Question 1 (1.3 points)

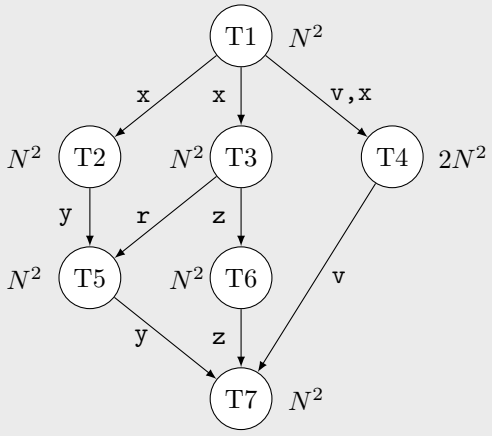
Given the following function, where functions `T1` to `T7` modify only their first argument and where the cost of each of those functions is N^2 flops, except function `T4`, whose cost is $2N^2$ flops:

```
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) Draw the task dependency graph.

Solution:

Even though it is not requested, the graph below includes the cost of the tasks and the variable corresponding to the dependencies between tasks, which is useful for the next sections.



0.7 p. (b) Implement an MPI parallel version, using the appropriate number of processes to maximize parallelism. Take into account that vector `v` is stored initially only in process 0, and that the value returned by the function must be correct also in process 0.

Solution: We will use 3 processes, since that is the maximum degree of concurrency, and the assignment: $P_0 : T_1, T_4, T_7$. $P_1 : T_2, T_5$. $P_2 : T_3, T_6$.

Such assignment maximizes parallelism, since independent tasks are assigned to different processes and the cost of the tasks done in parallel (tasks 2 to 6) is well balanced among the three processes. Also, communications are minimized by avoiding to send 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) Compute the parallel cost, detailing the calculations.

Solution:

We must compute the arithmetic cost:

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

and the communication cost, taking into account that there are 4 messages of length N elements and one of length one element:

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

The parallel cost is the sum of the two previous costs:

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

Question 2 (1.2 points)

Given the following function:

```
void normalize( 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) Write a parallel version using MPI, assuming that vector `v` is located initially only on process 0 and, at the end, process 0 must store the modified vector `v`. All the necessary data must be distributed in a way that all the calculations are shared equally among the processes. Note: You can assume that `N` is exactly divisible by the number of processes.

Solution:

```
void normalize( 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) Calculate the computational cost (in flops) of the sequential version and the parallel version developed in the previous section. Give also the cost of the communication operations used. Assume that the comparisons of real numbers have a cost of 1 flop.

Solution:

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

Parallel arithmetic cost: $t_a(N,p) = \sum_{i=0}^{\frac{N}{p}-1} 1 + \sum_{i=0}^{\frac{N}{p}-1} 1 = \frac{2N}{p}$ flops.

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

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

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

Parallel cost: $t(N,p) \approx 4pt_s + 2(N+p)t_w + \frac{2N}{p} + p$ flops.

Question 3 (1 point)

We want to implement an MPI function to communicate a square matrix of integers of dimension n, being n an even number, from process P0 to process P1, in such a way that process P1 receives the matrix with the adjacent columns exchanged; that is, if the matrix in process P0 is, for instance:

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

then process P1 will receive this matrix:

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

The communication of the matrix must be done by means of only two messages and without performing intermediate copies of the data. The prototype of the function to be implemented is:

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

where `A` is the matrix stored in process P0, `B` is the matrix where data will be received in process P1 and `rank` is the local process identifier (the program may have more than two processes).

Solution: Taking into account that rows of the matrices are stored in consecutive locations in memory and defining a new derived data type consisting of $n^2/2$ blocks of 1 element with a stride equal to 2, it is possible to obtain matrix B in process P_1 with only two messages.

```
void commun_matrix(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);
}
```