Parciales OpenMp 2021-2022

OpenMp 2021: Ejercicio 1

Dada la siguiente función:

```
void f(double A[N][N], double B[N][N], double C[N][N], double x[N], double z[N]) {
int i,j;
double sumz;
for (i=0;i<N;i++) {
      sumz = 0;
      for (j=i;j<N;j++)
            C[i][j] = A[i][j] * x[j];
      for (j=0;j<N;j++)
            sumz += B[i][j] * x[j];
      z[i] = sumz;
      }
}
```

a) Haz una versión paralela basada en la paralelización del bucle externo.

b) Modifica el apartado anterior para que se muestre una única vez el número de hilos que intervienen en la ejecución del bucle paralelo.

Solución:

a) Justo delante del bucle i, incluiremos:

```
#pragma omp parallel for private (j, sumz)
```

b)

```
...
double sumz;
#pragma omp parallel
#pragma omp master
printf("Numero de hilos: %d\n", omp_get_num_threads());
#pragma omp parallel for private (j, sumz)
for (i=0;i<N;i++) {
...
```

Otra solución

```
#pragma omp parallel private (j, sumz, id)
{
#pragma omp master
 printf("Numero de hilos: %d\n",omp_get_num_threads());
#pragma omp for
  for (i=0;i<N;i++) {
...
}
}
```

OpenMp 2021: Ejercicio 1

c)Haz una versión paralela basada en la paralelización de los dos bucles internos, usando una sola región paralela. Considera la conveniencia de usar la cláusula nowait y, tanto si la usas como si no, justifica porqué.

```
void f(double A[N][N], double B[N][N], double C[N][N], double x[N], double z[N]) {
int i,j;
double sumz;
for (i=0;i<N;i++) {
      sumz = 0;
      for (j=i;j<N;j++)
            C[i][j] = A[i][j] * x[j];
      for (j=0;j<N;j++)
            sumz += B[i][j] * x[j];
      z[i] = sumz;
      }
}
```

```
for (i=0;i<N;i++) {
      sumz = 0;
      #pragma omp parallel
       {
         #pragma omp for nowait
          for (j=i;j<N;j++)
              C[i][j] = A[i][j] * x[j];
         #pragma omp for reduction(+:sumz)
          for (j=0;j<N;j++)
              sumz += B[i][j] * x[j];
       }
      z[i] = sumz;
}
```

Se usa la cláusula **nowait** en el primer bucle, porque dicho bucle es independiente del segundo.

OpenMp 2021: Ejercicio 1

d) Calcula el coste secuencial y el coste paralelo (incluye en ambos casos el desarrollo completo) suponiendo que se paralelizara únicamente el segundo de los bucles internos. Calcula también en el mismo supuesto el Speed-up y la Eficiencia.

```
void f(…) {
int i,j;
double sumz;
for (i=0;i<N;i++) {
    sumz = 0;
    for (j=i;j<N;j++)
        C[i][j] = A[i][j] * x[j];
    for (j=0;j<N;j++)
        sumz += B[i][j] * x[j];
    z[i] = sumz;
    }
}
```

$$t(N) = \sum_{i=0}^{N-1} \left(\sum_{j=i}^{N-1} 1 + \sum_{j=0}^{N-1} 2 \right) \approx \sum_{i=0}^{N-1} (N - i + 2N) \approx 3N^2 - \frac{N^2}{2} = \frac{5N^2}{2} \text{ flops}$$

$$t(N,p) = \sum_{i=0}^{N-1} \left(\sum_{j=i}^{N-1} 1 + \sum_{j=0}^{\frac{N}{p}-1} 2 \right) \approx \sum_{i=0}^{N-1} \left(N - i + \frac{2N}{p} \right) \approx$$

$$\approx N^2 - \frac{N^2}{2} + \frac{2N^2}{p} = \frac{N^2}{2} + \frac{2N^2}{p} = \frac{p+4}{2p} N^2 \text{ flops}$$

$$S(N,p) = \frac{\frac{5}{2}N^2}{\frac{p+4}{2p}N^2} = \frac{5p}{p+4}$$

$$E(N,p) = \frac{\frac{5p}{p+4}}{p} = \frac{5}{p+4}$$

OpenMp 2021: Ejercicio 2

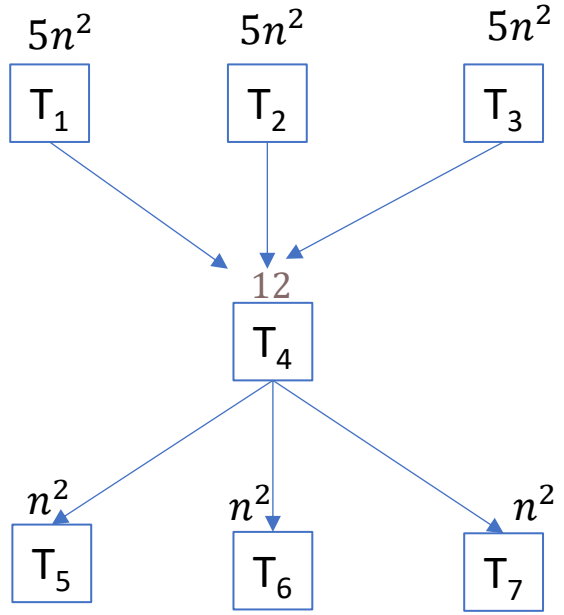
Dada la siguiente función, donde Image es un tipo de datos predefinido:

```
void transform(int n, Image im1, Image im2, float weights[3], float factor)
{
    weights[0] = channel_r(im1,im2,n); /* Tarea T1, coste 5*n^2 flops */
    weights[1] = channel_g(im1,im2,n); /* Tarea T2, coste 5*n^2 flops */
    weights[2] = channel_b(im1,im2,n); /* Tarea T3, coste 5*n^2 flops */
    factor *= combine(weights); /* Tarea T4; coste 12 flops */
    weights[0] += adjust_r(im2,n,factor); /* Tarea T5, coste n^2 flops */
    weights[1] += adjust_g(im2,n,factor); /* Tarea T6, coste n^2 flops */
    weights[2] += adjust_b(im2,n,factor); /* Tarea T7, coste n^2 flops */
}
```

Ninguna de las funciones modifica sus argumentos.

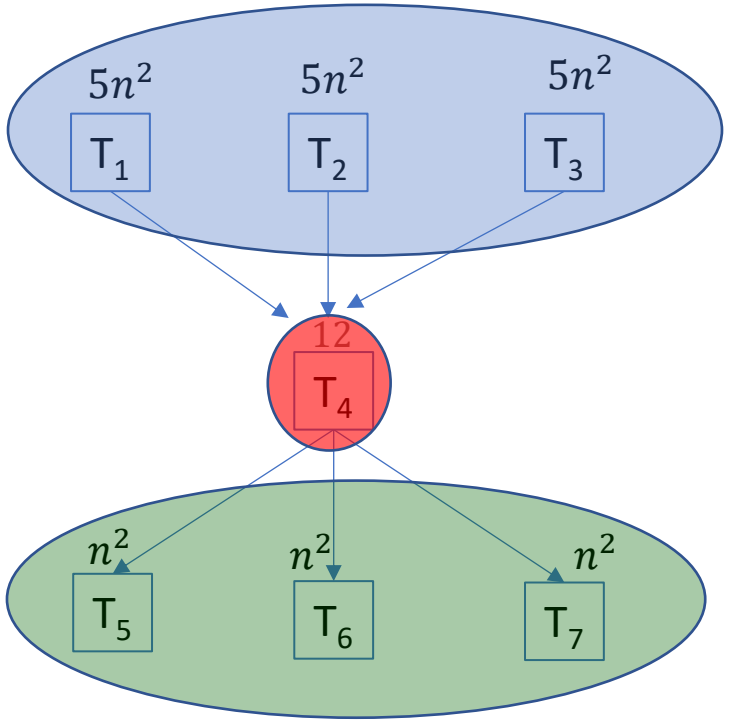
a) Dibuja el grafo de dependencias de datos entre las tareas.

Solución:
a)



OpenMp 2021: Ejercicio 2

b) Implementa una versión paralela mediante OpenMP utilizando una sola región paralela.




```
void transform_par(int n, Image im1, Image im2, float weights[3], float factor){
#pragma omp parallel
{
    #pragma omp sections
    {
        #pragma omp section
            T1
        #pragma omp section
            T2
        #pragma omp section
            T3
    }
    #pragma omp single
        T4
    #pragma omp sections
    {
        #pragma omp section
            T5
        #pragma omp section
            T6
        #pragma omp section
            T7
    }
}
}
```

OpenMp 2021: Ejercicio 3

La siguiente función actualiza una matriz **A** sumándole los **n** valores del vector **vals** (no tiene ningún cero) en las posiciones dadas por los vectores **rows** y **cols** que pueden tener valores repetidos.

```
void update( int n,int rows[],int cols[],double vals[], double A[M][N] ){
int i,j,k, row_max,col_max, cp = 0, cn = 0;
double x, max = -1e6;
for ( k = 0 ; k < n ; k++ ) {
        i = rows[k]; j = cols[k]; x = vals[k];
        if ( x > 0 ) cp++; else cn++;
        A[i][j] += x;
        if ( x > max ) {
                max = x; row_max = i; col_max = j;
        }
}
printf("%d actualizaciones positivas y %d negativas.\n",cp,cn);
printf("La mayor actualización ha sido de %.1f en la fila %d columna %d.\n",
max, row_max, col_max );
}
```

a) Paraleliza la función usando OpenMP.

```
void update( int n,int rows[],int cols[],double vals[], double A[M][N] ){
int i,j,k, row_max,col_max, cp = 0, cn = 0;
double x, max = -1e6;
#pragma omp parallel for private(i,j,x) reduction(+:cp,cn)
for ( k = 0 ; k < n ; k++ ) {
        i = rows[k]; j = cols[k]; x = vals[k];
        if ( x > 0 ) cp++; else cn++;
        #pragma omp atomic
        A[i][j] += x;
        if ( x > max )
                #pragma omp critical
                if ( x > max ) {
                        max = x; row_max = i; col_max = j;
                }
}
printf("%d actualizaciones positivas y %d negativas.\n",cp,cn);
printf("La mayor actualización ha sido de %.1f en la fila %d columna %d.\n",
max, row_max, col_max );
}
```

OpenMp 2021: Ejercicio 3

b) Modifica la paralelización del apartado anterior para que se muestre por pantalla el identificador del hilo que ha realizado más actualizaciones sobre la matriz A y cuántas han sido.

```
void update( int n,int rows[],int cols[],double vals[], double A[M][N] ){
int i,j,k, row_max,col_max, cp = 0, cn = 0;
double x, max = -1e6;
#pragma omp parallel for private(i,j,x) reduction(+:cp,cn)
for ( k = 0 ; k < n ; k++ ) {
        i = rows[k]; j = cols[k]; x = vals[k];
        if ( x > 0 ) cp++; else cn++;
        #pragma omp atomic
        A[i][j] += x;
        if ( x > max )
                #pragma omp critical
                if ( x > max ) {
                        max = x; row_max = i; col_max = j;
                }
}
printf("%d actualizaciones positivas y %d negativas.\n",cp,cn);
printf("La mayor actualización ha sido de %.1f en la fila %d columna %d.\n",
max, row_max, col_max );
}
```

```
void update( int n,int rows[],int cols[],double vals[], double A[M][N] ){
int i,j,k, row_max,col_max, cp = 0, cn = 0,  m =0 ;
double x, max = -1e6;
#pragma omp parallel
{ int c = 0, id;
   #pragma omp for private(i,j,x) reduction(+:cp,cn)   nowait
        for ( k = 0 ; k < n ; k++ ) {
                i = rows[k]; j = cols[k]; x = vals[k];
                c++;
                if ( x > 0 ) cp++; else cn++;
                #pragma omp atomic
                        A[i][j] += x;
                if ( x > max )
                 #pragma omp critical
                        if ( x > max ) {
                                max = x; row_max = i; col_max = j;
                        }
        }
        #pragma omp critical
        if ( c > m ) { m = c; id = omp_get_thread_num(); }
}
printf("%d actualizaciones positivas y %d negativas.\n",cp,cn);
printf("La mayor actualización ha sido de %.1f en la fila %d columna %d.\n",
max, row_max, col_max );
printf("El hilo %d es el que más actualizaciones ha realizado (%d).\n", id, m);
}
```

OpenMp 2022: Ejercicio 1

Dada la siguiente función

```
double f(double A[N][N], double B[N][N], double v[N]){
double x,p,sigma;
int i, j, c;
p = 1.0;
for (i=0; i<N; i++) {
      sigma=0;
      c=0;
      for (j=0; j<N; j++) {
            x=1.0/A[i][j];
            if (x>0) {
                  c++;
                  sigma+=x;
            }
      }
      for (j=0; j<=i; j++)
            p*=B[i][j];
      v[i]+=sigma/c;
}
return p;
}
```

#pragma omp parallel for private(sigma, c, j ,x) reduction(*:p)

a) Paraleliza el bucle externo mediante OpenMP.

OpenMp 2022: Ejercicio 1

b) Paraleliza los dos bucles internos usando una sola región paralela. Elimina las barreras implícitas innecesarias, si las hubiera.

```
double f(double A[N][N], double B[N][N], double v[N]){
double x,p,sigma;
int i, j, c;
p = 1.0;
for (i=0; i<N; i++) {
      sigma=0;
      c=0;
      for (j=0; j<N; j++) {
            x=1.0/A[i][j];
            if (x>0) {
                  c++;
                  sigma+=x;
            }
      }
      for (j=0; j<=i; j++)
            p*=B[i][j];
      v[i]+=sigma/c;
}
return p;
}
```

```
double f(double A[N][N], double B[N][N], double v[N]){
double x,p,sigma;
int i, j, c;
p = 1.0;
for (i=0; i<N; i++) {
      sigma=0;
      c=0;
      #pragma omp parallel
      {
          #pragma omp for private(x) reduction(+:c,sigma) nowait
           for (j=0; j<N; j++) {
                x=1.0/A[i][j];
                if (x>0) {
                      c++;
                      sigma+=x;
                }
           }
           #pragma omp for reduction(*:p)
           for (j=0; j<=i; j++)
                p*=B[i][j];
           v[i]+=sigma/c;
       }
}
return p;
}
```

OpenMp 2022: Ejercicio 1

c) Calcula el coste secuencial, indicando todos los pasos.

```
double f(double A[N][N], double B[N][N], double v[N]){
double x,p,sigma;
int i, j, c;
p = 1.0;
for (i=0; i<N; i++) {
    sigma=0;
    c=0;
    for (j=0; j<N; j++) {
        x=1.0/A[i][j];
        if (x>0) {
            c++;
            sigma+=x;
        }
    }
    for (j=0; j<=i; j++)
        p*=B[i][j];
    v[i]+=sigma/c;
}
return p;
}
```

$$t(n) = \sum_{i=0}^{N-1}\left(\sum_{j=0}^{N-1} 2 + \sum_{j=0}^{i} 1 + 2\right) = \sum_{i=0}^{N-1}(2N + i + 1) = 2N^2 + \frac{N^2}{2} + N \approx \frac{5N^2}{2} \text{ flops}$$

OpenMp 2022: Ejercicio 1

d) Supongamos que se paraleliza solo el primer bucle j.
Calcula el coste paralelo, indicando todos los pasos.
Calcula el speedup cuando *p* tiende a infinito.

$$t(n,p)=\sum_{i=0}^{N-1}\left(\sum_{j=0}^{N/p-1}2+\sum_{j=0}^{i}1+2\right)=\sum_{i=0}^{N-1}\left(\frac{2N}{p}+i+1\right)=$$

$$=\frac{2N^2}{p}+\frac{N^2}{2}+N\approx\frac{2N^2}{p}+\frac{N^2}{2}\ \text{flops}$$

$$\lim_{p\to\infty}t(n,p)=\lim_{p\to\infty}\left(\frac{2N^2}{p}+\frac{N^2}{2}\right)=\frac{N^2}{2}\ \text{flops}$$

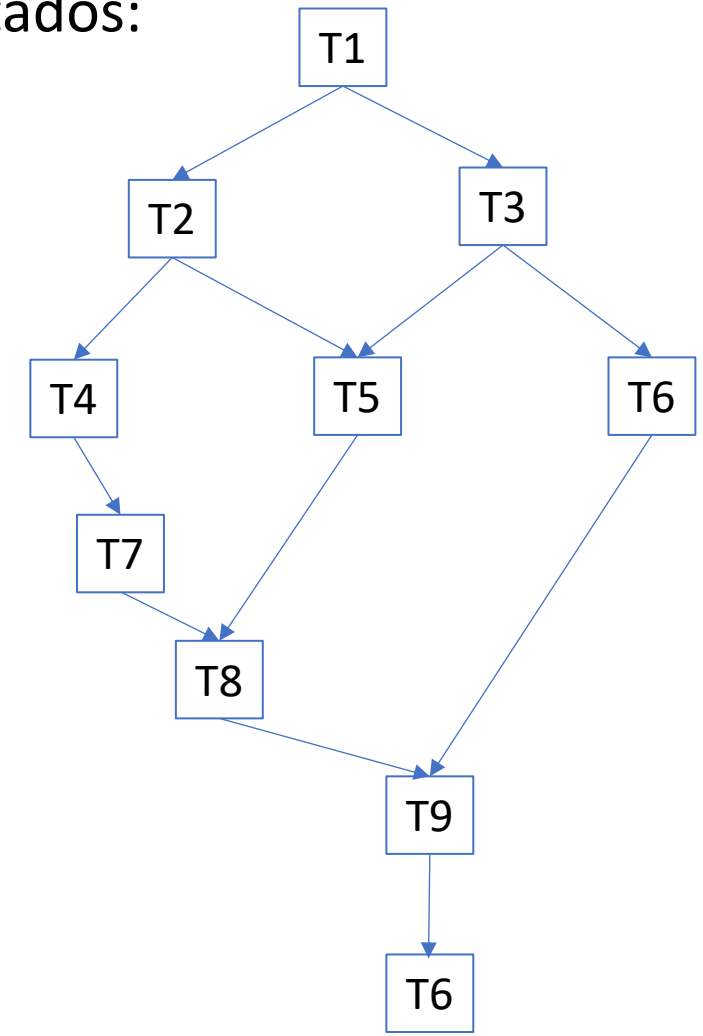
$$S(n,p)=\frac{\frac{5N^2}{2}}{\frac{N^2}{2}}=5$$

```
double f(double A[N][N], double B[N][N], double v[N]){
double x,p,sigma;
int i, j, c;
p = 1.0;
for (i=0; i<N; i++) {
      sigma=0;
      c=0;
      #pragma omp parallel
      {
          #pragma omp for private(x) reduction(+:c,sigma) nowait
           for (j=0; j<N; j++) {
                x=1.0/A[i][j];
                if (x>0) {
                     c++;
                     sigma+=x;
                }
           }
           #pragma omp for reduction(*:p)
           for (j=0; j<=i; j++)
                p*=B[i][j];
           v[i]+=sigma/c;
       }
}
return p;
}
```

OpenMp 2022: Ejercicio 2

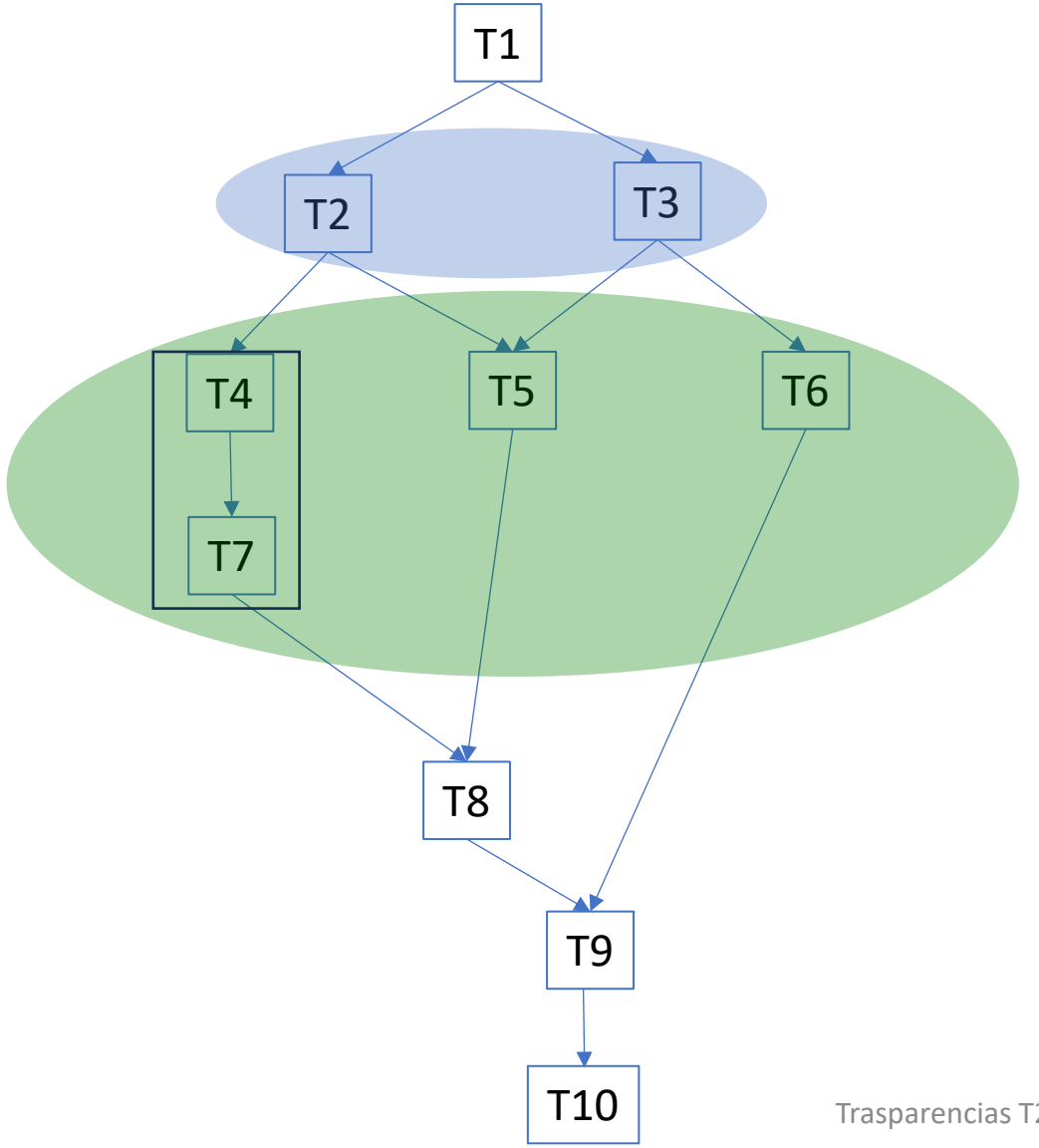
Dado el siguiente fragmento de código, donde n es una constante predefinida, suponemos que las matrices han sido rellenadas previamente, y teniendo en cuenta que las tres funciones f1, f2 y f3 modifican el segundo argumento y tienen un coste computacional de $1/3n^3$ flops, n^3flops y $2n^3$ flops respectivamente, realiza los siguientes apartados:

```
double A[n][n], B[n][n], C[n][n], D[n][n], E[n][n], F[n][n];
f1(n,A); /* Tarea T1 */
f2(n,D,A); /* Tarea T2 */
f2(n,F,A); /* Tarea T3 */
f2(n,B,D); /* Tarea T4 */
f3(n,E,F,D); /* Tarea T5 */
f3(n,C,F,F); /* Tarea T6 */
f1(n,B); /* Tarea T7 */
f2(n,E,B); /* Tarea T8 */
f3(n,C,E,E); /* Tarea T9 */
f1(n,C); /* Tarea T10 */
```



OpenMp 2022: Ejercicio 2

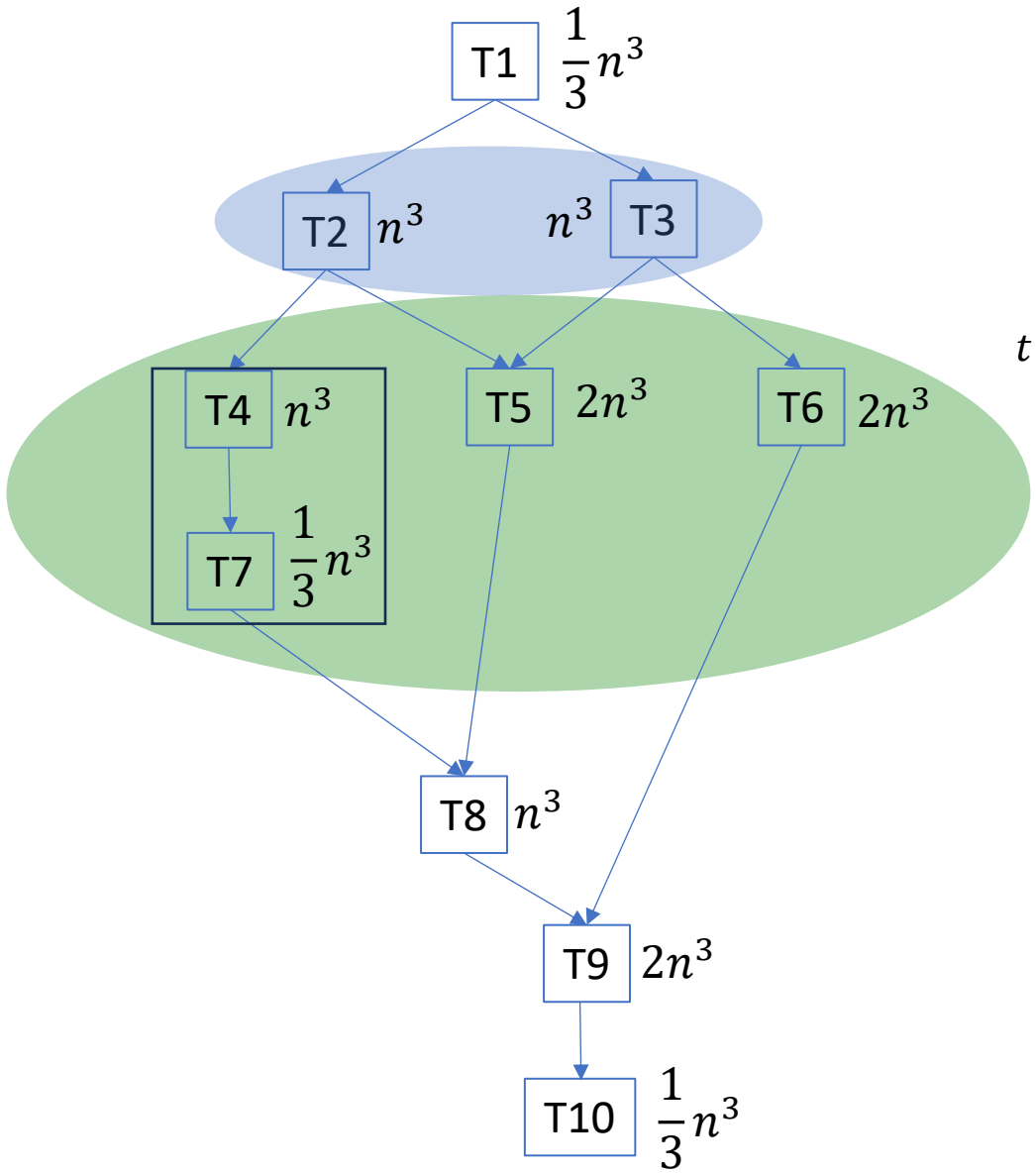
b) Implementa una versión paralela mediante OpenMP utilizando una sola región paralela.




```
f1(n,A); /* Tarea T1 */
#pragma omp parallel
{
        #pragma omp sections
        {
            #pragma omp section
                f2(n,D,A); /* Tarea T2 */
            #pragma omp section
                f2(n,F,A); /* Tarea T3 */
        }
        #pragma omp sections
        {
            #pragma omp section
            {
                f2(n,B,D); /* Tarea T4 */
                f1(n,B); /* Tarea T7 */
            }
            #pragma omp section
                f3(n,E,F,D); /* Tarea T5 */
            #pragma omp section
                f3(n,C,F,F); /* Tarea T6 */
        }
}
f2(n,E,B); /* Tarea T8 */
f3(n,C,E,E); /* Tarea T9 */
f1(n,C); /* Tarea T10 */
```

OpenMp 2022: Ejercicio 2

c) Obtén el speedup y la eficiencia de la versión paralela del apartado anterior suponiendo que se ejecuta con 4 hilos en un computador con 4 procesadores (núcleos).



$$t(n) \approx 3 \cdot \frac{1}{3}n^3 + 4 \cdot n^3 + 3 \cdot 2n^2 = 11n^3 \; flops$$

$$t(n, p = 4) = \frac{1}{3}n^3 + \max(n^3, n^3) + \max\left(n^3 + \frac{1}{3}n^3, 2n^3, 2n^3\right) + n^3 + 2n^3 + \frac{1}{3}n^3 =$$

$$= \frac{1}{3}n^3 + n^3 + 2n^3 + n^3 + 2n^3 + \frac{1}{3}n^3 = \frac{20n^3}{3} \; flops$$

$$S(n, p) \approx \frac{11n^2}{\frac{20}{3}n^2} = 1.65$$

$$E(n, p) \approx \frac{1.65}{4} = 0.41$$

OpenMp 2022, ejercicio 3: Dada la siguiente función, en la que la llamada a la función aleatorio devuelve un entero aleatorio entre los límites indicados en sus argumentos, Paraleliza, usando una única región paralela, esta función de la forma más eficiente posible utilizando OpenMP.

```
float valor(int n){
int i, j, ix, iy;
int hit[100][100];
float result, x, y;
float in=0.0, out=0.0;
int imax=0, jmax=0, max=0;
for (i=0;i<100;i++)
        for (j=0;j<100;j++)
                hit[i][j]=0;
for (i=0;i<n;i++) {
        ix = aleatorio(0,100);
        iy = aleatorio(0,100);
        hit[ix][iy]++;
}
for (i=0;i<100;i++)
        for (j=0;j<100;j++)
                if (hit[i][j]>max) {
                        max = hit[i][j];
                        imax=i; jmax=j;
                }
printf("Posición (%d,%d) con %d\n",imax,jmax,max);
for (i=0;i<100;i++) {
        x = fabs(50-i)/50.0;
        for (j=0;j<100;j++) {
                y = fabs(50-j)/50.0;
                if (sqrt(x*x+y*y)<1)
                        in+=hit[i][j];
                else
                        out+=hit[i][j];
        }
}
printf("%f - %f\n", in, out);
result = 4*in/(in+out);
return result;
}
```

```
#pragma omp parallel
{
    #pragma omp for private (j)
    for (i=0;i<100;i++)
        .......
    #pragma omp for private(ix, iy)
    for (i=0;i<n;i++) {
        .....
        #pragma omp atomic
            hit[ix][iy]++;
    }
    #pragma omp for private (j)
    for (i=0;i<100;i++)
        for (j=0;j<100;j++)
            if (hit[i][j]>max)
                #pragma omp critical
                    if (hit[i][j]>max) {
                        max = hit[i][j];
                        imax=i; jmax=j;
                    }
    #pragma omp single nowait
        printf("Posición (%d,%d) con %d\n",imax,jmax,max);
    #pragma omp for private (x, j, y) reduction(+:in) reduction(+:out)
    for (i=0;i<100;i++) {
        x = fabs(50-i)/50.0;
        for (j=0;j<100;j++) {
            ......
        }
    }
}
```