**Computació Paral · lela**
Grau en Enginyeria Informàtica (ETSINF)
Curs 2023/24 ◇ Examen parcial 17/1/24 ◇ Bloc MPI ◇ Duració: 1h 45m

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

**Qüestió 1** (1 punt)

El següent programa escala una matriu dividint tots els seus elements pel valor del major d'ells i posteriorment mostra per pantalla el menor valor dels elements del resultat. La matriu es llig primer de disc i s'escriu una altra vegada en disc després del seu escalat (funcions `llig` i `escriu`).

```
#include <stdio.h>

#define M 2000
#define N 1000

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

  llig(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];
    }

  escriu(A);
  printf("El menor valor es %g\n",min);

  return 0;
}
```

Paral · lelitza este programa amb MPI procurant que el treball dels bucles quede repartit entre tots els processos disponibles. Utilitza operaciones de comunicació col · lectiva on siga possible. Només el procés 0 té accés al disc i a la pantalla, pel que les operacions `llig`, `escriu` i el `printf` ha de fer-les este procés. Se suposa que `M` i `N` són múltiples exactes del nombre de processos.

**Solució:**

```
#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 ) llig(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 ) {
    escriu(A);
    printf("El menor valor es %g\n",x);
  }

  MPI_Finalize();

  return 0;
}
```

**Qüestió 2** (1.2 punts)

Donada una matriu cuadrada $A \in \mathbb{R}^{n \times n}$, el següent codi seqüencial implementa el càlcul de la seua $\infty$-norma, que es defineix com el màxim de les sumes dels valors absoluts dels elements de cada fila: $\max\limits_{i=1..n} \left\{ \sum\limits_{j=0}^{n-1} |a_{i,j}| \right\}$.

La funció `fabs` de la biblioteca matemàtica torna el valor absolut d'un nombre en coma flotant.

```
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) Implementa una versió paral · lela mitjançant MPI que realitze una distribució de la matriu per blocs de COLUMNES. La matriu està inicialment emmagatzemada en $P_0$ i el resultat ha de quedar també en $P_0$ (raó per la qual es recomana no paral · lelitzar l'últim dels bucles). Es pot assumir que el valor de `N` es un múltiple exacte del nombre de processos. En la implementació, per a realitzar la distribució de la matriu NO s'ha d'usar comunicació col · lectiva, sinó únicament primitives de comunicació punt a punt, però garantint que s'envia un únic missatge a cada procés des de $P_0$. Per a altres parts de l'algoritme sí es possible utilitzar comunicació col · lectiva.

Nota: es suggereix la següent capçalera per a la funció paral · lela, on `Aloc` és una matriu que se suposa ja reservada en memòria, i que pot ser utilitzada per la funció per a emmagatzemar la part local de la matriu `A` (encara que la seua grandària es major de la necessària i s'hauran d'usar només les primeres columnes de l'esmentada matriu, a partir de la columna 0).

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

**Solució:** Cal definir un tipus de dades derivat que corresponga a `N/p` columnes de la matriu. Las sumes s'emmagatzemen en un vector auxiliar `sumloc`, al qual s'haurà d'aplicar una reducció després del càlcul.

```
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) Obtín el cost computacional i de comunicacions de l'algoritme paral · lel. Es pot assumir que l'operació `fabs` té un cost menyspreable, així com les comparacions.

**Solució:** Denotem amb $n$ la grandària del problema (`N`). El cost inclou tres etapes:

- Cost del repartiment: $(p-1) \cdot \left(t_s + n \cdot \frac{n}{p} \cdot t_w\right)$
- Cost del processat de $\frac{n}{p}$ columnes: $n \cdot \frac{n}{p} = \frac{n^2}{p}$Flops
- Cost de la reducció (implementació trivial): $(p-1) \cdot (t_s + nt_w) + (p-1)n$

Per tant, el cost total és aproximadament: $2pt_s + (n^2 + pn)t_w + \frac{n^2}{p} + pn$

**Qüestió 3** (1.3 punts)

Donat el següent programa seqüencial, on el cost computacional i el tractament dels arguments de cadascuna de les funcions es descriuen a continuació:

- `lligmatrius(A, B, C, N)` modifica els tres primers arguments i té un cost de $3N^2$ Flops.
- `pot(A, exp, N)` modifica únicament el primer argument i té un cost de $exp \times 2N^3$ Flops.
- `nm = sumnor(A, C, N)` no modifica cap argument i té un cost de $3N^2$ Flops.
- `nm = norsc(A, nm, N)` no modifica cap argument i té un cost de $N^2$ Flops.

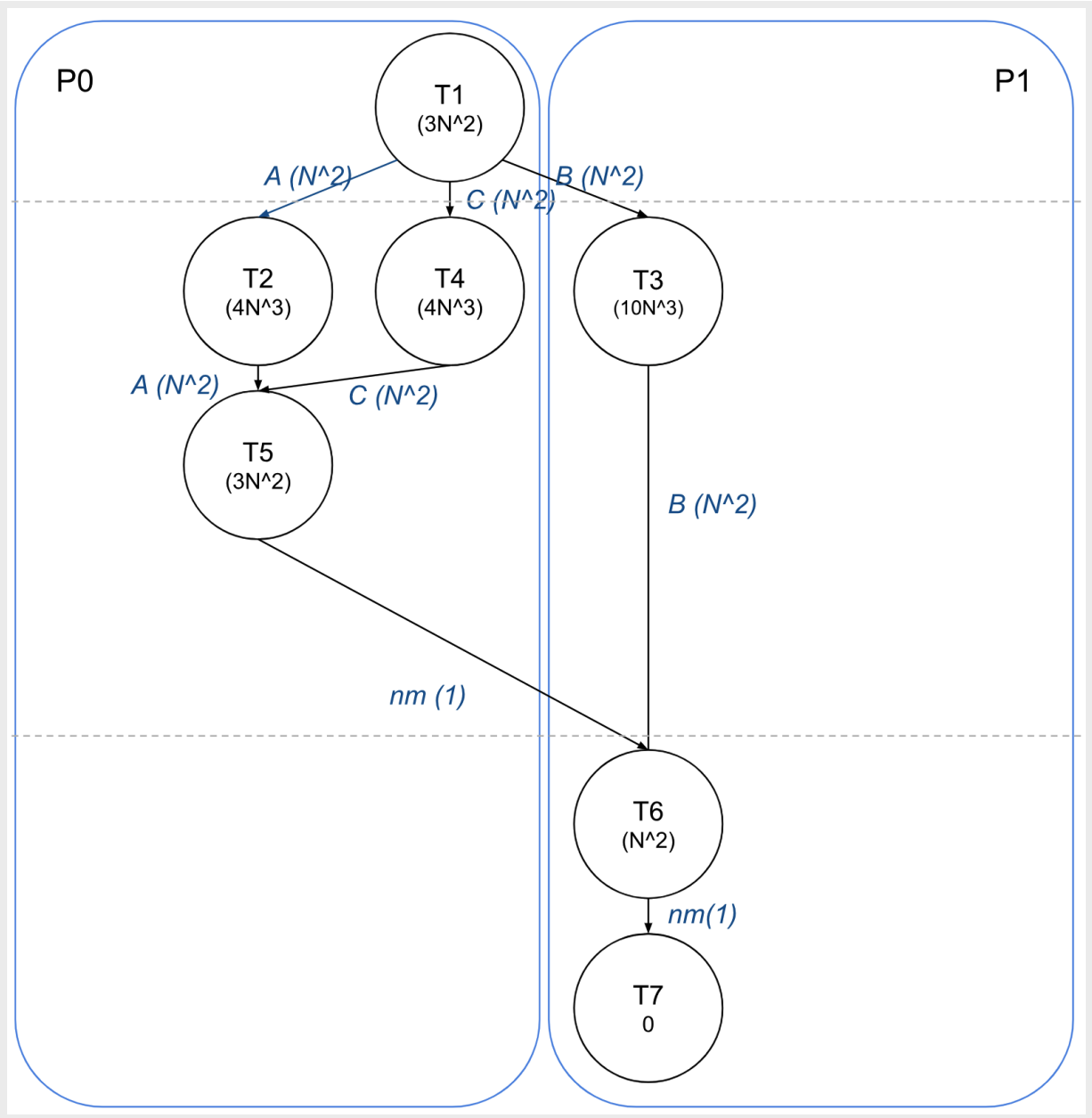
```
int main(int argc, char *argv[]){
  double A[N][N], B[N][N], C[N][N];
  double nm;

  lligmatrius(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) Realitza el graf de dependències del programa seqüencial.

**Solució:**




0.7 p.

(b) Realitza una versió paral · lela mitjançant MPI utilitzant 2 processos, maximitzant el paral · lelisme i reduint el temps de comunicacions. La crida a la funció `lligmatrius` haurà de fer-se des del procés 0 i l'eixida per pantalla s'ha de fer una únicament una vegada.

**Solució:**

```
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) {
    lligmatrius(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) Calcula l'expresió del temps seqüencial i el temps paral · lel per a dos processos. Calcula també els valors de Speed Up i eficiència quan la grandària del problema ($N$) tendeix a infinit.

**Solució:**

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

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

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

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

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

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

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

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