Cuestión 1 (1.1 puntos)

Dado el siguiente código principal de un programa, donde suponemos que `n` es una constante definida con un valor entero:

```
float a;
float A[n][n], X[n][n], Y[n][n], Z[n][n];
T1( A, X, Y, Z );
a = T2( A );
T3( a, X );
T4( a, Y );
T5( A, A, Z );
T6( X, Y, Z );
```

y dado el código de las siguientes funciones:

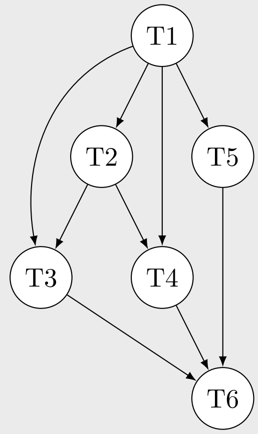
```
float T2( float A[n][n] ) {
  float a = 0.0;
  for( int i=0; i<n; i++ )
    for( int j=i; j<n; j++ )
      for( int k=i; k<n; k++ )
        a = a + A[i][k];
  return a;
}

void T5( float X[n][n], float Y[n][n], float Z[n][n] ) {
  float aux;
  for( int i=0; i<n; i++ )
    for( int j=0; j<n; j++ ) {
      aux=0;
      for ( int k=0; k<n; k++ )
        aux += X[i][k]*Y[k][j];
      Z[i][j]=aux;
    }
}
```

donde sabemos que la función `T1` modifica todos sus argumentos y tiene un coste de n^3 flops, mientras que las funciones `T3`, `T4` y `T6` modifican solo su último argumento y tienen también un coste de n^3 flops. La función `T2` tiene un coste de $\frac{n^3}{3}$ y la función `T5` tiene un coste de $2n^3$.

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

Solución:



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

Solución:

```
float a;
float A[n][n], X[n][n], Y[n][n], Z[n][n];
T1( A, X, Y, Z );
#pragma omp parallel
{
  #pragma omp sections
  {
    #pragma omp section
    a = T2( A );
    #pragma omp section
    T5( A, A, Z );
  }
  #pragma omp sections
  {
    #pragma omp section
    T3( a, X );
    #pragma omp section
    T4( a, Y );
  }
} /* Fin del parallel */
T6( X, Y, Z );
```

0.2 p. (c) Obtén el speedup de la versión paralela del apartado anterior para 2 procesadores.

Solución:
Tiempo de ejecución secuencial:

$$t(n) = n^3 + \frac{n^3}{3} + n^3 + n^3 + 2n^3 + n^3 = \left(6 + \frac{1}{3}\right) n^3 = \frac{19}{3} n^3 \text{ flops}$$

Tiempo de ejecución paralelo para $p = 2$:

$$t(n,p) = n^3 + \max\left(\frac{n^3}{3}, 2n^3\right) + \max(n^3, n^3) + n^3 = 5n^3 \text{ flops}$$

Speedup:

$$S = \frac{\frac{19}{3}n^3}{5n^3} = \frac{19}{15} = 1{,}27$$

Cuestión 2 (1.2 puntos)

Se quiere gestionar un concurso de `nP` participantes, por parte de un jurado compuesto por `nM` miembros, numerados desde la posición 0 en adelante. Cada jurado emite dos votos, uno de 10 puntos para un participante y otro de 5 para otro. Para ello, la función `gestiona_votos` recibe el vector `votos_jurado`, de `nM`*2 componentes, con los dos votos emitidos por cada integrante del jurado y completa el vector `ptos_participantes`, de `nP` elementos, con la puntuación alcanzada por cada participante. Además de calcular cuál ha sido el participante que ha ganado el concurso (máxima puntuación), la función completa los vectores `m10ptos` y `m5ptos` con los identificadores de los miembros del jurado que han concedido 10 puntos o 5 puntos, respectivamente, al participante indicado como argumento de la función. Al final, obtiene en la variable `nPCeroPtos` el número de participantes sin ningún voto.

```
void gestiona_votos(int votos_jurado[], int participante) {
  int i, p10ptos, p5ptos, ptos_participantes[nP];
  int m10ptos[nM], m5ptos[nM];
  int nM10ptos=0, nM5ptos=0, nPCeroPtos=0;
  int ptos_max=0, ganador;
  ...
  for (i=0;i<nM;i++) {
    // Acumulamos los puntos a los participantes
    p10ptos=votos_jurado[i*2];
    p5ptos=votos_jurado[i*2+1];
    ptos_participantes[p10ptos]+=10;
    ptos_participantes[p5ptos]+=5;
    // Obtenemos los miembros que han votado al participante indicado
    if (p10ptos==participante) {
      m10ptos[nM10ptos]=i;
      nM10ptos++;
    }
    else if (p5ptos==participante) {
      m5ptos[nM5ptos]=i;
      nM5ptos++;
    }
  }
  // Calculamos el ganador y el número de participantes con 0 puntos
  for (i=0;i<nP;i++) {
    if (ptos_participantes[i]>ptos_max) {
      ptos_max=ptos_participantes[i];
      ganador=i;
    }
    else if (ptos_participantes[i]==0)
      nPCeroPtos++;
  }
  ...
}
```

Paralelizar `gestiona_votos` de la forma más eficiente posible, empleando para ello una única región paralela.

Solución:

```
void gestiona_votos(int votos_jurado[], int participante) {
  int i, p10ptos, p5ptos, ptos_participantes[nP];
  int m10ptos[nM], m5ptos[nM];
  int nM10ptos=0, nM5ptos=0, nPCeroPtos=0;
  int ptos_max=0, ganador;
  ...
  #pragma omp parallel
  {
```

```
      #pragma omp for private(p10ptos, p5ptos)
      for (i=0;i<nM;i++) {
        // Acumulamos los puntos a los participantes
        p10ptos=votos_jurado[i*2];
        p5ptos=votos_jurado[i*2+1];
        #pragma omp atomic
        ptos_participantes[p10ptos]+=10;
        #pragma omp atomic
        ptos_participantes[p5ptos]+=5;
        // Obtenemos los miembros que han votado al participante indicado
        if (p10ptos==participante) {
          #pragma omp critical (ptos10)
          {
            m10ptos[nM10ptos]=i;
            nM10ptos++;
          }
        }
        else if (p5ptos==participante) {
          #pragma omp critical (ptos5)
          {
            m5ptos[nM5ptos]=i;
            nM5ptos++;
          }
        }
      }
      // Calculamos el ganador y el número de participantes con 0 puntos
      #pragma omp for reduction(+:nPCeroPtos)
      for (i=0;i<nP;i++) {
        if (ptos_participantes[i]>ptos_max) {
          #pragma omp critical
          if (ptos_participantes[i]>ptos_max) {
            ptos_max=ptos_participantes[i];
            ganador=i;
          }
        }
        else if (ptos_participantes[i]==0)
          nPCeroPtos++;
      }
    }
    ...
  }
```

Cuestión 3 (1.2 puntos)

Dada la siguiente función:

```
void normaliza_mat(double A[N][N]){
  int i, j;
  double s, norm1=0, norm2=0;
  for (i = 0; i < N; i++){
      s = 0;
      for (j=0; j<N; j++){
          norm1+=A[i][j];
          s+= fabs(A[i][j]);
      }
      if (s>norm2)
         norm2= s;
  }
```

```
    norm1=sqrt(norm1)*norm2;
    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
            A[i][j]*=norm1;
  }
```

0.9 p. (a) Paraleliza mediante OpenMP la función anterior, usando para ello una sola región paralela.

Solución:

```
#pragma omp parallel
{
  #pragma omp for private(j,s) reduction(+:norm1) reduction(max:norm2)
   for (i = 0; i < N; i++){
       ...
   }
  #pragma omp single
  norm1=sqrt(norm1)*norm2;
  #pragma omp for private(j)
   for (i = 0; i < N; i++)
       for (j = 0; j < N; j++)
           A[i][j]*=norm1;
}
```

0.2 p. (b) Calcula el coste secuencial y paralelo, suponiendo que N es divisible entre el número de hilos p y que el coste de las funciones `fabs` y `sqrt` es de 1 flop. Indicar los cálculos intermedios para alcanzar la solución.

Solución: Coste secuencial:

$$t(N) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} 3 + 2 + \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} 1 = 3N^2 + 2 + N^2 \approx 4N^2 flops.$$

Coste paralelo:

$$t(N,p) = \sum_{i=0}^{N/p-1} \sum_{j=0}^{N-1} 3 + 2 + \sum_{i=0}^{N/p-1} \sum_{j=0}^{N-1} 1 = \frac{3N^2}{p} + 2 + \frac{N^2}{p} \approx \frac{4N^2}{p} flops.$$

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

Solución:

$$S(n,p) = \frac{4N^2}{\frac{4N^2}{p}} = p.$$

$$E(n,p) = \frac{S(n,p)}{p} = 1.$$