Computació Paral · lela
Grau en Enginyeria Informàtica (ETSInf)
Curs 2022-23 ◇ Examen final 5/2/24 ◇ Bloc OpenMP ◇ Duració: 1h 30m

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Qüestió 1 (1.1 punts)
Donat el següent codi:

```
double f(double A[N][N], double B[N][N], double C[N][N], double t) {
  int i,j,k,c=0;
  double f=0, s=0, m, aux;
  for (i=0; i<N; i++) {
    m=0;
    for (j=0; j<i; j++) {
      aux=0;
      for (k=0; k<N; k++)
        aux += A[i][k]*B[k][j]*B[k][j];
      C[i][j] = aux;
      if (aux>m) m = aux;
      if (aux>t) c++;
      f += aux*aux;
    }
    s += m;
  }
  return (s+f)/c;
}
```

0.35 p. (a) Fes una versió paral · lela basada en la paral · lelització del bucle `i`.

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

```
#pragma omp parallel for private(m,j,aux,k) reduction(+:c,f,s)
```

0.45 p. (b) Fes una versió paral · lela basada en la paral · lelització del bucle `j`.

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

```
#pragma omp parallel for private(aux,k) reduction(max:m) reduction(+:c,f)
```

0.3 p. (c) Calcula el temps d'execució seqüencial en flops, detallant els passos. Se suposarà que les comparacions entre números reals no aporten cap flop.

Solució:

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

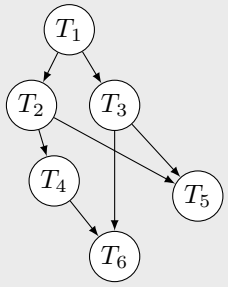
Qüestió 2 (1.2 punts)
Donada la següent funció, on `n` és una constant predefinida, suposem que les matrius `A` i `B` han sigut inicialitzades prèviament, i a més:

- La funció `processcol(A,i,x)` modifica la columna `i` de la matriu `A` 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.

```
double myfun(double A[n][n], double B[n][n], double C[n][n], double D[n][n]) {
  int i,j;
  double alpha=0.0,beta=1.0;
  for (i=0;i<n;i++) alpha += fabs(A[i][i]-B[i][i]);
  for (i=0;i<n;i++) processcol(A,i,alpha);
  for (i=0;i<n;i++) processcol(B,i,alpha);
  for (i=0;i<n;i++) beta *= A[i][n-i-1];
  for (i=0;i<n;i++) {
    for (j=0;j<n;j++) {
      C[i][j] = A[i][j]+0.5*B[i][j];
    }
  }
  for (i=0;i<n;i++) {
    for (j=0;j<n;j++) {
      D[i][j] = beta*B[i][j];
    }
  }
}
```

0.3 p. (a) Dibuixa el graf de dependències de dades entre les tasques, suposant que hi han 6 tasques corresponents a cadascun dels bucles `i`.

Solució:



0.6 p. (b) Implementa una versió paral · lela, basada en el graf i mitjançant OpenMP, utilitzant una sola regió paral · lela i usant paral · lelisme de tasques. Tin en compte els costos de cadascuna de les tasques per a tractar de reduir el temps d'execució de la funció paral · lela.

Solució: La tasca T_1 no és concurrent amb cap altra, pel que es pot fer fora de la regió paral · lela. Quant a les altres tasques, el graf ens ofereix diferents implementacions possibles. Per a determinar quina és la més adequada, cal tindre en compte el cost de les tasques:

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

La millor implementació usant la construcció `sections` seria agrupant les tasques T_4 i T_6.

```
double myfun_par(double A[n][n], double B[n][n], double C[n][n], double D[n][n]) {
  int i,j;
```

```
double alpha=0.0,beta=1.0;
for (i=0;i<n;i++) alpha += fabs(A[i][i]-B[i][i]);   /* T1 */
#pragma omp parallel private(i,j)
{
  #pragma omp sections
  {
    #pragma omp section
    for (i=0;i<n;i++) processcol(A,i,alpha);   /* T2 */
    #pragma omp section
    for (i=0;i<n;i++) processcol(B,i,alpha);   /* T3 */
  }
  #pragma omp sections
  {
    #pragma omp section
    {
      for (i=0;i<n;i++) beta *= A[i][n-i-1];   /* T4 */
      for (i=0;i<n;i++) {                      /* T6 */
        for (j=0;j<n;j++) {
          D[i][j] = beta*B[i][j];
        }
      }
    }
    #pragma omp section
    {
      for (i=0;i<n;i++) {                      /* T5 */
        for (j=0;j<n;j++) {
          C[i][j] = A[i][j]+0.5*B[i][j];
        }
      }
    }
  }
}
```

0.3 p. (c) Obtín el grau mitjà de concurrència del graf.

Solució: Tenint en compte els costos obtinguts en l'apartat anterior, el cost seqüencial és:

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

El camí crític és T_1 – T_3 – T_5, amb una longitud de:

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

Per tant, el grau mitjà de concurrència és:

$$M = \frac{t(n)}{L} = \frac{7n^2}{4n^2} = \frac{7}{4}$$

Qüestió 3 (1.2 punts)
Donada la següent funció:

```
int fun(int v[], int n) {
  int i, max, sum, ind;
  int count[100];

  for (i=0;i<100;i++)
    count[i]= 0;

  for (i=0;i<n;i++)
     count[v[i]%100]++;

  sum = 0;
  for (i=0;i<100;i++)
     sum += count[i];
  sum /= 100;

  max = count[0];
  ind = 0;
  for (i=1;i<100;i++)
     if (count[i]>sum)
       if (count[i]>max) {
          max = count[i];
          ind = i;
       }
  return ind;

}
```

1 p. (a) Realitza una versió paral · lela eficient mitjançant OpenMP. Nota: No és necessari utilitzar una única regió paral · lela.

Solució:

```
int funpar(int v[], int n) {
  int i, max, ind, sum=0;
  int count[100];

  #pragma omp parallel for
  for (i=0;i<100;i++)
    count[i]= 0;

  #pragma omp parallel for
  for (i=0;i<n;i++)
     #pragma omp atomic
     count[v[i]%100]++;

  sum = 0;
  #pragma omp parallel for reduction(+:sum)
  for (i=0;i<100;i++)
     sum += count[i];
  sum /= 100;
```

```
        max = count[0];
        ind = 0;
        #pragma omp parallel for
        for (i=1;i<100;i++)
           if (count[i]>sum)
             if (count[i]>max)
               #pragma omp critical
               if (count[i]>max) {
                  max = count[i];
                  ind = i;
               }
        return ind;

      }
```

0.2 p.

(b) Realitza una nova versió paral · lela, el més eficient possible, si la part final de la funció (càlcul de `max` e `ind`) es modificara d'acord amb el següent fragment de codi:

```
...
max = count[0];
for (i=1;i<100;i++)
  if (count[i]>sum)
     if (count[i]>max)
        max = count[i];
return max;
```

Solució:

```
...
max = count[0];
#pragma omp parallel for reduction(max:max)
for (i=1;i<100;i++)
  if (count[i]>sum)
    if (count[i]>max)
        max = count[i];
return max;
```