

Actividad - 2

Juan Diego Valencia Alomia

1.a

En esta implementación se usó un buffer circular con `'pthread_mutex_t'`. La sincronización evita condiciones de carrera y demuestra cómo compartir recursos de forma segura en memoria compartida.

```
japetc@73921a3c2dfe:~/app/Actividad2$ ./a.out
Productor 0 produjo: 0
Productor 0 produjo: 1
Productor 0 produjo: 2
Productor 0 produjo: 3
Productor 0 produjo: 4
Productor 0 produjo: 5
Productor 0 produjo: 6
Productor 0 produjo: 7
Productor 0 produjo: 8
Productor 0 produjo: 9
Consumidor 0 consumió: 0
Productor 0 produjo: 10
```

1.b

La implementación de la multiplicación de matriz por vector divide las filas de la matriz entre los hilos. Cada hilo procesa un subconjunto de filas y acumula sus resultados en el vector final. Esto permite aprovechar el paralelismo reduciendo el tiempo de cómputo frente a la versión secuencial.

```
japetc@73921a3c2dfe:~/app/Actividad2$ g++ P1-MatVec.cpp -lpthread
japetc@73921a3c2dfe:~/app/Actividad2$ ./a.out
Resultado y = A * x :
168
186
204
222
```

1.c

El método de la regla trapezoidal fue paralelizado dividiendo el intervalo de integración entre los hilos. Cada hilo calcula su aproximación local y se acumula en una variable compartida protegida por exclusión mutua. Los resultados obtenidos coinciden con la integral esperada con un error mínimo.

```
japeto@73921a3c2dfe:~/app/Actividad2$ ./a.out
Método del trapecio paralelo
Integral en [0.000000, 3.141593]  $\approx$  1.999998
Error absoluto: 0.000002
japeto@73921a3c2dfe:~/app/Actividad2$
```

1.d

En la versión paralela del algoritmo Count Sort, cada hilo procesa un segmento del arreglo y determina las posiciones de sus elementos en el arreglo resultado. Los tiempos de ejecución muestran una mejora significativa frente a la versión secuencial.

```
japeto@73921a3c2dfe:~/app/Actividad2$ g++ sumMat.cpp -lpthread
japeto@73921a3c2dfe:~/app/Actividad2$ ./a.out
La suma total es: 504911263
Verificación secuencial: 504911263
japeto@73921a3c2dfe:~/app/Actividad2$
```

2.a

En este apartado se implementó la suma de un arreglo grande utilizando hilos POSIX y exclusión mutua. Cada hilo calcula la suma parcial de su segmento y actualiza una variable compartida protegida con un mutex. Se evidencia una mejora significativa ante su versión secuencial

```
japeto@73921a3c2dfe:~/app/Actividad2$ ./a.out
La suma total (paralelo) es: 505007927
Tiempo CPU paralelo: 0.000366 segundos
Verificación secuencial: 505007927
Tiempo CPU secuencial: 0.026243 segundos
```

2.b

Aquí se presenta la versión con OpenMP, usando la directiva `#pragma omp parallel for reduction(+:sum)`. Esta técnica evita la necesidad de exclusión mutua y reduce la sobrecarga de sincronización, logrando un mejor desempeño en el cálculo de la suma. Los tiempos obtenidos confirman la eficiencia de esta aproximación.

```
japeto@73921a3c2dfe:~/app/Actividad2$ ./a.out
Suma paralela: 505076299
Tiempo paralelo: 0.0116411 segundos
```

3.

La multiplicación de matrices se paralelizó asignando bloques de filas de la matriz A a distintos hilos, quienes escriben sus resultados en la matriz C. Este enfoque evita conflictos de escritura y mejora los tiempos de ejecución a medida que las dimensiones de las matrices crecen. La comparación con la versión secuencial confirma la ganancia en eficiencia.

```
japeto@73921a3c2dfe:~/app/Actividad2$ g++ P3-MatMult.cpp -lpthread
japeto@73921a3c2dfe:~/app/Actividad2$ ./a.out
Matriz A:
    902      710      275      977      713      948
    549      411      583      540      365      336
    668      854      565      109      880      577
    514      261       73      833       32      328
    288      261       96      735       77      332
    791      152      441      665      543      908
Matriz B:
    961      305      895      398      408       76
    539       63      318       2      706      734
    432      347      673      123      611      156
     82       66      677      167      847      922
    189      588      201       71      976      258
    510      676      762      960      612      432
Resultado C = A * B:
  2066663  1539839  2745263  1518103  3140884  2126876
  1285599   873035  1709389   729688  1889623  1171548
  1815862  1368283  1940024   971670  2525010  1342546
   907803   494066  1412466   670336  1376100  1160004
   703062   455813  1171422   573886  1261307  1069398
  1652828  1380840  2304318  1390653  2348410  1385960
Tiempo de ejecucion: 0.000649765 segundos
```