Computación Paralela
Grado en Ingeniería Informática (ETSInf)
Curso 2022-23 ◇ Examen parcial 9/11/22 ◇ Bloque OpenMP ◇ Duración: 1h 30m

Cuestión 1 (1.2 puntos)
Dada la siguiente función:

```
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) Paraleliza el bucle externo mediante OpenMP.

Solución: Bastaría con añadir la siguiente directiva justo antes del bucle:

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

0.5 p. (b) Paraleliza los dos bucles internos usando una sola región paralela. Elimina las barreras implícitas innecesarias, si las hubiera.

Solución:

```
...
  c=0;            /* Todo igual hasta esta línea */
  #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;   /* Todo igual después de esta línea */
  ...
```

0.1 p. (c) Calcula el coste secuencial, indicando todos los pasos.

Solución:

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

0.3 p. (d) Supongamos que se paraleliza solo el primer bucle `j`. Calcula el coste paralelo, indicando todos los pasos. Calcula el speedup cuando p tiende a infinito.

Solución: Coste paralelo:

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

Cuando p tiende a infinito, $t(N,p) \approx \frac{N^2}{2}$, y por tanto el speedup será

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

Cuestión 2 (1.3 puntos)

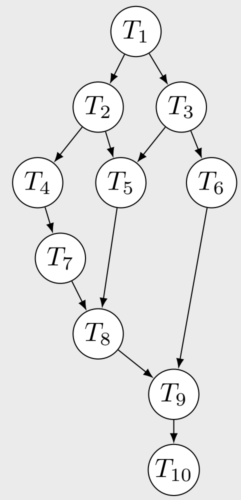
Dado el siguiente fragmento de código, donde `n` es una constante predefinida, suponemos que las matrices han sido rellenadas previamente, y teniendo en cuenta que las tres funciones `f1, f2 y f3` modifican el segundo argumento y tienen un coste computacional de $\frac{1}{3}n^3$ flops, n^3 flops y $2n^3$ flops respectivamente, realiza los siguientes apartados:

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

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

0.3 p. (a) Dibuja el grafo de dependencias de datos entre las tareas.

Solución:



0.6 p. (b) Implementa una versión paralela mediante OpenMP utilizando una sola región paralela.

Solución: La tarea T_1 no es concurrente con ninguna otra, por lo que se puede hacer fuera de la región paralela. Las tareas T_7, T_8, T_9, y T_{10} se han de ejecutar necesariamente de forma secuencial, una detrás de la otra. Por tanto, se pueden dejar fuera de la región paralela. La mejor solución consistiría en agregar las tareas T_4 y T_7 para que las realice el mismo hilo (en la misma sección). De esa forma, la tarea T_7 se hará en paralelo con las tareas T_5 y T_6.

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

0.4 p. (c) Obtén el speedup y la eficiencia de la versión paralela del apartado anterior suponiendo que se ejecuta con

4 hilos en un computador con 4 procesadores (núcleos).

Solución: Tiempo de ejecución secuencial:

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

Tiempo de ejecución paralelo para $p = 4$:

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

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

Speedup:

$$S(n,p) = \frac{11n^3}{\frac{20}{3}n^3} = 1{,}65$$

Eficiencia:

$$E(n,p) = \frac{1{,}65}{4} = 0{,}41$$

Cuestión 3 (1 punto)

Dada la siguiente función, en la que la llamada a la función `aleatorio` devuelve un entero aleatorio entre los límites indicados en sus argumentos.

```
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 = aleatorio(0,100);
    iy = aleatorio(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("Posición (%d,%d) con %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;
}
```

Paraleliza, usando una única región paralela, esta función de la forma más eficiente posible utilizando OpenMP.

Solución:

```
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 = aleatorio(0,100);
      iy = aleatorio(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("Posición (%d,%d) con %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;
}
```