Qüestió 1 (1 punt)

Donat on següent codi, on `N` i `M` són constants enteres:

```
double func(double A[][N], double B[][N], double w[]) {
  int i,j,k;
  double x,wt,wk,pxc,val=1000;
  for (j=0; j<N; j++) {
    pxc=1.2;
    for (i=M; i<M+N; i++) {
      x=0.5;
      wt=0.5;
      for (k=-M; k<=M; k++) {
        wk = w[k+M];
        x += A[i+k][j]*wk;
        wt += wk;
      }
      B[i][j] = x/wt;
      pxc *= B[i][j];
    }
    if (pxc<val) val=pxc;
  }
  return val;
}
```

0.2 p. (a) Fes una versió paral·lela basada en la paral·lelització del bucle més intern (`k`).

Solució: S'afigiria la següent directiva just abans del bucle.

```
#pragma omp parallel for private(wk) reduction(+:x,wt)
```

0.2 p. (b) Fes una versió paral·lela basada en la paral·lelització del bucle intermedi (`i`).

Solució: S'afigiria la següent directiva just abans del bucle.

```
#pragma omp parallel for private(x,wt,k,wk) reduction(*:pxc)
```

0.2 p. (c) Fes una versió paral·lela basada en la paral·lelització del bucle més extern (`j`).

Solució: S'afigiria la següent directiva just abans del bucle.

```
#pragma omp parallel for private(pxc,i,x,wt,k,wk) reduction(min:val)
```

0.3 p. (d) Calcula el temps d'execució seqüencial en flops, detallant els passos.

Solució:

$$t = \sum_{j=0}^{N-1} \sum_{i=M}^{M+N-1} \left(2 + \sum_{k=-M}^{M} 3 \right) \approx \sum_{j=0}^{N-1} \sum_{i=M}^{M+N-1} 6M = \sum_{j=0}^{N-1} 6MN = 6MN^2 \text{ flops.}$$

0.1 p. (e) Suposem que s'ha medit el temps d'execució de manera experimental, obtenint un temps de 40 segons amb un processador i 10 segons amb 5 processadors. Indica el speedup i l'eficiència corresponents.

Solució:

$$S = \frac{40}{10} = 4, \quad E = \frac{4}{5} = 0.8.$$

Qüestió 2 (1.3 punts)

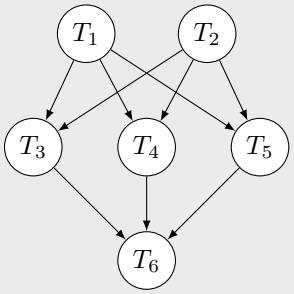
Donada la següent funció, on `n` és una constant predefinida, suposem que les matrius `A`, `B` i `C` han sigut emplenades prèviament, i a més:

- La funció `processcol(M,i,x)` modifica la columna `i` de la matriu `M` a partir de cert valor `x`. El seu cost és $2n$ flops.
- La funció del sistema `fabs` torna el valor absolut d'un número en coma flotant i es pot considerar que té un cost de 1 flop.

```
void computemat(double A[n][n], double B[n][n], double C[n][n], double D[n][n]) {
  int i,j;
  double alpha=0.0,beta=0.0;
  for (i=0;i<n;i++) alpha += fabs(A[i][i]-B[i][i]);
  for (i=0;i<n;i++) beta += (A[i][i]+B[i][i])/2.0 - C[i][i];
  for (i=0;i<n;i++) processcol(A,i,alpha);
  for (i=0;i<n;i++) processcol(B,i,beta);
  for (i=0;i<n;i++) processcol(C,i,alpha*beta);
  for (i=0;i<n;i++) {
    for (j=0;j<n;j++) {
      D[i][j] = A[i][j]+0.5*B[i][j]-C[j][i];
    }
  }
}
```

0.4 p. (a) Dibuixa el graf de dependències de dades entre les tasques, suposant que hi ha 6 tasques, corresponents a cadascun dels bucles `i`. Indica quin es el camí crític i calcula el grau mitjà de concurrència del graf.

Solució: A més de les dependències de flux, existeixen anti-dependències degudes a que T_3, T_4 i T_5 modifiquen las matrius, que són entrada de T_1 i T_2.



Per a obtindre el grau mitjà de concurrència necessitem calcular els costos de cada tasca:

T_1	$3n$	T_2	$4n$	T_3	$2n^2$	T_4	$2n^2$	T_5	$2n^2$	T_6	$3n^2$

Tenint en compte els costos obtinguts, el cost seqüencial és:

$$t(n) = 3n + 4n + 2n^2 + 2n^2 + 2n^2 + 3n^2 = 7n + 9n^2 \approx 9n^2 \text{ flops.}$$

El camí crític és $T_2 - T_4 - T_6$ (o també $T_2 - T_5 - T_6$, o $T_2 - T_3 - T_6$), la longitud del qual és:

$$L = 4n + 2n^2 + 3n^2 = 4n + 5n^2 \approx 5n^2 \text{ flops.}$$

Per tant, el grau mitjà de concurrència és:

$$M = \frac{t(n)}{L} = \frac{9n^2}{5n^2} = \frac{9}{5} = 1,8.$$

0.5 p. (b) Implementa una versió paral·lela OpenMP amb una sola regió paral·lela, basada en el paral·lelisme de tasques.

Solució: La tasca T_6 no és concurrent amb cap altra, pel que es pot fer fora de la regió paral·lela. Per a la resta de tasques utilitzem dos construccions `sections`, amb 2 i 3 tasques concurrents, respectivament. Les variables `alpha` i `beta` són compartides, ja que no hi ha possibilitat de que fils distints lligen i escriguen simultàniament eixes variables. La variable `i` és necesari fer-la privada explícitament perquè no estem usant la directiva `for`.

```
void computemat(double A[n][n], double B[n][n], double C[n][n], double D[n][n]) {
  int i,j;
  double alpha=0.0,beta=0.0;
  #pragma omp parallel private(i)
  {
```

```
      #pragma omp sections
      {
        #pragma omp section
        for (i=0;i<n;i++) alpha += fabs(A[i][i]-B[i][i]);          /* T1 */
        #pragma omp section
        for (i=0;i<n;i++) beta += (A[i][i]+B[i][i])/2.0 - C[i][i];  /* T2 */
      }
      #pragma omp sections
      {
        #pragma omp section
        for (i=0;i<n;i++) processcol(A,i,alpha);       /* T3 */
        #pragma omp section
        for (i=0;i<n;i++) processcol(B,i,beta);        /* T4 */
        #pragma omp section
        for (i=0;i<n;i++) processcol(C,i,alpha*beta);  /* T5 */
      }
    }
    for (i=0;i<n;i++) {                      /* T6 */
      for (j=0;j<n;j++) {
        D[i][j] = A[i][j]+0.5*B[i][j]-C[j][i];
      }
    }
  }
```

0.4 p. (c) Implementa una versió paral·lela OpenMP amb una sola regió paral·lela, basada en el paral·lelisme de bucles. Quan siga possible, realitza optimitzacions tals com eliminar les barreres innecessàries.

Solució: En este cas sí que incloem la tasca 6 dins de la regió paral·lela.

```
void computemat(double A[n][n], double B[n][n], double C[n][n], double D[n][n]) {
  int i,j;
  double alpha=0.0,beta=0.0;
  #pragma omp parallel
  {
    #pragma omp for reduction(+:alpha) nowait
    for (i=0;i<n;i++) alpha += fabs(A[i][i]-B[i][i]);
    #pragma omp for reduction(+:beta)
    for (i=0;i<n;i++) beta += (A[i][i]+B[i][i])/2.0 - C[i][i];
    #pragma omp for nowait
    for (i=0;i<n;i++) processcol(A,i,alpha);
    #pragma omp for nowait
    for (i=0;i<n;i++) processcol(B,i,beta);
    #pragma omp for
    for (i=0;i<n;i++) processcol(C,i,alpha*beta);
    #pragma omp for private(j)
    for (i=0;i<n;i++) {
      for (j=0;j<n;j++) {
        D[i][j] = A[i][j]+0.5*B[i][j]-C[j][i];
      }
    }
  }
}
```

Per determinar quines barreres es poden eliminar o no, només cal tindre en compte les dependències de tasques mostrades en el graf de l'apartat a). En el primer bucle també és possible eliminar la barrera, encara que tinga una clàusula de reducció, ja que la variable `alpha` no s'utilitza fins després de la barrera del segon bucle.

Qüestió 3 (1.2 punts)

Mitjançant la següent funció, la DGT gestiona les infraccions comeses per un conjunt de conductors. Per a això, la funció rep com a dada d'entrada els vectors `Conductors`, `PtsAPerdre` i `Quanties`, de `NI` components, que contenen respectivament l'identificador del conductor sancionat i els punts i la quantia econòmica associada a

cadascuna de les `NI` infraccions comeses.

A partir d'eixos vectors, la funció actualitza els vectors `PtsConductors` i `nSancionsConductors` amb els punts que li quedaran a cada conductor després de la sanció i amb el número d'infraccions que ha comés al llarg del temps. Addicionalment, la funció completa el vector `ConductorsSensePts`, en el qual es guarden els identificadors dels conductors que hagen perdut tots els seus punts. La longitud d'eixe vector es torna com a dada d'eixida. Finalment, la funció obté i mostra per pantalla la major sanció econòmica i la suma dels diners totals recaptats amb les infraccions.

```
int processa_multes(int Conductors[],int PtsAPerdre[],float Quanties[],
                    int PtsConductors[],int nSancionsConductors[],
                    int ConductorsSensePts[]) {
  int i,conductor,punts,quantia,nCondSensePts=0;
  float QuantiaMax=0,TotalRecaptat=0;
  for (i=0;i<NI;i++) {
     conductor=Conductors[i];
     punts=PtsAPerdre[i];
     quantia=Quanties[i];
     TotalRecaptat+=quantia;
     nSancionsConductors[conductor]++;
     if (PtsConductors[conductor]>0) {
        PtsConductors[conductor]-=punts;
        if (PtsConductors[conductor]<=0) {
           PtsConductors[conductor]=0;
           ConductorsSensePts[nCondSensePts]=conductor;
           nCondSensePts++;
        }
     }
     if (quantia>QuantiaMax)
        QuantiaMax=quantia;
  }
  printf("Total recaptat: %.2f euros\n",TotalRecaptat);
  printf("Quantia maxima: %.2f euros\n",QuantiaMax);
  return nCondSensePts;
}
```

0.8 p. (a) Paral·lelitzar la funció mitjançant OpenMP de la forma més eficient.

Solució:

```
int processa_multes(int Conductors[],int PtsAPerdre[],float Quanties[],
                    int PtsConductors[],int nSancionsConductors[],
                    int ConductorsSensePts[]) {
  int i,conductor,punts,quantia,nCondSensePts=0;
  float QuantiaMax=0,TotalRecaptat=0;
  #pragma omp parallel for private(conductor,punts,quantia) \
              reduction(+:TotalRecaptat) reduction(max:QuantiaMax)
  for (i=0;i<NI;i++) {
    conductor=Conductors[i];
    punts=PtsAPerdre[i];
    quantia=Quanties[i];
    TotalRecaptat+=quantia;
    #pragma omp atomic
    nSancionsConductors[conductor]++;
    if (PtsConductors[conductor]>0) {
      #pragma omp critical
      if (PtsConductors[conductor]>0) {
        PtsConductors[conductor]-=punts;
        if (PtsConductors[conductor]<=0) {
          PtsConductors[conductor]=0;
          ConductorsSensePts[nCondSensePts]=conductor;
          nCondSensePts++;
        }
      }
```

```
      }
       if (quantia>QuantiaMax)
          QuantiaMax=quantia;
    }
    printf("Total recaptat: %.2f euros\n",TotalRecaptat);
    printf("Quantia maxima: %.2f euros\n",QuantiaMax);
    return nCondSensePts;
  }
```

0.4 p.

(b) Paral·lelitza de nou el bucle, però esta vegada sense utilitzar la directiva `for` d'OpenMP (és a dir, fent un repartiment explícit de les iteracions entre els fils).

Solució:

```
int processa_multes(int Conductors[],int PtsAPerdre[],float Quanties[],
                    int PtsConductors[],int nSancionsConductors[],
                    int ConductorsSensePts[]) {
  int i,conductor,punts,quantia,nCondSensePts=0,myid,nThreads;
  float QuantiaMax=0,TotalRecaptat=0;
  #pragma omp parallel private(conductor,punts,quantia,myid,i) \
              reduction(+:TotalRecaptat) reduction(max:QuantiaMax)
  {
    myid=omp_get_thread_num();
    nThreads=omp_get_num_threads();
    for (i=myid;i<NI;i+=nThreads) {
      ... /* Mateix codi de l'apartat anterior */
    }
  }
  printf("Total recaptat: %.2f euros\n",TotalRecaptat);
  printf("Quantia maxima: %.2f euros\n",QuantiaMax);
  return nCondSensePts;
}
```