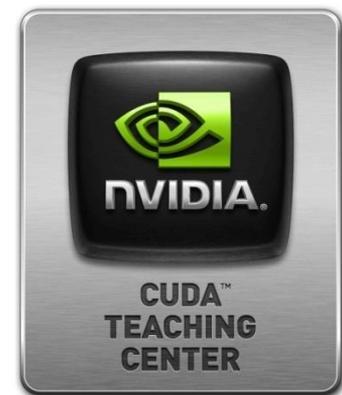


Programación masivamente paralela en procesadores gráficos (GPUs)

E. Dufrechou , P. Ezzatti y M. Pedemonte



Clase 8

Patrones de cómputo 2

Contenido

- **Stencil**
 - Tiling
 - Padding
 - Memoria constante y caché
- **Scan**

Stencil

Stencil

- **Se caracteriza por actualizar los elementos de una grilla de acuerdo a un patrón fijo conocido a priori (stencil - plantilla).**
- **Suelen usarse en métodos iterativos por lo que se realiza una secuencia de actualizaciones sobre la grilla.**
- **En general, la grilla suele ser de 2 o 3 dimensiones.**
- **En cada paso de tiempo, se actualizan todos los elementos de la grilla.**
- **Para calcular el nuevo valor de cada celda de la grilla, se utilizan los elementos adyacentes de la matriz usando un patrón regular de cómputo.**

Stencil

- Este tipo de patrón de cómputo es común en la resolución de ecuaciones diferenciales en derivadas parciales, autómatas celulares, el método Gauss-Seidel, procesamiento de imágenes.
- Un ejemplo conocido: El juego de la vida (Game of life)

<https://youtu.be/C2vgICfQawE>

Stencil

- **Convolución:**
 - **Se usa ampliamente en el procesamiento de audio, imágenes y video.**
 - **En general se utilizan como un filtro que transforma los valores de los pixeles.**
 - **Se pueden definir como una operación matricial en la que cada elemento de la salida es una suma ponderada de un subconjunto de los elementos de entrada vecinos.**
 - **Los pesos usados en el cálculo de la suma ponderada se definen a través de máscaras.**

Stencil

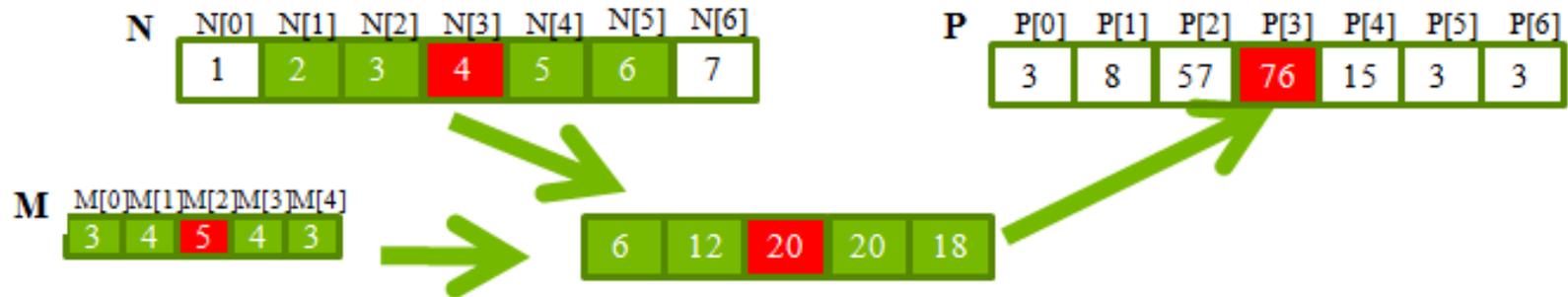
- Ejemplo de convolución 1D
- Se suelen usar para procesamiento de audio.
- Las máscaras suelen tener largo impar por simetría.



$$P[2] = N[0]*M[0] + N[1]*M[1] + N[2]*M[2] + N[3]*M[3] + N[4]*M[4]$$

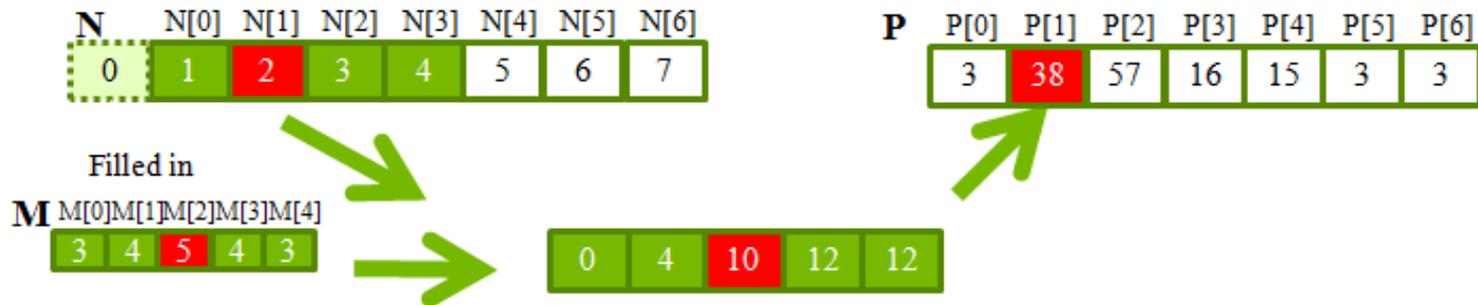
Stencil

- Ejemplo de convolución 1D



Stencil

- Ejemplo de convolución 1D



- En los bordes hay que hacer correcciones: agregar 0s, repetir los valores del borde, etc.
- Para “agregar 0s” basta con poner un if dentro del loop que calcula el nuevo valor.

Stencil

- **Ejemplo de convolución 1D**

```
__global__ void convolution_1D_basic_kernel(float *N, float *M,
    float *P, int Mask_Width, int Width)
{
    int i = blockIdx.x*blockDim.x + threadIdx.x;

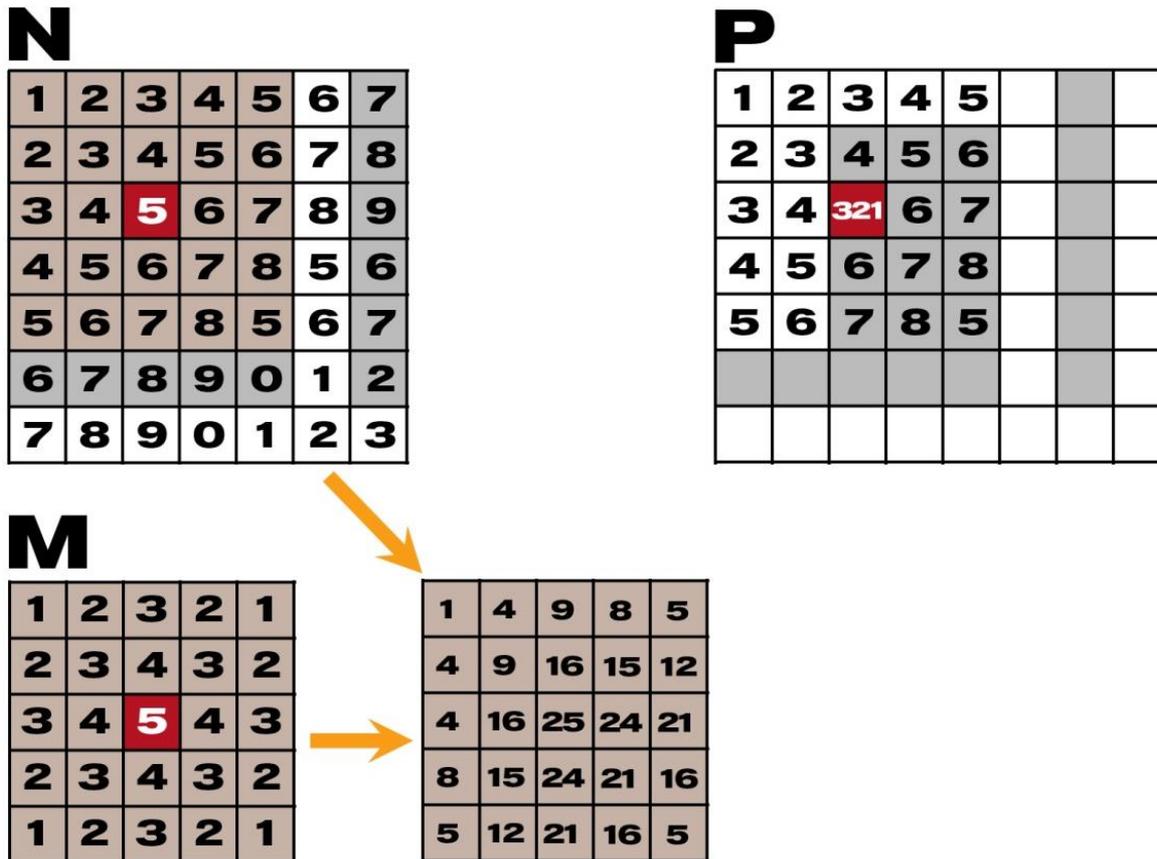
    float Pvalue = 0;
    int N_start_point = i - (Mask_Width/2);

    for (int j = 0; j < Mask_Width; j++) {
        if (N_start_point + j >= 0 && N_start_point + j < Width) {
            Pvalue += N[N_start_point + j]*M[j];
        }
    }

    P[i] = Pvalue;
}
```

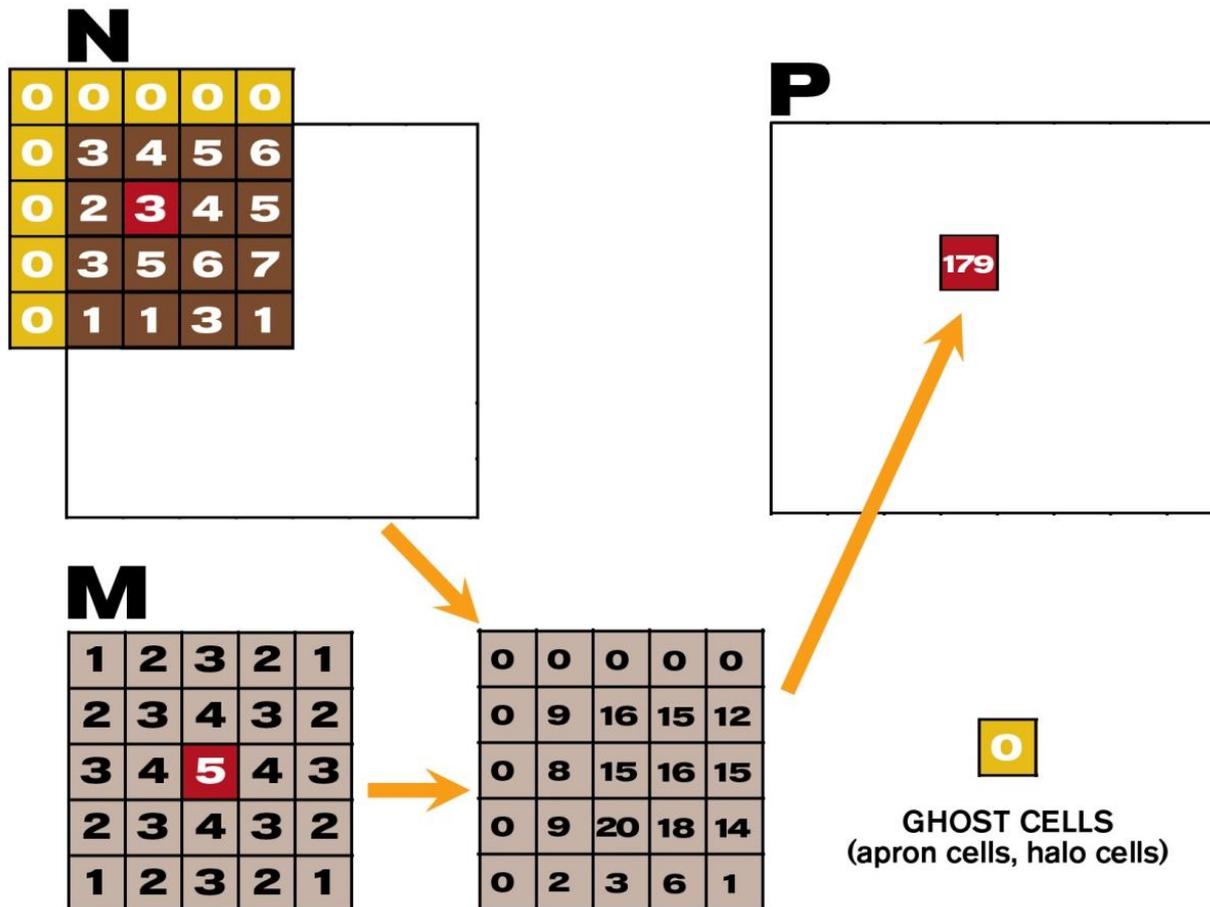
Stencil

- Ejemplo de convolución 2D



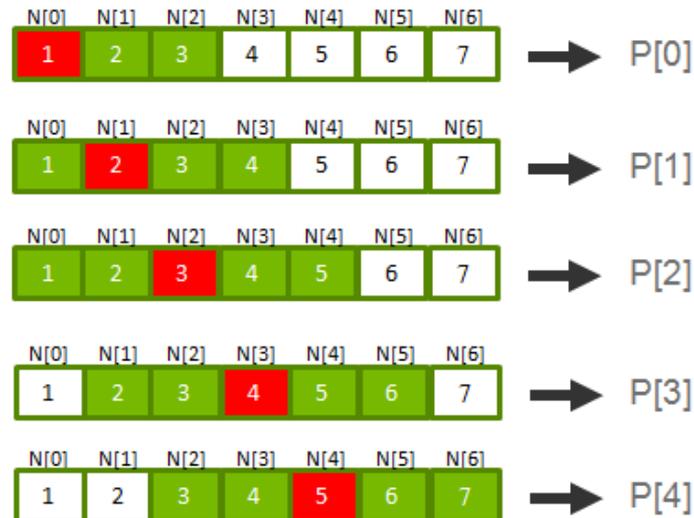
Stencil

- Ejemplo de convolución 2D



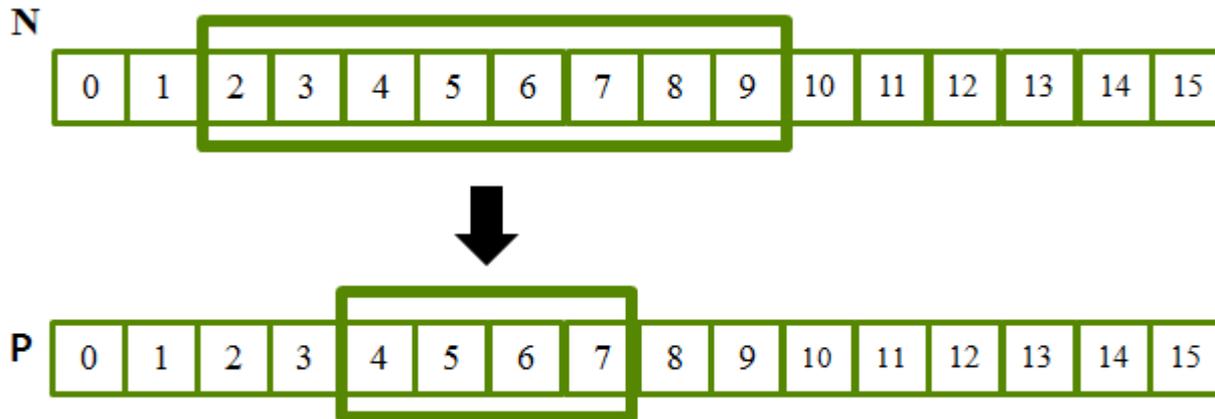
Stencil

- El cálculo de elementos de salida adyacente involucra el uso de elementos de entrada comunes.
- $N[2]$ se usa en el cálculo de $P[0]$, $P[1]$, $P[2]$, $P[3]$ y $P[4]$.
- Se pueden cargar los elementos necesarios para la ejecución de todos los hilos del bloque en memoria compartida para reducir el acceso a memoria global.



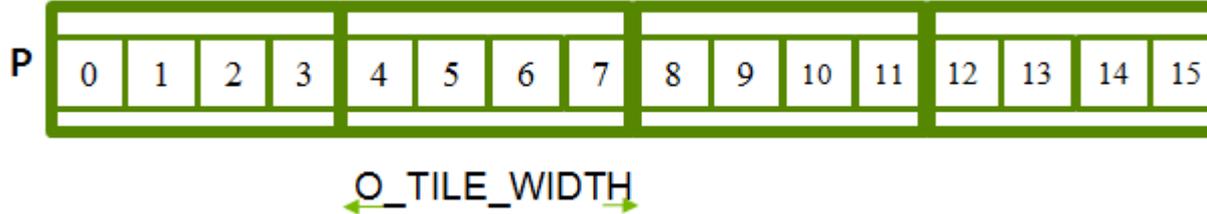
Stencil

- ¿Cuántos elementos se precisan?
- Si queremos que cada bloque calcule T elementos de salida:
 - $T + \text{Mask_Width} - 1$ elementos se necesitan para T salidas.
 - $T + \text{Mask_Width} - 1$ no es múltiplo de T (en general).
 - T suele ser mucho más grande que Mask_Width



Stencil

- Cada bloque de hilos calcula un tile de salida
- Cada tile de salida tiene como ancho `O_TILE_WIDTH`



Stencil

- **Alternativas de diseño 1:**

- El tamaño de cada bloque de hilos coincide con el tamaño del tile de salida.
- Todos los hilos participan en calcular elementos de la salida.
- Algunos hilos tienen que cargar más de un elemento en memoria compartida.

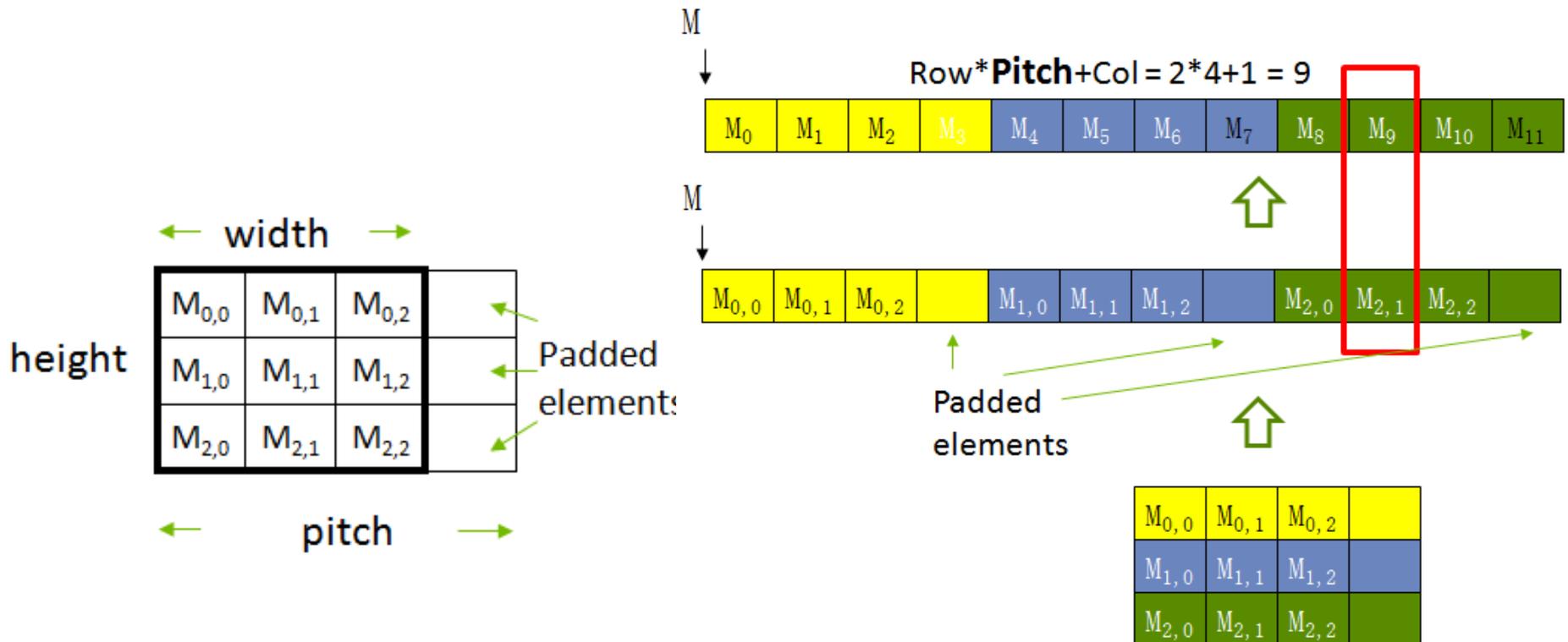
- **Alternativas de diseño 2:**

- El tamaño de cada bloque de hilos coincide con el tamaño del tile de entrada.
- Algunos hilos no participan en calcular elementos de la salida.
- Todos los hilos tienen que cargar un elemento en memoria compartida.

Stencil

- **Padding en stencils 2D:**

- Es deseable en algunas oportunidades completar cada fila de la matriz 2D para que coincida con los segmentos de la memoria global.
- Corresponde a agregar columnas.



Stencil

- **Memoria constante y caché para la máscara:**
 - La máscara es usada por todos los hilos pero no es modificada por la convolución.
 - Todos los hilos de un warp acceden a las mismas posiciones de la máscara en el mismo paso de tiempo.
 - La memoria constante utiliza fuertemente el caché de la GPU.
 - Es una forma efectiva de aumentar el ancho de banda real sin consumir shared memory.
 - Se puede indicar al compilador con `const __restrict__` que el parámetro que se usa para la máscara es elegible para ser cacheado en memoria constante.

```
global__ void convolution_2D_kernel(float *P,  
    float *N, height, width, channels,  
    const float __restrict__ *M) {
```

Scan

Scan

- Es un algoritmo clave ya que permite convertir una estrategia cómputo secuencial en una paralela.
- Es un patrón de computación paralela de los más estudiados.
- Se puede implementar de forma eficiente en paralelo.

Definición: La operación *scan* toma un operador binario (que tiene que ser asociativo) y un arreglo de n elementos $[x_0, x_1, \dots, x_{n-1}]$ y devuelve el arreglo $[x_0, (x_0 \oplus x_1), \dots, (x_0 \oplus x_1 \oplus \dots \oplus x_{n-1})]$.

Ejemplo: Si \oplus es la suma, aplicar la operación *scan* al arreglo $[3 \ 1 \ 7 \ 0 \ 4 \ 1 \ 6 \ 3]$ da como resultado $[3 \ 4 \ 11 \ 11 \ 15 \ 16 \ 22 \ 25]$.

Scan

- **Se busca convertir recurrencias secuenciales como:**

```
out[0] = in[0];  
for (j=1; j<n; j++)  
    out[j] = out[j-1] + f(in[j]);
```

- **En un cómputo paralelo como:**

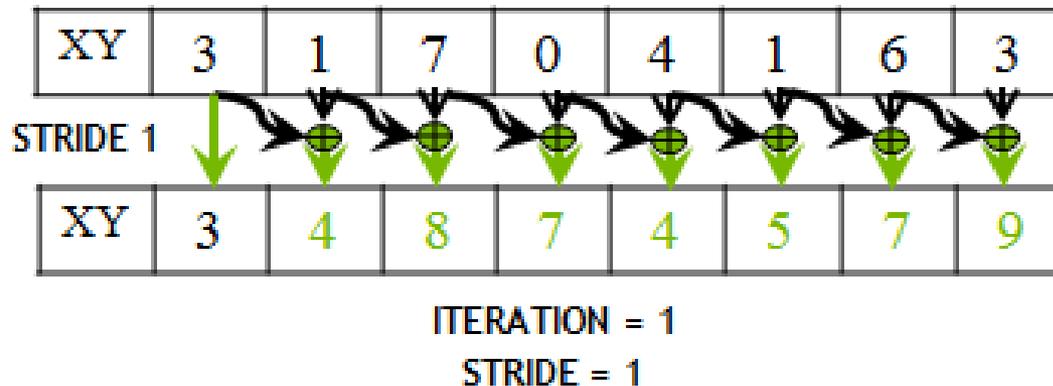
```
forall (j) {  
    temp[j] = f(in[j]);  
}  
scan(out, temp);
```

Scan

- Dada la secuencia $[x_0, x_1, x_2, \dots]$ calcular $[y_0, y_1, y_2, \dots]$ tal que:
 - $y_0 = x_0$
 - $y_1 = x_0 + x_1$
 - $y_2 = x_0 + x_1 + x_2$
- De forma secuencial se utiliza la definición recursiva $y_i = y_{i-1} + x_i$
- De forma paralela naïve se asigna un hilo a calcular cada elemento de y :
 - Cada hilo suma todos los elementos de x que se necesitan para el elemento y
 - $y_0 = x_0$
 - $y_1 = x_0 + x_1$
 - $y_2 = x_0 + x_1 + x_2$
 - Obviamente que este enfoque es muy ineficiente.

Scan

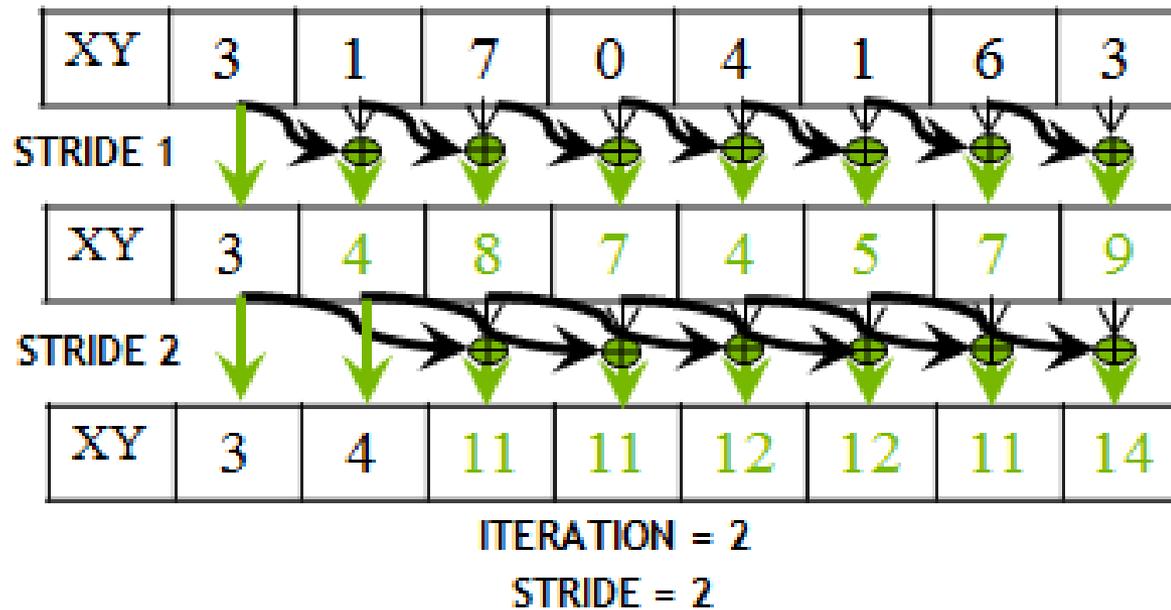
- Algoritmo paralelo:
 - Se leen los valores a memoria shared.
 - Se itera $\log(n)$ veces
 - El stride varía entre 1 y $n-1$, en cada iteración se dobla el stride.



- Requiere usar barreras de sincronización antes de leer y antes de escribir.

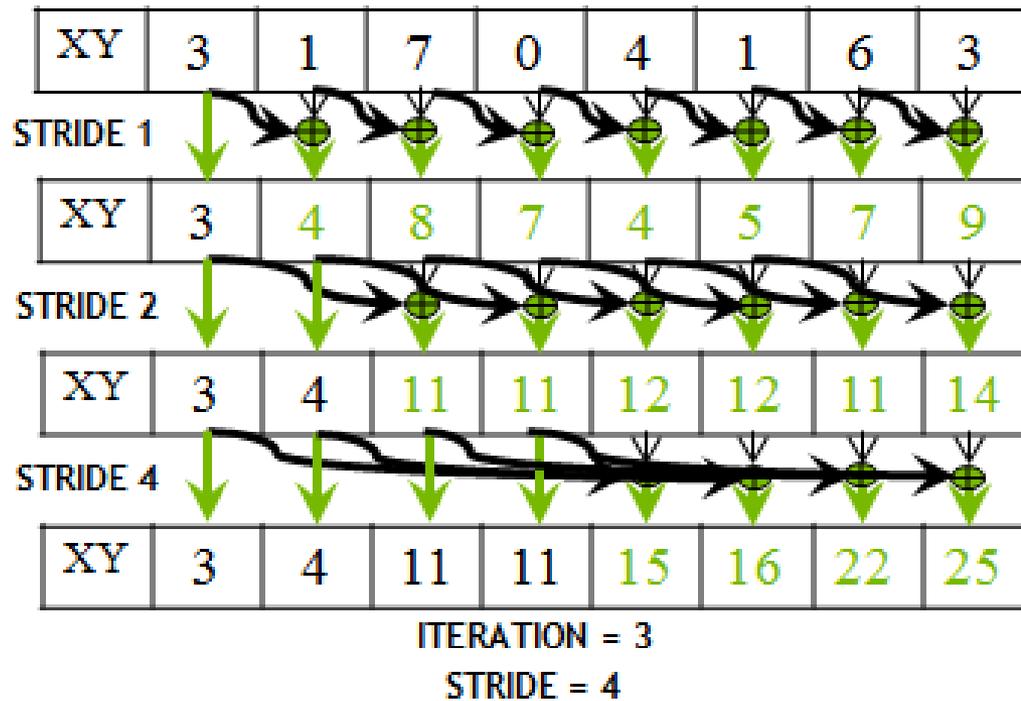
Scan

- Algoritmo paralelo:



Scan

- Algoritmo paralelo:

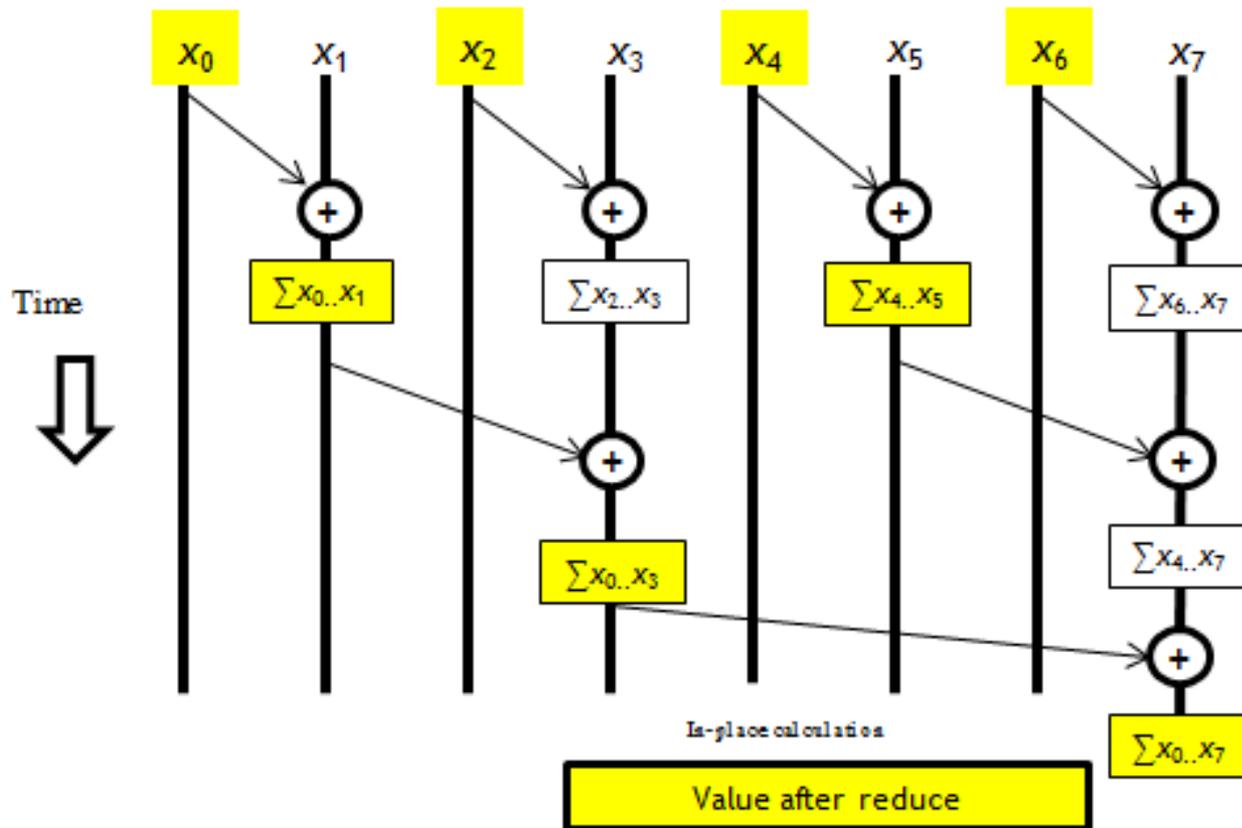


Scan

- Para mejorar la eficiencia se construye conceptualmente un árbol binario balanceado con los datos de entrada.
- Hay una primera etapa en la que se navega de las hojas a la raíz construyendo sumas parciales en los nodos internos del árbol.
- La raíz tiene como resultado la suma de todas las hojas.
- En una segunda etapa se navega en reversa para construir las salidas a partir de las sumas parciales.

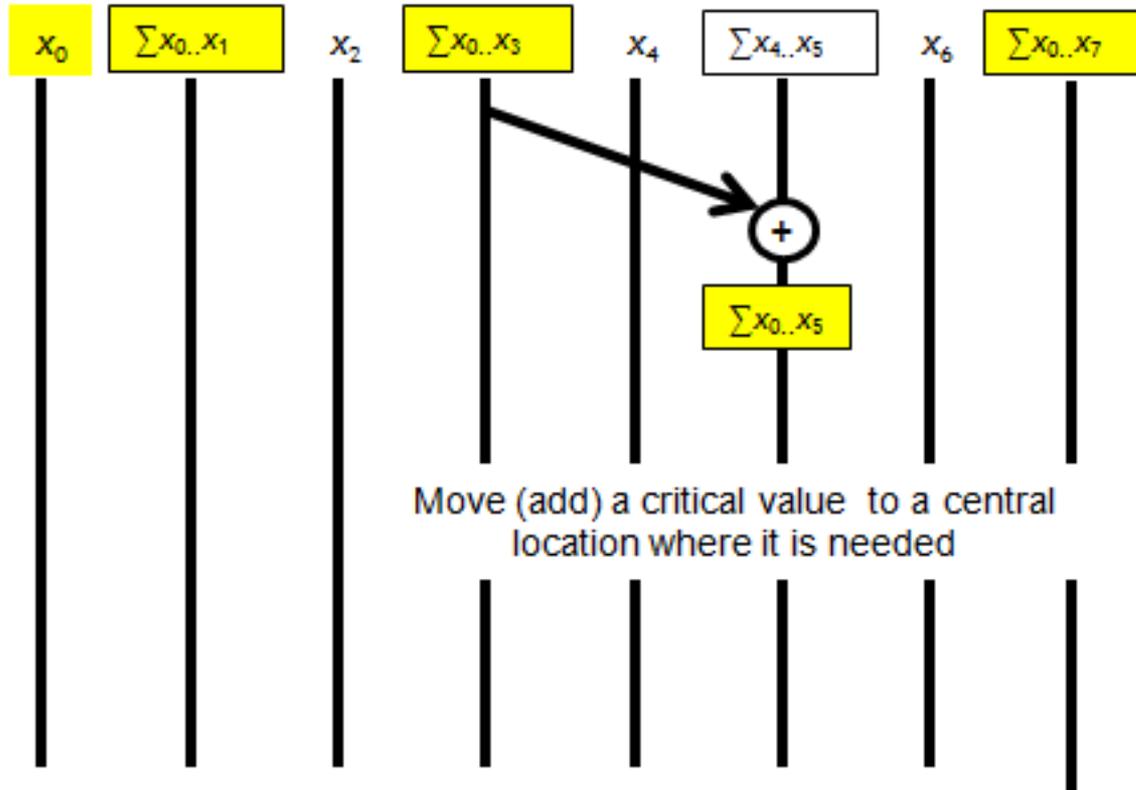
Scan

- Etapa de reducción:



Scan

- Fase reversa – post-reducción:



Scan

- Fase reversa – post-reducción:

