Cuestión 1 (1.3 puntos)

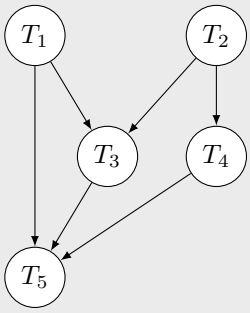
Dada la siguiente función, donde `NR` y `NC` son constantes enteras:

```
void calcular(double x[NR], double y[NR]) {
  double A[NR][NC], B[NR][NC], v1[NR];
  double s,t;
  task1(A,y,x);
  s=task2(B);
  t=task3(y,B,x);
  task4(v1,s);
  task5(x,A,v1,t);
}
```

y teniendo en cuenta que la función `task1` modifica sus dos primeros argumentos, mientras que el resto de funciones (`task2` a `task5`) modifican solo su primer argumento.

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

Solución: El grafo de dependencias es el siguiente:



1 p. (b) Escribe una versión paralela MPI de la función anterior para dos procesos, suponiendo que los vectores `x` e `y` están inicialmente disponibles en todos los procesos. El contenido final correcto del vector `x` debe quedar en el proceso 0 y el del vector `y` en el proceso 1.

La versión paralela deberá:

- Usar una asignación que maximice el paralelismo y minimice el coste de comunicaciones.
- Evitar posibles interbloqueos.

Solución:

El resultado final de `x` se calcula en la tarea T_5 y el de `y` en T_3. Dado que `x` debe quedar en P_0 e `y` en P_1, intentaremos que P_0 haga T_5 y P_1 haga T_3. Además, intentaremos que las tareas T_1 y T_5 las haga el mismo proceso, para no tener que comunicar la matriz `A`. Lo mismo ocurre con las tareas T_2 y T_3, en este caso por la matriz `B`.

De acuerdo con esto, la asignación óptima sería $P_0 : T_1, T_4, T_5$; $P_1 : T_2, T_3$.

```
void calcular_mpi(double x[NR], double y[NR]) {
  double s, t;
```

```
    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);
    if (rank==0) {
      double A[NR][NC], v1[NR];
      task1(A,y,x);
      MPI_Send(y,NR,MPI_DOUBLE,1,33,MPI_COMM_WORLD);
      MPI_Recv(&s,1,MPI_DOUBLE,1,33,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
      task4(v1,s);
      MPI_Recv(&t,1,MPI_DOUBLE,1,33,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
      task5(x,A,v1,t);
    }
    else if (rank==1) {
      double B[NR][NC];
      s=task2(B);
      /* Para evitar interbloqueos, primero recibimos y después enviamos
         (al revés que en el proceso 0) */
      MPI_Recv(y,NR,MPI_DOUBLE,0,33,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
      MPI_Send(&s,1,MPI_DOUBLE,0,33,MPI_COMM_WORLD);
      t=task3(y,B,x);
      MPI_Send(&t,1,MPI_DOUBLE,0,33,MPI_COMM_WORLD);
    }
  }
```

Cuestión 2 (1.2 puntos)

La siguiente función calcula el producto $A \times x$ en y y luego escala y de forma que el elemento de mayor valor absoluto sea 1.

```
void vector( double y[M], double A[M][N], double x[N] )
{ int i,j;
  double a,ma;

  ma = 0;
  for ( i = 0 ; i < M ; i++ ) {
    a = 0;
    for ( j = 0 ; j < N ; j++ )
      a += A[i][j] * x[j];
    y[i] = a;
    if ( a < 0 ) a = -a;
    if ( a > ma ) ma = a;
  }
  for ( i = 0 ; i < M ; i++ )
    y[i] /= ma;
}
```

1 p. (a) Paralelízala con MPI de forma que el trabajo de los bucles quede repartido entre todos los procesos disponibles. Utiliza operaciones de comunicación colectiva allá donde sea posible. Los parámetros de entrada a la función sólo tienen valores válidos en el proceso 0. Queremos el resultado (el vector y) en el proceso 0. Asumimos que `M` y `N` son un múltiplo exacto del número de procesos.

Solución:

```
  void vector( double y[M], double A[M][N], double x[N] )
  { int i,j, np,nb;
```

```
  double a,ma, Al[M][N],yl[M],mal;

  MPI_Comm_size(MPI_COMM_WORLD,&np);

  nb = M / np;
  MPI_Scatter(A,nb*N,MPI_DOUBLE,Al,nb*N,MPI_DOUBLE,0,MPI_COMM_WORLD);
  MPI_Bcast(x,N,MPI_DOUBLE,0,MPI_COMM_WORLD);

  mal = 0;
  for ( i = 0 ; i < nb ; i++ ) {
    a = 0;
    for ( j = 0 ; j < N ; j++ )
      a += Al[i][j] * x[j];
    yl[i] = a;
    if ( a < 0 ) a = -a;
    if ( a > mal ) mal = a;
  }

  MPI_Allreduce(&mal,&ma,1,MPI_DOUBLE,MPI_MAX,MPI_COMM_WORLD);

  for ( i = 0 ; i < nb ; i++ )
    yl[i] /= ma;

  MPI_Gather(yl,nb,MPI_DOUBLE,y,nb,MPI_DOUBLE,0,MPI_COMM_WORLD);
}
```

0.2 p. (b) Indica el coste de comunicaciones de cada operación de comunicación que hayas utilizado, suponiendo una implementación sencilla de las comunicaciones.

Solución: $t_{scat} = (p-1) * (t_s + M/pNt_w)$
$t_{bcast} = (p-1) * (t_s + Nt_w)$
$t_{allred} = 2 * (p-1) * (t_s + t_w)$
$t_{gat} = (p-1) * (t_s + M/pt_w)$

Cuestión 3 (1 punto)

Se tiene un programa MPI en el que dos procesos, P_0 y P_1, deben comunicarse determinados elementos de una matriz de números reales de doble precisión, representada por un array bidimensional `A` en el proceso emisor y `B` en el receptor. El algoritmo matricial requiere que se envíen los elementos de la antidiagonal principal (excepto el primero) junto con los elementos adyacentes mostrados en la figura, marcados como a y b, respectivamente, en la matriz `A`, de forma que el proceso receptor debe almacenar estos elementos desplazados una fila hacia arriba, es decir, con los valores b ocupando la antidiagonal principal y los valores a ocupando los elementos adyacentes mostrados en la figura, como se indica en la matriz `B`.

$$\texttt{A} = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & a & b \\ \cdot & \cdot & \cdot & a & b & \cdot \\ \cdot & \cdot & a & b & \cdot & \cdot \\ \cdot & a & b & \cdot & \cdot & \cdot \\ a & b & \cdot & \cdot & \cdot & \cdot \end{bmatrix}, \qquad \texttt{B} = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & a & b \\ \cdot & \cdot & \cdot & a & b & \cdot \\ \cdot & \cdot & a & b & \cdot & \cdot \\ \cdot & a & b & \cdot & \cdot & \cdot \\ a & b & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix}.$$

0.9 p. (a) Escribe el código necesario para realizar la comunicación (envío desde P_0 y recepción en P_1) utilizando un único mensaje. Es obligatorio el uso de tipos derivados. Se deberá usar la siguiente cabecera de función:

```
void comunica (double A[N][N], double B[N][N])
```

Solución: Para realizar la comunicación en un único mensaje, es necesario definir un nuevo tipo de datos derivados de MPI. Una vez hecho esto, simplemente hay que enviar un elemento de este tipo, usando los punteros apropiados para el buffer de envío y recepción, de forma que se lean y escriban los valores en las posiciones requeridas.

```
void comunica (double A[N][N], double B[N][N]) {
int rank;
MPI_Datatype diags;
MPI_Type_vector(N-1, 2, N-1, MPI_DOUBLE, &diags);
MPI_Type_commit(&diags);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
if (rank==0) {
  MPI_Send(&A[1][N-2], 1, diags, 1, 0, MPI_COMM_WORLD);
} else if (rank==1) {
  MPI_Recv(&B[0][N-2], 1, diags, 0, 0, MPI_COMM_WORLD,
           MPI_STATUS_IGNORE);
}
MPI_Type_free(&diags);
}
```

0.1 p.

(b) Indica cuál es el coste de comunicación.

Solución: Se envía un único mensaje, cuya longitud es $2*$`(N-1)`. Por tanto, el coste será

$$t_c = t_s + 2(N-1)t_w.$$