Práctica número 3
Programación con OpenMP

Ejercicio 1. El programa ejercicio1.c inicializa una matriz de NxN de la siguiente manera: A[i,j]=i*j, para todo i,j=0..N-1. Compile y ejecute. ¿Qué problemas tiene el programa? Corríjalo.

El problema inicial radica en la línea 22 donde podemos evidenciar la inicialización de la matriz:

```
21  for(i=0;i<N;i++){
22     #pragma omp parallel for shared(A) private(i,j)
23     for(j=0;j<N;j++){
24      A[i*N+j]=i*j;
25     }
26    }
```

esto implica que la parte que se hará paralelamente usando openmp va a ser solo el bucle más interno esto puede dar paso a errores como **acceder a zonas de memoria** en las cuales no podemos ya que cada hilo tendrá una copia de i distinta lo cual puede llegar a que cada hilo tenga i con valor basura, el codigo arreglado quedaria de la siguiente manera:

```
21  #pragma omp parallel for private(j)
22  for (i = 0; i < N; i++)
23  {
24    for (j = 0; j < N; j++)
25    {
26      A[i * N + j] = i * j;
27    }
28  }
```

esto puede ser de esta forma ya que por defecto la cláusula **for** tiene sus variables de iteración privadas, y el tema de remover la **A** como **shared()** se debe a que cada hilo trabajara sobre una fila distinta, esto lleva a que no haya solapamientos y este efecto conlleva a que no exista ninguna inconsistencia(j tiene que ser privado porque el **scope** de la paralelización es lo que está inmediatamente abajo, por ende el bucle **for** más interno actuaría como bucle **for** normal definido por el lenguaje).

Ejercicio 2. Analice y compile el programa ejercicio2.c. Ejecute varias veces y compare los resultados de salida para diferente número de threads. ¿Cuál es el problema? ¿Se le ocurre una solución?

Nota: al compilar, agregue el flag -lm.

```
17  #pragma omp parallel for
18   for(i=1;i≤N;i++){
19    x= x + sqrt(i*scale) + 2*x;
20   }
```

El problema está en este fragmento de código porque la suma que se hace es acumulativa sobre **x**. Al paralelizarlo, no tenemos la certeza de que los valores de **x** no se estén pisando. En este caso, no tiene mucho sentido paralelizar por la naturaleza de la cuenta. Lo único que se podría paralelizar es la parte que calcula la raíz, pero no creemos que tenga mucho sentido.

Ejercicio 3. El programa matrices.c realiza la multiplicación de 2 matrices cuadradas de NxN (C=AxB).
Utilizando la directiva parallel for paralelice de dos formas:
a. Repartiendo entre los threads el cálculo de las filas de C. Es decir, repartiendo el trabajo del primer for.

```
40  #pragma omp parallel for private(j,k)
41   for (i = 0; i < N; i++)
42   {
43     for (j = 0; j < N; j++)
44     {
45       C[i * N + j] = 0;
46       double sum = 0;
47       for (k = 0; k < N; k++)
48       {
49         sum += A[i * N + k] * B[k + j * N];
50       }
51       C[i * N + j] = sum;
52     }
53   }
```

b. Repartiendo el cálculo de las columnas de cada fila de C. Es decir, repartiendo el trabajo del segundo for.

```
40  for (i = 0; i < N; i++)
41  {
42    #pragma omp parallel for private(k) shared(i)
43    for (j = 0; j < N; j++)
44    {
45      C[i * N + j] = 0;
46      double sum = 0;
47      for (k = 0; k < N; k++)
48      {
49        sum += A[i * N + k] * B[k + j * N];
50      }
51      C[i * N + j] = sum;
52    }
53  }
```

Compare los tiempos de ambas soluciones variando el número de threads.
Debido a que en el inciso **b** nos piden paralelizar el cálculo de cada columna, esto genera a que por cada fila se generen varios hilos, el resultado puede llegar a ser el mismo pero estamos haciendo overhead en la creación de hilos, esto puede llevar a desmejoras en el rendimiento del algoritmo calculador, en concreto es un **~7%** menos rápido con respecto al paralelizar la matriz en su totalidad (por filas)

Ejercicio 4. El programa traspuesta.c calcula la transpuesta de una matriz triangular de NxN. Compile y ejecute para 4 threads comparándolo con el algoritmo secuencial.
Si bien el programa computa correctamente la transpuesta, éste tiene un problema desde el punto de vista del rendimiento. Analice las salidas y describa de qué problema se trata. ¿Qué cláusula se debe usar para corregir el problema? Describa brevemente la cláusula OpenMP que resuelve el problema y las opciones que tiene. Corrija el algoritmo y ejecute de nuevo comparando con los resultados anteriores.
Existe un desbalance de carga entre cada iteración, al inicio de ejecutar el programa sin ninguna cláusula adicional, los tiempos entre cada hilo eran bastante diferentes, pero al agregar la cláusula **schedule()**, logramos que esta carga se balancee debido a que le estamos diciendo al compilador cómo queremos que estos bucles sean repartidos a lo largo de la ejecución. En nuestro caso, la planificación que mejores resultados dio fue **dynamic**, esta planificación realiza la asignación de iteración en demanda, divide las iteraciones basadas en una variable que se le pasa como parámetro, en nuestro caso las mejores respuestas fueron bajo el número de *chunk 5 o a veces chunk 1*.

Ejercicio 5. El programa mxm.c realiza 2 multiplicaciones de matrices de MxM (D=AxB y E=CxB). Paralelizar utilizando sections de forma que cada una de las multiplicaciones se realice en una sección y almacenar el código paralelo como mxmSections.c. Compile y ejecute con 2 threads y luego con 4 threads, ¿se consigue mayor speedup al incrementar la cantidad de threads? ¿Por qué?

No se ven diferencias al incrementar los hilos porque al usar sections lo que se hace es asignar un hilo para cada sección. Aquí, como solo tenemos dos secciones, usar más de dos hilos no tiene sentido ya que no se utilizan. incluso es un poco peor incrementar la cantidad de hilos debido a que debe realizarse el cómputo de su creación y control. Por otro lado el **speedup** se calcula en base al tiempo que tarda el **algoritmo secuencial** con respecto al **algoritmo paralelo**, por ende en base a los resultados dados por cada algoritmo
Tamaño: 1024
observación: se hizo una pequeña optimización en el tema de acceso a memoria
mxm secuencial: 5,975561 s
mxm paralelo (sections): 2,990613 s

$$Speedup = \frac{Tsecuencial}{Tparalelo} = \frac{5{,}975561}{2{,}990613} = 1,998105$$

como puedes ver, el secuencial tarda aproximadamente **~200% (el doble)** más que el paralelo, esto debido a la naturaleza del problema, el secuencial calcula primero una multiplicación y luego la otra, en cambio la paralela realiza las dos en simultáneo por ende agiliza la tarea de computar la multiplicación, llevando a que el resultado sea aproximadamente el **doble**.