Sound and Partially-Complete Static Analysis of Data-Races in GPU Programs

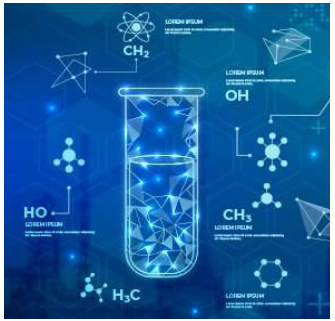
Dennis Liew, University of Massachusetts Boston, USA

Tiago Cogumbreiro, University of Massachusetts Boston, USA

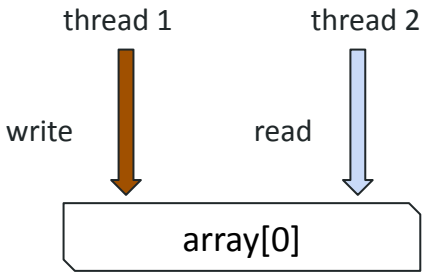
Julien Lange, Royal Holloway, University of London, United Kingdom

Graphics processing units (GPUs) specialize in highly parallel computations.

- used in 3D graphics, machine learning, systems biology, etc.



GPU programs are run by a large number of concurrent threads on multiple layers of shared memory. Failing to reason about correctness leads to concurrency errors, e.g. data-races.



Data-race: two or more threads access the same memory location, and at least one is writing.

Data-races cause:

- **undesired** non-deterministic behavior
- memory corruption
- program crashes

Data-race Freedom Verifier

Verifies the **absence** of data-races in a program.

No data-race exists in all possible program execution paths.

Data-race Finder

Detect the **presence** of data-races in a program.

Finds and reports a program execution that leads to a data-race.

Data-race Freedom Verifier

Property: Data-race Freedom

Soundness: If the tool can prove DRF, then the program must also be DRF.
(No False Negatives)

Completeness: If the program is DRF, then the tool must be able to prove DRF.
(No False Positives)

Data-race Finder

Property: Data-race

Completeness If the tool can prove a data-race, then the program must have a data-race.
(No False Positives)

Soundness If the program has a data-race, then the tool must be able to prove a data-race.
(No False Negatives)

Data-race Freedom Verifier

DRF verifier reports a DRF. ✔
(sound)

DRF verifier reports a data-race. ❓

Can report impossible data-races.
(false alarms, incomplete)

Data-race Finder

Data-race finder reports a data-race. ✔
(complete)

Data-race finder reports DRF. ❓

Can miss data-races.
(false negatives, unsound)

Data-race Freedom Verifier

DRF checking tools are *usually* **static**, without running the program (over-approximates).

Faial [2], GPUVerify [3], PUG [4]

Data-race Finder

Data-race finding tools are *usually* **dynamic**, while running the program (under-approximation).

HiRace [5], HAccRG [6], ScoRD [7]

building upon Faial. (Sound) DRF reports ✔
(Partially-complete) data-race reports ✔ ❓

We present, FaialAA, a **static data-race finder for GPU programs.**
while maintaining the capabilities of a DRF verifier.

- Characterize a class of programs where our analysis **produce only true alarms** (59.5% of programs).

- Formally establish the root causes of false alarms in FaialAA.
  i.e, determine the sources of approximation.

1. **Empirical Results for FaialAA**
   - Can FaialAA verify bug-fixes in open source projects?
   - Does FaialAA have fewer false alarms than the state-of-the-art?

2. **Theoretical Results** (mechanized 15,900 lines of Coq code )
   - Control-flow Independence & Data Independence
   - True Positives Theorem

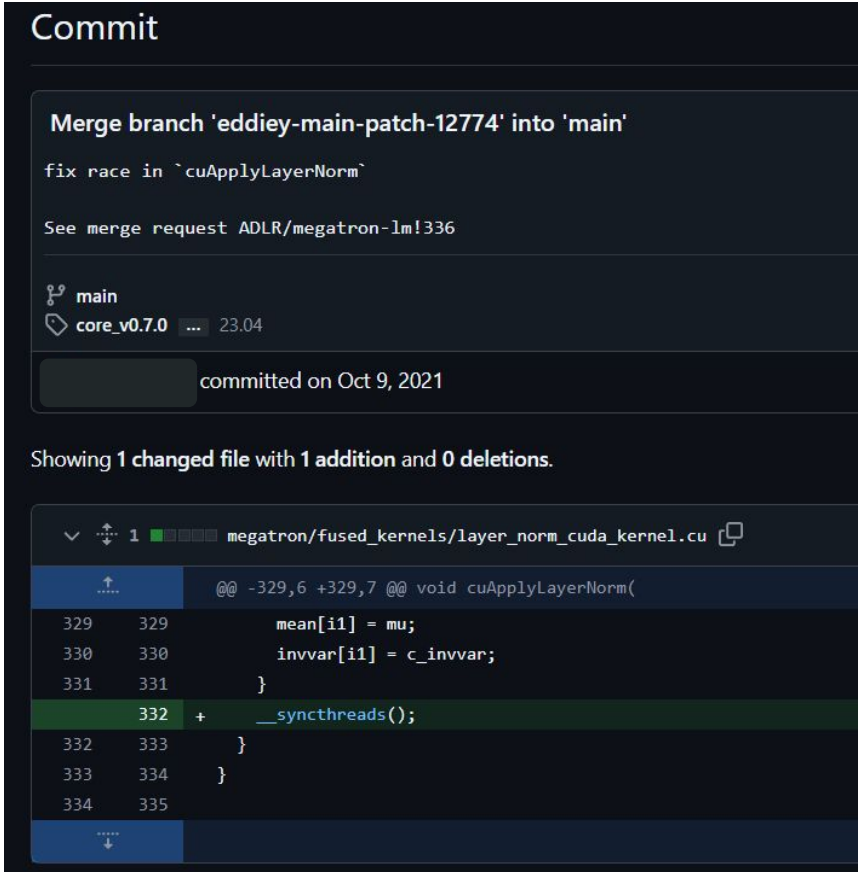
Can FaialAA verify bug-fixes in open source projects?

Data Selection:

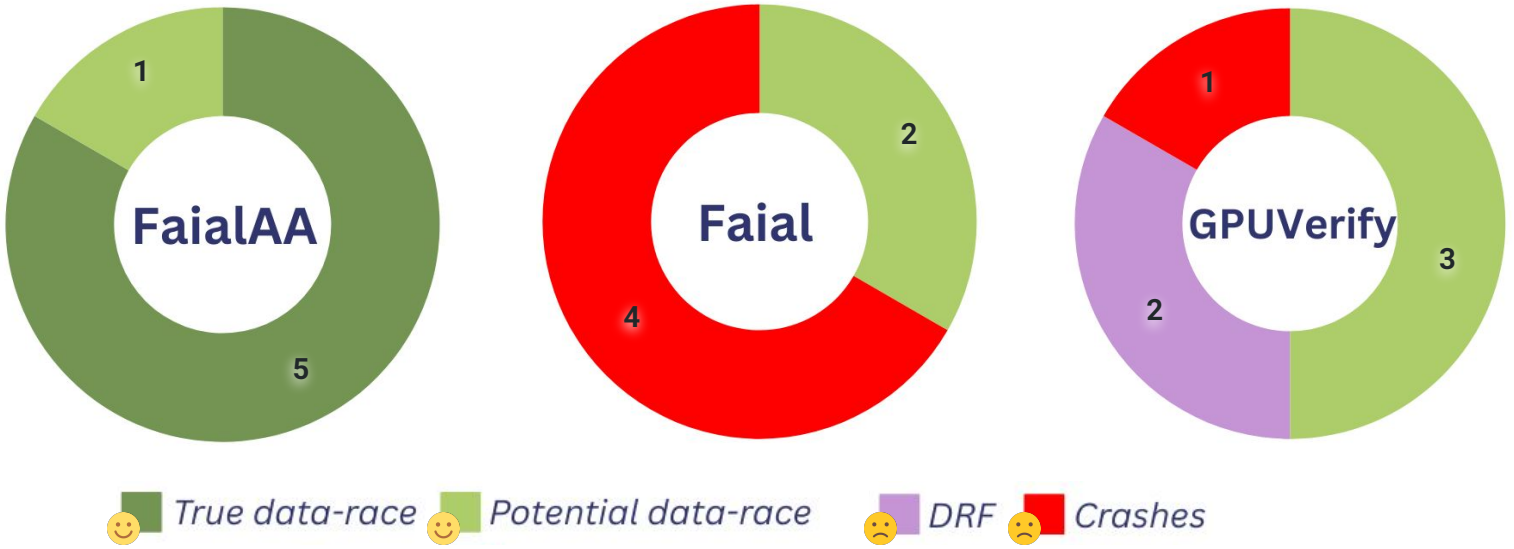
- 6 bug-fixing commits.
  - OpenMM, high performance library used for molecular simulation
  - Megatron-LM, library for training transformer models in LLMs by Nvidia

Experiment 1: Find bugs in buggy version

Experiment 2: Check DRF in fixed version

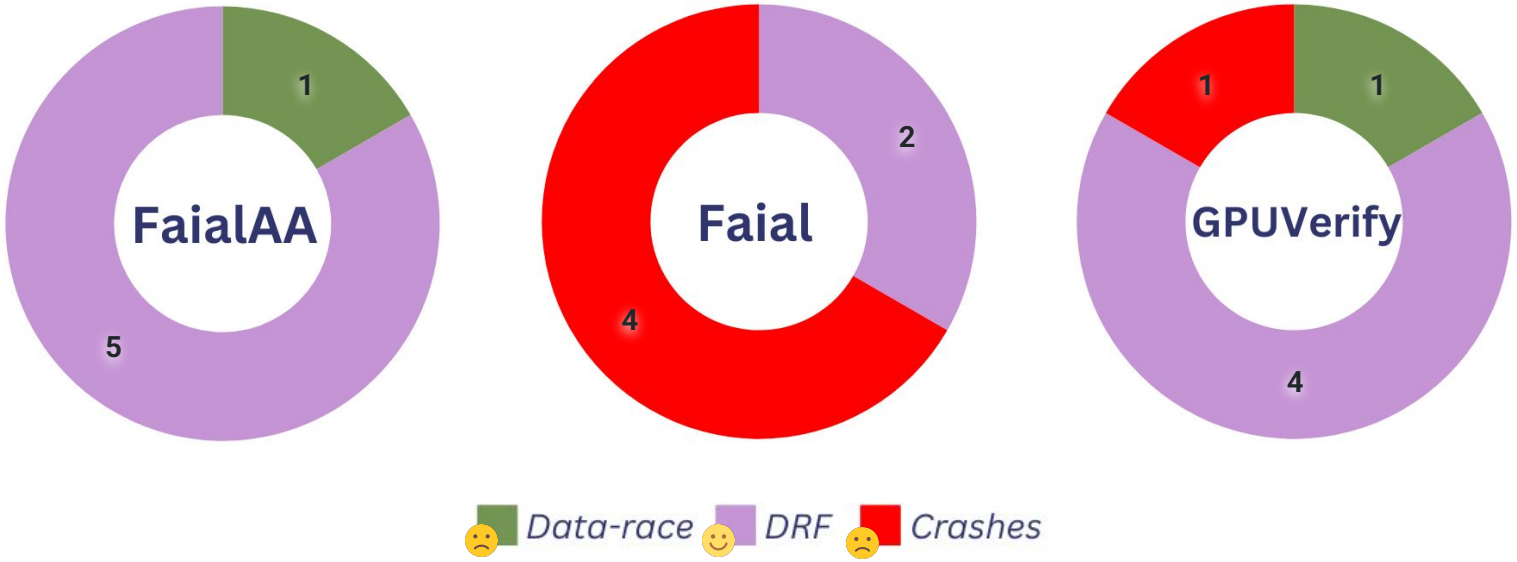


Racy (buggy) version



Only FaialAA is able to report confirm true data-race reports.

DRF (fixed) version



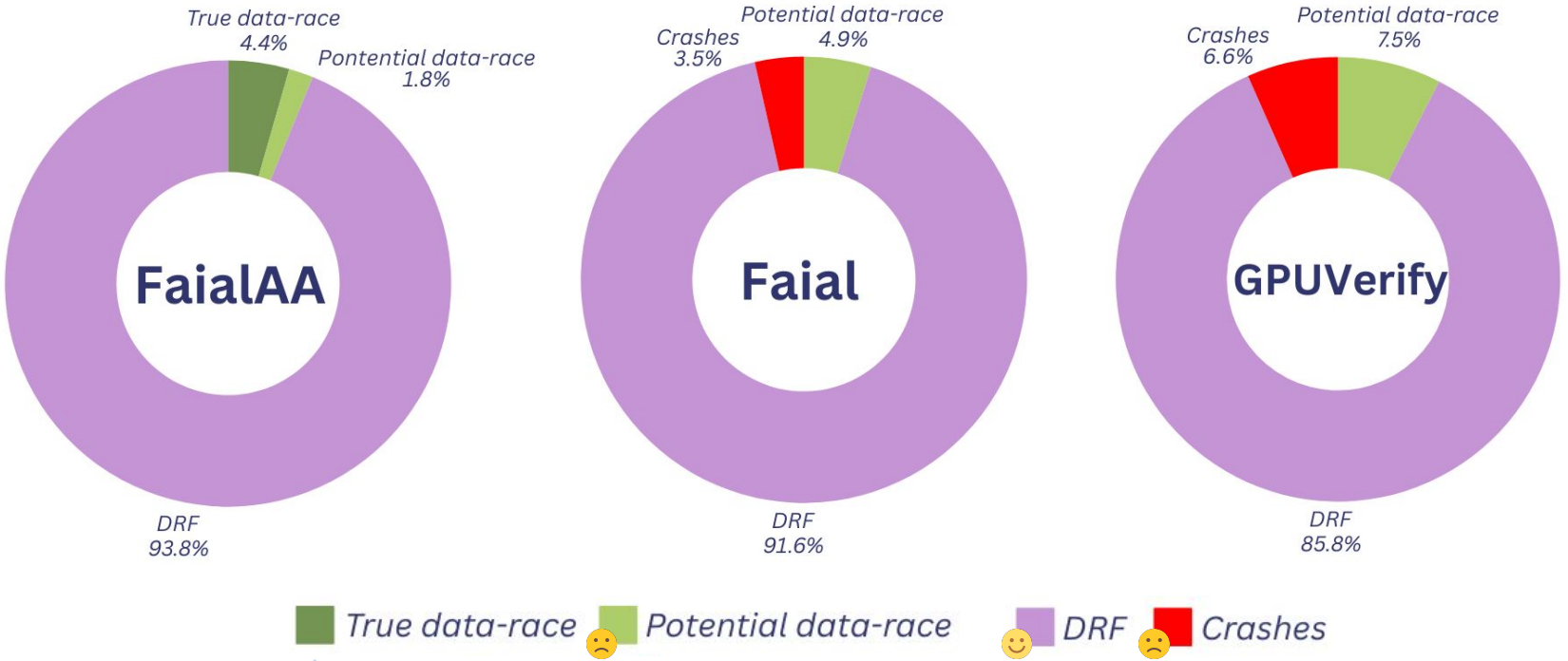
FaialAA confirms 5 out of 6 DRF programs.

Does FaialAA have fewer false alarms than the state-of-the-art?

Data Selection:

- 226 **reportedly DRF** programs
  - NVIDIA GPU Computing SDK v2.0
  - NVIDIA GPU Computing SDK v5.0
  - Microsoft C++ AMP Sample Projects
  - Gpgpu-sim benchmarks
- Reproduced experiment by Bardsley *et al.* CAV'14 [1].

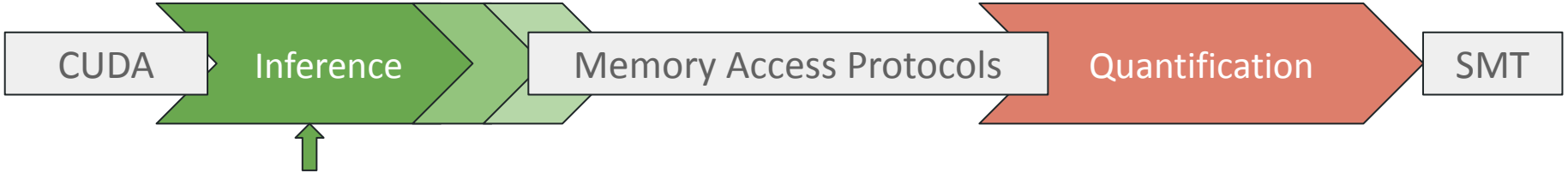
Experiment: Verify programs and report on outcomes of tools.



FaialAA was able to find **10 undocumented racy programs**, 6 of which are missed by both Faial and GPUVerify.

FaialAA reports **1.9× fewer potential alarms.**

Formalizing sources of False Alarms



- Memory Access Protocols codify the behavior of threads over shared memory.
- For scalability, Memory Access Protocols abstracts away what is **written to/read from** arrays [4].
- Contents of arrays are lost in the inference from CUDA to MAPs.

CUDA

```
__global__ void CIDIKernel(int* A) {
        int j = A[thread_id];
        A[thread_id] = random_int;
        }
```

Control-flow Independent (CI) protocol &
Data Independent (DI) protocol.

The inference process is exact (no approximations). The contents of array A, is independent to our analysis.

```
__global__ void CDKernel(int* A) {
        int j = A[thread_id];
        if (j > 0) {
                A[thread_id] = random_int;
        }
```

Control-flow Dependent (CD) protocol.

Write access may or may not execute. The reachability is inexact.

```
__global__ void DDKernel(int* A) {
        int j = A[thread_id];
        A[j] = random_int;
        }
```

Data Dependent (DD) protocol.

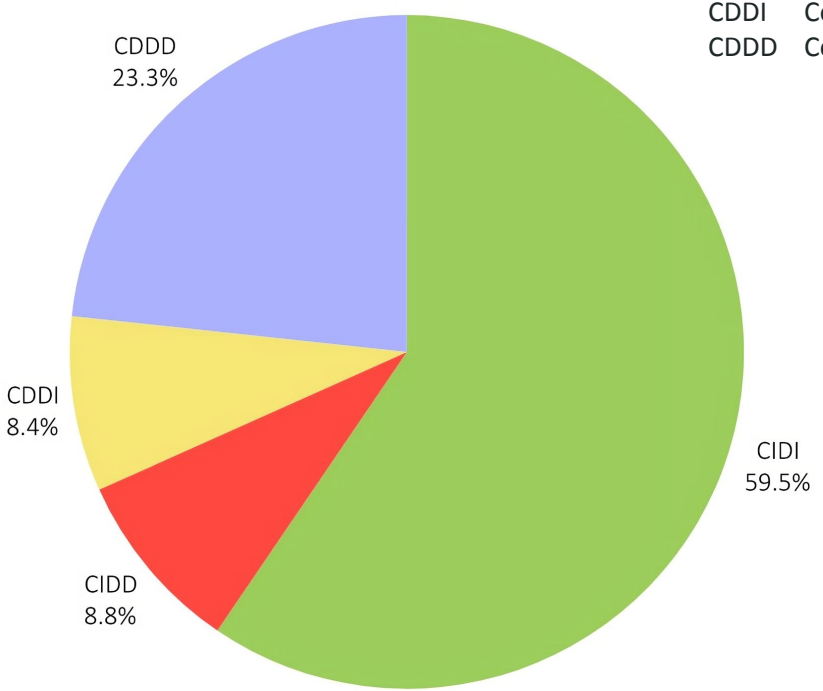
The reachability is exact, but the index location is inexact.

How common are CIDI programs?

We used approximation analysis to analyze 2770 programs from two datasets.

- CAV'14 (226 programs)
- GH'22 (2544 programs)
    - CUDA programs retrieved from GitHub's Search API

CIDI Control-flow Independent & Data Independent
CIDD Control-flow Independent & Data Dependent
CDDI Control-flow Dependent & Data Independent
CDDD Control-flow Dependent & Data Dependent



59.5% of the 2770 programs are CIDI.

$$X \vdash i \qquad X \vdash \mathsf{tid} \qquad \frac{x \in X}{X \vdash x} \qquad \frac{X \vdash e_1 \qquad X \vdash e_2}{X \vdash e_1 \star e_2} \qquad X \vdash b$$

$$\boxed{X \vdash e} \qquad \boxed{X \vdash c}$$

$$\frac{X \vdash e_1 \qquad X \vdash e_2}{X \vdash e_1 \diamond e_2} \qquad \frac{X \vdash c_1 \qquad X \vdash c_2}{X \vdash c_1 \circ c_2}$$

$$\boxed{X \vdash_\alpha p \quad \alpha \in \{\mathtt{CI}, \mathtt{DI}\}}$$

I-SKIP

$$\frac{}{X \vdash_\alpha \mathsf{skip}}$$

I-SEQ

$$\frac{X \vdash_\alpha p_1 \qquad X \vdash_\alpha p_2}{X \vdash_\alpha p_1 \; ; \; p_2}$$

I-VAR

$$\frac{X \vdash_\alpha p}{X \vdash_\alpha \mathsf{var}\, x.\, p}$$

$$\boxed{X \vdash_{\mathtt{CI}} p}$$

CI-ACC

$$\frac{}{X \vdash_{\mathtt{CI}} o[e]}$$

CI-IF

$$\frac{X \vdash c \qquad X \vdash_{\mathtt{CI}} p_1 \qquad X \vdash_{\mathtt{CI}} p_2}{X \vdash_{\mathtt{CI}} \mathsf{if}\ c\ \{p_1\}\ \mathsf{else}\ \{p_2\}}$$

CI-FOR

$$\frac{X \vdash e_1 \qquad X \vdash e_2 \qquad X \cup \{x\} \vdash_{\mathtt{CI}} p}{X \vdash_{\mathtt{CI}} \mathsf{for}\ x \in e_1..e_2\ \{p\}}$$

Approximation analysis: *Control-flow Independent* (CI) protocols

states that the control-flow in protocol *p* is unaffected by symbolic variables.

$$X \vdash i \qquad X \vdash \mathtt{tid} \qquad \frac{x \in X}{X \vdash x} \qquad \frac{X \vdash e_1 \qquad X \vdash e_2}{X \vdash e_1 \star e_2} \qquad X \vdash b \qquad \boxed{X \vdash e} \qquad \frac{X \vdash e_1 \qquad X \vdash e_2}{X \vdash e_1 \diamond e_2} \qquad \boxed{X \vdash c} \qquad \frac{X \vdash c_1 \qquad X \vdash c_2}{X \vdash c_1 \circ c_2}$$

$$\boxed{X \vdash_\alpha p \quad \alpha \in \{\mathtt{CI}, \mathtt{DI}\}}$$

$$\text{I-SKIP}\ \frac{}{X \vdash_\alpha \mathtt{skip}} \qquad \text{I-SEQ}\ \frac{X \vdash_\alpha p_1 \qquad X \vdash_\alpha p_2}{X \vdash_\alpha p_1 \ ; \ p_2} \qquad \text{I-VAR}\ \frac{X \vdash_\alpha p}{X \vdash_\alpha \mathtt{var}\ x.\ p}$$

$$\boxed{X \vdash_{\mathtt{DI}} p}$$

$$\text{DI-ACC}\ \frac{X \vdash e}{X \vdash_{\mathtt{DI}} o[e]} \qquad \text{DI-IF}\ \frac{X \vdash_{\mathtt{DI}} p_1 \qquad X \vdash_{\mathtt{DI}} p_2}{X \vdash_{\mathtt{DI}} \mathtt{if}\ c\ \{p_1\}\ \mathtt{else}\ \{p_2\}} \qquad \text{DI-FOR-D}\ \frac{X \vdash_{\mathtt{DI}} p}{X \vdash_{\mathtt{DI}} \mathtt{for}\ x \in e_1..e_2\ \{p\}} \qquad \text{DI-FOR-I}\ \frac{X \vdash e_1 \qquad X \vdash e_2 \qquad X \cup \{x\} \vdash_{\mathtt{DI}} p}{X \vdash_{\mathtt{DI}} \mathtt{for}\ x \in e_1..e_2\ \{p\}}$$

Approximation analysis: *Data Independent* (DI) protocols

states that the indexing expressions in protocol *p* is unaffected by non-deterministic variable declarations**.**

THEOREM 4.3 (ACTION-SET CORRESPONDENCE). *The following propositions hold.*

(1) $J\text{-}actions(s) \equiv_{\mathcal{I}} P\text{-}actions([[s]]) \cap reachable\text{-}actions(s)$.
(2) *If* $\emptyset \vdash_{\mathtt{CI}} [[s]]$, *then* $J\text{-}actions(s) \equiv_{\mathcal{I}} P\text{-}actions([[s]])$.
(3) *If* $\emptyset \vdash_{\mathtt{DI}} [[s]]$, *then* $J\text{-}actions(s) = P\text{-}actions([[s]]) \cap reachable\text{-}actions(s)$.
(4) *If* $\emptyset \vdash_{\mathtt{CI}} [[s]]$ *and* $\emptyset \vdash_{\mathtt{DI}} [[s]]$, *then* $J\text{-}actions(s) = P\text{-}actions([[s]])$.

(1) Any access in the program, is also in MAPs. (soundness)

(2) If MAPs is control-flow independent, we know that the accesses are **reachable** in the program, but the **index is imprecise.**

(3) If MAPs is data independent, then the accesses have **precise index**, but may be **unreachable** in the program.

(4) If MAPs is both control-flow and data independent, then the accesses are exactly **equal** to the CUDA program.

True Positives Theorem

THEOREM 4.5 (TRUE POSITIVES). *Let* $\emptyset \vdash_{\texttt{CI}} [[s]]$, $\emptyset \vdash_{\texttt{DI}} [[s]]$, *and* $\text{datarace}(\delta_1, \delta_2)$.
If $\delta_1 \in$ *P-actions*$([[s]])$ *and* $\delta_2 \in$ *P-actions*$([[s]])$, *then* $\delta_1 \in$ *J-actions*(s) *and* $\delta_2 \in$ *J-actions*(s).

If MAPs is CIDI, then data-race emitted by MAPS, must also be emitted by the CUDA program.

CIDI programs are the class of programs where our analysis **produce only true alarms.**

Conclusion

Approximation analysis a static analysis technique to detect true data-races in GPU programs.

- Assigns two dimensions of preciseness (CI and DI) to memory access protocols.

True Positive Theorem identifies the class of programs where our analysis only reports true alarms.

Implemented in FaialAA, static sound DRF checker and partially-complete data-race finder.

Our evaluation showed that:

- Emits 1.9× fewer potential alarms
- Found 10 undocumented data-races in a well-studied DRF dataset
- Confirms 5 pairs of racy and fixed programs from open source project
- Has exact analysis on 59.5% of programs.

References

[1] Ethel Bardsley, Adam Betts, Nathan Chong, Peter Collingbourne, Pantazis Deligiannis, Alastair F. Donaldson, Jeroen Ketema, Daniel Liew, and Shaz Qadeer. 2014. Engineering a Static Verification Tool for GPU Kernels. In Proceedings of CAV, Vol. 8559. Springer, Berlin, Heidelberg, 226–242. https://doi.org/10.1007/978-3-319-08867-9_15

[2] Tiago Cogumbreiro, Julien Lange, Dennis Liew Zhen Rong, and Hannah Zicarelli. 2023. Memory Access Protocols: Certified Data-Race Freedom for GPU Kernels. FMSD (2023), 38 pages. https://doi.org/10.1007/s10703-023-00415-0

[3] Adam Betts, Nathan Chong, Alastair F. Donaldson, Shaz Qadeer, and Paul Thomson. 2012. GPUVerify: a Verifier for GPU Kernels. In Proceedings of OOPSLA. ACM, New York, NY, USA, 113–132. https://doi.org/10.1145/2384616.2384625

[4] Guo dongLi and Ganesh Gopalakrishnan. 2010. Scalable SMT-based verification of GPU kernel functions. In Proceedings of FSE. ACM, New York, NY, USA, 187–196. https://doi.org/10.1145/1882291.1882320

[5] John Jacobson, Martin Burtscher, and Ganesh Gopalakrishnan. 2024. HiRace: Accurate and Fast Data Race Checking for GPU Programs. In Proceedings of SC. IEEE, Piscataway, NJ, USA, 12 pages.

[6] Anup Holey, Vineeth Mekkat, and Antonia Zhai. 2013. HAccRG: Hardware-Accelerated Data Race Detection in GPUs. In Proceedings of ICPP. IEEE, Piscataway, NJ, USA, 60–69. https://doi.org/10.1109/ICPP.2013.15

[7] Aditya K.Kamath,AlvinA.George,and Arkaprava Basu.2020. ScoRD:AScopedRaceDetectorforGPUs.InProceedings of ISCA. IEEE, Piscataway, NJ, USA, 1036–1049. https://doi.org/10.1109/ISCA45697.2020.00088