Practica 3

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 parallel for está mal implementado. En el código se ve así:

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

   Corrección:

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

   O

```
 for (i = 0; i < N; i++)
 {
       #pragma omp parallel for shared(A) private(j)
    for (j = 0; j < N; j++)
    {
       A[i * N + j] = i * j;
    }
 }
```

   Ambas son soluciones correctas pero la primera es mejor respecto a la segunda dado que evita el overhead de creación y destrucción constante de hilos, ya que se van a

crear y destruir N veces, mientras que en la primera solución solo se crean y se destruyen una sola vez.

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.

```
agusnfr@agusnfr-VirtualBox:~/Paralelos/Practica 3/fuentes_omp$ ./ejercicio2 10 2

 Resultado: 58454.162468
agusnfr@agusnfr-VirtualBox:~/Paralelos/Practica 3/fuentes_omp$ ./ejercicio2 10 2

 Resultado: 124810.382291
agusnfr@agusnfr-VirtualBox:~/Paralelos/Practica 3/fuentes_omp$ ./ejercicio2 10 2

 Resultado: 58454.162468
agusnfr@agusnfr-VirtualBox:~/Paralelos/Practica 3/fuentes_omp$ ./ejercicio2 10 2

 Resultado: 58454.162468
agusnfr@agusnfr-VirtualBox:~/Paralelos/Practica 3/fuentes_omp$ ./ejercicio2 10 4

 Resultado: 144560.575178
agusnfr@agusnfr-VirtualBox:~/Paralelos/Practica 3/fuentes_omp$ ./ejercicio2 10 4

 Resultado: 5457.622663
agusnfr@agusnfr-VirtualBox:~/Paralelos/Practica 3/fuentes_omp$ ./ejercicio2 10 4

 Resultado: 19701.291106
agusnfr@agusnfr-VirtualBox:~/Paralelos/Practica 3/fuentes_omp$ ./ejercicio2 10 6

 Resultado: 133531.770369
agusnfr@agusnfr-VirtualBox:~/Paralelos/Practica 3/fuentes_omp$ ./ejercicio2 10 6

 Resultado: 132466.048899
```

Siempre da distintos resultados, cuando no debería.
El algoritmo no precisa ser paralelizado, paralelizar implica una solución más compleja y que a la larga los overheads de la paralización harán que sea más eficiente respecto a la solución secuencial.

```c
// Ejercicio 2
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main(int argc, char *argv[])
```

```
{
   double x, partial, scale;
   int i;
   int N = atoi(argv[1]);
   scale = 2.78;
   x = 0.0;

   for (i = 1; i <= N; i++)
   {
       x = x + sqrt(i * scale) + 2 * x;
   }

   printf("\n Resultado: %f \n", x);

   return (0);
}
```

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

    Compare los tiempos de ambas soluciones variando el número de threads.

```
agusnfr@agusnfr-VirtualBox:~/Paralelos/Practica 3/fuentes_omp$ ./matricesA 1024 4
Tiempo en segundos 1.376577
Multiplicacion de matrices resultado correcto
agusnfr@agusnfr-VirtualBox:~/Paralelos/Practica 3/fuentes_omp$ ./matricesB 1024 4
Tiempo en segundos 1.356072
Multiplicacion de matrices resultado correcto
agusnfr@agusnfr-VirtualBox:~/Paralelos/Practica 3/fuentes_omp$ ./matricesA 1024 8
Tiempo en segundos 1.351759
Multiplicacion de matrices resultado correcto
agusnfr@agusnfr-VirtualBox:~/Paralelos/Practica 3/fuentes_omp$ ./matricesB 1024 8
Tiempo en segundos 1.536152
Multiplicacion de matrices resultado correcto
agusnfr@agusnfr-VirtualBox:~/Paralelos/Practica 3/fuentes_omp$ ./matricesA 1024 16
Tiempo en segundos 1.340538
Multiplicacion de matrices resultado correcto
agusnfr@agusnfr-VirtualBox:~/Paralelos/Practica 3/fuentes_omp$ ./matricesB 1024 16
Tiempo en segundos 1.547171
Multiplicacion de matrices resultado correcto
agusnfr@agusnfr-VirtualBox:~/Paralelos/Practica 3/fuentes_omp$ ./matricesA 1024 32
Tiempo en segundos 1.147607
Multiplicacion de matrices resultado correcto
agusnfr@agusnfr-VirtualBox:~/Paralelos/Practica 3/fuentes_omp$ ./matricesB 1024 32
Tiempo en segundos 1.567090
Multiplicacion de matrices resultado correcto
agusnfr@agusnfr-VirtualBox:~/Paralelos/Practica 3/fuentes_omp$ ./matricesA 2048 32
Tiempo en segundos 9.874625
Multiplicacion de matrices resultado correcto
agusnfr@agusnfr-VirtualBox:~/Paralelos/Practica 3/fuentes_omp$ ./matricesb 2048 32
bash: ./matricesb: No existe el archivo o el directorio
agusnfr@agusnfr-VirtualBox:~/Paralelos/Practica 3/fuentes_omp$ ./matricesB 2048 32
Tiempo en segundos 10.737145
Multiplicacion de matrices resultado correcto
agusnfr@agusnfr-VirtualBox:~/Paralelos/Practica 3/fuentes_omp$ ./matricesA 4096 32
Tiempo en segundos 73.262053
Multiplicacion de matrices resultado correcto
agusnfr@agusnfr-VirtualBox:~/Paralelos/Practica 3/fuentes_omp$ ./matricesB 4096 32
Tiempo en segundos 81.457214
Multiplicacion de matrices resultado correcto
```

La opción B es la que tiene un peor rendimiento por el mismo motivo que explique en el punto 1.

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.

```
agusnfr@agusnfr-VirtualBox:~/Paralelos/Practica 3/fuentes_omp$ ./ejercicio4Par 512 4
Tiempo en segundos para el thread 3: 0.000050
Tiempo en segundos para el thread 2: 0.000161
Tiempo en segundos para el thread 1: 0.000270
Tiempo en segundos para el thread 0: 0.000406
Resultado correcto
agusnfr@agusnfr-VirtualBox:~/Paralelos/Practica 3/fuentes_omp$ ./ejercicio4Par 1028 4
Tiempo en segundos para el thread 3: 0.000303
Tiempo en segundos para el thread 2: 0.001216
Tiempo en segundos para el thread 1: 0.001490
Tiempo en segundos para el thread 0: 0.001996
Resultado correcto
agusnfr@agusnfr-VirtualBox:~/Paralelos/Practica 3/fuentes_omp$ ./ejercicio4Par 2048 4
Tiempo en segundos para el thread 3: 0.001360
Tiempo en segundos para el thread 2: 0.011354
Tiempo en segundos para el thread 1: 0.033366
Tiempo en segundos para el thread 0: 0.049224
Resultado correcto
agusnfr@agusnfr-VirtualBox:~/Paralelos/Practica 3/fuentes_omp$ ./ejercicio4Par 4096 4
Tiempo en segundos para el thread 3: 0.014725
Tiempo en segundos para el thread 2: 0.091111
Tiempo en segundos para el thread 1: 0.153808
Tiempo en segundos para el thread 0: 0.205787
Resultado correcto
agusnfr@agusnfr-VirtualBox:~/Paralelos/Practica 3/fuentes_omp$ ./ejercicio4Sec 512
Tiempo en segundos: 0.000909
Resultado correcto
agusnfr@agusnfr-VirtualBox:~/Paralelos/Practica 3/fuentes_omp$ ./ejercicio4Sec 1028
Tiempo en segundos: 0.003999
Resultado correcto
agusnfr@agusnfr-VirtualBox:~/Paralelos/Practica 3/fuentes_omp$ ./ejercicio4Sec 2048
Tiempo en segundos: 0.073319
Resultado correcto
agusnfr@agusnfr-VirtualBox:~/Paralelos/Practica 3/fuentes_omp$ ./ejercicio4Sec 4096
Tiempo en segundos: 0.347134
Resultado correcto
```

Hay una notable diferencia en el tiempo de ejecución entre los distintos hilos del mismo programa. Se debería usar la cláusula **schedule static o dynamic** para mejorar el equilibrio de carga entre los hilos y reducir la variación en el tiempo de ejecución. De este modo va a haber distribución más equitativa del trabajo (caso static) o un hilo que terminó más rápido de realizar su trabajo va a poder solicitar más (caso dynamic)

chedule(política [,chunk]): especifica cómo se distribuyen las iteraciones entre los hilos.

static: divide en bloques de chunk iteraciones y las asigna en forma round-robin. Cuando chunk no se especifica, se dividen las iteraciones en bloques de tamaño aproximado.

dynamic: divide en bloques de chunk iteraciones y las asigna bajo demanda. Cuando chunk no se especifica, las iteraciones son asignadas de a 1.

```
#pragma omp parallel default(none) private(i, j, temp, timetick, tid) shared(A, N)
```

```
 {
   tid = omp_get_thread_num();
   timetick = dwalltime();
#pragma omp for private(i, j, temp) schedule(static, 1) nowait
   for (i = 0; i < N; i++)
   {
     for (j = i + 1; j < N; j++)
     {
       temp = A[i * N + j];
       A[i * N + j] = A[j * N + i];
       A[j * N + i] = temp;
     }
   }
   printf("Tiempo en segundos para el thread %d: %f \n", tid,
dwalltime() - timetick);
 }
```

```
#pragma omp parallel default(none) private(i, j, temp, timetick,
tid) shared(A, N)
 {
   tid = omp_get_thread_num();
   timetick = dwalltime();
#pragma omp for private(i, j, temp) schedule(dynamic, 1) nowait
   for (i = 0; i < N; i++)
   {
     for (j = i + 1; j < N; j++)
     {
       temp = A[i * N + j];
       A[i * N + j] = A[j * N + i];
       A[j * N + i] = temp;
     }
   }
   printf("Tiempo en segundos para el thread %d: %f \n", tid,
dwalltime() - timetick);
 }
```

Static:

```
agusnfr@agusnfr-VirtualBox:~/Paralelos/Practica 3/fuentes_omp$ ./ejercicio4Par 512 4
Tiempo en segundos para el thread 2: 0.000587
Tiempo en segundos para el thread 0: 0.000586
Tiempo en segundos para el thread 1: 0.000627
Tiempo en segundos para el thread 3: 0.000628
Resultado correcto
agusnfr@agusnfr-VirtualBox:~/Paralelos/Practica 3/fuentes_omp$ ./ejercicio4Par 1024 4
Tiempo en segundos para el thread 0: 0.002876
Tiempo en segundos para el thread 1: 0.002876
Tiempo en segundos para el thread 3: 0.002876
Tiempo en segundos para el thread 2: 0.002876
Resultado correcto
agusnfr@agusnfr-VirtualBox:~/Paralelos/Practica 3/fuentes_omp$ ./ejercicio4Par 2048 4
Tiempo en segundos para el thread 1: 0.029672
Tiempo en segundos para el thread 3: 0.029671
Tiempo en segundos para el thread 0: 0.034276
Tiempo en segundos para el thread 2: 0.026565
Resultado correcto
agusnfr@agusnfr-VirtualBox:~/Paralelos/Practica 3/fuentes_omp$ ./ejercicio4Par 4096 4
Tiempo en segundos para el thread 0: 0.133238
Tiempo en segundos para el thread 2: 0.137619
Tiempo en segundos para el thread 1: 0.137733
Tiempo en segundos para el thread 3: 0.147805
Resultado correcto
```

Dynamic:

```
agusnfr@agusnfr-VirtualBox:~/Paralelos/Practica 3/fuentes_omp$ ./ejercicio4Par 512 4
Tiempo en segundos para el thread 3: 0.000680
Tiempo en segundos para el thread 2: 0.000680
Tiempo en segundos para el thread 0: 0.000680
Tiempo en segundos para el thread 1: 0.000680
Resultado correcto
agusnfr@agusnfr-VirtualBox:~/Paralelos/Practica 3/fuentes_omp$ ./ejercicio4Par 1024 4
Tiempo en segundos para el thread 1: 0.003051
Tiempo en segundos para el thread 3: 0.001248
Tiempo en segundos para el thread 2: 0.003050
Tiempo en segundos para el thread 0: 0.003051
Resultado correcto
agusnfr@agusnfr-VirtualBox:~/Paralelos/Practica 3/fuentes_omp$ ./ejercicio4Par 2048 4
Tiempo en segundos para el thread 3: 0.022494
Tiempo en segundos para el thread 0: 0.022497
Tiempo en segundos para el thread 2: 0.022497
Tiempo en segundos para el thread 1: 0.022497
Resultado correcto
agusnfr@agusnfr-VirtualBox:~/Paralelos/Practica 3/fuentes_omp$ ./ejercicio4Par 4096 4
Tiempo en segundos para el thread 1: 0.112686
Tiempo en segundos para el thread 3: 0.112685
Tiempo en segundos para el thread 0: 0.112686
Tiempo en segundos para el thread 2: 0.112685
Resultado correcto
```

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?

N = 512

Tiempo en segundos 1.198990

Tiempo en segundos paralelo 2 0.654549

Tiempo en segundos paralelo 4 0.655228

Comparación respecto a 2 hilos → 1.198990/0.654549 = 1.832
Comparación respecto a 4 hilos →1.198990/0.655228 = 1.829

N = 1024

Tiempo en segundos 9.469963

Tiempo en segundos paralelo 2 4.782443

Tiempo en segundos paralelo 4 4.823066

Comparación respecto a 2 hilos →9.469963/4.782443 = 1.980
Comparación respecto a 4 hilos →9.469963/4.823066 = 1.963

N = 2048

Tiempo en segundos 72.926593

Tiempo en segundos paralelo 2 35.513278

Tiempo en segundos paralelo 4 35.440863

Comparación respecto a 2 hilos →72.926593/35.513278 = 2.053
Comparación respecto a 4 hilos →72.926593/35.440863 = 2.057

No, no se consigue mayor speedup al aumentar la cantidad de hilos. Cada sección es independiente de las demás y es ejecutada una sola vez por un único hilo. Como solamente hay 2 secciones, solo 2 hilos van a ejecutar las mismas a pesar de que se declare más o igual cantidad de hilos.