Parallel Computing
Degree in Computer Science Engineering (ETSInf)
Year 2023/24 ◇ Partial exam 17/1/24 ◇ Block MPI ◇ Duration: 1h 45m

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Question 1 (1 point)

The following program scales a matrix by dividing all its elements by the largest element and then displays the smallest value of the resulting elements. The matrix is first read from disk and written back to disk after scaling (functions `read` and `write`).

```
#include <stdio.h>

#define M 2000
#define N 1000

int main(int argc,char *argv[])
{ double A[M][N], max,min;
  int i,j;

  read(A);

  max = -1e6;
  for ( i = 0 ; i < M ; i++ )
    for ( j = 0 ; j < N ; j++ )
       if ( A[i][j] > max ) max = A[i][j];

  min = 1e6;
  for ( i = 0 ; i < M ; i++ )
    for ( j = 0 ; j < N ; j++ ) {
       A[i][j] /= max;
       if ( A[i][j] < min ) min = A[i][j];
  }

  write(A);
  printf("The smallest value is %g\n",min);

  return 0;
}
```

Parallelize this program with MPI, making sure that the work of the loops is distributed among all available processes. Use collective communication operations wherever possible. Only process 0 has access to the disk and the screen, so the operations `read`, `write` and the `printf` must be done by this process. We assume that `M` and `N` are an exact multiple of the number of processes.

Solution:

```
#include <stdio.h>
#include <mpi.h>

#define M 2000
```

```
#define N 1000

int main(int argc,char *argv[])
{ double A[M][N], max,min, B[M][N], x;
  int i,j, id,np, nb;

  MPI_Init(&argc,&argv);

  MPI_Comm_rank(MPI_COMM_WORLD,&id);
  MPI_Comm_size(MPI_COMM_WORLD,&np);
  nb = M / np;

  if ( id == 0 ) read(A);

  MPI_Scatter( A,nb*N,MPI_DOUBLE, B,nb*N,MPI_DOUBLE, 0, MPI_COMM_WORLD );

  max = -1e6;
  for ( i = 0 ; i < nb ; i++ )
     for ( j = 0 ; j < N ; j++ )
        if ( B[i][j] > max ) max = B[i][j];

  MPI_Allreduce( &max,&x,1,MPI_DOUBLE, MPI_MAX, MPI_COMM_WORLD );
  max = x;

  min = 1e6;
  for ( i = 0 ; i < nb ; i++ )
     for ( j = 0 ; j < N ; j++ ) {
        B[i][j] /= max;
        if ( B[i][j] < min ) min = B[i][j];
  }

  MPI_Gather( B,nb*N,MPI_DOUBLE, A,nb*N,MPI_DOUBLE, 0, MPI_COMM_WORLD);
  MPI_Reduce( &min,&x,1,MPI_DOUBLE, MPI_MIN, 0, MPI_COMM_WORLD );

  if ( id == 0 ) {
    write(A);
    printf("The smallest value is %g\n",x);
  }

  MPI_Finalize();

  return 0;
}
```

Question 2 (1.2 points)

Given a square matrix $A \in \mathbb{R}^{n \times n}$, the following sequential code implements the computation of its ∞-norm, which is defined as the maximum of the sums of the absolute values of the elements in each row: $\max_{i=1..n} \left\{ \sum_{j=0}^{n-1} |a_{i,j}| \right\}$. The math library function `fabs` returns the absolute value of a floating point number.

```
double infNorm(double A[N][N]) {
```

```
  int i,j;
  double sum[N],nrm=0.0;

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

0.9 p.

(a) Implement a parallel version using MPI that performs a distribution of the matrix by blocks of COLUMNS. The matrix is initially stored in P_0 and the result must also be in P_0 (which is why it is recommended do not parallelize the last of the loops) . It can be assumed that the value of `N` is an exact multiple of the number of processes. In the implementation, to perform the matrix distribution it is NOT necessary to use collective communication, only point-to-point communication primitives, but guaranteeing that a single message is sent to each process from P_0. For other parts of the algorithm it is possible to use collective communication.

Note: the suggested prototype of the parallel function is show below, where `Aloc` is a matrix that is assumed to be already allocated in memory, and that can be used by the function to store the local part of matrix `A` (although its size is larger than necessary and only the first columns of the matrix, starting from column 0, should be used).

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

Solution: A derived data type covering `N/p` columns of the matrix must be defined. The sums are stored in an auxiliary vector `sumloc`, to which a reduction must be applied after the calculation.

```
double infNormPar(double A[N][N], double Aloc[N][N]) {
  int i,j,p,rank,k;
  double sum[N],sumloc[N],nrm=0.0;
  MPI_Datatype submat;

  MPI_Comm_size(MPI_COMM_WORLD,&p);
  MPI_Comm_rank(MPI_COMM_WORLD,&rank);
  k = N/p;
  MPI_Type_vector(N,k,N,MPI_DOUBLE,&submat);
  MPI_Type_commit(&submat);
  if (rank == 0) {
    for (i=1; i<p; i++) {
      MPI_Send(&A[0][i*k],1,submat,i,0,MPI_COMM_WORLD);
    }
    MPI_Sendrecv(A,1,submat,0,0,Aloc,1,submat,0,0,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
  } else {
    MPI_Recv(Aloc,1,submat,0,0,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
  }
  for (i=0; i<N; i++) {
    sumloc[i]=0.0;
    for (j=0; j<k; j++)
      sumloc[i]+=fabs(Aloc[i][j]);
```

```
    }
    MPI_Reduce(sumloc,sum,N,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);
    if (rank == 0) {
      for (i=0; i<N; i++) {
        if (sum[i]>nrm) nrm=sum[i];
      }
    }
    MPI_Type_free(&submat);
    return nrm;
  }
```

0.3 p. (b) Obtain the computational and communication cost of the parallel algorithm. It can be assumed that the `fabs` operation has a negligible cost, as well as the comparisons.

Solution: We denote by n the size of the problem (`N`). The cost includes three stages:

- Cost of the data distribution: $(p-1) \cdot \left(t_s + n \cdot \frac{n}{p} \cdot t_w\right)$
- Cost of processing $\frac{n}{p}$ columns: $n \cdot \frac{n}{p} = \frac{n^2}{p}$Flops
- Cost of the reduction (trivial implementation): $(p-1) \cdot (t_s + nt_w) + (p-1)n$

Therefore, the total cost is approximately: $2pt_s + (n^2 + pn)t_w + \frac{n^2}{p} + pn$

Question 3 (1.3 points)

Given the following sequential program, in which the computational cost and the treatment of the arguments of each of the functions are described below:

- `readmat(A, B, C, N)` modifies the three first arguments and has a cost of $3N^2$ Flops.
- `pot(A, exp, N)` modifies only the first argument and has a cost of $exp \times 2N^3$ Flops.
- `nm = sumnor(A, C, N)` does not modify any argument and has a cost of $3N^2$ Flops.
- `nm = norsc(A, nm, N)` does not modify any argument and has a cost of N^2 Flops.

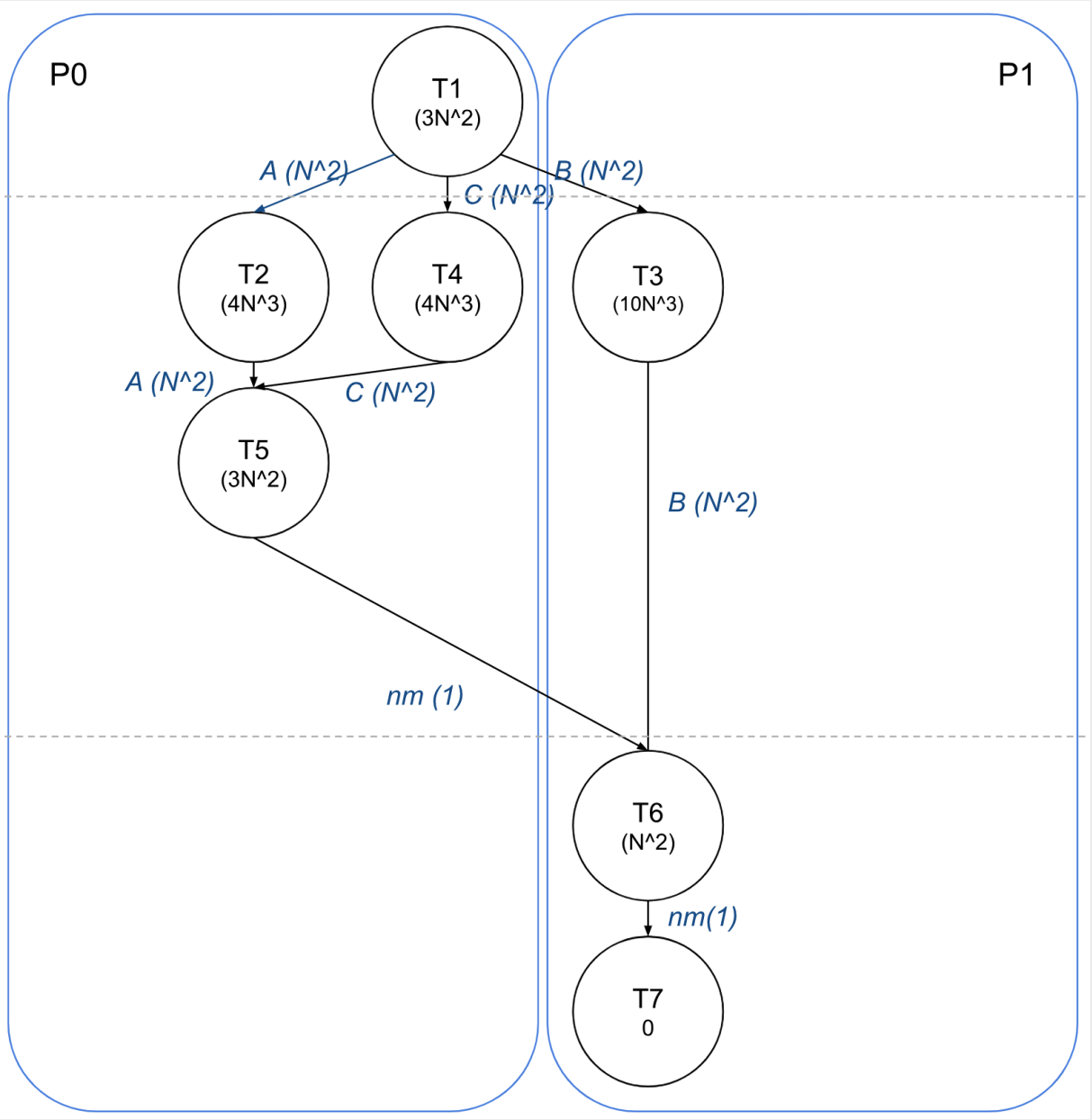
```
int main(int argc, char *argv[]){
  double A[N][N], B[N][N], C[N][N];
  double nm;

  readmat(A, B, C, N);      // T1
  pot(A, 2, N);             // T2
  pot(B, 5, N);             // T3
  pot(C, 2, N);             // T4
  nm = sumnor(A, C, N);     // T5
  nm = norsc(B, nm, N);     // T6
  printf ("%f\n", nm);      // T7

  return 0;
}
```

0.3 p. (a) Draw the task dependency graph of the sequential program.

Solution:



0.7 p.

(b) Write a parallel version via MPI using 2 processes, maximizing parallelism and reducing communication time. The function `readmat` must be called from process 0 and the screen output must be made only once.

Solution:

```
int main(int argc, char *argv[]) {
  double A[N][N], B[N][N], C[N][N];
  double nm;
  int rank;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  if (rank == 0) {
```

```
    readmat(A, B, C, N);      // T1
    MPI_Send(B, N*N, MPI_DOUBLE, 1, 0, MPI_COMM_WORLD);
    pot(A, 2, N);             // T2
    pot(C, 2, N);             // T4
    nm = sumnor(A, C, N);     // T5
    MPI_Send(&nm, 1, MPI_DOUBLE, 1, 0, MPI_COMM_WORLD);
  } else {
    MPI_Recv(B, N*N, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    pot(B, 5, N);             // T3
    MPI_Recv(&nm, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    nm = norsc(B, nm, N);     // T6
    printf ("%f\n", nm);      // T7
  }
  MPI_Finalize();
  return 0;
}
```

0.3 p. (c) Calculate the expression of sequential time and parallel time for two processes. Calculate also the values of speed-up and efficiency when the problem size (N) tends to infinity.

Solution:

$$t(N) = 3N^2 + 4N^3 + 10N^3 + 4N^3 + 3N^2 + N^2 = 18N^3 + 7N^2 \approx 18N^3 Flops$$

$$ta(N,2) = 3N^2 + max(4N^3 + 4N^3 + 3N^2, 10N^3) + N^2 = 10N^3 + 4N^2 \approx 10N^3 Flops$$

$$tc(N,2) = (t_s + N^2 t_w) + (t_s + t_w) \approx 2t_s + N^2 t_w$$

$$t(N,2) = 10N^3 + 2t_s + N^2 t_w$$

$$S(N,2) = \frac{18N^3}{10N^3 + 2t_s + N^2 t_w}$$

$$E(N,2) = \frac{18N^3}{2(10N^3 + 2t_s + N^2 t_w)} = \frac{9N^3}{10N^3 + 2t_s + N^2 t_w}$$

$$\lim_{N\to\infty} S(N,2) = \lim_{N\to\infty} \frac{18N^3}{10N^3 + 2t_s + N^2 t_w} = 18/10 = 1.8$$

$$\lim_{N\to\infty} E(N,p) = 1.8/2 = 0.9$$