Heterogeneous
Parallel
Programming

Lecture 4.2

Parallel Computation Patterns

A Basic Reduction Kernel

Wen-mei Hwu - University of Illinois at Urbana-Champaign

Objective

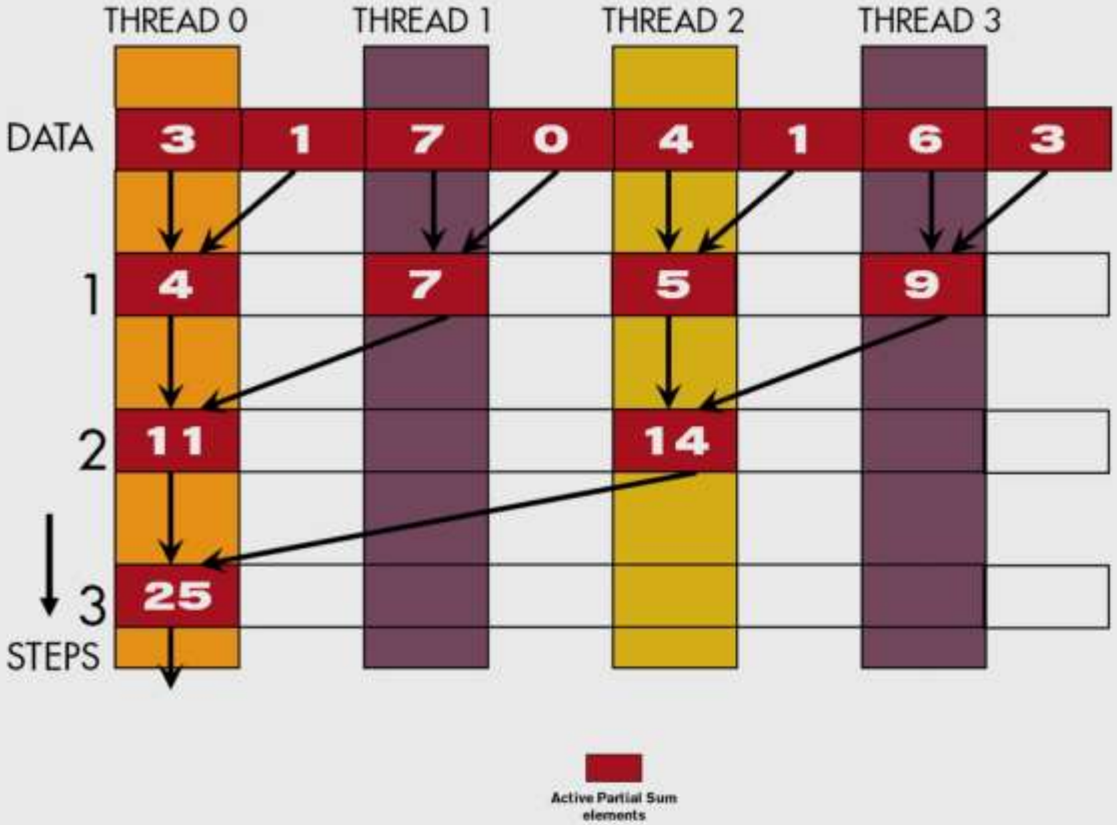
- To learn to write a basic reduction kernel
    - Thread to data index mapping
    - Turning off threads
    - Control divergence

Parallel Sum Reduction

- Parallel implementation:
  - Recursively halve # of threads, add two values per thread in each step
  - Takes log(n) steps for n elements, requires n/2 threads

- Assume an in-place reduction using shared memory
  - The original vector is in device global memory
  - The shared memory is used to hold a partial sum vector
  - Each step brings the partial sum vector closer to the sum
  - The final sum will be in element 0
  - Reduces global memory traffic due to partial sum values
  - Thread block size limits n to be less than or equal to 2,048

A Parallel Sum Reduction Example



A Naive Thread Index to Data Mapping

- Each thread is responsible of an even-index location of the partial sum vector

- After each step, half of the threads are no longer needed

- One of the inputs is always from the location of responsibility

- In each step, one of the inputs comes from an increasing distance away

A Simple Thread Block Design

- Each thread block takes 2* BlockDim.x input elements
- Each thread loads 2 elements into shared memory

```
__shared__ float partialSum[2*BLOCK_SIZE];

unsigned int t = threadIdx.x;
unsigned int start = 2*blockIdx.x*blockDim.x;
partialSum[t] = input[start + t];
partialSum[blockDim+t] = input[start + blockDim.x+t];
```

The Reduction Steps

```
for (unsigned int stride = 1;
       stride <= blockDim.x;  stride *= 2)
{
  __syncthreads();
  if (t % stride == 0)
    partialSum[2*t]+=
   partialSum[2*t+stride];
}
```

Why do we need syncthreads()?

Barrier Synchronization

- __syncthreads() are needed to ensure that all elements of each version of partial sums have been generated before we proceed to the next step

Back to the Global Picture

- At the end of the kernel, Thread 0 in each thread block writes the sum of the thread block in partialSum[0] into a vector indexed by the blockIdx.x

- There can be a large number of such sums if the original vector is very large
  - The host code may iterate and launch another kernel

- If there are only a small number of sums, the host can simply transfer the data back and add them together.

To learn more, read Section 6.1