Tarea 1 - Computación Científica y Ciencia de los Datos
Vectorización en Python: Generalización de autómatas de Conway

Prof: Pablo Román

20 de Marzo 2025

1 Objetivos

Construir programas vectorizados que permitan ejecución eficiente en Python. Esta técnica permite lograr velocidades comparables a un programa compilado en C e incluir aceleración vía GPU. La aceleración típica en relación a códigos realizados con ciclos de Python puede llegar a 3 órdenes de magnitud; esto se hace al costo de un mayor uso de memoria. Para fines de este ejercicio, se implementará el problema de la evolución generalizada del juego de la vida de Conway.

2 Contexto: El juego de la vida de Conway en su versión original

El juego de la vida de Conway (`https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life`) consiste en una matriz infinita de celdas indexadas. Cada celda puede tener un valor 0 o 1 que se interpretaba como inerte o vida. La matriz en el tiempo inicial ($t = 0$) tiene cierta configuración que es conocida. Posteriormente evoluciona en tiempos discretos según la siguiente regla:

Cada celda (i, j) dispone de 8 vecinos : $(i-1, j-1), (i-1, j), (i-1, j+1), (i, j-1), (i, j+1), (i+1, j-1), (i+1, j), (i+1, j+1)$.

- Si $V_{ij} = 0$ (caso inerte) y la suma de valores vecinos es igual a 3 entonces se cambia a $V_{ij} = 1$.
- Si $V_{ij} = 1$ (caso vida) y la suma de valores vecinos es 0 o 1 entonces se cambia a $V_{ij} = 0$.
- Si $V_{ij} = 1$ (caso vida) y la suma de valores vecinos es 2 o 3 entonces se mantiene el valor.
- Si $V_{ij} = 1$ (caso vida) y la suma de valores vecinos es igual a 4 o más entonces se cambia a $V_{ij} = 0$.

Siguiendo las reglas anteriores, evoluciona la matriz en tiempos discretos reproduciendo diversos patrones. La figura 1 muestra la evolución después de 155 aplicaciones de la regla desde una matriz con 5 celdas con valores 1.

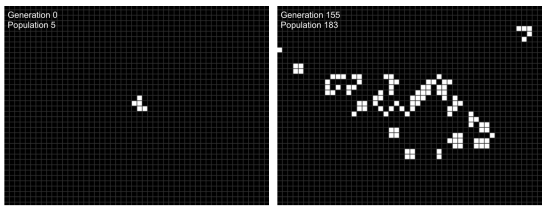

Figure 1: La figura de la izquierda muestra la matriz inicial, y la derecha muestra la iteración 155. (Fuente: Wikipedia)

Este sistema y su evolución se han estudiado desde el punto de vista de los patrones generados. Se generó toda una área de investigación al respecto para poder explicar cómo una simple regla de evolución puede lograr movimientos periódicos y estructuras que permanecen en movimiento. Este modelo de evolución fue posteriormente generalizado de varias maneras; la más importante es la de autómata celular.

3 Autómata celulares

Un autómata celular `https://en.wikipedia.org/wiki/Cellular_automaton` es un modelo de computación. Esto significa que es equivalente en ambos sentidos a cualquier programa que se ejecuta en un computador. Es decir, cualquier cómputo realizado se puede expresar como un autómata celular y viceversa. Un caso particular de autómata celular corresponde al juego de la vida de Conway. Históricamente, este se generalizó a varias dimensiones y varios estados por celda.

Un autómata celular corresponde a una grilla en d dimensiones, es decir, cada celda de la grilla se puede indexar con d índices $K = (k_0, \ldots, k_{d-1})$. Además, se define el concepto de celdas vecinas a una celda, donde B_K es el conjunto de celdas que se consideran vecinas a la celda K. Cada celda K de la grilla contiene un valor entero $V_K \in 0, \ldots, m$. Se definen también las reglas con las cuales evoluciona el sistema; estas se aplican en tiempos discretos desde una cierta configuración conocida de la grilla sobre todas las celdas en forma simultánea. Las reglas se pueden resumir en una función sobre las celdas y sus vecinas $F(K, V_K)$ y retornan el nuevo valor de la celda K.

La grilla y vecindades se pueden definir en espacios finitos como por ejemplo con $d = 2$ una matriz cuadrada de ancho n. En este caso, las vecindades en los bordes de la matriz estarían truncadas, porque no hay vecinos fuera de la matriz. Existe otra forma de definir vecinos, la grilla se podría ubicar en un toroide (Fig. 2). Es decir, los vecinos de una celda de la primera fila son aquellos que siguen en la última fila, del mismo modo con la primera y última columna.

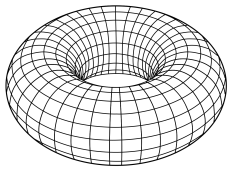

Figure 2: Grilla en toroide

4 Problema a resolver

Se debe realizar un simulador de un autómata en d dimensiones con $m + 1$ estados por celda. Este simulador debe ser eficiente; por lo tanto, debe ser construido utilizando broadcasting y cupy. Esto significa evitar en lo posible el uso de ciclos de Python. El único ciclo que se puede permitir es aquel que hace el avance del tiempo; es decir, actualiza la grilla por paso de tiempo. La actualización en un paso de tiempo debe ser sin ciclos. Las vecindades de una celda se considerarán al interior de cierto radio r. Debe considerar en ese caso bordes periódicos de la grilla; es decir, en un toroide de varias dimensiones. En cuanto a las reglas de modificación del estado de cada celda, nos centraremos en cierta regla particular.

Se considera la suma de valores S_K en la vecindad de radio r entorno a la celda K a actualizar; descontando la celda. Dicha suma tiene un valor máximo S^M igual a m por la cantidad de celdas vecinas dado r. Se generan 3 intervalos equiespaciados entre 0 y S^M. Si el número S_K se encuentra en el primer intervalo disminuye en una unidad. Si se encuentra en el segundo intervalo aumenta en una unidad, y si está en el tercero entonces disminuye en una unidad.

La evolución de este autómata generará diversos patrones de comportamiento. Entre los patrones que son interesantes están aquellos que generan ciertas periodicidades o semi-periodicidades. Ejemplos de patrones periódicos los puede encontrar en `https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life#Examples_of_patterns`. Se requiere que ud encuentre una configuración inicial que genere patrones al menos semi periódicos como los ejemplos anteriores para varias valores de d,m,r, y tamaño de la grilla.

5 Evaluación de la tarea

1. (1 pto) Documente en forma clara y prolija en jupyter notebook la resolución de sus tarea y experimentos realizados. Describa en detalle su enfoque utilizado e interprete sus resultados. Limite su documentación a un máximo de ocho páginas tamaño carta, sin contar los gráficos (según se mide en la exportación a PDF). Incluya explicaciones claras y utilice gráficos ilustrativos que apoyen la comunicación de su trabajo.

2. (3 pts) Implemente el simulador de manera eficiente utilizando cupy y sin usar ciclos según las especificaciones anteriores. Este simulador debe ser parametrizado con d,m,r, tamaño de la grilla y grilla de partida. Compare el tiempo que demora su código accelerado con una implementación con ciclos. Debe entregar 10 ejemplos como test de su implementación.

3. (2 ptos, 0 puntos si en la parte anterior su implementación no es eficiente) Encuentre patrones semi periódicos con su implementación eficiente en dimensión $d > 2$. Para ello debe encontrar un indicador numérico que muestre alguna oscilación al avanzar el tiempo y para poder visualizar en $d > 2$ puede por ejemplo graficar en una tajada de $d = 2$ haciendo el resto de las coordenadas fijas.

4. (BONUS 1pto) Se otorgará 1pto a la implementación más eficiente. Esto se medirá con alta dimensionalidad y cantidad de pixeles.

6 Entrega

Debe compartir enlace a dicho Jupyter Notebook en classroom con plazo al día Jueves 14 de Abril antes de las 23:55 hrs.