Reduction

What is a Reduction?

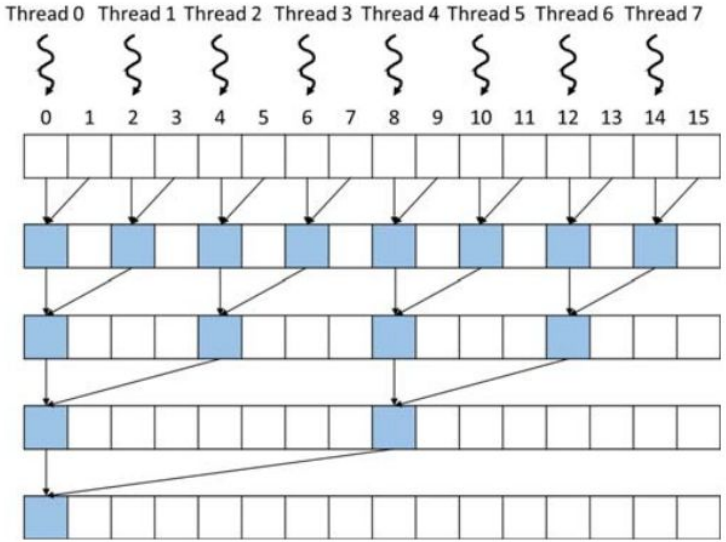
- Derives a single value from an array of values
- Can be performed by sequentially going through every element of the array
- Take as a reference the sequential implementation
- The complexity is $O(N)$
- `float` or `double`?

Usage Examples

- MPI exposes directly a MPI_Reduce and MPI_Allreduce
  - They are used quite a lot
- MapReduce relies on reductions
  - To analyze a large dataset
- Used as a primitive by different sorting algorithms

Naive GPU

- Each thread will be the “owner” of the location to which it is assigned
- The condition inside the loop verifies threads index multiple of 2^n
  - these threads will be active threads
  - the others will be inactive
  - fewer and fewer threads remain active
  - at the last iteration, only one remains active
- The __syncthreads ensure that the results have been written back



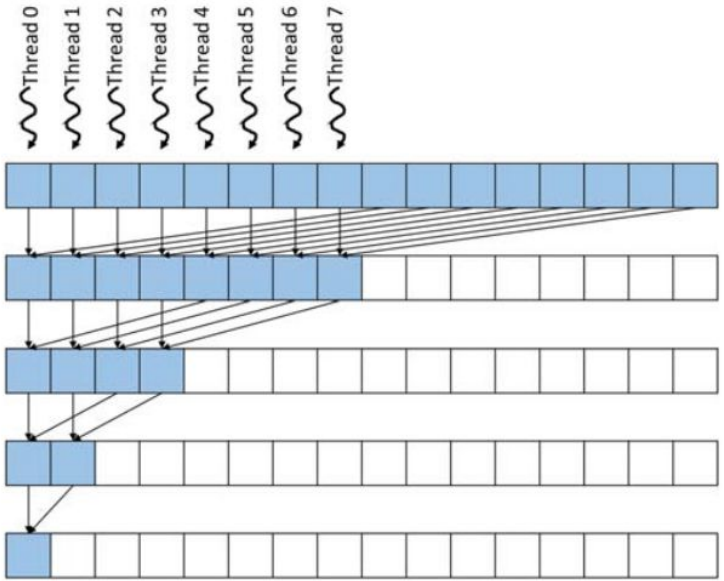
Problems and Optimizations

- Coalescing and Divergence
- Privatization
- Coarsening

Optimization	Benefit to compute cores	Benefit to memory	Strategies
Maximizing occupancy	More work to hide pipeline latency	More parallel memory accesses to hide DRAM latency	Tuning usage of SM resources such as threads per block, shared memory per block, and registers per thread
Enabling coalesced global memory accesses	Fewer pipeline stalls waiting for global memory accesses	Less global memory traffic and better utilization of bursts/ cache lines	Transfer between global memory and shared memory in a coalesced manner and performing uncoalesced accesses in shared memory (e.g., corner turning) Rearranging the mapping of threads to data Rearranging the layout of the data
Minimizing control divergence	High SIMD efficiency (fewer idle cores during SIMD execution)	–	Rearranging the mapping of threads to work and/or data Rearranging the layout of the data
Tiling of reused data	Fewer pipeline stalls waiting for global memory accesses	Less global memory traffic	Placing data that is reused within a block in shared memory or registers so that it is transferred between global memory and the SM only once
Privatization (covered later)	Fewer pipeline stalls waiting for atomic updates	Less contention and serialization of atomic updates	Applying partial updates to a private copy of the data and then updating the universal copy when done
Thread coarsening	Less redundant work, divergence, or synchronization	Less redundant global memory traffic	Assigning multiple units of parallelism to each thread to reduce the price of parallelism when it is incurred unnecessarily

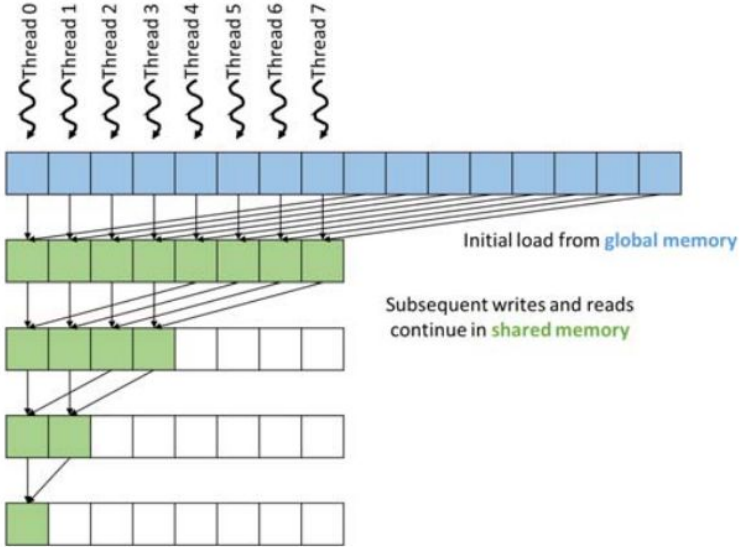
Reduce Memory Divergency

- With a lot of threads, the inactive threads become more visible
  - Previous solution can have different warps with only a punch of threads active
  - We can compact them to increase the scheduling efficiency
- Enable **coalesced** memory access
- Increase thread **convergence**



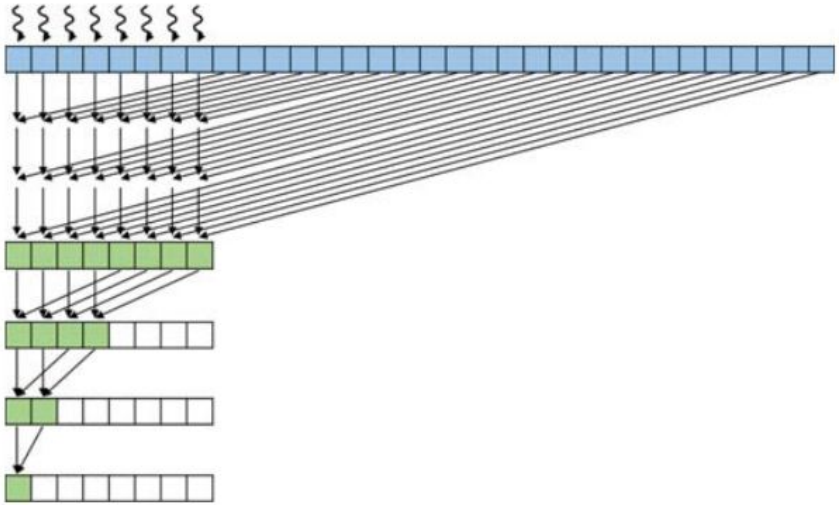
Introduce Privatization

- Threads write their partial sum result values to global memory
  - These values are reread by the same threads and others in the next iterations
- Shared memory has much shorter latency and higher bandwidth
  - It can fit the data required by the block's thread in this case
- Note
  - The `__syncthreads` have been moved up
  - The memory access is still coalesced



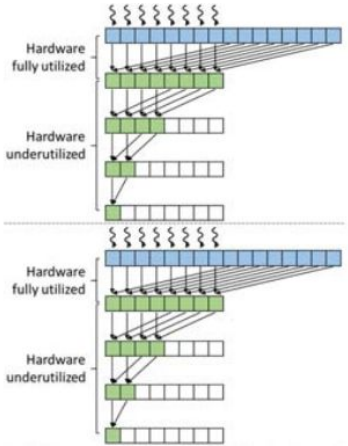
Reduce Overhead: Thread Coarsening

- The hardware is to serialize these thread blocks
  - we are better off serializing them ourselves in a more efficient manner
- The block's segment is identified by multiplying with the `COARSE_FACTOR`
  - threads are not responsible only for adding two elements
- **Trade-off:** we are scheduling fewer threads
  - The occupancy could be low
  - Fine-tuning is required

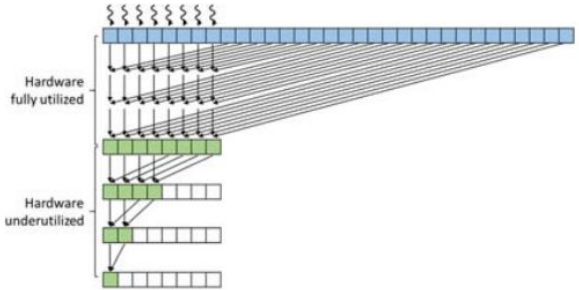


Reduce Overhead: Comparison

- We use the `COARSE_FACTOR` to fully utilized the hardware
  - The price of idle threads is softened much more
  - More steps fully utilized the hardware



(A) Execution of two original thread blocks serialized by the hardware

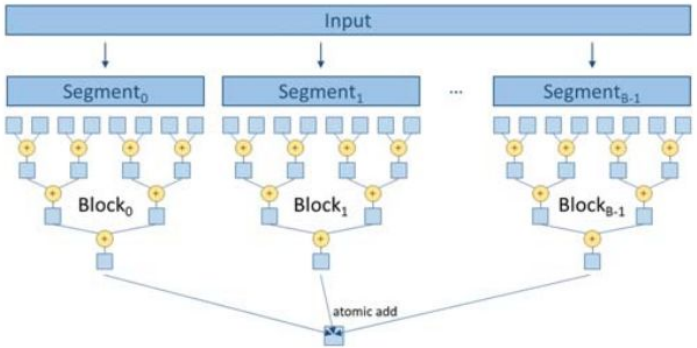


(B) Execution of one coarsened thread block doing the work of two original thread blocks

Code Hands-on

Bonus

- It is available in the Thrust Library
  - Documentation here
- You can do the same exercise with a different hierarchical approach
  - Each block works on a different input's segments
  - Instead, results are written to the same output using an `atomicAdd`
  - Try it as an exercise



Thank you for your attention!

Gianmarco Accordi
gianmarco.accordi@polimi.it