Parallel Computing
Degree in Computer Science Engineering (ETSINF)
Year 2012-13 ◇ Partial exam 5/11/2012 ◇ Duration: 1h 30m

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Question 1 (0.5 points)

Assuming the following function, which searches for a value in a vector, implement a parallel version using OpenMP. As in the original function, the parallel function should end as soon as the sought element is found.

```
int search(int x[], int n, int value)
{
   int found=0, i=0;
   while (!found && i<n) {
      if (x[i]==value) found=1;
      i++;
   }
   return found;
}
```

Question 2 (0.75 points)

The infinite-norm of a matrix $A \in \mathbb{R}^{n \times n}$ is defined as the maximum of the sum of the absolute values of the elements in each row:

$$\|A\|_\infty = \max_{i=0,\ldots,n-1} \left\{ \sum_{j=0}^{n-1} |a_{i,j}| \right\}$$

The following sequential code implements such operation for a square matrix.

```
#include <math.h>
#define DIMN 100

double infNorm(double A[DIMN][DIMN], int n)
{
  int i,j;
  double s,norm=0;

  for (i=0; i<n; i++) {
    s = 0;
    for (j=0; j<n; j++)
      s += fabs(A[i][j]);
    if (s>norm)
      norm = s;
  }
  return norm;
}
```

0.4 p. (a) Implement a parallel version of this algorithm using OpenMP. Justify each change introduced.

0.2 p. (b) Calculate the computational cost (in flops) for the original sequential version and for the parallel version implemented.
N.B.: Assume that the matrix dimension n is an exact multiple of the number of threads p. Assume that the computational cost for function `fabs` is 1 flop.

0.15 p. (c) Calculate the speed-up and efficiency of the parallel code when run with p processors.

Question 3 (1.25 points)
Given the following function:

```
double function(int n, double u[], double v[], double w[], double z[])
{
  int i;
  double sv,sw,res;

  compute_v(n,v);         /* task 1 */
  compute_w(n,w);         /* task 2 */
  compute_z(n,z);         /* task 3 */
  compute_u(n,u,v,w,z);   /* task 4 */
  sv = 0;
  for (i=0; i<n; i++)  sv = sv + v[i];         /* task 5 */
  sw = 0;
  for (i=0; i<n; i++)  sw = sw + w[i];         /* task 6 */
  res = sv+sw;
  for (i=0; i<n; i++)  u[i] = res*u[i];        /* task 7 */
  return res;
}
```

The four `compute_X` functions have as input vectors those received as arguments. Those functions modify the vector included in the name of the function. Each function only modifies the vector included in the name. Therefore, the function `compute_u` uses vectors `v`, `w` and `z` to perform computations that are stored in the vector `u`, but none of `v`, `w`, or `z` are updated.

Therefore, functions `compute_v`, `compute_w` and `compute_z` are independent and can be performed concurrently. However, function `compute_u` must wait for the other functions to complete, since it needs (`v`,`w`,`z`).

0.2 p. (a) Draw the dependency graph for the different tasks.

0.75 p. (b) Parallelize the function efficiently.

0.3 p. (c) Assuming that the cost of all the functions `compute_X` is the same and that the cost of the other loops is negligible, what will be the maximum speed-up?

Parallel Computing
Degree in Computer Science Engineering (ETSInf)
Year 2012-13 ◇ Final exam 21/1/2013 ◇ Duration: 3h

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Question 1 (1.25 points)
Given the following function:

```
double function(double A[M][N])
{
  int i,j;
  double sum;
  for (i=0; i<M-1; i++) {
    for (j=0; j<N; j++) {
      A[i][j] = 2.0 * A[i+1][j];
    }
  }
  sum = 0.0;
  for (i=0; i<M; i++) {
    for (j=0; j<N; j++) {
      sum = sum + A[i][j];
    }
  }
  return sum;
}
```

0.2 p. (a) Calculate the theoretical cost (in flops).

0.6 p. (b) Implement a parallel version using OpenMP. Justify any modifications that you make. Efficiency of the proposed solution will be taken into account.

0.3 p. (c) Calculate the speed-up that can be achieved with p processors assuming that M and N are exact multiples of p.

0.15 p. (d) Give an upper value for the speed-up (when p tends to infinity) in the case that only the first part is performed in parallel and the second part (the one related to the sum) is performed sequentially.

Question 2 (0.75 points)
Given the following function:

```
double fun_mat(double a[n][n], double b[n][n])
{
  int i,j,k;
  double aux,s=0.0;
  for (i=0; i<n; i++) {
    for (j=0; j<n; j++) {
      aux=0.0;
```

```
      s += a[i][j];
      for (k=0; k<n; k++) {
        aux += a[i][k] * a[k][j];
      }
      b[i][j] = aux;
    }
  }
  return s;
}
```

0.4 p. (a) Describe how each of the three loops can be parallelised using OpenMP. Which will be the most efficient one? Justify your answer.

0.15 p. (b) Assuming that only the outer loop is parallelised, indicate the a priori sequential and parallel costs in flops. Calculate also the speed-up assuming that the number of threads (and processors) is n.

0.2 p. (c) Add the code lines required for showing on the screen the number of iterations that thread 0 has performed, in the case that the outer loop is parallelised.

Question 3 (0.5 points)

Parallelize the following fragment of code by means of OpenMP sections. The second argument in functions `fun1`, `fun2` and `fun3` is an input-output parameter, that is, these functions use and modify the value of `a`.

```
int n=...;
double a,b[3];

a = -1.8;
fun1(n,&a);
b[0] = a;
a = 3.2;
fun2(n,&a);
b[1] = a;
a = 0.25;
fun3(n,&a);
b[2] = a;
```

Question 4 (1 point)

We want to parallelize the following code with MPI. Suppose that 3 processes are available.

```
double a[N],b[N],c[N],v=0.0,w=0.0;
T1(a,&v);
T2(b,&w);
T3(b,&v);
T4(c,&w);
T5(c,&v);
T6(a,&w);
```

All functions read and modify both arguments, also the vectors. Suppose that vectors `a`, `b` and `c` are stored in P_0, P_1 and P_2, respectively, and they are too large to afford its efficient sending from one process to another.

0.25 p. (a) Draw the dependency graph of the different tasks, indicating which task is assigned to each process.

0.75 p. (b) Write the MPI code that solves the problem.

Question 5 (1 point)

The ∞-norm of a matrix is defined as the maximum sum of the absolute values of the elements in each row: $\max_{i=1..n} \left\{ \sum_{j=0}^{m-1} |a_{i,j}| \right\}$. The following sequential code implements such operation for a square matrix.

```
#include <math.h>
#define N 800

double infNorm(double A[][N]) {
  int i,j;
  double s,nrm=0.0;

  for (i=0; i<N; i++) {
    s=0.0;
    for (j=0; j<N; j++)
      s+=fabs(A[i][j]);
    if (s>nrm)
      nrm=s;
  }
  return nrm;
}
```

0.5 p. (a) Implement an MPI parallel version using collective operations whenever possible. Assume that the size of the problem is an exact multiple of the number of processes. The matrix is stored initially in P_0 and the result must also end in P_0.

Note: suggest using the following header for the parallel function, where `Alocal` is a matrix that has already been allocated in memory and can be used by the function to store the local part of matrix `A`.

```
double infNormPar(double A[][N], double ALocal[][N])
```

0.25 p. (b) Obtain the computational and communication cost for the parallel algorithm. Assume that the `fabs` operation has a negligible cost, as well as in the case of comparisons.

0.25 p. (c) Obtain the speedup and efficiency when the problem size tends to infinity.

Question 6 (0.5 points)

Let `A` be a bidimensional array of double precision real numbers, of dimension `N`×`N`. Define an MPI derived data type that allows to send a submatrix of size 3×3. For instance, the submatrix that starts in `A[0][0]` would be the elements marked with a $\star$:

$$\mathtt{A} = \begin{bmatrix} \star & \star & \star & \cdot & \cdot & \cdot \\ \star & \star & \star & \cdot & \cdot & \cdot \\ \star & \star & \star & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix}.$$

(a) Write the corresponding calls for sending the block in the figure from P_0 and receiving it in P_1.

(b) Indicate what should be modified in the previous code so that the block sent by P_0 is the one that starts in the position (0,3), and is received in P_1 overwriting the block that starts in the position (3,0),

Parallel Computing
Degree in Computer Science Engineering (ETSInf)
Year 2013-14 ◇ Partial exam 11/11/2013 ◇ Duration: 1h 30m

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Question 1 (1 point)
The following code has to be efficiently parallelised using OpenMP.

```
int cmp(int n, double x[], double y[], int z[])
{
  int i, v, equal=0;
  double aux;
  for (i=0; i<n; i++) {
    aux = x[i] - y[i];
    if (aux > 0) v = 1;
    else if (aux < 0) v = -1;
    else v = 0;
    z[i] = v;
    if (v == 0) equal++;
  }
  return equal;
}
```

0.4 p. (a) Implement a parallel version using only `parallel for` constructions.

0.6 p. (b) Implement a parallel version without using any of the following primitives: `for`, `section`, `reduction`.

Question 2 (0.8 points)
Given the following function:

```
void normalize(double A[N][N])
{
  int i,j;
  double sum=0.0,factor;
  for (i=0; i<N; i++) {
    for (j=0; j<N; j++) {
      sum = sum + A[i][j]*A[i][j];
    }
  }
  factor = 1.0/sqrt(sum);
  for (i=0; i<N; i++) {
    for (j=0; j<N; j++) {
      A[i][j] = factor*A[i][j];
    }
  }
}
```

0.4 p. (a) Implement a parallel version with OpenMP using two separated parallel regions (blocks).

0.4 p. (b) Parallelize it using a single OpenMP parallel region for both pairs of loops. In this case, could we use the `nowait` clause? Justify your answer.

Question 3 (1.2 points)

Considering the definition of the following functions :

```
/* Matrix product C = A*B */
void matmult(double A[N][N],
       double B[N][N],double C[N][N])
{
  int i,j,k;
  double sum;
  for (i=0; i<N; i++) {
    for (j=0; j<N; j++) {
      sum = 0.0;
      for (k=0; k<N; k++) {
        sum = sum + A[i][k]*B[k][j];
      }
      C[i][j] = sum;
    }
  }
}
```

```
/* Generate a symmetric matrix A+A' */
void symmetrize(double A[N][N])
{
  int i,j;
  double sum;
  for (i=0; i<N; i++) {
    for (j=0; j<=i; j++) {
      sum = A[i][j]+A[j][i];
      A[i][j] = sum;
      A[j][i] = sum;
    }
  }
}
```

we want to parallelize the following code:

```
matmult(X,Y,C1);    /* T1 */
matmult(Y,Z,C2);    /* T2 */
matmult(Z,X,C3);    /* T3 */
symmetrize(C1);     /* T4 */
symmetrize(C2);     /* T5 */
matmult(C1,C2,D1);  /* T6 */
matmult(D1,C3,D);   /* T7 */
```

0.3 p. (a) Implement a parallel version based on loops.

0.4 p. (b) Draw the task dependency graph, considering that in this case that each call to the `matmult` and `symmetrize` functions is an independent task. Indicate the maximum degree of concurrency, the length of the critical path and the average degree of concurrency. Note: to compute such values, you should obtain the cost in flops for both functions.

0.5 p. (c) Implement a parallelisation based on sections, according to the previous task dependency graph.

Parallel Computing
Degree in Computer Science Engineering (ETSInf)
Year 2013-14 ◇ Partial exam 13/1/2014 ◇ Duration: 1h 30m

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Question 1 (1.2 points)

Next program computes the number of occurrences for a specific value in a matrix.

```
#include <stdio.h>
#define DIM 1000

void read(double A[DIM][DIM], double *x)
{ ... }

int main(int argc, char *argv[])
{
  double A[DIM][DIM], x;
  int i,j,cont;

  read(A,&x);
  cont=0;
  for (i=0; i<DIM; i++)
    for (j=0; j<DIM; j++)
      if (A[i][j]==x) cont++;
  printf("%d occurrences\n", cont);
  return 0;
}
```

0.8 p. (a) Implement an MPI parallel version of the program above, using collective communication operations if possible and convenient. The function `read` should be called only by process 0. It can be assumed that `DIM` is exactly divisible by the number of processes. Note: Write the whole program including the declaration of variables and the necessary calls to initialize and finalize MPI.

0.4 p. (b) Obtain the parallel execution time, assuming that the cost of comparing two real numbers is 1 flop. Note: for the communication cost, consider a simple implementation of the collective operations used.

Question 2 (1 point)

An MPI program has a vector x, of n real elements that is cyclically distributed among p processes. Each process stores its elements in the array `xloc`.

Implement the following function that will be called by all the processes to perform the proper communication operations to receive in the `x` array in process 0 a copy of the complete distributed vector. The elements in `x` should be correctly ordered according to their real global index.

Each process (from 1 to $p-1$) will send all the elements to process 0 in a single message.

```
void communicate_vec(double xloc[], int n, int p, int rank, double x[])
/* rank is the index of the local process calling the function */
/* This function assumes that n is an exact multiple of p      */
```

Question 3 (0.8 points)

The following fragment of code is incorrect (from the semantic point of view, not because of syntax errors). Describe the reason and propose two different solutions.

```
MPI_Status stat;
int sbuf[N], rbuf[N], rank, size, src, dst;
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
src = (rank==0)? size-1: rank-1;
dst = (rank==size-1)? 0: rank+1;
MPI_Ssend(sbuf, N, MPI_INT, dst, 111, MPI_COMM_WORLD);
MPI_Recv(rbuf, N, MPI_INT, src, 111, MPI_COMM_WORLD, &stat);
```

Parallel Computing
Degree in Computer Science Engineering (ETSInf)
Year 2013-14 ◇ Final exam 27/1/14 ◇ Block OpenMP ◇ Duration: 1h 30m

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Question 1 (1 point)

The following function normalizes the elements of a vector of positive real numbers in a way that the end values remain between 0 and 1, using the maximum and the minimum.

```
void normalize(double *a, int n)
{
  double mx, mn, factor;
  int i;

  mx = a[0];
  for (i=1;i<n;i++) {
    if (mx<a[i]) mx=a[i];
  }
  mn = a[0];
  for (i=1;i<n;i++) {
    if (mn>a[i]) mn=a[i];
  }
  factor = mx-mn;
  for (i=0;i<n;i++) {
    a[i]=(a[i]-mn)/factor;
  }
}
```

0.75 p. (a) Parallelize the program with OpenMP in the most efficient way, by means of a single parallel region. We assume that the value of `n` is very large and we want that the parallelization works efficiently for an arbitrary number of threads.

0.25 p. (b) Include the necessary code so that the number of used threads is printed once.

Question 2 (1 point)

Given the following code fragment, where the vector of indices `ind` contains integer values between 0 and $m-1$ (being m the dimension of `x`), possibly with repetitions:

```
for (i=0; i<n; i++) {
   s = 0;
   for (j=0; j<i; j++) {
      s += A[i][j]*b[j];
   }
   c[i] = s;
   x[ind[i]] += s;
}
```

0.5 p. (a) Write a parallel implementation with OpenMP, in which the iterations of the outer loop are shared.

0.25 p. (b) Write a parallel implementation with OpenMP, in which the iterations of the inner loop are shared.

0.25 p. (c) For the implementation of item (a), indicate if we can expect performance differences depending on the schedule used. In this case, which schedule schemes would be better and why?

Question 3 (1 point)

In the following function, `T1`, `T2`, `T3` modify `x`, `y`, `z`, respectively.

```
double f(double x[], double y[], double z[], int n)
{
   int i, j;
   double s1, s2, a, res;

   T1(x,n);    /* Task T1 */
   T2(y,n);    /* Task T2 */
   T3(z,n);    /* Task T3 */

   /* Task T4 */
   for (i=0; i<n; i++) {
      s1=0;
      for (j=0; j<n; j++) s1+=x[i]*y[i];
      for (j=0; j<n; j++) x[i]*=s1;
   }

   /* Task T5 */
   for (i=0; i<n; i++) {
      s2=0;
      for (j=0; j<n; j++) s2+=y[i]*z[i];
      for (j=0; j<n; j++) z[i]*=s2;
   }

   /* Task T6 */
   a=s1/s2;
   res=0;
   for (i=0; i<n; i++) res+=a*z[i];
   return res;
}
```

0.2 p. (a) Draw the task dependency graph.

0.5 p. (b) Make a task-level parallelization by means of OpenMP (not a loop-level parallelization), based on the dependency graph.

0.3 p. (c) Indicate the a priori cost of the sequential algorithm, the parallel algorithm, and the resulting speedup. Suppose the cost of tasks 1, 2 and 3 is $2n^2$ flops each.

Question 4 (0.8 points)

We want to measure the latency of a ring of `p` MPI processes, where the latency must be understood as the time that a message of length 0 spends when circulating across all processes. A ring of `p` MPI processes works as follows: P_0 sends the message to P_1, when the reception is complete, it resends the message to P_2, and so on until it arrives to P_{p-1} who will send it again to P_0. Write an MPI program that implements this communication scheme and shows the latency. It is recommended that the message goes around the ring several times, and then take the average time in order to get a more reliable measurement.

Question 5 (1.2 points)

The following MPI program must compute the sum of two matrices A and B of dimensions $M \times N$ using a row cyclic distribution, assuming that the number of processes `p` is a divisor of M and having into account that P_0 has initially stored matrices A and B.

```
int p, rank, i, j, mb;
double A[M][N], B[M][N], Al[M][N], Bl[M][N];

MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &p);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
if (rank==0) read(A,B);

/* (a) Cyclic distribution of rows of A and B */
/* (b) Local computation of Al+Bl */
/* (c) Gather the results in process 0 */

if (rank==0) write(A);
MPI_Finalize();
```

0.5 p. (a) Implement the cyclic distribution of rows of matrices A and B, where `Al` and `Bl` are the local matrices. In order to achieve this distribution you must either define a new MPI datatype or use collective communications.

0.2 p. (b) Implement the local computation of the sum `Al+Bl`, storing the result in `Al`.

0.5 p. (c) Write the necessary code so that P_0 stores in `A` the matrix $A+B$. For this, P_0 must receive from the rest of processes the local matrices `Al` obtained in the previous step.

Question 6 (1 point)

We want to implement the computation of the ∞-norm of a square matrix, which is obtained as the maximum of the sums of the absolute values of the elements in each row,

$\max_{i=0}^{n-1} \left\{ \sum_{j=0}^{n-1} |a_{i,j}| \right\}$. For this, a master-slave scheme is proposed. The next function corresponds to the master (the process with identifier 0). The matrix is stored by rows in a uni-dimensional array, and we assume that it is sparse (it has many zeros), and therefore the master sends only the nonzero elements (function `compress`).

```
int compress(double *A,int n,int i,double *buf)
{
  int j,k = 0;
  for (j=0;j<n;j++)
    if (A[i*n+j]!=0.0) { buf[k] = A[i*n+j]; k++; }
  return k;
}
double master(double *A,int n)
{
  double buf[n];
  double norm=0.0,value;
  int row,complete=0,size,i,k;
  MPI_Status status;
  MPI_Comm_size(MPI_COMM_WORLD,&size);
  for (row=0;row<size-1;row++) {
    if (row<n) {
      k = compress(A, n, row, buf);
      MPI_Send(buf, k, MPI_DOUBLE, row+1, TAG_ROW, MPI_COMM_WORLD);
    } else
      MPI_Send(buf, 0, MPI_DOUBLE, row+1, TAG_END, MPI_COMM_WORLD);
  }
  while (complete<n) {
    MPI_Recv(&value, 1, MPI_DOUBLE, MPI_ANY_SOURCE, TAG_RESU,
             MPI_COMM_WORLD, &status);
    if (value>norm) norm=value;
    complete++;
    if (row<n) {
      k = compress(A, n, row, buf);
      row++;
      MPI_Send(buf, k, MPI_DOUBLE, status.MPI_SOURCE, TAG_ROW, MPI_COMM_WORLD);
    } else
      MPI_Send(buf, 0, MPI_DOUBLE, status.MPI_SOURCE, TAG_END, MPI_COMM_WORLD);
  }
  return norma;
}
```

Implement the part of the working processes, completing the following function:

```
void worker(int n)
{
  double buf[n];
```

Note: For the absolute value you can use

```
double fabs(double x)
```

Remember that `MPI_Status` contains, among other, the fields `MPI_SOURCE` and `MPI_TAG`.

Parallel Computing
Degree in Computer Science Engineering (ETSInf)
Year 2014-15 ◇ Partial exam 3/11/2014 ◇ Duration: 1h 30m

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Question 1 (1 point)
Given the following function:

```
double ej(double x[M], double y[N], double A[M][N])
{
   int i,j;
   double aux,s=0.0;
   for (i=0; i<M; i++)
      x[i] = x[i]*x[i];
   for (i=0; i<N; i++)
      y[i] = 1.0+y[i];
   for (i=0; i<M; i++)
      for (j=0; j<N; j++) {
         aux = x[i]-y[j];
         A[i][j] = aux;
         s += aux;
     }
   return s;
}
```

0.6 p. (a) Implement an efficient parallel version using OpenMP, using just one parallel region.

0.3 p. (b) Compute the number of flops of the initial solution and of the parallel version.

0.1 p. (c) Obtain the speed-up and the efficiency.

Question 2 (1 point)
Given the following fragment of a code:

```
minx = minimum(x,n);       /* T1 */
maxx = maximum(x,n);       /* T2 */
compute_z(z,minx,maxx,n); /* T3 */
compute_y(y,x,n);          /* T4 */
compute_x(x,y,n);          /* T5 */
compute_v(v,z,x);          /* T6 */
```

0.3 p. (a) Draw a dependency graph of the tasks, taking into account that functions `minimum` and `maximum` do not change their arguments, and the rest of the functions only change the first argument.

0.4 p. (b) Implement a parallel version of the code using OpenMP.

0.3 p. (c) If the cost of the tasks is n flops (except for task 4 which takes $2n$ flops), calculate the length of the critical path and the average concurrency degree. Compute the speed-up and efficiency of the implementation written in the previous part, if 5 processors were used.

Question 3 (1 point)

Given the function:

```
int function(int n, double v[])
{
  int i, max_pos=-1;
  double sum, norm, aux, max=-1;

  sum = 0;
  for (i=0;i<n;i++)
    sum = sum + v[i]*v[i];
  norm = sqrt(sum);

  for (i=0;i<n;i++)
    v[i] = v[i] / norm;

  for (i=0;i<n;i++) {
    aux = v[i];
    if (aux < 0)  aux = -aux;
    if (aux > max) {
      max_pos = i;  max = aux;
    }
  }
  return max_pos;
}
```

0.6 p. (a) Implement a parallel version using OpenMP, using a single parallel region.

0.2 p. (b) Would it be reasonable to use the `nowait` clause to any of the loops? Why? Justify each loop separately.

0.2 p. (c) What will you add to guarantee that all the iterations in all the loops are distributed in blocks of two iterations among the threads?

Parallel Computing
Degree in Computer Science Engineering (ETSInf)
Year 2014-15 ◇ Partial exam 12/1/2015 ◇ Duration: 1h 30m

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Question 1 (1 point)

The next function displays on the screen the maximum of a vector `v` with `n` elements and its location:

```
void func(double v[], int n) {
   double max = v[0];
   int i, posmax = 0;
   for (i=1; i<n; i++) {
      if (v[i]>max) {
         max = v[i];
         posmax=i;
      }
   }
   printf("Maxim: %f. Position: %d\n", max, posmax);
}
```

Write an MPI parallel version using the next header, where arguments `rank` and `np` have been obtained calling `MPI_Comm_rank` and `MPI_Comm_size`, respectively.

```
void func_par(double v[], int n, int rank, int np)
```

The function should assume that initially, process 0 will have the vector in the array `v`. In the rest of the processes, this array can be used to store each local part. You should exchange the necessary data to ensure that the computation of the maximum is balanced among all the processes. Finally, only process 0 should show the message on the screen. **You must use point-to-point communications calls and not collective operations**.

N.B. You can assume that `n` is an exact multiple of the number of processes.

Question 2 (1 point)

0.6 p. (a) Implement a function that sums two square matrices `a` and `b` using MPI collective communication primitives. The function will store the result in `a`. Matrices `a` and `b` are initially stored in the memory of process P_0 and the final result should also be stored in P_0. We assume that the number of rows from the matrices (`N`, constant) is an exact multiple of the number of processes. The header for the function will be:

```
void sum_mat(double a[N][N],double b[N][N])
```

0.4 p. (b) Obtain the parallel time, the speed-up and the efficiency of the implementation proposed in the previous part. Describe how the cost of the collective operations is computed (number of messages, ant their length). You can assume a simple implementation of such collective operations.

Question 3 (1 point)

Implement a function where, given a matrix A of $N \times N$ real numbers and an index k (between 0 and $N-1$), the row k and column k of the matrix are communicated from process 0 to the rest of processes (without communicating any other element of the matrix). The header of the function will be:

```
void bcast_row_col(double A[N][N], int k)
```

You should create and use a datatype for representing a column from the matrix. It is not necessary that you send both the column and the row in the same message, you can send them separately.

Parallel Computing
Degree in Computer Science Engineering (ETSInf)
Year 2014-15 ◇ Final exam 21/1/15 ◇ Block OpenMP ◇ Duration: 1h 30m

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Question 1 (1 point)

Given the following C function:

```
double fun( int n, double a[], double b[] )
{
  int i,ac,bc;
  double asum,bsum,thres;

  asum = 0; bsum = 0;
  for (i=0; i<n; i++)
    asum += a[i];
  for (i=0; i<n; i++)
    bsum += b[i];
  thres = (asum + bsum) / 2.0 / n;

  ac = 0; bc = 0;
  for (i=0; i<n; i++) {
    if (a[i]>thres) ac++;
    if (b[i]>thres) bc++;
  }
  return thres/(ac+bc);
}
```

0.7 p. (a) Parallelize it efficiently with OpenMP directives (without changing the existing code).

0.3 p. (b) Indicate the a priori cost in flops, both sequential and parallel (considering only floating-point operation and assuming that a comparison implies a subtraction). What is the speed-up and efficiency for p processors? Assume that n is an exact multiple of p.

Question 2 (1.3 points)

We want to parallelize the following program by means of OpenMP, where `generate` is a function previously defined elsewhere.

```
double fun1(double a[],int n,
            int v0)
{
  int i;
  a[0] = v0;
  for (i=1;i<n;i++)
    a[i] = generate(a[i-1],i);
}
```

```
double compare(double x[],double y[],int n)
{
  int i;
  double s=0;
  for (i=0;i<n;i++)
    s += fabs(x[i]-y[i]);
  return s;
}
```

```
/* fragment of the main program */
int i, n=10;
double a[10], b[10], c[10], x=5, y=7, z=11, w;
fun1(a,n,x);           /* T1 */
fun1(b,n,y);           /* T2 */
fun1(c,n,z);           /* T3 */
x = compare(a,b,n);    /* T4 */
y = compare(a,c,n);    /* T5 */
z = compare(c,b,n);    /* T6 */
w = x+y+z;             /* T7 */
printf("w:%f\n", w);
```

0.2 p. (a) Parallelize the code efficiently at the level of the loops.

0.3 p. (b) Draw the task dependency graph, according to the numbering of tasks indicated in the code.

0.5 p. (c) Parallelize the code efficiently in terms of tasks, from the previous dependency graph.

0.3 p. (d) Obtain the sequential time (assume that a call to functions `generate` and `fabs` costs 1 flop) and the parallel time for each of the two versions assuming that there are 3 processors. Compute the speed-up in each case.

Question 3 (0.7 points)

Parallelize by means of OpenMP the following fragment of code, where `f` and `g` are two functions that take 3 arguments of type `double` and return a `double`, and `fabs` is the standard function that returns the absolute value of a `double`.

```
double x,y,z,w=0.0;
double x0=1.0,y0=3.0,z0=2.0;     /* starting point */
double dx=0.01,dy=0.01,dz=0.01;  /* increments */

x=x0;y=y0;z=z0;    /* search in x */
while (fabs(f(x,y,z))<fabs(g(x0,y0,z0))) x += dx;
w += (x-x0);

x=x0;y=y0;z=z0;    /* search in y */
while (fabs(f(x,y,z))<fabs(g(x0,y0,z0))) y += dy;
w += (y-y0);

x=x0;y=y0;z=z0;    /* search in z */
while (fabs(f(x,y,z))<fabs(g(x0,y0,z0))) z += dz;
w += (z-z0);

printf("w = %g\n",w);
```

Question 4 (1 point)

Given the following function, where we suppose that the functions `T1`, `T3` and `T4` have a cost of n and the functions `T2` and `T5` of $2n$, being n a constant value.

```
double example(int i,int j)
{
  double a,b,c,d,e;
  a = T1(i);
  b = T2(j);
  c = T3(a+b,i);
  d = T4(a/c);
  e = T5(b/c);
  return d+e;   /* T6 */
}
```

0.2 p. (a) Draw the dependency graph and compute the sequential cost.

0.6 p. (b) Parallelize it using MPI with two processes. Both processes invoke the function with the same value of the arguments `i`, `j` (it is not necessary to communicate them). The return value of the function must be correct in process 0 (it is not necessary that it is available in both processes).

0.2 p. (c) Compute the parallel execution time (computation and communication) and the speedup with two processes.

Question 5 (1 point)

Given the following code fragment of a parallel program:

```
int j, proc;
double A[NROW][NCOL], col[NROW];

MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &p);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
...
```

Write the code necessary to define an MPI datatype that allows any column of `A` being sent or received with a single message. Use the datatype so that process 0 sends column `j` to process `proc`, who will receive it into array `col`, change the sign of its elements, and return it to process 0, who will receive it again into column `j` of matrix `A`.

Question 6 (0.4 points)

The following fragment of code uses point-to-point communication primitives for a communication pattern that can be realized by means of a single collective operation.

```
#define TAG 999
int i, k, sz, rank;
float z[LNZ], zfull[LNFULL];     /* we suppose LNFULL>=LNZ*sz */
...
MPI_Comm_size(comm, &sz);
MPI_Comm_rank(comm, &rank);
if (!rank) {
  for (i=0;i<LNZ;i++) zfull[i] = z[i];
  for (k=1;k<sz;k++) {
    MPI_Recv(&zfull[k*LNZ], LNZ, MPI_FLOAT, k, TAG, comm,
             MPI_STATUS_IGNORE);
  }
} else {
  MPI_Send(z, LNZ, MPI_FLOAT, 0, TAG, comm);
}
```

Write the call to the MPI primitive of the equivalent collective communication operation.

Question 7 (0.6 points)

Given the following call to a collective communication primitive:

```
long int factor=..., product;
MPI_Allreduce(&factor, &product, 1, MPI_LONG, MPI_PROD, comm);
```

Write an equivalent fragment of code (must perform the same communication and associated arithmetic operations), but using only point-to-point communication primitives.

Parallel Computing
Degree in Computer Science Engineering (ETSInf)
Year 2015-16 ◇ Partial exam 9/11/2015 ◇ Duration: 1h 30m

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Question 1 (1.2 points)
Given the following function:

```
#define N 6000
#define STEPS 6

double function1(double A[N][N], double b[N], double x[N])
{
  int i, j, k, n=N, steps=STEPS;
  double max=-1.0e308, q, s, x2[N];
  for (k=0;k<steps;k++) {
    q=1;
    for (i=0;i<n;i++) {
      s = b[i];
      for (j=0;j<n;j++)
        s -= A[i][j]*x[j];
      x2[i] = s;
      q *= s;
    }
    for (i=0;i<n;i++)
      x[i] = x2[i];
    if (max<q)
      max = q;
  }
  return max;
}
```

0.7 p. (a) Parallelize the code using OpenMP. Explain why you do it that way. Those solutions that take into account efficiency will get a higher mark.

0.25 p. (b) Indicate the theoretical cost (in flops) of an iteration of the `k` loop in the sequential code.

0.25 p. (c) Considering a single iteration of the `k` loop (`STEPS`=1), indicate the speedup and efficiency that can be attained with p threads, assuming that there are as many cores/processors as threads and that `N` is an exact multiple of p.

Question 2 (1 point)
The following function processes a series of bank transfers. Each transfer has an origin account, a destination account, and an amount of money that is moved from the origin account to the destination account. The function updates the amount of money in each account (`balance` array) and also returns the maximum amount that is transferred in a single operation.

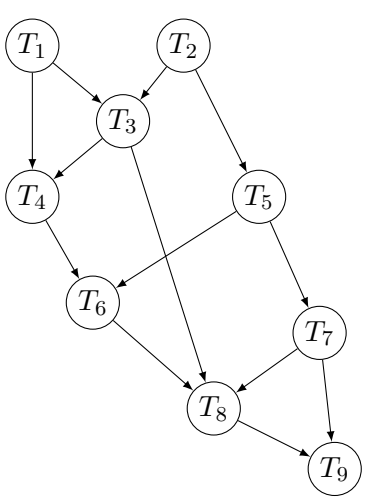
```
double transfers(double balance[], int origins[], int destinations[],
       double quantities[], int n)
{
   int i, i1, i2;
   double money, maxtransf=0;

   for (i=0; i<n; i++) {
      /* Process transfer i: The transferred quantity is quantities[i],
       * that is moved from account origins[i] to account destinations[i].
       * Balances of both accounts are updated and the maximum quantity */
      i1 = origins[i];
      i2 = destinations[i];
      money = quantities[i];
      balance[i1] -= money;
      balance[i2] += money;
      if (money>maxtransf) maxtransf = money;
   }
   return maxtransf;
}
```

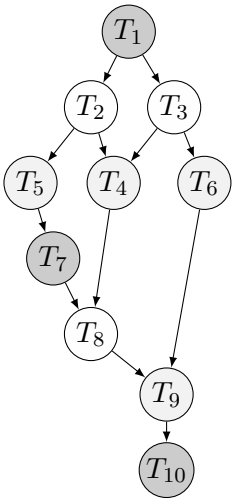
0.7 p. (a) Parallelize the function in an efficient way by means of OpenMP.

0.3 p. (b) Modify the previous solution so that the index of the transfer with more money is printed.

Question 3 (0.8 points)

Given the following two task dependency graphs:





T_1, T_7, T_{10}	$\frac{1}{3}n^3$
T_2, T_3, T_8	n^3
T_4, T_5, T_6, T_9	$2n^3$

0.4 p. (a) For the left graph, indicate which sequence of graph nodes constitute the critical path. Compute the length of the critical path and the average concurrency degree. Note: no cost information is provided, one can assume that all tasks have the same cost.

0.4 p. (b) Repeat the same for the graph on the right. Note: in this case the cost of each task is given in flops (for a problem size n) according to the table shown.

Parallel Computing
Degree in Computer Science Engineering (ETSInf)
Year 2015-16 ◇ Partial exam 15/1/2016 ◇ Duration: 1h 30m

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Question 1 (1 point)

We want to implement a function to distribute a square matrix across the processes of an MPI program, with the following header:

```
void communicate(double A[N][N], double Aloc[][N], int proc_row[N], int root)
```

The matrix `A` is stored initially in process `root`, and must be distributed by rows across the processes, in a such way that each row `i` must go to process `proc_row[i]`. The content of array `proc_row` is valid in all processes. Each process (including the `root`) must store the rows that have been assigned to it in the local matrix `Aloc`, occupying the first rows (that is, if a given process is assigned `k` rows, these must be stored in the first `k` rows of `Aloc`).

Example for 3 processes:

`A`

11	12	13	14	15
21	22	23	24	25
31	32	33	34	35
41	42	43	44	45
51	52	53	54	55

`proc_row`

0
2
0
1
1

`Aloc` in P_0

11	12	13	14	15
31	32	33	34	35

`Aloc` in P_1

41	42	43	44	45
51	52	53	54	55

`Aloc` in P_2

21	22	23	24	25

0.8 p. (a) Write the code of the function.

0.2 p. (b) In a general case, would it be possible to use MPI's *vector* data type (`MPI_Type_vector`) to send all rows assigned to a given process by means of a single message? If it is possible, write the instructions to define it. Otherwise, justify why.

Question 2 (1 point)

The following function computes the scalar product of two vectors:

```
double scalarprod(double X[], double Y[], int n) {
  double prod=0.0;
  int i;
  for (i=0;i<n;i++)
    prod += X[i]*Y[i];
  return prod;
}
```

0.5 p. (a) Implement a function to perform the scalar product in parallel by means of MPI, using collective operations whenever possible. The data are supposed to be available in process P_0 and the result must also be left in P_0 (the function's return value need only be correct in P_0). It is allowed to assume that the problem size `n` is exactly divisible by the number of processes.

Note: we next show the header of the function to be implemented, including the declaration of the local vectors (assume that `MAX` is sufficiently large for any value of n and the number of processes).

```
double pscalarprod(double X[], double Y[], int n)
{
  double Xlcl[MAX], Ylcl[MAX];
```

0.3 p. (b) Compute the speed-up. If for a sufficiently large message size, the sending time per element was equivalent to 0.1 flops, which would be the maximum speed-up that could be attained when the problem size tends to infinity and for a sufficiently large number of processes?

0.2 p. (c) Modify the previous code so that the return value is correct in all processes.

Question 3 (1 point)

We want to distribute across 4 processes a square matrix of order $2N$ ($2N$ rows by $2N$ columns) defined by blocks as

$$A = \begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix},$$

where each block A_{ij} corresponds to a square matrix of order N, in such a way that we want process P_0 to store locally matrix A_{00}, P_1 matrix A_{01}, P_2 matrix A_{10} and P_3 matrix A_{11}.

For example, the following matrix with $N = 2$ would be distributed as shown:

$$A = \left(\begin{array}{cc|cc} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ \hline 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{array}\right) \qquad \begin{array}{ll} \text{In } P_0\text{:} \begin{pmatrix} 1 & 2 \\ 5 & 6 \end{pmatrix} & \text{In } P_1\text{:} \begin{pmatrix} 3 & 4 \\ 7 & 8 \end{pmatrix} \\ \text{In } P_2\text{:} \begin{pmatrix} 9 & 10 \\ 13 & 14 \end{pmatrix} & \text{In } P_3\text{:} \begin{pmatrix} 11 & 12 \\ 15 & 16 \end{pmatrix} \end{array}$$

0.8 p. (a) Implement a function that performs the described distribution, by defining the necessary MPI data type. The header of the function would be:

```
void communicate(double A[2*N][2*N], double B[N][N])
```

where `A` is the initial matrix, stored in process 0, and `B` is the local matrix where each process must store the block of A assigned to it.

Note: it is allowed to assume that the number of processes in the communicator is 4.

0.2 p. (b) Compute the communication time.

Parallel Computing
Degree in Computer Science Engineering (ETSInf)
Year 2015-16 ◇ Final exam 26/1/16 ◇ Block OpenMP ◇ Duration: 1h 30m

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Question 1 (0.8 points)
Given the following function:

```
double fn1(double A[M][N], double b[N], double c[M], double z[N])
{
   double s, s2=0.0, elem;
   int i, j;

   for (i=0; i<M; i++) {
      s = 0.0;
      for (j=0; j<N; j++)
         s = s + A[i][j]*b[j];
      elem = s*s;
      if (elem>c[i])
         c[i] = elem;
      s2 = s2+s;
   }

   for (i=0; i<N; i++)
      z[i] = 2.0/3.0*b[i];

   return s2;
}
```

0.4 p. (a) Parallelize it with OpenMP in an efficient way. If possible, use a single parallel region.

0.2 p. (b) Modify the code so that each thread prints a line with its identifier and the number of iterations of the first `i` loop that it has performed.

0.2 p. (c) In the first parallelized loop, justify if one can expect differences in performance among the following schedule policies for the loop: `schedule(static)`, `schedule(static,1)`, `schedule(dynamic)`.

Question 2 (1.2 points)
Given the following function:

```
double f(int n, double vec[])
{
  double res, v[NMAX], w[NMAX];
  A(n,v,vec);          /* Copy vec in v, cost n */
  B(n,w,vec);          /* Copy vec in w, cost n */
  C(n,vec);            /* Update vec, cost n    */
```

```
  D(n,vec);            /* Update vec, cost n    */
  E(n,v);              /* Update v, cost 3n     */
  F(n,w);              /* Update w, cost 2n     */
  res = G(n,vec,v,w);  /* Compute res, cost 3n  */
  return res;
}
```

0.5 p. (a) Draw the dependency graph, indicating the maximum concurrency degree, the length of the critical path and the average concurrency degree.

0.5 p. (b) Parallelize it with OpenMP.

0.2 p. (c) Compute the maximum speedup and efficiency if the code is run with 2 threads.

Question 3 (1 point)

Given the following function:

```
double function(double A[N][N],double B[N][N])
{
  int i,j;
  double aux, maxi;
  for (i=1; i<N; i++) {
    for (j=0; j<N; j++) {
      A[i][j] = 2.0+A[i-1][j];
    }
  }
  for (i=0; i<N-1; i++) {
    for (j=0; j<N-1; j++) {
      B[i][j] = A[i+1][j]*A[i][j+1];
    }
  }
  maxi = 0.0;
  for (i=0; i<N; i++) {
    for (j=0; j<N; j++) {
      aux = B[i][j]*B[i][j];
      if (aux>maxi) maxi = aux;
    }
  }
  return maxi;
}
```

0.8 p. (a) Parallelize the previous code with OpenMP. Explain the decisions that you take. A higher consideration will be given to those solutions that are more efficient.

0.2 p. (b) Compute the sequential cost, the parallel cost, the speedup and the efficiency that could be obtained with p processors assuming that N is divisible by p.

Question 4 (1 point)

Given the sequential code:

```
int i, j;
double A[N][N];
for (i=0;i<N;i++)
    for(j=0;j<N;j++)
        A[i][j]= A[i][j]*A[i][j];
```

0.8 p. (a) Implement an equivalent parallel version using MPI, taking into account the following aspects:

- Process P_0 initially obtains matrix `A`, performing a call `read(A)`, where `read` is a function already implemented.
- Matrix `A` must be distributed by blocks of rows among all processes.
- Finally P_0 must contain the resulting matrix `A`.
- Use collective communication whenever possible.

We assume that N is divisible by the number of processes and that the declaration of the used matrices is

```
double A[N][N], B[N][N];  /* B: distributed matrix */
```

0.2 p. (b) Compute the speedup and efficiency.

Question 5 (0.8 points)

Develop a function that can be used to send a submatrix from process 0 to process 1, where it will be stored as a vector. A new data type must be employed, so that a single message is sent. Remember that matrices in C are stored in memory by rows.

The header of the function will be:

```
void send(int m,int n,double A[M][N],double v[MAX],MPI_Comm comm)
```

Note: we can assume that `m*n<=MAX` and that the submatrix to be sent starts at element `A[0][0]`.

Example with $M = 4$, $N = 5$, $m = 3$, $n = 2$:

`A` (in P_0)						`v` (in P_1)
1	2	0	0	0		
3	4	0	0	0	→	1 2 3 4 5 6
5	6	0	0	0		
0	0	0	0	0		

Question 6 (1.2 points)

Given the following sequential program:

```
int compute_value(int num);  /* given function, defined elsewhere */

int main(int argc,char *argv[])
{
  int n,i,val,min;

  printf("Number of iterations: ");
  scanf("%d",&n);
  min = 100000;
  for (i = 1; i <= n; i++) {
    val = compute_value(i);
    if (val < min) min = val;
  }
  printf("Minimum: %d\n",min);
  return 0;
}
```

0.8 p. (a) Parallelize it by means of MPI using point to point communication operations. The data input and output must be done only by process 0. One can assume that n is an exact multiple of the number of processes.

0.4 p. (b) What should be changed in order to use collective communication operations where possible?

Parallel Computing
Degree in Computer Science Engineering (ETSInf)
Year 2016-17 ◇ Partial exam 9/11/2016 ◇ Duration: 1h 30m

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Question 1 (1 point)
Given the following function:

```
double calcula()
{
  double A[N][N],B[N][N],a,b,x,y,z;

  rellena(A,B);              /* T1 */
  a = calculos(A);           /* T2 */
  b = calculos(B);           /* T3 */
  x = suma_menores(B,a);     /* T4 */
  y = suma_en_rango(B,a,b); /* T5 */
  z = x + y;                 /* T6 */
  return z;
}
```

Function `rellena` receives as input two matrices and fills them up with values generated internally. All the arguments in the rest of the functions are input parameters and are not modified. Functions `rellena` and `suma_en_rango` have a cost of $2n^2$ flops each ($n = \mathtt{N}$), whereas the cost for the rest of the functions is of n^2 flops.

0.3 p. (a) Draw the dependency graph and compute its maximum degree of concurrency, the critical path and its length and the average concurrency degree.

0.4 p. (b) Parallelise the function using OpenMP.

0.3 p. (c) Compute the sequential execution time, the parallel execution time, the speed-up and the efficiency of the previous code, assuming that we have 3 threads.

Question 2 (0.9 points)
Given the following function:

```
void updatemat(double A[N][N])
{
  int i,j;
  double s[N];
  for (i=0; i<N; i++) {     /* sum by rows */
    s[i] = 0.0;
    for (j=0; j<N; j++)
      s[i] += A[i][j];
  }
  for (i=1; i<N; i++)       /* prefix sum  */
    s[i] += s[i-1];
```

```
  for (j=0; j<N; j++) {     /* column scaling */
    for (i=0; i<N; i++)
      A[i][j] *= s[j];
  }
}
```

0.15 p. (a) Compute the theoretical cost (in flops) of the above function.

0.5 p. (b) Parallelise it with OpenMP using a single parallel region.

0.25 p. (c) Compute the speed-up that can be obtained using p processors and assuming that N is an exact multiple of p.

Question 3 (1.1 points)

The next function computes the points that will be obtained by the different teams in a soccer league.

The season has celebrated `NJ` fixtures (that is, league weeks), each one involving `NPJ` matches. The information from all the matches is stored in a matrix `partidos`. Each element of this matrix is a `SPartido` data structure that contains the data for the match (which teams have played and how many goals each one scored).

With this information, the function computes the points (`puntos`) that each one of the `NE` teams sum up after each match. If a team wins a match, it gets 3 points and 0 points if it loses it. If the result of the match is even, both teams get 1 point.

The function also returns the number of goals of the season and eventually it gets for each fixture, the maximum number of goals in a match (array `maxg_jornada`).

```
int func(SPartido partidos[NJ][NPJ], int puntos[NE], int maxg_jornada[NJ]) {
   int i, j, maxg, eq1, eq2, g1, g2, goles_temporada;
   goles_temporada = 0;
   for (i=0; i<NJ; i++) {      /* for each fixture */
      maxg = 0;
      for (j=0; j<NPJ; j++) {  /* for each match in a fixture */
         eq1 = partidos[i][j].eq1;
         eq2 = partidos[i][j].eq2;
         g1 = partidos[i][j].goles1;
         g2 = partidos[i][j].goles2;
         if (g1>g2)
            puntos[eq1] += 3;  /* Team 1 has won */
         else if (g1<g2)
            puntos[eq2] += 3;  /* Team 2 has won */
         else {
            puntos[eq1] += 1;  /* Even */
            puntos[eq2] += 1;
         }
         if (g1+g2>maxg) maxg = g1+g2;
         goles_temporada += g1+g2;
      }
      maxg_jornada[i] = maxg;
   }
   return goles_temporada;
}
```

0.6 p. (a) Parallelise efficiently loop `i`.

0.5 p. (b) Parallelise efficiently loop `j`. Bear in mind that a team only plays in one match per fixture. Indicate clearly the changes with respect to the original code (do not use the code from the previous sub-question).

Parallel Computing
Degree in Computer Science Engineering (ETSINF)
Year 2016-17 ◇ Partial exam 16/1/2017 ◇ Duration: 1h 30m

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Question 1 (1 point)

We want to distribute, using MPI, the square sub-matrix blocks in the diagonal of a square matrix of dimension $3 \cdot \text{DIM}$ among 3 processes. For example, if the matrix is of dimension 6 (DIM=2), the distribution will be as follows:

$$\begin{pmatrix} a_{00} & a_{01} & \dots & \dots & \dots & \dots \\ a_{10} & a_{11} & \dots & \dots & \dots & \dots \\ \dots & \dots & a_{22} & a_{23} & \dots & \dots \\ \dots & \dots & a_{32} & a_{33} & \dots & \dots \\ \dots & \dots & \dots & \dots & a_{44} & a_{45} \\ \dots & \dots & \dots & \dots & a_{54} & a_{55} \end{pmatrix} \rightarrow \begin{array}{c} P_0 \\ \\ P_1 \\ \\ P_2 \end{array} \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} \begin{bmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} a_{44} & a_{45} \\ a_{54} & a_{55} \end{bmatrix}$$

Implement a parallel function that sends the square blocks of the matrix with the minimum number of messages. We provide the header of the function to ease its implementation. Process 0 has the full matrix in `A` and after the call, every process must have its corresponding square block in `Alcl`. Use point-to-point communication primitives.

```
void SendBAD(double A[3*DIM][3*DIM], double Alcl[DIM][DIM]) {
```

Question 2 (1 point)

Next program reads a square matrix A of dimension N and computes a vector v of N elements. Element i in this vector contains the sum of all the elements of the $i-th$ row in A. Then, the program prints vector v.

```
int main(int argc, char *argv[])
{
  int i,j;
  double A[N][N],v[N];
  read_mat(A);
  for (i=0;i<N;i++) {
    v[i] = 0.0;
    for (j=0;j<N;j++)
      v[i] += A[i][j];
  }
  write_vec(v);
  return 0;
}
```

0.75 p. (a) Implement an MPI parallel program, using collective communication primitives whenever possible, that will perform the same computations as the sequential code. Take into account the next points:

- Process P_0 reads matrix A.
- P_0 distributes matrix A among all the processes.
- Each process computes its local part of v.
- P_0 composes vector v by collecting the local parts from each process.
- P_0 writes vector v.

N.B.: You can assume that N is an exact multiple of the number of processes.

0.25 p. (b) Obtain the sequential and parallel time, ignoring the cost of the read and write functions. Indicate separately the cost of each one of the collective operations.

Question 3 (1 point)

Function `example` executes several tasks (`T1`–`T6`). The cost of the functions `T1`, `T2` and `T3` is $7n$, and the cost of the functions `T5` and `T6` is n, being n a constant value.

```
double example(int val[3])
{
  double a,b,c,d,e,f;
  a = T1(val[0]);
  b = T2(val[1]);
  c = T3(val[2]);
  d = a+b+c;       /* T4 */
  e = T5(val[2],d);
  f = T6(val[0],val[1],e);
  return f;
}
```

0.25 p. (a) Draw the dependency graph for the function and compute the sequential cost.

0.5 p. (b) Implement a parallel version using MPI, assuming that there are three processes. All the processes will call the function with the same value for `val` (you do not need to communicate it). The return value only needs to be correct in process 0 (the value returned in the rest of the processes is irrelevant).

N.B.: Only use collective communication primitives.

0.25 p. (c) Compute the parallel execution time (arithmetic and communications) and the Speed-Up with three processes. Compute also the asymptotic Speed-Up (that is, the limit when n tends to infinity).

Parallel Computing
Degree in Computer Science Engineering (ETSINF)
Year 2016-17 ◇ Final exam 25/1/17 ◇ Block OpenMP ◇ Duration: 1h 30m

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Question 1 (0.9 points)

```
double func(double x[], double A[N][N])
{
  int i, j, hit;
  double elem, val=0;
  for (i=0; i<N; i++) {
    elem = x[i];
    hit = 0;
    j = N-1;
    while (j>=0 && !hit) {
      if (elem<sqrt(A[i][j])) hit = 1;
      j--;
    }
    if (hit) {
      elem *= 2.0;
      x[i] /= 2.0;
      val += elem;
    }
  }
  return val;
}
```

0.4 p. (a) Parallelise loop `i` from the previous function efficiently, using OpenMP.

0.5 p. (b) Paralleise loop `j` from the previous function efficiently, using OpenMP.

Question 2 (1.1 points)

Next function computes all the row and column positions where the maximum value in a matrix appears.

```
int funcion(double A[N][N],double positions[][2])
{
  int i,j,k=0;
  double maximum;
  /* Compute maximum */
  maximum = A[0][0];
  for (i=0;i<N;i++) {
    for (j=0;j<N;j++) {
      if (A[i][j]>maximum) maximum = A[i][j];
```

```
    }
  }
  /* Once calculated, we seek for their occurrences */
  for (i=0;i<N;i++) {
    for (j=0;j<N;j++) {
      if (A[i][j] == maximum) {
        positions[k][0] = i;
        positions[k][1] = j;
        k = k+1;
      }
    }
  }
  return k;
}
```

0.6 p. (a) Parallelise this function efficiently using OpenMP, using a single parallel region.

0.5 p. (b) Modify the previous parallel code so that each thread prints on the screen its identifier and the amount of maximum values it found.

Question 3 (1 point)

We want to parallelise the next code for processing images, which receives as input 4 similar images (e.g. video frames `f1`, `f2`, `f3`, `f4`) and returns two resulting images (`r1`, `r2`). The pixels in the images are represented as floating point values. Type `image` is a type defined that consists on a matrix of `N`×`M` `doubles`.

```
typedef double image[N][M];

void procesa(image f1,image f2,image f3,image f4,image r1,image r2)
{
  image d1,d2,d3;
  difer(f2,f1,d1);          /* Task 1 */
  difer(f3,f2,d2);          /* Task 2 */
  difer(f4,f3,d3);          /* Task 3 */
  sum(d1,d2,d3,r1);         /* Task 4 */
  difer(f4,f1,r2);          /* Task 5 */
}
```

```
void difer(image a,image b,image d)
{
  int i,j;
  for (i=0;i<N;i++)
    for (j=0;j<M;j++)
      d[i][j] = fabs(a[i][j]-b[i][j]);
}
```

```
void sum(image a,image b,image c,image s)
{
  int i,j;
  for (i=0;i<N;i++)
    for (j=0;j<M;j++)
      s[i][j] = a[i][j]+b[i][j]+c[i][j];
}
```

0.5 p. (a) Draw the dependency graph of tasks, and compute the maximum and average concurrency degree, considering the actual cost in flops (assuming that `fabs` does not cost any flop).

0.5 p. (b) Parallelise the function `procesa` using OpenMP, without changing neither `difer` nor `sum`.

Question 4 (0.9 points)

Given a matrix with `NF` rows and `NC` columns, initially stored in process 0, we want to distribute it by blocks of columms between processes 0 and 1. Process 0 will keep the first half of the columns and process 1 will get the second half (we will assume that `NC` is even).

Implement a function, using the header provided, that will implement this distribution using MPI. You should define the data type required to ensure that the elements that belong to process 1 are sent with a single message. When the function finishes, both processes must have in `Aloc` its corresponding block of columns. The number of processes could be larger than 2, and then, only processes 0 and 1 will store its column block in `Aloc`.

```
void distribute(double A[NF][NC], double Aloc[NF][NC/2])
```

Question 5 (1.1 points)

Given the next sequential function:

```
int count(double v[], int n)
{
  int i, cont=0;
  double mean=0;

  for (i=0;i<n;i++)
    mean += v[i];
  mean = mean/n;

  for (i=0;i<n;i++)
    if (v[i]>mean/2.0 && v[i]<mean*2.0)
      cont++;

  return cont;
}
```

0.7 p. (a) Implement a parallel version using MPI, assuming that vector `v` is initially only in process 0, and that the result returned by the function needs to be correct only in process 0. You should distribute the data for achieving a balanced distribution of the processing. N.B.: You can assume that `n` is an exact multiple of the number of processes.

0.4 p. (b) Compute the execution time of the parallel version in the previous part, as well as the asympthotic value of the Speed-up when `n` tends to infinite. If you have used collective operations, indicate the cost that you have considered for each one of them.

Question 6 (1 point)

Implement a *ping-pong* program.

The *ping-pong* parallel program will be executed by 2 processes, repeating 200 times sending one message with 100 integer values from process 0 to process 1 and then receiving the same message from process 1 in process 0. The program must print on the screen the average time for sending one integer value, obtained from the total time required for sending and receiving those messages.

The program can start like this:

```
int main(int argc,char *argv[])
{
  int v[100];
```

0.8 p. (a) Implement the parallel *ping-pong* program.

0.2 p. (b) Compute the theoretical communication cost for the program.

Parallel Computing
Degree in Computer Science Engineering (ETSInf)
Year 2017-18 ◇ Partial exam 8/11/2017 ◇ Duration: 1h 30m

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Question 1 (1 point)

The next function computes matrix A from a node array (array `node`). A coefficient is computed for each node. This coefficient is used to modify three elements from A (two of them located in the diagonal, and the other one located out of the diagonal)

```
double function(double A[DIM][DIM], NodeData node[], int n)
{
  int i,f,c;
  double coef, sum;
  preparar(A);
  sum = 0;
  for (i=0; i<n; i++) {
    f = node[i].fila;
    c = node[i].columna;
    coef = compute_coef(node[i]);
    /* Update elements from the diagonal of A */
    A[f][f] += coef;
    A[c][c] += coef;
    /* Update element located out of the diagonal of A */
    A[f][c] -= coef;
    sum += coef;
  }
  return sum;
}
```

0.8 p. (a) Implement an efficient parallel version using OpenMP. We know that two nodes may update the same diagonal element, but they will not modify the same element located out of the diagonal. Briefly justify your approach.

0.2 p. (b) Obtain the a-priori sequential execution time, assuming that function `preparar` has a cost of n flops and `compute_coef` has a cost of 5 flops. Compute the parallel execution time and the Speed-up. Compute the maximum speed-up if we used an infinite number of processors.

Question 2 (0.8 points)

Given the following code:

```
int funcion(double A[M][N])
{
  int i,j,answer=1;
  double row_sum,diag,elem;
  i = 0;
```

```
  while (i<M && answer==1) {
    row_sum = 0;
    diag = fabs(A[i][i]);
    for (j=0;j<N;j++) {
      if (i!=j) {
        elem = fabs(A[i][j]);
        row_sum += elem;
      }
    }
    if (diag<row_sum) answer = 0;
    i++;
  }
  return answer;
}
```

0.5 p. (a) Describe how the outermost loop (the `while` loop) can be efficiently parallelized.

0.3 p. (b) Parallelize the second loop (`for` loop).

Question 3 (1.2 points)

In the next function (`matrix_computations`), no function call (`A`,`B`,`C`,`D`) changes any of its parameters:

```
double matrix_computations(double mat[n][n])
{
  double x,y,z,aux,total;
  x = A(mat);          /* task A, cost: 3 n^2         */
  aux = B(mat);        /* task B, cost: n^2           */
  y = C(mat,aux);      /* task C, cost: n^2           */
  z = D(mat);          /* task D, cost: 2 n^2         */
  total = x + y + z;   /* task E  (compute the cost)  */
  return total;
}
```

0.3 p. (a) Draw its dependency graph and compute the maximum concurrency degree, the length of the critical path (indicate a critical path) and the average concurrency degree.

0.4 p. (b) Implement a parallel version using OpenMP.

0.3 p. (c) Obtain the sequential time in flops. Assuming that the parallel version is executed with 2 threads, obtain the parallel time, the speed-up and the efficiency, in the best case.

0.2 p. (d) Modify the parallel cost to print on the screen (only once) the number of threads used and the execution time in seconds.

Parallel Computing
Degree in Computer Science Engineering (ETSInf)
Year 2017-18 ◇ Partial exam 15/1/2018 ◇ Duration: 1h 30m

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Question 1 (1 point)

Given the following function:

```
double computing(double v[n])
{
  double x,y,a,b,c;
  x = f1(v);        /* task 1, cost n     */
  y = f2(v);        /* task 2, cost n     */
  a = f3(v,x+y);    /* task 3, cost 2n+1  */
  b = f4(v,x-y);    /* task 4, cost 3n+1  */
  c = a + b;        /* task 5              */
  return c;
}
```

0.1 p. (a) Considering that no function call modifies vector v, draw the dependency graph.

0.7 p. (b) Implement a parallel version on MPI using only point-to-point communication operations. Vector v is already replicated in all the processes. The result is only needed at process 0. Ensure that the parallel version cannot suffer from deadlocks.

0.2 p. (c) Obtain the theoretical computational cost of the parallel algorithm implemented, including also the communication cost.

Question 2 (1.1 points)

Next piece of code implements the operation $C = aA + bB$, where A, B and C are matrices of size $M \times N$ and a and b are real numbers.

```
int main(int argc, char *argv[]) {
  int i, j;
  double a, b, A[M][N], B[M][N], C[M][N];
  ReadOperands(A, B, &a, &b);
  for (i=0; i<M; i++) {
    for (j=0; j<N; j++) {
      C[i][j] = a*A[i][j] + b*B[i][j];
    }
  }
  WriteMatrix(C);
  return 0;
}
```

Write a parallel version using MPI and using collective operations, assuming that:

- P_0 will get matrices A and B, as well as real numbers a and b, by calling function `ReadOperands`.
- Only P_0 should have the whole resulting matrix C, and it will be the process that calls function `WriteMatrix`.
- M is an exact multiple of the number of processes.
- Matrices A and B must be distributed cyclically by rows among the processes involved, which will execute in parallel the operation.

Question 3 (0.9 points)

We want to implement a communication operation among three MPI processes of a matrix `A` of size $N \times N$, stored at P_0. In this operation Process P_1 will receive the submatrix composed of the rows with even index and process P_2 will receive the submatrix composed of the rows with odd index. You must use MPI Derived types to minimize the number of messages. Each submatrix received at P_1 and P_2 must be stored in an $N/2 \times N$ local matrix `B`. N.B.: You can assume that N is an even number.

For example: If the matrix stored in P_0 is

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

The submatrix in P_1 must be:

$$B = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 9 & 10 & 11 & 12 \end{pmatrix}$$

And the submatrix in P_2 must be:

$$B = \begin{pmatrix} 5 & 6 & 7 & 8 \\ 13 & 14 & 15 & 16 \end{pmatrix}.$$

0.7 p. (a) Implement a parallel MPI function for the above operation, using the following header:

```
void communicate(double A[N][N], double B[N/2][N])
```

0.2 p. (b) Compute the communication time of the implemented function.

Parallel Computing
Degree in Computer Science Engineering (ETSInf)
Year 2017-18 ◇ Final exam 25/1/18 ◇ Block OpenMP ◇ Duration: 1h 30m

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Question 1 (1.1 points)

We want to implement a parallel version of the next function, where `read_data` updates its three arguments and `f5` reads and writes its two first arguments. The rest of the functions do not modify any of their arguments.

```
void function() {
  double x,y,z,a,b,c,d,e;
  int n;
  n = read_data(&x,&y,&z);     /* Task 1 (n flops)    */
  a = f2(x,n);                 /* Task 2 (2n flops)   */
  b = f3(y,n);                 /* Task 3 (2n flops)   */
  c = f4(z,a,n);               /* Task 4 (n^2 flops)  */
  d = f5(&x,&y,n);             /* Task 5 (3n^2 flops) */
  e = f6(z,b,n);               /* Task 6 (n^2 flops)  */
  write_results(c,d,e);        /* Task 7 (n flops)    */
}
```

0.2 p. (a) Draw the dependency graph for the different tasks involved in the function.

0.5 p. (b) Parallelise the function efficiently using OpenMP.

0.2 p. (c) Compute speed-up and efficiency if 3 processing units are used.

0.2 p. (d) Paying attention to the costs of each task (see the comments in the code of the function), obtain the critical path and the average concurrency degree.

Question 2 (0.7 points)

Given the next code:

```
int i,j,k;
double a,b,aux,A[N][N],B[N][N],C[N][N];
scanf("%lf",&a);
scanf("%lf",&b);
generate1(A,a);    /* Cost N^3   */
generate2(B,b);    /* Cost 2*N^3 */
for (i=0; i<N; i++) {
   for (j=0; j<N; j++) {
      aux = 0.0;
      for (k=0; k<N; k++)
         aux += A[i][k]*B[k][j];
      C[i][j] = aux;
   }
}
```

where `generate1` and `generate2` are two functions previously declared and implemented.

Efficiently parallelize the above code using OpenMP, using only one parallel region.

Question 3 (1.2 points)

Given the next function:

```
double function(double A[M][N], double maxim, double pf[])
{
   int i,j,j2;
   double a,x,y;
   x = 0;
   for (i=0; i<M; i++) {
      y = 1;
      for (j=0; j<N; j++) {
         a = A[i][j];
         if (a>maxim) a = 0;
         x += a;
      }
      for (j2=1; j2<i; j2++) {
         y *= A[i][j2-1]-A[i][j2];
      }
      pf[i] = y;
   }
   return x;
}
```

0.2 p. (a) Implement a parallel version based on the parallelisation of loop `i` using OpenMP.

0.5 p. (b) Implement another parallel version by parallelising the iterations of loops `j` and `j2` (efficiently and for any number of threads).

0.2 p. (c) Compute the sequential cost:

0.3 p. (d) For each one of the parallelised loops, analyse if you may expect differences in performance due to different scheduling policies. If so, write down the best scheduling for such loop.

Question 4 (1 point)

A parallel program has already distributed a vector among the processes, using a block-oriented distribution, so each process stores its block in an array called `vloc`.

Implement a parallel function that will shift the elements of the vector one position to the right. The last element of the vector will be placed in the first position of the vector. For example, if we had 3 processes and the initial status is:

	P_0	P_1	P_2
`vloc`	`[2 5 3]`	`[7 1 0]`	`[6 4 9]`

The final status will be:

	P_0	P_1	P_2
`vloc`	`[9 2 5]`	`[3 7 1]`	`[0 6 4]`

The function will ensure that no deadlocks may happen. The header of the function will be:

```
void shift(double vloc[], int mb)
```

Where `mb` is the number of elements of local vector `vloc` (we will assume `mb` > 1).

Question 5 (0.8 points)

Implement a function in C to send the three main diagonals of a square matrix to all the processes. You must consider neither the first nor the final rows of the matrix. For example, the elements that must be considered for a matrix of size 6 are the ones marked with `x`:

```
+ + + + + +
x x x + + +
+ x x x + +
+ + x x x +
+ + + x x x
+ + + + + +
```

The function must define a new MPI datatype that could be used to send the whole tridiagonal block in a single message. Bear in mind that matrices in C are stored in memory by rows. Use the following header for the function:

```
void send_tridiagonal(double A[N][N],int root,MPI_Comm comm)
```

where

- `N` is the number of rows and columns of the matrix.
- `A` is the matrix with the data to be sent (in the process that sends the data) and the matrix where the data must be received (in the rest of the processes).

- Parameter `root` indicates the process that initially has the data to be sent in matrix `A`.
- `comm` is the communicator for all the processes that will have the tridiagonal part of `A` in their memories.

For example, if the function is called as:

```
send_tridiagonal(A,5,comm);
```

Process 5 will be the one that has the valid data in `A` when the function is called, and at the return of the call, all processes in communicator `comm` will have the tridiagonals (except first and last rows) in `A`.

Question 6 (1.2 points)

The following program counts the number of elements of a matrix larger than their mean.

```
int AboveMean(double A[N][N], double mean, int n, int m) {
  int i,j,nc=0;
  for (i=0;i<n;i++)
    for (j=0;j<m;j++)
      if (A[i][j]>mean) nc++;
  return nc;
}
double Sum(double A[N][N], int n, int m) {
  int i,j;
  double sum=0.0;
  for (i=0;i<n;i++)
    for (j=0;j<m;j++)
      sum += A[i][j];
  return sum;
}
int main(int argc, char *argv[]) {
  int nc;
  double mean, A[N][N];
  Fillin(A,N,N);
  mean = Sum(A,N,N);
  mean /= N*N;
  printf("Mean: %5.2f\n", mean);
  nc = AboveMean(A,mean,N,N);
  printf("Elements above the mean: %d\n",nc);
  return 0;
}
```

0.9 p. (a) Implement an MPI parallel version of the above program, only changing function `main`. Assume that only process 0 must fill in the matrix and show the messages on the screen. You can assume that `N` is an exact multiple of the number of processes.

0.3 p. (b) Compute the sequential and parallel cost, as well as the speed-up and the asymptotic value of the speed-up when the size of the problem tends to infinity. N.B.: Assume that

the computational cost of function `fillin` is 0, and comparing two real numbers costs 1 Flop.

Parallel Computing
Degree in Computer Science Engineering (ETSInf)
Year 2018-19 ◇ Partial exam 8/11/2018 ◇ Duration: 1h 30m

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Question 1 (1.25 points)

Given the following function, where all the function calls modify only the first vector received as an argument:

```
double f(double x[], double y[], double z[], double v[], double w[]) {
   double r1, res;
   A(x,v);              /* Task A. Cost of 2*n^2 flops */
   B(y,v,w);            /* Task B. Cost of   n   flops */
   C(w,v);              /* Task C. Cost of   n^2 flops */
   r1=D(z,v);           /* Task D. Cost of 2*n^2 flops */
   E(x,v,w);            /* Task E. Cost of   n^2 flops */
   res=F(z,r1);         /* Task F. Cost of 3*n   flops */
   return res;
}
```

0.4 p. (a) Draw the dependency graph. Identify a critical path and obtain its length. Calculate the average concurrency degree.

0.5 p. (b) Implement an efficient parallel version of the function.

0.35 p. (c) Let's suppose the code in the previous part is executed using only 2 threads. Compute the parallel execution cost, the speed-up and the efficiency in the best case. Reason the answer.

Question 2 (1.25 points)

Given the following function:

```
void func(double A[M][P], double B[P][N], double C[M][N], double v[M]) {
   int i, j, k;
   double mf, val;
   for (i=0; i<M; i++) {
      mf = 0;
      for (j=0; j<N; j++) {
         val = 2.0*C[i][j];
         for (k=0; k<i; k++) {
            val += A[i][k]*B[k][j];
         }
         C[i][j] = val;
         if (val<mf) mf = val;
      }
      v[i] += mf;
   }
}
```

0.4 p. (a) Implement a parallel version based on the parallelisation of the `i` loop.

0.4 p. (b) Implement a parallel version based on the parallelisation of the `j` loop.

0.25 p. (c) Obtain the *a-priori* sequential execution time of a single iteration of the `i` loop, as well as the *a-priori* sequential execution time of the entire function. Let's suppose that the cost of comparing two floating point numbers is 1 flop.

0.2 p. (d) Analyse if there would be a good load balance if the clause `schedule(static)` is used in the parallelisation of the loop of the first part. Reason the answer.

Question 3 (1 point)

In a photography contest, each member of the jury evaluates the photographs she or he considers relevant. We have implemented a function that receives all the scores given by all the members of the jury and a vector `totals` where the total score for each photograph will be computed. The vector `totals` is initially filled with zeros.

The function computes the total score for each photograph, showing on the screen the two highest scores given by any member of the jury. It also computes and displays on the screen the average score of all the photographs, as well as the number of photos that will pass to the next phase of the contest, which are the photos which got a score higher than or equal to 20 points.

The element `k` of vector `scores[k]` has the score of photo number `index[k]`. Obviously, a photo can obtain several scores from several members of the jury.

Implement an efficient parallel version using a single parallel OpenMP region.

```
/* nf = number of photos, nv = number of scores */
void contest(int nf, int totals[], int nv, int index[], int scores[])
{
  int k,i,p,t, pass=0, max1=-1,max2=-1, total=0;
  for (k = 0; k < nv; k++) {
    i = index[k]; p = scores[k];
    totals[i] += p;
    if (p > max2)
      if (p > max1) { max2 = max1; max1 = p; } else max2 = p;
  }
  printf("The two highest scores have been %d and %d.\n",max1,max2);
  for (k = 0; k < nf; k++) {
    t = totals[k];
    if (t >= 20) pass++;
    total += t;
  }
  printf("Average Score: %g. %d photos pass to the next round.\n",
   (float)total/nf, pass);
}
```

Parallel Computing
Degree in Computer Science Engineering (ETSInf)
Year 2018-19 ◇ Partial exam 14/1/2019 ◇ Duration: 1h 30m

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Question 1 (1.3 points)

In the next sequential program, where the computational cost of each function is indicated in the associated comment, all the invoked functions only modify the first argument. Take into account that `A`, `D` and `E` are vectors, while `B` and `C` are matrices.

```
#include <stdio.h>
int main (int argc, char *argv[]) {
    double A[N], B[N][N], C[N][N], D[N], E[N], res;
    read(A);              // T0, cost N
    generate(B,A);        // T1, cost 2N
    process2(C,B);        // T2, cost 2N^2
    process3(D,B);        // T3, cost 2N^2
    process4(E,C);        // T4, cost N^2
    res = process5(E,D); // T5, cost 2N
    printf("Result: %f\n", res);
    return 0;
}
```

0.2 p. (a) Draw the associated dependency graph.

0.8 p. (b) Implement a parallel version using MPI, taking into account the following aspects:

- Use the most appropriate number of parallel processes to obtain the fastest run. Show an error message if the number of processes used when running the program is not the previous number. Only process P_0 should execute the calls to functions `read` and `printf`.
- Pay attention to the size of the messages and use merging and replication techniques if appropriate.
- Write the implementation as a whole program.

0.3 p. (c) Compute the sequential and parallel cost, speed-up and efficiency.

Question 2 (1.2 points)

Given a matrix `A` with `M` rows and `N` columns, the next function returns a vector `sup` with the number of elements in each row that are larger than the mean.

```
void func(double A[M][N], int sup[M]) {
   int i, j;
   double mean = 0;
   /* Computes the mean of matrix A */
   for (i=0; i<M; i++)
      for (j=0; j<N; j++)
         mean += A[i][j];
```

```
   mean = mean/(M*N);
   /* Counts the number of elements > mean in each row */
   for (i=0; i<M; i++) {
      sup[i] = 0;
      for (j=0; j<N; j++)
         if (A[i][j]>mean) sup[i]++;
   }
}
```

Write a parallel version of the previous function using collective communication MPI calls whenever possible. Take into account that matrix `A` is initially stored in process 0 and vector `sup` should also be in process 0 when the function ends. The computations inside the function should be evenly distributed among all the processes. You can assume that the number of rows of the matrix is an exact multiple of the number of processes.

Question 3 (1 point)

The next MPI code fragment implements an algorithm in which each process computes a matrix of `M` rows and `N` columns. All those matrices are collected in process P_0 forming a global matrix with `M` rows and `N*p` columns (where `p` is the number of processes). In this global matrix, we have first the columns of P_0, then the columns of P_1, followed by the columns of P_2 and so on.

```
int rank, i, j, k, p;
double alocal[M][N];
MPI_Comm_size(MPI_COMM_WORLD,&p);
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
/* initialization of alocal omitted here */
if (rank==0) {
  double aglobal[M][N*p];
  /* copy part belonging to P0 */
  for (i=0;i<M;i++)
    for (j=0;j<N;j++)
      aglobal[i][j] = alocal[i][j];
  /* receive data from other processes */
  for (k=1;k<p;k++)
    for (i=0;i<M;i++)
      MPI_Recv(&aglobal[i][k*N],N,MPI_DOUBLE,k,33,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
  write(p,aglobal);
} else {
  for (i=0;i<M;i++)
    MPI_Send(&alocal[i][0],N,MPI_DOUBLE,0,33,MPI_COMM_WORLD);
}
```

0.8 p. (a) Change the previous code so that each process sends a single message, instead of one message per row. For this purpose, you should define an MPI derived type for the reception of the message.

0.2 p. (b) Obtain the communication cost for both the original and the modified versions.

Parallel Computing
Degree in Computer Science Engineering (ETSInf)
Year 2018-19 ◇ Final exam 24/1/19 ◇ Block OpenMP ◇ Duration: 1h 30m

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Question 1 (1 point)
Given the next function:

```
double cuad_mat(double a[N][N], double b[N][N])
{
   int i,j,k;
   double aux, s=0.0;
   for (i=0; i<N; i++) {
      for (j=0; j<N; j++) {
         aux = 0.0;
         for (k=i; k<N; k++)
            aux += a[i][k] * a[k][j];
         b[i][j] = aux;
         s += aux*aux;
      }
   }
   return s;
}
```

0.6 p. (a) Implement an efficient parallel version of the previous code using OpenMP. Out of the potential schedulings, which would be the most efficient ones? Reason your answer.

0.4 p. (b) Obtain the sequential cost in flops of the algorithm.

Question 2 (1.35 points)
In the next function none of the functions called modify their arguments.

```
int exercise(double v[n],double x)
{
  int i,j,k=0;
  double a,b,c;
  a = task1(v,x);   /* task 1, cost n flops */
  b = task2(v,a);   /* task 2, cost n flops */
  c = task3(v,x);   /* task 3, cost 4n flops */
  x = x + a + b + c;   /* task 4 */
  for (i=0; i<n; i++) {   /* task 5 */
    j = f(v[i],x);   /* each function call costs 6 flops */
    if (j>0 && j<4) k++;
  }
  return k;
}
```

0.1 p. (a) Compute the sequential execution time.

0.4 p. (b) Draw a dependeny graph at the task level (considering task 5 as a whole), and obtain the maximum concurrency degree, the length of the critical path and the average concurrency degree.

0.6 p. (c) Implement an efficient parallel OpenMP version using a single parallel region. Parallelize both the tasks that could run concurrently and also the loop of task 5.

0.25 p. (d) Considering that the program is run using 6 threads (and assuming that n is an exact multiple of 6), obtain the parallel execution time, the speed-up and the efficiency.

Question 3 (1.15 points)

The next function processes the billing, at the end of the month, of all the songs downloaded by a group of users in a music virtual shop. The shop registers the user identifier and the song identifier of each one of the n downloads performed. Those identifiers are stored in the vectors `users` and `songs` respectively. Each song has a different price, stored in the vector `prices`. The function also displays on the screen the identifier of the most downloaded song. The vectors `ndownloads` and `billing` are set to zero before the function is called.

```
void monthbilling(int n, int users[], int songs[], float prices[],
                  float billing[], int ndownloads[])
{
  int i,u,c,best_song=0;
  float p;
  for (i=0;i<n;i++) {
     u = users[i];
     c = songs[i];
     p = prices[c];
     billing[u] += p;
     ndownloads[c]++;
  }
  for (i=0;i<NC;i++) {
     if (ndownloads[i]>ndownloads[best_song])
        best_song = i;
  }
  printf("Song %d has been downloaded most\n",best_song);
}
```

0.5 p. (a) Implement an efficient parallel version in OpenMP of the above function using a single parallel region.

0.15 p. (b) Would it be reasonable to use the `nowait` clause in the first loop?

0.5 p. (c) Update the previous parallel code so that each thread displays on the screen the identifier and the number of iterations from the first loop that the thread has processed.

Question 4 (1.2 points)

The next program reads a vector from a file, modifies it, and displays a summary on the screen, as well as writing the modified vector on a file.

```
double facto(int m,double x)
{
  int i;
  double p = 1.0;
  for (i=1; i<=m; i++) {
    p = p * x;
    x = x + 1.0;
  }
  return p;
}
```

```
int main(int argc,char *argv[])
{
  int i, n;
  double a = 1.0, v[MAXN];

  n = read_vector(v);
  for (i=0; i<n; i++) {
    v[i] = facto(n,v[i]);
    a = a * v[i];
  }
  printf("Factor alfalfa: %.2f\n",a);
  write_vector(n,v);
  return 0;
}
```

0.7 p. (a) Implement an MPI parallel version using collective communication primitives wherever possible. The input/output to the file and the display on the screen must be done only by process 0. You can assume that the size of the vector (`n`) is an exact multiple of the number of processes. Note that the size of the vector is not known a priori and it is returned by function `read_vector`.

0.2 p. (b) Obtain the sequential execution time.

0.3 p. (c) Obtain the parallel execution time, clearly indicating the communication cost of each operation. Do not simplify the expressions.

Question 5 (1 point)

We want to distribute a matrix `A` with `F` rows and `C` columns among the processes in an MPI communicator, using a distribution based on blocks of columns. The number of processes is `C`/2, and `C` is an even number. The local matrix `Aloc` in each process will hold two columns.

Implement a function using the following header to perform the previous distribution and using point-to-point communication primitives. The matrix `A` is initially in process 0, and at the end of the function each process should have in `Aloc` the corresponding local part of the global matrix.

Use the proper MPI data types to ensure that only one message per process is sent.

```
void distrib(double A[F][C], double Aloc[F][2], MPI_Comm com)
```

Question 6 (1.3 points)

Given the following function, which computes the sum of a vector with `N` elements:

```
double sum(double v[N])
{
  int i;
  double s = 0.0;
  for (i=0; i<N; i++) s += v[i];
  return s;
}
```

0.7 p. (a) Implement an MPI parallel version using only point-to-point communication primitives. The vector `v` is initially in process 0 and the result must be correct in all the processes. You can assume that the size of the vector (`N`) is an exact multiple of the number of processes.

0.6 p. (b) Implement another MPI parallel version of the previous algorithm under the same conditions but using collective communications whenever is more convenient.

Parallel Computing
Degree in Computer Science Engineering (ETSInf)
Year 2019-20 ◇ Partial exam 30/10/2019 ◇ Duration: 1h 30m

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Question 1 (1.3 points)

```
void matmult(double A[N][N],
             double B[N][N],double C[N][N]) {
  int i,j,k;
  double sum;
  for (i=0; i<N; i++) {
    for (j=0; j<N; j++) {
      sum = 0.0;
      for (k=0; k<N; k++) {
        sum += A[i][k]*B[k][j];
      }
      C[i][j] = sum;
    }
  }
}
```

```
void normalize(double A[N][N]) {
  int i,j;
  double sum=0.0,factor;
  for (i=0; i<N; i++) {
    for (j=0; j<N; j++) {
      sum += A[i][j]*A[i][j];
    }
  }
  factor = 1.0/sqrt(sum);
  for (i=0; i<N; i++) {
    for (j=0; j<N; j++) {
      A[i][j] *= factor;
    }
  }
}
```

Given the definition of the functions above, we want to parallelize the following code:

```
matmult(A,B,R1);    /* T1 */
matmult(C,D,R2);    /* T2 */
normalize(R1);      /* T3 */
normalize(R2);      /* T4 */
matmult(A,R2,M1);   /* T5 */
matmult(B,R2,M2);   /* T6 */
matmult(C,R1,M3);   /* T7 */
matmult(D,R1,M4);   /* T8 */
```

0.6 p. (a) Draw the task dependency graph. Indicate which is the length of the critical path and the average degree of concurrency. Note: to determine these values, it is necessary to obtain the cost in flops of both functions. Assume that `sqrt` costs 5 flops.

0.7 p. (b) Do the parallelization based on sections, using the previous task dependency graph.

Question 2 (1 point)

We want to obtain the distribution of grades obtained by the CPA students, classifying the grades in five categories: *suspenso*, *aprobado*, *notable*, *sobresaliente* and *matrícula de honor*.

```
void histogram(int histo[], float marks[], int n) {
  int i, mark;
  float rmark;
  for (i=0;i<5;i++) histo[i] = 0;
  for (i=0;i<n;i++) {
```

```
    rmark = round(marks[i]*10)/10.0;
    if (rmark<5) mark = 0;        /* suspenso */
    else
      if (rmark<7) mark = 1;      /* aprobado */
      else
        if (rmark<9) mark = 2;    /* notable */
        else
          if (rmark<10) mark = 3; /* sobresaliente */
          else
            mark = 4;             /* matricula de honor */
    histo[mark]++;
  }
}
```

0.4 p. (a) Parallelize function `histogram` appropriately with OpenMP.

0.6 p. (b) Modify function `histogram` so that it prints the number of the student with the best mark and its mark, and the value of the worst mark (both of them without rounding).

Question 3 (1.2 points)

Given the following function:

```
double f(double A[N][N], double B[N][N], double vs[N], double bmin) {
  int i, j;
  double x, y, aux, stot=0;
  for (i=0; i<N; i++) {
    aux = 0;
    for (j=0; j<N; j++) {
      x = A[i][j]*A[i][j]/2.0;
      A[i][j] = x;
      aux += x;
    }
    for (j=i; j<N; j++) {
      if (B[i][j]<bmin) y = bmin;
      else              y = B[i][j];
      B[i][j] = 1.0/y;
    }
    vs[i] = aux;
    stot += vs[i];
  }
  return stot;
}
```

0.4 p. (a) Paralellize (efficiently) the `i` loop by means of OpenMP.

0.4 p. (b) Paralellize (efficiently) both `j` loops by means of OpenMP.

0.2 p. (c) Compute the sequential cost of the original code.

0.2 p. (d) Assuming that we parallelize only the first `j` loop, compute the parallel cost of such version. Obtain the speedup and efficiency in case we have `N` processors available.

Parallel Computing
Degree in Computer Science Engineering (ETSInf)
Year 2019-20 ◇ Partial exam 7/1/2020 ◇ Duration: 1h 50m

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Question 1 (1.1 points)

We want to parallelize the following code with MPI.

```
void calculate(int n, double x[], double y[], double z[]) {
  int i;
  double alpha, beta;

  /* Read vectors x, y, z, of dimension n */
  read(n, x, y, z);           /* task 1 */

  normalize(n,x);             /* task 2 */
  beta = obtain(n,y);         /* task 3 */
  normalize(n,z);             /* task 4 */

  /* task 5 */
  alpha = 0.0;
  for (i=0; i<n; i++)
    if (x[i] > 0.0) { alpha = alpha + beta*x[i]; }
    else { alpha = alpha + x[i]*x[i]; }

  /* task 6 */
  for (i=0; i<n; i++) z[i] = z[i] + alpha*y[i];
}
```

Suppose we are using 3 processes, from which only one has to call function `read`. We can assume that the value of `n` is available in all processes. The final result (`z`) may be stored in any of the 3 processes. Function `read` modifies the three vectors, function `normalize` modifies its second argument and function `obtain` does not modify any of its arguments.

0.25 p. (a) Draw the task dependency graph.

0.85 p. (b) Write the MPI code that solves the problem using an assignment that maximizes the parallelism and minimizes the cost of communications.

Question 2 (1.3 points)

We want to implement with MPI the sending by process 0 (and reception by the rest of processes) of the main diagonal and antidiagonal of a matrix A, using derived data types (one type per each class of diagonal) and the smallest possible number of messages. Suppose that:

- N is a known constant.
- The elements of the main diagonal are: $A_{0,0}$, $A_{1,1}$, $A_{2,2}$, ..., $A_{N-1,N-1}$.

- The elements of the antidiagonal are: $A_{0,N-1}$, $A_{1,N-2}$, $A_{2,N-3}$, ..., $A_{N-1,0}$.
- Only process 0 owns matrix A and will send the full diagonals to the rest of processes.

An example for a matrix of size $N = 5$ would be: $A = \begin{pmatrix} * & & & & * \\ & * & & * & \\ & & * & & \\ & * & & * & \\ * & & & & * \end{pmatrix}$

0.6 p. (a) Complete the following function, where the processes from rank 1 onward will store on matrix `A` the received diagonals:

```
void sendrecv_diagonals(double A[N][N]) {
```

0.7 p. (b) Complete this other function, a variant of the previous one, where all processes (including process 0) will store on vectors `maind` and `antid` the corresponding diagonals:

```
void sendrecv_diagonals(double A[N][N], double maind[N], double antid[N]) {
```

Question 3 (1.1 points)

Observe the following function, that counts the number of occurences of a number in a matrix and also indicates the first row in which it appears:

```
void search(double A[M][N], double x) {
  int i,j,first,count;
  first = M ; count = 0;
  for (i=0; i<M; i++)
    for (j=0; j<N; j++)
      if (A[i][j] == x) {
        count++;
        if (i < first) first = i;
      }
  printf("%g is found %d times, the first one in row %d.\n",x,count,first);
}
```

0.9 p. (a) Parallelize it by means of MPI distributing the `A` matrix among all available processes. Both the matrix and the value to be sought are initially only available at process `owner`. We assume that the number of rows and columns of the matrix is an exact multiple of the number of processes. The `printf` that shows the result on the screen must be done only by one process.

Use collective communication operations whenever possible.

For this, complete this function:

```
void par_search(double A[M][N], double x, int owner) {
  double Aloc[M][N];
```

0.2 p. (b) Indicate the communication cost of each communication operation that has been used in the previous code. Assume a basic implementation of the communications.

Parallel Computing
Degree in Computer Science Engineering (ETSInf)
Year 2019-20 ◇ Final exam 16/1/20 ◇ Block OpenMP ◇ Duration: 1h 30m

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Question 1 (1.25 points)
Given the following function:

```
double myadd(double A[N][M])
{
   double sum=0, maximum;
   int i,j;

   for (i=0; i<N; i++) {
      maximum=0;
      for (j=0; j<M; j++) {
         if (A[i][j]>maximum) maximum = A[i][j];
      }
      for (j=0; j<M; j++) {
         if (A[i][j]>0.0) {
            A[i][j] = A[i][j]/maximum;
            sum = sum + A[i][j];
         }
      }
   }
   return sum;
}
```

0.5 p. (a) Parallelize the function efficiently by means of OpenMP.

0.2 p. (b) Indicate its theoretical parallel cost (in flops), assuming that `N` is a multiple of the number of threads. To evaluate the cost consider the worst case, that is, when all comparisons are true. Also, suppose that the cost of comparing two real numbers is 1 *flop*.

0.55 p. (c) Modify the code so that each thread shows a single message with its thread number and the number of elements that it has summed.

Question 2 (1.25 points)
Given the following code:

```
double a,b,c,e,d,f;
T1(&a,&b); // Cost: 10 flops
c=T2(a);   // Cost: 15 flops
c=T3(c);   // Cost:  8 flops
d=T4(b);   // Cost: 20 flops
e=T5(c);   // Cost: 30 flops
f=T6(c);   // Cost: 35 flops
b=T7(c);   // Cost: 30 flops
```

0.3 p. (a) Obtain the task dependency graph and explain which type of dependencies occur between T_2 and T_3 and between T_4 and T_7, in case there is one.

0.1 p. (b) Compute the length of the critical path, and indicate the tasks that it contains.

0.6 p. (c) Implement a parallel version as efficient as possible of the previous code by means of sections, employing just one parallel region.

0.25 p. (d) Compute the speedup and efficiency in the case of using 4 threads to run the parallel code.

Question 3 (1 point)

The following function manages a certain number of trips, that have taken place during a concrete period of time, by means of the public bicycle service of a city. For each of the trips, the identifiers of the source and destination stations are stored, together with the elapsed time (expressed in minutes) in each of them. The vector `num_bikes` stores the number of bicycles available in each station. Furthermore, the function computes between which stations the longest and shortest trips took place, together with the average elapsed time of all trips.

```
struct trip {
   int source_station;
   int dest_station;
   float time_minutes;
};
void update_bikes(struct trip trips[],int num_trips,int num_bikes[]) {
   int i,source,dest,srcmax,srcmin,destmax,destmin;
   float time,tmax=0,tmin=9999999,tavg=0;
   for (i=0;i<num_trips;i++) {
      source = trips[i].source_station;
      dest   = trips[i].dest_station;
      time   = trips[i].time_minutes;
      num_bikes[source]--;
      num_bikes[dest]++;
      tavg += time;
      if (time>tmax) {
         tmax=time; srcmax=source; destmax=dest;
      }
      if (time<tmin) {
         tmin=time; srcmin=source; destmin=dest;
      }
   }
   tavg /= num_trips;
   printf("Average time of trips: %.2f minutes\n",tavg);
   printf("Longest trip (%.2f min.) station %d to %d\n",tmax,srcmax,destmax);
   printf("Shortest trip (%.2f min.) station %d to %d\n",tmin,srcmin,destmin);
}
```

Parallelize the function by means of OpenMP in the most efficient way.

Question 4 (1.2 points)

0.6 p. (a) The following fragment of code uses point-to-point communication primitives for a communication pattern that can be achieved by means of a single collective operation.

```
#define TAG 999
int sz, rank;
double val,res,aux;
MPI_Comm comm=MPI_COMM_WORLD;
MPI_Status stat;
val = ...
MPI_Comm_size(comm, &sz);
if (sz==1) res = val;
else {
  MPI_Comm_rank(comm, &rank);
  if (rank==0) {
    MPI_Recv(&aux, 1, MPI_DOUBLE, rank+1, TAG, comm, &stat);
    res = aux + val;
  } else if (rank==sz-1) {
    MPI_Send(&val, 1, MPI_DOUBLE, rank-1, TAG, comm);
  } else {
    MPI_Recv(&aux, 1, MPI_DOUBLE, rank+1, TAG, comm, &stat);
    aux = aux + val;
    MPI_Send(&aux, 1, MPI_DOUBLE, rank-1, TAG, comm);
  }
}
```

Write the call to the equivalent MPI primitive of collective communication, with the corresponding arguments.

0.6 p. (b) Given the following call to a collective communication primitive:

```
double val=...;
MPI_Bcast(&val, 1, MPI_DOUBLE, 0, comm);
```

Write the equivalent fragment of code (must realize the same communication) but using only point-to-point communication primitives.

Question 5 (1.3 points)

Write an MPI parallel program in which process 0 reads a matrix of $M \times N$ real numbers from disk (with function `read_mat`) and this matrix is being passed from one process to the next until it reaches the last one, who will return it to process 0. The program must measure the total execution time, without considering the reading from disk, and show it on the screen.

Use the following header for the main function:

```
int main(int argc,char *argv[])
```

and take into account that the function for reading the matrix has this header:

```
void read_mat(double A[M][N]);
```

1 p. (a) Write the requested program.

0.3 p. (b) Indicate the total theoretical cost of the communications.

Question 6 (1 point)

We want to distribute a matrix of M rows and N columns that is stored in process 0 among 4 processes by means of a cyclic column distribution. As an example, we show the case of a matrix of 6 rows and 8 columns.

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 \\ 21 & 22 & 23 & 24 & 25 & 26 & 27 & 28 \\ 31 & 32 & 33 & 34 & 35 & 36 & 37 & 38 \\ 41 & 42 & 43 & 44 & 45 & 46 & 47 & 48 \\ 51 & 52 & 53 & 54 & 55 & 56 & 57 & 58 \end{bmatrix}$$

It would be distributed in the following way:

$$P_0 = \begin{bmatrix} 1 & 5 \\ 11 & 15 \\ 21 & 25 \\ 31 & 35 \\ 41 & 45 \\ 51 & 55 \end{bmatrix}, \quad P_1 = \begin{bmatrix} 2 & 6 \\ 12 & 16 \\ 22 & 26 \\ 32 & 36 \\ 42 & 46 \\ 52 & 56 \end{bmatrix}, \quad P_2 = \begin{bmatrix} 3 & 7 \\ 13 & 17 \\ 23 & 27 \\ 33 & 37 \\ 43 & 47 \\ 53 & 57 \end{bmatrix}, \quad P_3 = \begin{bmatrix} 4 & 8 \\ 14 & 18 \\ 24 & 28 \\ 34 & 38 \\ 44 & 48 \\ 54 & 58 \end{bmatrix}$$

Implement a function using MPI that carries out the sending and reception of the matrix, by means of point-to-point primitives, as efficiently as possible. N.B.: The recepction of the matrix must be done in a compact matrix (in `lmat`), as shown in the previous example. N.B.: You can assume that the number of columns is a multiple of 4 and that it is distributed always among 4 processes.

For the implementation we recommend to use the following header:

```
int MPI_Distrib_col_cyc(float mat[M][N], float lmat[M][N/4])
```