Question 1 (1 point)

Implement a function with the following prototype by means of MPI point-to-point operations:

```
void comunicate(double x[], int sizes[], double xloc[], int nloc, int root)
```

It must distribute vector `x`, which is stored in the process with index `root`, among all processes of the MPI program, in such a way that process 0 obtains the first block of elements, of size `sizes[0]`, process 1 gets the next block, of size `sizes[1]`, etc. The length of vector `sizes` is equal to the number of processes.

The arguments `x` and `sizes` are only valid in process `root`. The value of `nloc`, which can be different in each process, indicates the size of the block that corresponds to the process itself. Each process, including the `root` process, will store in array `xloc` the block assigned to it.

For instance, if `root` is 0 and the rest of arguments are:

	P_0	P_1	P_2
`x`	1 4 5 8 9 3	-	-
`sizes`	2 1 3	-	-
`nloc`	2	1	3

the final content of `xloc` in each process must be:

	P_0	P_1	P_2
`xloc`	1 4	5	8 9 3

Solution:

```
void comunicate(double x[], int sizes[], double xloc[], int nloc, int root) {
  int p, rank, i, iproc, k;
  MPI_Comm_size(MPI_COMM_WORLD,&p);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  if (rank==root) {
    k = 0;
    for (iproc=0; iproc<p; iproc++) {
      if (iproc==rank) {
        /* Copy block of x to xloc */
        for (i=0; i<nloc; i++) xloc[i] = x[k+i];
      }
      else
        MPI_Send(&x[k],sizes[iproc],MPI_DOUBLE,iproc,0,MPI_COMM_WORLD);
      k += sizes[iproc];
    }
  }
  else {
    MPI_Recv(xloc,nloc,MPI_DOUBLE,root,0,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
  }
}
```

Question 2 (1.2 points)

The following function multiplies element by element the values of a matrix A by another matrix B by a real number a, in order to obtain the resulting matrix C:

```
void product_elements(double a,double A[M][N],double B[M][N],double C[M][N]) {
   int i,j;
   for (i=0;i<M;i++) {
      for (j=0;j<N;j++) {
         C[i][j]=a*A[i][j]*B[i][j];
      }
   }
}
```

0.9 p. (a) Parallelize the function by means of MPI collective operations, distributing the matrices by blocks of consecutive rows. We assume that:

- Process 0 initially stores the value of a and matrices A and B.
- At the end of the function, all processes must store the full C matrix.
- The number of rows M of the matrices will always be a multiple of the number of used processes.

Solution:

```
void product_elements(double a,double A[M][N],double B[M][N],double C[M][N]) {
   int i,j,k,np;
   double Alocal[M][N],Blocal[M][N],Clocal[M][N];
   MPI_Comm_size(MPI_COMM_WORLD,&np);
   k=M/np;
   MPI_Bcast(&a,1,MPI_DOUBLE,0,MPI_COMM_WORLD);
   MPI_Scatter(A,k*N,MPI_DOUBLE,Alocal,k*N,MPI_DOUBLE,0,MPI_COMM_WORLD);
   MPI_Scatter(B,k*N,MPI_DOUBLE,Blocal,k*N,MPI_DOUBLE,0,MPI_COMM_WORLD);
   for (i=0;i<k;i++) {
      for (j=0;j<N;j++) {
         Clocal[i][j]=a*Alocal[i][j]*Blocal[i][j];
      }
   }
   MPI_Allgather(Clocal,k*N,MPI_DOUBLE,C,k*N,MPI_DOUBLE,MPI_COMM_WORLD);
}
```

0.3 p. (b) Calculate the arithmetic cost and the communications cost of the implemented function. Clearly indicate the total cost corresponding to each of the collective operations.

Solution: First, the arithmetic cost is:

$$t_a(M,p) = \sum_{i=0}^{M/p-1} \sum_{j=0}^{N-1} 2 = \sum_{i=0}^{M/p-1} 2N = \frac{2MN}{p} flops$$

Second, the communication cost is:

$$t_c(M,p) = t_{Bcast} + t_{ScatterA} + t_{ScatterB} + t_{Allgather}$$

In detail:

$$\begin{aligned} t_{Bcast} &= (p-1)(t_s + t_w) \\ t_{ScatterA} &= (p-1)(t_s + \tfrac{MN}{p} t_w) \\ t_{ScatterB} &= (p-1)(t_s + \tfrac{MN}{p} t_w) \\ t_{Allgather} &= (p-1)(t_s + \tfrac{MN}{p} t_w) + (p-1)(t_s + MN t_w) \end{aligned}$$

Question 3 (1.3 points)

The following program computes the norm of a square matrix of size $3N \times 3N$:

```
double norma(double A[N*3][N*3]) {
    int i, j;
    double norm=0.0;

    for (i=0;i<N*3;i++)
      for (j=0;j<N*3;j++)
         norm += A[i][j]*A[i][j];

    norm=sqrt(norm);
    return norm;
}
```

1 p. (a) Implement an MPI parallel version for 9 processes. The function must distribute the matrix among the 9 processes using a bi-dimensional arrangement as the one shown in the example. Each process will receive a square submatrix of `A` of size $N \times N$ that will be stored in another matrix also of size $N \times N$. The number of messages should be minimized and intermediate copies avoided. NOTE: The example is for a 6×6 matrix but the program must work for any matrix of size $3N \times 3N$. We assume that the number of processes will be 9. The final result (`norm`) must be correct in process 0.

$$A = \left(\begin{array}{cc|cc|cc} a_{00} & a_{01} & a_{02} & a_{03} & a_{04} & a_{05} \\ a_{10} & a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ \hline a_{20} & a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{30} & a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ \hline a_{40} & a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{50} & a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} P_0 & P_1 & P_2 \\ \hline P_3 & P_4 & P_5 \\ \hline P_6 & P_7 & P_8 \end{array}\right)$$

Solution:

```
double norma(double A[N*3][N*3]) {
    int rank;
    int i, j, proc;
    double norm=0.0,lnorm, X[N][N];
    MPI_Datatype type;

    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    MPI_Type_vector(N, N, 3*N, MPI_DOUBLE, &type);
    MPI_Type_commit(&type);

    if (rank==0) {
       proc=1;
       for (i=0;i<3;i++)
         for (j=0;j<3;j++)
           if (i==0 && j==0)
              MPI_Sendrecv(&A[0][0], 1, type, 0, 100, X, N*N,
                    MPI_DOUBLE, 0, 100, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
           else {
              MPI_Send(&A[i*N][j*N], 1, type, proc, 100, MPI_COMM_WORLD);
              proc++;
              }
```

```
    } else
       MPI_Recv(X, N*N, MPI_DOUBLE, 0, 100, MPI_COMM_WORLD, MPI_STATUS_IGNORE);

    lnorm=0.0;
    for (i=0;i<N;i++)
      for (j=0;j<N;j++)
         lnorm += X[i][j]*X[i][j];

    MPI_Reduce(&lnorm, &norm, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
    norm=sqrt(norm);
    MPI_Type_free(&type);
    return norm;
}
```

0.3 p. (b) Obtain the sequential time and the parallel time of the implemented version, assuming that the square root has a cost of 5 flops.

Solution:

$$t(N) = \sum_{i=0}^{3N}\sum_{i=0}^{3N} 2 + 5 \approx 18N^2 flops$$

$$t(N,p) = t_a(N,p) + t_c(N,p)$$

$$t_a(N,p) = \sum_{i=0}^{N}\sum_{i=0}^{N} 2 + 5 \approx 2N^2 flops$$

$$t_c(N,p) = 8(t_s + N^2 t_w) + (p-1)(t_s + t_w) + (p-1),$$

where the cost $8(t_s + t_w) + 8$ is from reduction operation (8 messages of 1 element and 8 flops for the additions). In resume:

$$t(N,p) = 2N^2 + 8 + 16t_s + (N^2 + 8)t_w \approx 2N^2 + 16t_s + N^2 t_w$$