Cuestión 1 (1.1 puntos)
Dada la siguiente función:

```
void f(double A[N][N], double B[N][N], double v[N]){
  double s;
  int i, j, k;
  for(k=0; k<N; k++){
    s=0.0;
    for(i=0; i<N; i++)
      for(j=0; j<N; j++)
        s+=A[i][j]*B[i][j];
    v[k]=v[k]/s;
    for(i=0; i<N; i++)
      for(j=0; j<i; j++)
        A[i][j]+=B[i][j]/s;
  }
}
```

0.2 p. (a) Paraleliza, si es posible, el bucle más externo del código anterior, justificando la respuesta.

Solución: No es posible paralelizar el bucle exterior, pues hay dependencia entre las iteraciones del bucle k, ya que en cada iteración del bucle la matriz A es modificada.

0.6 p. (b) Paraleliza de la forma más eficiente posible los dos bucles internos (bucles con la variable i), usando una sola región paralela.

Solución:

```
void f(double A[N][N], double B[N][N], double v[N]){
  double s;
  int i, j, k;
  for(k=0; k<N; k++){
    s=0.0;
    #pragma omp parallel
      {
      #pragma omp for private(j) reduction(+:s)
      for(i=0; i<N; i++)
       for(j=0; j<N; j++)
          s+=A[i][j]*B[i][j];
      #pragma omp single nowait
      v[k]=v[k]/s;
      #pragma omp for private(j)
      for(i=0; i<N; i++)
        for(j=0; j<i; j++)
          A[i][j]+=B[i][j]/s;
      }
```

```
    }
  }
```

0.3 p. (c) Calcula el tiempo secuencial, el tiempo paralelo, el speedup y la eficiencia correspondiente a una sola iteración del bucle k, suponiendo que únicamente se ha paralelizado el primer bucle interno (primer bucle i).

Solución:

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

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

$$S(N,p) = \frac{t(N)}{t(N,p)} = \frac{3N^2}{\left(\frac{2}{p}+1\right)N^2} = \frac{3}{\frac{2}{p}+1}.$$

$$E(N,p) = \frac{S(N,p)}{p} = \frac{3}{2+p}.$$

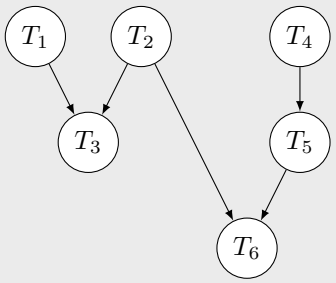
Cuestión 2 (1.2 puntos)

Dada la siguiente función, donde cada una de las funciones invocadas modifica solo el vector que recibe como primer argumento:

```
void func(double a[], double b[], double c[], double d[], int n) {
  double x;
  tarea1(b,a,n);      /* Coste: 4n flops */
  x=tarea2(c,a,n);    /* Coste: 3n flops */
  tarea3(b,c,n);      /* Coste: 2n flops */
  tarea4(d,a,n);      /* Coste:  n flops */
  tarea5(d,a,n);      /* Coste:  n flops */
  tarea6(d,x,n);      /* Coste: 3n flops */
}
```

0.4 p. (a) Dibuja el grafo de dependencias de datos entre las tareas. Señala el camino crítico indicando su longitud y obtén el grado medio de concurrencia.

Solución:



Camino crítico: T1→T3 o bien T2→T6.

Longitud del camino crítico: $L = 4n + 2n = 6n$ flops

Grado medio de concurrencia:

$$M = \frac{4n + 3n + 2n + n + n + 3n}{6n} = \frac{14n}{6n} = 2{,}33$$

0.6 p. (b) Haz una versión paralela usando OpenMP. Debe minimizarse el tiempo de ejecución.

Solución:

```
void func(double a[], double b[], double c[], double d[], int n) {
  double x;
  #pragma omp parallel
  {
    #pragma omp sections
    {
      #pragma omp section
         tarea1(b,a,n);
      #pragma omp section
         x=tarea2(c,a,n);
      #pragma omp section
      {
         tarea4(d,a,n);
         tarea5(d,a,n);
      }
    }
    #pragma omp sections
    {
      #pragma omp section
         tarea3(b,c,n);
      #pragma omp section
         tarea6(d,x,n);
    }
  }
}
```

0.2 p. (c) Obtén el speedup de la versión paralela del apartado anterior si se ejecuta con 3 procesadores.

Solución: Tiempo de ejecución secuencial:

$$t(n) = 4n + 3n + 2n + n + n + 3n = 14n \text{ flops}$$

El tiempo de ejecución paralelo sería $4n$ flops para el primer `sections`, más $3n$ para el segundo `sections`:

$$t(n,p) = 4n + 3n = 7n \text{ flops}$$

Speedup:

$$S(n,p) = \frac{14n}{7n} = 2$$

Cuestión 3 (1.2 puntos)

La función `pluviometría` calcula diferentes valores estadísticos relacionados con la precipitación de agua ocurrida durante un conjunto de ND días en un territorio para un total de NM meses. A su vez, la función `mes_dia` proporciona el identificador del mes al que pertenece un día en concreto. Paraleliza la funcion mediante OpenMP empleando una única región paralela. El orden de los días en el vector `dias_riesgo` no es relevante en la versión paralela.

```
void pluviometria(float precipitacion[ND],int litros_riesgo, float  lluvia_media_mes[NM]) {
   int i, mes, ndias_lluviosos=0, dia_max, ndias_lluviosos_mes[NM];
   int ndias_riesgo=0, dias_riesgo[ND];
```

```
  float suma_lluvia=0, precipitacion_max=0, lluvia_media;
  float suma_lluvia_mes[NM];

  /* Inicialización de los elementos de los vectores a 0 */
  ...

  for (i=0;i<ND;i++) {
    if (precipitacion[i]>0) {
       suma_lluvia+=precipitacion[i];
       ndias_lluviosos++;
       if (precipitacion[i]>litros_riesgo) {
          dias_riesgo[ndias_riesgo]=i+1;
          ndias_riesgo++;
       }
       if (precipitacion[i]>precipitacion_max) {
          precipitacion_max=precipitacion[i];
          dia_max=i;
       }
       mes=mes_dia(i);
       ndias_lluviosos_mes[mes]++;
       suma_lluvia_mes[mes]+=precipitacion[i];
    }
  }
  lluvia_media=suma_lluvia/ndias_lluviosos;
  for (i=0;i<NM;i++)
     lluvia_media_mes[i]=suma_lluvia_mes[i]/ndias_lluviosos_mes[i];

  /* El código sigue.... */
  ...
 }
```

Solución:

```
void pluviometria(float precipitacion[ND],int litros_riesgo, float  lluvia_media_mes[NM]) {
   int i, mes, ndias_lluviosos=0, dia_max, ndias_lluviosos_mes[NM];
   int ndias_riesgo=0, dias_riesgo[ND];
   float suma_lluvia=0, precipitacion_max=0, lluvia_media;
   float suma_lluvia_mes[NM] ;

   /* Inicialización de los elementos de los vectores a 0 */
   ...

   #pragma omp parallel
   {
      #pragma omp for private(mes) reduction(+:suma_lluvia,ndias_lluviosos)
      for (i=0;i<ND;i++) {
         if (precipitacion[i]>0) {
            suma_lluvia+=precipitacion[i];
            ndias_lluviosos++;
            if (precipitacion[i]>litros_riesgo) {
               #pragma omp critical (riesgo)
```

```
                {
                   dias_riesgo[ndias_riesgo]=i+1;
                   ndias_riesgo++;
                }
             }
             if (precipitacion[i]>precipitacion_max) {
                #pragma omp critical (maximo)
                {
                   if (precipitacion[i]>precipitacion_max) {
                      precipitacion_max=precipitacion[i];
                      dia_max=i;
                   }
                }
             }
             mes=mes_dia(i);
             #pragma omp atomic
             ndias_lluviosos_mes[mes]++;
             #pragma omp atomic
             suma_lluvia_mes[mes]+=precipitacion[i];
          }
       }
       lluvia_media=suma_lluvia/ndias_lluviosos;
       #pragma omp for
       for (i=0;i<NM;i++)
          lluvia_media_mes[i]=suma_lluvia_mes[i]/ndias_lluviosos_mes[i];

      /* El código sigue.... */
       ...
}
```