GPU Teaching Kit

Accelerated Computing

Module 7.5 – Parallel Computation Patterns (Histogram)

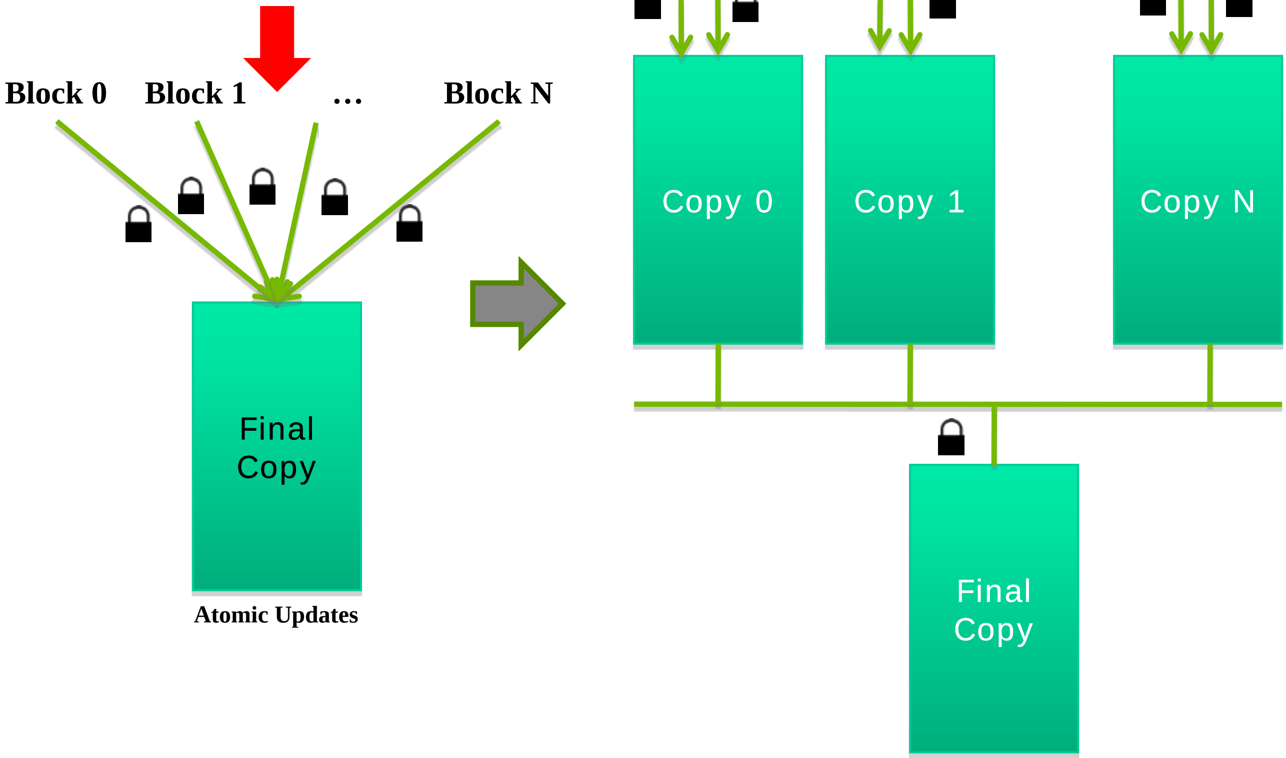
Privatization Technique for Improved Throughput

Objective

- Learn to write a high performance kernel by privatizing outputs
  - Privatization as a technique for reducing latency, increasing throughput, and reducing serialization
  - A high performance privatized histogram kernel
  - Practical example of using shared memory and L2 cache atomic operations

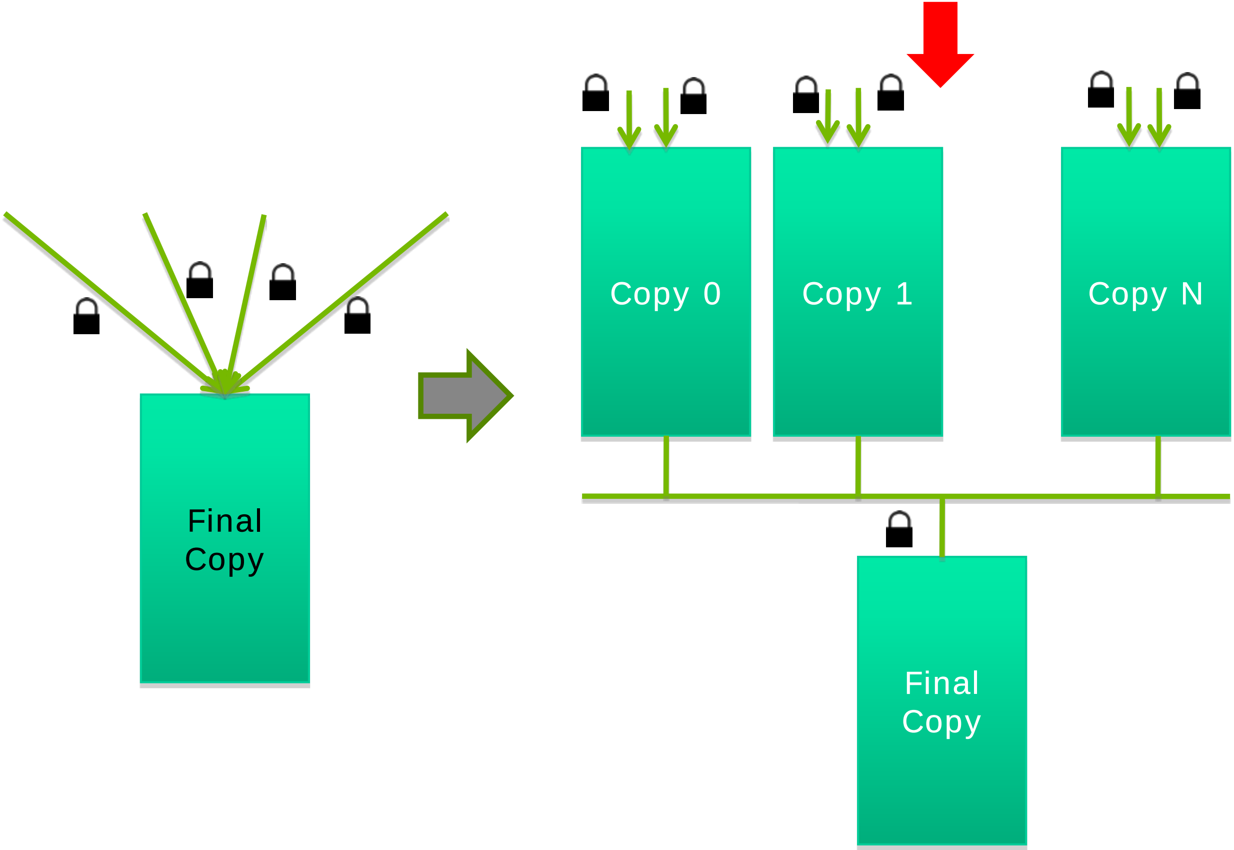
Privatization

Heavy contention and serialization

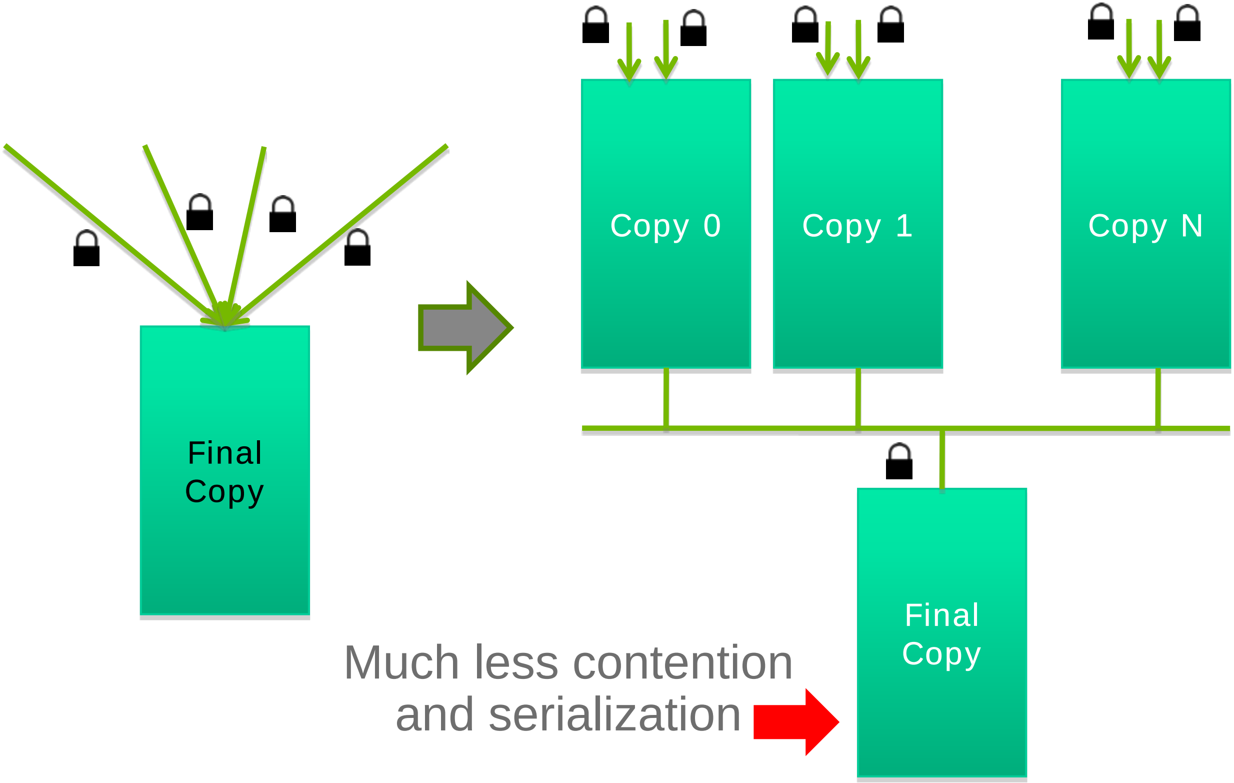


Privatization (cont.)

Much less contention and serialization



Privatization (cont.)



Much less contention
and serialization

Cost and Benefit of Privatization

- Cost
  - Overhead for creating and initializing private copies
  - Overhead for accumulating the contents of private copies into the final copy

- Benefit
  - Much less contention and serialization in accessing both the private copies and the final copy
  - The overall performance can often be improved more than 10x

Shared Memory Atomics for Histogram

- Each subset of threads are in the same block
- Much higher throughput than DRAM (100x) or L2 (10x) atomics
- Less contention – only threads in the same block can access a shared memory variable
- This is a very important use case for shared memory!

Shared Memory Atomics Requires Privatization

– Create private copies of the histo[] array for each thread block

```
__global__ void histo_kernel(unsigned char *buffer,
             long size, unsigned int *histo)
{
   __shared__ unsigned int histo_private[7];
```

Shared Memory Atomics Requires Privatization

- Create private copies of the histo[] array for each thread block

```
__global__ void histo_kernel(unsigned char *buffer,
            long size, unsigned int *histo)
{
    __shared__ unsigned int histo_private[7];

   if (threadIdx.x < 7) histo_private[threadidx.x] = 0;
   __syncthreads();
```

Initialize the bin counters in the private copies of histo[]

Build Private Histogram

```
  int i = threadIdx.x + blockIdx.x * blockDim.x;
// stride is total number of threads
  int stride = blockDim.x * gridDim.x;
  while (i < size) {
      atomicAdd( &(private_histo[buffer[i]/4), 1);
      i += stride;
  }
```

Build Final Histogram

```
 // wait for all other threads in the block to finish
__syncthreads();

if (threadIdx.x < 7) {
        atomicAdd(&(histo[threadIdx.x]), private_histo[threadIdx.x] );
}

}
```

More on Privatization

- Privatization is a powerful and frequently used technique for parallelizing applications

- The operation needs to be associative and commutative
  - Histogram add operation is associative and commutative
  - No privatization if the operation does not fit the requirement

- The private histogram size needs to be small
  - Fits into shared memory

- What if the histogram is too large to privatize?
  - Sometimes one can partially privatize an output histogram and use range testing to go to either global memory or shared memory

NVIDIA

GPU Teaching Kit

Accelerated Computing

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN