Qüestió 1 (1.3 punts)
Donada la següent funció:

```
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 versió paral·lela mitjançant MPI, assumint que les dades d'entrada es troben en el procés 0 i que els resultats han de trobar-se complets en aquest procés al final de l'execució. Es pot assumir que la grandària del problema és un múltiple del nombre de processos.

Solució:

```
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) Obteniu l'expressió del temps d'execució paral·lel, indicant el cost de comunicació de cada operació col·lectiva utilitzada.

Solució:

$$t(N,p) = t_{comm}(N,p) + t_a(N,p)$$

$$t_{comm}(N,p) = t_{Scatter} + t_{Bcast} + t_{Gather}$$

$$t_{Scatter} = (p-1)(t_s + \frac{N}{p})t_w$$

$$t_{Bcast} = (p-1)(t_s + Nt_w)$$

$$t_{Gather} = (p-1)(t_s + N\frac{N}{p}t_w)$$

$$t_a(N,p) = \sum_{i=0}^{\frac{N}{p}-1}\sum_{j=0}^{N} 1$$

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

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

Qüestió 2 (1.1 punts)

Pretenem enviar les primeres i últimes files i columas d'una matriu rectangular de grandària $M \times N$ des del procés identificat com *root* a la resta de processos. A continuació es mostra un exemple per a una matriu de dimensions $M = 4$ i $N = 5$, on els termes identificats amb el símbol x es corresponen amb tots aquells 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) Completeu el cos de la funció la capçalera de la qual s'inclou a continuació per a dur a terme l'enviament. Els paràmetres de la funció es corresponen amb l'identificador del procés que la invoca (*myid*), el nombre total de processos (*np*) i l'identificador del procés arrel que disposa inicialment de la matriu A de partida i que duu a terme l'enviament (*root*).

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

Tots els processos amb un identificador diferent a *root* hauran d'emmagatzemar les dades rebudes en la mateixa matriu A que es proporciona com a paràmetre a la funció. Per a això, s'hauran d'emprar operacions de comunicació punt a punt i tipus de dades derivades, de manera que es minimitze el nombre d'enviaments a realitzar per part del procés *root* a la resta. Es valorarà que cap element siga enviat més d'una vegada a cada procés (especialment, els elements de les 4 cantonades de la matriu).

Solució:

```
// ALTERNATIVA 1
void envia_perimetre_matriu(double A[M][N], int myid, int np, int root) {
  int p;
  MPI_Datatype columna;
  MPI_Datatype files_primera_ultima;
  MPI_Type_vector(M,1,N,MPI_DOUBLE,&columna);
  MPI_Type_vector(2,N-2,(M-1)*N,MPI_DOUBLE,&files_primera_ultima);
  MPI_Type_commit(&files_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,files_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,files_primera_ultima,root,0,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
    }
    MPI_Type_free(&files_primera_ultima);
    MPI_Type_free(&columna);
  }

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

0.2 p. (b) Obteniu el cost de les comunicacions.

Solució:

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:

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

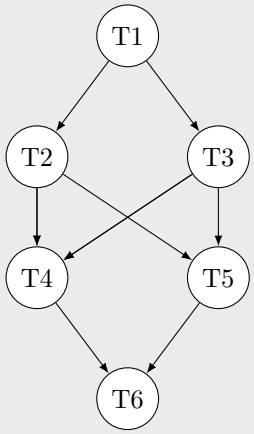
Qüestió 3 (1.1 punts)

Donada la següent funció, on les funcions corresponents a les tasques (`T1` a `T6`) modifiquen només el seu últim argument i on el cost de cadascuna d'aquestes funcions és $4N^2$ flops, excepte les funcions `T2` i `T4`, el cost de les quals és de $3N^2$ flops cadascuna.

```
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) Dibuixa el graf de dependències de dades entre les tasques.

Solució:



0.6 p. (b) Implementa una versió paral·lela amb MPI per a 2 processos, utilitzant operacions de comunicació punt a punt. Se suposarà que la matriu `A` es troba inicialment en el procés `0`. Respecte al vector `w`, el seu contingut inicial no s'utilitza i el seu contingut final correcte pot quedar en un qualsevol dels processos. Justifica l'assignació de tasques utilitzada.

Solució: S'utilitzarà l'assignació:

$P_0 : T_1, T_2, T_5$

$P_1 : T_3, T_4, T_6$

Aquesta assignació maximitza el paral·lelisme, ja que les tasques independents estan en processos diferents. A més, es minimitzen comunicacions evitant comunicar la matriu `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 cost seqüencial i el cost paral·lel.

Solució: Cost seqüencial:

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

Cost paral·lel:

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