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?
CPU vs. GPU Design

From: NVIDIA CUDA Programming Guide, Version 1.1
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

```c
extern __shared__ int s[ ];
__global__ void kernel(int *out) {
    int id = threadIdx.x;
    int nt = blockDim.x;
    s[id] = id;
    out = s[(id + 1) % nt];
}
```
Automatic Instrumentation

Original CUDA Source Code

Intermediate Representation

Instrumentation

Instrumented CUDA Source Code

Compile

Execute

Output: Race Conditions Detected?
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:

```c
#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

<table>
<thead>
<tr>
<th>Threads</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Banks</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

Stride = 1

<table>
<thead>
<tr>
<th>Threads</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Banks</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

Stride = 3
Conflicting Access Patterns

Threads:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

Banks:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

Stride = 4

Threads:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

Banks:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

Stride = 16
Impact of Bank Conflicts

- No Bank Conflicts
- Maximal Bank Conflicts
- Global Memory
Automatic Instrumentation

1. Original CUDA Source Code
2. Intermediate Representation
3. Instrumentation
4. Instrumented CUDA Source Code
5. Compile
6. Execute
7. Output: Bank Conflicts Detected?
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

<table>
<thead>
<tr>
<th>Code Version</th>
<th>Execution Environment</th>
<th>Average Runtime</th>
<th>Slowdown (Relative to Native)</th>
<th>Slowdown (Relative to Emulation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>Native</td>
<td>0.4 ms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Original</td>
<td>Emulation</td>
<td>27 ms</td>
<td>62x</td>
<td></td>
</tr>
<tr>
<td>Instrumented (bank conflicts)</td>
<td>Emulation</td>
<td>71 ms</td>
<td>163x</td>
<td>2.6x</td>
</tr>
<tr>
<td>Instrumented (race conditions)</td>
<td>Emulation</td>
<td>324 ms</td>
<td>739x</td>
<td>12x</td>
</tr>
</tbody>
</table>
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