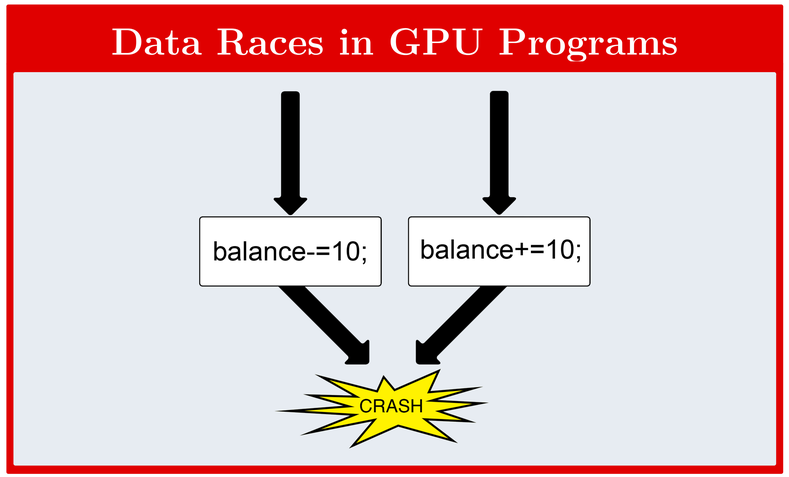
# Gklee$_{pp}$: Parallelizing a symbolic GPU race checker

Student: Mark Baranowski, Advisor: Ganesh Gopalakrishnan
University of Utah, School of Computing

SCHOOL OF COMPUTING
THE UNIVERSITY OF UTAH

## Data Races in GPU Programs

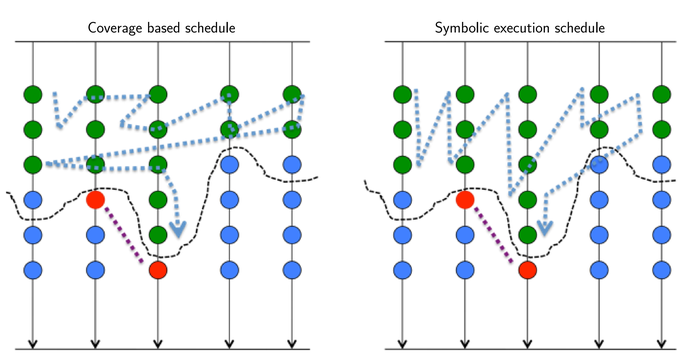



## Effects of Data Races

Data races in the best case will crash your program. In the worst case they will silently lead to corrupt results. Gklee$_{pp}$ can detect such errors and produce traces to diagnose the cause.

## Race Detection Approaches

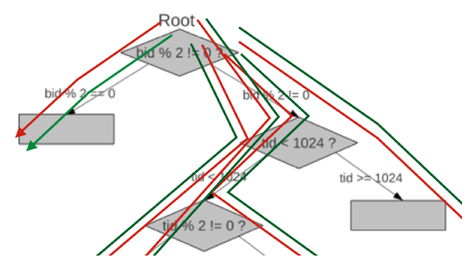
- Conventional race checking tools use tests to expose programming errors. These tools are reliant on the quality of code coverage and may not expose bugs which depend on certain schedules.
- Symbolic tools, such as Gklee$_{pp}$, can detect races with little user intervention. Gklee$_{pp}$ is also scheduler insensitive, as a canonical schedule can detect a race.




## Symbolic Execution Approaches

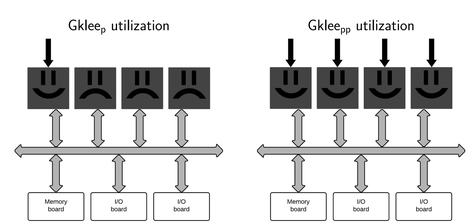
- Full Symbolic Execution
  - Models all threads
  - Very slow when checking many threads
- Parametric Execution (used by Gklee$_{pp}$)
  - Take advantage of symmetry between threads modeling threads as 'parametric flows'
  - Scalable to more threads

Representation of parametric flows




## Disadvantage of Symbolic Execution

- Slower than instrumentation based approaches
- Most time spent solving SMT queries
  - SMT solvers are difficult to parallelize
- Symbolic execution is done in series




## Potential Parallelization Approaches and Challenges

We identified two approaches to adding parallel execution to Gklee$_p$:

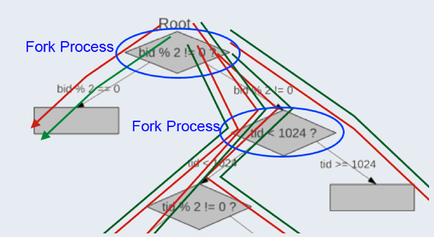
- Execute parametric flows in parallel
  - Allows a less time consuming part of the execution to be done in parallel
  - If we restrict Gklee$_{pp}$ to checking races only within a flow, we get parallelized race checking as well
- Execute queries to the solver in parallel
  - Parallelizes the most time consuming portion of execution
  - This is proved difficult to do with threads due to the structure of the used SMT solver, STP
    - The Gklee$_p$ code base is also very difficult to parallelize due to its reliance on shared data

## Parallelization Approach

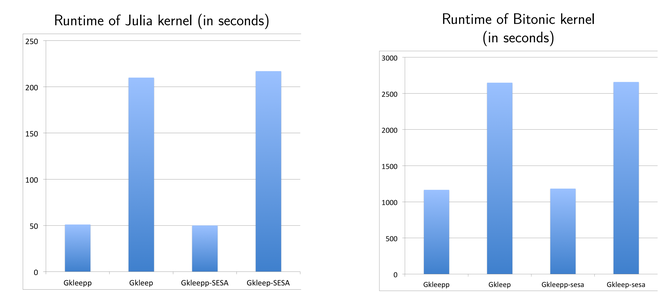
We chose to implement parallel parametric flow execution.

- Fork a process when flows fork
- Similar to the approach used by Cloud9[1]
- Races are checked per flow

Forking Flows




## Results




We measured a speed up of a factor of 2-8 in testing. The disparity is due to some forked flows exiting earlier than others.

## Future Work

- We plan to add support for communicating changes in forked processes
- Implement better load balancing for parallel flow execution
- A solver server for parallelized race checking
- We are investigating using Swift/T to execute solver instances
  - Swift/T automates load balancing and ensures dependencies are met before launching processes

## References

[1] Liviu Ciortea, Cristian Zamfir, Stefan Bucur, Vitaly Chipounov, and George Candea. Cloud9: A software testing service. *ACM SIGOPS Operating Systems Review*, 43(4):5–10, 2010.