

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:

```
Consumidor: consumió: 1
Consumidor: consumió: 2
Consumidor: consumió: 3
Consumidor: consumió: 4
Consumidor: consumió: 5
Consumidor: consumió: 6
Consumidor: consumió: 7
Consumidor: consumió: 8
Consumidor: consumió: 9
Productor: produjo: 10
Productor: produjo: 11
Productor: produjo: 12
Productor: produjo: 13
Productor: produjo: 14
Productor: produjo: 15
Productor: produjo: 16
Productor: produjo: 17
Productor: produjo: 18
Productor: produjo: 19
Consumidor: consumió: 10
Consumidor: consumió: 11
Consumidor: consumió: 12
Consumidor: consumió: 13
Consumidor: consumió: 14
Consumidor: consumió: 15
Consumidor: consumió: 16
Consumidor: consumió: 17
Consumidor: consumió: 18
Consumidor: consumió: 19
japeto@8672d524be2e:~/app$
```

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:

```
japeto@8672d524be2e:~/app/resultados$ ./punto2a
Suma total = 2.48486e+06
Tiempo de ejecución: 0.00215582 segundos
japeto@8672d524be2e:~/app/resultados$
```

Parte b: OpenMP con reducción

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

Captura:

```
japeto@8672d524be2e:~/app/resultados$ ./punto2b
Suma total 7.61186e+06
Tiempo de ejecución: 0.000146307 segundos
japeto@8672d524be2e:~/app/resultados$
```

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:

```
japeto@8672d524be2e:~/app/resultados$ ./punto3
```

```
A:
```

```
254 255 956 626 174 12  
374 336 244 437 400 990  
563 215 53 391 512 419  
231 560 796 818 747 626  
359 140 572 302 171 167  
34 777 422 990 755 596
```

```
B:
```

```
2 130 284 598 919 37  
588 835 252 993 226 764  
413 810 676 561 980 775  
187 339 915 760 993 438  
927 27 216 701 369 971  
649 371 453 286 321 725
```

```
C = A * B:
```

```
831424 1241669 1398462 1542589 1917612 1396960  
1394117 1053053 1290557 1589844 1558093 1757198  
969107 597467 908064 1355808 1329617 1198351  
1910199 1672107 1938220 2465137 2407792 2590758  
642648 795842 912825 1071747 1338713 982935  
1903049 1572146 1829650 2480746 2073389 2520761
```

```
Tiempo de ejecución: 0.00069384 segundos
```

```
japeto@8672d524be2e:~/app/resultados$
```

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 práctico 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.