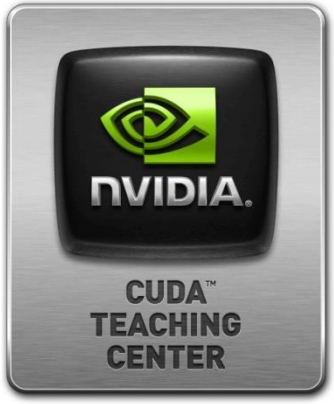
Programación masivamente paralela en procesadores gráficos (GPUs)

E. Dufrechou, M. Freire, P. Ezzatti y M. Pedemonte



Clase 10
Ecosistema CUDA 2 (bibliotecas)

Contenido

- Introducción
- Building blocks
- Bibliotecas
  - Thrust
  - CUB
  - Modern GPU
- Ejemplos

Introducción

- Existen tipos de problemas muy repetidos por lo que es interesante hacer soluciones estándar (patrones de cómputo). Ej: Scan, Reduce
- La programación en CUDA es compleja y requiere una gran curva de aprendizaje
- Tiene sentido hacer implementaciones (relativamente) eficientes y genéricas que puedan ser usadas por programadores con poca experiencia

Building blocks

Building Blocks

- **En general se piensan como funciones sobre vectores o conjuntos**
- **Algunos ejemplos de estos “building blocks” son:**
  - Copy
  - Foreach
  - Transform
  - Reduce
  - Scan
  - Sort
  - Merge
  - Reorder
  - Search

Copia

- Genera una copia de un vector, eventualmente con modificaciones

Copia

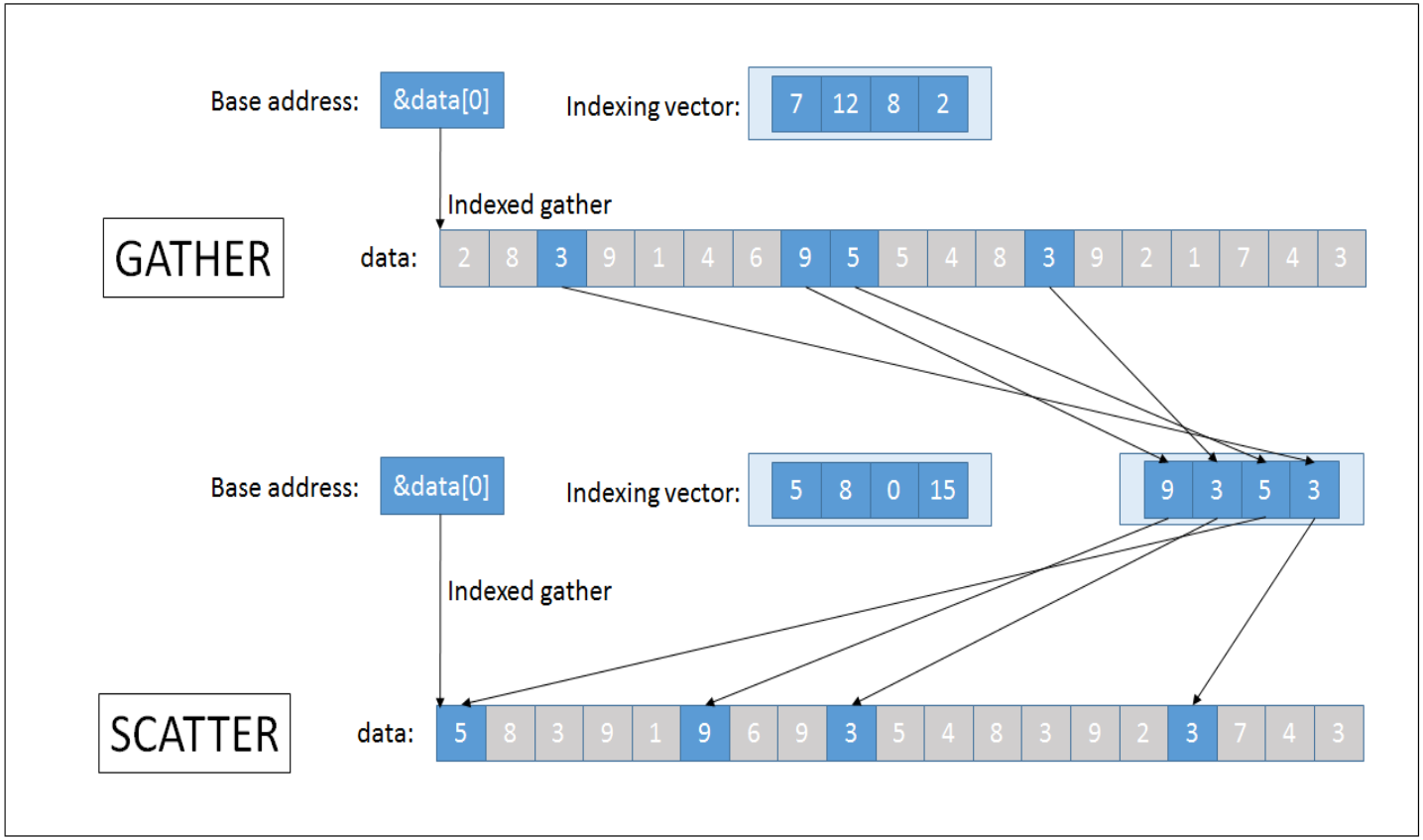
- **Genera una copia de un vector, eventualmente con modificaciones**
  - **Copy**
  - Swap
  - Gather
  - Scatter

Copia

- **Genera una copia de un vector, eventualmente con modificaciones**
  - Copy
  - **Swap**
  - Gather
  - Scatter

Copia

- **Genera una copia de un vector, eventualmente con modificaciones**
  - Copy
  - Swap
  - **Gather**
  - **Scatter**



Foreach

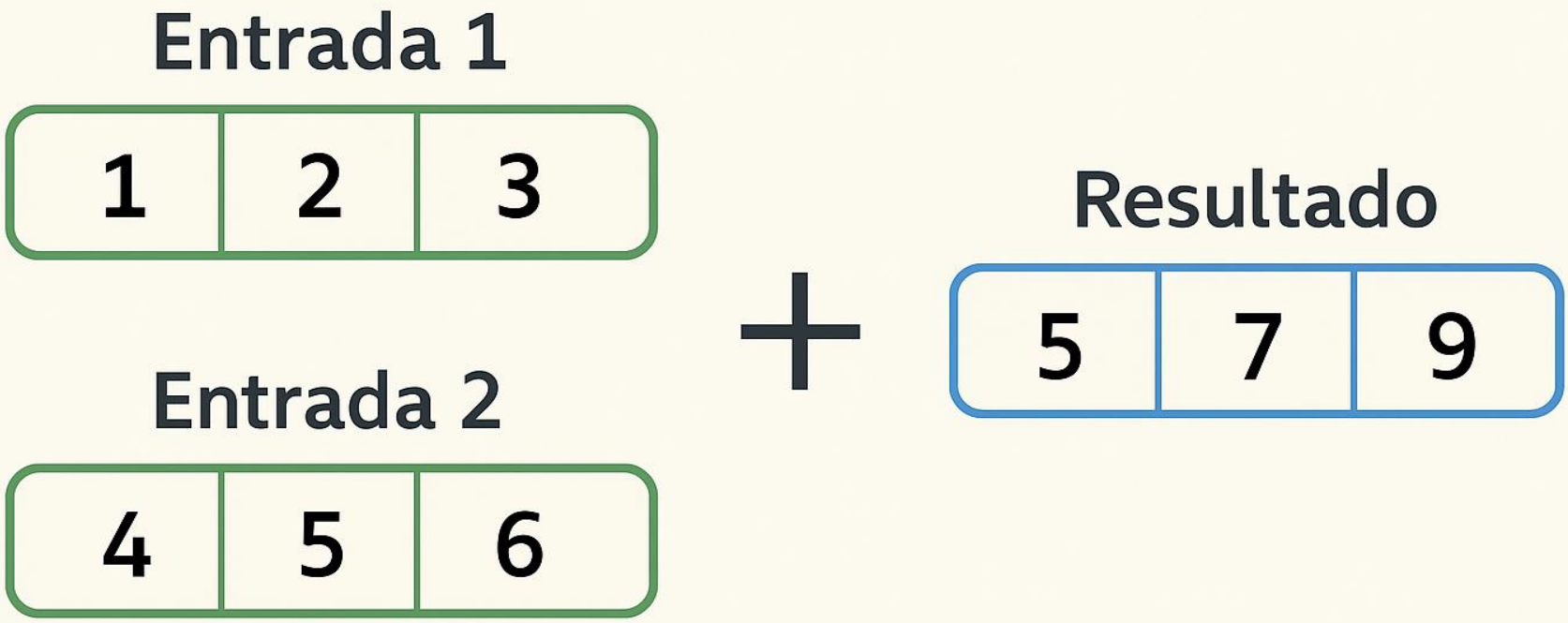
- **¿Qué hace?**
  - Aplica una función a todos los elementos de un vector
    - Generalmente in-place
- **¿Cuándo se usa?**
  - Escalar los elementos de un vector
  - Aplicar funciones sencillas (x^2, log(x), etc.)

Foreach

Transform

- **¿Qué hace?**
  - Aplica una función a un vector o conjunto de vectores (similar a foreach pero con dos diferencias principales)
    - Generalmente out-of-place (retorno)
    - Pueden ser funciones binarias (o de más parámetros)
- **¿Cuándo se usa?**
  - Para aplicar funciones de más de un operador
  - Cuando se quiere mantener el vector original

Transform



Reduce

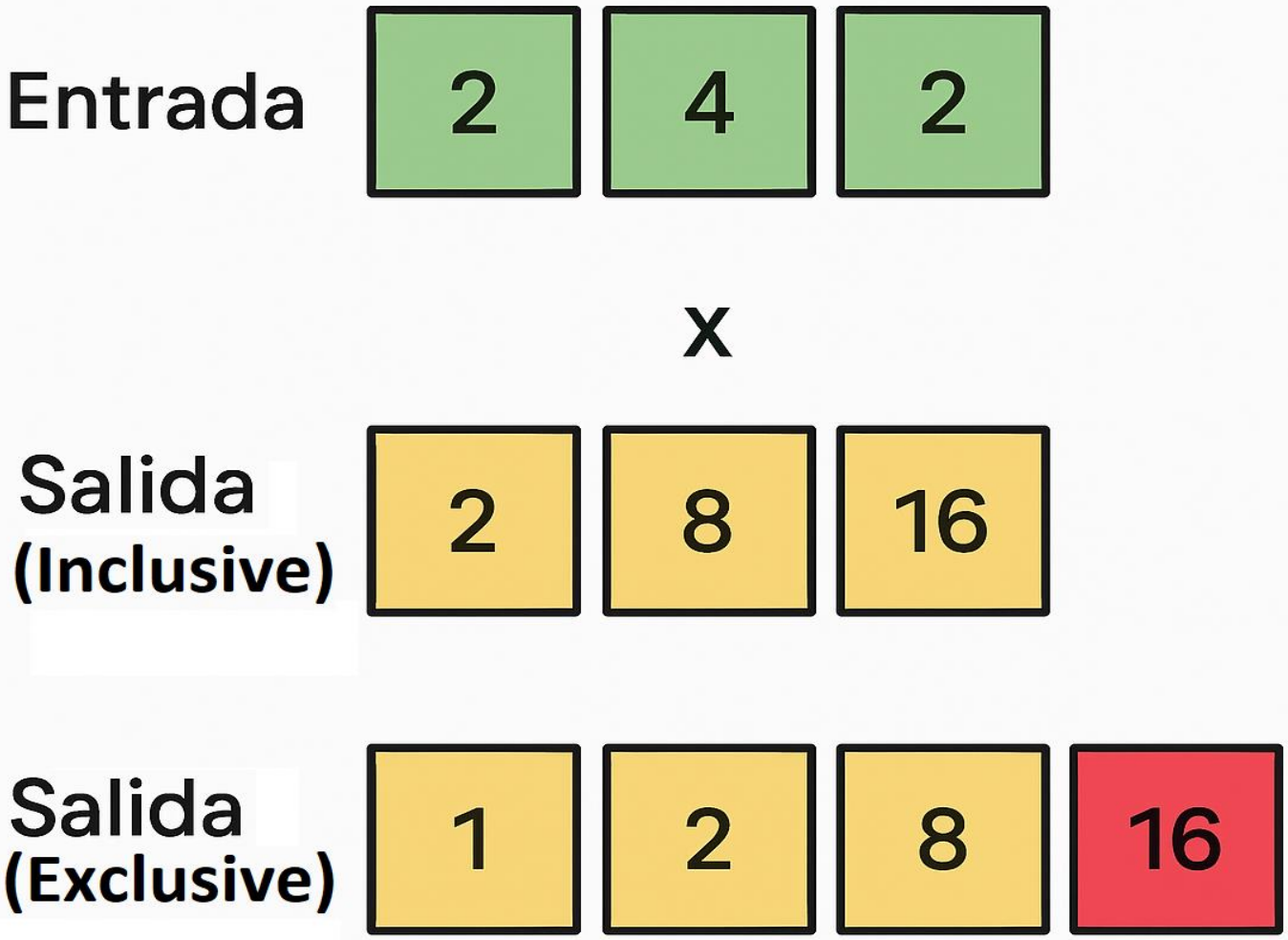
- **¿Qué hace?**
  - Acumula los elementos de un vector en un único valor, usando una operación binaria (ej: suma, max)
- **¿Cuándo se usa?**
  - Para obtener máximo/mínimo global de un vector
  - Obtener la suma/multiplicación de todos los elementos

Reduce

Scan

- **¿Qué hace?**
  - Acumula los elementos de un vector, manteniendo los resultados parciales usando una operación binaria (ej: suma, max)
  - Puede incluir el elemento actual (inclusive) o no (exclusive)
  - Debe brindarse un valor inicial (típicamente 0 o 1)
- **¿Cuándo se usa?**
  - Posiciones acumuladas (offsets)
  - Compactación

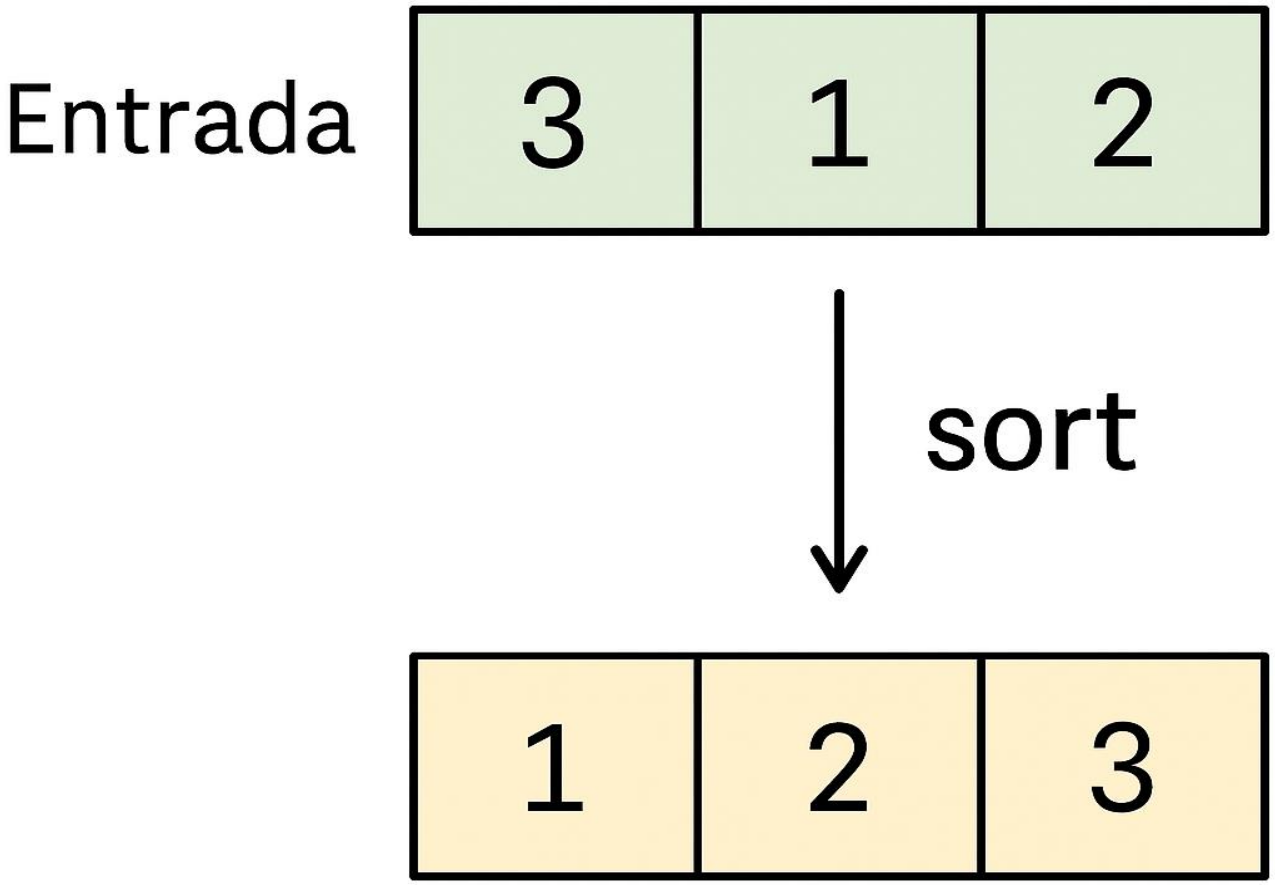
Scan


Entrada
2
4
2
x
Salida
(Inclusive)
2
8
16
Salida
(Exclusive)
1
2
8
16

Sort

- **¿Qué hace?**
  - Ordena un vector, puede mantener el orden parcial entre elementos (stable) o no

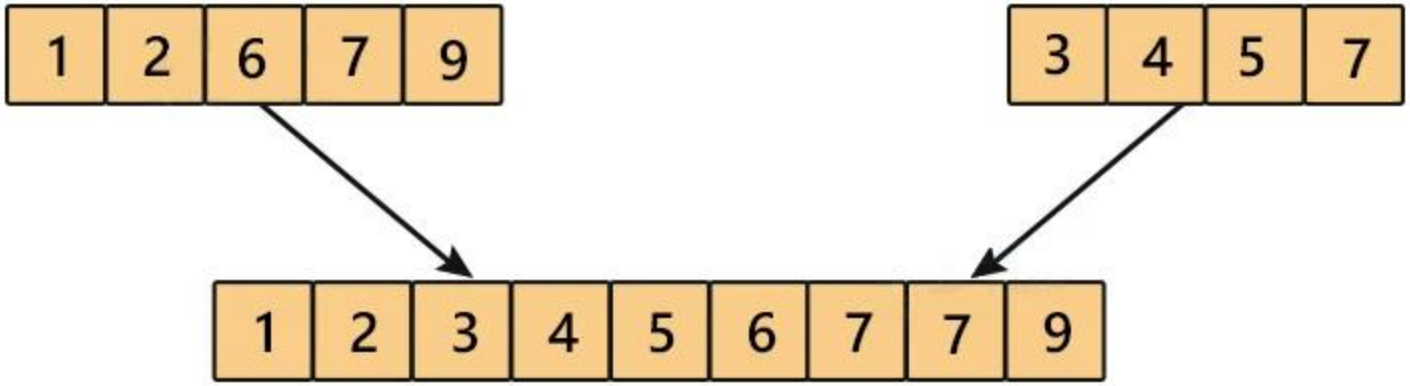
Sort



Merge

- **¿Qué hace?**
  - Unir dos vectores ordenados en un único vector de n+m entradas

Merge



Reorder

- **¿Qué hace?**
  - Reordernar elementos en un vector
- **¿Cuándo se usa?**
  - Particiones: Clasificación binaria (pej: positivos vs negativos)
  - Shuffle: Algoritmos aleatorios (pej: Métodos Montecarlo)
  - Filtrado (o compactado): dada una condición eliminar los elementos de un vector que cumplen una conidición

Reorder


Aleatorio
Entrada:
1 2 3 3 1 5
Resultado:
1 1 5 3 2 3
Particion
Entrada:
2 3 4 1 6 5
Condición: x % 2 == 0
2 4 6 3 1 5
Compactacion
Entrada:
0 1 7 8
Condición: x < 5
0 1

Search

- **¿Qué hace?**
  - Buscar un elemento en un vector, distintos tipos:
    - **Binaria**

Search

- **¿Qué hace?**
  - Buscar un elemento en un vector, distintos tipos:
    - Binaria
    - **Iterativa**

Search

- **¿Qué hace?**
  - Buscar un elemento en un vector, distintos tipos:
    - Binaria
    - Iterativa
    - **Missmatch**

Thrust

Conceptos generales

- Extensión de C++ Standard Template Library (STL) para usar en GPUs (y otros dispositivos de cómputo heterogéneo)
- Abstrae el manejo de memoria utilizando wrappers de punteros (device, host) y contenedores (vector)
- Para CUDA existen implementaciones eficientes de los principales algoritmos (merge, sort, scan, reduce, etc.)
- Interoperabilidad con el resto de CUDA

Manejo de memoria

- Puede hacerse de manera abstracta (con contenedores) o utilizando punteros (usando un wrapper para device)

- Al usar contenedores la reserva y liberación se hace automática (aunque es más eficiente liberar manual!!!)

- Los wrappers mantienen información (tipo, dispositivo, etc), no son únicamente punteros.

- Siempre se puede obtener un puntero con los datos utilizando la función data() (y casteando a raw pointer)

Algoritmos

Se desarrolla mediante el uso de algoritmos de alto nivel, delegando a la biblioteca como se implementa.

Los algoritmos básicos:

- **for_each**
- **scan**
- **sort**
- **reduce**
- **merge**
- **find**
- **transformaciones**
- **etc.**

No se especifica configuración de ejecución !!!

Cub

Conceptos generales

- Surge de Thrust por necesidades de mantenimiento y portabilidad de su rendimiento reusando primitivas a niveles más bajos (block, warp)
- Ligeramente más bajo nivel que thrust
- Es una librería de rutinas implementada específicamente para CUDA C++
- El manejo de memoria es idéntico al de CUDA (cudaMalloc, cudaFree, punteros) y corre por el programador
- Implementaciones (más) eficientes de los principales algoritmos (merge, sort, scan, reduce, etc.)

Algoritmos

- Se dividen en tres grupos según su granularidad (warp, block y device)
- Además de los mencionados anteriormente presenta otros "específicos" tanto a nivel de bloque (discontinuity, adjacent difference) como device (select, runLength encoding, SpMV)
- Se puede trabajar en problemas a batch (segmented sort)

ModernGPU

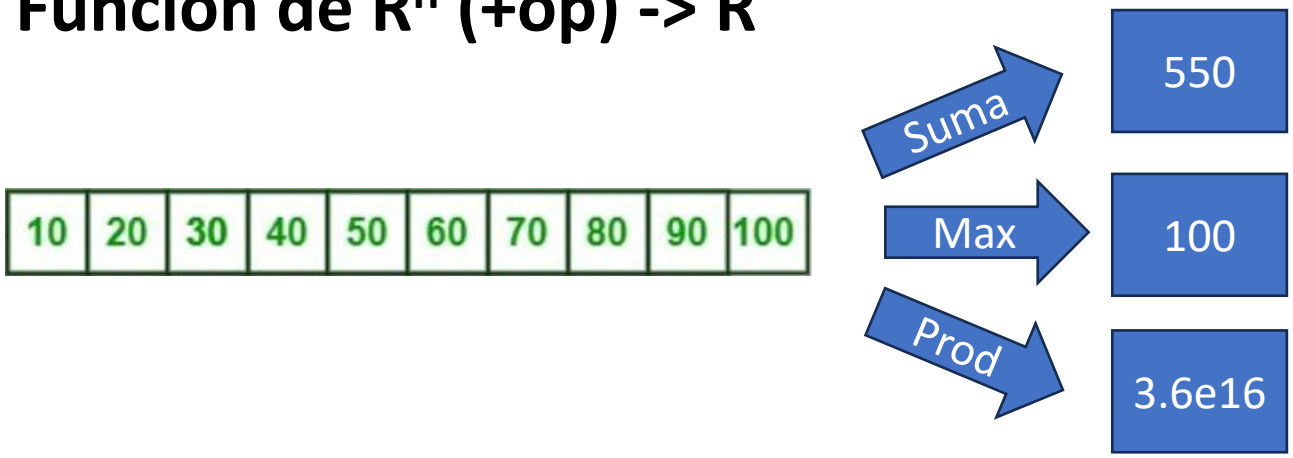
Modern GPU

- Biblioteca pensada para un uso educacional
- Prioriza la claridad y el reuse por sobre la eficiencia computacional
- ModernGPU no ejecuta con valores “grandes”
- No es utilizada en la práctica

Ejemplos de uso

Ejemplo 1 - Reducción

- La reducción aplica un operador de dos (o más) parámetros a un conjunto (vector) de datos
- Función de R^n (+op) -> R



- Es muy usada en GPU para “juntar” los resultados parciales de hilos o bloques por lo que es importante que esté implementada eficientemente

Ejemplo 1 - Reducción

- **Veamos la reducción por suma**
- **Recordemos la implementación**

```
extern __shared__ long int intermedio[];
int idx = blockIdx.x * blockDim.x + threadIdx.x;
if(idx < sz)
    intermedio[threadIdx.x] = input[idx];
else{
    intermedio[threadIdx.x] = 0;
}
__syncthreads();
int i = blockDim.x/2;
while (i != 0) {
if (threadIdx.x < i)
    intermedio[threadIdx.x] =
    intermedio[threadIdx.x] +
    intermedio[threadIdx.x + i];
__syncthreads();
i = i / 2;
}//endwhile

__syncthreads();
if (threadIdx.x==0 && idx < sz)
   output[blockIdx.x] = intermedio[0];
```

Ejemplo 1 - Reducción

- **Veamos la reducción por suma**
- **¿Cómo queda en Thrust?**

```
thrust::device_ptr<long int> temp_ptr(d_in);
thrust::device_vector<long int> in_thrust(temp_ptr, temp_ptr + size);
d_out_h[0]= thrust::reduce(in_thrust.begin(),in_thrust.begin()+size);
cudaDeviceSynchronize();
```

Ejemplo 1 - Reducción

- **Veamos la reducción por suma**
- **¿Cómo queda en Cub?**

```
void     *d_temp_storage = NULL;
size_t   temp_storage_bytes = 0;
cub::DeviceReduce::Sum(
  d_temp_storage, temp_storage_bytes, d_in, d_out,
num_items);

// Allocate temporary storage
cudaMalloc(&d_temp_storage, temp_storage_bytes);

// Run sum-reduction
cub::DeviceReduce::Sum(d_temp_storage, temp_storage_bytes,
d_in, d_out, num_items);
```

Ejemplo 1 - Reducción

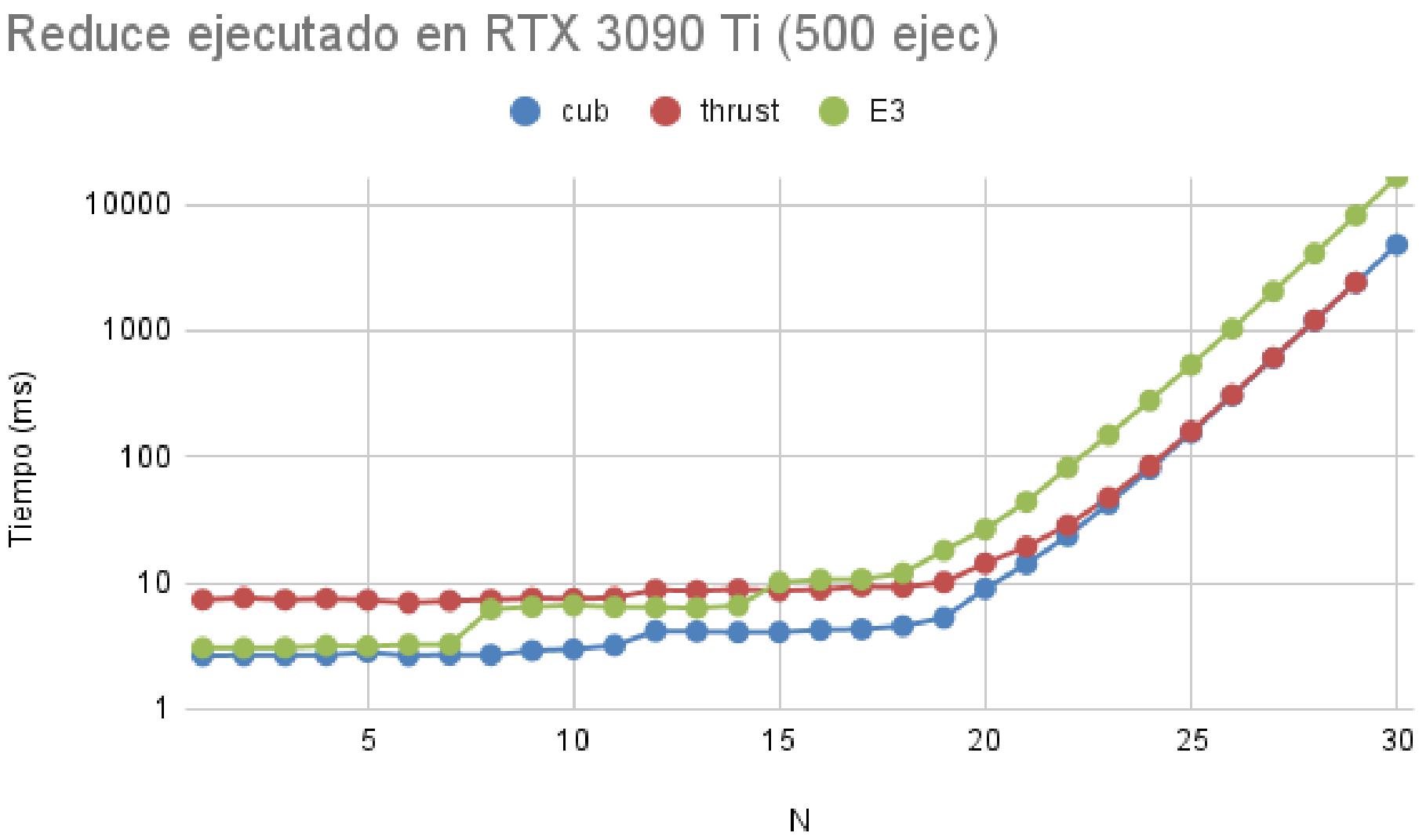
- **Se eligió la primitiva reduce por su extensa utilización**
- **¿Cómo queda en ModernGPU?**

```
ContextPtr context = CreateCudaDevice(0)
MGPU_MEM(int) data = context.GenRandom<int>(size, 0, size);
int total = Reduce(data->get(), size, context);
```

Ejemplo 1 - Reducción

- Se ejecutó cada algoritmo 500 veces en una tarjeta Nvidia RTX 3090 Ti
- Se evaluó el tiempo de ejecución ignorando la primera ejecución (overhead de las bibliotecas)
- Se probaron distintos tamaños medios y grandes (1024*2^N)
- ModernGPU no ejecuta con valores de N>10
- Thrust no ejecuta con el valor de N=30

Ejemplo 1 - Reducción



Ejemplo 1 - Reducción

- **¿Qué pasa si quiero usar otro operador?**
- **Existen otros ya implementados como máximo o mínimo**
- **¿Y si no? Se pueden definir nuevos operadores**

Thrust

```
struct customOP : public
thrust::binary_function<int, int, int> {
   __device__
      int operator()(int fst, int snd) {
         return 7 * snd + fst;
      };
};
```

Cub

```
struct customOP{
   template <typename T>
   __host__ __forceinline__
   T operator()(const T &fst, const T &snd) const{
      return 7 * snd + fst
   }
};
```

Ejemplo 2 – Compresión de matrices

- **Las bibliotecas no solo sirven para paralelizar primitivas, también se puede programar rutinas complejas combinándolas**
- **Tomemos como ejemplo el cargado de matrices dispersas**
  - **Generalmente se guardan como una lista de los valores con sus índices de fila o columna (COO)**
  - **Esta idea se puede mejorar comprimiendo un índice (CSR)**

Ejemplo 2 – Compresión de matrices

Dense

3	0	2	0
0	1	0	2
0	0	5	0
9	0	0	1

COO

row	0	0	1	1	2	3	3
col	0	2	1	3	2	0	3
data	3	2	1	2	5	9	1

CSR

ind_ptr	0	2	4	5	7		
indices	0	2	1	3	2	0	3
data	3	2	1	2	5	9	1

Ejemplo 2 – Compresión de matrices

- **Partimos de tres vectores ¿qué necesitamos?**
  - Cantidad de filas
  - Largo de cada fila
  - Agrupar por filas y ordenar (si ya no está hecho)
- **A priori puede parecer un problema no paralelizable**
- **Podemos combinar varias primitivas para hacer una rutina que pase de COO a CSR**

Ejemplo 2 – Compresión de matrices

```
    // Sort by row
    thrust::device_vector<int> sorted_indices(d_row.size());
    thrust::sequence(sorted_indices.begin(), sorted_indices.end());
    thrust::sort_by_key(d_row.begin(), d_row.end(), sorted_indices.begin());

    // Sort columns and data according to the sorted row indices
    thrust::device_vector<int> d_col_sorted(d_col.size());
    thrust::device_vector<int> d_data_sorted(d_data.size());
    thrust::gather(sorted_indices.begin(), sorted_indices.end(), d_col.begin(), d_col_sorted.begin());
    thrust::gather(sorted_indices.begin(), sorted_indices.end(), d_data.begin(),
d_data_sorted.begin());

    // Count the number of elements per row
    thrust::device_vector<int> row_counts(d_row.size());
    thrust::constant_iterator<int> const_iter(1);
    auto new_end = thrust::reduce_by_key(d_row.begin(), d_row.end(), const_iter,
thrust::make_discard_iterator(), row_counts.begin());
    int num_unique_rows = new_end.second - row_counts.begin();
```

Ejemplo 2 – Compresión de matrices

Vector de 1 a N

```
    // Sort by row
    thrust::device_vector<int> sorted_indices(d_row.size());
    thrust::sequence(sorted_indices.begin(), sorted_indices.end());
    thrust::sort_by_key(d_row.begin(), d_row.end(), sorted_indices.begin());

    // Sort columns and data according to the sorted row indices
    thrust::device_vector<int> d_col_sorted(d_col.size());
    thrust::device_vector<int> d_data_sorted(d_data.size());
    thrust::gather(sorted_indices.begin(), sorted_indices.end(), d_col.begin(), d_col_sorted.begin());
    thrust::gather(sorted_indices.begin(), sorted_indices.end(), d_data.begin(),
d_data_sorted.begin());

    // Count the number of elements per row
    thrust::device_vector<int> row_counts(d_row.size());
    thrust::constant_iterator<int> const_iter(1);
    auto new_end = thrust::reduce_by_key(d_row.begin(), d_row.end(), const_iter,
thrust::make_discard_iterator(), row_counts.begin());
    int num_unique_rows = new_end.second - row_counts.begin();
```

Ejemplo 2 – Compresión de matrices

Ordeno el vector usando la fila como clave

Para cada entrada el vector sorted dice su posición ordenada

Vector de 1 a N

```
    // Sort by row
    thrust::device_vector<int> sorted_indices(d_row.size());
    thrust::sequence(sorted_indices.begin(), sorted_indices.end());
    thrust::sort_by_key(d_row.begin(), d_row.end(), sorted_indices.begin());

    // Sort columns and data according to the sorted row indices
    thrust::device_vector<int> d_col_sorted(d_col.size());
    thrust::device_vector<int> d_data_sorted(d_data.size());
    thrust::gather(sorted_indices.begin(), sorted_indices.end(), d_col.begin(), d_col_sorted.begin());
    thrust::gather(sorted_indices.begin(), sorted_indices.end(), d_data.begin(),
d_data_sorted.begin());

    // Count the number of elements per row
    thrust::device_vector<int> row_counts(d_row.size());
    thrust::constant_iterator<int> const_iter(1);
    auto new_end = thrust::reduce_by_key(d_row.begin(), d_row.end(), const_iter,
thrust::make_discard_iterator(), row_counts.begin());
    int num_unique_rows = new_end.second - row_counts.begin();
```

Ejemplo 2 – Compresión de matrices

Ordeno el vector usando la fila como clave

Vector de 1 a N

Para cada entrada el vector sorted dice su posición ordenada

```
    // Sort by row
    thrust::device_vector<int> sorted_indices(d_row.size());
    thrust::sequence(sorted_indices.begin(), sorted_indices.end());
    thrust::sort_by_key(d_row.begin(), d_row.end(), sorted_indices.begin());

    // Sort columns and data according to the sorted row indices
    thrust::device_vector<int> d_col_sorted(d_col.size());
    thrust::device_vector<int> d_data_sorted(d_data.size());
    thrust::gather(sorted_indices.begin(), sorted_indices.end(), d_col.begin(), d_col_sorted.begin());
    thrust::gather(sorted_indices.begin(), sorted_indices.end(), d_data.begin(),
d_data_sorted.begin());

    // Count the number of elements per row
    thrust::device_vector<int> row_counts(d_row.size());
    thrust::constant_iterator<int> const_iter(1);
    auto new_end = thrust::reduce_by_key(d_row.begin(), d_row.end(), const_iter,
thrust::make_discard_iterator(), row_counts.begin());
    int num_unique_rows = new_end.second - row_counts.begin();
```

Uso sorted para ordenar los otros dos vectores

Ejemplo 2 – Compresión de matrices

Ordeno el vector usando la fila como clave

Para cada entrada el vector sorted dice su posición ordenada

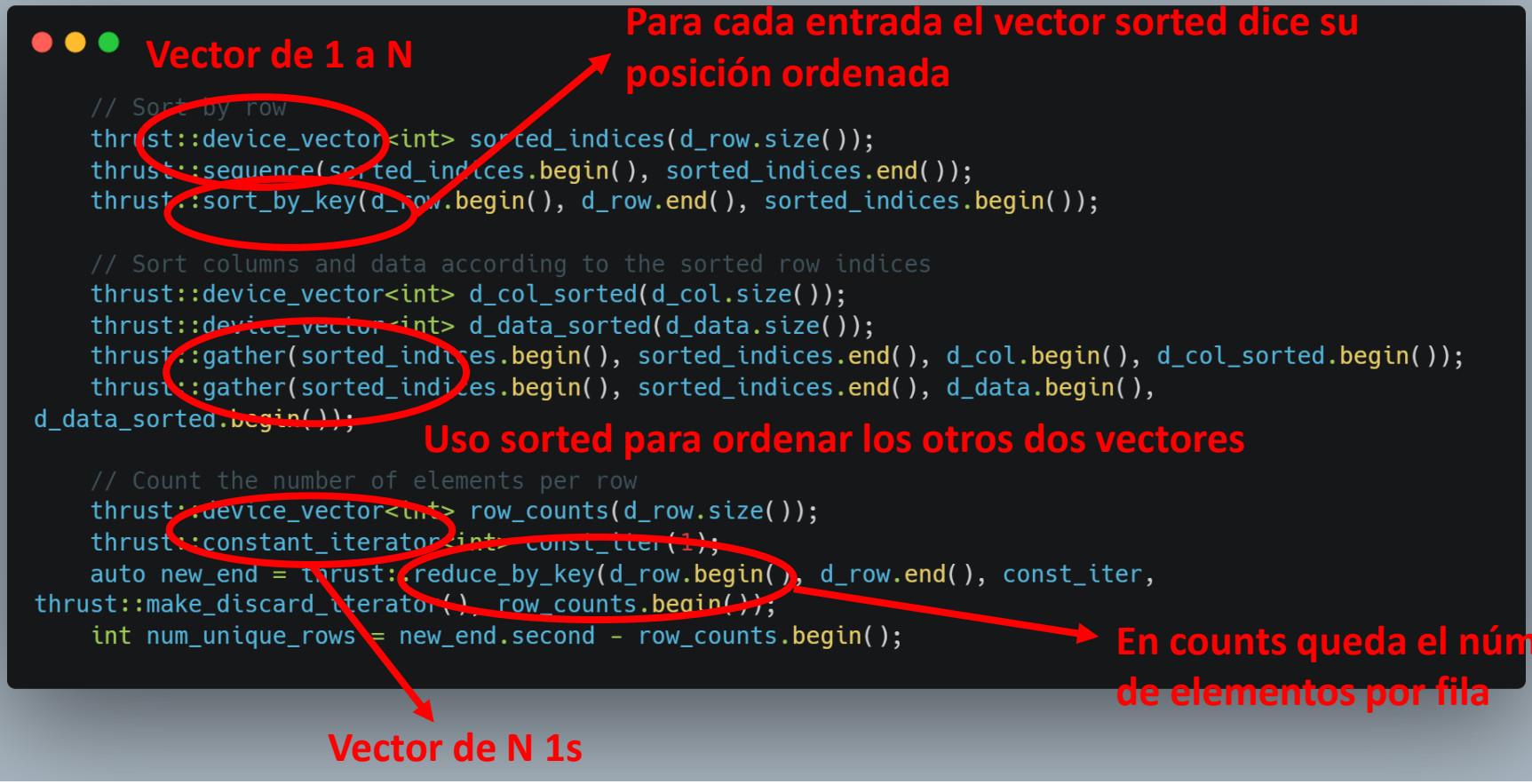
Vector de 1 a N

```
    // Sort by row
    thrust::device_vector<int> sorted_indices(d_row.size());
    thrust::sequence(sorted_indices.begin(), sorted_indices.end());
    thrust::sort_by_key(d_row.begin(), d_row.end(), sorted_indices.begin());

    // Sort columns and data according to the sorted row indices
    thrust::device_vector<int> d_col_sorted(d_col.size());
    thrust::device_vector<int> d_data_sorted(d_data.size());
    thrust::gather(sorted_indices.begin(), sorted_indices.end(), d_col.begin(), d_col_sorted.begin());
    thrust::gather(sorted_indices.begin(), sorted_indices.end(), d_data.begin(),
d_data_sorted.begin());

    // Count the number of elements per row
    thrust::device_vector<int> row_counts(d_row.size());
    thrust::constant_iterator<int> const_iter(1);
    auto new_end = thrust::reduce_by_key(d_row.begin(), d_row.end(), const_iter,
thrust::make_discard_iterator(), row_counts.begin());
    int num_unique_rows = new_end.second - row_counts.begin();
```

Uso sorted para ordenar los otros dos vectores

Vector de N 1s

Ejemplo 2 – Compresión de matrices



Ejemplo 2 – Compresión de matrices