Qüestió 1 (1.2 punts)
Donada la següent funció

```
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;
  }
}
```

0.2 p. (a) Feu una versió paral·lela basada en la paral·lelització del bucle extern.

Solució: Just davant del bucle `i`, inclouríem:

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

0.2 p. (b) Modifica l'apartat anterior per a que es mostre una única volta el nombre de fils que intervenen en la execució del bucle paral·lel.

Solució:

```
...
double sumz;

#pragma omp parallel
{
  #pragma omp single
  printf("Nombre de fils: %d\n", omp_get_num_threads());
}

#pragma omp parallel for private (j, sumz)
for (i=0;i<N;i++) {
...

// Altra possible solució:

...
```

```
double sumz;

#pragma omp parallel private (j, sumz, id)
{
  #pragma omp single
  printf("Nombre de fils: %d\n", omp_get_num_threads());

  #pragma omp for
  for (i=0;i<N;i++) {
    ...
  }
}
```

0.5 p. (c) Feu una versión paral·lela basada en la paral·lelització dels dos bucles interns, usant una sola regió paral·lela. Considera la conveniència d'usar la clàusula `nowait` i, tant si la uses com si no, justifica per què.

Solució:

```
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;
}
```

S'utilitza la clàusula `nowait` en el primer bucle, perquè eixe bucle és independent del segon.

0.3 p. (d) Calcula el cost seqüencial i el cost paral·lel (inclou en ambdós casos el desenvolupament complet) suposant que es paral·lelitzara únicament el segon dels bucles interns. Calcula també en el mateix supòsit el speed-up i l'eficiència.

Solució:

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

$$t(N,p) = \sum_{i=0}^{N-1}(\sum_{j=i}^{N-1} 1 + \sum_{j=0}^{\frac{N}{p}-1} 2) \approx \sum_{i=0}^{N-1}(N - i + \frac{2N}{p}) \approx N^2 - \frac{N^2}{2} + \frac{2N^2}{p} = \frac{N^2}{2} + \frac{2N^2}{p} = \frac{p+4}{2p}N^2 flops$$

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

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

Qüestió 2 (1.1 punts)

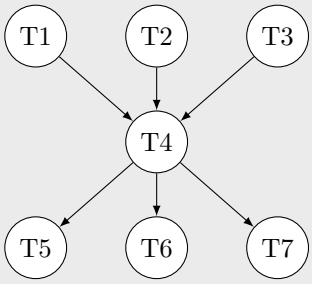
Donada la següent funció, on `Image` és un tipus de dades predefinit:

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

Cap de les funcions modifica els seus arguments.

0.3 p. (a) Dibuixeu el graf de dependències de dades entre les tasques.

Solució:



0.5 p. (b) Implementeu una versió paral·lela mitjançant OpenMP utilitzant una sola regió paral·lela.

Solució:

```
void transform_par(int n, Image im1, Image im2, float weights[3], float factor)
{
  #pragma omp parallel
  {
    #pragma omp sections
    {
      #pragma omp section
      weights[0] = channel_r(im1,im2,n);
      #pragma omp section
      weights[1] = channel_g(im1,im2,n);
      #pragma omp section
      weights[2] = channel_b(im1,im2,n);
    }
    #pragma omp single
    {
      factor *= combine(weights);
    }
    #pragma omp sections
    {
      #pragma omp section
      weights[0] += adjust_r(im2,n,factor);
      #pragma omp section
```

```
            weights[1] += adjust_g(im2,n,factor);
            #pragma omp section
            weights[2] += adjust_b(im2,n,factor);
          }
        }
      }
```

0.3 p. (c) Obteniu el speedup i l'eficiència de la versió paral·lela de l'apartat anterior suposant que s'executa amb 4 fils en un computador amb 4 processadors (nuclis).

Solució: Temps d'execució seqüencial:

$$t(n) = 5n^2 + 5n^2 + 5n^2 + 12 + n^2 + n^2 + n^2 \approx 18n^2 \text{ flops}$$

Temps d'execució paral·lel per a $p = 4$:

$$t(n,p) = 5n^2 + 12 + n^2 \approx 6n^2 \text{ flops}$$

Speedup:

$$S(n,p) = \frac{18n^2}{6n^2} = 3$$

Eficiència:

$$E(n,p) = \frac{3}{4} = 0.75$$

Qüestió 3 (1.2 punts)

La següent funció actualitza una matriu A sumant-li els n valors del vector *vals* (no té cap zero) en les posicions donades pels vectors *rows* i *cols* que poden tindre valors repetits.

```
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 actualitzacions positives i %d negatives.\n",cp,cn);
  printf("La major actualització ha sigut de %.1f en la fila %d columna %d.\n",
   max, row_max, col_max );
}
```

0.8 p. (a) Paral·lelitza la funció usant OpenMP.

Solució:

```
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 actualitzacions positives i %d negatives.\n",cp,cn);
  printf("La major actualització ha sigut de %.1f en la fila %d columna %d.\n",
   max, row_max, col_max );
}
```

0.4 p. (b) Modifica la paral·lelització anterior per a que es mostre per pantalla l'identificador del fil que ha realitzat más actualitzacions sobre la matriu A i quantes han sigut.

Solució:

```
#include <omp.h>
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, id, m = -1, c;
  double x, max = -1e6;

  #pragma omp parallel private(c)
  { c = 0;
    #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 actualitzacions positives i %d negatives.\n",cp,cn);
    printf("La major actualització ha sigut de %.1f en la fila %d columna %d.\n",
     max, row_max, col_max );

    printf("El fil %d és el que més actualitzacions ha realitzat (%d).\n",id,m);
  }
```