CENG443
Heterogeneous Parallel Programming

Reduction

Partition and Summarize

A commonly used strategy for processing large input data sets

There is no required order of processing elements in a data set

- Partition the data set into smaller chunks
- Have each thread to process a chunk
- Use a reduction tree to summarize the results from each chunk into the final answer

e.g., Google and Hadoop MapReduce frameworks support this strategy

Reduction Computation

Summarize a set of input values into one value using a “reduction operation”

- Binary
- Associative

Max, min, sum, product

Sequential Reduction - O(n)

Initialize the result as an identity value for the reduction operation

- Smallest possible value for max reduction
- Largest possible value for min reduction
- 0 for sum reduction
- 1 for product reduction

Iterate through the input and perform the reduction operation between the result value and the current input value

Parallel Reduction

Each thread adds two values in each step

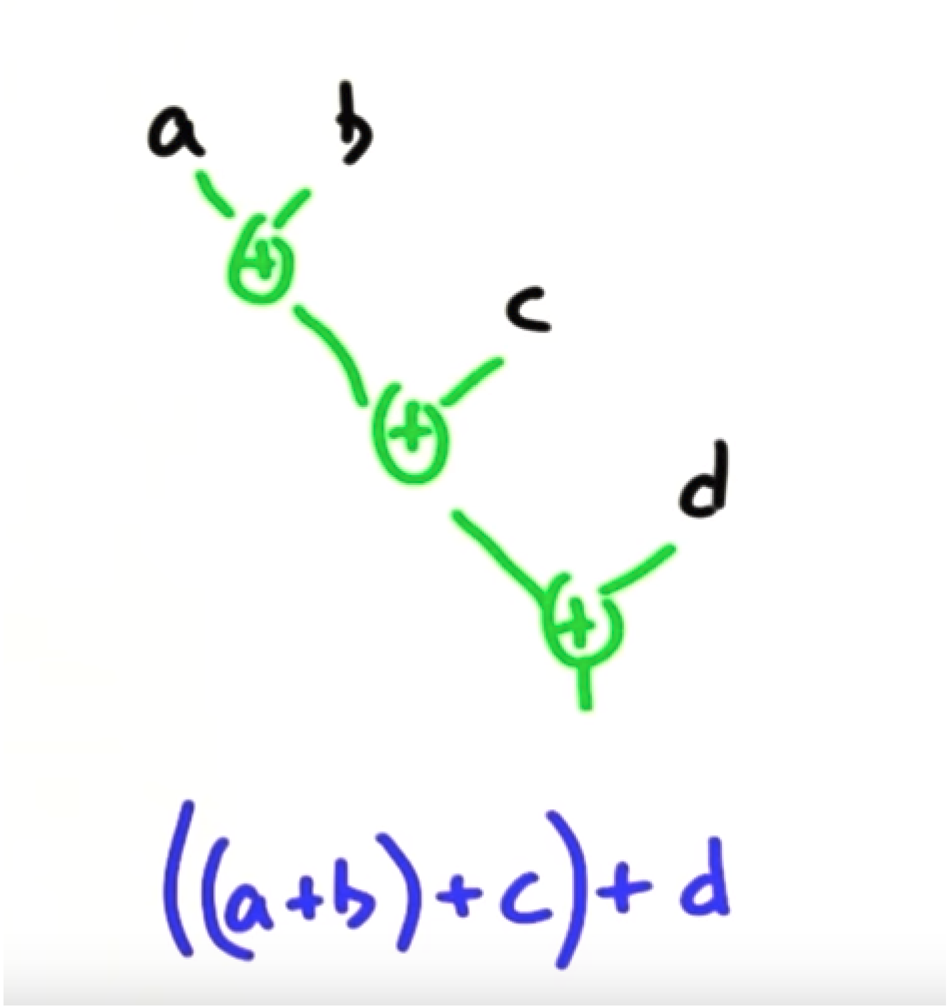
Recursively halve # of threads

Takes log(n) steps for n elements, requires n/2 threads

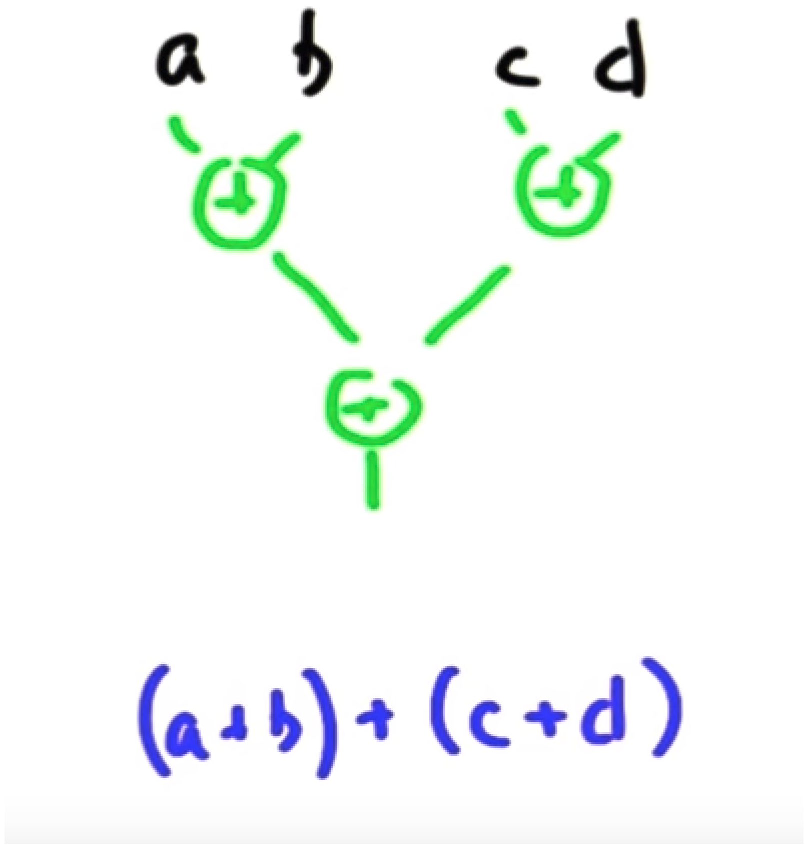
Work complexity: O(n)

Step complexity: n-1 operations in $\log_2(n)$ step

Parallel Reduction

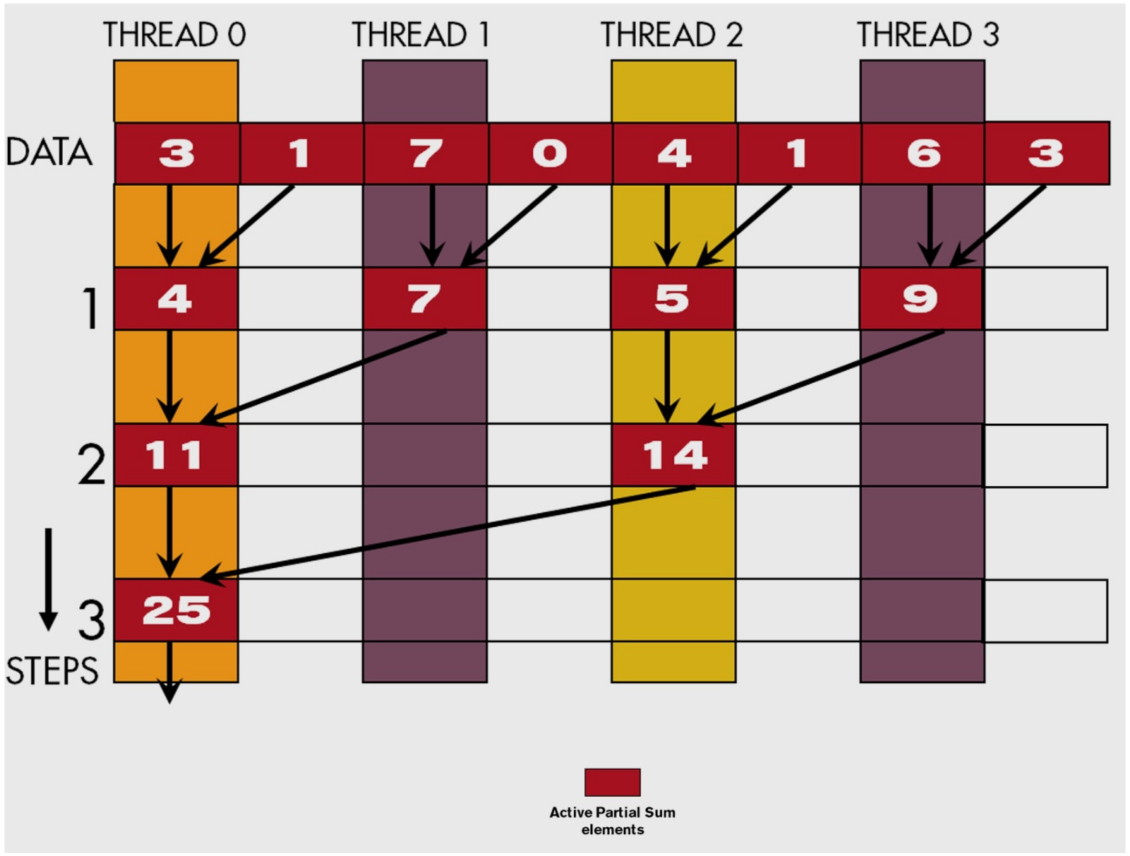


$((a+b)+c)+d$



$(a+b)+(c+d)$

Parallel Sum Reduction Example



Parallel Reduction

For n input values, the reduction tree performs (1/2)n + (1/4)n + (1/8)n + … (1)n = (1- (1/n))n = n-1 operations

1,000,000 input values take 20 steps

500,000 threads in the first step

Parallel Sum Reduction Kernel

The original vector is in device global memory

The shared memory is used to hold a partial sum vector

Initially, the partial sum vector is simply the original vector

Each step brings the partial sum vector closer to the sum

The final sum will be in element 0 of the partial sum vector

Reduces global memory traffic due to reading and writing partial sum values

A Naive Thread to Data Mapping

- Each thread is responsible for an even-index location of the partial sum vector
- After each step, half of the threads are no longer needed
- Inputs are always from the partial sum vector
- In each step, one of the inputs comes from an increasing distance away

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];
```

Reduction Steps

```
for (unsigned int stride = 1; stride <= blockDim.x;  stride *= 2)
{
  __syncthreads();
//all elements of each version of partial sums have been generated
before we proceed to the next step
  if (t % stride == 0)
    partialSum[2*t]+= partialSum[2*t+stride];
}
```

About Naive Reduction Kernel

In each iteration, two control flow paths will be sequentially traversed for each warp

- Threads that perform addition and threads that do not
- Threads that do not perform addition still consume execution resources

Half or fewer of threads will be executing after the first step

- All odd-index threads are disabled after first step
- After the 5th step, entire warps in each block will fail the if test, poor resource utilization but no divergence
- This can go on for a while, up to 6 more steps (stride = 32, 64, 128, 256, 512, 1024), where each active warp only has one productive thread until all warps in a block retire

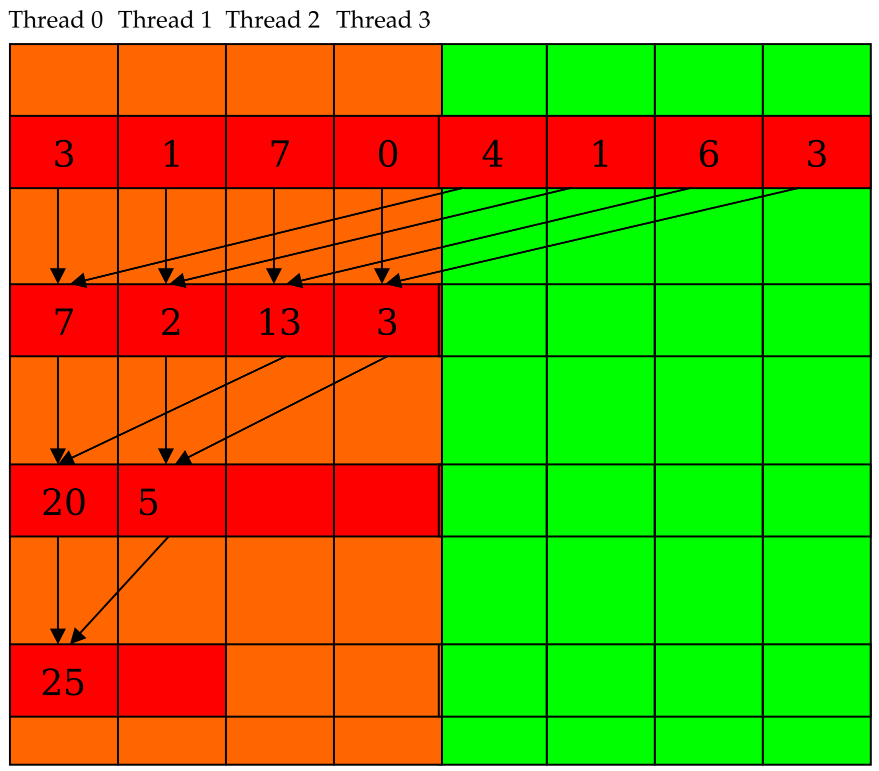
Thread Index

Shift the index usage to improve the divergence behavior (for commutative and associative operators)

Always compact the partial sums into the front locations in the partialSum[ ] array

Keep the active threads consecutive

Example with 4 threads



Better Reduction Kernel

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

Example Case

For a 1024 thread block

- No divergence in the first 5 steps
- 1024, 512, 256, 128, 64, 32 consecutive threads are active in each step
- All threads in each warp either all active or all inactive
- The final 5 steps will still have divergence