% of a program is sequential, and the remaining part is perfectly parallelizable, what is the maximum speedup achievable with infinite cores?
A) 1.2
B) 2
C) 5
D) 10

4. Gustafson's Law differs from Amdahl's Law in that it:
A) Accounts for the overhead of parallelization
B) Focuses on the speedup achievable with finite problem sizes
C) Assumes that the entire program can be parallelized
D) All of the above

5. If a program takes 100 seconds to execute on a single core and 20 seconds on four cores, what is the speedup?
A) 2
B) 4
C) 5
D) 20

6. The formula to calculate MIPS is:
A) MIPS = Clock speed / (1 million)
B) MIPS = Clock speed * (1 million)
C) MIPS = Instructions executed / (Execution time * 1 million)
D) MIPS = (Execution time * 1 million) / Instructions executed

7. Which of the following is not a factor affecting performance according to Amdahl's Law?
A) Number of cores
B) Execution time of parallelizable portion
C) Execution time of serial portion
D) Cache hit ratio

8. Assume 1% of the runtime of a program is not parallelizable. This program is run on 61 cores of an Intel Xeon Phi. Under the assumption that the program runs at the same speed on all of those cores, and that there are no additional overheads, what is the parallel speedup?

$$\frac{1}{0.01 + \frac{0.99}{61}} = 38.125$$

A) 38.125
B) 1.0099
C) 61
D) None of the above

9. Assume that the program invokes a broadcast operation. This broadcast adds overhead, depending on the number of cores involved. There is a broadcast implementation that adds a parallel overhead of 0.0001 × n? How much speedup will you achieve with 100 cores Below is a variant of Amdahl's law that include overhead as well represented by O

Assuming 1% of the runtime

$$speedup = \frac{1}{O + s + \frac{(1-s)}{n}}$$

A) 50
B) Greater than 40 and less than 50
C) Greater than 50 and less than 60
D) None of the above

10. Assume that the program invokes a broadcast operation. This broadcast adds overhead, depending on the number of cores involved. There is a broadcast implementation that adds a parallel overhead of 0.0001 × n? How much speedup will you achieve with 101 cores Below is a variant of Amdahl's law that include overhead as well represented by O

$$speedup = \frac{1}{O + s + \frac{(1-s)}{n}}$$

A) 50
B) Greater than 40 and less than 50
C) Greater than 50 and less than 60
D) None of the above

11. Assume that the program invokes a broadcast operation. This broadcast adds overhead, depending on the number of cores involved. There is a broadcast implementation that adds a parallel overhead of 0.0001 × n? How much speedup will you achieve with 99 cores Below is a variant of Amdahl's law that include overhead as well represented by O

$$speedup = \frac{1}{O + s + \frac{(1-s)}{n}}$$

A) 50
B) Greater than 40 and less than 50
C) Greater than 50 and less than 60
D) None of the above

12. The analysis of a program has shown a speedup of 3 when running on 4 cores. What is the serial fraction according to Gustafson's law?
A) 11.111111%
B) 22.222222%
C) 33.333333%
D) None of the above

13. The analysis of a program has shown a speedup of 3 when running on 4 cores. What is the serial fraction according to Amdahl's law (assuming best possible speedup)?
A) 11.111111%
B) 22.222222%
C) 33.333333%
D) None of the above

14. If a program achieves a speedup of 10x on 10 cores for a problem size of 1000, what is the expected speedup when the problem size is increased to 2000, assuming perfect scaling?
A) 10
B) 20
C) 100
D) 200

15. A program with a speedup of 2x on 4 cores compared to 1 core has an efficiency of:
A) 25%
B) 50%
C) 100%
D) 200%

16. What will be the output of the following code, when run on 2 processes

```
int main(int argc,char*argv[])
{
char message[20];
int i, rank, size;
MPI_Status status;
int root = 0;
MPI_Init(&argc, &argv) ;
MPI_Comm_size (MPI_COMM_WORLD,&size);
MPI_Comm_rank (MPI_COMM_WORLD,&rank);
if (rank==root){
strcpy(message,"Hello, PDC class");
MPI_Bcast(message,5,MPI_CHAR,root,MPI_COMM_WORLD);
printf("Message from %d process : %s",rank, message);
MPI_Finalize();
printf("\n");}
return 0;

}
```

A. Message from 1 process: Hello Message from 0 process: Hello, PDC class
B. Message from 1 process: Hello, PDC class Message from 0 process: Hello
C. Message from 1 process: Hello
D. Message from 0 Process: Hello, PDC class

17. What will be the output of the following code, when run on 2 processes

```
int main(int argc,char*argv[])
{
int i,rank,nproc;
int sendBuf[3];
int recvBuf[12]; //for 4 processes, 3 ints for each
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD,&nproc);
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
for (i=0;i<3;i++)
sendBuf[i]=rank;

MPI_Gather(sendBuf,3,MPI_INT,recvBuf,3,MPI_INT,0,MPI_COMM_WORLD);
if(rank==0) {
for(i=0;i<3;i++) {
printf("\nProcess # %d send data : %d",i/3, recvBuf[i]);
}
}
MPI_Finalize();
return 0;
}
```

A. Process # 0 send data : 0
   Process # 0 send data : 0
   Process # 0 send data : 0
B. Process # 1 send data : 0
   Process # 0 send data : 0
   Process # 0 send data : 0
C. Process # 0 send data : 1
   Process # 0 send data : 1
   Process # 0 send data : 1
D. Process # 0 send data : 0
   Process # 0 send data : 0
   Process # 0 send data : 0
   Process # 1 send data : 1
   Process # 1 send data : 1
   Process # 1 send data : 1

18. What will be the output of the following code, when run on 2 processes

```
int main(int argc, char **argv) {
    MPI_Init(&argc, &argv);
    int rank, size;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    int nstrings=5; // for 5 different processes
    const char *const strings[5] = {"This ","is","Sessional2","Exam","of PDC"};
    char *mystring = (char *)strings[rank];
    int mylen = strlen(mystring);
    const int root = 0;
    int *recvcounts = NULL;
    if (rank == root)
        recvcounts = malloc(size * sizeof(int)) ;
    MPI_Gather(&mylen, 1, MPI_INT,recvcounts, 1, MPI_INT,root, MPI_COMM_WORLD);
    int totlen = 0;
    int *displs = NULL;
    char *totalstring = NULL;
    int i=0;
    if (rank == root) {
        displs = malloc(size * sizeof(int));
        displs[0] = 0;
        totlen = totlen + (recvcounts[0]+1);
        for (i=1; i<size; i++) {
           totlen = totlen + (recvcounts[i]+1);
           displs[i] = displs[i-1] + recvcounts[i-1] + 1;
        }
        totalstring = malloc(totlen * sizeof(char));
        for (i=0; i<totlen-1; i++)
            totalstring[i] = ' ';
        totalstring[totlen-1] = '\0';
    }     MPI_Gatherv(mystring, mylen, MPI_CHAR, totalstring, recvcounts, displs,
MPI_CHAR, root, MPI_COMM_WORLD);
    if (rank == 1) {
        printf("Root process %d: <%s>\n", rank, totalstring);
        free(totalstring);
        free(displs);
        free(recvcounts);
    }
    if(rank==0)
    printf("\n I need to prepare well for finals ");
MPI_Finalize();
    return 0;
}
```

Too long not doing it :)

A. Root process 1: This is sessional2 Exam of PDC
I need to prepare well for finals
B. Root process 1: <(null)>
I need to prepare well for finals
C. Root process 0: <(null)>
I need to prepare well for finals
D. I need to prepare well for finals

19. In the work-sharing construct omp for, the initial chunk size by default will be:

a) total_no_of_remaining_iterations / total_threads
b) total_no_of_iterations / total_threads
c) total_no_of_iterations / total_cpus
d) total_no_of_remaining_iterations / total_cpus
e) same size as per chunk size mentioned with dynamic scheduling

20. Which of the following loop scheduling in OpenMP has less overhead and good load balancing:

a) default
b) static
c) dynamic
d) guided
e) default and static

21. Assume we have heterogeneous CPU in the machine, considering the machine architecture determine which loop scheduling will attain most of the load balance:

a) default
b) static
c) dynamic
d) guided
e) Not possible in heterogeneous CPU

22. Which of the following OpenMP construct creates the execution model similar to the SIMD execution model:

A. Parallel
B. Critical Section
C. Single
D. Master
E. None of the above

23. What will be the output of the following code:

```
int main(int argc, char* arv[]) {
  int i;
  int j;
  int n=10;
  int t=20;
  omp_set_num_threads(2);

  #pragma omp parallel private(j)
  {
      # pragma omp for lastprivate(t)
      for (i=0; i<n; i++) {
          printf("a\n");
         t=omp_get_thread_num();
      }

      for(j=0; j<n; j++) {
          printf("b\n");
         t=omp_get_thread_num()+2;
      }
  }

 return 0;
}
```

a) 10 a's and 20 b's
b) 20 a's and 10 b's, 3
c) 10 a's and 20 b's, 3
d) 10 a's and 20 b's, 1
e) 20 a's and 10 b's, 1

24. What will be the output of the following code:

```
int main (int argc, char *argv[])
{
       int i, p,
       x[5] = {1,2,3,4,5},
       y[5]={5,4,3,2,1};
       #pragma omp parallel num_threads(5) private (p)
       {
              p=5;
              #pragma omp for
              for (i=0; i<5; i++)
                     x[i]+=y[i]+p;

       } /* omp end parallel */

       for (i=4; i>=0; i-=3)
              printf("%d    ",x[i]);
       return 0;
}
```

a) 11 11 11
b) 11 11 11 11
c) 10 10 10
d) 10 10 10 10
e) None of the above

25. What will be the output of the following code:

```
int main (int argc, char *argv[])
{
      int i;
      int var = 1;
      #pragma omp parallel num_threads(5) reduction(+:var)
      {
            var = var + omp_get_num_threads() + omp_get_thread_num();
      }

      printf("The output is %d\n", var);
      return 0;
}
```

a) 32
b) 33
c) 34
d) 35
e) None of the above

26. What will be the output of the following code:

```
int main (int argc, char *argv[])
{
   int j, num, flag,rem,res;
   #pragma omp parallel num_threads(4) private(num,res,rem)
   {
     int i=1634;
     rem=0,res=0;
     i+=omp_get_thread_num();
     num=i;

     while (num != 0) {
           rem = num % 10;
           res += (rem * rem * rem * rem);
           num /= 10;
     }

     if(!omp_get_thread_num())
           printf("number= %d \n",res);
   }
   return 0;
}
```

a) 1634
b) 4361
c) 3461
d) 6134
e) 6314

27. What will be the output of the following code:

```
int main()
{
  int i;
  int value = 2;
   #pragma omp parallel for num_threads(4) lastprivate(value) shared(i)
ordered
    for (i=1; i <= 4; i++)
            value = omp_get_thread_num();

   printf("The value is %d\n", value);
   return 0;
}
```

a) 2
b) 3
c) 4
d) 5
e) Run-time error

28. Which of the following construct enables the execution of the loop iterations in the same sequence as they are executed in a serial program.
A. nowait
B. collapse
C. schedule (static)
D. omp for
E. None of the above

29. What does the **nowait** clause do?
A. Skips to the next OpenMP construct
B. Prioritizes the boost CPU clock
C. Removes implicit barrier from work-sharing construct
D. Removes implicit barrier from parallel region

30. When we execute the following code, the code will have:

```
int main() {
    int i;
    int sharedInt = 1;
    #pragma omp parallel num_threads(1000) {
            #pragma omp master
            sharedInt = sharedInt + omp_get_thread_num( );

    }
    printf("value: %d\n",sharedInt); return 0;
}
```

A. Deadlock
B. Data-race
C. Run-time error
D. Syntax error
E. None of the above

31. When will be the output of the following code:

```
int main() {
    int arr[12] = {-2,0,0,0,0,0,0,0,0,0,0,0};
    int x=0, y=0, j;

     omp_set_num_threads(6);
     #pragma omp for schedule(dynamic,2) private(j)
     for(j=0; j<12; j++)
     {
            x += arr[j] + omp_get_thread_num();
     }

     #pragma omp parallel for schedule(static,4)
     for(j=11; j>0; j--)
     {
                #pragma omp critical
                y += arr[j] + omp_get_num_threads() ;
     }

     printf("%d,%d",x,y) ;
     printf("\n");
    return 0;
}
```

A. Deadlock
B. Data-race
C. Run-time error
D. Valid output

32. Which of the following MPI collective communication operation can also be used in non-blocking fashion:
A. MPI_Allgather
B. MPI_Alltoall
C. MPI_Bcast
D. MPI_Reduce
E. None of the above

33. Consider the following code, how many time the message "first print." Will be displayed on the screen:

```
int main (int argc, char *argv[])  {
      int i;
      omp_set_num_threads(4);
      #pragma omp parallel
      for (i=0; i < 10000; i++)
      {
            #pragma omp critical
            printf("first print.\n");
      }
      return 0;
}
```

A. 10000 times
B. 20000 times
C. 30000 times
D. 40000 times
E. Syntax error

34. Consider the following code, how many time the message "Print once." Will be displayed on the screen:

```
int main (int argc, char *argv[]) {
      int i;
      omp_set_num_threads(10);
      #pragma omp parallel
      {
          for(i=0; i < 100; i++)
          {
              #pragma omp master
              printf("Print once.\n");
          }
      }
      return 0;
}
```

A. 10 times
B. 100 times
C. 1000 times
D. 10000 times
E. Syntax error

35. Consider the following code, how many time the message "third print." Will be displayed on the screen:

```
int main (int argc, char *argv[])  {
    int i;
    omp_set_num_threads(10);
    #pragma omp parallel for
    for (i=0; i < 100; i++)
    {
        #pragma omp critical
        printf("third print.\n");
    }
    return 0;
}
```

A. 10 times
B. 100 times
C. 1000 times
D. 10000 times
E. Syntax error

36. Which of the following statements about the omp_get_num_threads() function in OpenMP is true?
A. It returns the number of threads currently executing in the parallel region.
B. It returns the maximum number of threads that can be used in the parallel region.
C. It returns the ID of the current thread within the parallel region.
D. It returns the number of threads in the parallel region at the time of its invocation.

37. Consider the following code, how many time the message "third print." Will be displayed on the screen:

```
int main() {
    int sum = 0,i;
    omp_set_num_threads(3);

    #pragma omp parallel for schedule(dynamic, 2) reduction(+:sum)
    for (i = 0; i < 10; ++i) {
        printf("Thread %d is processing iteration %d\n",
omp_get_thread_num(), i);
        sum += i;
    }

    printf("Sum: %d\n", sum);
    return 0;
}
```

A. The order of iteration processing is deterministic, but the sum may vary depending on scheduling.
B. The sum is always 45, regardless of the scheduling.
C. The sum is always 45, and the order of iteration processing is deterministic.
D. The sum is always 45, and first round of scheduling is deterministic.
E. Both the sum and the order of iteration processing are non-deterministic.

38. What is the purpose of the ordered directive in OpenMP?
A. It specifies that loop iterations should be executed in the order defined by the program.
B. It ensures that all threads execute a block of code in the order specified by their thread IDs.
C. It specifies that a block of code should be executed in the order in which threads arrive at a barrier.
D. It allows loop iterations to be executed in any order, improving performance.

39. **Which of the following MPI operation can also be used in non-blocking fashion:?**
A. MPI_Allgather
B. MPI_Alltoall
C. MPI_MPI_Probe
D. MPI_Reduce
E. None of the above

40. In the context of implementing MPI_Allreduce using point-to-point communication, what is the primary advantage of the recursive doubling algorithm over other methods?
a) Minimal memory consumption
b) Lower latency
c) Better scalability
d) Simplicity in implementation

41. In MPI collective communication, what is the role of the root process in the MPI_Gather operation?
a) It sends data to all other processes.
b) It receives data from all other processes.
c) It collects data from all other processes.
d) It performs reduction operation on data from all other processes.

42. We have an array of size 15 with values [13,4,5,1,2,3,44,15,16,7,8,9,78,65,36] What will be the values assigned to each to the process having rank 2 if displs array [0,5,9,12] and sendcounts array [5,3,2,4] is passed to MPI_ScatterV function.

```
int MPI_Scatterv(const void *sendbuf, const int *sendcounts, const int
*displs,MPI_Datatype sendtype, void *recvbuf, int recvcount,
MPI_Datatype recvtype,int root, MPI_Comm comm)
```

A. [7,8]
B. [13,4,5,1,2,3,44,15,16]
C. [16,7,8]
D. [0,5]

43. In MPI collective communication, which routine is used to scatter data from the root process to all other processes?
a) MPI_Bcast
b) MPI_Scatter
c) MPI_Gather
d) MPI_Reduce

44. For MPI_Alltoallv, which of the following is NOT true

a. messages to different processes can only have fixed length
b. messages to different processes can have different length
c. both fixed and different lengths can be set
d. each process performs a scatter followed by gather process

45. When MPI_STATUS_IGNORE is used

a. if we don't need any additional information for communication
b. if you want to halt the communication
c. if only 1 byte of data to be send
d. all of the above

$$3 = \frac{1}{f_s + \left(\frac{1-f_s}{4}\right)}$$

$$3 = \frac{1}{\left(\frac{3f_s+1}{4}\right)}$$

$$3 = \frac{1}{\frac{4f_s + (1-f_s)}{4}}$$

$$3\left(\frac{3f_s+1}{4}\right) = 1$$

End of paper
Best of luck ☺

$$\frac{9f_s + 3}{4} = 1$$

$$9f_s + 3 = 4$$

$$9f_s = 1.$$

$$f_s = \frac{1}{9} =$$