Qüestió 1 (1.3 punts)

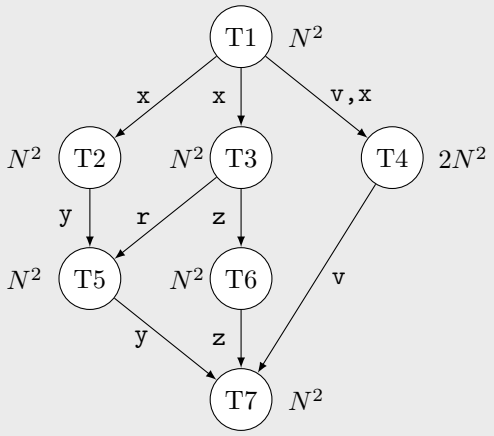
Donada la següent funció, on les funcions `T1` a `T7` modifiquen només el seu primer argument i on el cost de cadascuna d'estes funcions és N^2 flops, excepte la funció `T4`, el cost del qual és de $2N^2$ flops:

```
double func(double v[N]) {
  double x[N],y[N],z[N],r,s;
  T1(x,v);
  T2(y,x);
  r=T3(z,x);
  T4(v,x);
  T5(y,r);
  T6(z);
  s=T7(v,y,z);
  return s;
}
```

0.4 p. (a) Dibuixa el graf de dependències de dades entre les tasques.

Solució:

S'inclou en el graf, encara que no es demana, els costos de les tasques i les dades corresponents a les dependències entre tasques, el que és útil per als següents apartats.



0.7 p. (b) Implementa una versió paral·lela amb MPI, utilitzant el nombre de processos adequat per a maximitzar el paral·lelisme. S'ha de tenir en compte que el vector `v` es troba inicialment només en el procés 0, i que el valor retornat per la funció ha de ser el correcte també en el procés 0.

Solució: S'utilitzaran 3 processos, ja que eixe és el grau màxim de concurrència, i l'assignació:
$P_0 : T_1, T_4, T_7$. $P_1 : T_2, T_5$. $P_2 : T_3, T_6$.
Esta assignació maximitza el paral·lelisme, ja que les tasques independents estan en processos diferents i el cost de les tasques que es fan en paral·lel (tasques 2 a 6) està equilibrat entre els tres processos. A més, es minimitzen comunicacions evitant comunicar el vector `v`.

```
double func_par(double v[N]) {
  double x[N],y[N],z[N],r,s;
  int rank;
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  if (rank==0) {
    T1(x,v);
    MPI_Send(x,N,MPI_DOUBLE,1,0,MPI_COMM_WORLD);
    MPI_Send(x,N,MPI_DOUBLE,2,0,MPI_COMM_WORLD);
    T4(v,x);
    MPI_Recv(y,N,MPI_DOUBLE,1,0,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
```

```
      MPI_Recv(z,N,MPI_DOUBLE,2,0,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
      s=T7(v,y,z);
    }
    else if (rank==1) {
      MPI_Recv(x,N,MPI_DOUBLE,0,0,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
      T2(y,x);
      MPI_Recv(&r,1,MPI_DOUBLE,2,0,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
      T5(y,r);
      MPI_Send(y,N,MPI_DOUBLE,0,0,MPI_COMM_WORLD);
    }
    else if (rank==2) {
      MPI_Recv(x,N,MPI_DOUBLE,0,0,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
      r=T3(z,x);
      MPI_Send(&r,1,MPI_DOUBLE,1,0,MPI_COMM_WORLD);
      T6(z);
      MPI_Send(z,N,MPI_DOUBLE,0,0,MPI_COMM_WORLD);
    }
    return s;
  }
```

0.2 p. (c) Calcula el cost paral·lel, detallant els càlculs.

Solució:

Cal calcular el temps aritmètic:

$$t_a(N,3) = N^2 + \max(N^2+N^2, N^2+N^2, 2N^2) + N^2 = 4N^2 \text{ flops},$$

i el temps de comunicacions, tenint en compte que hi ha 4 missatges de N elements i un d'un element:

$$t_c(N,3) = 4(t_s + Nt_w) + (t_s + t_w) = 5t_s + (4N+1)t_w \approx 5t_s + 4Nt_w.$$

El cost paral·lel és la suma dels dos temps anteriors:

$$t(N,3) = 4N^2 \text{ flops} + 5t_s + 4Nt_w.$$

Qüestió 2 (1.2 punts)

Donada la següent funció:

```
void normalitza( float v[N] ) {
  int i;
  float max = -FLT_MAX;
  for (i=0;i<N;i++)
    if( v[i] > max )
      max = v[i];
  for (i=0;i<N;i++)
    v[i] /= max;
}
```

0.9 p. (a) Fes una versió paral·lela usant MPI, suposant que el vector `v` es troba inicialment només en el procés 0 i, en finalitzar, el procés 0 ha de contenir el vector `v` modificat. Hauran de distribuir-se les dades necessàries perquè tots els càlculs es repartisquen de manera equitativa. Nota: Es pot assumir que `N` és divisible entre el nombre de processos.

Solució:

```
void normalitza( float v[N] ) {
  int i, p;
  float max, max_loc=-FLT_MAX;
  float vloc[N];
  MPI_Comm_size( MPI_COMM_WORLD, &p );
  MPI_Scatter( v, N/p, MPI_FLOAT, vloc, N/p, MPI_FLOAT, 0, MPI_COMM_WORLD );
  for (i=0;i<N/p;i++)
    if( vloc[i] > max_loc )
```

```
        max_loc = vloc[i];
    MPI_Allreduce( &max_loc, &max, 1, MPI_FLOAT, MPI_MAX, MPI_COMM_WORLD );
    for (i=0;i<N/p;i++)
      vloc[i] /= max;
    MPI_Gather( vloc, N/p, MPI_FLOAT, v, N/p, MPI_FLOAT, 0, MPI_COMM_WORLD );
  }
```

0.3 p.

(b) Calcula el cost computacional (en flops) de la versió seqüencial i de la versió paral·lela desenvolupada en l'apartat anterior. Detalla el cost de les operacions de comunicacions emprades. Assumeix que las comparacions entre números reals costen 1 flop.

Solució:

Cost seqüencial: $t(N) = \sum_{i=0}^{N-1} 1 + \sum_{i=0}^{N-1} 1 = 2N$ flops.

Cost aritmètic paral·lel: $t_a(N,p) = \sum_{i=0}^{\frac{N}{p}-1} 1 + \sum_{i=0}^{\frac{N}{p}-1} 1 = \frac{2N}{p}$ flops.

Cost Scatter: $(p-1)\left(t_s + \frac{N}{p}t_w\right) \approx pt_s + Nt_w$.

Cost Allreduce (Reduce + Bcast, $p-1$ flops): $2(p-1)(t_s+t_w) + (p-1) \approx 2pt_s + 2pt_w + p$.

Cost Gather: $(p-1)\left(t_s + \frac{N}{p}t_w\right) \approx pt_s + Nt_w$.

Cost paral·lel: $t(N,p) \approx 4pt_s + 2(N+p)t_w + \frac{2N}{p} + p$ flops.

Qüestió 3 (1 punt)

Es vol implementar una funció MPI per a transmetre una matriu quadrada d'enters de dimensió n, sent n un nombre parell, del procés P0 al procés P1, de manera que el procés P1 reba la matriu amb les columnes adjacents intercanviades; és a dir, si la matriu del procés P0 és, per exemple:

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{bmatrix}$$

el procés P1 rebrà la matriu:

$$B = \begin{bmatrix} 2 & 1 & 4 & 3 \\ 6 & 5 & 8 & 7 \\ 10 & 9 & 12 & 11 \\ 14 & 13 & 16 & 15 \end{bmatrix}.$$

La transmissió de la matriu ha de fer-se mitjançant només dos missatges i sense realitzar còpies intermèdies de les dades. La capçalera de la funció a implementar serà:

```
comunica_matriz(int A[n][n], int B[n][n], int rank)
```

on `A` és la matriu emmagatzemada en el procés P0, `B` és la matriu on s'han de rebre les dades en el procés P1 y `rank` és l'identificador del procés local (el programa pot tindre més de dos processos).

Solució: Tenint en compte que les files de les matrius es troben emmagatzemades en posicions consecutives de memòria i definint un nou tipus de dades derivat consistent en $n^2/2$ blocs d'1 element amb un `stride` igual a 2, es pot obtenir la matriu B en el procés P_1 amb només dos missatges.

```
void comunica_matriz(int A[n][n], int B[n][n], int rank){
  MPI_Status stat;
  MPI_Datatype int_col;
  MPI_Type_vector(n*n/2, 1, 2, MPI_INT, &int_col);
  MPI_Type_commit(&int_col);
  if (rank==0) {
    MPI_Send(&A[0][0], 1, int_col, 1, 100, MPI_COMM_WORLD);
    MPI_Send(&A[0][1], 1, int_col, 1, 200, MPI_COMM_WORLD);
```

```
    }
    else if (rank==1) {
      MPI_Recv(&B[0][1], 1, int_col, 0, 100, MPI_COMM_WORLD, &stat);
      MPI_Recv(&B[0][0], 1, int_col, 0, 200, MPI_COMM_WORLD, &stat);
    }
    MPI_Type_free(&int_col);
  }
```