Link GitHub: https://github.com/benjamin94773/CL05_20082025.git

1.

Punto a

```
japeto@747987e7eddb:~/app/taller$ g++ -o prodcons_pthreads producer_consumer_pthread.cpp -lpthread
japeto@747987e7eddb:~/app/taller$ ./prodcons_pthreads
Producer 1 produjo 3 en pos 0
Producer 2 produjo 8 en pos 1
Consumer 1 consumió 3 en pos 0
Consumer 2 consumió 8 en pos 1
```

```
japeto@747987e7eddb:~/app/taller$ g++ -o prodcons_omp producer_consumer_omp.cpp -fopenmp
japeto@747987e7eddb:~/app/taller$ ./prodcons_omp
Producer 1 produjo 1 en pos 0
Consumer 2 consumió 1 en pos 0
Producer 2 produjo 3 en pos 0
Consumer 1 consumió 3 en pos 0
```

Punto b

```
japeto@747987e7eddb:~/app/taller$ g++ -o matvec_pthreads matrix_vector_pth.cpp -lpthread
japeto@747987e7eddb:~/app/taller$ ./matvec_pthreads
Matriz A:
0 9 5 7
3 0 5 9
7 5 1 0
4 0 4 2
9 7 6 6
6 9 0 1

Vector x:
9 4 1 6

Resultado y = A * x:
83 86 84 52 151 96
```

```
japeto@747987e7eddb:~/app/taller$ g++ -o matvec_omp matrix_vector_omp.cpp -fopenmp
/matvec_ompjapeto@747987e7eddb:~/app/taller$ ./matvec_omp
Matriz A:
2 9 6 0
6 5 3 4
4 2 2 7
5 9 6 1
3 3 8 9
4 3 1 7

Vector x:
3 2 4 6

Resultado y = A * x:
48 64 66 63 101 64
```

Punto c

```
japeto@747987e7eddb:~/app/taller$ g++ trap_pthreads.cpp -o trap_pthreads -lpthread -lm
japeto@747987e7eddb:~/app/taller$ ./trap_pthreads 4 0 1 1000000
Integral de 0.00 a 1.00 = 0.333333333333497
Tiempo (Pthreads, 4 hilos): 0.000540 segundos
```

```
japeto@747987e7eddb:~/app/taller$ g++ -fopenmp trap_omp.cpp -o trap_omp -lm
japeto@747987e7eddb:~/app/taller$ ./trap_omp 4 0 1 1000000
Integral de 0.00 a 1.00 = 0.333333333333497
Tiempo (OpenMP, 4 hilos): 0.001438 segundos
```

Punto d

```
japeto@747987e7eddb:~/app/taller$ g++ count_sort_omp.cpp -fopenmp -o count_sort_omp
japeto@747987e7eddb:~/app/taller$ ./count_sort_omp 4 10000
Tiempo (OpenMP con 4 hilos): 0.105596 segundos
```

```
japeto@747987e7eddb:~/app/taller$ g++ count_sort_pthreads.cpp -o count_sort_pthreads -lpthread
japeto@747987e7eddb:~/app/taller$ ./count_sort_pthreads 4 10000
Tiempo (Pthreads con 4 hilos): 0.361151 segundos
```

1.

```
japeto@747987e7eddb:~/app/taller$ g++ matrix_mul_pthreads.cpp -o mat_pth -pthread
mat_pthjapeto@747987e7eddb:~/app/taller$ ./mat_pth
Tiempo CPU: 0.000823404 segundos
Multiplicación completada (Pthreads)
107 317 232 259 236 336 264 252 176 228
84 203 220 182 89 183 176 201 147 185
120 285 259 247 184 308 253 230 158 212
122 191 182 175 139 275 230 144 176 154
58 139 162 192 144 183 88 156 132 122
146 237 287 314 202 308 233 218 238 232
95 152 140 165 130 184 172 120 174 150
101 308 310 312 209 307 233 287 188 242
148 231 180 204 142 283 239 133 191 182
138 310 282 299 228 342 283 270 236 255
```

```
japeto@747987e7eddb:~/app/taller$ g++ matrix_mul_openmp.cpp -o mat_omp -fopenmp
t_ompjapeto@747987e7eddb:~/app/taller$ ./mat_omp
Tiempo CPU: 0.0150516 segundos
Multiplicación completada (OpenMP)
256 198 294 206 276 365 197 292 199 302
232 211 286 172 245 317 189 252 193 217
245 202 272 180 225 311 189 226 204 231
141 140 174 121 178 229 135 201 120 184
150 114 175 106 109 169 86 130 157 123
141 138 165 147 164 219 142 195 115 190
252 157 284 218 259 362 213 332 175 304
277 225 312 220 221 350 228 272 230 278
232 182 234 183 221 346 204 243 190 282
209 152 247 159 243 292 163 253 155 206
```

2.

```
japeto@747987e7eddb:~/app/taller$ g++ ejercicio2pthread.cpp -o suma_pthreads -pthread
japeto@747987e7eddb:~/app/taller$ ./suma_pthreads
Hilo creado con ID: 133816710620928 trabajando desde 0 hasta 6250000
Hilo creado con ID: 133816702228224 trabajando desde 6250000 hasta 12500000
Hilo creado con ID: 133816693835520 trabajando desde 12500000 hasta 18750000
Hilo creado con ID: 133816618317568 trabajando desde 18750000 hasta 25000000
Hilo creado con ID: 133816609924864 trabajando desde 25000000 hasta 31250000
Hilo creado con ID: 133816601532160 trabajando desde 31250000 hasta 37500000
Hilo creado con ID: 133816593139456 trabajando desde 37500000 hasta 43750000
Hilo creado con ID: 133816584746752 trabajando desde 43750000 hasta 50000000
Hilo creado con ID: 133816576354048 trabajando desde 50000000 hasta 56250000
Hilo creado con ID: 133816567961344 trabajando desde 56250000 hasta 62500000
Hilo creado con ID: 133816559568640 trabajando desde 62500000 hasta 68750000
Hilo creado con ID: 133816551175936 trabajando desde 68750000 hasta 75000000
Hilo creado con ID: 133816542783232 trabajando desde 75000000 hasta 81250000
Hilo creado con ID: 133816534390528 trabajando desde 81250000 hasta 87500000
Hilo creado con ID: 133816517605120 trabajando desde 93750000 hasta 100000000
Hilo creado con ID: 133816525997824 trabajando desde 87500000 hasta 93750000
Suma total (Pthreads) = 450017114
Tiempo CPU: 0.0319636 segundos
```

```
japeto@747987e7eddb:~/app/taller$ g++ ejercicio2omp.cpp -o ejerc2 -fopenmp
japeto@747987e7eddb:~/app/taller$ ./ejerc2
Hilo 14 de 16 hilos creados
Hilo 9 de 16 hilos creados
Hilo 12 de 16 hilos creados
Hilo 4 de 16 hilos creados
Hilo 3 de 16 hilos creados
Hilo 8 de 16 hilos creados
Hilo 6 de 16 hilos creados
Hilo 15 de 16 hilos creados
Hilo 0 de 16 hilos creados
Hilo 2 de 16 hilos creados
Hilo 13 de 16 hilos creados
Hilo 10 de 16 hilos creados
Hilo 5 de 16 hilos creados
Hilo 11 de 16 hilos creados
Hilo 1 de 16 hilos creados
Hilo 7 de 16 hilos creados
Suma total (OpenMP) = 450052108
Tiempo CPU: 0.0417253 segundos
```

Comparación de tiempos de ejecución

Punto 1.c – Regla del trapecio

Implementación	Hilos	Tiempo CPU (s)	Resultado
OpenMP	4	0.001438	0.3333333333
Pthreads	4	0.000540	0.3333333333

Conclusión: Pthreads resultó más rápido que OpenMP en este caso, aunque ambos dieron el mismo resultado numérico.

Punto 1.d – Count Sort

Implementación	Hilos	Tamaño n	Tiempo CPU (s)
OpenMP	4	10000	0.105596
Pthreads	4	10000	0.361151

Conclusión: OpenMP fue claramente más eficiente para este algoritmo, mostrando mejor paralelización que Pthreads.

Punto 2 – Multiplicación de matrices

Implementación	Tiempo CPU (s)
Pthreads	0.000823
OpenMP	0.015051

Conclusión: Pthreads fue mucho más rápido en esta prueba de matrices, posiblemente por una división más eficiente de las filas entre hilos.

Punto 3 – Suma de un arreglo grande

Implementación	Tiempo CPU (s)
OpenMP	0.041725
Pthreads	0.031963

Conclusión: Pthreads también fue más eficiente en la suma de un arreglo, aunque la diferencia con OpenMP fue menor.