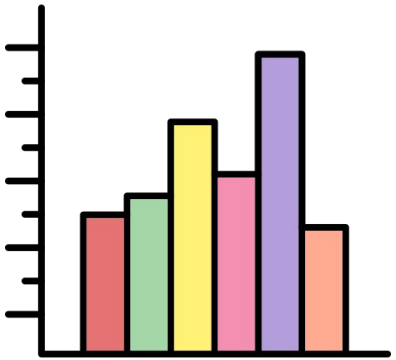
Histogram

Profiling and Optimizations

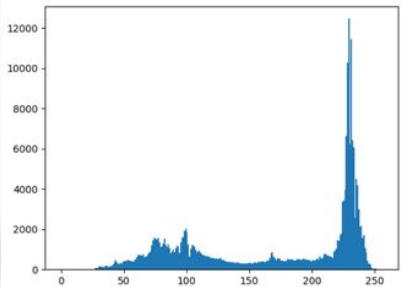
Algorithm Description

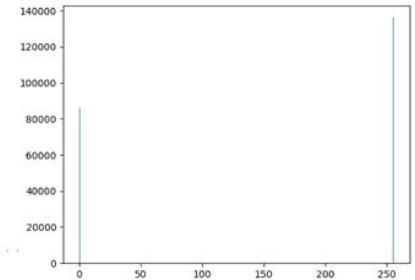
- A histogram is a display of the number count or percentage of occurrences of data values in a dataset
- The shape of the histogram provides a quick way to determine whether there are significant phenomena in the dataset.



Algorithm Usage

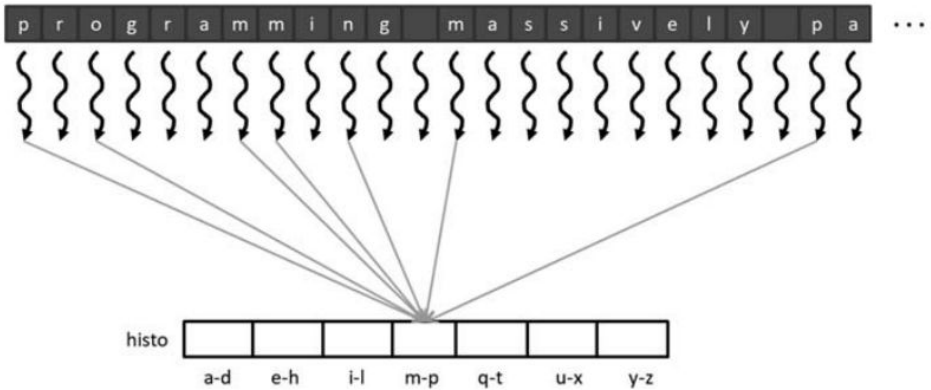
- Computing histograms of image subareas is an important approach to ***feature extraction***
  - in computer vision, in which the features refer to patterns of interest in image
- They are used whenever there is a large volume of data that needs to be analyzed to distill interesting events





Parallelization and Concurrency

- You can launch as many threads as there are data elements and have each thread process one input element
  - multiple threads can modify the same counter causing *output interference*



Race Conditions

- The outcome of two or more simultaneous update operations varies depending on the relative timing of the operations that are involved

Time	Thread 1	Thread 2
1	(0) Old ← histo[x]	
2	(1) New ← Old + 1	
3	(1) histo[x] ← New	
4		(1) Old ← histo[x]
5		(2) New ← Old + 1
6		(2) histo[x] ← New

(A)

Time	Thread 1	Thread 2
1	(0) Old ← histo[x]	
2	(1) New ← Old + 1	
3		(0) Old ← histo[x]
4	(1) histo[x] ← New	
5		(1) New ← Old + 1
6		(1) histo[x] ← New

(B)

Atomic Operations

- Sequences of instructions that guarantee atomic accesses and updates of data
  - Read, modify, and write parts of the operation form an undividable unit
- Realized with hardware support to lock the memory unit from other's usage
- They do not enforce any particular execution order

Atomic Operations in CUDA

- Performs a read-modify-write atomic operation on one 32-bit, 64-bit, or 128-bit word residing in global or shared memory
- They are *intrinsic functions*
  - Compiled into a hardware atomic operation instruction, using CAS for example
- Programing guide reference

Latency and Throughput

- Consider a system with
  - 64-bit(8 byte) double data rate DRAM per channel
  - 8 channel
  - 1 GHz clock frequency
  - 200 clocks access latency
- The number of data read varies
  - **NO** atomic operations

    *8(bytes/transfer) * 2(transfers per clock per channel) * 1 GHz * 8(channels) = 128 GB/s*

    If access 4 bytes data we read *32 G* element per second
  - **WITH** atomic operations, with 400 cycles per atomic (read plus write)

    1/400 atomics/clock 1 GHz = 2.5 M atomics/second

    Performed on a single memory location

Naive Implementation

- Every threads process a section of the input
  - The input is divided equally between threads
- Consecutive threads will access contiguous memory region
  - Coalesced memory access
- The kernel spawn a grid with a number of thread equal to the input chars
  - **No** coarsening is performed
  - There is a limit to the number of thread and block scheduled on an SM
    - Otherwise spilling is performed
- Use of CUDA `atomicAdd`
  - Commit to global memory

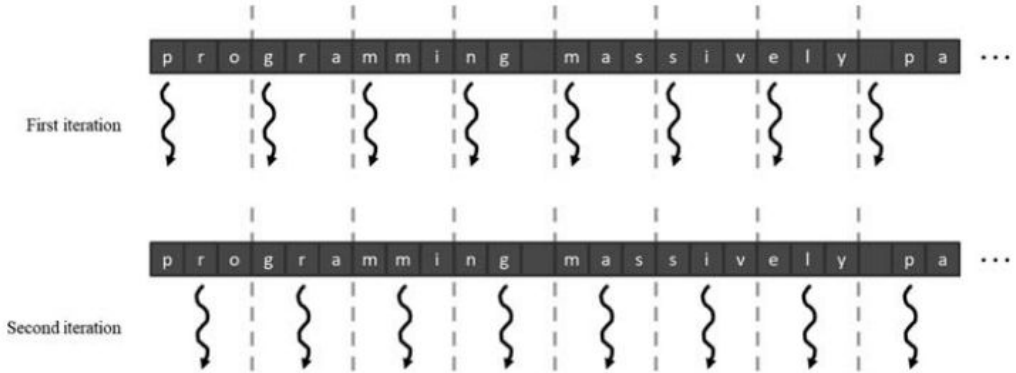
Possible Optimizations

- Coarsening
- Coalescing
- Privatization
  - Aggregation

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

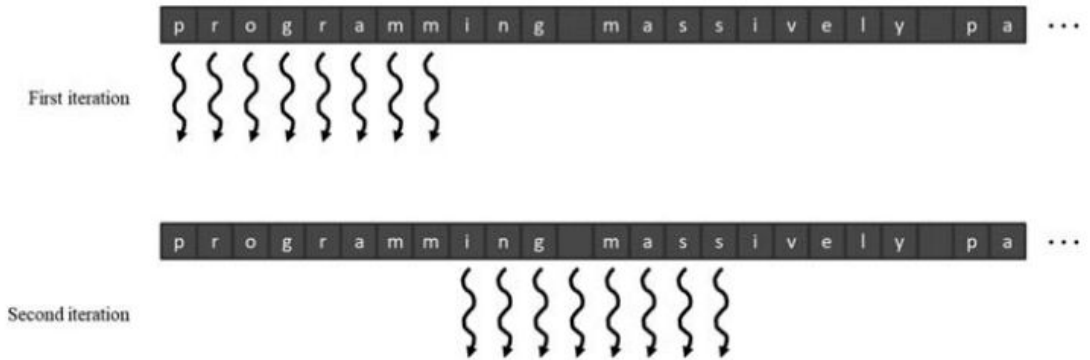
Coarsening Contiguous Partitioning

- Each threads process multiple input elements
- Each thread takes a contiguous input section
- Segment of equal size among threads



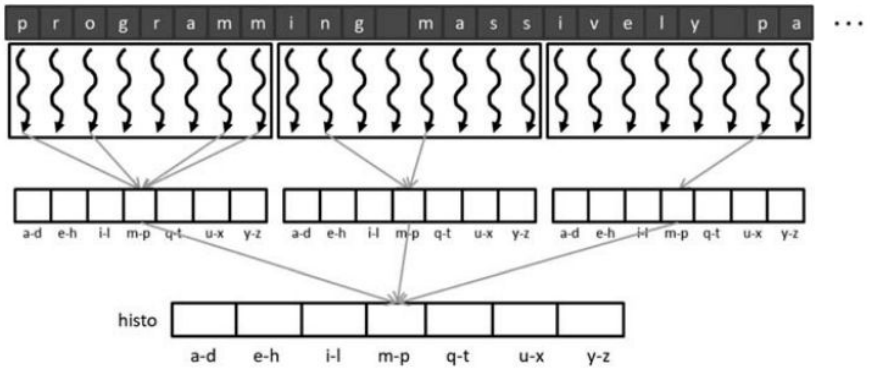
Coarsening Interleaves Partitioning

- Each threads process multiple input elements
- Consecutive threads access contiguous input elements
  - Coalesced memory access



Privatization

- Histogram private copies for each thread
  - Reduce data contention
- Once the computation is done commit back to global memory
- Private copies can be in
  - Global memory
  - Shared memory
  - Registers



Aggregation

- Some datasets have a large concentration of identical data values in localized areas
  - High contention
  - Reduced throughput
- Each thread aggregates consecutive updates into a single update if they are updating the same bin

Code Hands-on

Thank you for your attention!

Gianmarco Accordi
gianmarco.accordi@polimi.it