GRace: A Low-Overhead Mechanism for Detecting Data Races in GPU Programs

Mai Zheng, Vignesh T. Ravi, Feng Qin, and Gagan Agrawal

Dept. of Computer Science and Engineering
The Ohio State University
GPU Programming Gets Popular

- Many domains are using GPUs for high performance
  
  - **GPU-accelerated** Molecular Dynamics
  - **GPU-accelerated** Seismic Imaging

- Available in both high-end/low-end systems
  - 3 of the **top** 10 supercomputers were based on GPUs [Meuer+:10]
  - Many desktops and laptops are equipped with GPUs
Data Races in GPU Programs

- **A typical mistake**

  1. `__shared__ int s[];
  2. `int tid=threadIdx.x;
  3. `s[tid] = 3; //W
  4. `result[tid] = s[tid+1] * tid; //R

- **May lead to severe problems later**
  - E.g. crash, hang, silent data corruption
Why Data Races in GPU Programs

- **In-experienced programmers**
  - GPUs are accessible and affordable to developers that never used parallel machines in the past

- **More and more complicated applications**
  - E.g. programs running on a cluster of GPUs involve other programming model like MPI (Message Passing Interfaces)

- **Implicit kernel assumptions broken by kernel users**
  - E.g. “max # of threads per block will be 256”,
    “initialization values of the matrix should be within a certain range”,
  - Otherwise, may create overlapped memory indices among different threads
State-of-the-art Techniques

- **Data race detection for multithreaded CPU programs**
  - Lockset [Savage+:97] [Choi+:02]
  - Happens-before [Dinning+:90] [Perkovic+:96] [Flanagan+:09] [Bond+:10]
  - Hybrid [O’Callahan+:03][Pozninansky+:03][Yu+:05]

  Inapplicable or unnecessarily expensive in barrier-based GPU programs 😞

- **Data race detection for GPU programs**
  - SMT(Satisfiability Modulo Theories)-based verification [Li+:10]

    False positives & State explosion 😞

  - Dynamically tracking all shared-variable accesses [Boyer+:08]

    False positives & Huge overhead 😞
Our Contributions

- **Statically-assisted dynamic approach**
  - Simple static analysis *significantly* reduces overhead

- **Exploiting GPU’s thread scheduling and execution model**
  - Identify **key difference** between data race on GPU/CPU
  - Avoid false positives

- **Making full use of GPU’s memory hierarchy**
  - Reduce overhead further

**Precise**: no false positives in our evaluation

**Low-overhead**: as low as 1.2x on real GPU
Outline

- Motivation
- **What’s new in GPU programs**
- GRace
- Evaluation
- Conclusions
Execution Model

- GPU architecture and SIMT (Single-Instruction Multiple-Thread)

- Streaming Multiprocessors (SMs) execute blocks of threads
- Threads inside a block use barrier for synchronization
- A block of threads are further divided into groups called **Warps**
  - 32 threads per warp
  - Scheduling unit in SM
Our Insights

Two different types of data races between barriers

- **Intra-warp races**
  - Threads within a warp can only cause data races by executing the same instruction

- **Inter-warp races**
  - Threads across different warps can have data races by executing the same or different instructions

1. `__shared__` int `s[]`;
2. int `tid` = `threadIdx.x`;
   ...
3. `s[tid]` = 3; //W
4. `result[tid]` = `s[tid+1]` * `tid`; //R
   ...

```
Warp 0  Warp 1  Warp 0
T30 T31 T32 T33
R(s+32) W(s+32) R(s+31) W(s+31)
```

**Inter-warp race**

**Impossible!**

**No Intra-warp race**
Memory Hierarchy

- Memory constraints

- Performance-critical
  - Frequently accessed variables are usually stored in shared memory
  - Dynamic tool should also try to use shared memory whenever possible
Outline

- Motivation
- What’s new in GPU programs
- GRace
- Evaluation
- Conclusions
GRace: Design Overview

- Statically-assisted dynamic analysis

![Diagram of the GRace design overview]

1. Original GPU Code
2. Static Analyzer
   - Statically Detected Bug
3. Annotated GPU Code
4. Dynamic Checker
   - Execute the code
   - Dynamically Detected Bug
5. Original GPU Code
Simple Static Analysis Helps A Lot

**Observation I:**
- Many conflicts can be easily determined by static technique

1. `__shared__` int `s[]`;
2. int `tid`=`threadIdx.x`;
   ...  
3. `s[tid] = 3;` //W
4. `result[tid] = s[tid+1] * tid;` //R
   ...  

<table>
<thead>
<tr>
<th>W</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s[tid]$</td>
<td>($s+0$) ~ ($s+511$)</td>
</tr>
</tbody>
</table>

1. Statically detect certain data races &
2. Prune memory access pairs that cannot be involved in data races

Overlapped!
Static analysis can help in other ways

- **How about this...**

1. `__shared__` float s[];
   ...
2. `for(r=0; ...; ...)`
3. `{ ...
4. `for(c=0; ...; ...)`
5. `{ ...
6. `temp = s[input[c]]; //R`
7. }
8. }

- **Observation II:**
  - Some accesses are loop-invariant
  - `R(s+input[c])` is irrelevant to `r`
  - Don’t need to monitor in every `r` iteration

- **Observation III:**
  - Some accesses are tid-invariant
  - `R(s+input[c])` is irrelevant to `tid`
  - Don’t need to monitor in every thread

3. **Further reduce runtime overhead by identifying loop-invariant & tid-invariant accesses**
**Static Analyzer: Workflow**

1. Race detected!
   - check_loop_invariant(addr, iter_id)
   - check_tid_invariant(addr)

2. No race
   - don’t need to monitor
     - gen_constraints(tid, iter_id, addr)
     - add_params(tid, iter_id, max_tid, max_iter)
     - Linear constraint solver

3. Mark loop-invariant & tid-invariant
   - Mark as monitor at runtime

- pairs of mem accesses
- statically determinable?
GRace: Design Overview

- Statically-assisted dynamic analysis

```
Original GPU Code → Static Analyzer → Annotated GPU Code → Dynamic Checker

Statically Detected Bug

Dynamically Detected Bug

Execute the code
```
Reminder:

- **Intra-warp races**: caused by threads within a warp
- **Inter-warp races**: caused by threads across different warps
Intra-warp Race Detection

- Check conflicts among the threads within a warp
  - Perform detection immediately after each monitored memory access

1. Record access type
2. Record addresses
3. If W, check conflicts in parallel

<table>
<thead>
<tr>
<th>R/W</th>
<th>Address 0</th>
<th>Address 1</th>
<th>Address 2</th>
<th>Address 3</th>
<th>...</th>
<th>Address 31</th>
</tr>
</thead>
<tbody>
<tr>
<td>T0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

warpTable (for each warp)

Recyclable and small enough to reside in Shared Memory

Fast & Scalable
Dynamic Checker

Reminder:
- **Intra-warp races**: caused by threads within a warp
- **Inter-warp races**: caused by threads across different warps

Check **Intra-warp** data races in Shared Memory

Check **Inter-warp** data races:
Two design choices with diff. trade-offs
GRace-stmt: Inter-warp Race Detection I

- Check conflicts among the threads from different warps
  - After each monitored mem. access, record info. to BlockStmtTable
  - At synchronization call, check conflicts between diff. warps

Write & check in parallel

Accurate diagnostic info.
GRace-addr: Inter-warp Race Detection II

- Check conflicts among the threads from different warps
  - After each monitored mem access, update corresponding counters
  - At synchronization call, infer races based on local/global counters

![Diagram showing shared memory and local tables for different warps](image)

Simple & Scalable

Shared Memory

Local tables for Warp-0
(in Device Mem.)

Local tables for Warp-N

Global tables for all Warps within the Block
Outline

- Motivation
- What’s new in GPU programs
- GRace
  - Static analyzer
  - Dynamic checker
- Evaluation
- Conclusions
Methodology

**Hardware**

- **GPU**: NVIDIA Tesla C1060
  - 240 cores (30 × 8), 1.296GHz
  - 16KB shared memory per SM
  - 4GB device memory
- **CPU**: AMD Opteron 2.6GHz × 2
  - 8GB main memory

**Software**

- Linux kernel 2.6.18
- CUDA SDK 3.0
- PipLib (Linear constraint solver)

**Applications**

- co-cluster, em, scan
Overall Effectiveness

- Accurately report races in three applications
- No false positives reported

### Table: Overall Effectiveness

<table>
<thead>
<tr>
<th>Apps</th>
<th>GRace(W/-stmt)</th>
<th>GRace(W/-addr)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R-Stmt#</td>
<td>R-Mem#</td>
</tr>
<tr>
<td>co-cluster</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>em</td>
<td>14</td>
<td>384</td>
</tr>
</tbody>
</table>

3 pairs of racing statements are detected by Static Analyzer.

- **R-Stmt:** pairs of conflicting accesses
- **R-Mem:** memory addresses invoked in data races
- **R-Thd:** pairs of racing threads
- **R-Wp:** pairs of racing warps
- **FP:** false positive
- **RP:** race number reported by B-tool

### Table: Additional Data

<table>
<thead>
<tr>
<th>Apps</th>
<th>B-tool</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RP#</td>
</tr>
<tr>
<td>co-cluster</td>
<td>1</td>
</tr>
<tr>
<td>em</td>
<td>200,445</td>
</tr>
<tr>
<td>scan</td>
<td>Error</td>
</tr>
</tbody>
</table>
Runtime Overhead

- GRace(W/-addr): very modest
- GRace(W/-stmt): higher overhead with diagnostic info., but still faster than previous tool

![Graph showing runtime overhead comparisons between Native, GRace(-stmt), GRace(-addr), and B-tool for co-cluster and em kernels.](image)

- GRace-addr can answer quickly
- GRace-stmt can tell exactly

Is there any data race in my kernel?

Where is it?
## Benefits from Static Analysis

- Simple static analysis can significantly reduce overhead

<table>
<thead>
<tr>
<th>Apps</th>
<th>Without Static Analyzer</th>
<th>With Static Analyzer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Stmt</td>
<td>MemAcc</td>
</tr>
<tr>
<td>co-cluster</td>
<td>10,524,416</td>
<td>10,524,416</td>
</tr>
<tr>
<td>em</td>
<td>19,070,976</td>
<td>54,460,416</td>
</tr>
</tbody>
</table>

**Execution # of monitored statements and memory accesses**

- Stmt: statements
- MemAcc: memory access
Conclusions and Future Work

- **Conclusions**
  - Statically-assisted dynamic analysis
  - Architecture-based approach: Intra/Inter-warp race detection
  - Precise and Low-overhead

- **Future work**
  - Detect races in device memory
  - Rank races

Thanks!