Lec13 Shared Memory Case Study 1

Yun Tian (Tony) (Ph.D)

CSCD 439/539 GPU Computing

Last Lecture

- Different types of GPU memory
- __syncthreads() for threads within blocks.

Outline for Today

- Case study of using shared memory
  - diff() function that uses shared memory

Simple diff()

```
// CUDA kernel. Each thread takes care of one
element
__global__ void diffKernel( float *in, float *out,
int n )
{
   // Get our global thread ID
   int id = blockIdx.x * blockDim.x + threadIdx.x;

   // Make sure we do not go out of bounds
   if (id > 0 && id < n)
      out[id] = in[id] - in[id - 1];
}
```

Simple diff()

- How many time is element in[i] loaded by thread(s)?

two times. one by thread i, another by thread i+1. Which means one global memory read is redundant.

Simple diff()

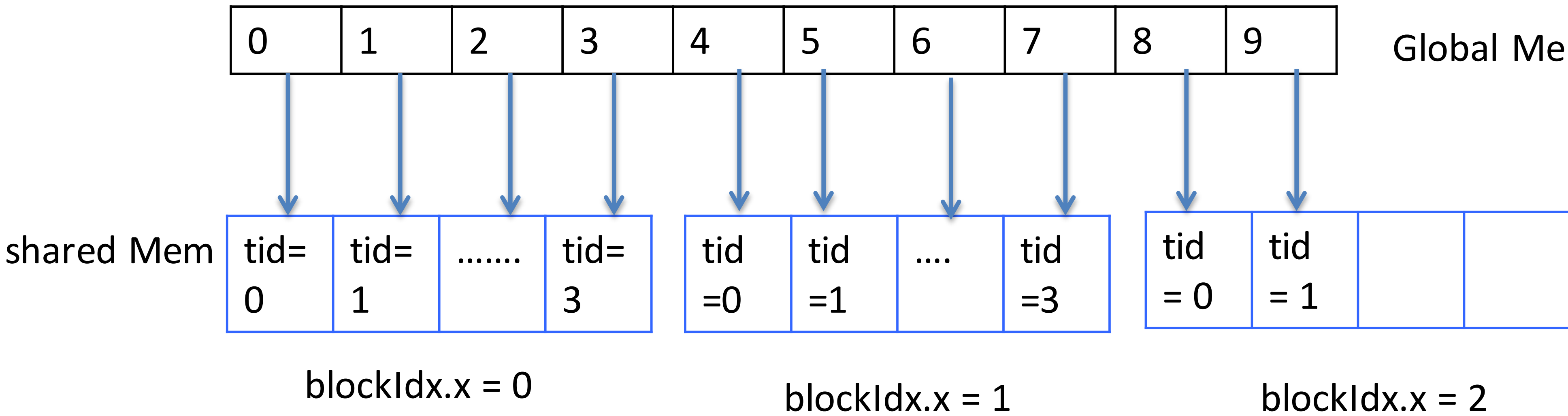
- As we analyzed in Lab2, the simple kernel in previous slide has weakness,
  - Each thread in a block has to make two global memory access operations.
  - Global memory access is more than 100 times slower than shared memory.
- Note that: multithreads could read a same data element at the same time.
  - They cannot read and write at the same time, →race condition.
  - They cannot write to a same location at the same time, →race condition.

Simple diff()

- We learned shared memory.
- Motivation of Shared memory
  - Reduce the redundant global memory read.
  - All threads in a same block can access variables in the shared memory allocated to that block.
  - Equivalent to the local (regional) subset of results in pthread programming. →Good locality.

Improved diff()

- Instead we use shared memory



Assume here, blockDim.x = 4, size of input vector is 10;

```
// optimized version of adjacent difference
__global__ void adj_diff(int *result, int *input, int n)
{
  // shorthand for threadIdx.x
  int tx = threadIdx.x;
  // allocate a __shared__ array, one element per thread
  __shared__ int s_data[BLOCK_SIZE];
  // each thread reads one element to s_data
  unsigned int i = blockDim.x * blockIdx.x + tx;
  if( i < n ){
      s_data[tx] = input[i];
  }
  // avoid race condition: ensure all loads
  // complete before continuing
  __syncthreads();
  //continued on next slide
  //…
```

```
  //...
  if(tx > 0)
    result[i] = s_data[tx] - s_data[tx-1];
  else if(i > 0)
  {
    // handle thread block boundary
    result[i] = s_data[tx] - input[i-1];
  }
}
```

Improved diff()

- Data first loaded from global memory into shared memory.
  - Each thread loads one element at an unique position to an unique position in shared memory.
  - Using global data index **i**, and thread id **tx** as index into shared memory.
  - Thinking how global index i=4 corresponds to tx = 0 of block 1?
  - Thinking why it requires a synchthreads() after shared memory data loading?

Improved diff()

- For this simple example, if blockDim.x = 4, size of input vector is 10;
- How many instances of tx in the device? What is the range of tx?
- How many instances of array s_data[] in the device?
- How many instances of i in the device? What is the range of i?
- Note that, maximum shared memory size for each block is 48K on our device.

Improved diff()

- Then we reduced the total number of global memory,
  - Each thread does one data load into shared mem.
  - First thread in a block needs another one global memory access for boundary element.

Wrap Up

- We learned an example that use shared memory.
- You have to get used to per-thread thinking, and per-block thinking.
- Next Class, we will analyze quantitatively how and why shared memory affect GPU performance?