Parallel Algorithms: Map - Reduction

Tran Giang Son, tran-giang.son@usth.edu.vn

ICT Department, USTH

Introduction

What?

- Application: needs of algorithm sets

What?

- Application: needs of algorithm sets
- Design pattern
  - Representation of a common programming problem
  - Tested, efficient solution
  - Can/should be reused

What?

- Application: needs of algorithm sets
- Design pattern
  - Representation of a common programming problem
  - Tested, efficient solution
  - Can/should be reused
- Parallel pattern
  - Design patterns with parallelization in mind
  - A set of building blocks for parallel algorithms

What?

- Map
- Reduction
- Gather
- Scatter
- Partition
- Scan
- Pack

Why?

- Improve productivity of experts
  - Focus on high level algorithms
- Guide relatively inexperienced users
  - Like a cookbook
- Software reusability and modularity

Map

What?

- Simple operation that applies to all element in an array
- Doesn't care about neighbour
- Best for embarrassingly parallel problems
- Can achieve near linear speedup

What?

- Given
  - Array of data elements A
  - Function $f(x)$

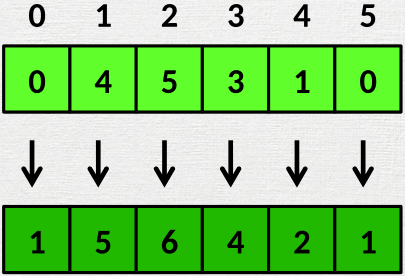
$$map(A, f) = [f(x_i), \forall x_i \in A]$$

What?

- One to one relationship between input and output
- Bidirectional relationship
- Every input location has a corresponding output location, and
- Every output location has a corresponding input location

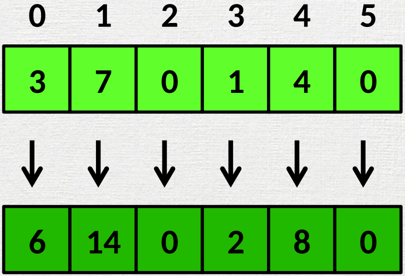
What?

Example: Add 1 to every element in the array



What?

Example: Double every element in the array



What?

Extension: N-ary map

	0	1	2	3	4	5	6	7	8	9	10	11
x	3	7	0	1	4	0	0	4	5	3	1	0
y	2	4	2	1	8	3	9	5	5	1	2	1
result	5	11	2	2	12	3	9	9	10	4	3	1

How?

- Straightforward
  - `kernel`, A is a 1D array
  - Map function `f()`

```
def kernel(src, dst):
    tid = ...
    dst[tid] = f(src[tid])
```

Labwork 6: Map

- Implement labwork 6a: grayscale image binarization
- Implement labwork 6b: brightness control
- Implement labwork 6c: blending two images
- Write a report (in LaTeX)
  - Name it « Report.6.map.tex »
  - Explain how you implement the labworks
  - Try experimenting with different 2D block size values
- Push the report and your code to your forked repository

Extra 6a: Binarization

- Converting a grayscale pixel to a binary value, i.e. 0 or 1
- Easiest method: thresholding
  - Define a threshold $\tau \in [0..255]$
  - Intensity of pixel $\Phi(x, y)$

$$b(x, y) = \begin{cases} 0 & \Phi(x, y) < \tau \\ 1 & \Phi(x, y) \geq \tau \end{cases}$$

- You can handle an extra parameter from the command line for threshold using `argv`

Extra 6b: Brightness control

- Brightness of each pixel is represented by its value
- Increase brightness: increase all pixel values
- Reduce brightness: decrease all pixel values

Extra 6c: Blending images

- Combines two images $\Phi_1(x, y)$ and $\Phi_2(x, y)$ of the same size into one output $Q(x, y)$
- Coefficient c defines the "weight" of each image

$$Q(x, y) = c \times \Phi_1(x, y) + (1 - c) \times \Phi_2(x, y)$$

- You can handle an extra parameter from the command line for second image using `argv`
- Can work on both grayscale or RGB images (individual channels for RGB)

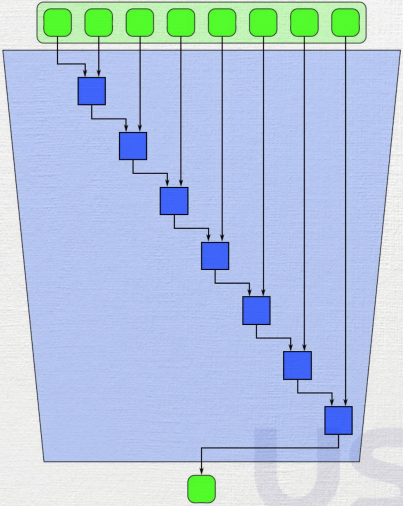
Reduction

What?

- A set of two-to-one associative operators
- Combines all elements in an array
- Produces an output from this combination
- Example
  - Sum of all elements in an array
  - Product of all elements in an array
  - Getting max/min value of an array

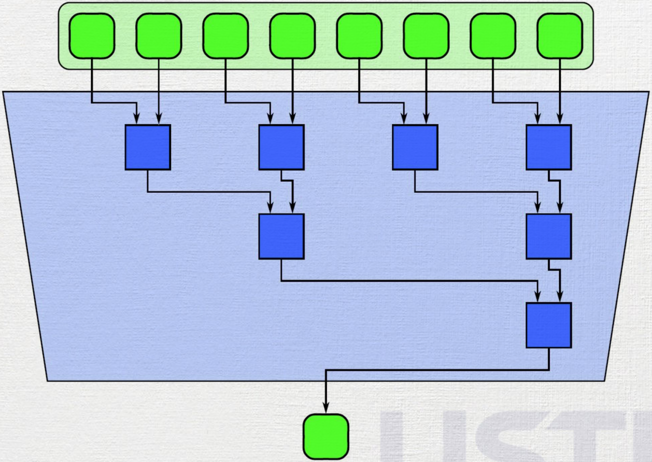
What?

Serial reduction



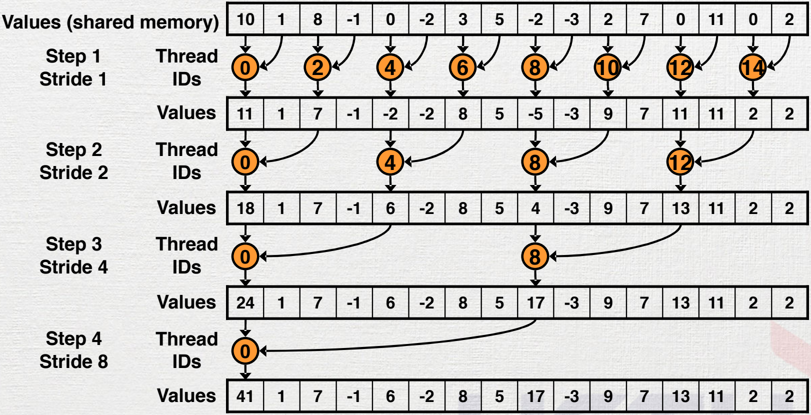
What?

Parallel reduction

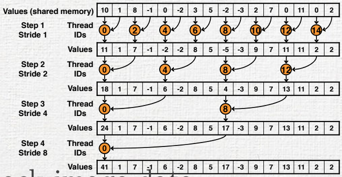


What?

Parallel reduction: Sum example



How?



- Use shared memory to "cache" block image data
- Loop for each step
  - Use only one thread (even `tid`) for each pair
  - Perform sum, store to the cache
  - Wait until all threads to finish the sum
- Write result from cache to global memory

How?

```
@cuda.jit
def sum1D(src, dst, sharedSize):
    # shared memory declaration for caching block content
    cache = cuda.shared.array((sharedSize, ), np.float32)
    localtid = threadIdx.x
    tid = threadIdx.x + blockIdx.x * blockDim.x
    # copy local block from src to cache (in shared memory)
    cache[localtid] = src[tid]
    cuda.syncthreads()
    # reduction in cache
    s = 1
    while s < blockDim.x:
        if localtid % (s * 2) == 0:
            cache[tid] += cache[tid + s]
        cuda.syncthreads()
        s = s * 2
    # only first thread writes back to dst
    if localtid == 0: dst[blockIdx.x] = cache[0]
```

How?

It works, but...

How?

It works, but…

Problem?

How?

```
# reduction in cache
s = 1
while s < blockDim.x:
    if localtid % (s * 2) == 0:
        cache[tid] += cache[tid + s]
    cuda.syncthreads()
    s = s * 2
```

How?

```
# reduction in cache
s = 1
while s < blockDim.x:
    if localtid % (s * 2) == 0:      # <-- branch diversion
        cache[tid] += cache[tid + s]
    cuda.syncthreads()
    s = s * 2
```

Let's speed it up.

Optimize: branch diversion

```
# reduction in cache
s = 1
while s < blockDim.x:
    if localtid % (s * 2) == 0:      # <-- branch diversion
        cache[tid] += cache[tid + s]
    cuda.syncthreads()
    s = s * 2
```

Optimize: branch diversion

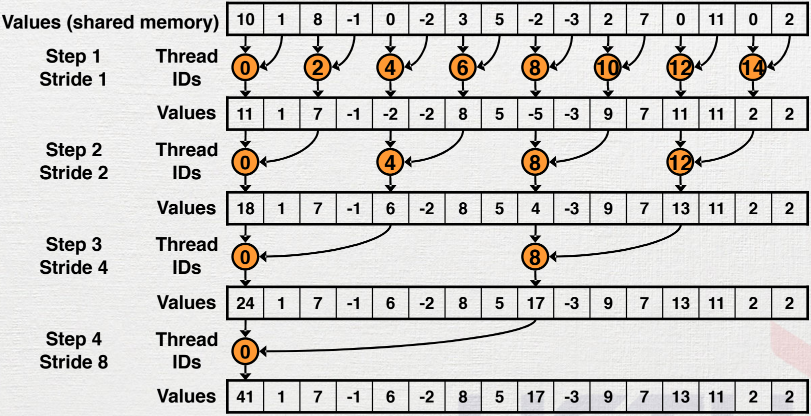
```
# reduction in cache
s = 1
while s < blockDim.x:
    if localtid % (s * 2) == 0:      # <-- branch diversion
        cache[tid] += cache[tid + s]
    cuda.syncthreads()
    s = s * 2
```

$$\Downarrow$$

```
s = 1
while s < blockDim.x:
    index = s * 2 * localtid
    if index < blockDim.x:      # <-- no branch diversion
        cache[index] += cache[index + s]
    cuda.syncthreads()
    s = s * 2
```

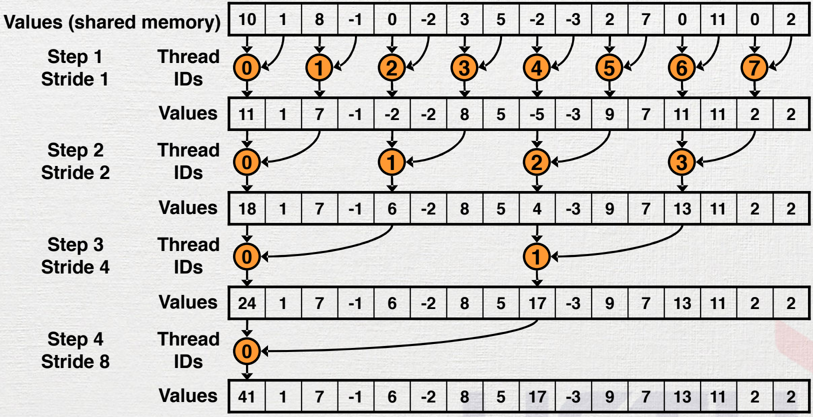
Optimize: branch diversion

Before, with branch diversion



Optimize: branch diversion

After, without branch diversion



Optimize

Problem?

Optimize

Problem?

Shared memory bank conflicts

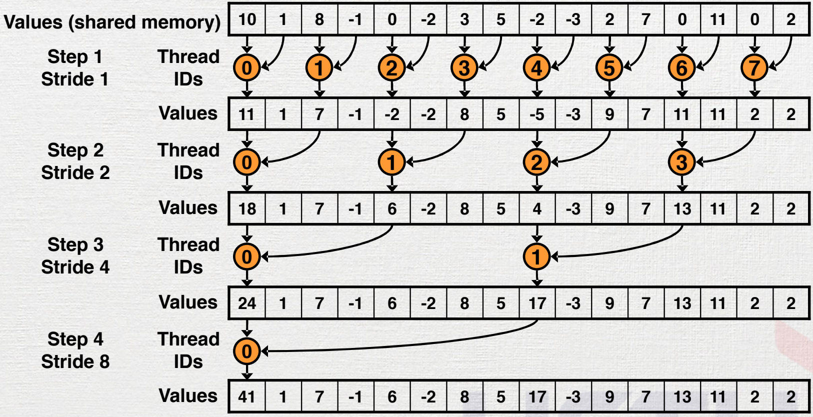
- Shared memory is divided into banks
- Each bank can address one data at a time per warp
- Load/store data from/to the same bank has to wait
- Similar to hostok memory Dual-Triple-Quad Channels

Optimize: memory bank conflicts

- Particularly, for read operations of first step: two cycles
  - #0 #2 #4 #6 #8 #10 #12 #14
  - #1 #3 #5 #7 #9 #11 #13 #15
- Can be avoided by
  - Making read access to two banks
  - #0#8 #1#9 #2#10...
  - Compacting write operations to one bank
  - #0 #1 #2 #3 #4 #5 #6 #7

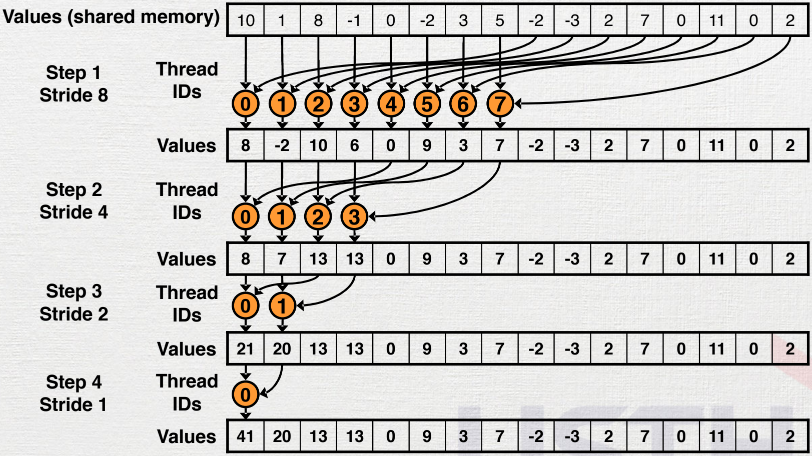
Optimize: memory bank conflicts

Before, with bank conflicts:



Optimize: memory bank conflicts

After, without bank conflicts



Optimize: memory bank conflicts

```
s = int(blockDim.x / 2)
while s > 0:
    if (localtid < s):
        cache[localtid] += cache[localtid + s]
    cuda.syncthreads()
    s = s / 2
```

Optimize

Problem?

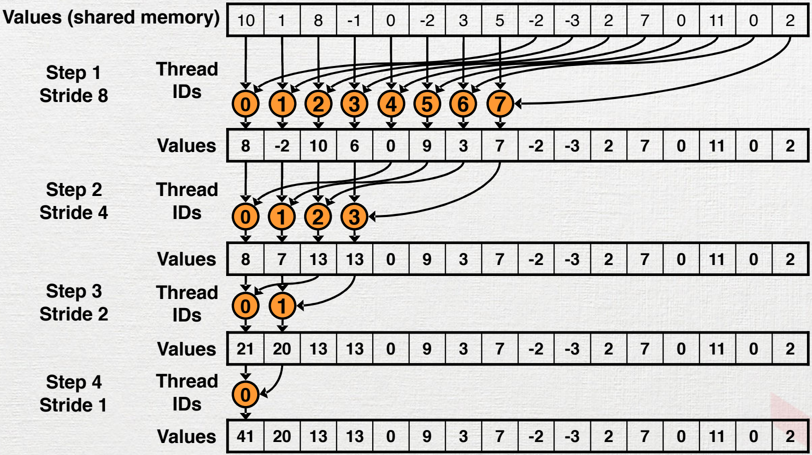
```
s = int(blockDim.x / 2)
while s > 0:
    if (localtid < s):
        cache[localtid] += cache[localtid + s]
    cuda.syncthreads()
    s = s / 2
```

Optimize

Problem?

```
s = int(blockDim.x / 2)
while s > 0:
    if (localtid < s):      # <-- 1st step : idle threads
        cache[localtid] += cache[localtid + s]
    cuda.syncthreads()
    s = s / 2
```

Optimize: Idle threads



TID 8-15 are not doing anything.

Optimize: Idle threads

- Reduce block size
  - Smaller block size ⇒ more blocks ⇒ more perf.
- Precompute first reduction step before putting into cache

Optimize: Idle threads

- Reduce block size

```
tid = blockIdx.x * blockDim.x + threadIdx.x
```

$$\Downarrow$$

```
tid = blockIdx.x * blockDim.x * 2 + threadIdx.x
```

Don't forget to reduce block size in kernel launch.

Optimize: Idle threads

- Precompute first reduction step

```
cache[localtid] = src[tid]
```

$$\Downarrow$$

```
cache[localtid] = src[tid] + src[tid + blockDim.x]
```

Optimize: Final

```python
# shared memory declaration for caching block content
cache = cuda.shared.array((sharedSize, ), np.float32)
localtid = threadIdx.x
tid = threadIdx.x + blockIdx.x * blockDim.x
# copy local block from src to cache (in shared memory)
cache[localtid] = src[tid] + src[tid + blockDim.x]
cuda.syncthreads()
# reduction in cache
s = int(blockDim.x / 2)
while s > 0:
    if (localtid < s):
        cache[localtid] += cache[localtid + s]
    cuda.syncthreads()
    s = s / 2
# only first thread writes back to dst
if localtid == 0: dst[blockIdx.x] = cache[0]
```

Optimize

More possible optimization

- Loop unrolling
  - Manually
  - With templates

Recap

- Several iterations of binary associative operators
- Combines all elements in an array to produce an output
- Performance optimization
  - Branch divergence
  - Memory bank conflicts
  - Better block size

Labwork 7: Reduction

- Implement labwork 7: grayscale stretch
- Write a report (in LaTeX)
    - Name it « Report.7.reduce.tex »
    - Explain how you implement the labworks
    - Explain and measure speedup, if you have performance optimizations
    - Try experimenting with different 2D block size values
- Push the report and your code to your forked repository

Extra: Grayscale Stretch

- 3 steps:
  - Convert image to gray (MAP)
  - Find max/min intensity of image (REDUCE)
  - Linearly recalculate intensity for each pixel (MAP)
    - From $[min, max]$ to $[0, 255]$

$$g' = \frac{g - min}{max - min} \times 255$$