Computación Paralela
Grado en Ingeniería Informática (ETSInf)
Curso 2021-22 ◇ Recuperación 28/1/22 ◇ Bloque OpenMP ◇ Duración: 1h 30m

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Cuestión 1 (1.2 puntos)
Dada la siguiente función:

```
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) Paraleliza mediante OpenMP la función anterior, usando para ello una sola región paralela.

Solución:

```
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 tiempo secuencial y el tiempo paralelo, teniendo en cuenta que el coste de cada una de las comparaciones `A[i][j]>cs`, `A[i][j]<ci` y `A[i][j]>c` es de 1 flop. Indica todos los pasos en el cálculo de los tiempos.

Solución:
Coste secuencial:

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

Coste paralelo:

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

0.2 p.

(c) Modifica la implementación paralela del apartado (a) de manera que cada hilo muestre en pantalla el número de veces que `A[i][j]>c` en el bloque de la matriz `A` que le corresponde; es decir, el número de contribuciones de cada uno de los hilos en el valor final de `count`. Por ejemplo, si el valor final de `count` fuese igual a 10 y se tienen 4 hilos, una posible salida por pantalla sería la siguiente:

```
Hilo 1: 2
Hilo 0: 3
Hilo 3: 3
Hilo 2: 2.
```

Solución:

```
#pragma omp parallel
{
   int hilo, 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;
   hilo=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("Hilo %d: %d \n",hilo,countp);
}
return count;
```

Cuestión 2 (1.1 puntos)

Dadas las siguientes funciones:

```
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];
    }
  }
}
```

tenemos un programa que realiza la siguiente secuencia de operaciones:

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

0.2 p. (a) Calcula el coste secuencial asintótico en flops de cada una de las dos funciones.

Solución:

Coste de `prod`:

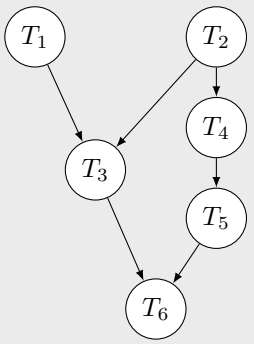
$$\sum_{i=0}^{N-1}\left(2+\sum_{j=0}^{i-1}2\right)=\sum_{i=0}^{N-1}(2+2i)\approx 2\sum_{i=0}^{N-1}i\approx N^2 \text{ flops}$$

Coste 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) Dibuja el grafo de dependencias de tareas. Indica cuál es el grado máximo de concurrencia, el camino crítico y su longitud y el grado medio de concurrencia.

Solución:



Camino crítico: T2→T3→T6.

Longitud del camino crítico: $L = 3N^2 + 3N^2 + N^2 = 7N^2$ flops

Grado máximo de concurrencia: 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) Haz una versión paralela basada en secciones, a partir del grafo de dependencias anterior. Debe minimizarse el tiempo de ejecución.

Solución:

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

Cuestión 3 (1.2 puntos)

La función que a continuación se recoge recibe un vector `V` que almacena, en cada uno de sus componentes, el número de ordenadores infectados en los laboratorios informáticos de la UPV durante un mes (identificado del 1 al 12) del año pasado por parte de alguno de los `NV` virus conocidos hasta el momento (numerados del 0 en adelante). A partir de esos valores, la función muestra por pantalla, para cada virus, su identificador, el número total de ordenadores infectados por el mismo a lo largo del año y el identificador del mes en el que infectó a más ordenadores de la UPV. Adicionalmente, la función devuelve como resultado el número total de virus que infectaron a algún ordenador, de los `NV` conocidos, y completa el vector de salida `idVirus` con el identificador de los mismos.

```
int gestiona_virus(struct Tvirus V[], int n,int idVirus[]) {
  int i, virus, mes, infectados;
  int total_infectados[NV],infectados_max[NV],mes_max[NV];
  int nVirus=0;

  for (i=0;i<NV;i++) {
    total_infectados[i]=0;
    infectados_max[i]=0;
    mes_max[i]=0;
  }

  for (i=0;i<n;i++) {
    infectados=V[i].infectados;
    mes=V[i].mes;
    virus=V[i].virus;
    total_infectados[virus]+=infectados;
    if (infectados>infectados_max[virus]) {
      infectados_max[virus]=infectados;
```

```
      mes_max[virus]=mes;
    }
  }

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

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

  return nVirus;
}
```

Paraleliza la función anterior de forma eficiente mediante OpenMP, empleando una única región paralela. No paralelices ni el primero ni el último de los bucles, encargados de inicializar los vectores a 0 y de mostrar los resultados por pantalla.

Solución:

```
int gestiona_virus(struct Tvirus V[], int n,int idVirus[]) {
  int i, virus, mes, infectados;
  int total_infectados[NV],infectados_max[NV],mes_max[NV];
  int nVirus=0;

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

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

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

    return nVirus;
  }
```