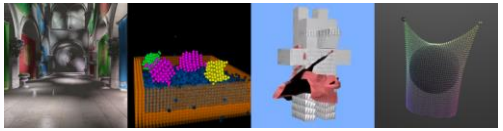

CUDA Atomics

Patrick Cozzi
University of Pennsylvania
CIS 565 - Fall 2016

Atomic Functions

- What is the value of `count` if 8 threads execute `++count`?

```
__device__ unsigned int count = 0;
// ...
++count;
```


Atomic Functions

- Read-modify-write atomic operation
  - Guaranteed no interference from other threads
  - No guarantee on order
- Shared or global memory
- Requires compute capability 1.1 (> G80)

See G.1 in the NVIDIA CUDA C Programming Guide for full compute capability requirements

Atomic Functions

- What is the value of `count` if 8 threads execute `atomicAdd` below?

```
__device__ unsigned int count = 0;
// ...
// atomic ++count
atomicAdd(&count, 1);
```

Atomic Functions

- How do you implement `atomicAdd`?

```
__device__ int atomicAdd(
  int *address, int val);
```

5

Atomic Functions

- How do you implement `atomicAdd`?

```
__device__ int atomicAdd(int *address, int val)
{ // Made up keyword: __lock
  int old;
  __lock (address) {
    old = *address;
    *address += val;
  }
  return old;
}
```

6

Atomic Functions

- How do you implement `atomicAdd` **without** locking?

7

Atomic Functions

- How do you implement `atomicAdd` **without** locking?
- What if you were given an atomic compare and swap?

```
int atomicCAS(int *address, int
  compare, int val);
```

8

Atomic Functions

- `atomicCAS` pseudo implementation

```
int atomicCAS(int *address,
  int compare, int val)
{ // Made up keyword
  __lock(address) {
    int old = *address;
    *address = (old == compare) ? val : old;
    return old;
  }
}
```

9

Atomic Functions

- `atomicCAS` pseudo implementation

```
int atomicCAS(int *address,
  int compare, int val)
{ // Made up keyword
  __lock(address) {
    int old = *address;
    *address = (old == compare) ? val : old;
    return old;
  }
}
```

10

Atomic Functions

- `atomicCAS` pseudo implementation

```
int atomicCAS(int *address,
  int compare, int val)
{ // Made up keyword
  __lock(address) {
    int old = *address;
    *address = (old == compare) ? val : old;
    return old;
  }
}
```

11

Atomic Functions

- Example:

```
*addr = 1;

atomicCAS(addr, 1, 2);
atomicCAS(addr, 1, 3);
atomicCAS(addr, 2, 3);
```

12

Atomic Functions

- Example:

```
*addr = 1;

atomicCAS(addr, 1, 2);  // returns 1
atomicCAS(addr, 1, 3);  // *addr = 2
atomicCAS(addr, 2, 3);
```

13

Atomic Functions

- Example:

```
*addr = 1;

atomicCAS(addr, 1, 2);
atomicCAS(addr, 1, 3);  // returns 2
atomicCAS(addr, 2, 3);  // *addr = 2
```

14

Atomic Functions

- Example:

```
*addr = 1;

atomicCAS(addr, 1, 2);
atomicCAS(addr, 1, 3);
atomicCAS(addr, 2, 3);  // returns 2
                        // *addr = 3
```

15

Atomic Functions

- Again, how do you implement `atomicAdd` given `atomicCAS`?

```
__device__ int atomicAdd(
  int *address, int val);
```

16

Atomic Functions

```
__device__ int atomicAdd(int *address, int val)
{
  int old = *address, assumed;
  do {
    assumed = old;
    old = atomicCAS(address,
      assumed, val + assumed);
  } while (assumed != old);
  return old;
}
```


Atomic Functions

```
__device__ int atomicAdd(int *address, int val)
{
  int old = *address, assumed;
  do {
    assumed = old;
    old = atomicCAS(address,
      assumed, val + assumed);
  } while (assumed != old);
  return old;
}
```

Read original value at `*address`.

Atomic Functions

```
__device__ int atomicAdd(int *address, int val)
{
  int old = *address, assumed;
  do {
    assumed = old;
    old = atomicCAS(address,
      assumed, val + assumed);
  } while (assumed != old);
  return old;
}
```

If the value at `*address` didn't change, increment it.

Atomic Functions

```
__device__ int atomicAdd(int *address, int val)
{
  int old = *address, assumed;
  do {
    assumed = old;
    old = atomicCAS(address,
      assumed, assumed + val);
  } while (assumed != old);
  return old;
}
```

Otherwise, loop until `atomicCAS` succeeds.

The value of `*address` after this function returns is not necessarily the original value of `*address + val`, why?

Atomic Functions

- Lots of atomics:

```
// Arithmetic          // Bitwise
atomicAdd()            atomicAnd()
atomicSub()            atomicOr()
atomicExch()           atomicXor()
atomicMin()
atomicMax()
atomicAdd()
atomicDec()
atomicCAS()
```

See B.10 in the NVIDIA CUDA C Programming Guide

Atomic Functions

- How can threads from different blocks work together?
- Use atomics sparingly. Why?