Qüestió 1 (1.2 punts)

Donada la següent funció:

```
int genera_mat(double x[], double y[], double A[][N]) {
   int i, j, count=0;
   double ci=x[0]*y[0], cs=ci, c;
   for (i=0; i<N; i++) {
      for (j=0; j<N; j++){
         A[i][j]=x[i]*y[j];
         if (A[i][j]>cs) cs=A[i][j];
         if (A[i][j]<ci) ci=A[i][j];
      }
   }
   c=(ci+cs)/2.0;
   for (i=0; i<N; i++)
      for (j=0; j<N; j++)
         if (A[i][j]>c) ++count;
   return count;
}
```

0.8 p. (a) Paral·lelitza mitjançant OpenMP la funció anterior, usant una sola regió paral·lela.

Solució:

```
double ci=x[0]*y[0], cs=ci, c;
#pragma omp parallel
{
    #pragma omp for private(j) reduction(max:cs) reduction(min:ci)
    for(i=0; i<N; i++) {
       ...
    }
    c=(ci+cs)/2.0;
    #pragma omp for private(j) reduction(+:count)
    for(i = 0; i < N; i++)
       ...
}
return count;
```

0.2 p. (b) Calcula el temps seqüencial i el temps paral·lel, tenint en compte que el cost de cadascuna de les comparacions `A[i][j]>cs`, `A[i][j]<ci` i `A[i][j]>c` és d'1 flop. Indica tots els passos en el càlcul dels temps.

Solució:

Cost seqüencial:

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

Cost paral·lel:

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

0.2 p.

(c) Modifica la implementació paral·lela de l'apartat (a) de manera que cada fil mostre en pantalla el nombre de voltes que `A[i][j]>c` en el bloc de la matriu `A` que li correspon; és a dir, el nombre de contribucions de cada un dels fils en el valor final de `count`. Per exemple, si el valor final de `count` fos igual a 10 i es tenen 4 fils, una possible eixida per pantalla seria la següent:

```
Fil 1:  2
Fil 0:  3
Fil 3:  3
Fil 2:  2.
```

Solució:

```
#pragma omp parallel
{
   int fil, countp=0;
   #pragma omp for private(j) reduction(max:cs) reduction(min:ci)
   for(i=0; i<N; i++)
      for (j=1; j<N; j++){
        .....
      }
   c=(ci+cs)/2.0;
   fil=omp_get_thread_num();
   #pragma omp for private(j) reduction(+:count)
   for(i=0; i<N; i++)
       for(j=0; j<N; j++){
          if (A[i][j]>c) {++count;++countp;}
       }
   printf("Fil %d: %d \n",fil,countp);
}
return count;
```

Qüestió 2 (1.1 punts)

Donades les següents funcions:

```
void prod(double A[M][N], double B[N][N]) {
  int i,j;
  for (i=0; i<N; i++) {
    A[i][i] = (A[i][i]+B[i][i])/2.0;
    for (j=0; j<i; j++) {
      A[i][j] = A[i][j]+A[i][j]*B[i][j];
    }
  }
}

void square(double A[M][N], double B[N][N]) {
  int i,j;
  for (i=0; i<N; i++) {
```

```
    for (j=0; j<N; j++) {
      A[i][j] = 2.0*A[i][j]+B[i][j]*B[i][j];
    }
  }
}
```

tenim un programa que realitza la següent seqüència d'operacions:

```
prod(D,C);     /* Tasca 1 */
square(E,C);   /* Tasca 2 */
square(C,E);   /* Tasca 3 */
prod(F,E);     /* Tasca 4 */
prod(F,F);     /* Tasca 5 */
prod(F,C);     /* Tasca 6 */
```

0.2 p. (a) Calcula el cost seqüencial asimptòtic en flops de cadascuna de les dues funcions.

Solució:

Cost de `prod`:

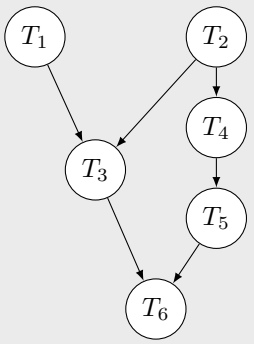
$$\sum_{i=0}^{N-1}\left(2+\sum_{j=0}^{i-1}2\right)=\sum_{i=0}^{N-1}(2+2i)\approx 2\sum_{i=0}^{N-1}i\approx N^2 \text{ flops}$$

Cost de `square`:

$$\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}3=\sum_{i=0}^{N-1}3N=3N^2 \text{ flops}$$

0.4 p. (b) Dibuixa el graf de dependències de tasques. Indica quin és el grau màxim de concurrència, el camí crític i la seua longitud i el grau mitjà de concurrència.

Solució:



Camí crític: T2→T3→T6.

Longitud del camí crític: $L = 3N^2 + 3N^2 + N^2 = 7N^2$ flops

Grau màxim de concurrència: 2

$$M=\frac{N^2+3N^2+3N^2+N^2+N^2+N^2}{7N^2}=\frac{10N^2}{7N^2}=\frac{10}{7}$$

0.5 p. (c) Implementa una versió paral·lela basada en seccions, a partir del graf de dependències anterior. Deu minimitzar-se el temps d'execució.

Solució:

```
#pragma omp parallel
{
  #pragma omp sections
  {
    #pragma omp section
    prod(D,C);     /* Tasca 1 */
    #pragma omp section
    square(E,C);   /* Tasca 2 */
  }
  #pragma omp sections
  {
    #pragma omp section
    square(C,E);   /* Tasca 3 */
    #pragma omp section
    {
      prod(F,E);     /* Tasca 4 */
      prod(F,F);     /* Tasca 5 */
    }
  }
} /* Fi del parallel */
prod(F,C);     /* Tasca 6 */
```

Qüestió 3 (1.2 punts)

La funció que es mostra a continuació rep un vector `V` que emmagatzema, en cadascun dels seus components, el nombre d'ordinadors infectats en els laboratoris informàtics de la UPV durant un mes (identificat de l'1 al 12) de l'any passat per part d'algun dels `*NV` virus coneguts fins al moment (numerats del 0 en avant). A partir d'eixos valors, la funció mostra per pantalla, per a cada virus, el seu identificador, el nombre total d'ordinadors infectats per ell al llarg de l'any i l'identificador del mes en què va infectar a més ordenadors de la UPV. Addicionalment, la funció retorna com a resultat el número total de virus que van infectar a algun ordinador, dels `*NV` coneguts, i completa el vector d'eixida `*idVirus` amb l'identificador d'aquests.

```
int gestiona_virus(struct Tvirus V[], int n,int idVirus[]) {
  int i, virus, mes, infectats;
  int total_infectats[NV],infectats_max[NV],mes_max[NV];
  int nVirus=0;

  for (i=0;i<NV;i++) {
    total_infectats[i]=0;
    infectats_max[i]=0;
    mes_max[i]=0;
  }

  for (i=0;i<n;i++) {
    infectats=V[i].infectats;
    mes=V[i].mes;
    virus=V[i].virus;
    total_infectats[virus]+=infectats;
    if (infectats>infectats_max[virus]) {
      infectats_max[virus]=infectats;
      mes_max[virus]=mes;
```

```
    }
  }

  for (i=0;i<NV;i++) {
    if (total_infectats[i]>0) {
      idVirus[nVirus]=i;
      nVirus++;
    }
  }

  for (i=0;i<NV;i++) {
    if (total_infectats[i]>0)
      printf("%d %d %d\n",i,total_infectats[i],mes_max[i]);
  }

  return nVirus;
}
```

Paralelitza la funció anterior de forma eficient mitjançant OpenMP, emprant una única regió paral·lela. No paral·lelitzes ni el primer ni l'últim dels bucles, encarregats d'inicialitzar els vectors a 0 i de mostrar els resultats per pantalla.

Solució:

```
int gestiona_virus(struct Tvirus V[], int n,int idVirus[]) {
  int i, virus, mes, infectats;
  int total_infectats[NV],infectats_max[NV],mes_max[NV];
  int nVirus=0;

  for (i=0;i<NV;i++) {
    total_infectats[i]=0;
    infectats_max[i]=0;
    mes_max[i]=0;
  }
  #pragma omp parallel
  {
    #pragma omp for private(infectats,mes,virus)
    for (i=0;i<n;i++) {
      infectats=V[i].infectats;
      mes=V[i].mes;
      virus=V[i].virus;
      #pragma omp atomic
      total_infectats[virus]+=infectats;
      if (infectats>infectats_max[virus]) {
        #pragma omp critical
        {
          if (infectats>infectats_max[virus]) {
            infectats_max[virus]=infectats;
            mes_max[virus]=mes;
          }
        }
      }
```

```
      }
      #pragma omp for
      for (i=0;i<NV;i++) {
        if (total_infectats[i]>0) {
          #pragma omp critical
          {
            idVirus[nVirus]=i;
            nVirus++;
          }
        }
      }
    }

    for (i=0;i<NV;i++) {
      if (total_infectats[i]>0)
        printf("%d %d %d\n",i,total_infectats[i],mes_max[i]);
    }

    return nVirus;
  }
```