

CUDA Parallel Operations

Advanced Aspects

Sergio Orts Escolano
Albert García García

sorts @ ua.es
agarcia @ dtic.ua.es

Contents

Histogram

Reduce

Scan

Histogram

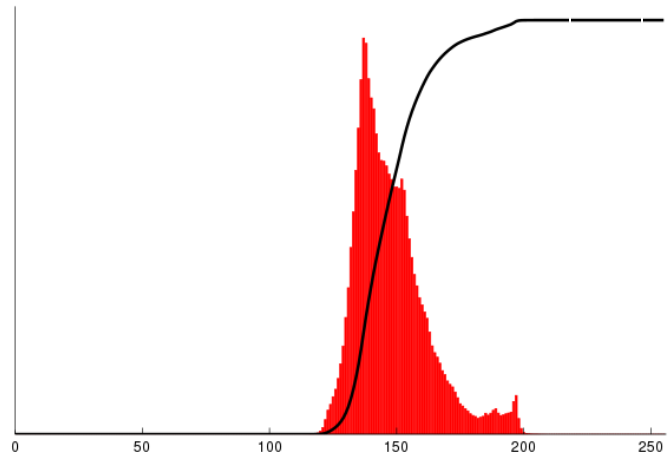
Histogram

Problem statement

- A histogram is an estimate of the probability distribution of a continuous variable (only one).
 - First, **bin** the range of values (divide the range of values into intervals).
 - Count how many values fall into each **bin** or interval.
- Use case: Histogram Equalization (“a method in image processing of contrast adjustment”).
 - Increase the global contrast of the image.
 - Spread intensity values over the histogram.

Histogram

Histogram equalization



Source: https://en.wikipedia.org/wiki/Histogram_equalization

Histogram

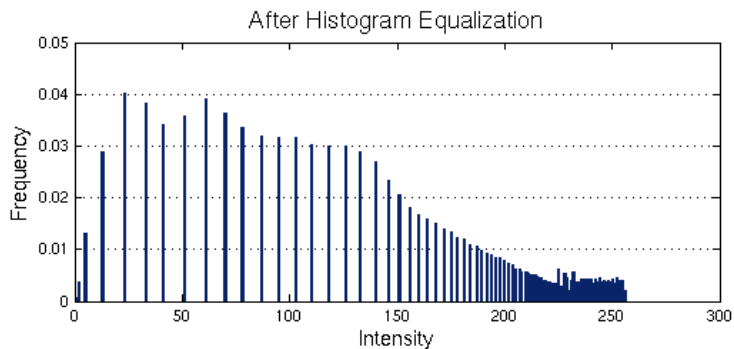
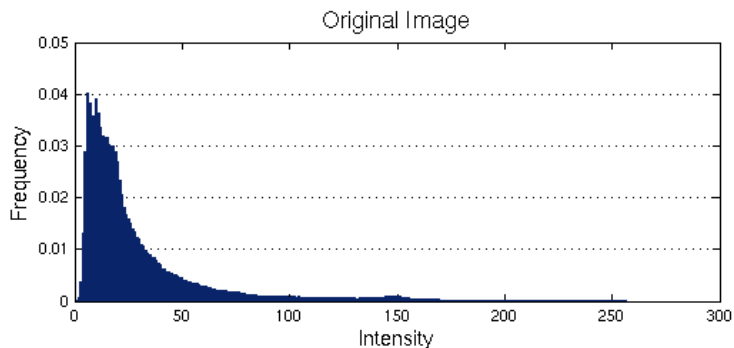
Histogram equalization



Source: https://en.wikipedia.org/wiki/Histogram_equalization

Histogram

Histogram equalization



Source: <http://www.cs.utah.edu/~jfishbau/improc/project2/>

Histogram

Histogram equalization pseudocode

```
hist = create histogram for input_image

cdf(0) = hist(0)
for i in range(1, len(hist))
    cdf(i) = cdf(i-1) + hist(i)
end

for p in input_image
    intensity = p
    p = cdf(intensity)
end
```


Histogram

Grayscale image histogram on CPU

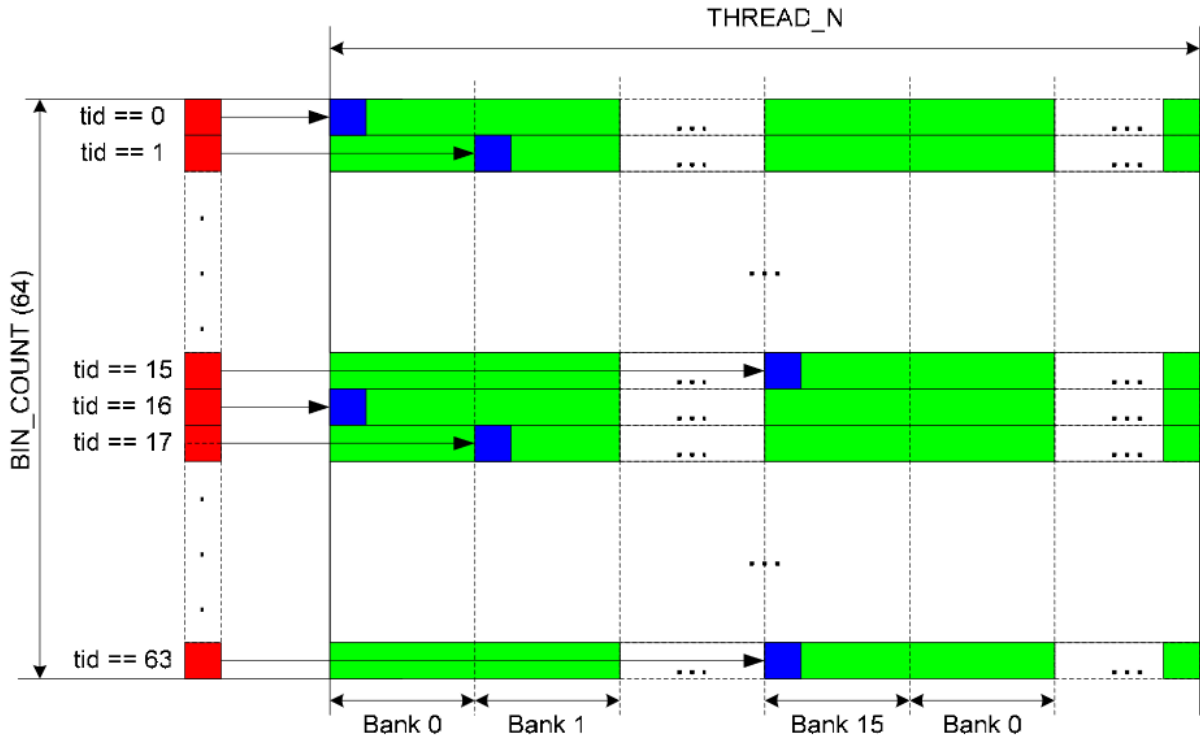
```
const int BIN_COUNT = 256;
int histogram[BIN_COUNT];

for (int i = 0; i < BIN_COUNT; ++i)
    histogram[i] = 0;

for (int x = 0; x < IMG_WIDTH; ++x)
    for (int y = 0; y < IMG_HEIGHT; ++y)
        histogram[image[y][x]]++;
```

Histogram

Grayscale image histogram on GPU



Histogram

Grayscale image histogram on GPU

```
__global__  
void comp_hist(char *image, int *hist)  
{  
    int idx_x = blockIdx.x * blockDim.x + threadIdx.x;  
    int idx_y = blockIdx.y * blockDim.y + threadIdx.y;  
  
    int idx = idx_y * IMG_WIDTH + idx_x;  
  
    if (idx < IMG_WIDTH * IMG_HEIGHT)  
    {  
        bin = image[idx];  
        atomicAdd(&(hist[bin]), 1)  
    }  
}
```

Reduce

Reduce

Problem statement and CPU implementation

- Summarize an input set of values into a single value by applying the same operation to all elements.
- Common examples: sum, maximum, minimum, multiplication...
- For instance: Vector Sum on CPU

```
const int N= 256;  
int x[N];  
int acc = 0;  
  
for (int i = 0; i < N; ++i)  
    acc += x[i]
```

N additions!

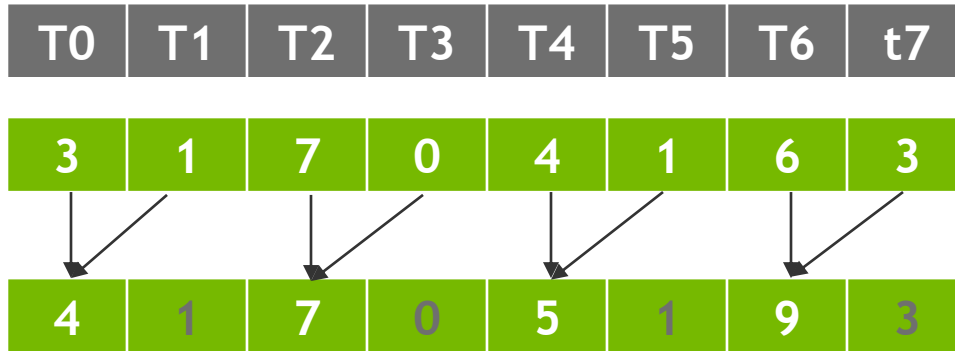
Reduce

Naïve GPU implementation

T0	T1	T2	T3	T4	T5	T6	t7
3	1	7	0	4	1	6	3

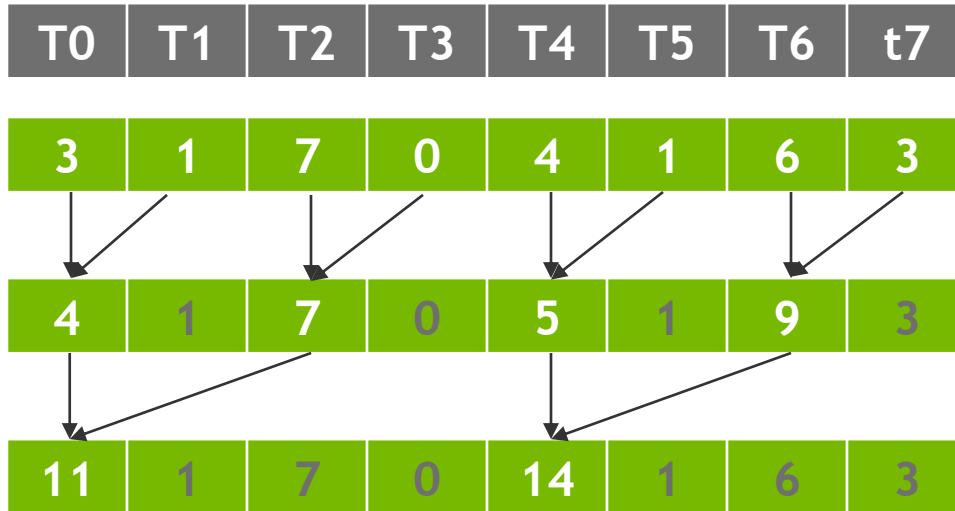
Reduce

Naïve GPU implementation



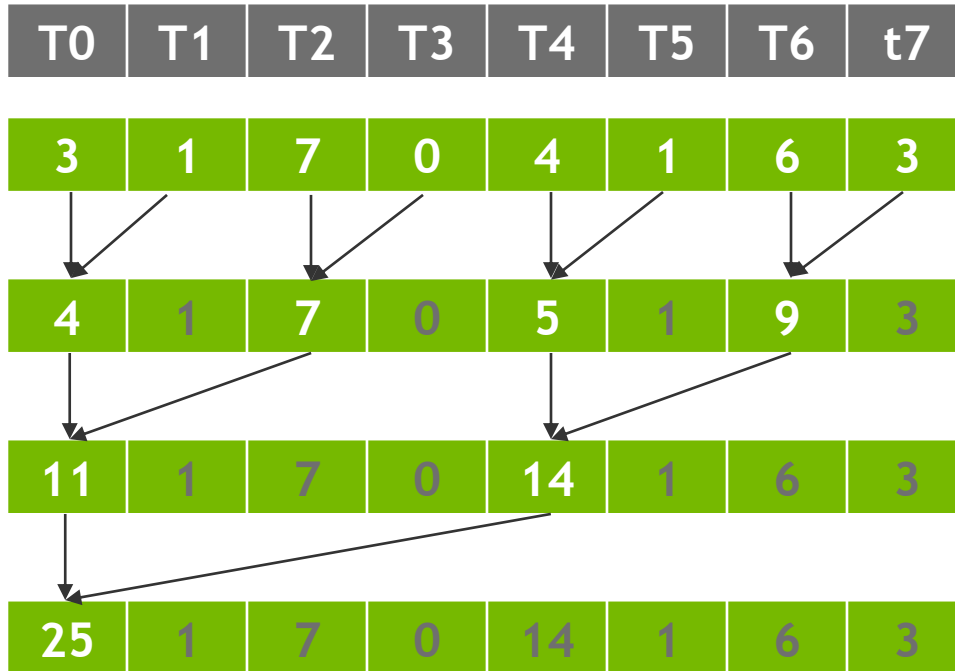
Reduce

Naïve GPU implementation



Reduce

Naïve GPU implementation



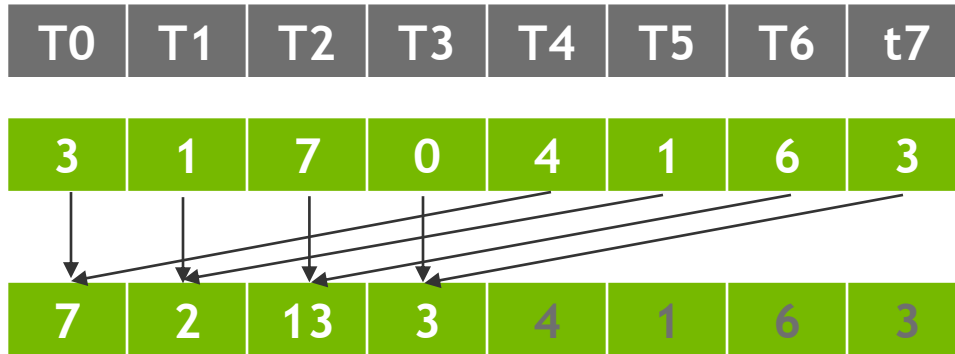
Reduce

Improved GPU implementation

T0	T1	T2	T3	T4	T5	T6	t7
3	1	7	0	4	1	6	3

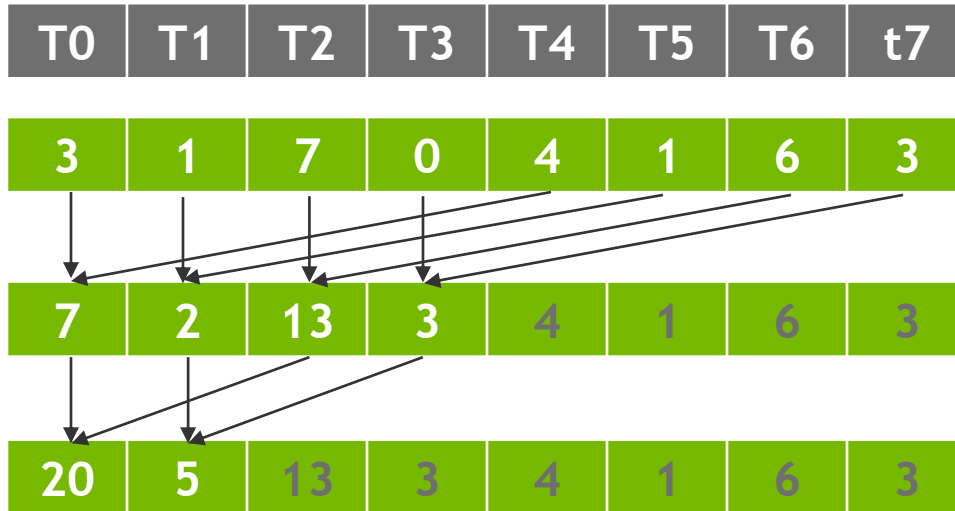
Reduce

Improved GPU implementation



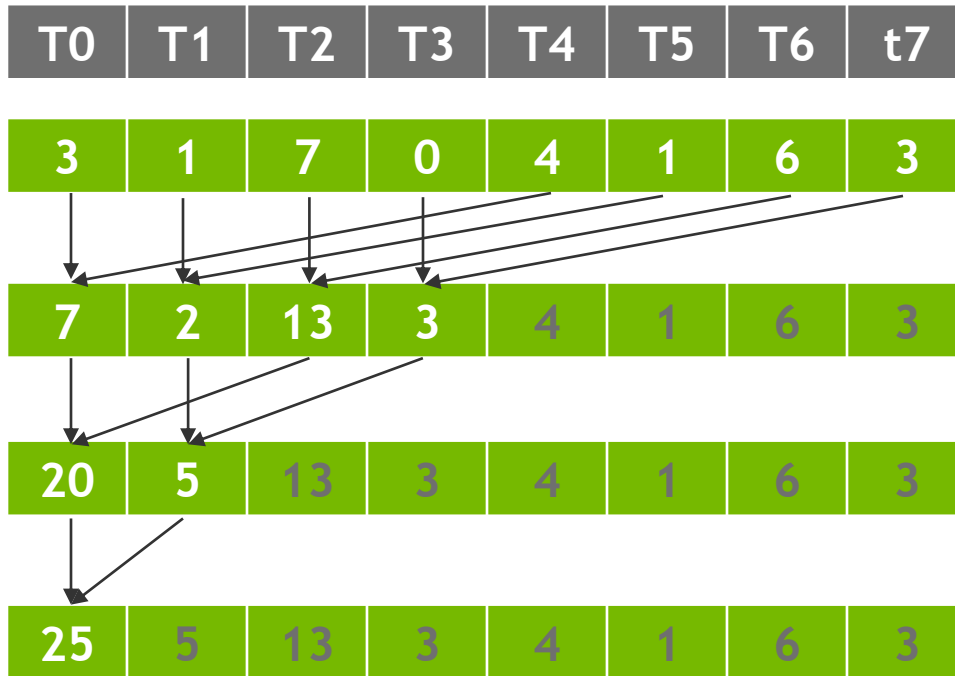
Reduce

Improved GPU implementation



Reduce

Improved GPU implementation



Reduce

More optimizations

- Shared Memory
- Warp Unrolling
- Multiple Elements per Thread
 - http://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86_website/projects/reduction/doc/reduction.pdf

Scan

Scan

Problem statement

- Also named prefix-sum, takes:
 - A binary associative operator \oplus like $+$, $*$, $/$... with identity I
 - An array of N elements: $[a_0, a_1, \dots, a_{N-1}]$
 - Returns the array:
 - $[I, a_0, (a_0 \oplus a_1), \dots, (a_0 \oplus a_1 \oplus \dots \oplus a_{N-2})]$ if exclusive
 - $[I, a_0, (a_0 \oplus a_1), \dots, (a_0 \oplus a_1 \oplus \dots \oplus a_{N-1})]$ if inclusive
- For instance:
 - For the array $[1, 2, 3, 4, 5, 6, 7, 8]$
 - Sum (+) scan operation.
 - Result is $[0, 1, 3, 6, 10, 15, 21, 28]$ for exclusive scan
 - Result is $[0, 1, 3, 6, 10, 15, 21, 28, 36]$ for inclusive scan

Scan

Scan (+) on CPU

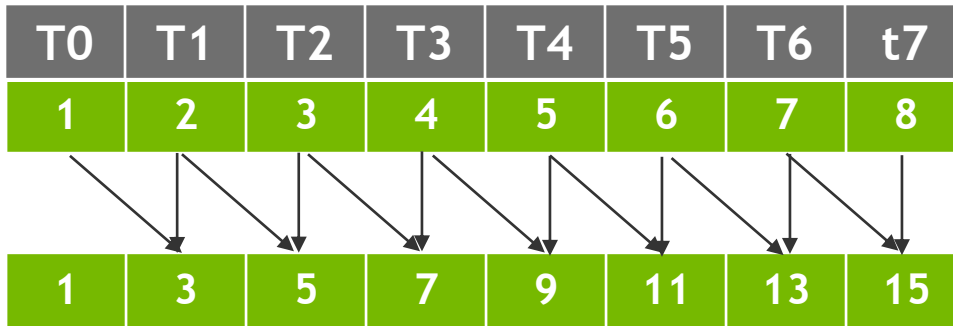
```
void scan(float *input, float *output, int N)
{
    output[0] = 0;

    for (int i = 1; i < N; ++i)
    {
        output[i] = input[i-1] + output[i-1]
    }
}
```

N-1 additions!

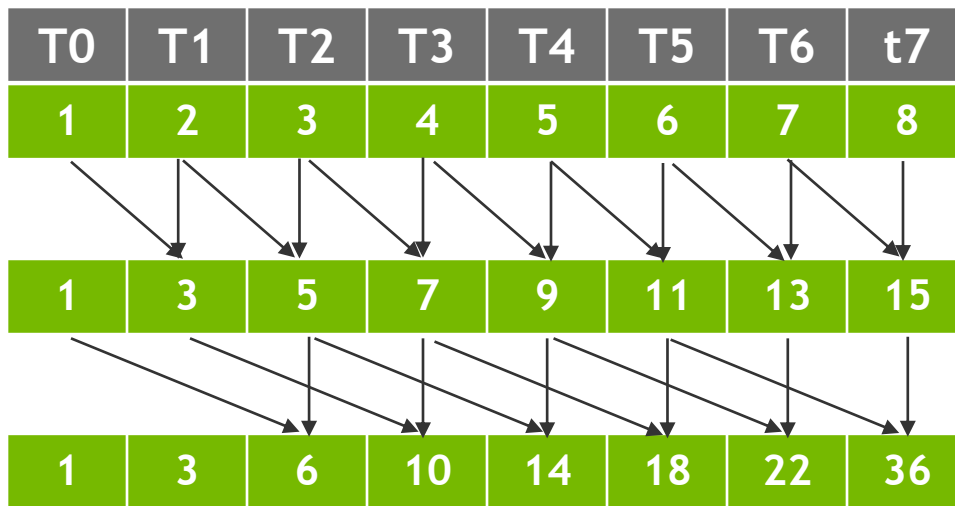
Scan

Naïve GPU implementation (Hillis-Steele)



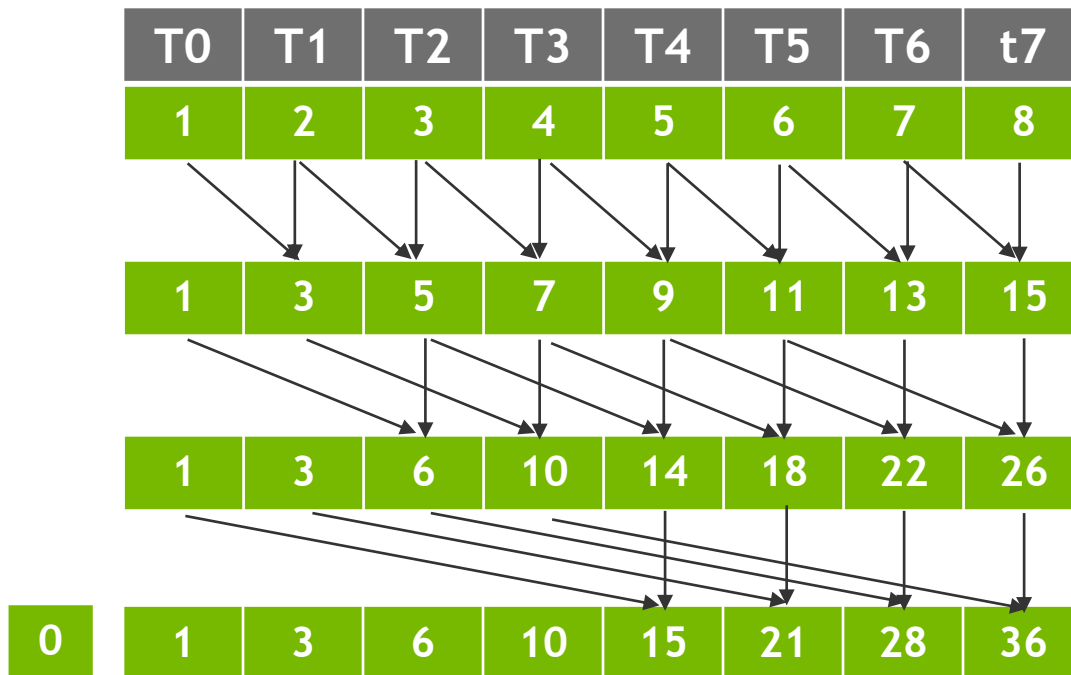
Scan

Naïve GPU implementation (Hillis-Steele)



Scan

Naïve GPU implementation (Hillis-Steele)



And more!

And more!

Other GPU parallel operations

- Radix Sort: <https://www.youtube.com/watch?v=dPwAA7j-8o4>
- Warp-level Primitives: <https://devblogs.nvidia.com/using-cuda-warp-level-primitives/>
- Merge Sort: <https://onezork.wordpress.com/2014/08/29/gpu-mergesort/>

CUDA Parallel Operations

Advanced Aspects

Thanks for your attention!

These slides have been modified/remixed using the TeachingKit licensed by NVIDIA and the University of Illinois under the [Creative Commons Attribution-NonCommercial 4.0 International License](https://creativecommons.org/licenses/by-nc/4.0/).



Sergio Orts Escolano
Albert García García

sorts @ ua.es
agarcia @ dtic.ua.es