Problem 1 (12 Points)

Circle whether the following statements are true or false. Each question is worth 2 points. No explanations are necessary.

1. It is possible that the isoefficiency function cannot be derived for a parallel algorithm.

   a. True b. False

2. Parallel prefix sum algorithm requires more basic computations (floating point addition) in total than sequential prefix sum algorithm. The reason it can be faster is because parallel processors are usually faster than sequential processors.

   a. True b. False

3. Non-blocking communication can help reduce execution time by overlapping communication with computation.

   a. True b. False

4. MPI one-sided communication decouples data movement from process synchronization.

   a. True b. False

5. MPI uses a distributed memory model. Thus, it cannot be run on a processor with multiple cores that share memory.

   a. True b. False

6. In a Charm++ program with P objects and P processors, the runtime system will be able to automatically balance load across processors.

   a. True b. False

Problem 2 (12 Points)

For the following questions, circle the ONE answer that applies. Each question is worth 3 points. No explanations are necessary.

1. A program (or the corresponding parallel algorithm) is said to be scalable, in the sense of isoefficiency, if:
    a. Parallel efficiency on a given number of processors is greater than 50%.
    b. As you keep on using more processors for solving the same problem instance, the execution time decreases.
    c. As you increase the number of processors, it is possible to increase the problem size such that you can regain the efficiency.
    d. The serial fraction decreases with increased number of processors.

2. Consider parallel matrix-matrix multiplication operation on square matrices A, B and C of size NxN each (C = A*B), with the total number of processors P. Which of the following statement is true?
    a. The sequential computational complexity is $O(N^2)$.
    b. The 3D algorithm needs $P = N^3$ processors to work.
    c. In Cannon's algorithm, local submatrix size of each processor is (N/P) x (N/P).
    d. The Fox algorithm (which uses row/column broadcasts) uses more memory than Cannon's algorithm, which involves steps that shift each matrix tile to its neighbors.

3. An MPI OpenMP hybrid program is run on a small cluster of 4 nodes. Each node has 8 cores. Which of the following MPI OpenMP distributions can NOT utilize the entire cluster.
    a. 32 MPI processes with 1 OpenMP thread each.
    b. 16 MPI processes with 2 OpenMP threads each.
    c. 2 MPI processes with 16 OpenMP threads each.
    d. 8 MPI processes with 4 OpenMP threads each.

4. ~~A benefit of AMPI over MPI is:~~
    ~~a. AMPI automatically parallelizes your application.~~
    ~~b. AMPI automatically overlaps communication and computation.~~
    ~~c. AMPI automatically selects the right grain size for communication.~~
    d. ~~AMPI allows processes to share global variables.~~

Problem 4 (4 Points)

Define parallel speedup and parallel efficiency in terms of sequential execution time Ts, number of processors P, and parallel execution time Tp. (Write down the expression only.)

Parallel speedup =

Parallel efficiency =

Problem 5 (4 Points)

For each of the following scenarios, give one MPI call that can be used to perform the necessary operation. The operations are run using P processes and each operation is independent of each other. Only mention the MPI function name, do not give function arguments or explanations. Each part is worth 2 points.

a. Process with rank 0 has a large array of size N which it needs to distribute evenly to all the processes such that each process holds a unique array of size N/P (assume P divides N).

b. Each process has an array of n elements. Process with rank 0 needs to collect these arrays in order to construct a large array such that the total number of elements is P*n. Also, the array from process i must be placed before the array from process j if i < j.

Problem 6 (8 Points)

This problem is about your ability to read and understand MPI program. Consider the following MPI code snippet. Assume all variables are suitably initialized. An in-order tree has been constructed out of all the P processes in the communicator such that integer variables `parent`, `leftchild` and `rightchild` hold the ranks of parent, left child and right child in the tree representation. The value is -1 if no such process exists. [This is similar to the tree construction in MP3.]

```
leftval = rightval = 0;

if(leftchild != -1)
   MPI_Recv(&leftval, 1, MPI_DOUBLE, leftchild, 0,
                    MPI_COMM_WORLD, MPI_STATUS_IGNORE);
if(rightchild != -1)
   MPI_Recv(&rightval, 1, MPI_DOUBLE, rightchild, 0,
                    MPI_COMM_WORLD, MPI_STATUS_IGNORE);

val = leftval + rightval + myval;

if(parent != -1) {
   MPI_Send(&val, 1, MPI_DOUBLE, parent, 0, MPI_COMM_WORLD);
   MPI_Recv(&val, 1, MPI_DOUBLE, parent, 1, MPI_COMM_WORLD,
                    MPI_STATUS_IGNORE);
}

if(leftchild != -1)
   MPI_Send(&val, 1, MPI_DOUBLE, leftchild, 1,
                   MPI_COMM_WORLD);

if(rightchild != -1)
   MPI_Send(&val, 1, MPI_DOUBLE, rightchild, 1,
                   MPI_COMM_WORLD);
```

a. Each process had a different `myval` before the code snippet. Describe what the final value of the variable `val` will be for each process when the code snippet is run.

b. Rewrite the above code snippet with one MPI collective call such that the variable `val` holds the same value as in the code snippet. Supply not only the function name but also the complete argument list.

Problem 7 (9 Points)

Prefix sum: for a given array of 6 elements, suppose you have 6 processors and each holds a single element. Manually perform recursive doubling algorithm and write down the values being held by each processor in each step.

Rank	**P0**	**P1**	**P2**	**P3**	**P4**	**P5**
Initial value	-2	3	1	9	-2	0
After 1st step						
After 2nd step						
After 3rd step						

Problem 8 (10 Points)

In this problem you are asked to design a parallel algorithm for dot-product computation. Given two vector arrays **A** and **B**, both have N scalar elements. Assume you have a distributed memory system of p processors, and n >> p. For simplicity, assume n is divisible by p, and each processor initially gets a consecutive chunk of size n/p from both vectors **A** and **B** and the location is aligned properly for the computation.

$$\mathbf{A} \cdot \mathbf{B} = \sum_{i=1}^{n} A_i B_i = A_1 B_1 + A_2 B_2 + \cdots + A_n B_n$$

Image source: Wikipedia. A_i and B_i are scalar elements of the vector arrays

a. Briefly sketch a parallel algorithm that computes dot product x = **A**·**B**. (x is a scalar) At the end of the computation, all processors hold the final result of the dot product in a scalar variable x. You don't have to write the actual program but only have to describe conceptually the computation and communication that have to happen in different steps.

Steps:

b. Analyze the computation cost and communication cost of your algorithm. Derive the isoefficiency function.

Problem 9 (8 Points)

The following code snippets are used in an MPI OpenMP hybrid program with MPI_THREAD_MULTIPLE. For each of these snippets, mention if the code is correct or not. Give a brief explanation. Assume the variables are initialized properly.

a.

```
#pragma omp parallel
{
    tid = omp_get_thread_num();
    if (tid  == 0)
       MPI_Bcast(&N, 1, MPI_INT, 0, comm);
    else  if (tid == 1)
       MPI_Allreduce(&n, &s, 1, MPI_DOUBLE, MPI_SUM, comm);
}
```

Explanation:

b. Assume `nb` is a neighbor rank that has been correctly initialized.

```
#pragma omp parallel
{
    tid = omp_get_thread_num();
    if (tid  == 0)
        MPI_Send(&n, 1, MPI_INT, nb, 0, comm);
    else  if (tid == 1)
        MPI_Recv(&m, 1, MPI_INT, nb, 0, comm, &status);
}
```

Explanation:

Problem 10 (10 Points)

Consider a program that takes 20 hours to execute with no checkpoint mechanism. If a fault occurs, the program is restarted from the beginning, and this process is repeated till the complete program runs without any fault.

a. (2 point) The program starts running at T = 0 and the system fails once at T = 14 hrs. What is the total time required to run the program?

b. (3 points) Now assume a checkpoint mechanism has been introduced to the program which creates a checkpoint after 2 hours of program execution. Each dump takes 15 minutes and a restart, if required, takes 30 minutes. How long does the program take now, if there are no faults.

c. (5 points) Now, assume the same checkpoint mechanism as in part b and give the time taken to run the program if the program begins at T = 0, and system fails at T = 14 hrs (as in part a).

Problem 11 (12 Points)

Consider matrix vector multiplication with an N x N matrix A and an N x 1 vector X. To perform this operation in parallel, P^2 processes arranged in a P x P grid are used. The matrix A is divided into N/P x N/P blocks and assigned to the processes. The vector X is divided into P chunks of N/P elements and the chunks are assigned to the diagonal processes such that process (i, i) has the i^{th} portion. The result vector B = A X is also divided and allotted similar to X, that is, at the end of the computation, the diagonal processes must hold corresponding portions of final B. The following illustration explains this partition for P = 3.

Matrix A:

Process (0, 0)	Process (0, 1)	Process (0, 2)
Process (1, 0)	Process (1, 1)	Process (1, 2)
Process (2, 0)	Process (2, 1)	Process (2, 2)

Vectors X and B

Process (0, 0)
Process (1, 1)
Process (2, 2)

Write an MPI program which performs this multiplication. Assume the function initialize initializes the arrays A, B, X and the variable n provided the row and column numbers. The variables X and B will be null if the process does not hold that array. The setup and initialization has been done for you. Complete the program by filling the gaps. Assume n = N/P.

```
int main(int argc, char **argv) {
  int n, P, rank, row, col;
  double **A, *X, *B;
  MPI_Comm comm2D, comm_row, comm_col;
  int dimsize[2], periodic[2], mycoords[2];

  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD, &P);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  P = sqrt(P);
  dimsize[0]=dimsize[1] = P;
  periodic[0] = periodic[1] = 0;
  MPI_Cart_create(MPI_COMM_WORLD, 2, dimsize, periodic, 0,&comm2D);
  MPI_Cart_coords(comm2D, myrank, 2, mycoords);
```

```
  row = mycoords[0]; col = mycoords[1];
  MPI_Comm_split(comm2D, row, col, &comm_row);
  MPI_Comm_split(comm2D, col, row, &comm_col);

  // Initializes the arrays and n
  Initialize(&A, &X, &B, &n, r, c);

  // Insert declarations here









  // Perform matrix vector multiplication

  MPI_Finalize();
}
```