Computación Paralela
Grado en Ingeniería Informática (ETSInf)
Curso 2012-13 ◇ Examen parcial 5/11/2012 ◇ Duración: 1h 30m

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Cuestión 1 (0.5 puntos)

Dada la siguiente función, que busca un valor en un vector, paralelízala usando OpenMP. Al igual que la función de partida, la función paralela deberá terminar la búsqueda tan pronto como se encuentre el elemento buscado.

```
int busqueda(int x[], int n, int valor)
{
   int encontrado=0, i=0;
   while (!encontrado && i<n) {
      if (x[i]==valor) encontrado=1;
      i++;
   }
   return encontrado;
}
```

Cuestión 2 (0.75 puntos)

La infinito-norma de una matriz $A \in \mathbb{R}^{n \times n}$ se define como el máximo de las sumas de los valores absolutos de los elementos de cada fila:

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

El siguiente código secuencial implementa dicha operación para el caso de una matriz cuadrada.

```
#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) Realiza una implementación paralela mediante OpenMP de dicho algoritmo. Justifica la razón por la que introduces cada cambio.

0.2 p. (b) Calcula el coste computacional (en flops) de la versión original secuencial y de la versión paralela desarrollada.
Nota: Se puede asumir que la dimensión de la matriz n es un múltiplo exacto del número de hilos p.
Nota 2: Se puede asumir que el coste de la función `fabs` es de 1 Flop.

0.15 p. (c) Calcula el speedup y la eficiencia del código paralelo ejecutado en p procesadores.

Cuestión 3 (1.25 puntos)
Dada la siguiente función:

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

  calcula_v(n,v);         /* tarea 1 */
  calcula_w(n,w);         /* tarea 2 */
  calcula_z(n,z);         /* tarea 3 */
  calcula_u(n,u,v,w,z);   /* tarea 4 */
  sv = 0;
  for (i=0; i<n; i++)  sv = sv + v[i];         /* tarea 5 */
  sw = 0;
  for (i=0; i<n; i++)  sw = sw + w[i];         /* tarea 6 */
  res = sv+sw;
  for (i=0; i<n; i++)  u[i] = res*u[i];        /* tarea 7 */
  return res;
}
```

Las funciones `calcula_X` tienen como entrada los vectores que reciben como argumentos y con ellos modifican el vector X indicado. Cada función **únicamente** modifica el vector que aparece en su nombre. Por ejemplo, la función `calcula_u` utiliza los vectores `v`, `w` y `z` para realizar unos cálculos que guarda en el vector `u`, pero no modifica ni `v`, ni `w`, ni `z`.

Esto implica, por ejemplo, que las funciones `calcula_v`, `calcula_w` y `calcula_z` son independientes y podrían realizarse simultáneamente. Sin embargo, la función `calcula_u` necesita que hayan terminado las otras, porque usa los vectores que ellas rellenan (`v`,`w`,`z`).

0.2 p. (a) Dibuja el grafo de dependencias de las diferentes tareas.

0.75 p. (b) Paraleliza la función de forma eficiente.

0.3 p. (c) Si suponemos que el coste de todas las funciones `calcula_X` es el mismo y que el coste de los bucles posteriores es despreciable, ¿cuál sería el speedup máximo posible?

Computación Paralela
Grado en Ingeniería Informática (ETSInf)
Curso 2012-13 ◇ Examen final 21/1/2013 ◇ Duración: 3h

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Cuestión 1 (1.25 puntos)
Dada la siguiente función:

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

0.2 p. (a) Indica su coste teórico (en flops).

0.6 p. (b) Paralelízalo usando OpenMP. ¿Por qué lo haces así? Se valorarán más aquellas soluciones que sean más eficientes.

0.3 p. (c) Indica el speedup que podrá obtenerse con p procesadores suponiendo M y N múltiplos exactos de p.

0.15 p. (d) Indica una cota superior del speedup (cuando p tiende a infinito) si no se paralelizara la parte que calcula la suma (es decir, sólo se paraleliza la primera parte y la segunda se ejecuta secuencialmente).

Cuestión 2 (0.75 puntos)
Dada la siguiente función:

```
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) Indica cómo se paralelizaría mediante OpenMP cada uno de los tres bucles. ¿Cuál de las tres formas de paralelizar será la más eficiente y por qué?

0.15 p. (b) Suponiendo que se paraleliza el bucle más externo, indica los costes a priori secuencial y paralelo, en flops, y el speedup suponiendo que el número de hilos (y procesadores) coincide con n.

0.2 p. (c) Añade las líneas de código necesarias para que se muestre en pantalla el número de iteraciones que ha realizado el hilo 0, suponiendo que se paraleliza el bucle más externo.

Cuestión 3 (0.5 puntos)

Paraleliza el siguiente fragmento de código mediante secciones de OpenMP. El segundo argumento de las funciones `fun_` es de entrada-salida, es decir, estas funciones utilizan y modifican el valor de `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;
```

Cuestión 4 (1 punto)

Se quiere paralelizar el siguiente código mediante MPI. Suponemos que se dispone de 3 procesos.

```
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);
```

Todas las funciones leen y modifican ambos argumentos, también los vectores. Suponemos que los vectores `a`, `b` y `c` están almacenados en P_0, P_1 y P_2, respectivamente, y son demasiado grandes para poder ser enviados eficientemente de un proceso a otro.

0.25 p. (a) Dibuja el grafo de dependencias de las diferentes tareas, indicando qué tarea se asigna a cada proceso.

0.75 p. (b) Escribe el código MPI que resuelve el problema.

Cuestión 5 (1 punto)

La infinito-norma de una matriz se define como el máximo de las sumas de los valores absolutos de los elementos de cada fila: $\max_{i=0}^{n-1} \left\{ \sum_{j=0}^{m-1} |a_{i,j}| \right\}$. El siguiente código secuencial implementa dicha operación para el caso de una matriz cuadrada.

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

double infNorm(double A[][N]) {
  int i,j;
  double s,norm;

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

0.5 p. (a) Implementar una versión paralela mediante MPI utilizando operaciones colectivas en la medida de lo posible. Asumir que el tamaño del problema es un múltiplo exacto del número de procesos. La matriz está inicialmente almacenada en P_0 y el resultado debe quedar también en P_0.

Nota: se sugiere utilizar la siguiente cabecera para la función paralela, donde `Alocal` 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`.

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

0.25 p. (b) Obtener el coste computacional y de comunicaciones del algoritmo paralelo. Asumir que la operación `fabs` tiene un coste despreciable, así como las comparaciones.

0.25 p. (c) Calcular el speed-up y la eficiencia cuando el tamaño del problema tiende a infinito.

Cuestión 6 (0.5 puntos)

Sea `A` un array bidimensional de números reales de doble precisión, de dimensión `N`×`N`, define un tipo de datos derivado MPI que permita enviar una submatriz de tamaño 3×3. Por ejemplo, la submatriz que empieza en `A[0][0]` serían los elementos marcados con $\star$:

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

Realiza las correspondientes llamadas para el envío desde P_0 y la recepción en P_1 del bloque de la figura.

Computación Paralela
Grado en Ingeniería Informática (ETSInf)
Curso 2013-14 ◇ Examen parcial 11/11/2013 ◇ Duración: 1h 30m

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Cuestión 1 (1 punto)

Se quiere paralelizar de forma eficiente la siguiente función mediante OpenMP.

```
int cmp(int n, double x[], double y[], int z[])
{
  int i, v, iguales=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) iguales++;
  }
  return iguales;
}
```

0.4 p. (a) Paralelízala utilizando construcciones de tipo `parallel for`.

0.6 p. (b) Paralelízala sin usar ninguna de las siguientes primitivas: `for`, `section`, `reduction`.

Cuestión 2 (0.8 puntos)

Dada la siguiente función:

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

0.4 p. (a) Paralelízala con OpenMP usando dos regiones paralelas.

0.4 p. (b) Paralelízala con OpenMP usando una única región paralela que englobe a todos los bucles. En este caso, ¿tendría sentido utilizar la cláusula `nowait`? Justifica la respuesta.

Cuestión 3 (1.2 puntos)

Teniendo en cuenta la definición de las siguientes funciones:

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

```
/* simetriza una matriz como A+A' */
void simetriza(double A[N][N])
{
  int i,j;
  double suma;
  for (i=0; i<N; i++) {
    for (j=0; j<=i; j++) {
      suma = A[i][j]+A[j][i];
      A[i][j] = suma;
      A[j][i] = suma;
    }
  }
}
```

se pretende paralelizar el siguiente código:

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

0.3 p. (a) Realiza una paralelización basada en los bucles.

0.4 p. (b) Dibuja el grafo de dependencias de tareas, considerando en este caso que las tareas son cada una de las llamadas a `matmult` y `simetriza`. Indica cuál es el grado máximo de concurrencia, la longitud del camino crítico y el grado medio de concurrencia. Nota: para determinar estos últimos valores, es necesario obtener el coste en flops de ambas funciones.

0.5 p. (c) Realiza la paralelización basada en secciones, a partir del grafo de dependencias anterior.

Computación Paralela
Grado en Ingeniería Informática (ETSInf)
Curso 2013-14 ◇ Examen parcial 13/1/2014 ◇ Duración: 1h 30m

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Cuestión 1 (1.2 puntos)

El siguiente programa cuenta el número de ocurrencias de un valor en una matriz.

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

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

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

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

0.8 p. (a) Haz una versión paralela MPI del programa anterior, utilizando operaciones de comunicación colectiva cuando sea posible. La función `leer` deberá ser invocada solo por el proceso 0. Se puede asumir que `DIM` es divisible entre el número de procesos. Nota: hay que escribir el programa completo, incluyendo la declaración de las variables y las llamadas necesarias para iniciar y cerrar MPI.

0.4 p. (b) Calcula el tiempo de ejecución paralelo, suponiendo que el coste de comparar dos números reales es de 1 flop. Nota: para el coste de las comunicaciones, suponer una implementación sencilla de las operaciones colectivas.

Cuestión 2 (1 punto)

En un programa MPI, un vector x, de dimensión n, se encuentra distribuido de forma cíclica entre p procesos, y cada proceso guarda en el array `xloc` los elementos que le corresponden.

Implementa la siguiente función, de forma que al ser invocada por todos los procesos, realice las comunicaciones necesarias para que el proceso 0 recoja en el array `x` una copia del vector completo, con los elementos correctamente ordenados según el índice global.

Cada proceso del 1 al $p-1$ deberá enviar al proceso 0 todos sus elementos mediante un único mensaje.

```
void comunica_vector(double xloc[], int n, int p, int rank, double x[])
/* rank es el indice del proceso local                                  */
/* Esta funcion asume que n es múltiplo exacto de p                     */
```

Cuestión 3 (0.8 puntos)

El siguiente fragmento de código es incorrecto (desde el punto de vista semántico, no porque haya un error en los argumentos). Indica por qué y propón dos soluciones distintas (si las dos respuestas son ligeras variaciones de la misma solución, se tendrá en cuenta solo una).

```
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, MPI_STATUS_IGNORE);
```

Computación Paralela
Grado en Ingeniería Informática (ETSInf)
Curso 2013-14 ◇ Examen final 27/1/14 ◇ Bloque OpenMP ◇ Duración: 1h 30m

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Cuestión 1 (1 punto)

La siguiente función normaliza los valores de un vector de números reales positivos de forma que los valores finales queden entre 0 y 1, utilizando el máximo y el mínimo.

```
void normalize(double *a, int n)
{
  double max, min, factor;
  int i;

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

0.75 p. (a) Paraleliza el programa con OpenMP de la manera más eficiente posible, mediante una única región paralela. Suponemos un valor de `n` muy grande y se quiere que la paralelización funcione para un número arbitrario de hilos.

0.25 p. (b) Incluye el código necesario para que se imprima una sola vez el número de hilos utilizados.

Cuestión 2 (1 punto)

Dado el siguiente fragmento de código, donde el vector de índices `ind` contiene valores enteros entre 0 y $m-1$ (siendo m la dimensión de `x`), posiblemente con repeticiones:

```
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) Realiza una implementación paralela mediante OpenMP, en la que se reparten las iteraciones del bucle externo.

0.25 p. (b) Realiza una implementación paralela mediante OpenMP, en la que se reparten las iteraciones del bucle interno.

0.25 p. (c) Para la implementación del apartado (a), indica si cabe esperar que haya diferencias de prestaciones dependiendo de la planificación empleada. Si es así, ¿qué planificaciones serían mejores y por qué?

Cuestión 3 (1 punto)

En la siguiente función, `T1`, `T2`, `T3` modifican `x`, `y`, `z`, respectivamente.

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

   T1(x,n);    /* Tarea 1 */
   T2(y,n);    /* Tarea 2 */
   T3(z,n);    /* Tarea 3 */

   /* Tarea 4 */
   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;
   }

   /* Tarea 5 */
   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;
   }

   /* Tarea 6 */
   a=s1/s2;
   res=0;
   for (i=0; i<n; i++) res+=a*z[i];
   return res;
}
```

0.2 p. (a) Dibuja el grafo de dependencia de las tareas.

0.5 p. (b) Realiza una paralelización mediante OpenMP a nivel de tareas (no de bucles), basándote en el grafo de dependencias.

0.3 p. (c) Indica el coste a priori del algoritmo secuencial, el del algoritmo paralelo y el speedup resultante. Supón que el coste de las tareas 1, 2 y 3 es de $2n^2$ flops cada una.

Cuestión 4 (0.8 puntos)

Queremos medir la latencia de un anillo de `p` procesos en MPI, entendiendo por latencia el tiempo que tarda un mensaje de tamaño 0 en circular entre todos los procesos. Un anillo de `p` procesos MPI funciona de la siguiente manera: P_0 envía el mensaje a P_1, cuando éste lo recibe, lo renvía a P_2, y así sucesivamente hasta que llega a P_{p-1} que lo enviará a P_0. Escribe un programa MPI que implemente el esquema de comunicación previo y muestre la latencia. Es recomendable hacer que el mensaje dé varias vueltas al anillo, y luego sacar el tiempo medio por vuelta, para obtener una medida más fiable.

Cuestión 5 (1.2 puntos)

El siguiente programa paralelo MPI debe calcular la suma de dos matrices A y B de dimensiones $M \times N$ utilizando una distribución cíclica de filas, suponiendo que el número de procesos `p` es divisor de M y teniendo en cuenta que P_0 tiene almacenadas inicialmente las matrices A y 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) leer(A,B);

/* (a) Reparto cíclico de filas de A y B */
/* (b) Cálculo local de Al+Bl */
/* (c) Recogida de resultados en el proceso 0 */

MPI_Finalize();
```

0.5 p. (a) Implementa el reparto cíclico de filas de las matrices A y B, siendo `Al` y `Bl` las matrices locales. Para realizar esta distribución **debes** o bien definir un nuevo tipo de dato de MPI o bien usar comunicaciones colectivas.

0.2 p. (b) Implementa el cálculo local de la suma `Al`+`Bl`, almacenando el resultado en `Al`.

0.5 p. (c) Escribe el código necesario para que P_0 almacene en `A` la matriz $A + B$. Para ello, P_0 debe recibir del resto de procesos las matrices locales `Al` obtenidas en el apartado anterior.

Cuestión 6 (1 punto)

Se quiere implementar el cálculo de la ∞-norma de una matriz cuadrada, que se obtiene como el máximo de las sumas de los valores absolutos de los elementos de cada fila, $\max_{i=0}^{n-1} \left\{ \sum_{j=0}^{n-1} |a_{i,j}| \right\}$. Para ello, se propone un esquema maestro-trabajadores. A continuación, se muestra la función correspondiente al maestro (el proceso con identificador 0). La matriz se almacena por filas en

un array uni-dimensional, y suponemos que es muy dispersa (tiene muchos ceros), por lo que el maestro envía únicamente los elementos no nulos (función `comprime`).

```
int comprime(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 maestro(double *A,int n)
{
  double buf[n];
  double norma=0.0,valor;
  int fila,completos=0,size,i,k;
  MPI_Status status;
  MPI_Comm_size(MPI_COMM_WORLD,&size);
  for (fila=0;fila<size-1;fila++) {
    if (fila<n) {
      k = comprime(A, n, fila, buf);
      MPI_Send(buf, k, MPI_DOUBLE, fila+1, TAG_FILA, MPI_COMM_WORLD);
    } else
      MPI_Send(buf, 0, MPI_DOUBLE, fila+1, TAG_END, MPI_COMM_WORLD);
  }
  while (completos<n) {
    MPI_Recv(&valor, 1, MPI_DOUBLE, MPI_ANY_SOURCE, TAG_RESU,
             MPI_COMM_WORLD, &status);
    if (valor>norma) norma=valor;
    completos++;
    if (fila<n) {
      k = comprime(A, n, fila, buf);
      fila++;
      MPI_Send(buf, k, MPI_DOUBLE, status.MPI_SOURCE, TAG_FILA, MPI_COMM_WORLD);
    } else
      MPI_Send(buf, 0, MPI_DOUBLE, status.MPI_SOURCE, TAG_END, MPI_COMM_WORLD);
  }
  return norma;
}
```

Implementa la parte de los procesos trabajadores, completando la siguiente función:

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

Notas: Para el valor absoluto se puede usar

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

Recuerda que `MPI_Status` contiene, entre otros, los campos `MPI_SOURCE` y `MPI_TAG`.

Computación Paralela
Grado en Ingeniería Informática (ETSInf)
Curso 2014-15 ◇ Examen parcial 3/11/2014 ◇ Duración: 1h 30m

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Cuestión 1 (1 punto)
Dada la siguiente función:

```
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) Paralelízala eficientemente mediante OpenMP.

0.3 p. (b) Calcula el número de flops de la función inicial y de la función paralelizada.

0.1 p. (c) Determina el speedup y la eficiencia.

Cuestión 2 (1 punto)
Dado el siguiente fragmento de código:

```
minx = minimo(x,n);       /* T1 */
maxx = maximo(x,n);       /* T2 */
calcula_z(z,minx,maxx,n); /* T3 */
calcula_y(y,x,n);         /* T4 */
calcula_x(x,y,n);         /* T5 */
calcula_v(v,z,x);         /* T6 */
```

0.3 p. (a) Dibuja el grafo de dependencias de las tareas, teniendo en cuenta que las funciones `minimo` y `maximo` no modifican sus argumentos, mientras que las demás funciones modifican sólo su primer argumento.

0.4 p. (b) Paraleliza el código mediante OpenMP.

0.3 p. (c) Si el coste de las tareas es de n flops, excepto el de la tarea 4 que es de $2n$ flops, indica la longitud del camino crítico y el grado medio de concurrencia. Obtén el speedup y la eficiencia de la implementación del apartado anterior, si se ejecutara con 5 procesadores.

Cuestión 3 (1 punto)

Dada la siguiente función:

```
int funcion(int n, double v[])
{
  int i,pos_max=-1;
  double suma,norma,aux,max=-1;

  suma = 0;
  for (i=0;i<n;i++)
    suma = suma + v[i]*v[i];
  norma = sqrt(suma);

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

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

0.6 p. (a) Paralelízala con OpenMP, usando una única región paralela.

0.2 p. (b) ¿Tendría sentido poner una cláusula `nowait` a alguno de los bucles? ¿Por qué? (Justifica cada bucle separadamente.)

0.2 p. (c) ¿Qué añadirías para garantizar que en todos los bucles las iteraciones se reparten de 2 en 2 entre los hilos?

Computación Paralela
Grado en Ingeniería Informática (ETSInf)
Curso 2014-15 ◇ Examen parcial 12/1/2015 ◇ Duración: 1h 30m

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Cuestión 1 (1 punto)

La siguiente función muestra por pantalla el máximo de un vector `v` de `n` elementos y su posición:

```
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("Máximo: %f. Posición: %d\n", max, posmax);
}
```

Escribe una versión paralela MPI con la siguiente cabecera, donde los argumentos `rank` y `np` han sido obtenidos mediante `MPI_Comm_rank()` y `MPI_Comm_size()`, respectivamente.

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

La función debe asumir que el array `v` del proceso 0 contendrá inicialmente el vector, mientras que en el resto de procesos dicho array podrá usarse para almacenar la parte local que corresponda. Deberán comunicarse los datos necesarios de forma que el cálculo del máximo se reparta de forma equitativa entre todos los procesos. Finalmente, sólo el proceso 0 debe mostrar el mensaje por pantalla. Se deben utilizar operaciones de comunicación punto a punto (no colectivas).

Nota: se puede asumir que `n` es múltiplo del número de procesos.

Cuestión 2 (1 punto)

0.6 p. (a) Implementa mediante comunicaciones colectivas una función en MPI que sume dos matrices cuadradas `a` y `b` y deje el resultado en `a`, teniendo en cuenta que las matrices `a` y `b` se encuentran almacenadas en la memoria del proceso P_0 y el resultado final también deberá estar en P_0. Supondremos que el número de filas de las matrices (`N`, constante) es divisible entre el número de procesos. La cabecera de la función es:

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

0.4 p. (b) Determina el tiempo paralelo, el speed-up y la eficiencia de la implementación detallada en el apartado anterior, indicando brevemente para su cálculo cómo se realizarían cada una de las operaciones colectivas (número de mensajes y tamaño de cada uno). Puedes asumir una implementación sencilla (no óptima) de estas operaciones de comunicación.

Cuestión 3 (1 punto)

Implementa una función que, dada una matriz A de $N \times N$ números reales y un índice k (entre 0 y $N-1$), haga que la fila k y la columna k de la matriz se comuniquen desde el proceso 0 al resto de procesos (sin comunicar ningún otro elemento de la matriz).

La cabecera de la función sería así:

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

Deberás crear y usar un tipo de datos que represente una columna de la matriz.

No es necesario que se envíen juntas la fila y la columna. Se pueden enviar por separado.

Computación Paralela
Grado en Ingeniería Informática (ETSInf)
Curso 2014-15 ◇ Examen final 21/1/15 ◇ Bloque OpenMP ◇ Duración: 1h 30m

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Cuestión 1 (1 punto)
Dada esta función en C:

```
double fun( int n, double a[], double b[] )
{
  int i,ac,bc;
  double asuma,bsuma,cota;

  asuma = 0; bsuma = 0;
  for (i=0; i<n; i++)
    asuma += a[i];
  for (i=0; i<n; i++)
    bsuma += b[i];
  cota = (asuma + bsuma) / 2.0 / n;

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

0.7 p. (a) Paralelízala eficientemente mediante directivas OpenMP (sin alterar el código ya existente).

0.3 p. (b) Indica el coste a priori en flops, tanto secuencial como paralelo (considerando sólo operaciones en coma flotante y asumiendo que una comparación implica una resta). ¿Cuál sería el speed-up y la eficiencia para p procesadores? Asumir que n es un múltiplo exacto de p.

Cuestión 2 (1.3 puntos)
Se quiere paralelizar el siguiente programa mediante OpenMP, donde `genera` es una función previamente definida en otro lugar.

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

```
double compara(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;
}
```

```
/* fragmento del programa principal (main) */
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 = compara(a,b,n);    /* T4 */
y = compara(a,c,n);    /* T5 */
z = compara(c,b,n);    /* T6 */
w = x+y+z;             /* T7 */
printf("w:%f\n", w);
```

0.2 p. (a) Paraleliza el código de forma eficiente a nivel de bucles.

0.3 p. (b) Dibuja el grafo de dependencias de tareas, según la numeración de tareas indicada en el código.

0.5 p. (c) Paraleliza el código de forma eficiente a nivel de tareas, a partir del grafo de dependencias anterior.

0.3 p. (d) Obtén el tiempo secuencial (asume que una llamada a las funciones `genera` y `fabs` cuesta 1 flop) y el tiempo paralelo para cada una de las dos versiones asumiendo que hay 3 procesadores. Calcular el speed-up en cada caso.

Cuestión 3 (0.7 puntos)

Paraleliza mediante OpenMP el siguiente fragmento de código, donde `f` y `g` son dos funciones que toman 3 argumentos de tipo `double` y devuelven un `double`, y `fabs` es la función estándar que devuelve el valor absoluto de un `double`.

```
double x,y,z,w=0.0;
double x0=1.0,y0=3.0,z0=2.0;     /* punto inicial */
double dx=0.01,dy=0.01,dz=0.01;  /* incrementos */

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

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

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

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

Cuestión 4 (1 punto)

Dada la siguiente función, donde suponemos que las funciones `T1`, `T3` y `T4` tienen un coste de n y las funciones `T2` y `T5` de $2n$, siendo n un valor constante.

```
double ejemplo(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) Dibuja el grafo de dependencias y calcula el coste secuencial.

0.6 p. (b) Paralelízalo usando MPI con **dos** procesos. Ambos procesos invocan la función con el mismo valor de los argumentos `i`, `j` (no es necesario comunicarlos). El valor de retorno de la función debe ser correcto en el proceso 0 (no es necesario que esté en ambos procesos).

0.2 p. (c) Calcula el tiempo de ejecución paralelo (cálculo y comunicaciones) y el speedup con dos procesos.

Cuestión 5 (1 punto)

Dado el siguiente fragmento de código de un programa paralelo:

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

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

Escribe el código necesario para definir un tipo de datos MPI que permita que una columna cualquiera de `A` se pueda enviar o recibir con un sólo mensaje. Utiliza el tipo de datos para hacer que el proceso 0 le envíe la columna `j` al proceso `proc`, quien la recibirá sobre el array `col`, le cambiará el signo a los elementos, y se la devolverá al proceso 0, que la recibirá de nuevo sobre la columna `j` de la matriz `A`.

Cuestión 6 (1 punto)

0.4 p. (a) El siguiente fragmento de código utiliza primitivas de comunicación punto a punto para un patrón de comunicación que puede efectuarse mediante una única operación colectiva.

```
#define TAG 999
int i, k, sz, rank;
float z[LNZ], zfull[LNFULL];     /* suponemos LNFULL>=LNZ*sz */
MPI_Comm_size(MPI_COMM_WORLD, &sz);
MPI_Comm_rank(MPI_COMM_WORLD, &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, MPI_COMM_WORLD,
             MPI_STATUS_IGNORE);
  }
} else {
  MPI_Send(z, LNZ, MPI_FLOAT, 0, TAG, MPI_COMM_WORLD);
}
```

Escribe la llamada a la primitiva MPI de la operación de comunicación colectiva equivalente.

0.6 p. (b) Dada la siguiente llamada a una primitiva de comunicación colectiva:

```
long int factor, producto;
MPI_Allreduce(&factor, &producto, 1, MPI_LONG, MPI_PROD, MPI_COMM_WORLD);
```

Escribe un fragmento de código equivalente (debe realizar la misma comunicación y operaciones aritméticas asociadas) pero utilizando únicamente primitivas de comunicación punto a punto.

Computación Paralela
Grado en Ingeniería Informática (ETSInf)
Curso 2015-16 ◇ Examen parcial 9/11/2015 ◇ Duración: 1h 30m

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Cuestión 1 (1.2 puntos)
Dada la siguiente función:

```
#define DIM 6000
#define PASOS 6

double funcion1(double A[DIM][DIM], double b[DIM], double x[DIM])
{
  int i, j, k, n=DIM, pasos=PASOS;
  double max=-1.0e308, q, s, x2[DIM];
  for (k=0;k<pasos;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) Paraleliza el código usando OpenMP. ¿Por qué lo haces así? Se valorarán más aquellas soluciones que sean más eficientes.

0.25 p. (b) Indica el coste teórico (en flops) que tendría una iteración del bucle `k` del código secuencial.

0.25 p. (c) Considerando una única iteración del bucle `k` (`PASOS`=1), indica el speedup y la eficiencia que podrá obtenerse con p hilos, suponiendo que hay tantos núcleos/procesadores como hilos y que `DIM` es un múltiplo exacto de p.

Cuestión 2 (1 punto)
La siguiente función procesa una serie de transferencias bancarias. Cada transferencia tiene una cuenta origen, una cuenta destino y una cantidad de dinero que se mueve de la cuenta origen a la cuenta destino. La función actualiza la cantidad de dinero de cada cuenta (array `saldos`), y además devuelve la cantidad máxima que se transfiere en una sola operación.

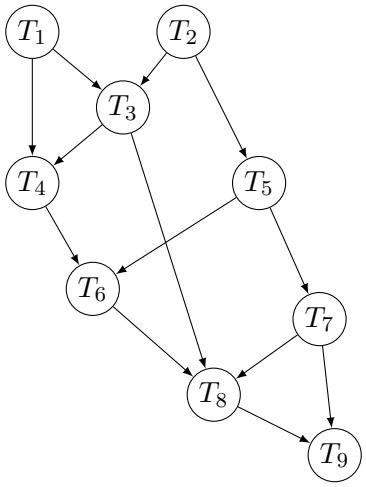
```
double transferencias(double saldos[], int origenes[], int destinos[],
       double cantidades[], int n)
{
   int i, i1, i2;
   double dinero, maxtransf=0;

   for (i=0; i<n; i++) {
      /* Procesar transferencia i: La cantidad transferida es cantidades[i],
       * que se mueve de la cuenta origenes[i] a la cuenta destinos[i]. Se
       * actualizan los saldos de ambas cuentas, y la cantidad maxima */
      i1 = origenes[i];
      i2 = destinos[i];
      dinero = cantidades[i];
      saldos[i1] -= dinero;
      saldos[i2] += dinero;
      if (dinero>maxtransf) maxtransf = dinero;
   }
   return maxtransf;
}
```

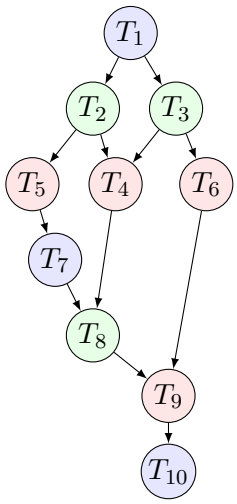
0.7 p. (a) Paraleliza la función de forma eficiente mediante OpenMP.

0.3 p. (b) Modifica la solución del apartado anterior para que se imprima el índice de la transferencia con más dinero.

Cuestión 3 (0.8 puntos)

Dados los siguientes grafos de dependencias de tareas:





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) Para el grafo de la izquierda, indica qué secuencia de nodos del grafo constituye el camino crítico. Calcula la longitud del camino crítico y el grado medio de concurrencia. Nota: no se ofrece información de costes, se puede suponer que todas las tareas tienen el mismo coste.

0.4 p. (b) Repite el apartado anterior para el grafo de la derecha. Nota: en este caso el coste de cada tarea viene dado en flops (para un tamaño de problema n) según la tabla mostrada.

Computación Paralela
Grado en Ingeniería Informática (ETSInf)
Curso 2015-16 ◇ Examen parcial 15/1/2016 ◇ Duración: 1h 30m

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Cuestión 1 (1 punto)

0.8 p. (a) Implementa una función para distribuir una matriz cuadrada entre los procesos de un programa MPI, con la siguiente cabecera:

```
void comunica(double A[N][N], double Aloc[][N], int proc_fila[N], int root)
```

La matriz `A` se encuentra inicialmente en el proceso `root`, y debe distribuirse por filas entre los procesos, de manera que cada fila `i` debe ir al proceso `proc_fila[i]`. El contenido del array `proc_fila` es válido en todos los procesos. Cada proceso (incluido el `root`) debe almacenar las filas que le correspondan en la matriz local `Aloc`, ocupando las primeras filas (o sea, si a un proceso se le asignan `k` filas, éstas deben quedar almacenadas en las primeras `k` filas de `Aloc`).

Ejemplo para 3 procesos:

`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_fila`

0
2
0
1
1

`Aloc` en P_0

11	12	13	14	15
31	32	33	34	35

`Aloc` en P_1

41	42	43	44	45
51	52	53	54	55

`Aloc` en P_2

21	22	23	24	25

0.2 p. (b) En un caso general, ¿se podría usar el tipo de datos *vector* de MPI (`MPI_Type_vector`) para enviar a un proceso todas las filas que le tocan mediante un solo mensaje? Si se puede, escribe las instrucciones para definirlo. Si no se puede, justifica por qué.

Cuestión 2 (1 punto)

La siguiente función calcula el producto escalar de dos vectores:

```
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) Implementa una función para realizar el producto escalar en paralelo mediante MPI, utilizando en la medida de lo posible operaciones colectivas. Se supone que los datos están disponibles en el proceso P_0 y que el resultado debe quedar también en P_0 (el valor de

retorno de la función solo es necesario que sea correcto en P_0). Se puede asumir que el tamaño del problema `n` es exactamente divisible entre el número de procesos.

Nota: a continuación se muestra la cabecera de la función a implementar, incluyendo la declaración de los vectores locales (suponemos que `MAX` es suficientemente grande para cualquier valor de n y número de procesos).

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

0.3 p. (b) Calcula el speed-up. Si para un tamaño suficientemente grande de mensaje, el tiempo de envío por elemento fuera equivalente a 0.1 flops, ¿qué speed-up máximo se podría alcanzar cuando el tamaño del problema tiende a infinito y para un valor suficientemente grande de procesos?

0.2 p. (c) Modifica el código anterior para que el valor de retorno sea el correcto en todos los procesos.

Cuestión 3 (1 punto)

Se desea distribuir entre 4 procesos una matriz cuadrada de orden $2N$ ($2N$ filas por $2N$ columnas) definida a bloques como

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

donde cada bloque A_{ij} corresponde a una matriz cuadrada de orden N, de manera que se quiere que el proceso P_0 almacene localmente la matriz A_{00}, P_1 la matriz A_{01}, P_2 la matriz A_{10} y P_3 la matriz A_{11}.

Por ejemplo, la siguiente matriz con $N = 2$ quedaría distribuida como se muestra:

$$A = \left(\begin{array}{cc|cc} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ \hline 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{array}\right)$$

En P_0: $\begin{pmatrix} 1 & 2 \\ 5 & 6 \end{pmatrix}$ En P_1: $\begin{pmatrix} 3 & 4 \\ 7 & 8 \end{pmatrix}$

En P_2: $\begin{pmatrix} 9 & 10 \\ 13 & 14 \end{pmatrix}$ En P_3: $\begin{pmatrix} 11 & 12 \\ 15 & 16 \end{pmatrix}$

0.8 p. (a) Implementa una función que realice la distribución mencionada, definiendo para ello el tipo de datos MPI necesario. La cabecera de la función sería:

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

donde `A` es la matriz inicial, almacenada en el proceso 0, y `B` es la matriz local donde cada proceso debe guardar el bloque que le corresponda de A.

Nota: se puede asumir que el número de procesos del comunicador es 4.

0.2 p. (b) Calcula el tiempo de comunicaciones.

Computación Paralela
Grado en Ingeniería Informática (ETSInf)
Curso 2015-16 ◇ Examen final 26/1/16 ◇ Bloque OpenMP ◇ Duración: 1h 30m

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Cuestión 1 (0.8 puntos)
Dada la siguiente función:

```
double funcion(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) Paralelízala con OpenMP de forma eficiente. Utiliza si es posible una sola región paralela.

0.2 p. (b) Haz que cada hilo muestre una línea con su identificador y el número de iteraciones del primer bucle `i` que ha realizado.

0.2 p. (c) En el primer bucle paralelizado, justifica si cabe esperar diferencias en prestaciones entre las siguientes planificaciones para el bucle: `schedule(static)`, `schedule(static,1)`, `schedule(dynamic)`.

Cuestión 2 (1.2 puntos)
Dada la siguiente función:

```
double f(int n, double vec[])
{
  double res, v[NMAX], w[NMAX];
  A(n,v,vec);          /* Copia vec en v, coste n */
  B(n,w,vec);          /* Copia vec en w, coste n */
  C(n,vec);            /* Actualiza vec, coste n  */
```

```
  D(n,vec);            /* Actualiza vec, coste n  */
  E(n,v);              /* Actualiza v, coste 3n   */
  F(n,w);              /* Actualiza w, coste 2n   */
  res = G(n,vec,v,w);  /* Calcula res, coste 3n   */
  return res;
}
```

0.5 p. (a) Dibuja el grafo de dependencias, indicando el grado máximo de concurrencia, la longitud del camino crítico y el grado medio de concurrencia.

0.5 p. (b) Paralelízala con OpenMP.

0.2 p. (c) Calcula el speedup y la eficiencia máximos si se ejecuta con 2 hilos.

Cuestión 3 (1 punto)

Sea la siguiente función:

```
double funcion(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) Paraleliza el código anterior usando para ello OpenMP. Explica las decisiones que tomes. Se valorarán más aquellas soluciones que sean más eficientes.

0.2 p. (b) Calcula el coste secuencial, el coste paralelo, el speedup y la eficiencia que podrán obtenerse con p procesadores suponiendo que N es divisible entre p.

Cuestión 4 (1 punto)

Sea el código secuencial:

```
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) Implementa una versión paralela equivalente utilizando MPI, teniendo en cuenta los siguientes aspectos:

- El proceso P_0 obtiene inicialmente la matriz `A`, realizando la llamada `leer(A)`, siendo `leer` una función ya implementada.
- La matriz `A` se debe distribuir por bloques de filas entre todos los procesos.
- Finalmente P_0 debe contener el resultado en la matriz `A`.
- Utiliza comunicaciones colectivas siempre que sea posible.

Se supone que N es divisible entre el número de procesos y que la declaración de las matrices usadas es

```
double A[N][N], B[N][N];  /* B: matriz distribuida */
```

0.2 p. (b) Calcula el speedup y la eficiencia.

Cuestión 5 (0.8 puntos)

Desarrolla una función que sirva para enviar una submatriz desde el proceso 0 al proceso 1, donde quedará almacenada en forma de vector. Se debe utilizar un nuevo tipo de datos, de forma que se utilice un único mensaje. Recuérdese que las matrices en C están almacenadas en memoria por filas.

La cabecera de la función será así:

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

Nota: se asume que `m*n<=MAX` y que la submatriz a enviar empieza en el elemento `A[0][0]`.

Ejemplo con $M = 4$, $N = 5$, $m = 3$, $n = 2$:

`A` (en P_0)						`v` (en P_1)
1	2	0	0	0		
3	4	0	0	0	$\rightarrow$	1 2 3 4 5 6
5	6	0	0	0		
0	0	0	0	0		

Cuestión 6 (1.2 puntos)

Dado el siguiente programa secuencial:

```
int calcula_valor(int vez);  /* función dada, definida en otro sitio */

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

  printf("Número de iteraciones: ");
  scanf("%d",&n);

  min = 100000;
  for (i = 1; i <= n; i++) {
    val = calcula_valor(i);
    if (val < min) min = val;
  }
  printf("Mínimo: %d\n",min);

  return 0;
}
```

0.8 p. (a) Paralelízalo mediante MPI usando operaciones de comunicación punto a punto. La entrada y salida de datos debe hacerla únicamente el proceso 0. Puede asumirse que n es un múltiplo exacto del número de procesos.

0.4 p. (b) ¿Qué se cambiaría para usar operaciones de comunicación colectivas allá donde sea posible?