Cuestión 1 (1 punto)

Implementa, mediante operaciones punto a punto, una función con la siguiente cabecera:

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

la cual debe repartir el vector `x`, que se encuentra en el proceso de índice `root`, entre todos los procesos del programa MPI, de manera que el proceso 0 obtenga el primer bloque de elementos, de tamaño `sizes[0]`, el proceso 1 obtenga el siguiente bloque, de tamaño `sizes[1]`, etc. La longitud del vector `sizes` es igual al número de procesos.

Los argumentos `x` y `sizes` son solo válidos en el proceso `root`. El valor de `nloc`, que puede ser distinto en cada proceso, indica el tamaño del bloque que le corresponde al propio proceso. Cada proceso, incluido el proceso `root`, almacenará en el array `xloc` el bloque que le corresponde.

Por ejemplo, si root vale 0 y el resto de argumentos son:

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

el contenido final de `xloc` en cada proceso debe ser:

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

Solución:

```
void comunica(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) {
        /* Copia del bloque de x a 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);
  }
}
```

Cuestión 2 (1.2 puntos)

La siguiente función multiplica elemento a elemento los valores de una matriz A por una matriz B por un número real a, a fin de obtener otra matriz C resultante:

```
void producto_elementos(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) Paralelizar dicha función mediante operaciones colectivas de MPI, repartiendo las matrices por bloques de filas consecutivas. Se entenderá que:

- El proceso 0 será el que disponga inicialmente del valor de a y de las matrices A y B.
- Al finalizar la función, todos los procesos deberán disponer de la matriz C completa.
- El número de filas M de las matrices siempre será múltiplo del número de procesos empleados.

Solución:

```
void producto_elementos(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) Calcula el coste aritmético y el coste de las comunicaciones de la función implementada en el apartado anterior. Indica claramente el coste total correspondiente a cada una de las operaciones colectivas.

Solución: En primer lugar, el coste aritmético sería:

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

En segundo lugar, el coste de las comunicaciones se correspondería con:

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

En detalle:

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

Cuestión 3 (1.3 puntos)

El siguiente programa realiza el cálculo de la norma de una matriz cuadrada de tamaño $3N \times 3N$:

```
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) Implementa una versión paralela en MPI para 9 procesos. La función debe distribuir la matriz entre los 9 procesos usando una estructura bidimensional como la que se muestra en el ejemplo. Cada proceso recibirá una submatriz cuadrada de A de tamaño $N \times N$ que almacenará en otra matriz también de tamaño $N \times N$. Se debe minimizar el número de mensajes y evitar copias intermedias. NOTA: El ejemplo es para una matriz 6×6 pero el programa deberá funcionar para una matriz cualquiera de tamaño $3N \times 3N$. Se asume que el número de procesos será 9. El resultado final (`norm`) debe ser correcto en el proceso 0.

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

Solución:

```
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) Obtener el tiempo secuencial y el tiempo paralelo de la versión implementada, asumiendo que la raíz cuadrada tiene un coste de 5 flops.

Solución:

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

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

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

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

donde el coste de $8(t_s + t_w) + 8$ corresponde a la operación de reducción (8 mensajes de 1 elemento y 8 flops para las sumas). Por tanto:

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