

Automated Dynamic Analysis of CUDA Programs

Michael Boyer, Kevin Skadron*, and Westley Weimer
University of Virginia
{boyer,skadron,weimer}@cs.virginia.edu

* currently on sabbatical with NVIDIA Research



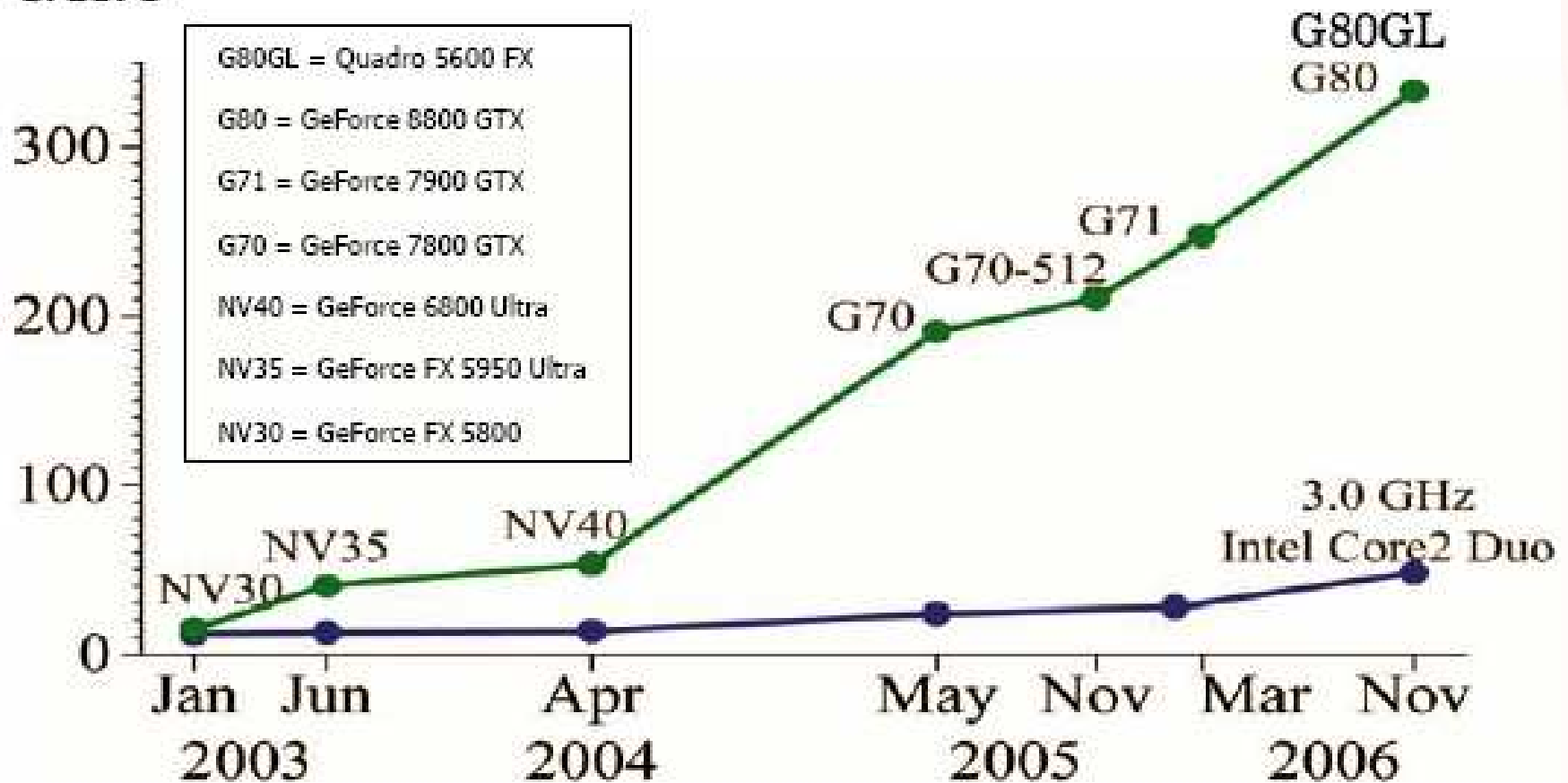
Outline

- GPGPU
- CUDA
- Automated analyses
 - Correctness: race conditions
 - Performance: bank conflicts
- Preliminary results
- Future work
- Conclusion

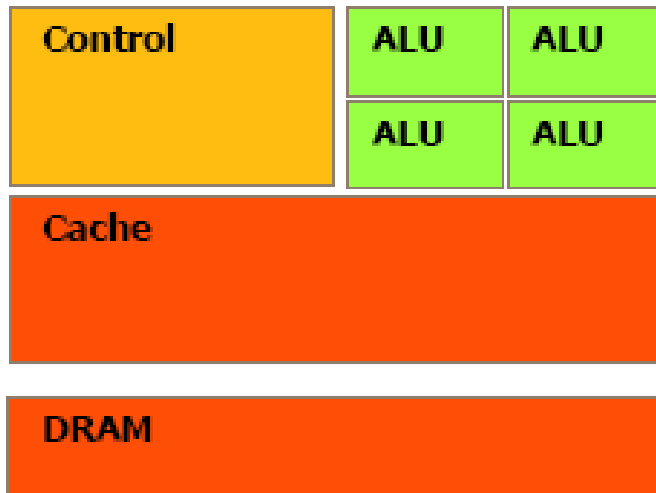


Why GPGPU?

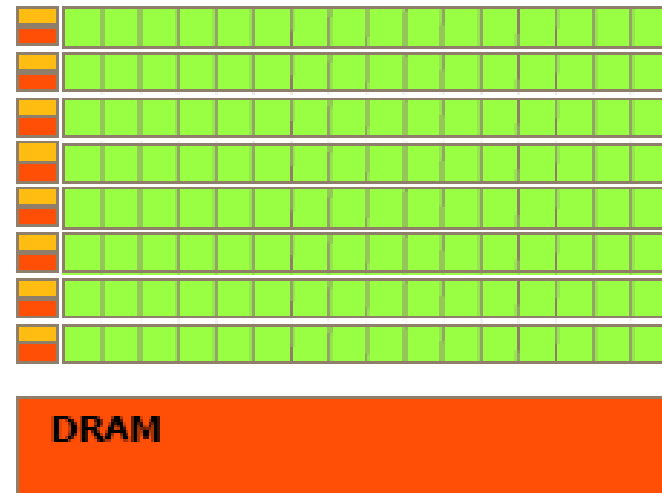
GFLOPS



CPU vs. GPU Design



CPU



GPU

Single-Thread
Latency

Aggregate
Throughput

GPGPU Programming

- Traditional approach: graphics APIs
- ATI/AMD: Close-to-the-Metal (CTM)
- NVIDIA: Compute Unified Device Architecture (CUDA)



CUDA: Abstractions

- Kernel functions
- Scratchpad memory
- Barrier synchronization



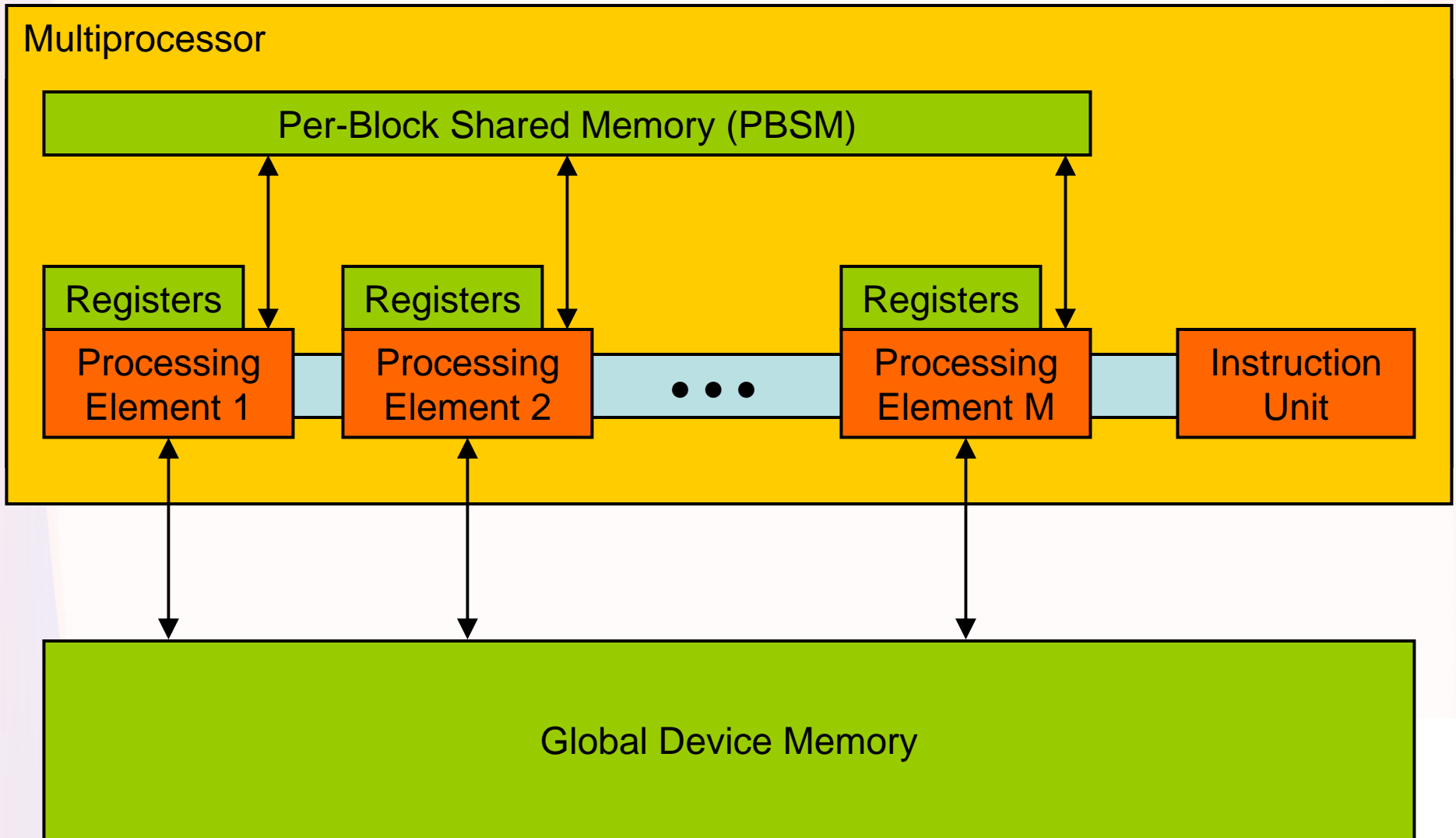
CUDA: Example Program

```
__host__ void example(int *cpu_mem) {  
    cudaMalloc(&gpu_mem, mem_size);  
    cudaMemcpy(gpu_mem, cpu_mem, HostToDevice);  
    kernel <<< grid, threads, mem_size >>> (gpu_mem);  
    cudaMemcpy(cpu_mem, gpu_mem, DeviceToHost);  
}
```

```
__global__ void kernel(int *mem) {  
    int thread_id = threadIdx.x;  
    mem[thread_id] = thread_id;  
}
```



CUDA: Hardware



Outline

- GPGPU
- CUDA
- Automated analyses
 - **Correctness: race conditions**
 - Performance: bank conflicts
- Preliminary results
- Future work
- Conclusion



Race Conditions

- Ordering of instructions among multiple threads is arbitrary
- Relaxed memory consistency model
- Synchronization: `__syncthreads()`
 - Barrier / memory fence

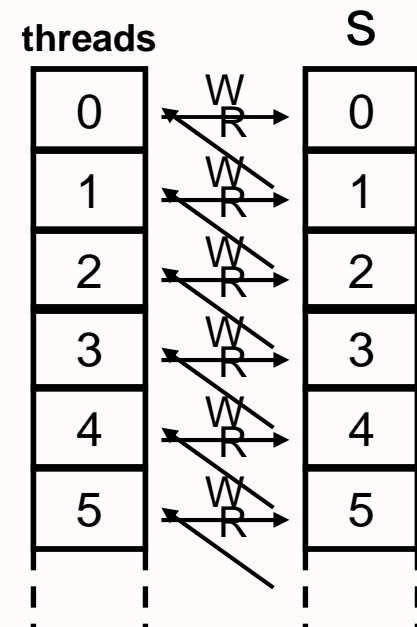


Race Conditions: Example

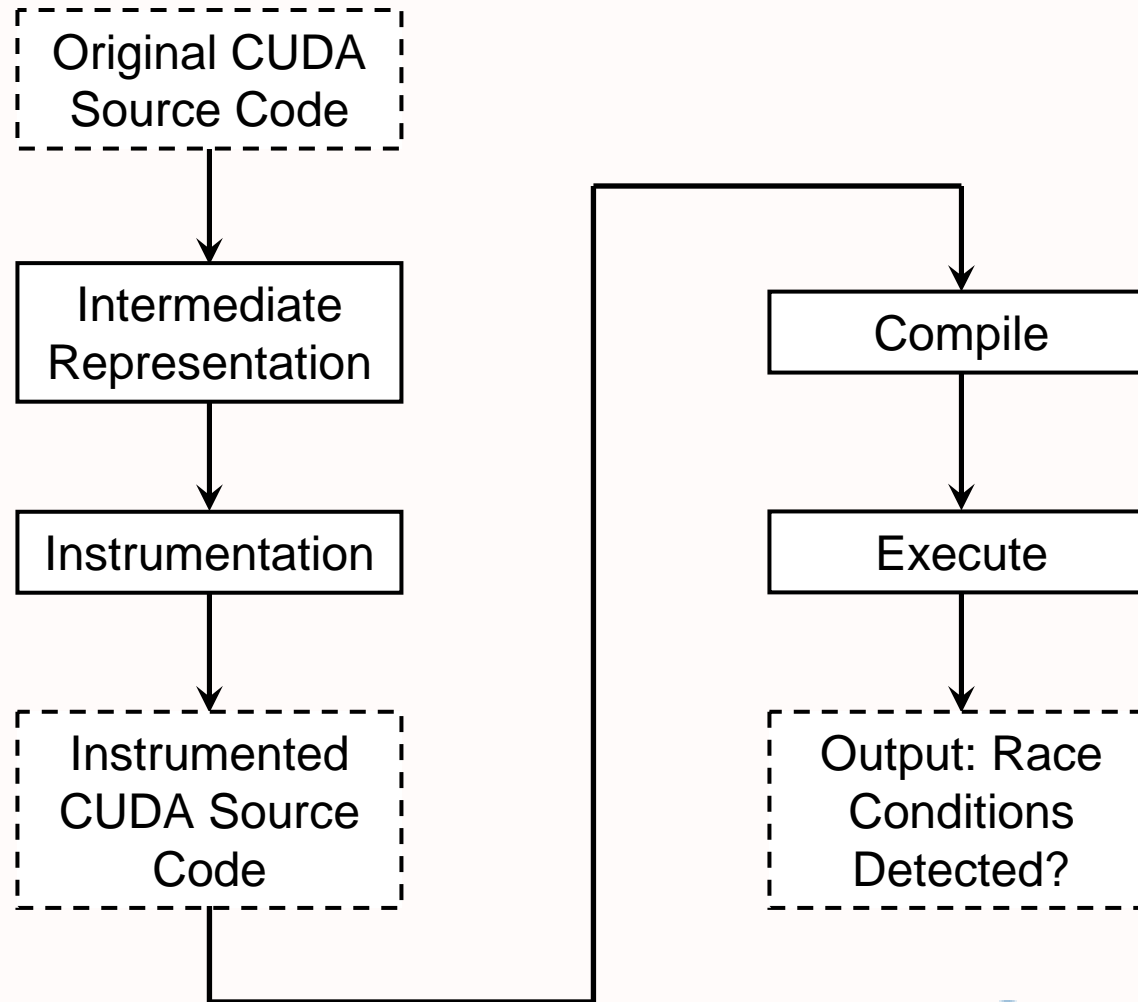
```

1  extern __shared__ int s[ ];
2
3  __global__ void kernel(int *out) {
4      int id = threadIdx.x;
5      int nt = blockDim.x;
6
7      s[id] = id;
8      out = s[(id + 1) % nt];
9  }

```



Automatic Instrumentation



Race Condition Instrumentation

- Two global bookkeeping arrays:
 - Reads & writes of all threads
- Two per-thread bookkeeping arrays:
 - Reads & writes of a single thread
- After each shared memory access:
 - Update bookkeeping arrays
 - Detect & report race conditions



Race Condition Detection

Original code

RAW hazard at expression:

```
#line 8 out[id] = s[(id + 1) % nt];
```

Add synchronization between lines 7 and 8

No race conditions detected



Outline

- GPGPU
- CUDA
- Automated analyses
 - Correctness: race conditions
 - **Performance: bank conflicts**
- Preliminary results
- Future work
- Conclusion

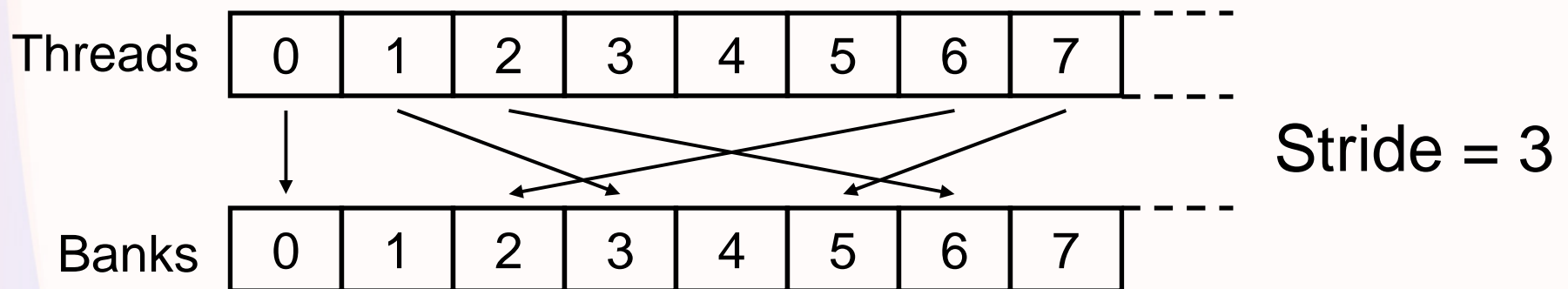
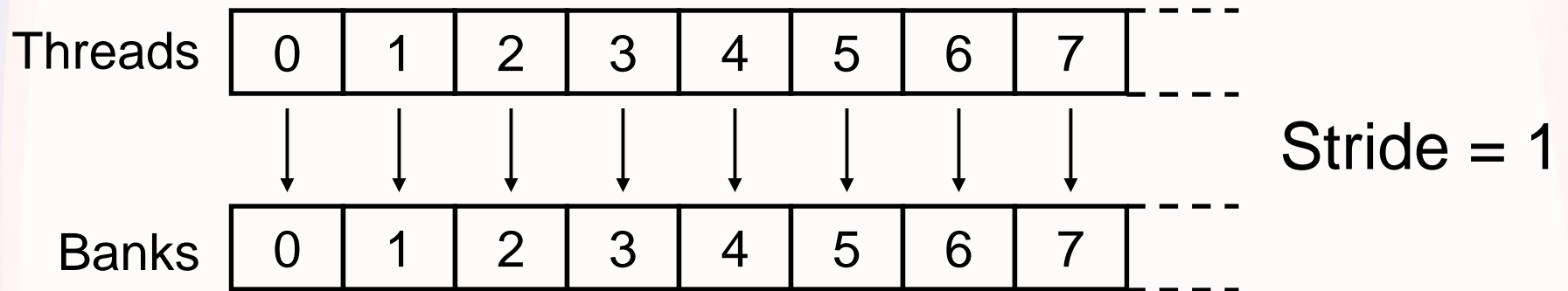


Bank Conflicts

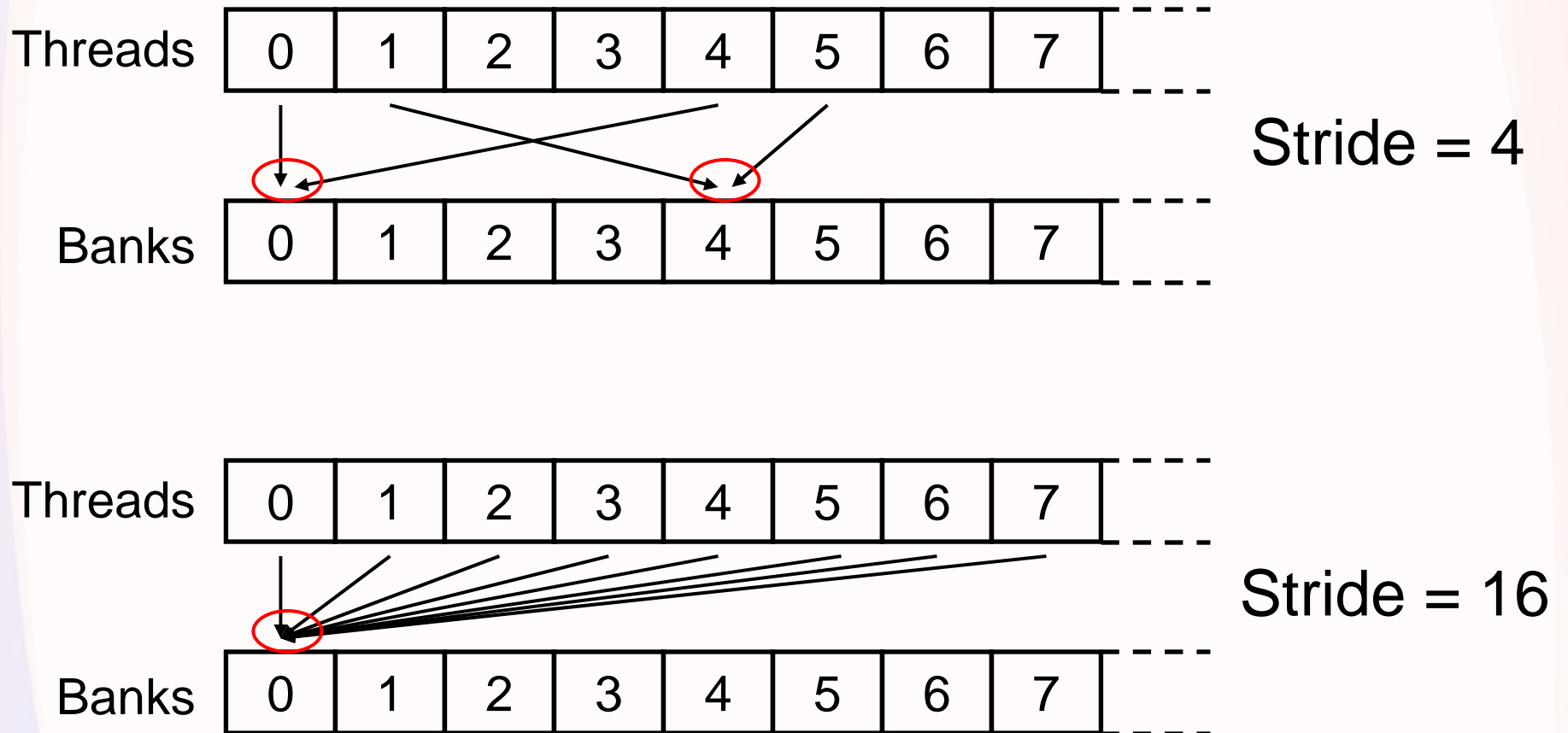
- PBSM is fast
 - Much faster than global memory
 - Potentially as fast as register access
- ...assuming no bank conflicts
 - Bank conflicts cause serialized access



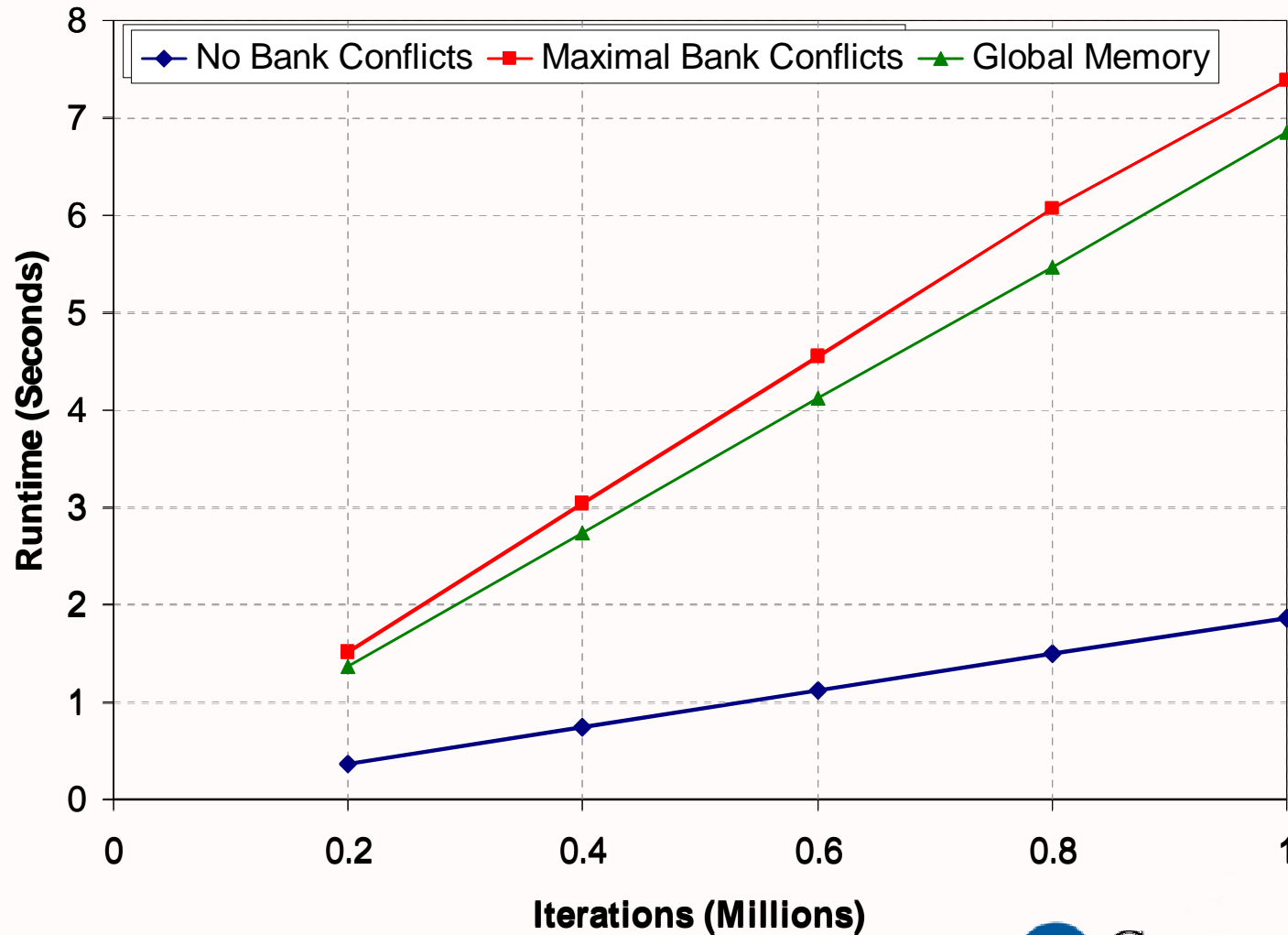
Non-Conflicting Access Patterns



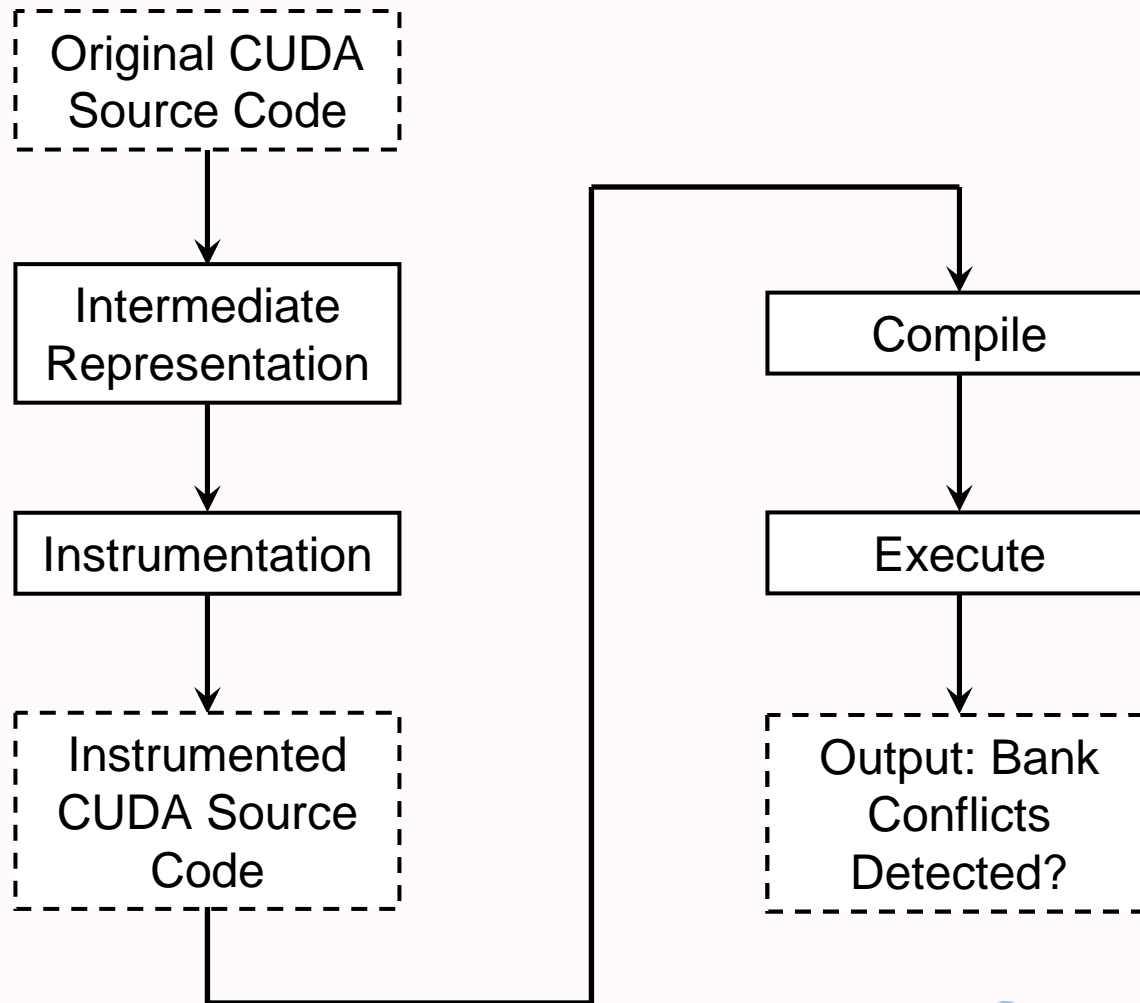
Conflicting Access Patterns



Impact of Bank Conflicts



Automatic Instrumentation



Bank Conflict Instrumentation

- Global bookkeeping array:
 - Tracks address accessed by each thread
- After each PBSM access:
 - Each thread updates its entry
 - One thread computes and reports bank conflicts



Bank Conflict Detection

CAUSE BANK CONFLICTS = false

No bank conflicts at:

```
#line 14 mem[j]++
```

CAUSE BANK CONFLICTS = true

Bank conflicts at: #line 14 mem[j]++

Bank:	0	1	2	3	4	5	6	7	8	9	...
Accesses:	16	0	0	0	0	0	0	0	0	0	...

Preliminary Results

- **Scan**
 - Included in CUDA SDK
 - All-prefix sums operation
 - 400 lines of code
 - Explicitly prevents race conditions and bank conflicts



Preliminary Results: Race Condition Detection

- Original code:
 - No race conditions detected
- Remove any synchronization calls:
 - Race conditions detected



Preliminary Results: Bank Conflict Detection

- Original code:
 - Small number of minor bank conflicts
- Enable bank conflict avoidance macro:
 - Bank conflicts increased!
 - Confirmed by manual analysis
 - Culprit: incorrect emulation mode



Instrumentation Overhead

- Two sources:
 - Emulation
 - Instrumentation
- Assumption: for debugging, programmers will already use emulation mode



Instrumentation Overhead

Code Version	Execution Environment	Average Runtime	Slowdown (Relative to Native)	Slowdown (Relative to Emulation)
Original	Native	0.4 ms		
Original	Emulation	27 ms	62x	
Instrumented (bank conflicts)	Emulation	71 ms	163x	2.6x
Instrumented (race conditions)	Emulation	324 ms	739x	12x



Future Work

- Find more types of bugs
 - Correctness: array bounds checking
 - Performance: memory coalescing
- Reduce instrumentation overhead
 - Execute instrumented code natively



Conclusion

- GPGPU: enormous performance potential
 - But parallel programming is challenging
- Automated instrumentation can help
 - Find synchronization bugs
 - Identify inefficient memory accesses
 - And more...



Questions?

Instrumentation tool will be available at:
<http://www.cs.virginia.edu/~mwb7w/cuda>

