

Lab4 CUDA Advance

Nov, 2021 Parallel Programming

Overview

- ❖ Review
- ❖ Coalesced Memory Access
- ❖ Lower Precision
- ❖ Shared Memory
- ❖ Lab4

Review

- ❖ In last lab, some of people paralleled the y-axis

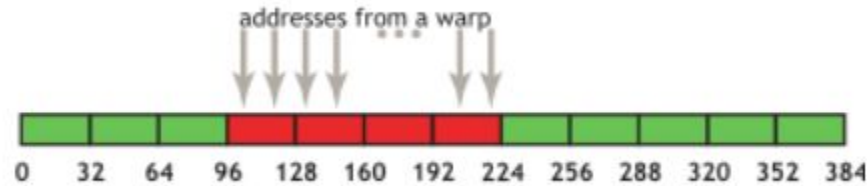
```
int y = blockIdx.x * blockDim.x + threadIdx.x;  
if (y >= height) return;  
  
// for (int y = 0; y < height; ++y) {
```

- ❖ Launch $\text{height} / \text{num_threads} + 1$ blocks

```
// decide to use how many blocks and threads  
const int num_threads = 256;  
const int num_blocks = height / num_threads + 1;  
  
// launch cuda kernel  
sobel << <num_blocks, num_threads>>> (dsrc, ddst, height, width, channels);
```

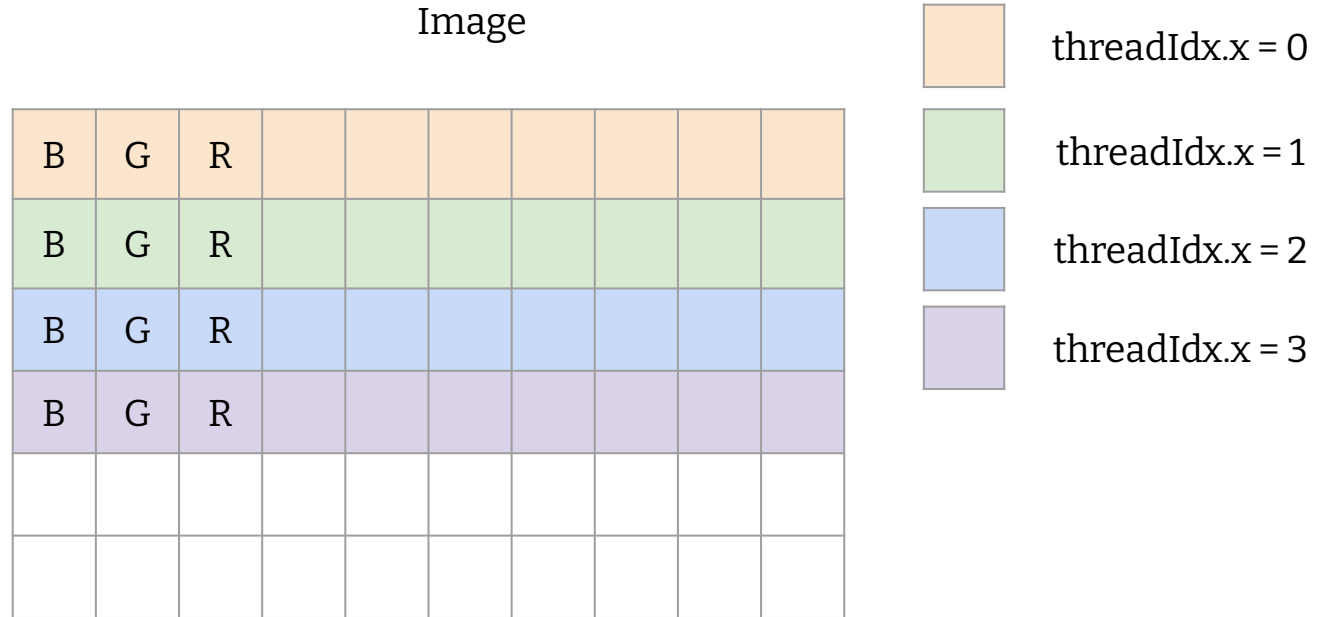
Coalesced Memory Access

- ❖ In short,
 - Concurrent memory accesses in a warp are continuous



- ❖ Why
 - GPU has L2 (32 bytes), L1 (128 bytes) cache
 - If memory accesses in a warp are continuous, it can utilize the cache
- ❖ Details
 - [CUDA Best Practices](#)

Problem of Paralleling Y-axis



Better Access Pattern

- ❖ In thread level, we should parallel x-axis
 - Different with CPU
- ❖ How to parallel y-axis and x-axis
 - Use block to parallel y
 - Launch 2D block
 - Combine both

Block & Threads

```
// decide to use how many blocks and threads
const int num_threads = 256;
const int num_blocks = 2048;

// launch cuda kernel
sobel << <num_blocks, num_threads>>> (dsrc, ddst, height, width, channels);
```

```
for (int y = blockIdx.x; y < height; y += blockDim.x) {
    for (int x = threadIdx.x; x < width; x += blockDim.x) {
        /* Z axis of filter */
```

2D Block

```
// Dim3 var(number of x, number of y)
dim3 num_threads(128, 4);
const int num_blocks = 2048;

// launch cuda kernel
sobel << <num_blocks, num_threads>>> (dsrc, ddst, height, width, channels);
```

```
for (int y = blockIdx.x * blockDim.y + threadIdx.y; y < height; y += gridDim.x * blockDim.y) {
    for (int x = threadIdx.x; x < width; x += blockDim.x) {
```


2D Block & 2D threads

```
// Dim3 var(number of x, number of y)
dim3 num_threads(128, 4);
dim3 num_blocks(1, 2048);
```

Practice x, y index by yourself

Coalesced Memory Access

Image

B	G	R	B	G	R	B	G	R	B	G	R



threadIdx.x = 0



threadIdx.x = 1



threadIdx.x = 2



threadIdx.x = 3

3 bytes * 32 threads = 96
bytes

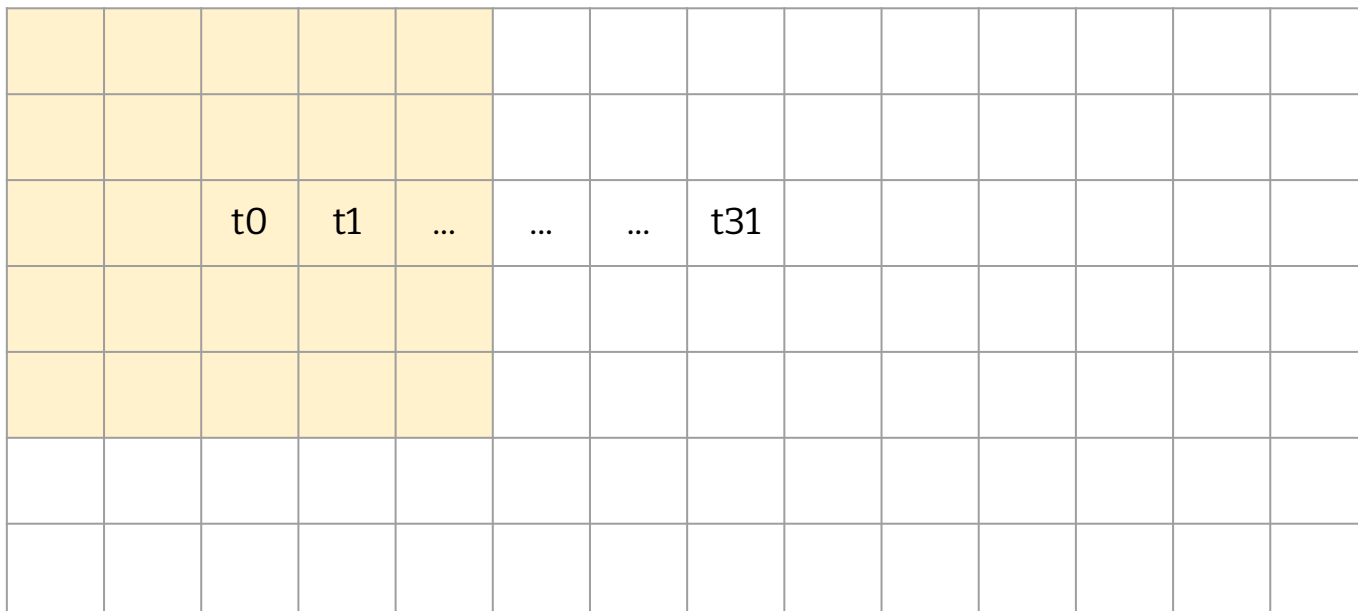
Mixed-Precision

- ❖ Use lower precision when available
- ❖ Use float to replace double
- ❖ Use fp16 to replace float
- ❖ Make sure using lower precision does not affect the results

Shared Memory

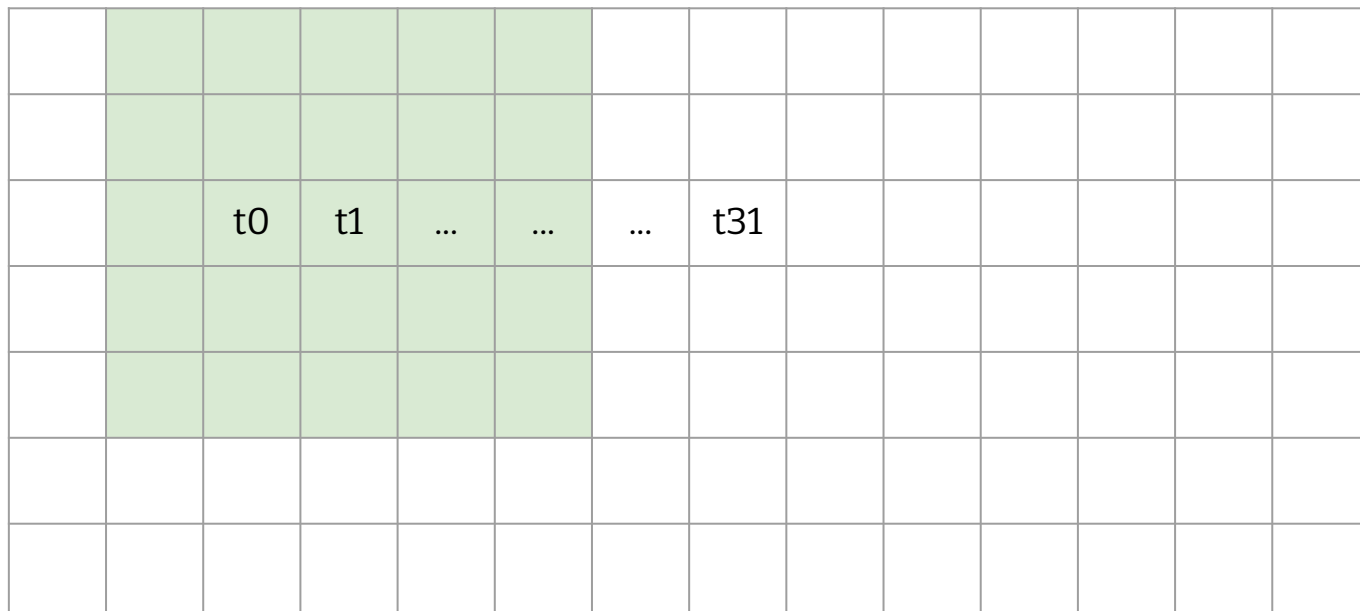
- ❖ Shared memory is powerful when the data has locality
- ❖ Convolution is a good case

Shared Memory with Sobel



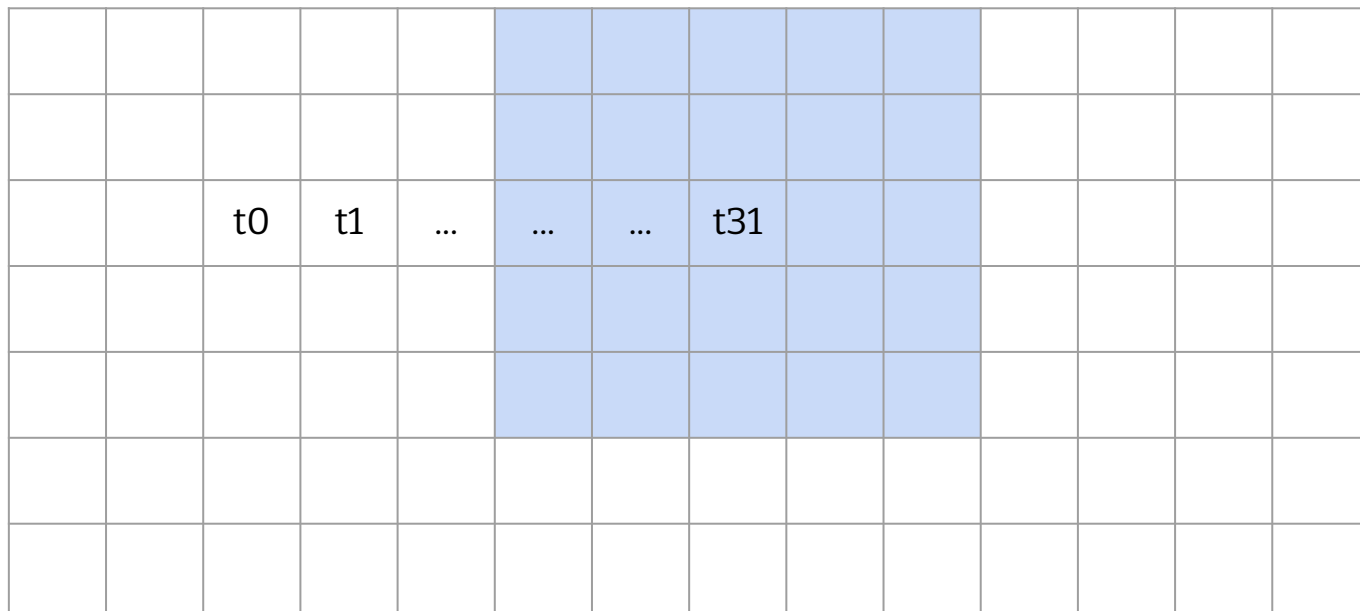
Required data by t_0

Shared Memory with Sobel



Required data by t1

Shared Memory with Sobel



Required data by t31

Using Shared Memory in Sobel

- ❖ Move the required data into shared memory
- ❖ Compute
- ❖ Update shared memory

[illegible]

Lab4

- ❖ Optimize the sobel CUDA implementation
- ❖ TAs provide simple CUDA version
 - You are asked to accelerate it over **13x**
 - Materials are under `/home/pp21/share/lab4`
- ❖ Name your kernel as “sobel”
- ❖ We accept little pixel errors

Submission

- ❖ Finish it before 12/2 23:59
- ❖ Submit your code and Makefile (optional) to eeclass
- ❖ You can use **lab4-judge** for pre-check