Cuestión 1 (1.3 puntos)
Dada la siguiente función:

```
void fun1(double A[N][N], double x[], double y[]) {
    int i,j;

    for (i=0;i<N;i++) {
       for (j=0;j<N;j++)
          A[i][j]= x[i]*y[j];
    }

}
```

1 p. (a) Implementa una versión paralela mediante MPI, asumiendo que los datos de entrada se encuentran en el proceso 0 y que los resultados deben de encontrarse completos en dicho proceso al final de la ejecución. Se puede asumir que el tamaño del problema es un múltiplo del número de procesos.

Solución:

```
void fun1_par(double A[N][N], double x[N], double y[N]) {
    int p, np;
    int i,j;
    double xlcl[N];
    double Alcl[N][N];

    MPI_Comm_size(MPI_COMM_WORLD, &p);

    np = N/p;
    MPI_Scatter(x, np, MPI_DOUBLE, xlcl, np, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    MPI_Bcast(y, N, MPI_DOUBLE, 0, MPI_COMM_WORLD);

    for (i=0;i<np;i++)
       for (j=0;j<N;j++)
          Alcl[i][j] = xlcl[i]*y[j];
    MPI_Gather(Alcl, np*N, MPI_DOUBLE, A, np*N, MPI_DOUBLE, 0, MPI_COMM_WORLD);

}
```

0.3 p. (b) Obtener la expresión del tiempo de ejecución paralelo, indicando el coste de comunicación de cada operación colectiva utilizada.

Solución:

$$t(N,p) = t_{comm}(N,p) + t_a(N,p)$$
$$t_{comm}(N,p) = t_{Scatter} + t_{Bcast} + t_{Gather}$$
$$t_{Scatter} = (p-1)(t_s + \frac{N}{p})t_w$$
$$t_{Bcast} = (p-1)(t_s + Nt_w)$$
$$t_{Gather} = (p-1)(t_s + N\frac{N}{p}t_w)$$
$$t_a(N,p) = \sum_{i=0}^{\frac{N}{p}-1}\sum_{j=0}^{N} 1$$

$$t(N,p) = (p-1)(t_s + \frac{N}{p})t_w + (p-1)(t_s + Nt_w) + \sum_{i=0}^{\frac{N}{p}-1}\sum_{j=0}^{N} 1 + (p-1)(t_s + N\frac{N}{p}t_w)$$

$$t(N,p) \approx 3pt_s + (pN + N^2)t_w + \frac{N^2}{p}$$

Cuestión 2 (1.1 puntos)

Pretendemos enviar las primeras y últimas filas y columas de una matriz rectangular de tamaño $M \times N$ desde el proceso identificado como *root* al resto de procesos. A continuación se muestra un ejemplo para una matriz de dimensiones $M = 4$ y $N = 5$, donde los términos identificados con el símbolo x se corresponden con todos aquellos a enviar:

$$A = \begin{pmatrix} x & x & x & x & x \\ x & \cdot & \cdot & \cdot & x \\ x & \cdot & \cdot & \cdot & x \\ x & x & x & x & x \end{pmatrix}$$

0.9 p. (a) Completar el cuerpo de la función cuya cabecera se incluye a continuación para llevar a cabo el envío. Los parámetros de la función se corresponden con el identificador del proceso que la invoca (*myid*), el número total de procesos (*np*) y el identificador del proceso raíz que dispone inicialmente de la matriz A de partida y que lleva a cabo el envío (*root*).

```
void envia_perimetro_matriz(double A[M][N], int myid, int np, int root);
```

Todos los procesos con un identificador diferente a *root* deberán almacenar los datos recibidos en la misma matriz A que se proporciona como parámetro a la función. Para ello, se deberán emplear operaciones de comunicación punto a punto y tipos de datos derivados, de manera que se minimice el número de envíos a realizar por parte del proceso *root* al resto. Se valorará que ningún elemento sea enviado más de una vez a cada proceso (en especial, los elementos de las 4 esquinas de la matriz).

Solución:

```
// ALTERNATIVA 1
void envia_perimetro_matriz(double A[M][N], int myid, int np, int root) {
  int p;
  MPI_Datatype columna;
  MPI_Datatype filas_primera_ultima;
  MPI_Type_vector(M,1,N,MPI_DOUBLE,&columna);
  MPI_Type_vector(2,N-2,(M-1)*N,MPI_DOUBLE,&filas_primera_ultima);
  MPI_Type_commit(&filas_primera_ultima);
  MPI_Type_commit(&columna);
  if (myid==root) {
    for (p=0;p<np;p++) {
      if (p!=root) {
        MPI_Send(A,1,columna,p,0,MPI_COMM_WORLD);
        MPI_Send(&A[0][N-1],1,columna,p,0,MPI_COMM_WORLD);
        MPI_Send(&A[0][1],1,filas_primera_ultima,p,0,MPI_COMM_WORLD);
      }
    }
  }
  else {
    MPI_Recv(A,1,columna,root,0,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
    MPI_Recv(&A[0][N-1],1,columna,root,0,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
    MPI_Recv(&A[0][1],1,filas_primera_ultima,root,0,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
  }
  MPI_Type_free(&filas_primera_ultima);
  MPI_Type_free(&columna);
}

  // ALTERNATIVA 2
void envia_perimetro_matriz(double A[M][N], int myid, int np, int root) {
  int p;
```

```
  MPI_Datatype filas_consecutivas;
  MPI_Datatype filas_primera_ultima;
  MPI_Type_vector(M-1,2,N,MPI_DOUBLE,&filas_consecutivas);
  MPI_Type_vector(2,N-1,(M-1)*N+1,MPI_DOUBLE,&filas_primera_ultima);
  MPI_Type_commit(&filas_consecutivas);
  MPI_Type_commit(&filas_primera_ultima);
  if (myid==root) {
    for (p=0;p<np;p++) {
      if (p!=root) {
        MPI_Send(&A[0][N-1],1,filas_consecutivas,p,0,MPI_COMM_WORLD);
        MPI_Send(A,1,filas_primera_ultima,p,0,MPI_COMM_WORLD);
      }
    }
  }
  else {
    MPI_Recv(&A[0][N-1],1,filas_consecutivas,root,0,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
    MPI_Recv(A,1,filas_primera_ultima,root,0,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
  }
  MPI_Type_free(&filas_consecutivas);
  MPI_Type_free(&filas_primera_ultima);
}
```

0.2 p. (b) Obtener el coste de las comunicaciones.

Solución:

Alternativa 1:

$$t_c = (p-1)\left(2\left(t_s + Mt_w\right) + t_s + 2\left(N-2\right)t_w\right)$$

Alternativa 2:

$$t_c = (p-1)\left(t_s + 2\left(M-1\right)t_w + t_s + 2\left(N-1\right)t_w\right)$$

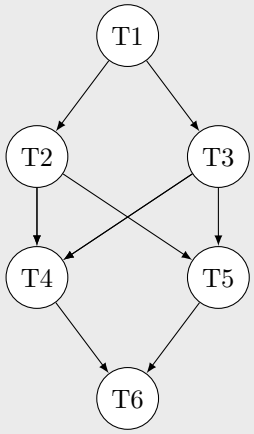
Cuestión 3 (1.1 puntos)

Dada la siguiente función, donde las funciones correspondientes a las tareas (`T1` a `T6`) modifican solo su último argumento y donde el coste de cada una de dichas funciones es $4N^2$ flops, excepto las funciones `T2` y `T4`, cuyo coste es de $3N^2$ flops cada una.

```
void func(double A[N][N], double w[N]) {
  double x[N],y[N],v[N],alfa;
  T1(A,x);
  T2(A,x,y);
  T3(x,v);
  T4(y,v,w);
  T5(A,y,v,&alfa);
  T6(alfa,w);
}
```

0.3 p. (a) Dibuja el grafo de dependencias de datos entre las tareas.

Solución:



0.6 p.

(b) Implementa una versión paralela con MPI para 2 procesos, utilizando operaciones de comunicación punto a punto. Se supondrá que la matriz `A` se encuentra inicialmente en el proceso `0`. Respecto al vector `w`, su contenido inicial no se utiliza y su contenido final correcto puede quedar en uno cualquiera de los procesos. Justifica la asignación de tareas utilizada.

Solución: Se utilizará la asignación:
$P_0 : T_1, T_2, T_5$
$P_1 : T_3, T_4, T_6$
Dicha asignación maximiza el paralelismo, ya que las tareas independientes están en procesos distintos. Además, se minimizan comunicaciones evitando comunicar la matriz `A`.

```
void func_par(double A[N][N], double w[N]) {
  double x[N],y[N],v[N],alfa;
  int rank;
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  if (rank==0) {
    T1(A,x);
    MPI_Send(x,N,MPI_DOUBLE,1,0,MPI_COMM_WORLD);
    T2(A,x,y);
    MPI_Sendrecv(y,N,MPI_DOUBLE,1,0,v,N,MPI_DOUBLE,1,0,
        MPI_COMM_WORLD,MPI_STATUS_IGNORE);
    T5(A,y,v,&alfa);
    MPI_Send(&alfa,1,MPI_DOUBLE,1,0,MPI_COMM_WORLD);
  } else if (rank==1) {
    MPI_Recv(x,N,MPI_DOUBLE,0,0,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
    T3(x,v);
    MPI_Sendrecv(v,N,MPI_DOUBLE,0,0,y,N,MPI_DOUBLE,0,0,
        MPI_COMM_WORLD,MPI_STATUS_IGNORE);
    T4(y,v,w);
    MPI_Recv(&alfa,1,MPI_DOUBLE,0,0,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
    T6(alfa,w);
  }
}
```

0.2 p.

(c) Calcula el coste secuencial y el coste paralelo.

Solución: Coste secuencial:

$$t(N) = 4 \cdot 4N^2 + 2 \cdot 3N^2 = 22N^2 \text{ flops}$$

Coste paralelo:

$$\begin{aligned} t_a(N,2) &= 4N^2 + 4N^2 + 4N^2 + 4N^2 = 16N^2 \text{ flops} \\ t_c(N,2) &= 3(t_s + Nt_w) + (t_s + t_w) = 4t_s + (3N+1)t_w \approx 4t_s + 3Nt_w \\ t(N,2) &= 16N^2 \text{ flops} + 4t_s + 3Nt_w \end{aligned}$$