Universidad Técnica Federico Santa María
Campus Santiago San Joaquín
Departamento de Informática
2° semestre 2020

Taller 2: Lobos y Conejos

Taller de Modelos y Métodos Cuantitativos
Profesor Álvaro Salinas
Estudiantes Ethiel Carmona 201773533-3,
Joaquín Castillo 201773520-1,
María Paz Morales 201773505-8,
Axel Reyes 201773502-3

3 de diciembre de 2020

1. Especificaciones

   En el presente taller se utilizó la siguiente tarjeta gráfica:

   - **Nombre:** NVIDIA GeForce
   - **Modelo:** GTX 1050 TI
   - **Compute Capability:** Versión 6.1
   - **Arquitectura:** Pascal
   - **Cuda Cores:** 768
   - **Frame Buffer:** 4 GB GDDR5
   - **Velocidad de memoria:** 7 Gbps

2. Desarrollo

   [**Pregunta**] ¿Cuál de las dos implementaciones presenta un mejor desempeño? Comente al respecto.

   La implementación hecha por la GPU es la que presenta mejor desempeño. La implementación en CPU se hizo de manera secuencial, donde se itera en cada arreglo, posición por posición. Cuando se realizó la implementación en GPU, se definieron las hebras necesarias para que cada una acceda a una posición de los arreglos en Cuda, lo que permitió paralelizar todo el proceso y reducir los tiempos, obteniendo así un mejor desempeño. En CPU esto no se puede hacer porque no hay suficientes núcleos, por lo que se realizó de manera secuencial utilizando un sólo núcleo.

   [**Pregunta**] ¿Enfrentó problemas relacionados a posibles condiciones de carrera? De ser así, descríbalos e indique cómo los solucionó.

   Sí, se presentan problemas de condiciones de carrera en ambas implementaciones (GPU y CPU), principalmente en la función de movimiento y la de reproducción. En el movimiento, si un conejo o un lobo se mueven a una casilla que aún no ha implementado su movimiento, estaría borrando a este lobo o conejo. Entonces, si el desplazamiento es en sentido de lectura del

tablero, estaría realizando dos veces el movimiento. Este problema también se presenta en la reproducción, en donde se reemplazaría un par de conejos o lobos que iban a ser reproducidos por sólo un conejo o sólo un lobo, perdiéndose así uno de ellos. Para resolver esto, en ambas implementaciones (GPU y CPU) se manejaron dos arreglos adicionales, uno para los conejos y otro para los lobos. De esta forma, la acción de movimiento y la de reproducción se ven reflejadas en el arreglo libre, es decir, si el arreglo 1 de conejos tiene el resultado de la ronda anterior, el resultado del movimiento estará en el arreglo 2. Y análogamente, si el resultado del movimiento está en el arreglo 2, entonces el resultado de la reproducción estará en el arreglo 1.

Dentro de la acción de movimiento, la secuencia es la siguiente: se escoge el movimiento y se suma 1 a la casilla correspondiente (inicialmente están en 0), esto permite saber cuando dos lobos o dos conejos caen en la misma casilla. Para la implementación de la CPU no se presentan problemas, ya que la ejecución es secuencial y se modifica una casilla a la vez. En cambio, en la implementación de GPU se tendrían hasta cuatro hebras accediendo a la misma casilla, lo que podría generar pérdida de información dependiendo de qué hebra acceda primero. Para resolver esto, la suma de 1 a la casilla correspondiente se hizo con una suma atómica, es decir, en la que sólo 1 hebra puede acceder a la casilla, y las demás deben esperar.

Un problema similar se produce en el algoritmo de GPU al contar la cantidad de lobos y conejos finales. En este caso son varias hebras las que intentan acceder al contador global. Para solucionar esto se implementaron dos formas diferentes, la primera es utilizar la misma estrategia de antes utilizando una suma atómica, esto es similar al contador secuencial que se utiliza en CPU por lo que no se está sacando provecho a las hebras. Es por eso que se implementó el algoritmo de reducción paralela visto en clases, así se evitó que más de una hebra accediera al mismo contador.

[**Pregunta**] ¿Cuál es la versión más eficiente entre la secuencial y los dos *kernels*? Concluya respecto a los tiempos observados.

La versión más eficiente es la de suma atómica implementada en *Kernel*. En comparación a la implementación de CPU, es más rápido porque la GPU permite ejecutar una hebra por casilla haciendo más rápida la suma a pesar de que sea una casilla por una, no como la CPU que ejecuta las operaciones en un solo núcleo. Por otro lado en comparación a la implementación de reducción paralela, esta última utiliza memoria compartida lo que produce una lectura y escritura en esta, resultando en una diferencia de aproximadamente $0,5$ [ms] en comparación a la suma atómica. Considerando además que la suma atómica implementa un caché especial donde almacena resultados de la suma para permitir paralelismo, y que este caché es igual o más rápido de acceder que la memoria compartida (ya que se ha optimizado a lo largo de los años), esto se traduce en que utilizar operaciones atómicas sea más rápido que reducción por memoria compartida.

[**Informe**] Presente sus resultados (número de conejos y lobos vivos).

Los resultados de contar el número de conejos y lobos finales luego de simular 1000 turnos (utilizando las semillas 432 para CPU y 0 para GPU respectivamente), se muestran en la tabla 1 a continuación.

Ejecución	Tarea	Cant. Lobos	Cant. Conejos
CPU	Conteo de lobos y conejos	1.112.032	0
GPU	Conteo con operaciones atómicas	1.114.422	0
GPU	Conteo con memoria compartida	1.114.422	0

Tabla 1: Resultados del conteo de conejos y lobos.

[**Informe**] Presente una tabla con los tiempos comparados tanto para la simulación de las 1000 rondas como para el conteo final de animales vivos.

Los resultados del tiempo que se tarda en simular 1000 turnos (utilizando las semillas 432 para CPU y 0 para GPU respectivamente) y sumar los lobos y conejos finales, se muestran en la tabla 2 a continuación.

Ejecución	Tarea	Tiempo [ms]
CPU	Simulación 1000 rondas	114373
CPU	Conteo de lobos y conejos	13
GPU	Simulación 1000 rondas	2910,406738
GPU	Conteo con operaciones atómicas	0,241664
GPU	Conteo con memoria compartida	0,945568

Tabla 2: Resultados de los tiempos de la simulación de 1000 rondas y conteo de conejos y lobos.

3. Conclusiones

La implementación de reducción con memoria compartida resultó ser bastante más compleja que la de conteo atómico, por lo que se esperaba que su tiempo de ejecución fuese menor que la de este último. Sin embargo, la implementación de conteo atómico fue hasta incluso casi 5 veces más eficiente que la de reducción con memoria compartida. Esto hace que sea preferible optar por implementar un conteo con operaciones atómicas debido a su sencilla implementación y su eficiente resultado.

En cuanto al desarrollo de los programas en general, queda demostrado que para estos casos es mucho mejor utilizar algoritmos de GPU que algoritmos de CPU, pero es necesario tener en cuenta las condiciones de carrera que se pueden llegar a generar e implementar un algoritmo que solucione este problema.

Por último, señalar que nos puso muy triste que todos los conejos se murieran :c ojalá la vida les dé la posibilidad de conquistar a sus depredadores.