Computación Paralela
Grado en Ingeniería Informática (ETSInf)
Curso 2023/24 ◇ Examen parcial 17/1/24 ◇ Bloque MPI ◇ Duración: 1h 45m

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Cuestión 1 (1 punto)

El siguiente programa escala una matriz dividiendo todos sus elementos por el valor del mayor de ellos y posteriormente muestra por pantalla el menor valor de los elementos del resultado. La matriz se lee primero de disco y se escribe otra vez en el disco después de su escalado (funciones `lee` y `escribe`).

```
#include <stdio.h>

#define M 2000
#define N 1000

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

  lee(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];
    }

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

  return 0;
}
```

Paraleliza este programa con MPI procurando que el trabajo de los bucles quede repartido entre todos los procesos disponibles. Utiliza operaciones de comunicación colectiva allá donde sea posible. Sólo el proceso 0 tiene acceso al disco y a la pantalla, con lo que las operaciones `lee`, `escribe` y el `printf` debe hacerlas este proceso. Asumimos que `M` y `N` son un múltiplo exacto del número de procesos.

Solución:

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

  MPI_Finalize();

  return 0;
}
```

Cuestión 2 (1.2 puntos)

Dada una matriz cuadrada $A \in \mathbb{R}^{n \times n}$, el siguiente código secuencial implementa el cálculo de su ∞-norma, que se define como el máximo de las sumas de los valores absolutos de los elementos de cada fila: $\max\limits_{i=1..n} \left\{ \sum\limits_{j=0}^{n-1} |a_{i,j}| \right\}$.

La función `fabs` de la biblioteca matemática devuelve el valor absoluto de un número de coma flotante.

```
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 versión paralela mediante MPI que realice una distribución de la matriz por bloques de COLUMNAS. La matriz está inicialmente almacenada en P_0 y el resultado debe quedar también en P_0 (razón por la cual se recomienda no paralelizar el último de los bucles). Se puede asumir que el valor de `N` es un múltiplo exacto del número de procesos. En la implementación, para realizar la distribución de la matriz NO hay que usar comunicación colectiva, únicamente primitivas de comunicación punto a punto, pero garantizando que se envía un único mensaje a cada proceso desde P_0. Para otras partes del algoritmo sí es posible utilizar comunicación colectiva.

Nota: se sugiere utilizar la siguiente cabecera para la función paralela, donde `Aloc` es una matriz que se supone ya reservada en memoria, y que puede ser utilizada por la función para almacenar la parte local de la matriz `A` (aunque su tamaño es mayor del necesario y deberán usarse únicamente las primeras columnas de dicha matriz, a partir de la columna 0).

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

Solución: Hay que definir un tipo de datos derivado que cubra `N/p` columnas de la matriz. Las sumas se almacenan en un vector auxiliar `sumloc`, al que se deberá aplicar una reducción tras el cálculo.

```
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) Obtén el coste computacional y de comunicaciones del algoritmo paralelo. Se puede asumir que la operación `fabs` tiene un coste despreciable, así como las comparaciones.

Solución: Denotamos por n el tamaño del problema (`N`). El coste incluye tres etapas:

- Coste del reparto: $(p-1)\cdot\left(t_s + n\cdot\frac{n}{p}\cdot t_w\right)$
- Coste del procesado de $\frac{n}{p}$ columnas: $n\cdot\frac{n}{p} = \frac{n^2}{p}$Flops
- Coste de la reducción (implementación trivial): $(p-1)\cdot(t_s + nt_w) + (p-1)n$

Por tanto, el coste total es aproximadamente: $2pt_s + (n^2+pn)t_w + \frac{n^2}{p} + pn$

Cuestión 3 (1.3 puntos)

Dado el siguiente programa secuencial, en el que el coste computacional y el tratamiento de los argumentos de cada una de las funciones se describen a continuación:

- `leematrices(A, B, C, N)` modifica los tres primeros argumentos y tiene un coste de $3N^2$ Flops.
- `pot(A, exp, N)` modifica únicamente el primer argumento y tiene un coste de $exp \times 2N^3$ Flops.
- `nm = sumnor(A, C, N)` no modifica ningún argumento y tiene un coste de $3N^2$ Flops.
- `nm = norsc(A, nm, N)` no modifica ningún argumento y tiene un coste de N^2 Flops.

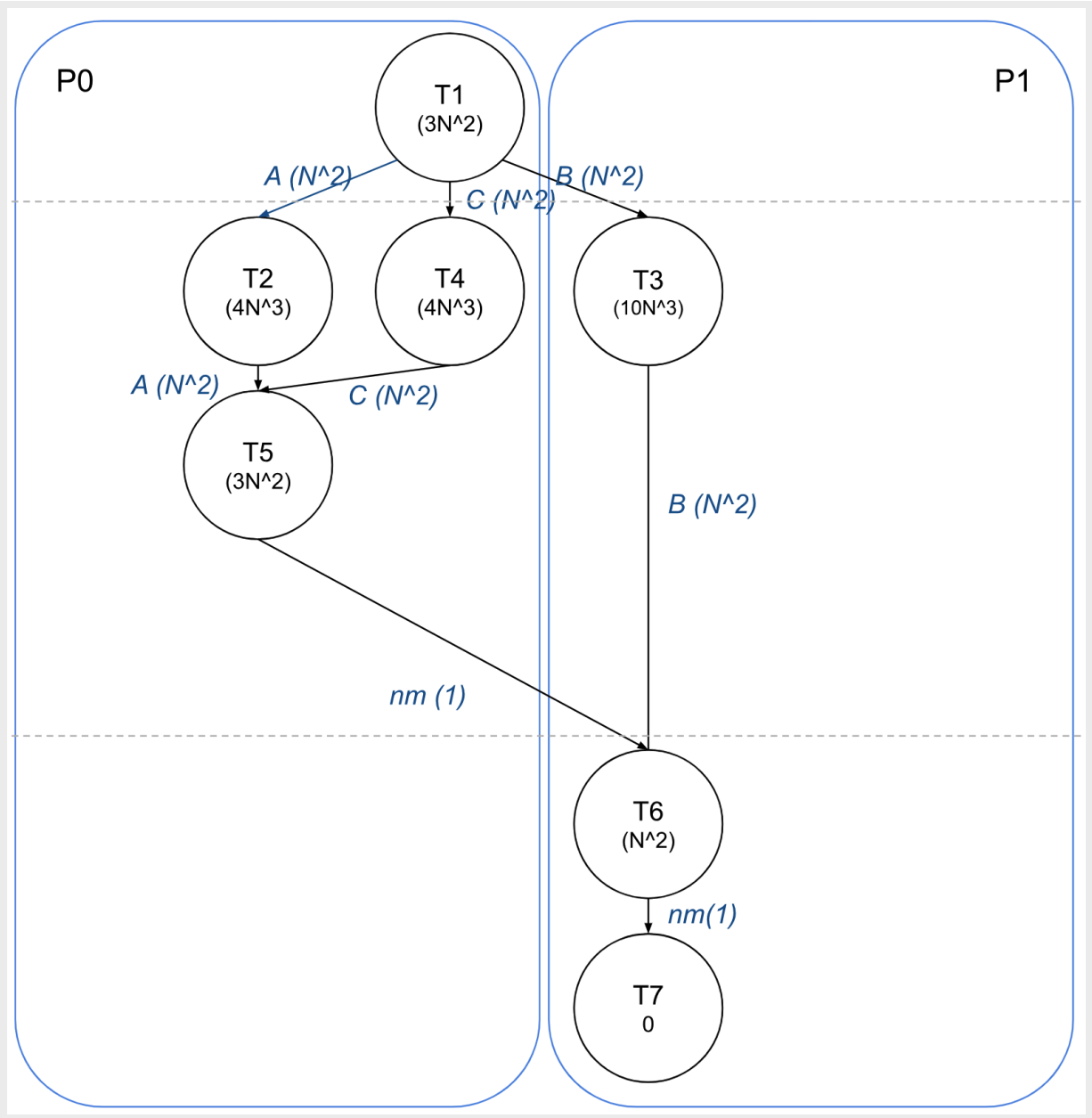
```
int main(int argc, char *argv[]){
  double A[N][N], B[N][N], C[N][N];
  double nm;

  leematrices(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) Realiza el grafo de dependencias del programa secuencial.

Solución:



0.7 p.

(b) Realiza una versión paralela mediante MPI utilizando 2 procesos, maximizando el paralelismo y reduciendo el tiempo de comunicaciones. La llamada a la función `leematrices` deberá hacerse desde el proceso 0 y la salida por pantalla debe realizarse únicamente una vez.

Solución:

```
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) {
    leematrices(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 la expresión del tiempo secuencial y el tiempo paralelo para dos procesos. Calcula también los valores de Speed Up y eficiencia cuando el tamaño del problema (N) tiende a infinito.

Solución:

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

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

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

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

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

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

$$\lim_{N\to\infty} S(N,2) = \lim_{N\to\infty} \frac{18N^3}{10N^3 + 2t_s + N^2 t_w} = 18/10 = 1{,}8$$

$$\lim_{N\to\infty} E(N,p) = 1{,}8/2 = 0{,}9$$