CSC266 Introduction to Parallel Computing using GPUs
Synchronization and Communication

Sreepathi Pai

November 8, 2017

URCS

Outline

Barriers

Atomics

Warp Primitives

Memory Fences

Outline

Barriers

Atomics

Warp Primitives

Memory Fences

Thread Block barriers

- Reads and Writes to the same location in shared memory must be separated by a barrier

```
__syncthreads()
```

Warp Synchronous Programming

Is this code okay?

```
__shared__ volatile int values[8];

int warpid = threadIdx.x / 32;

if(threadIdx.x % 32 == 0)
  values[warpid] = threadIdx.x;

printf("%d\n", values[warpid]);
```

Global barriers?

- Does the GPU support global barriers?
- If not, why not?

Unsafe Global Barriers

- Calculate residency R of kernel (use `cudaOccupancyMaxActiveBlocksPerMultiprocessor`)
- Launch $nSM * R$ blocks, where nSM is number of SMs
- Use global barrier between blocks!
- Usually known as *Persistent Threads*

CUDA 9: Global Barriers/Cooperative Kernels

- Adds Global Barriers support to CUDA
- And lots more! (See "Cooperative Groups")

```
grid_group grid = this_grid()
grid.sync()
```

- Instead of assuming which blocks are running, discover that at runtime
- Limit synchronization to those blocks

CUDA 9: Warp Barriers

- Volta GPUs no longer execute warps in lockstep
  - "Warp-synchronous" in CUDA literature
  - They can, but it is no longer required
- Every thread has its own PC
- New `__syncwarp()` barrier for threads in warp
  - Largely for use by code that assumes warp-synchronous execution

Outline

Barriers

Atomics

Warp Primitives

Memory Fences

Atomic Compare and Swap

```
atomicCAS(address, compare, val)
```

Pseudocode, `ATOMIC` means it executes all the instructions inside "atomically".

```
ATOMIC {
   cur = *address;
   if(cur == compare)
       *address = val;

   return cur;
}
```

Other Atomic Functions

- Arithmetic
  - `atomicAdd`, `atomicSub`
- Minimum/Maximum
  - `atomicMin`, `atomicMax`
- Increase/Decrease
  - `atomicInc`, `atomicDec`
  - Always increase or decrease by 1
  - Read definitions carefully!
- Bitwise
  - `atomicAnd`, `atomicOr`, `atomicXor`

How Expensive are Atomics?

- Atomics to same location must be executed serially
- Atomics to different locations in same cache line can be executed in parallel
- Atomics to different locations execute in parallel
    - "As cheap as writes"

Other Atomic Functions as CAS

- Can we implement all other atomic functions using just CAS?

Locks with Atomic CAS?

```
while(atomicCAS(lock, UNLOCKED, LOCKED) == UNLOCKED);
// do something
lock = UNLOCKED;
```

Why Spinlocks Don't Work on GPUs

- What happens when two threads in the same warp try to obtain the lock?

Outline

Barriers

Atomics

Warp Primitives

Memory Fences

Warp Primitives

- Allow threads in warps to communicate without using shared memory
- All deprecated in CUDA 9.0 and replaced by more general functions
- See CUDA 8.0 documentation for now

Warp Votes

- `__any(predicate)`
  - Each thread in warp gets 1 if any predicate is 1
- `__all(predicate)`
  - Each thread in warp gets 1 if only all predicates are 1
- `__ballot(predicate)`
  - Each thread gets bit pattern of predicates
  - Use `__popc()` and `__ffs` for further manipulation

Warp Shuffles

- Allow warps to transfer data to each other
- All functions below are deprecated in CUDA 9.0

```
T __shfl(T var, int srcLane, int width=warpSize);
T __shfl_up(T var, unsigned int delta, int width=warpSize);
T __shfl_down(T var, unsigned int delta, int width=warpSize);
T __shfl_xor(T var, int laneMask, int width=warpSize);
```

Warp Permutation using Warp Shuffles

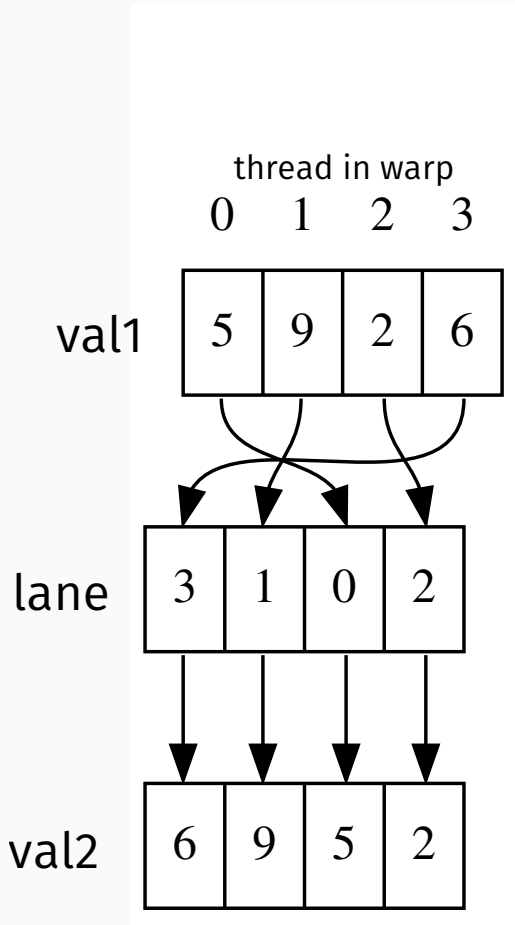
- Assume 4 threads in warp, i.e. 4 lanes
- Can read other lane's values using lane index
- Example: Permutation (pictured)

```
val2 = __shfl(val1, lane)
```

- Example: All lanes read `val` from lane 0 (broadcast)

```
val_t0 = __shfl(val, 0)
```

- `__shfl_up` and `__shfl_down` "shift" values across lanes



Reducing the number of atomics

Adapted from the CUDA programming guide:

```
{
    unsigned int writemask = __ballot(1);
    unsigned int total = __popc(writemask);
    unsigned int prefix = __popc(writemask & __lanemask_lt());
    // Find the lowest-numbered active lane
    int elected_lane = __ffs(writemask) - 1;
    int base_offset = 0;
    if (prefix == 0) {
        base_offset = atomicAdd(p, total);
    }
    base_offset = __shfl(base_offset, elected_lane);
    int thread_offset = prefix + base_offset;
    return thread_offset;
}
```

Outline

Barriers

Atomics

Warp Primitives

Memory Fences

Memory Consistency (or Ordering)

- Assume all variables below start as 0

```
/* Thread 0 */
   val_t0 = 100;
   written_val_t0 = 1;

/* Thread 1 */
   while(write_val_t0 == 0);
   printf("%d\n", val_t0);
```

Will the program ever print zero?

Memory Consistency

In what order are reads and writes in one thread seen by other threads?

Enforcing Memory Ordering – Memory Fences

- `__threadfence_block()` – X: block
- `__threadfence()` – X: GPU
- `__threadfence_system()` – X: system

All writes made by threads in same X before executing fence are ordered before writes made after executing fence.

```
// writes before fence
fence()
// writes after fence
```