**Carlos Manuel Vélez, Andres Martínez y Miguel Angel Tovar**

El objetivo de este reporte es explicar el funcionamiento de un N-Body Simulator desarrollado en Python utilizando la biblioteca PyCUDA para realizar diversas tareas en paralelo. Este simulador tiene como objetivo modelar el movimiento de un conjunto de partículas en un espacio tridimensional, teniendo en cuenta las fuerzas gravitacionales entre ellas. El uso de PyCUDA permite aprovechar la capacidad de cómputo de la GPU para acelerar significativamente los cálculos necesarios en este tipo de simulaciones.

***Descripción general del Código:***

El código se organiza en varias secciones, cada una de las cuales desempeña un papel fundamental en la simulación:

**Importaciones de Librerías:**

Se importan las librerías necesarias para la ejecución del código, incluyendo numpy para operaciones numéricas, pycuda para la programación en CUDA, y matplotlib para la visualización de datos.

**Definición de la Clase Particle:**

Se define una clase llamada Particle, que representa las partículas de la simulación. Cada partícula tiene una posición, una velocidad y una masa. Al inicializar la clase, se generan aleatoriamente las posiciones, velocidades y masas de un determinado número de partículas.

**Kernel de CUDA:**

El código incluye un kernel de CUDA que define dos funciones: calculateForces e integrate.

**Calcular Fuerzas (calculateForces):**

Para cada partícula, calcula la fuerza resultante debido a todas las otras partículas, evitando divisiones por cero y asegurando que no se calcula la fuerza de una partícula sobre sí misma.

Usa la ley de la gravitación universal para calcular la magnitud de las fuerzas.

**Integrar Movimiento (integrate):**

Para cada partícula, calcula la aceleración basada en la fuerza resultante y la masa.

Actualiza la velocidad y posición de la partícula usando la aceleración y un factor de amortiguamiento.

Asegura que las partículas permanezcan dentro de un volumen cúbico definido, ajustando posiciones y velocidades para simular rebotes en las paredes del cubo.

**Función run\_simulation:**

Esta función coordina la ejecución de la simulación. Comienza compilando el código del kernel de CUDA utilizando la clase SourceModule de PyCUDA. Luego asigna memoria en la GPU para reservar espacio y almacenar las posiciones, velocidades, fuerzas y masas de las partículas. Durante cada iteración de la simulación, se ejecutan las funciones del kernel para calcular y actualizar el movimiento de las partículas. Se utiliza matplotlib para visualizar en tiempo real el estado de la simulación y trazar las trayectorias de las partículas al finalizar la misma.

***Detalle del funcionamiento:***

**Inicialización de las Partículas:**

* **self.position** inicializa las posiciones de las partículas con una distribución normal (media mean, desviación estándar std\_dev).
* **self.velocity** inicializa las velocidades de las partículas con una distribución normal centrada en 0.
* **self.mass** asigna masas a las partículas con valores aleatorios entre 10 y 110.

**Compilación del Kernel de CUDA:**

* **Función calculateForces**

Calcula las fuerzas entre las partículas debido a la interacción gravitacional.

* + - **Variables:**
      * **i**: Índice de la partícula actual, calculado a partir del índice del bloque (blockIdx.x), el tamaño del bloque (blockDim.x) y el índice del hilo (threadIdx.x).
      * **myPosition**: Posición de la partícula actual (positions[i]).
      * **force**: Fuerza acumulada sobre la partícula actual, inicializada a cero.
    - **Ciclo for:**

Calcula la fuerza que todas las otras partículas ejercen sobre la partícula i.

* + - * **j**: Índice de las otras partículas en el sistema.
      * **otherPosition**: Posición de la partícula j.
      * **diff**: Vector de diferencia entre las posiciones de las partículas i y j.
      * **distSqr**: Distancia al cuadrado entre las partículas i y j, con un término pequeño añadido (1e-9) para evitar divisiones por cero.
      * **invDist**: Inverso de la distancia entre las partículas i y j.
      * **invDist3**: Inverso de la distancia al cubo, utilizado para calcular la magnitud de la fuerza gravitacional.
      * **If** **condition**: Se asegura de que la distancia no sea cero para evitar divisiones por cero y que i no sea igual a j.
    - **Cálculo de la fuerza:**

Las componentes de la fuerza se calculan y se suman a force para la partícula i.

* + - **Resultado:**

**forces[i]** se establece con la fuerza total calculada.

* **Función integrate**

Actualizar las posiciones y velocidades de las partículas utilizando las fuerzas calculadas.

* + - **Variables:**
      * **i:** Índice de la partícula actual.
      * **velocity**: Velocidad actual de la partícula (velocities[i]).
      * **position**: Posición actual de la partícula (positions[i]).
      * **force**: Fuerza actuando sobre la partícula (forces[i]).
      * **mass**: Masa de la partícula (masses[i]).
      * **acceleration**: Aceleración de la partícula, calculada como fuerza dividida por masa.
    - **Cálculo de la aceleración:**
      * **acceleration**: Se calcula dividiendo la fuerza por la masa para obtener la aceleración de la partícula.
    - **Actualización de la velocidad:**
      * **Amortiguamiento:** Se aplica un factor de amortiguamiento (damping = 0.99) para evitar que las velocidades se vuelvan excesivamente grandes.
      * **velocity.x, velocity.y, velocity.z:** Se actualizan sumando la aceleración multiplicada por el paso de tiempo dt y luego aplicando el amortiguamiento.
    - **Actualización de la posición:**
      * **position.x, position.y, position.z:** Se actualizan sumando la velocidad multiplicada por el paso de tiempo dt.
    - **Condiciones de frontera:**
      * Se verifica si la partícula ha excedido los límites del espacio cúbico definido entre -1e6 \* 0.5 y 1e6 \* 0.5 para cada coordenada (x, y, z).
      * Si una partícula excede estos límites, su posición se ajusta al límite y su velocidad se invierte para simular un rebote.
    - **Resultado:**

**velocities[i]** y **positions[i]** se actualizan con los nuevos valores calculados.

**Ejecución de la Simulación:**

* **Inicialización y Copia de Memoria**
  + Se compilan los kernels CUDA y se llama a las funciones calculateForces y integrate.
  + Se asigna memoria en la GPU para posiciones, velocidades, fuerzas y masas de las partículas.
  + Se copian los datos iniciales desde el host (CPU) a la memoria del dispositivo (GPU).
* **Configuración de CUDA**
  + Se define el tamaño del bloque y el grid para la ejecución de los kernels en CUDA.

**Visualización de la Simulación:**

Se utiliza matplotlib para visualizar en tiempo real el estado de la simulación, mostrando las posiciones de las partículas en un gráfico tridimensional. Además, se trazan las trayectorias de las partículas al finalizar la simulación.

**Conclusiones**

El simulador desarrollado utilizando PyCUDA proporciona una herramienta eficaz para modelar el movimiento de partículas en un espacio tridimensional. La utilización de la GPU para realizar cálculos en paralelo permite acelerar significativamente la simulación, lo que facilita el estudio de sistemas de partículas complejos y de gran tamaño. Además, la capacidad de visualización en tiempo real proporcionada por matplotlib permite una comprensión intuitiva del comportamiento de las partículas durante la simulación.

En resumen, el simulador de partículas utilizando PyCUDA es una herramienta poderosa y versátil que puede ser utilizada en una amplia gama de aplicaciones científicas y de ingeniería, desde la física de partículas hasta la simulación de sistemas biológicos y la dinámica de fluidos gracias a la posibilidad de ejecutar una gran cantidad de operaciones simultáneamente, obteniendo resultados en una menor cantidad de tiempo. Su flexibilidad y rendimiento lo convierten en una herramienta valiosa para investigadores y profesionales en diversos campos.