Samuel Francisco Moncayo Moncayo - Informe

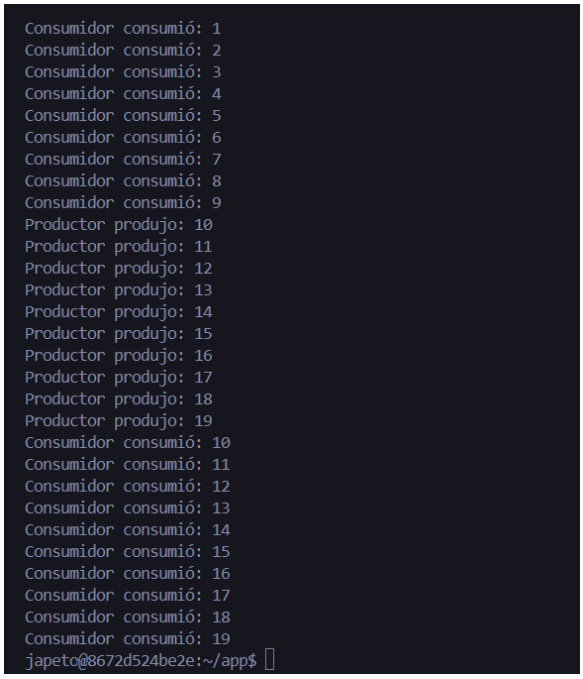
Ejercicio 1: Lectura (Implementación de fragmentos)

1. Productor–Consumidor

Se implementó un buffer circular compartido.
- El productor inserta datos en el buffer.
- El consumidor retira los datos.
- Se usaron pthread_mutex_t y variables de condición (pthread_cond_t) para evitar condiciones de carrera.

Beneficio: permite entender sincronización de hilos en memoria compartida.

Captura:



2. Multiplicación Matriz–Vector

Cada hilo procesa un conjunto de filas de la matriz A, multiplicándolas por el vector x.
- M filas se dividen en bloques iguales.
- Cada hilo actualiza las posiciones correspondientes del vector y.

Beneficio: divide el trabajo de manera balanceada y sencilla.

Captura:

```
japeto@8672d524be2e:~/app$ ./punto1mv
Resultado y = A * x :
6.000000
10.000000
14.000000
18.000000
japeto@8672d524be2e:~/app$
```

3. Regla Trapezoidal

Se aproximó la integral definida en [a,b] con n trapecios.
- Cada hilo calcula su intervalo local.
- Se usó mutex para acumular la suma parcial en la variable integral.

Beneficio: ejemplo de reducción con exclusión mutua en memoria compartida.

Captura:

```
japeto@8672d524be2e:~/app$ g++ -o punto1t trapezoidal.cpp -lpthread
japeto@8672d524be2e:~/app$ ./punto1t
Integral aproximada en [0.000000, 3.141593] = 1.999998
japeto@8672d524be2e:~/app$
```

4. Count Sort Paralelo

Se implementó una versión paralela del algoritmo count sort:
- Cada hilo calcula la posición de los elementos de un subarreglo en el resultado.
- Los resultados se almacenan directamente en el arreglo ordenado B.

Beneficio: muestra cómo algoritmos simples pueden paralelizarse sin conflictos de escritura.

Captura:

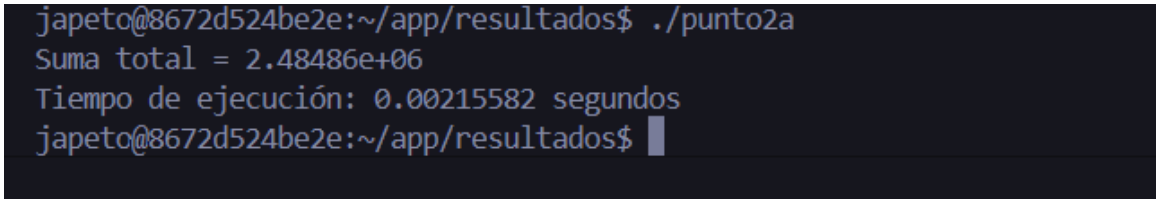
```
japeto@8672d524be2e:~/app$ g++ -o punto1cs count-sort.cpp -lpthread
japeto@8672d524be2e:~/app$ ./punto1cs
Arreglo ordenado:
0 1 2 3 4 5 6 7 8 9
japeto@8672d524be2e:~/app$
```

Ejercicio 2: Suma de un Arreglo Grande

Parte a: Pthreads con exclusión mutua

- Cada hilo calcula suma parcial.
- Se sincroniza con pthread_mutex_t.

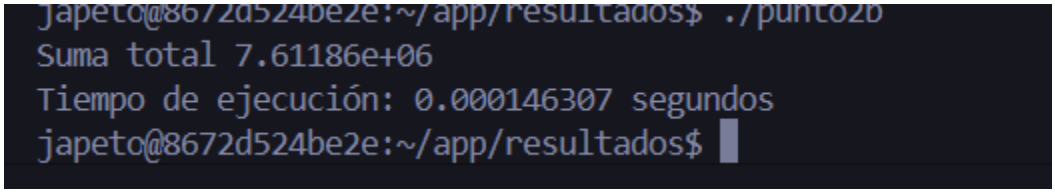
Captura:



Parte b: OpenMP con reducción

- Se paraleliza el bucle con #pragma omp parallel for reduction(+:sum).

Captura:



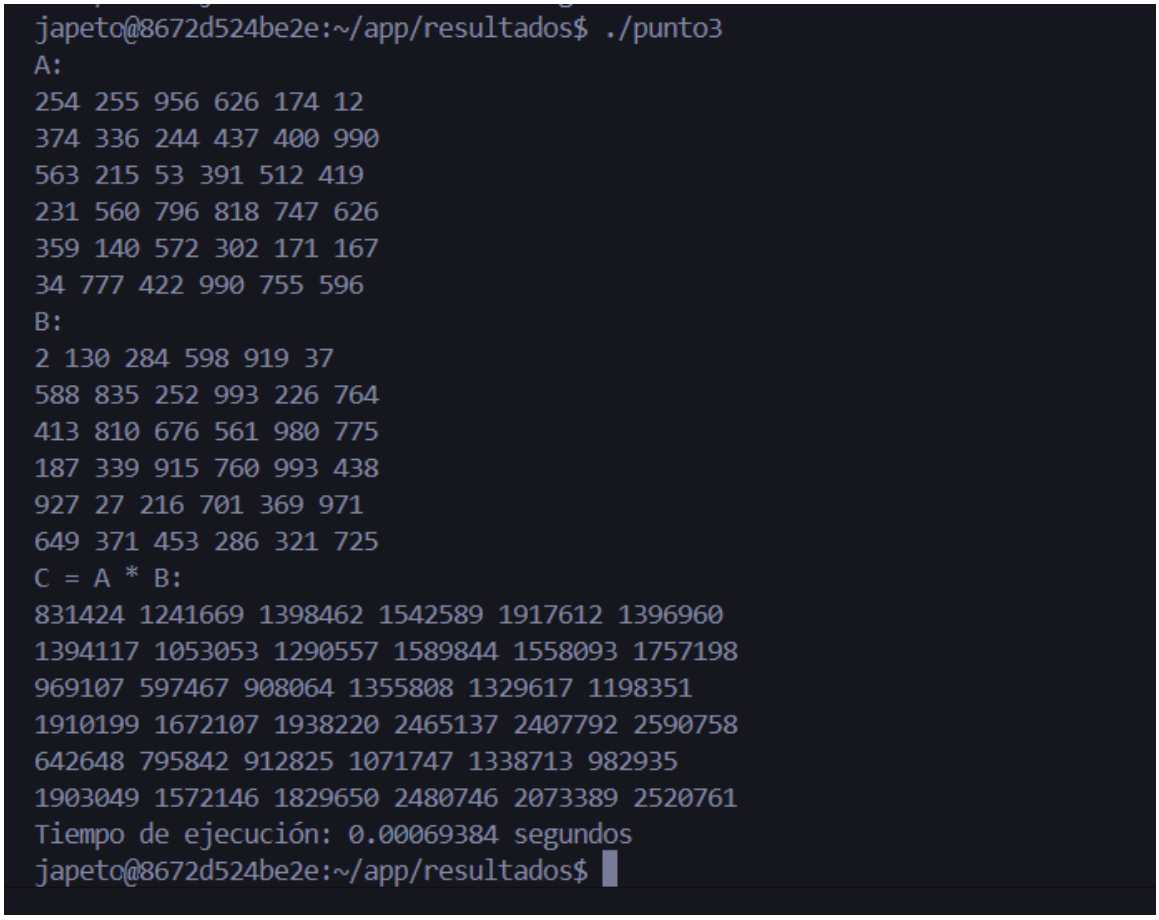
Resultados: En los archivos .out se midió el tiempo de ejecución. La versión OpenMP mostró menor tiempo frente a la implementación con pthread_mutex_t, evidenciando la optimización que ofrece la directiva reduction.

Ejercicio 3: Multiplicación de Matrices

Se implementó la multiplicación C = A * B donde cada hilo procesa un bloque de filas de A.

- Se generaron matrices aleatorias de prueba.
- Cada hilo escribe solo en su zona de C, evitando conflictos.

Captura:



Resultados: Se observaron mejoras en los tiempos paralelos frente a la versión secuencial. El tiempo exacto se encuentra en resultados/punto3.

Análisis general:

- La paralelización con OpenMP resulta más eficiente y sencilla, sin embargo cabe resaltar que la transformación del código entre más compleja sea, menos practico puede ser su uso.
- Pthreads otorga control detallado, pero con más complejidad en sincronización.
- La ganancia de rendimiento depende de la cantidad de hilos y el tamaño de los datos.