Lecture 9: Reductions

Mark Saroufim

Follow along

Chapter 10 of PMPP book

Locally or remotely https://lightning.ai/

git clone https://github.com/cuda-mode/lectures

cd lecture9

nvcc -o sum *.cu

ncu sum

What’s a reduction

Operations that reduce the output size

Most typical take a vector and produce a scalar

min, max, argmax, argmin norm, sum, prod, mean, unique

Demo: torch_reductions.py

Reductions are everywhere

- Mean/Max pooling
- Classification: Argmax
- Loss calculations
- Softmax normalization

Reductions in PyTorch

https://github.com/pytorch/pytorch/blob/main/aten/src/ATen/native/cuda/ReduceOps.cpp

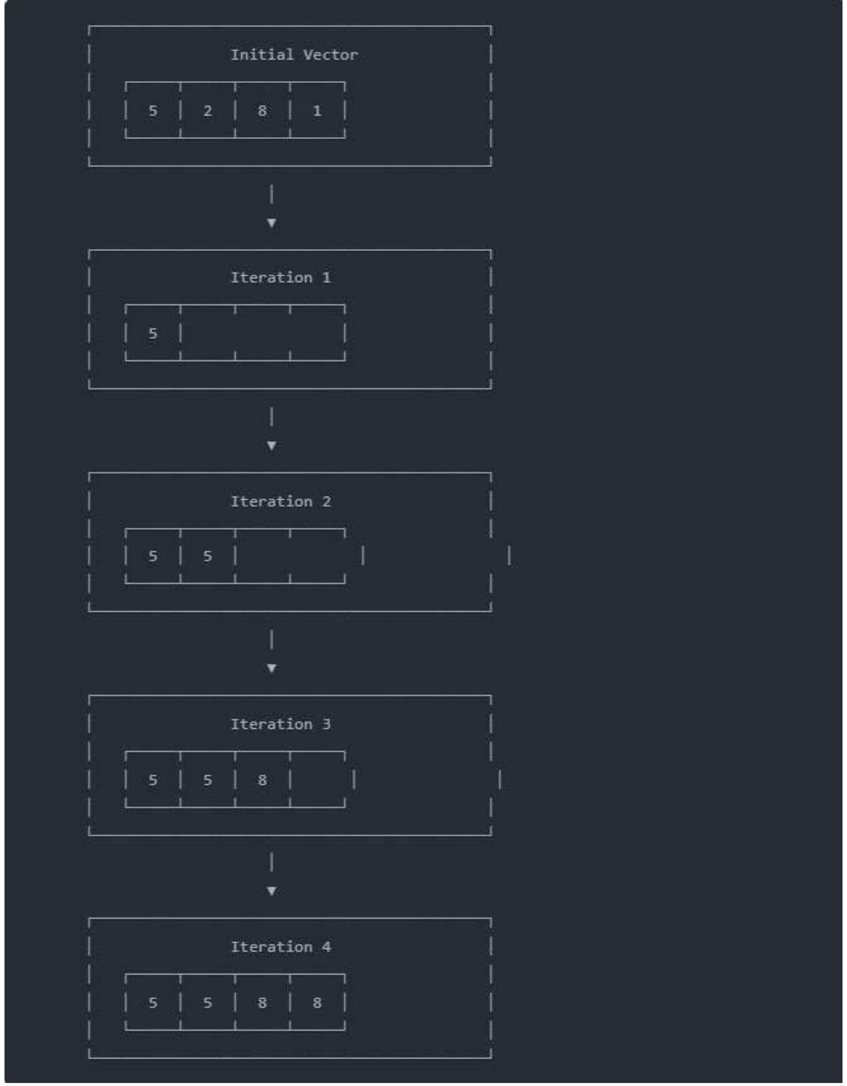
```
>>> a = torch.randn(1, 3)
>>> a
tensor([[ 0.6763,  0.7445, -2.2369]])
>>> torch.max(a)
tensor(0.7445)
```

Serial reduction example

Max operation

Go through elements 1 by 1

Compare new number to old max if greater then update



More general formulation

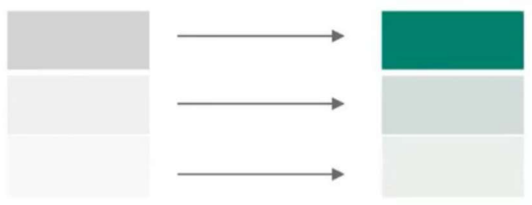
```python
def reduce(data, identity, op):
    result = identity
    for element in data:
        result = op(result, element)
    return result

# Example usage:

# Summation
data = [1, 2, 3, 4, 5]
print(reduce(data, 0, lambda a, b: a + b))  # Output: 15

# Product
print(reduce(data, 1, lambda a, b: a * b))  # Output: 120

# Maximum
print(reduce(data, float('-inf'), max))  # Output: 5

# Minimum
print(reduce(data, float('inf'), min))  # Output: 1
```

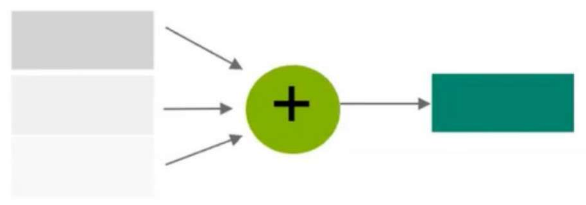
https://gist.github.com/msaroufim/a062aa0b08a4cc57e02db634a67c6b20

Transformation vs reduction

What should the thread strategy be?

Output size < Input size that's why we call them reductions





Transformation:

e.g. `c[i] = a[i] + 10;`

Thread strategy: one thread per output point

Reduction:

e.g. $*c = \Sigma\, a[i]$

Thread strategy: ??

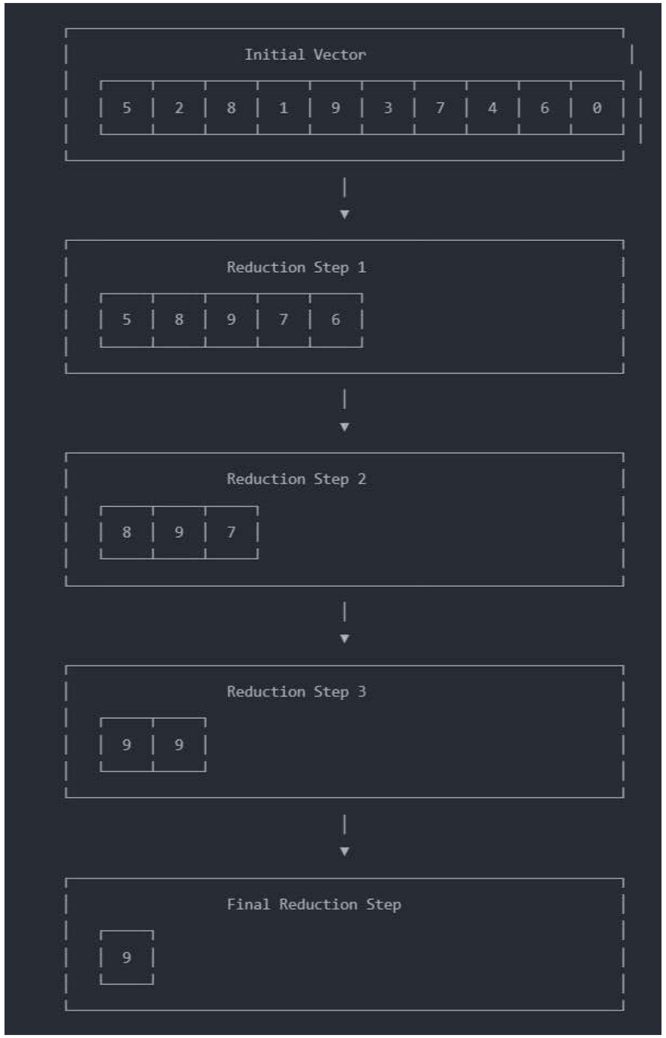
https://www.youtube.com/watch?v=D4I1YMsGNIU&t=1763s

Parallel Reduction visualization

At each step take a pair of elements and compute their max and store the new max in new vector

Continue until there is 1 element in the vector

O(log n) steps



Reduction Trees:

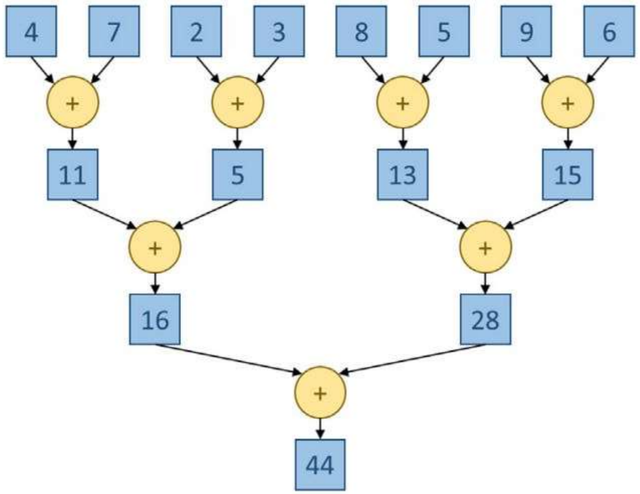


FIGURE 10.5

A parallel sum reduction tree.

Non determinism and accuracy

torch.use_deterministic_algorithms(True)

Demo

- nondeterminism.py
- accuracy.py

Reduction Kernel

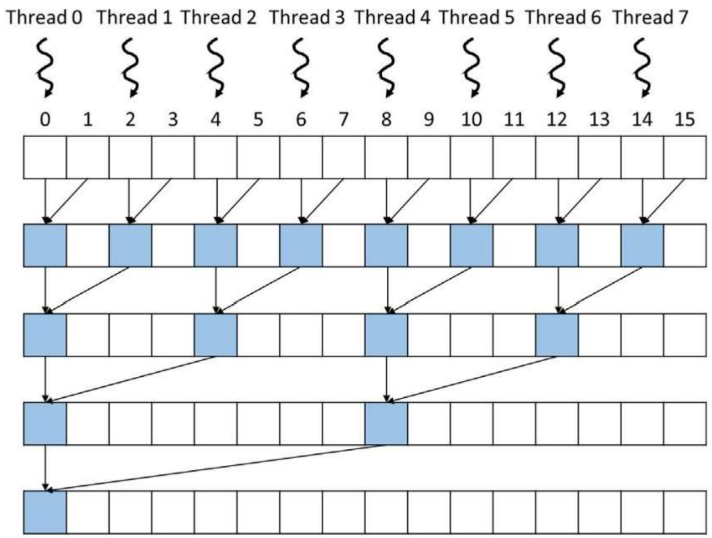


FIGURE 10.7

The assignment of threads ("owners") to the `input` array locations and progress of execution over time for the `SimpleSumReudctionKernel` in Fig. 10.6. The time progresses from top to bottom, and each level corresponds to one iteration of the for-loop.

simple_reduce.cu

- A lot of threads will be inactive :(
- A lot of warps (groups of 32 threads) will be inactive :(
- Let's check ncu -set full

Remember the performance checklist

Lecture 8!

- Control divergence
- Memory divergence
- Minimize global memory access
- Thread coarsening

Minimize Control Divergence

Ensure threads and their owned positions remain close together as time progresses

Quiz: Which other problem does this fix?

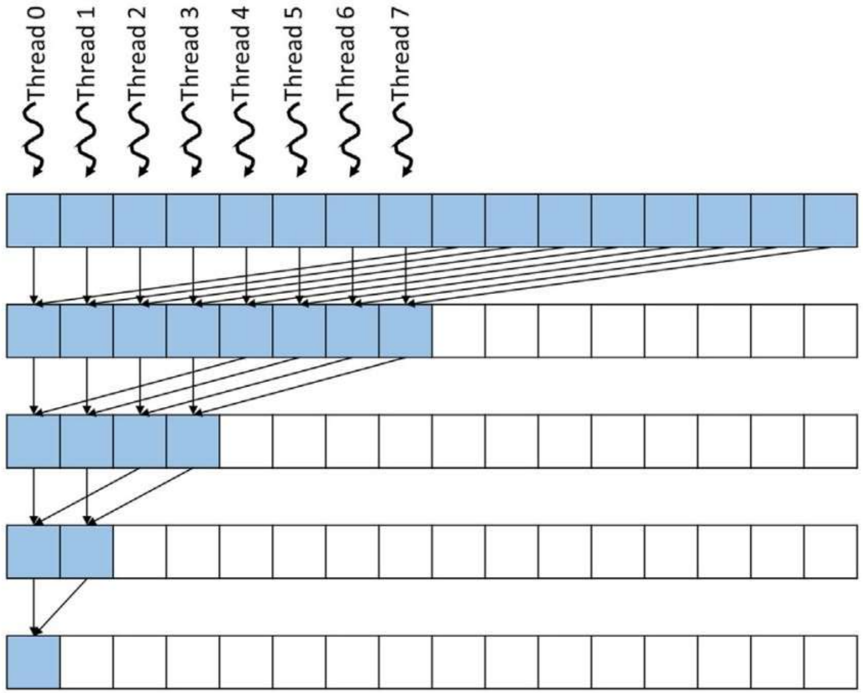


FIGURE 10.8

A better assignment of threads to input array locations for reduced control divergence.

control_divergence_reduce

Minimize Global Memory ACcess

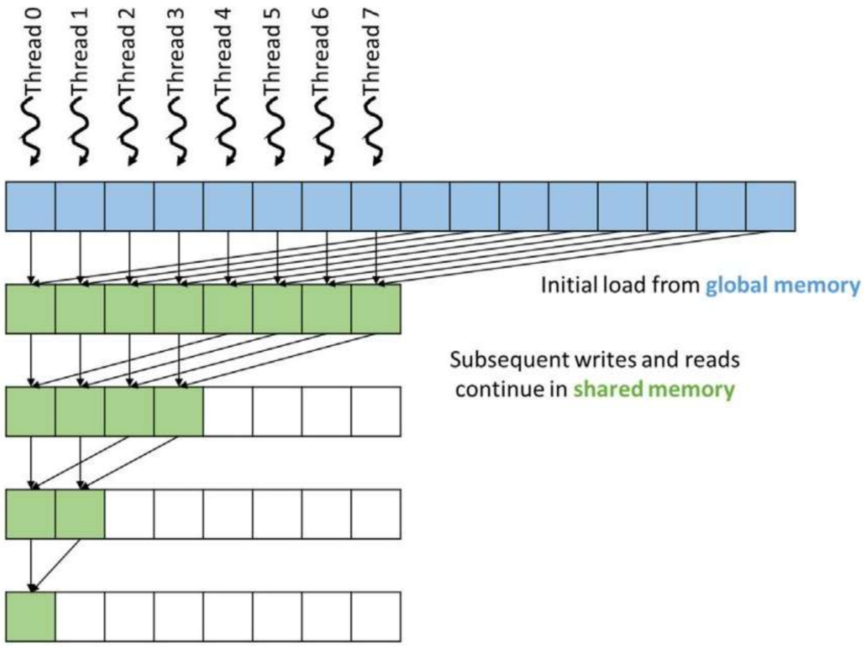


FIGURE 10.10

Using shared memory to reduce accesses to the global memory.

shared_reduce.cu

Hierarchical reduction

Let's try running input size 4096

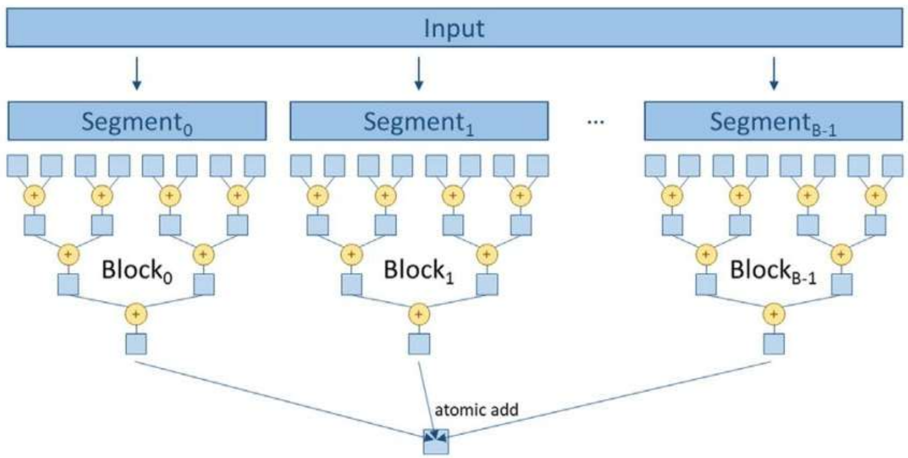


FIGURE 10.12

Segmented multiblock reduction using atomic operations.

segment_reduce.cu

Thread Coarsening (Andreas' favorite optimization)

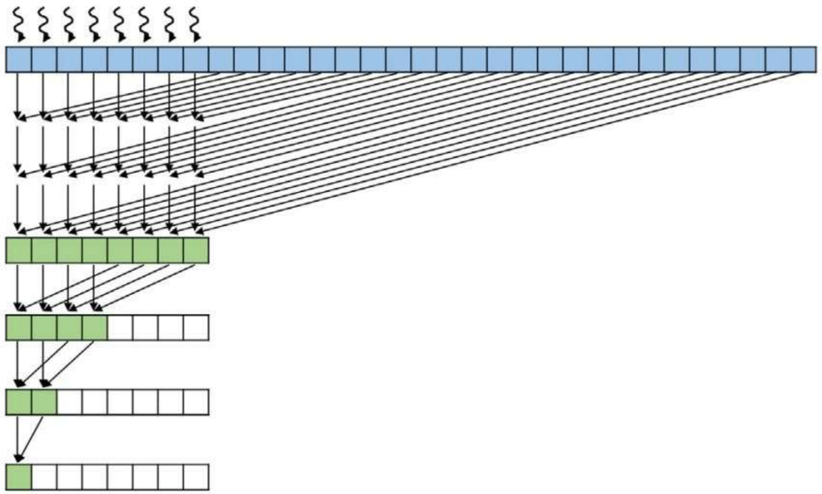


FIGURE 10.14

Thread coarsening in reduction.

reduce_coarsening.cu

Next steps

Lecture 1-8 gave you everything you need to start writing, profiling and shipping kernels in PyTorch so start picking a project - Look for collaborators in #general to stay motivated

Next Lecturer is Oscar who will talk about shipping production CUDA libraries

Looking for lecturers interested in covering prefix sum (scan) and NCCL

Bonus slides: Reductions in the real world

Example of reductions

User facing ops

How reductions are implemented in PyTorch

- https://github.com/pytorch/pytorch/blob/4b494d075093096d822b9d614e2719a0e821c6af/aten/src/ATen/native/cuda/ReduceMaxValuesKernel.cu#L53
- https://github.com/pytorch/pytorch/blob/main/aten/src/ATen/native/cuda/Reduce.cuh
- https://github.com/pytorch/pytorch/blob/main/aten/src/ATen/native/metal/ops/MetalReduce.mm
- CPP style of CUDA (Might need its own lecture)

Key ideas

- Implementation is accumulator and reduction op agnostic
- TensorIterator to iterate over tensor elements
- ReduceConfig: Has kernel launch parameters like block size and number of threads, grid etc.. and its set in setReduceConfig
- Reduce_kernel is where it gets launched
- Reduction strategies: thread level, block level x,y, or global reduce
- Vectorization: Over input and/or output

torch.compile!

To the notebook - reduce_compile.py

Look out for

- ReductionHint
- tl.sum
- triton_heuristics

Triton

https://github.com/openai/triton/blob/main/lib/Conversion/TritonGPUToLLVM/ReduceOpToLLVM.cpp

```
// First reduce all the values along axis within each thread.
reduceWithinThreads(helper, srcValues, accs, indices, rewriter);
// Then reduce across threads within a warp.
reduceWithinWarps(helper, accs, rewriter);
```