Qüestió 1 (1.2 punts)

Donada la següent funció:

```
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;
}
```

0.3 p. (a) Paral·lelitza el bucle extern mitjançant OpenMP.

Solució: Caldria afegir la següent directiva just abans del bucle:

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

0.5 p. (b) Paral·lelitza els dos bucles interns usant una sola regió paral·lela. Elimina les barreres implícites innecessàries, si hi ha.

Solució:

```
...
  c=0;             /* Tot igual fins aquesta línia */
  #pragma omp parallel
  {
    #pragma omp for private(x) reduction(+:c,sigma) nowait
    for (j=...) {
      ...
    }
    #pragma omp for reduction(*:p)
    for (j=...) {
```

```
            ...
          }
        }
        v[i]+=sigma/c;   /* Tot igual després d'aquesta línia */
      ...
```

0.1 p. (c) Calcula el cost seqüencial, indicant tots els passos.

Solució:

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

0.3 p. (d) Suposem que es paral·lelitza només el primer bucle `j`. Calcula el cost paral·lel, indicant tots els passos. Calcula l'speedup quan p tendeix a infinit.

Solució: Cost paral·lel:

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

Quan p tendeix a infinit, $t(N,p) \approx \frac{N^2}{2}$, i per tant l'speedup serà

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

Qüestió 2 (1.2 punts)

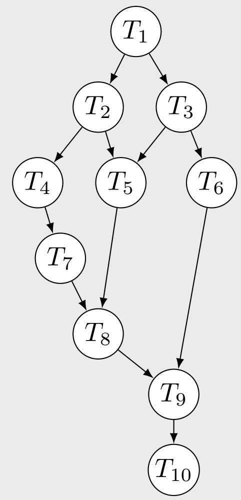
Donat el següent fragment de codi, on `n` és una constant predefinida, suposem que les matrius han sigut omplides prèviament, i tenint en compte que les tres funcions `f1, f2 i f3` modifiquen el segon argument i tenen un cost computacional de $\frac{1}{3}n^3$ flops, n^3 flops i $2n^3$ flops respectivament, realitza els següents apartats:

```
double A[n][n], B[n][n], C[n][n], D[n][n], E[n][n], F[n][n];

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

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

Solució:



0.6 p. (b) Implementa una versió paral·lela mitjançant OpenMP utilitzant una sola regió paral·lela.

Solució: La tasca T_1 no és concurrent amb ninguna altra, i per tant es pot fer fora de la regió paral·lela. Les tasques T_7, T_8, T_9, i T_{10} s'han d'executar necessàriament de forma seqüencial, una darrere de l'altra. Per tant, es poden deixar fora de la regió paral·lela. La millor solució consistiria en agregar les tasques T_4 i T_7 per a que les realitze el mateix fil (en la mateixa secció). D'aquesta manera, la tasca T_7 es farà en paral·lel amb les tasques T_5 i T_6.

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

0.3 p. (c) Calcula l'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:

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

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

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

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

Speedup:

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

Eficiència:

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

Qüestió 3 (1 punt)

Donada la següent funció, en la que la crida a la funció `aleatori` retorna un enter aleatori entre els límits indicats en els seus arguments.

```
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 = aleatori(0,100);
    iy = aleatori(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("Posició (%d,%d) amb %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;
}
```

Paral·lelitza, usant una única regió paral·lela, aquesta funció de la forma més eficient possible utilizant OpenMP.

Solució:

```c
float valorpar(int n) {
  int i, j, ix, iy;
  int hit[100][100];
  float result, x, y;
  float in=0.0, out=0.0;
  int max=0, imax=0, jmax=0;

  #pragma omp parallel
  {
    #pragma omp for private (j)
    for (i=0;i<100;i++)
      for (j=0;j<100;j++)
        hit[i][j]=0;

    #pragma omp for private(ix, iy)
    for (i=0;i<n;i++) {
      ix = aleatori(0,100);
      iy = aleatori(0,100);
      #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("Posició (%d,%d) amb %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++) {
        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 = %f\n", in, out, in+out);
  result = 4*in/(in+out);

  return result;
}
```