Lab4 CUDA Advance

Nov, 2022 Parallel Programming

Overview

- Further optimize the code with following techniques:
  - Coalesced Memory Access
  - Lower Precision
  - Shared Memory
- Lab4

Review

- In last lab, some of people parallelled the y-axis

```
int y = blockIdx.x * blockDim.x + threadIdx.x;
if (y >= height) return;

// for (int y = 0; y < height; ++y) {
```

- Launch height / 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);
```

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 += gridDim.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) {
```

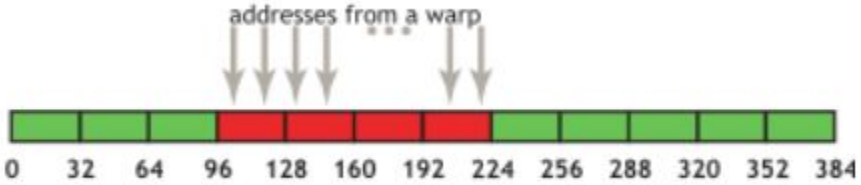
2D Block & 2D threads

```
// 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

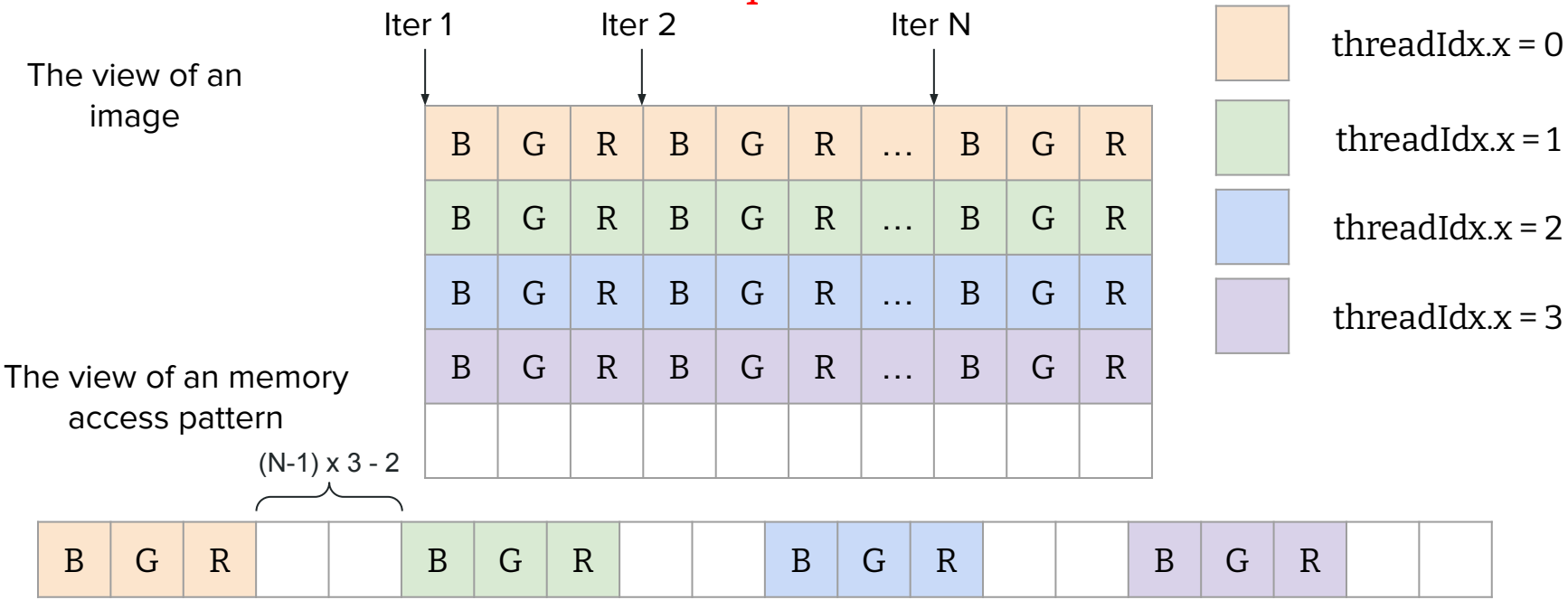
- In short,
  - Concurrent memory accesses in a warp are continuous



- 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

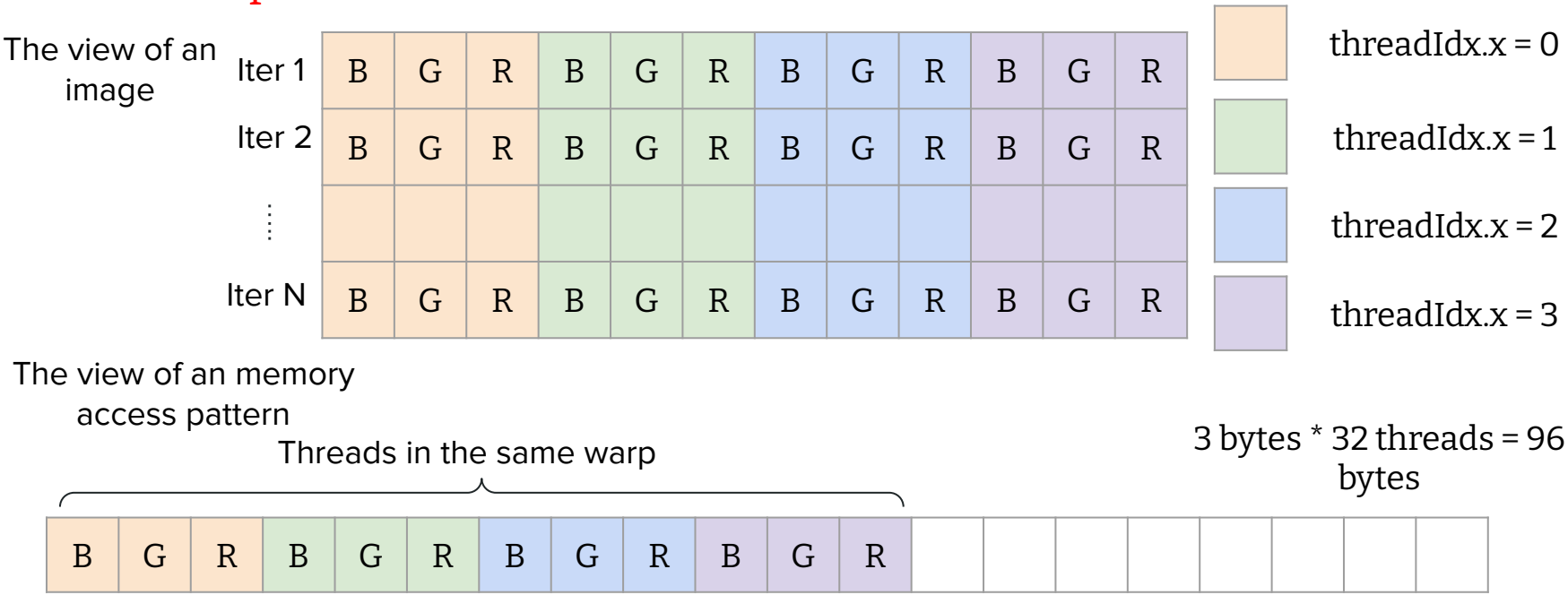
Access without Coalesced Memory

- If each thread compute a single row -> Failed to combine the locations with different colors into one request



Coalesced Memory Access

- The accesses can be combined into a single request if we change the access pattern



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

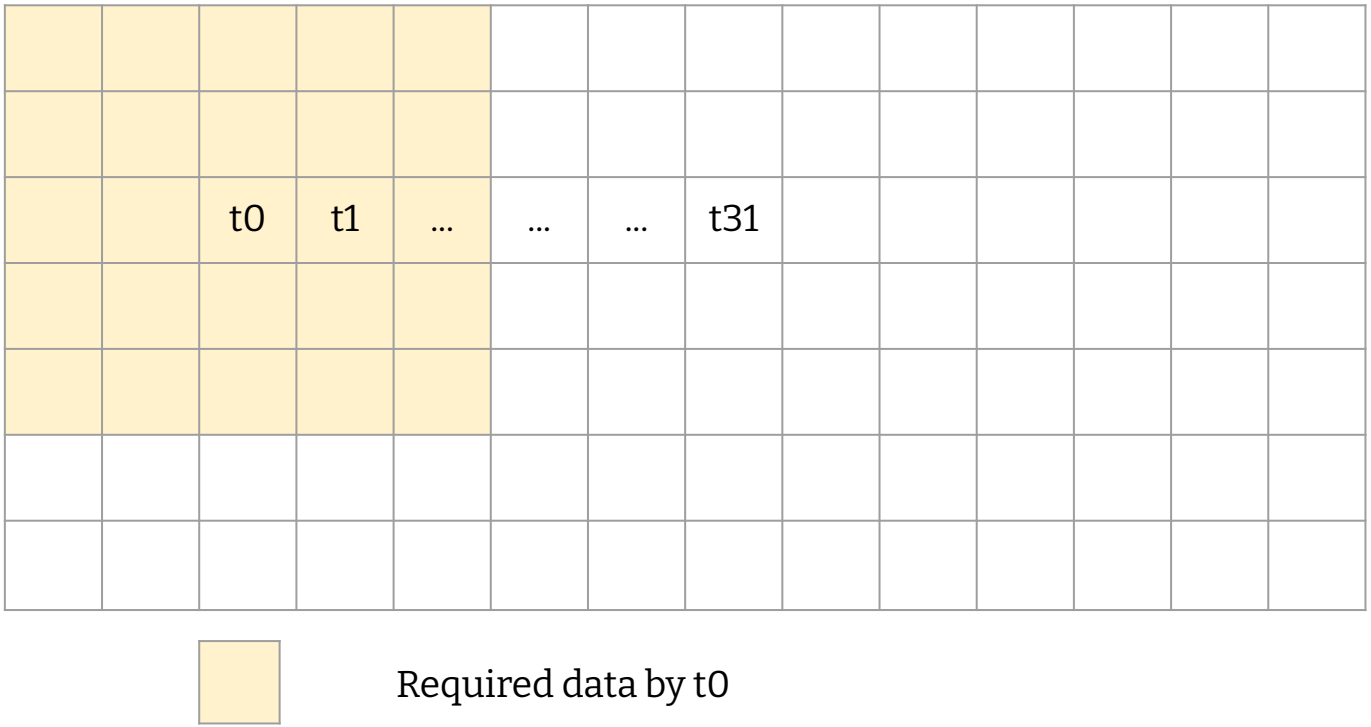
Mixed-Precision

- 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

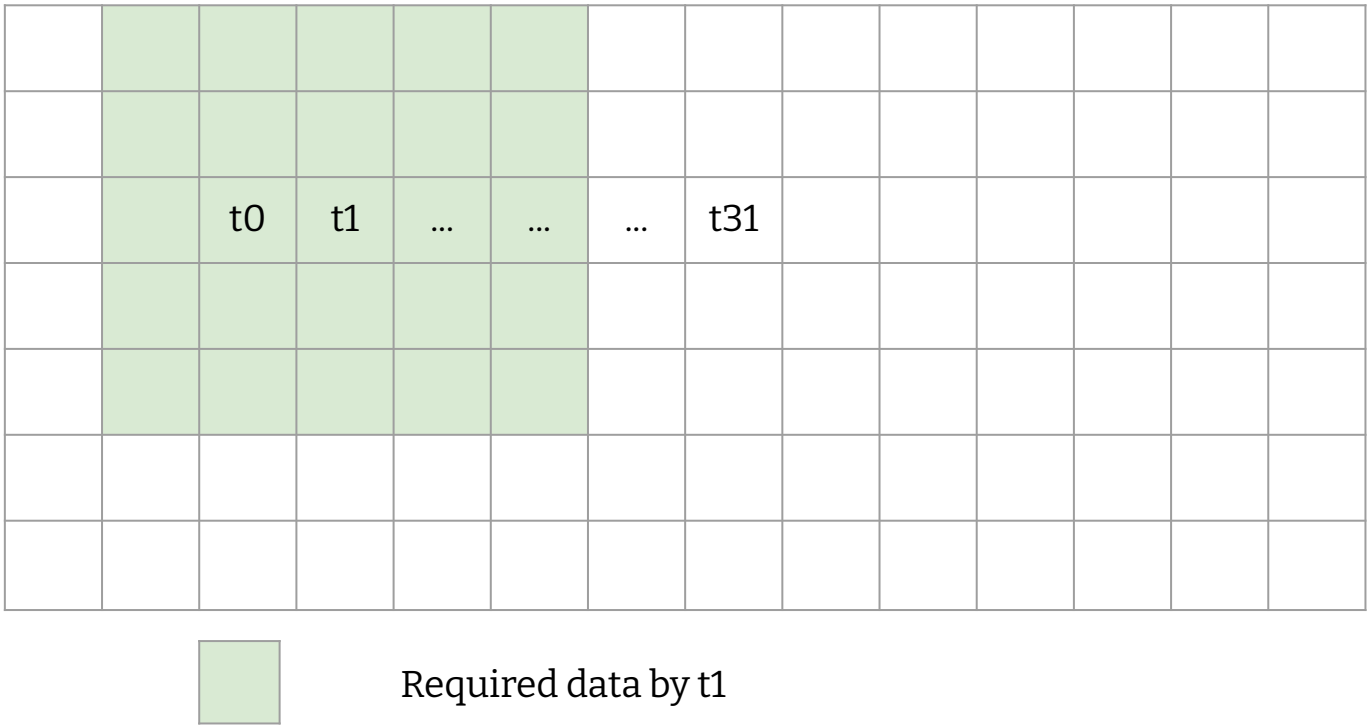
Shared Memory

- Shared memory can greatly reduce the access time of a reused data item
- Find the reuse data from your code and move it to shared memory before accessing

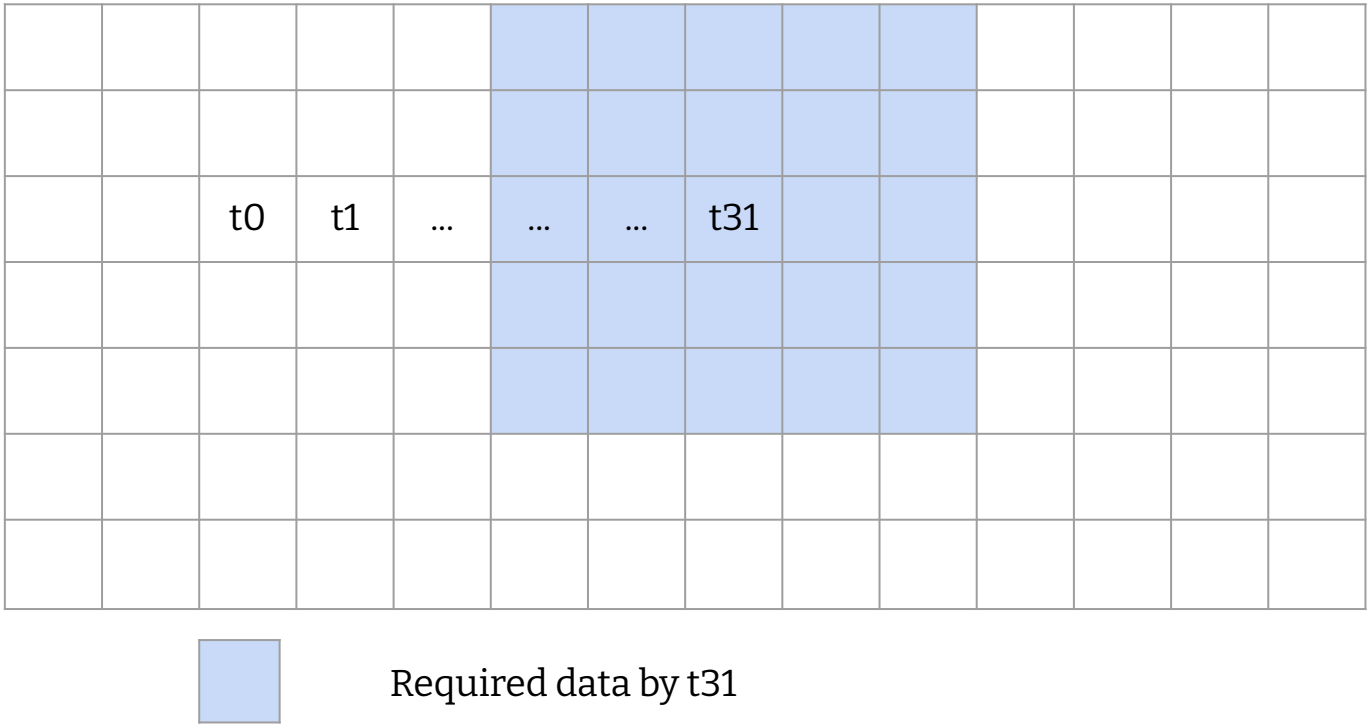
Shared Memory with Sobel



Shared Memory with Sobel

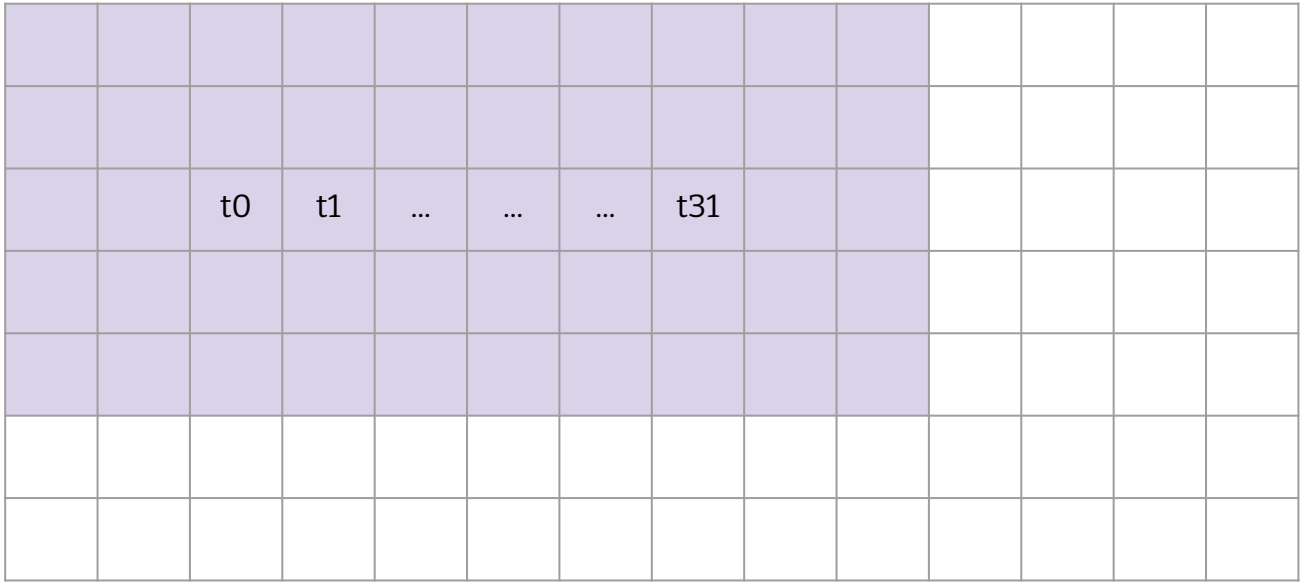


Shared Memory with Sobel



Using Shared Memory in Sobel

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



Lab4

- Optimize the sobel CUDA implementation
- TAs provide simple CUDA version
  - You are asked to accelerate it over **13x**
  - Materials are under **/home/pp22/share/lab4**
- Name your kernel as “sobel”
- We accept little pixel errors

Submission

- Finish it before 12/1 23:59
- Submit your code and Makefile (optional) to eeclass
- You can use **lab4-judge** for pre-check