High-Performance Computing para Modelamiento Matemático con (Py)CUDA

Sesión 6: Implementación paralela de modelos discretos.

Diego Maldonado

Universidad Católica del Maule

18 de diciembre de 2025

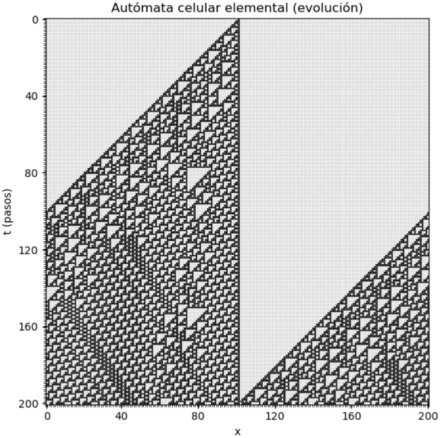
Índice

Definición

Un autómata celular es una tupla (d, S, N, f) donde S es un conjunto finito no vacío de estados, N es una vecindad y $f : S^N \to S$ es una función local. Definimos la función global $F : S^{\mathbb{Z}^d} \to S^{\mathbb{Z}^d}$ tal que

$$\forall x \in \mathbb{Z}^d : \; F(c)_x = f(c|_{N_x})$$

donde $c|_{N_x}$ es la restricción de c a la vecindad N_x.



Regla 110, c.i. en la fila superior y tiempo hacia abajo ↓.

Definición

Un autómata celular elemental (ACE) es un autómata celular $(\{0,1\}, N, f)$ donde $N = \{-1, 0, 1\}$ y $f : \{0,1\}^3 \to \{0,1\}$.

Notas

La regla de un ACE se puede representar como un número cuya representación binaria corresponda con la regla del autómata.

La regla local se puede representar con una tabla

	c_{-1}	c_0	c_1	$f(c\|_{N_x})$
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

Tabla de transición del autómata celular elemental 110.

Luego la tabla se transforma en binario (y viceversa):
$110 = 0 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$

Algoritmo general de simulación de un AC:

Entrada: Número de celdas n, vecindad $N \subset [0, ..., n-1]$, c.i. $c^0 \in S^n$, regla local f, número de pasos T

Salida: Evolución $c^0, c^1, \ldots, c^T$ del autómata celular 1D.

Para $t = 1, \ldots, T$ **hacer**
  Para $i = 1, \ldots, n-2$ **hacer**
    $c_i^{t+1} = f(c_{i+N}^t)$;
  Condición de contorno en $i = 0, n-1$
Devolver $\{c^t\}_{t=0}^T$;

Algoritmo general de simulación de un AC:

Entrada: Número de celdas n, vecindad $N \subset [0, ..., n-1]$, c.i. $c^0 \in S^n$, regla local f, número de pasos T

Salida: Evolución $c^0, c^1, \ldots, c^T$ del autómata celular 1D.

Para $t = 1, \ldots, T$ **hacer**
  Para $i = 1, \ldots, n-2$ **hacer**
    $c_i^{t+1} = f(c_{i+N}^t)$;
  Condición de contorno en $i = 0, n-1$
Devolver $\{c^t\}_{t=0}^T$;

Algoritmo de simuación de un ACE:

Entrada: Número de celdas n, c.i. $c^0 \in \{0,1\}^n$, regla $R \in [0, ..., 255]$, núm. pasos T

Salida: Evolución $s^0, s^1, \ldots, s^T$ del autómata celular 1D.

R_{list} = lista con los dígitos binarios de R;
Para $t = 1, \ldots, T$ **hacer**
  Para $i = 1, \ldots, n-2$ **hacer**
    $c_i^{t+1} = R_{list}[4 \cdot c_{-1} + 2 \cdot c_0 + 1 \cdot c_1]$;
  Condición de contorno en $i = 0, n-1$
Devolver $\{c^t\}_{t=0}^T$;

Arreglos 2D

¿Por qué no pasamos "matrices 2D" directamente al kernel?

- En CUDA, la memoria global del **device** se maneja como un bloque lineal de bytes: un puntero (`T*`) apunta a una región contigua.
- Un "arreglo 2D" en C/C++ puede significar cosas distintas:
  - **Contiguo** (`T[n][m]`) vs **arreglo de punteros** (`T**`).
  - El caso `T**` no garantiza contigüidad y requiere que exista además un arreglo de punteros (también en GPU).
- Para indexar `A[i][j]` el compilador necesita conocer el stride (ancho de fila). Si el tamaño es dinámico, lo habitual es pasar `n,m` y usar indexación lineal:

$$A[i,j] \longleftrightarrow \texttt{A[i*m + j]} \quad \text{(row-major)}$$

- Por rendimiento y simplicidad, se usa un vector 1D contiguo y dentro del kernel se interpreta como 2D.

Nota

El procedimiento es:

1. Se define A 2D y se inicializa
2. Se deja A contiguo en memoria con `A = np.ascontiguousarray(A)`
3. Se aplana A con `A_flat = A.ravel()`

```
n = 5
A = np.arange(n*n, dtype=np.float64).
    reshape(n, n)
A = np.ascontiguousarray(A)
A_flat = A.ravel()
```

Arreglos 2D
El truco

```
__global__ void kernel(double *flat_A, int n)
{
    int i = blockIdx.y * blockDim.y + threadIdx.y; // fila
    int j = blockIdx.x * blockDim.x + threadIdx.x; // col

    if (i < n && j < n) {
        #define A(ii,jj) flat_A[(ii)*n + (jj)] // define A en 2D
        A(i,j) = A(i,j) + 1.0;
        #undef A
    }
}
```

`#define A(ii,jj) flat_A[(ii)*n + (jj)]`

- Define un **alias** para interpretar el arreglo 1D `flat_A` como una **matriz** $n \times n$.
- `(ii)*n` avanza **ii filas** completas (cada fila tiene n elementos).
- `+(jj)` avanza **jj columnas** dentro de la fila.
- Por tanto, escribir `A(i,j)` equivale a acceder a: `flat_A[i*n + j]`

Ejercicio

Haga un programa en CUDA que calcule la transpuesta de una matriz en tiempo $\mathcal{O}(1)$ usando $\mathcal{O}(n)$ hilos.

Autómatas Celulares 2D

El Juego de la vida de Conway

Definición. El *Juego de la Vida* es un autómata celular 2D con celdas $c^t_{i,j} \in \{0,1\}$ (muerta/viva) en una grilla. La evolución se determina por una **regla local** aplicada en paralelo.

Vecindad de Moore (8 vecinos). Sea

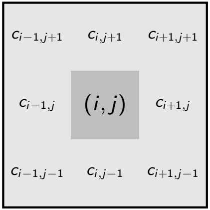
$$N^t_{i,j} = \sum_{(p,q)\in\{-1,0,1\}^2\setminus\{(0,0)\}} c^t_{i+p,\,j+q}.$$

Regla local (B3/S23).

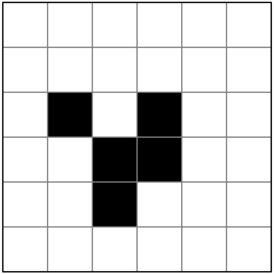
- **Supervivencia:** si $c^t_{i,j} = 1$ y $N^t_{i,j} \in \{2,3\}$ entonces $c^{t+1}_{i,j} = 1$.
- **Muerte:** si $c^t_{i,j} = 1$ y $N^t_{i,j} \notin \{2,3\}$ entonces $c^{t+1}_{i,j} = 0$.
- **Nacimiento:** si $c^t_{i,j} = 0$ y $N^t_{i,j} = 3$ entonces $c^{t+1}_{i,j} = 1$.

(Opcional) Condiciones de borde típicas: periódicas (toro) o bordes fijos.

Vecindad de Moore



Ejemplo de configuración



Autómatas Celulares 2D

El Juego de la vida de Conway

Definición. El *Juego de la Vida* es un autómata celular 2D con celdas $c_{i,j}^t \in \{0,1\}$ (muerta/viva) en una grilla. La evolución se determina por una **regla local** aplicada en paralelo.

Vecindad de Moore (8 vecinos). Sea

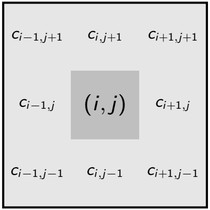
$$N_{i,j}^t = \sum_{(p,q)\in\{-1,0,1\}^2\setminus\{(0,0)\}} c_{i+p,\,j+q}^t.$$

Regla local (B3/S23).

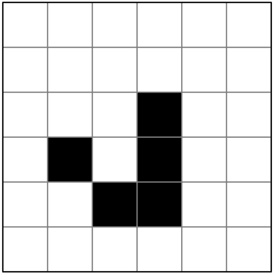
- **Supervivencia:** si $c_{i,j}^t = 1$ y $N_{i,j}^t \in \{2,3\}$ entonces $c_{i,j}^{t+1} = 1$.
- **Muerte:** si $c_{i,j}^t = 1$ y $N_{i,j}^t \notin \{2,3\}$ entonces $c_{i,j}^{t+1} = 0$.
- **Nacimiento:** si $c_{i,j}^t = 0$ y $N_{i,j}^t = 3$ entonces $c_{i,j}^{t+1} = 1$.

(Opcional) Condiciones de borde típicas: periódicas (toro) o bordes fijos.

Vecindad de Moore



Ejemplo de configuración



Autómatas Celulares 2D
El Juego de la vida de Conway

Definición. El *Juego de la Vida* es un autómata celular 2D con celdas $c_{i,j}^{t} \in \{0,1\}$ (muerta/viva) en una grilla. La evolución se determina por una **regla local** aplicada en paralelo.

Vecindad de Moore (8 vecinos). Sea

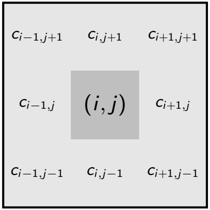
$$N_{i,j}^{t} = \sum_{(p,q)\in\{-1,0,1\}^2\setminus\{(0,0)\}} c_{i+p,\,j+q}^{t}.$$

Regla local (B3/S23).

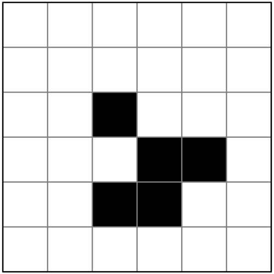
- **Supervivencia:** si $c_{i,j}^{t} = 1$ y $N_{i,j}^{t} \in \{2,3\}$ entonces $c_{i,j}^{t+1} = 1$.
- **Muerte:** si $c_{i,j}^{t} = 1$ y $N_{i,j}^{t} \notin \{2,3\}$ entonces $c_{i,j}^{t+1} = 0$.
- **Nacimiento:** si $c_{i,j}^{t} = 0$ y $N_{i,j}^{t} = 3$ entonces $c_{i,j}^{t+1} = 1$.

(Opcional) Condiciones de borde típicas: periódicas (toro) o bordes fijos.

Vecindad de Moore



Ejemplo de configuración



Definición. El *Juego de la Vida* es un autómata celular 2D con celdas $c_{i,j}^t \in \{0,1\}$ (muerta/viva) en una grilla. La evolución se determina por una **regla local** aplicada en paralelo.

Vecindad de Moore (8 vecinos). Sea

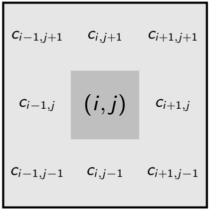
$$N_{i,j}^t = \sum_{(p,q)\in\{-1,0,1\}^2\setminus\{(0,0)\}} c_{i+p,\,j+q}^t.$$

Regla local (B3/S23).

- **Supervivencia:** si $c_{i,j}^t = 1$ y $N_{i,j}^t \in \{2,3\}$ entonces $c_{i,j}^{t+1} = 1$.
- **Muerte:** si $c_{i,j}^t = 1$ y $N_{i,j}^t \notin \{2,3\}$ entonces $c_{i,j}^{t+1} = 0$.
- **Nacimiento:** si $c_{i,j}^t = 0$ y $N_{i,j}^t = 3$ entonces $c_{i,j}^{t+1} = 1$.

(Opcional) Condiciones de borde típicas: periódicas (toro) o bordes fijos.

Vecindad de Moore



Ejemplo de configuración