Atomic Operations

Dmitri Nesteruk
@dnesteruk

pluralsight
hardcore developer training

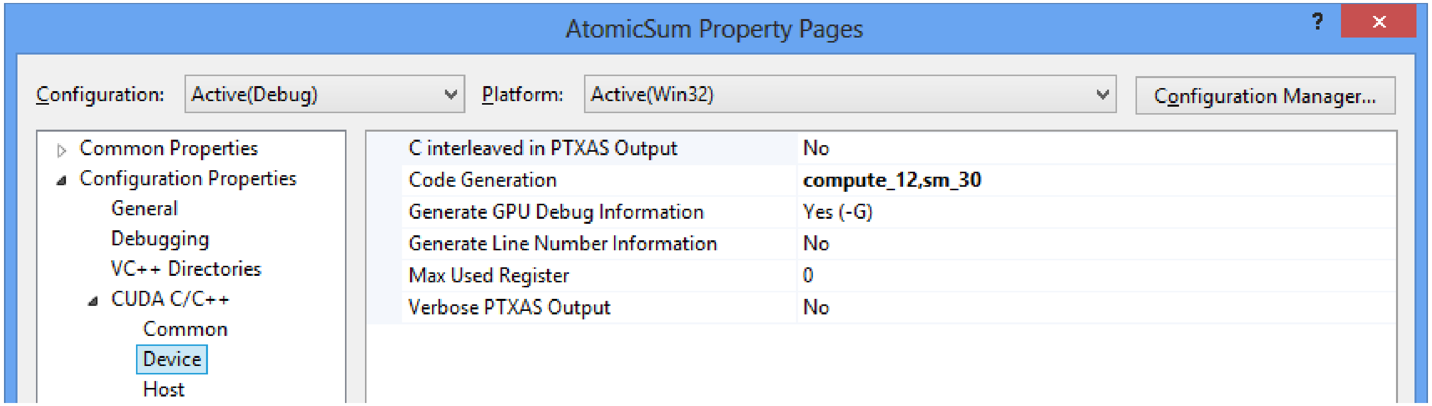
Overview

- Why atomics?
- Atomic Functions
- Atomic Sum
- Monte Carlo Pi

Why Atomics?

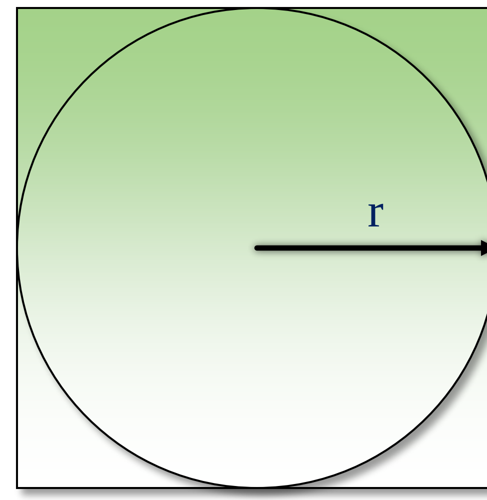
- **x++ is a read-modify-write operation**
  - Read x into a register
  - Increment register value
  - Write register back into x
  - Effectively `{ temp = x; temp = temp+1; x = temp; }`
- **If two threads do x++**
  - Each thread has its own temp (say t1 and t2)
  - ```
    { t1 = x; t1 = t1+1; x = t1;}
    { t2 = x; t2 = t2+1; x = t2;}
    ```
  - Race condition: the thread that writes to x first wins
  - Whoever wins, x gets incremented only *once*
    ```

Atomic Functions

- Problem: many threads accessing the same memory location
- Atomic operations ensure that only one thread can access the location
- Grid scope!
- `atomicOP(x,y)`
  - `t1 = *x;        // read`
  - `t2 = t1 OP y; // modify`
  - `*a = t2;       // write`
- Atomics need to be configured
  - `#include "sm_20_atomic_functions.h“`
  - 

Monte Carlo Pi

- Evaluate the value of π numerically
- Area of circle $S_c = \pi r^2$
- Area of square $S_s = 2r \times 2r = 4r^2$
- $\therefore \pi = \frac{4S_c}{S_s}$
- Generate random points within the square
- Count the number of points in the circle
  - $\sqrt{x^2 + y^2} < r$
- $\pi \approx \frac{4 \times \text{points in circle}}{\text{total number of points}}$



Summary

- **Atomic operations ensure operations on a variable cannot be interrupted by a different thread**
- **CUDA supports several atomic operations**
  - atomicAdd()
  - atomicOr()
  - atomicMin()
  - … and others
- **Atomics incur a heavy performance penalty**