Trabajo Final SDyP
Simulación N-Cuerpos

Romero Mateo (03261/0)
Grupo 26

1. Solución utilizando Pthreads

1.1. Explicación

Tomando como base la implementación secuencial del problema de los N-cuerpos, implementé el algoritmo descrito en la teoría para la resolución del problema utilizando Pthreads.

La solución consiste en dividir la simulación en dos etapas: el cálculo de fuerzas y el movimiento de los cuerpos. La segunda de estas etapas es vergonzosamente paralela, ya que cada hilo puede mover un cuerpo sin interferencias. En cambio, la primera requiere mayor cuidado, dado que varios hilos pueden estar modificando las fuerzas de un mismo cuerpo.

En la solución implementada, esto se resuelve creando una matriz local en la que cada thread posee una fila para guardar las fuerzas sobre los cuerpos. Esto asegura que cada uno solamente escriba en su sección del arreglo, garantizando así que no ocurran condiciones de carrera.

Además, estas dos etapas dependen una de la otra, por lo que es necesario sincronizar los hilos mediante una barrera para garantizar que todos avancen a la siguiente etapa al mismo tiempo.

1.2. Analisis de escalabilidad

1.2.1. Tiempos de ejecucion

	Tamaño del problema (N)			
Unidades de Procesamiento	**512**	**1024**	**2048**	**4096**
Secuencial	12.088	48.222	193.067	771.825
4	3.407	12.994	50.807	227.700
8	2.021	7.068	29.856	116.360

Cuadro 1: Tiempos de ejecución (en segundos)

1.2.2. Speedup

	Tamaño del problema (N)			
Unidades de Procesamiento	**512**	**1024**	**2048**	**4096**
Secuencial	1.00	1.00	1.00	1.00
4	3.55	3.71	3.80	3.39
8	5.98	6.82	6.47	6.63

Cuadro 2: Speedup respecto a ejecución secuencial

1.2.3. Eficiencia

	Tamaño del problema (N)			
Unidades de Procesamiento	**512**	**1024**	**2048**	**4096**
Secuencial	100.0	100.0	100.0	100.0
4	88.67	92.82	94.89	84.71
8	74.76	85.30	80.91	82.91

Cuadro 3: Eficiencia (%)

1.2.4. Estudio de las tablas

La eficiencia obtenida con Pthreads se mantiene alta en la mayoría de los casos, especialmente cuando la carga de trabajo es significativa. Esto sugiere que el costo de sincronización y otros factores no paralelizables se amortiza adecuadamente conforme se incrementa el esfuerzo computacional.

No obstante, al aumentar la cantidad de hilos sin modificar el problema, la eficiencia tiende a decaer. Esta limitación es coherente con la **Ley de Amdahl**, que señala cómo incluso una pequeña parte secuencial puede frenar las mejoras cuando se escalan los recursos.

Por otro lado, cuando el paralelismo crece acompañado de una mayor carga de trabajo, el rendimiento mejora o se mantiene estable. Este comportamiento se ajusta a la **Ley de Gustafson**, que propone que el impacto del código secuencial disminuye en problemas de mayor escala, habilitando mejores niveles de aprovechamiento.

Si bien los tiempos fueron medidos solo una vez y podrían presentar cierto margen de error, las tendencias generales observadas son claras. En este contexto, la solución con Pthreads no evidencia escalabilidad fuerte, pero sí muestra una **escalabilidad débil** al crecer proporcionalmente el trabajo y los recursos.

2. Solución Híbrida con OpenMPI y Pthreads

2.1. Explicación

Se empleó una solución sistólica, ya que reduce la cantidad de mensajes enviados en cada paso, lo cual es sumamente favorable dado que la comunicación resulta mucho más costosa en comparación con el cómputo.

A nivel de procesos, se decidió distribuir la carga de manera estática y proporcional, evitando los métodos de stripes o stripes inverso, que implican un alto costo de comunicación. Sin embargo, no es posible distribuir los cuerpos equitativamente entre los procesos, ya que esto generaría un desbalance de carga significativo.

Esto se debe a que, a medida que avanzamos sobre los cuerpos, cada uno debe realizar menos cálculos. Ilustrativamente:

$$\begin{array}{rcll} C_0 & \to & C_1, C_2, \ldots, C_{N-1} & \text{(N-1)} \\ C_1 & \to & C_2, C_3, \ldots, C_{N-1} & \text{(N-2)} \\ C_2 & \to & C_3, C_4, \ldots, C_{N-1} & \text{(N-3)} \\ \vdots & & \vdots & \\ C_{N-3} & \to & C_{N-2}, C_{N-1} & (2) \\ C_{N-2} & \to & C_{N-1} & (1) \\ C_{N-1} & \to & \text{Ninguno} & (0) \end{array}$$

A partir de esto, podemos determinar una sumatoria que expresa la cantidad de fuerzas que debe calcular cada proceso según su intervalo de cuerpos asignado:

$$\sum_{i=\text{inicio}}^{\text{fin}} (N - i - 1)$$

donde *inicio* y *fin* indican el rango de cuerpos que le corresponde a cada proceso.

Como deseamos que la distribución de carga sea equitativa, planteamos la siguiente igualdad, la cual determina la cantidad de interacciones totales para todos los cuerpos:

$$\sum_{i=0}^{N-1}(N - i - 1) = \frac{N(N-1)}{2}$$

Buscamos un valor $k \in (0, 1)$ tal que el trabajo desde $i = 0$ hasta $i = kN - 1$ represente la mitad del total:

$$\sum_{i=0}^{kN-1} (N - i - 1) = \frac{1}{2} \cdot \frac{N(N-1)}{2}$$

La suma del lado izquierdo puede reescribirse como:

$$\sum_{i=0}^{kN-1} (N - i - 1) = \sum_{j=N-kN}^{N-1} j = \frac{kN}{2} \cdot [(N-1) + (N - kN)]$$

$$= \frac{kN}{2} \cdot (2N - 1 - kN)$$

Igualando ambas expresiones y operando:

$$\frac{kN}{2} \cdot (2N - 1 - kN) = \frac{N(N-1)}{4}$$

$$2kN(2N - 1 - kN) = N(N-1)$$

$$2k(2N - 1 - kN) = N - 1$$

Para N grande, aproximamos:

$$2k(2N - kN) \approx N \Rightarrow 2k(2 - k) \approx 1$$

Resolviendo la ecuación:

$$2k(2-k) = 1 \Rightarrow 4k - 2k^2 = 1 \Rightarrow 2k^2 - 4k + 1 = 0$$

Aplicamos la fórmula cuadrática:

$$k = \frac{4 \pm \sqrt{(-4)^2 - 4 \cdot 2 \cdot 1}}{2 \cdot 2} = \frac{4 \pm \sqrt{16-8}}{4} = \frac{4 \pm \sqrt{8}}{4} = \frac{4 \pm 2\sqrt{2}}{4} = \frac{2 \pm \sqrt{2}}{2}$$

Como $k \in (0, 1)$, tomamos la solución válida:

$$k = \frac{2 - \sqrt{2}}{2} \approx 0{,}2929$$

Esto indica que la primera partición debe cubrir aproximadamente el 29,29 % de los cuerpos para lograr una distribución equitativa de la carga computacional.

Sin embargo, surge un problema: el valor $0{,}2929 \cdot N$ no siempre es un número entero. Al convertir este valor a una variable de tipo `int`, se produce un redondeo que puede provocar errores en el procesamiento —ya sea procesando un cuerpo de más o de menos—, lo cual es incorrecto.

Para evitar este desbalance, se definió el tamaño del bloque del primer proceso como:

$$\texttt{block\_size0} = \lfloor 0{,}2929 \cdot N \rfloor$$

y el del segundo proceso como:

$$\texttt{block\_size1} = N - \texttt{block\_size0}$$

De esta forma, se garantiza que ambos bloques sumen exactamente N cuerpos, sin solapamientos ni omisiones.

Ya habiendo distribuido la carga, procedemos con el algoritmo propiamente dicho. Se sigue una estrategia similar a la versión sistólica vista en la teoría. En cada paso, el proceso 1 comienza enviando sus cuerpos al proceso 0.

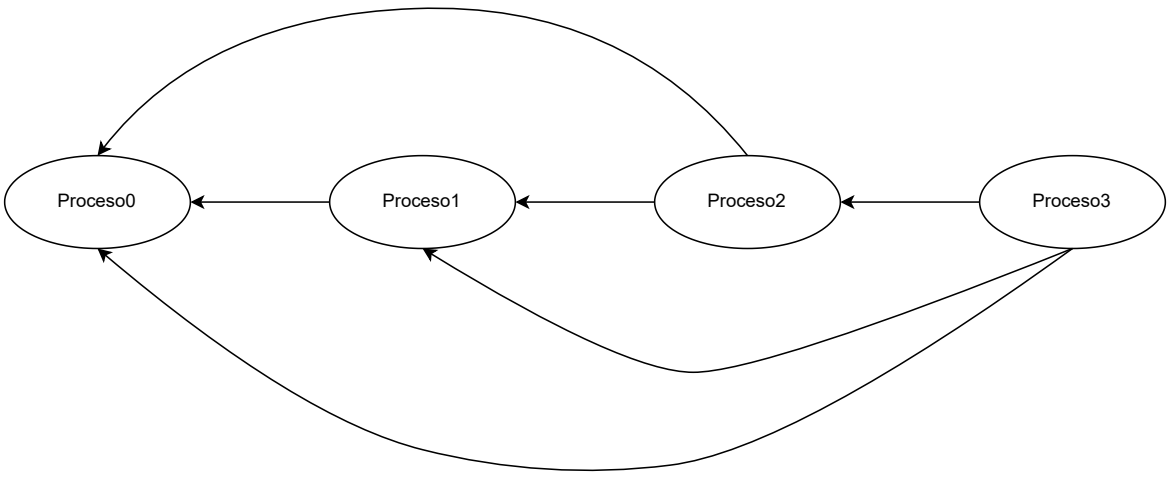

Figura 1: Gráfico explicativo de cómo se envían los cuerpos en el modelo sistólico general, extraído de la clase teórica.

Una vez completada esta comunicación, cada proceso calcula las fuerzas sobre su bloque de cuerpos. Específicamente, cada uno evalúa las fuerzas que afectan a los cuerpos de su bloque debido a todos los cuerpos comprendidos entre su propio inicio y el cuerpo $N-1$. Esta etapa se paraleliza internamente utilizando pthreads, aplicando una estrategia de distribución tipo Stripes, ya que el enfoque de Stripes inverso continúa siendo computacionalmente costoso.

Para evitar condiciones de carrera, cada hilo dentro de un proceso cuenta con una copia local del arreglo de fuerzas, estructurada como una matriz en la que cada fila corresponde a un hilo e incluye la información de todos los cuerpos.

Adicionalmente, según la tercera ley de Newton, cada proceso también calcula la fuerza que sus cuerpos ejercen sobre aquellos que se encuentran más adelante. Esto es posible porque los procesos anteriores ya han computado el efecto de sus propios cuerpos sobre los anteriores, evitando así redundancias.

Una vez finalizado el cálculo de fuerzas, cada proceso suma las contribuciones parciales de cada hilo en un arreglo de fuerzas totales. En esta etapa también se emplea una distribución Stripes.

Finalmente, el proceso 0 envía al proceso 1 las fuerzas externas correspondientes a su bloque. El proceso 1 las recibe y las suma a su propio arreglo de fuerzas totales.

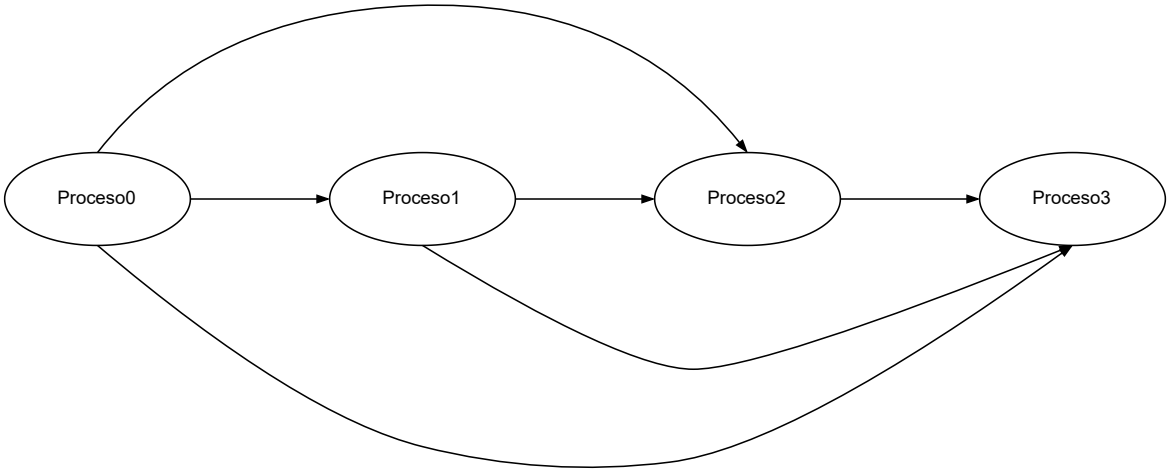

Figura 2: Gráfico explicativo de cómo se envían las fuerzas en el modelo sistólico general, extraído de la clase teórica.

Llegada esta etapa, cada proceso tiene las fuerzas totales que afectan a su bloque, por lo que lo único que resta es mover los cuerpos. Esta tarea también se distribuye siguiendo una estrategia tipo *Stripes*, y se utiliza el método *Leapfrog* para actualizar posiciones y velocidades.

El método *Leapfrog* consiste en un esquema de integración numérica en el que las posiciones y velocidades se actualizan de forma intercalada: primero se calcula la nueva velocidad a mitad del intervalo de tiempo, luego se usa esa velocidad para actualizar la posición completa. Este método es especialmente útil en simulaciones físicas por su buena conservación de la energía y estabilidad a largo plazo.

Estos pasos conforman el bucle principal de la simulación. Una vez finalizado dicho bucle,

el proceso 0 envía al proceso 1 sus cuerpos, de modo que este último posea la información actualizada de todos ellos. Luego, se realiza un broadcast para asegurar que ambos procesos tengan la posición correcta de todos los cuerpos.

2.2. Analisis de escalabilidad

2.2.1. Tiempos de ejecucion

	Tamaño del problema (N)			
Unidades de procesamiento	**512**	**1024**	**2048**	**4096**
Secuencial	12.088	48.222	193.067	771.825
4	3.985	13.768	52.086	203.370
8	2.719	8.123	28.299	119.387
16	2.161	5.297	17.987	63.602

Cuadro 4: Tiempos de ejecución (en segundos)

2.2.2. Speedup

	Tamaño del problema (N)			
Unidades de procesamiento	**512**	**1024**	**2048**	**4096**
Secuencial	1.00	1.00	1.00	1.00
4	3.03	3.50	3.71	3.80
8	4.44	5.94	6.82	6.47
16	5.59	9.10	10.73	12.13

Cuadro 5: Speedup respecto a ejecución secuencial

2.2.3. Eficiencia

	Tamaño del problema (N)			
Unidades de procesamiento	**512**	**1024**	**2048**	**4096**
Secuencial	100.0	100.0	100.0	100.0
4	75.63	87.44	92.78	94.88
8	55.50	74.29	85.29	80.88
16	34.94	56.88	67.04	75.79

Cuadro 6: Eficiencia (%)

2.2.4. Estudio de las tablas

En la solución híbrida se observa una mejora progresiva en la eficiencia a medida que aumenta la carga computacional. La combinación de procesos y hilos logra buenos resultados cuando el trabajo es lo suficientemente amplio como para justificar la sobrecarga de comunicación y coordinación.

En cambio, al escalar solo los recursos sin ampliar el problema, el rendimiento cae, como anticipa la **Ley de Amdahl**. La comunicación entre procesos MPI y la sincronización entre hilos impactan más en estos casos.

Cuando el problema crece junto con los recursos disponibles, la eficiencia mejora notablemente. Esto concuerda con la **Ley de Gustafson**, ya que el peso relativo de las secciones secuenciales disminuye, permitiendo que el paralelismo rinda más.

En definitiva, la solución híbrida tampoco logra escalabilidad fuerte. Sin embargo, el comportamiento general refleja una **escalabilidad débil**, con mejoras sostenidas cuando se incrementa proporcionalmente la carga y los medios de cómputo.

3. Aclaración

Durante las pruebas de las distintas implementaciones, observé que para casos con 512 y 1024 cuerpos, las posiciones finales no presentan diferencias significativas entre la versión secuencial y las versiones paralelizadas. Sin embargo, al aumentar la cantidad de cuerpos a 2048 y 4096, el error porcentual entre implementaciones crece de forma considerable.

Considero que la lógica del programa es correcta, por lo que no debería haber un problema funcional. Esto me lleva a pensar que la causa podría estar relacionada con errores de representación en punto flotante: al calcular un número muy elevado de interacciones entre cuerpos, es posible que se acumulen pequeños errores de precisión que terminan propagándose y afectando drásticamente el resultado final.