UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Question 1 (1.3 points)
Given the following function:

```
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) Implement a parallel version using MPI, assuming that the input data is in process 0 and that the results must be complete in that process at the end of the execution. The problem size can be assumed to be a multiple of the number of processes.

Solution:

```
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) Obtain the expression of the parallel execution time, indicating the communication cost of each collective operation used.

Solution:

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

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

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

Question 2 (1.1 points)
We want to send the first and last rows and columns of a rectangular matrix of size $M \times N$ from the process identified as *root* to the rest of the processes. Below is an example for a matrix of dimensions $M = 4$ and $N = 5$, where the terms identified with the symbol x correspond to all those to be sent:

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

0.9 p. (a) Complete the body of the function whose header is included below to carry out the communication. The parameters of the function correspond to the identifier of the invoking process (*myid*), the total number of processes (*np*) and the identifier of the root process that initially has the starting A array and performs the communication (*root*).

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

All processes with an identifier other than *root* must store the data received in the same A array provided as a parameter to the function. For this purpose, point-to-point communication operations and derived data types shall be used, so as to minimize the number of transmissions to be made by the process *root* to the rest. It will be valued that no element is sent more than once to each process (especially, the elements of the 4 corners of the matrix).

Solution:

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

  // ALTERNATIVE 2
void send_perimeter_matrix(double A[M][N], int myid, int np, int root) {
  int p;
  MPI_Datatype rows_consecutive;
```

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

0.2 p. (b) Obtain the communications cost.

Solution:
Alternative 1:

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

Alternative 2:

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

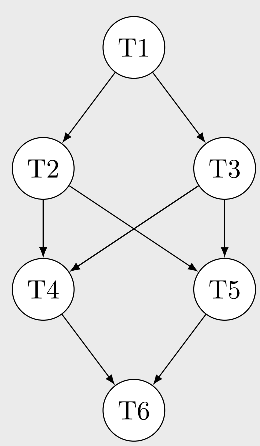
Question 3 (1.1 points)

Given the following function, where the functions corresponding to the tasks (`T1` to `T6`) modify only their last argument and where the cost of each of these functions is $4N^2$ flops, except the functions `T2` and `T4`, whose cost is $3N^2$ flops each.

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

0.3 p. (a) Draw the graph of data dependencies between tasks.

Solution:



0.6 p. (b) Implement a parallel version with MPI for 2 processes, using point-to-point communication operations. You can assume that the matrix `A` is initially in process `0`. Regarding the vector `w`, its initial content is not used and its correct final content can remain in any one of the processes. Justify the task assignment used.

Solution: We use the assignment:

$P_0 : T_1, T_2, T_5$

$P_1 : T_3, T_4, T_6$

Such an assignment maximizes parallelism, since the independent tasks are in different processes. In addition, communications are minimized by avoiding communicating the `A` matrix.

```
void func_par(double A[N][N], double w[N]) {
  double x[N],y[N],v[N],alpha;
  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,&alpha);
    MPI_Send(&alpha,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(&alpha,1,MPI_DOUBLE,0,0,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
    T6(alpha,w);
  }
}
```

0.2 p. (c) Calculate the sequential cost and the parallel cost.

Solution: Sequential cost:

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

Parallel cost:

$$t_a(N,2) = 4N^2 + 4N^2 + 4N^2 + 4N^2 = 16N^2 \text{ flops}$$

$$t_c(N,2) = 3(t_s + Nt_w) + (t_s + t_w) = 4t_s + (3N+1)t_w \approx 4t_s + 3Nt_w$$

$$t(N,2) = 16N^2 \text{ flops} + 4t_s + 3Nt_w$$