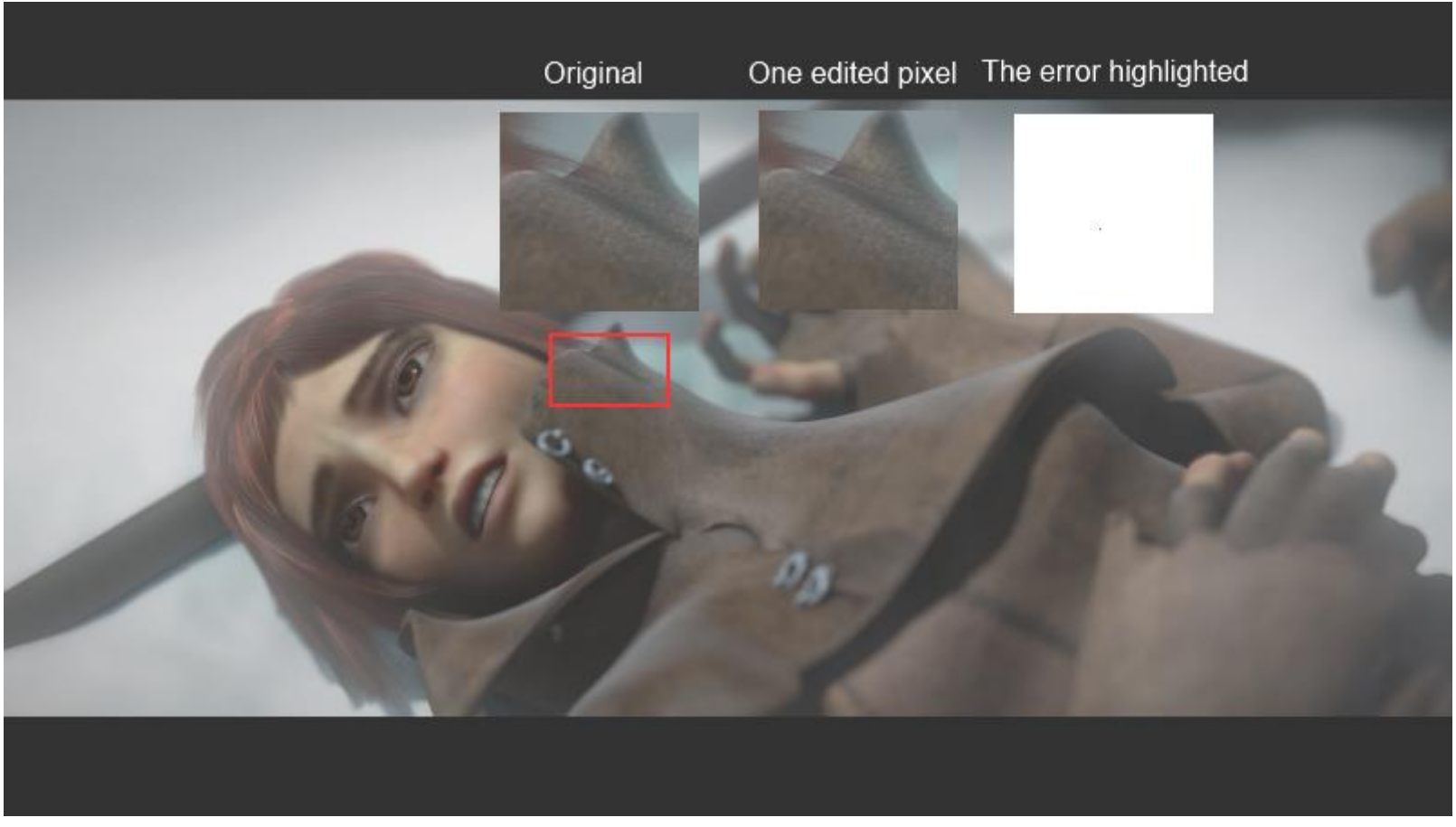
Fault Injection into GPGPU-Applications using GPU-Qin

Anne Gropler, Hasso-Plattner-Institute
Prof. Dr. Andreas Polze, Lena Herscheid, M.Sc., Daniel Richter, M.Sc.
Seminar: Fault Injection
25/06/15

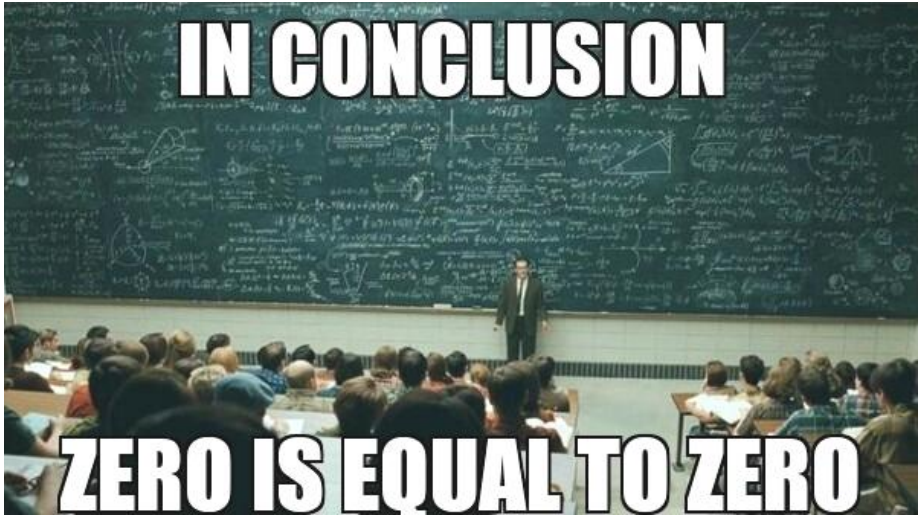
Motivation



Motivation

GPGPU Applications:

- DNA-Sequencing
- Simulations
- Linear Algebra
- Cryptography & Cryptanalysis



Permanent Hardware Faults

Causes:

- design faults
- manufacturing faults (single, series)

Example:

- Pentium-FDIV-Bug (Intel, 1994)

Transient Hardware Faults

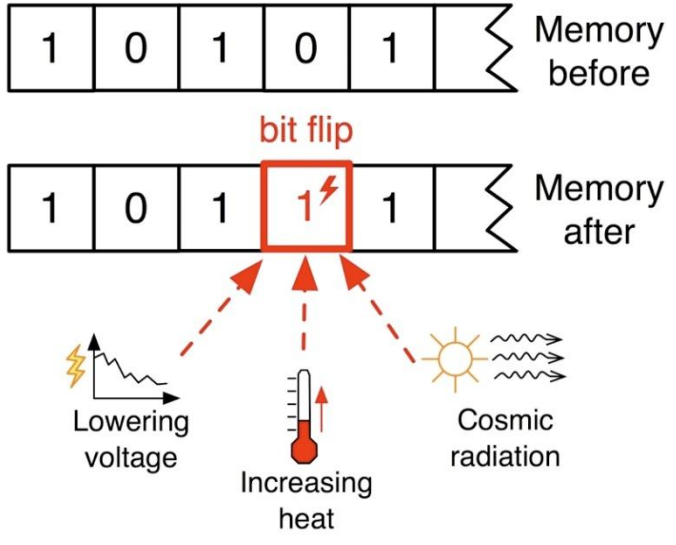
Occurrence: non-deterministically

- single and multi bit flip
- stuck at 0

Cause: external events

- cosmic rays
- over-heated components
- electrostatic discharge

Increasing rate of occurrence



Transient Hardware Faults

NVIDIA GPUs support error correction code (ECC) for

- register files
- DRAM
- cache
- on-chip memory space

They can also occur in functional units (ALU, LSU),
then propagate to registers and/or memory!

Outcomes

Benign outcomes:

- Error occured, but no failure

Failure outcomes:

- Crash: hardware exceptions
- Hangs: Timeout, infinite loop
- SDC: silent data corruptions, incorrect output (there might be no indication that something went wrong!)

$$\frac{4{,}195{,}835}{3{,}145{,}727} = 1.3338204491362410 02 \qquad \frac{4{,}195{,}835}{3{,}145{,}727} = 1.3337390689020375 89$$

GPU-Qin

Investigate error-resilience by performing fault-injection

Error-resilience:

- conditional property of the program not experiencing a failure given that a fault has occured

Long-time goal: develop fault-tolerance mechanisms

- application-specific
- software-based

Challenge: massive parallelism

- Representative coverage of execution paths
- Time-efficiency

Fault Model

Assumption: cache, memory, register files are protected by ECC (e.g. NIVIDIA Fermi GPU)

Simulation of transient hardware faults in functional units of the GPU processor (ALU, LSU)

What to inject?

- Single bit flips
- Multi bit flips (supported, but evaluation is future work)

GPU-Qin

Profiler & Fault Injector

Based on CUDA

- performed on same hardware platform: NVIDIA
- error resilience becomes property of the app alone
- SIMT: single instruction / multiple thread
- cuda-gdb: CUA GPU debugging tool

GPU-Qin: Methodology

Requirements (1 of 3)

Representativeness:

- injected faults should be *representative of the actual hardware faults* that occur at *runtime*
- faults should be injected *uniformly* over the *set of all instructions* executed by the application

GPU-Qin: Methodology

Requirements (2 of 3)

Efficiency:

- fault injection should be *fast enough* to allow the application to be executed to completion in *reasonable time*
- *statistically significant* estimates of error resilience needs thousands of fault-injection experiments!

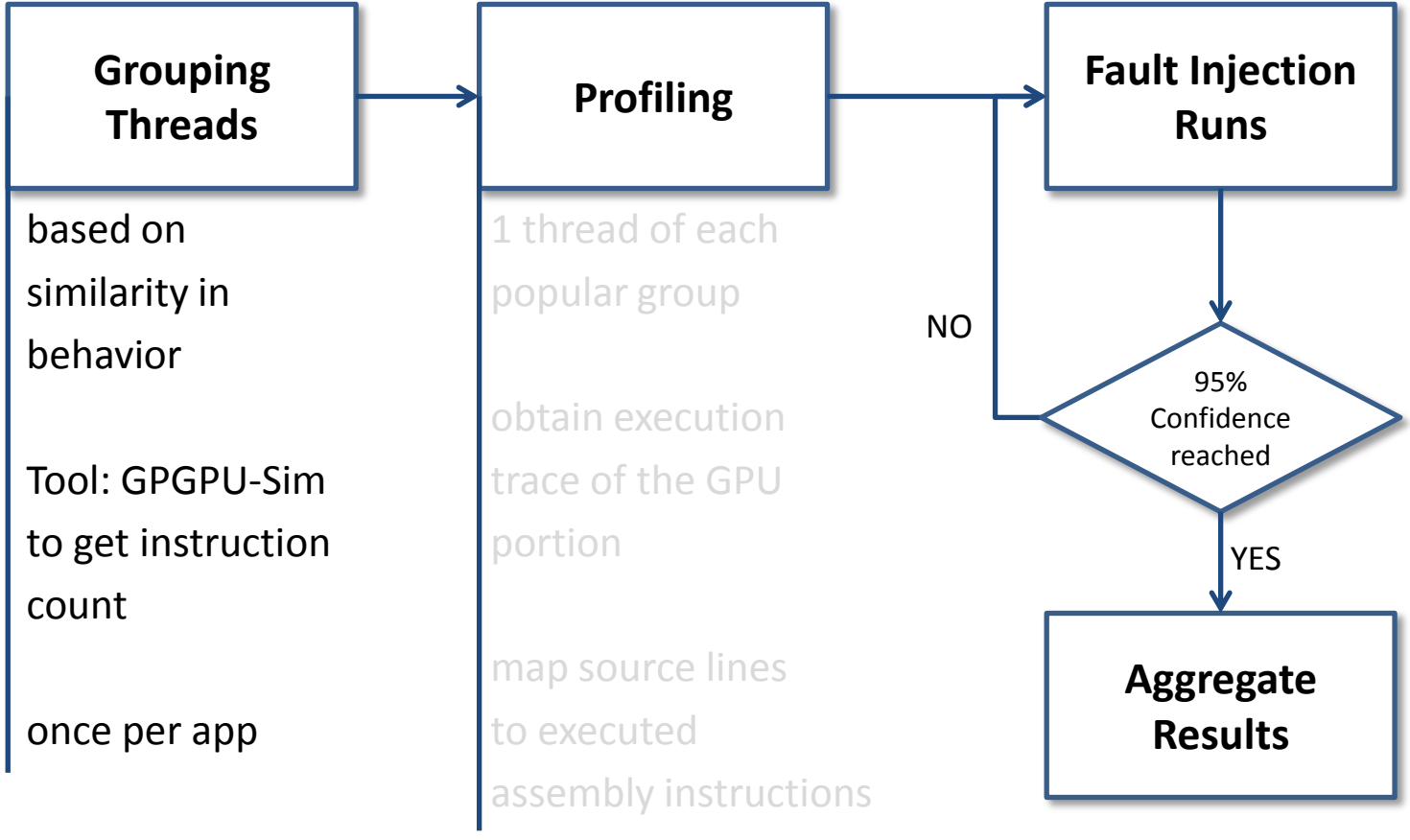
GPU-Qin: Methodology

Requirements (3 of 3)

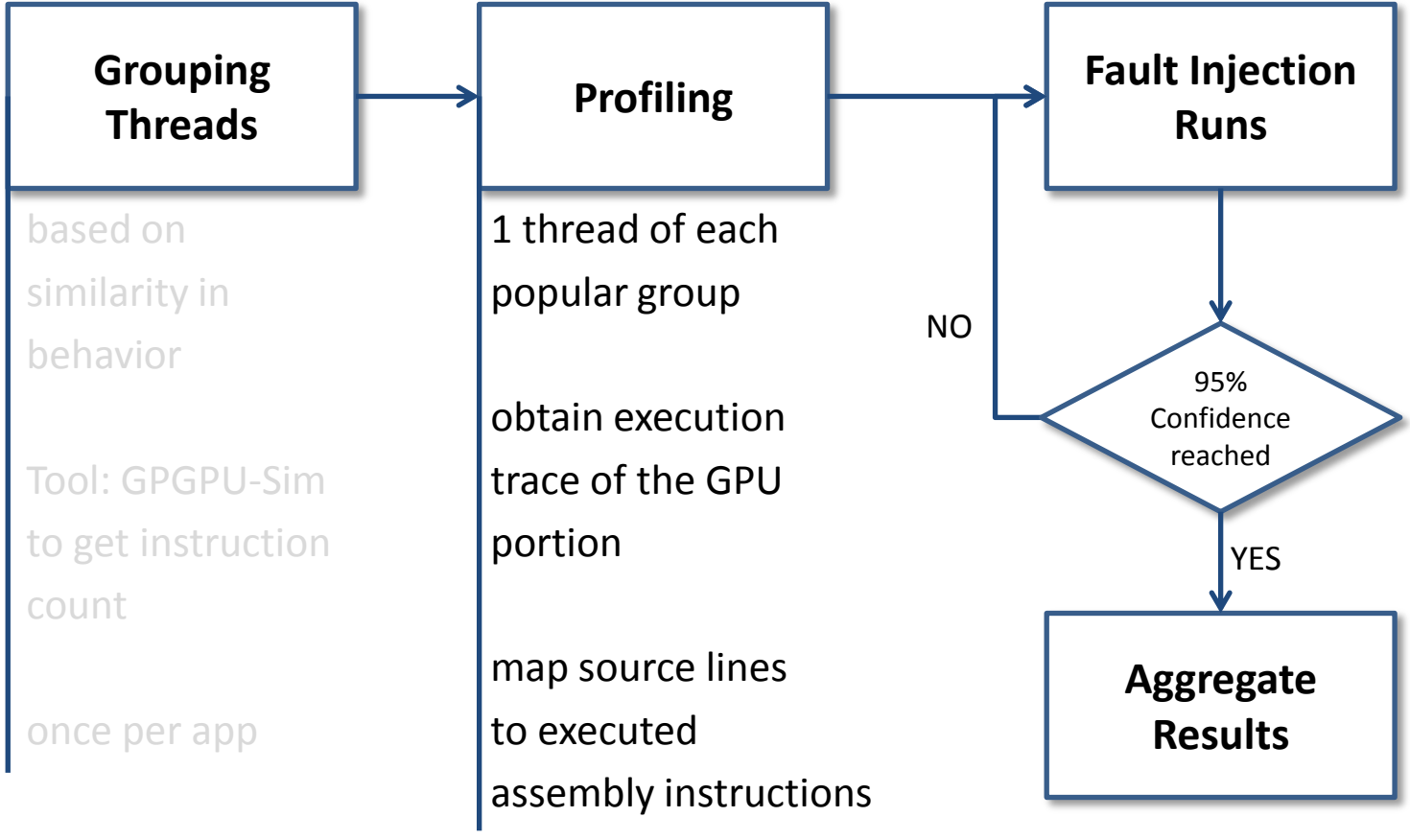
Minimum Interference:

- fault-injection experiment should *interfere minimally* with the original application
- minimal *modification of resilience characteristics* by the experiment
- fault injector *should not change code nor data*, other than for the ocjective of injecting the faults themselves

GPU-Qin: Phase I



GPU-Qin: Phase II



GPU-Qin: Phase III

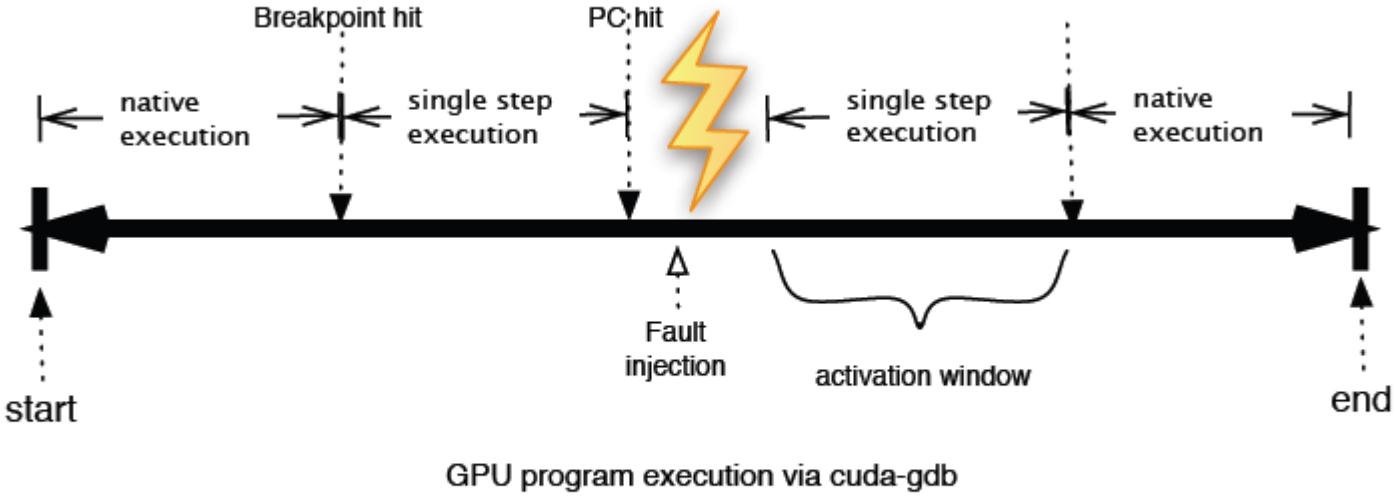


Fig. 4: Phase III - The fault-injection process

GPU-Qin: Phase III

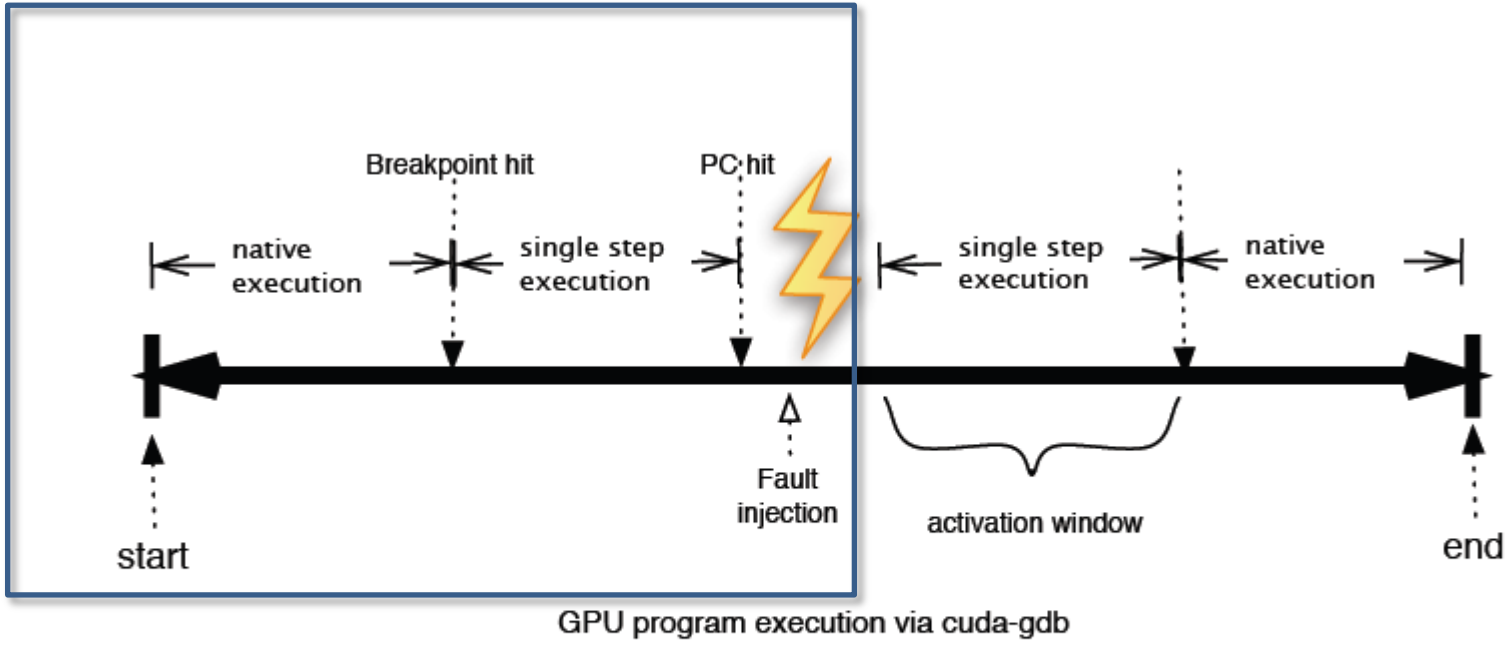


Fig. 4: Phase III - The fault-injection process

Instruction Types and their Injection

fault: injected by flipping a randomly chosen single bit

Instruction Type	Injection Location	What does that simulate?
Arithmetic	Destination register (vector with multiple destination register: randomly choose 1)	Error in ALU and FL-unit
Memory	Destination register or address register in LD/ST instructions	Faults in LSU
Control-Flow	Cuda-gdb doesn‘t allow to modify the predicate registers → inject in source operands of the instruction	Generally, „wrong decision“

GPU-Qin: Phase III

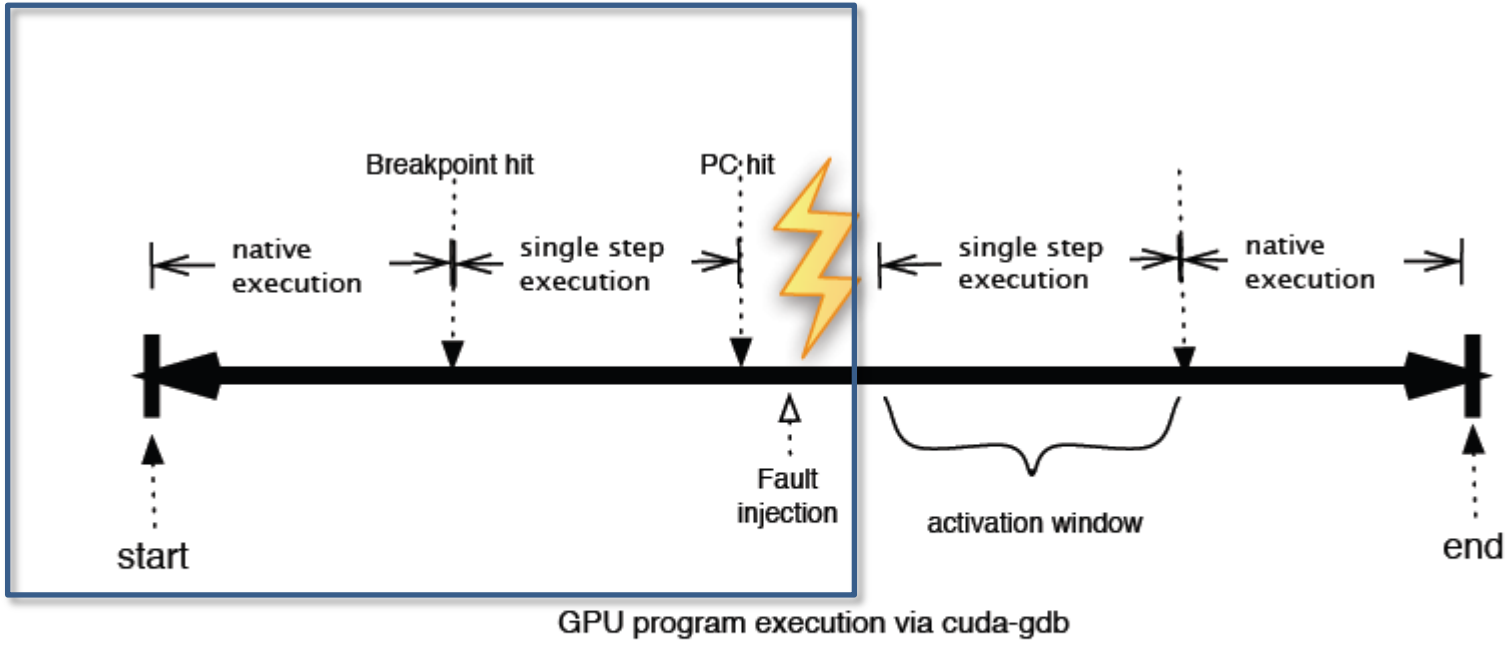


Fig. 4: Phase III - The fault-injection process

GPU-Qin: Phase III

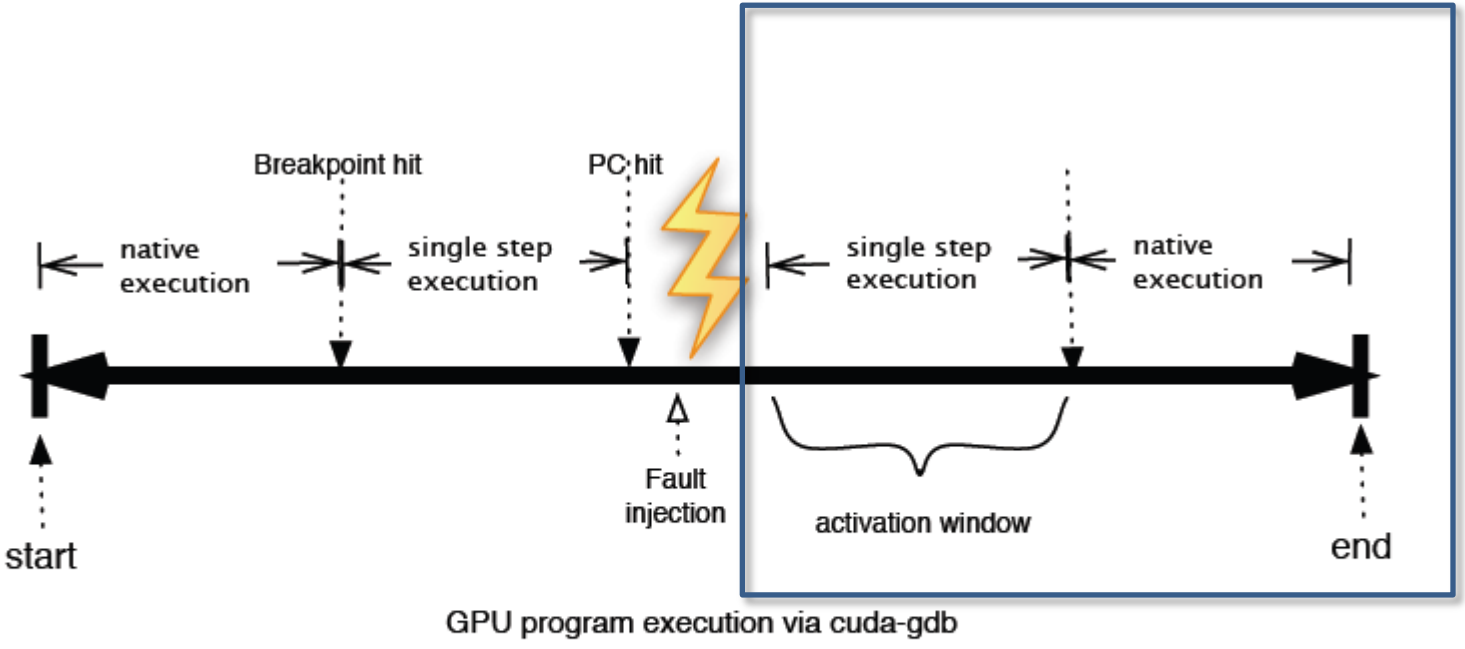
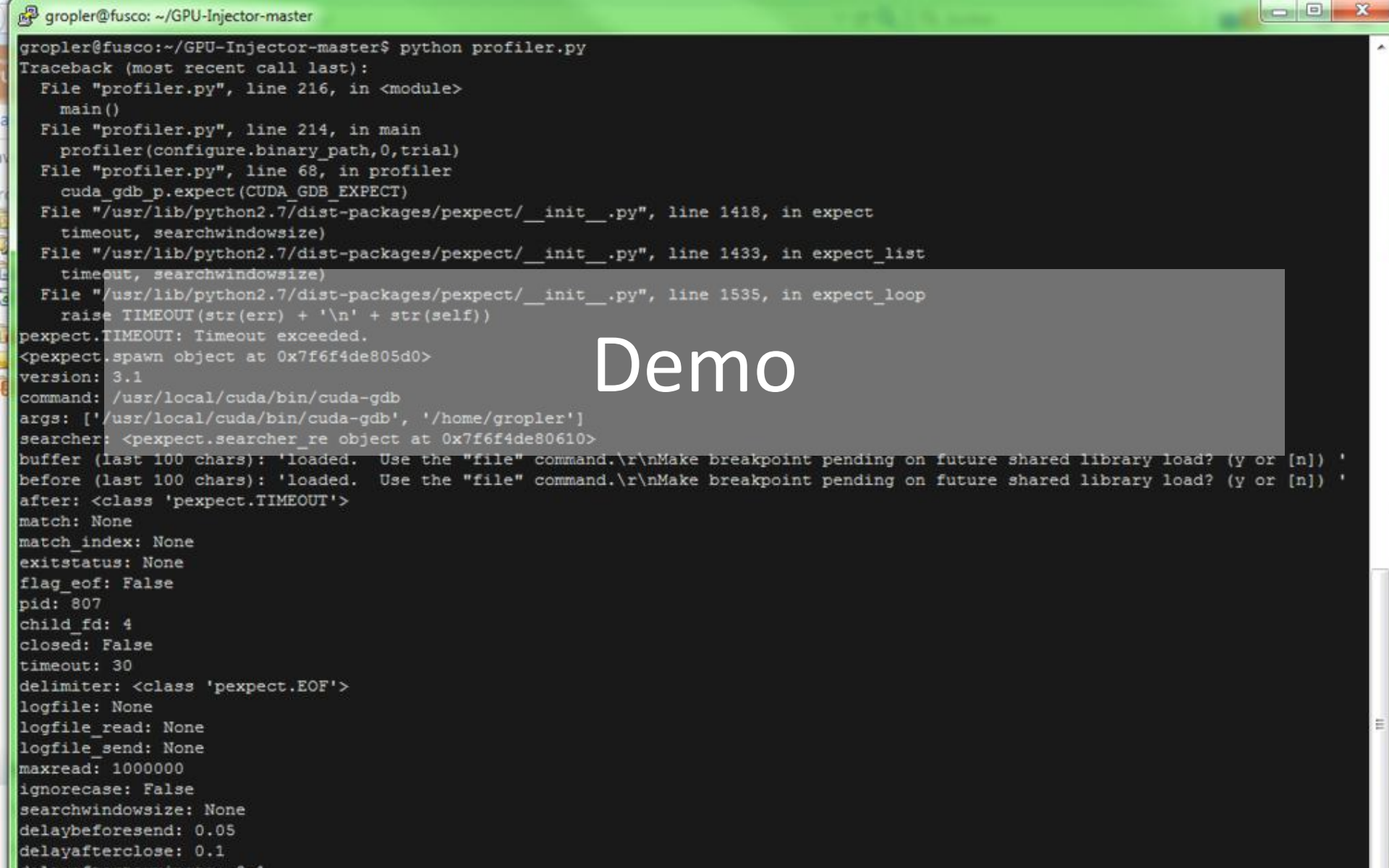


Fig. 4: Phase III - The fault-injection process


gropler@fusco: ~/GPU-Injector-master
gropler@fusco:~/GPU-Injector-master$ python profiler.py
Traceback (most recent call last):
File "profiler.py", line 216, in <module>
main()
File "profiler.py", line 214, in main
profiler(configure.binary_path,0,trial)
File "profiler.py", line 68, in profiler
cuda_gdb_p.expect(CUDA_GDB_EXPECT)
File "/usr/lib/python2.7/dist-packages/pexpect/__init__.py", line 1418, in expect
timeout, searchwindowsize)
File "/usr/lib/python2.7/dist-packages/pexpect/__init__.py", line 1433, in expect_list
timeout, searchwindowsize)
File "/usr/lib/python2.7/dist-packages/pexpect/__init__.py", line 1535, in expect_loop
raise TIMEOUT(str(err) + '\n' + str(self))
pexpect.TIMEOUT: Timeout exceeded.
<pexpect.spawn object at 0x7f6f4de805d0>
version: 3.1
command: /usr/local/cuda/bin/cuda-gdb
args: ['/usr/local/cuda/bin/cuda-gdb', '/home/gropler']
searcher: <pexpect.searcher_re object at 0x7f6f4de80610>
buffer (last 100 chars): 'loaded. Use the "file" command.\r\nMake breakpoint pending on future shared library load? (y or [n]) '
before (last 100 chars): 'loaded. Use the "file" command.\r\nMake breakpoint pending on future shared library load? (y or [n]) '
after: <class 'pexpect.TIMEOUT'>
match: None
match_index: None
exitstatus: None
flag_eof: False
pid: 807
child_fd: 4
closed: False
timeout: 30
delimiter: <class 'pexpect.EOF'>
logfile: None
logfile_read: None
logfile_send: None
maxread: 1000000
ignorecase: False
searchwindowsize: None
delaybeforesend: 0.05
delayafterclose: 0.1
Demo

gropler@fusco: ~/GPU-Injector-master

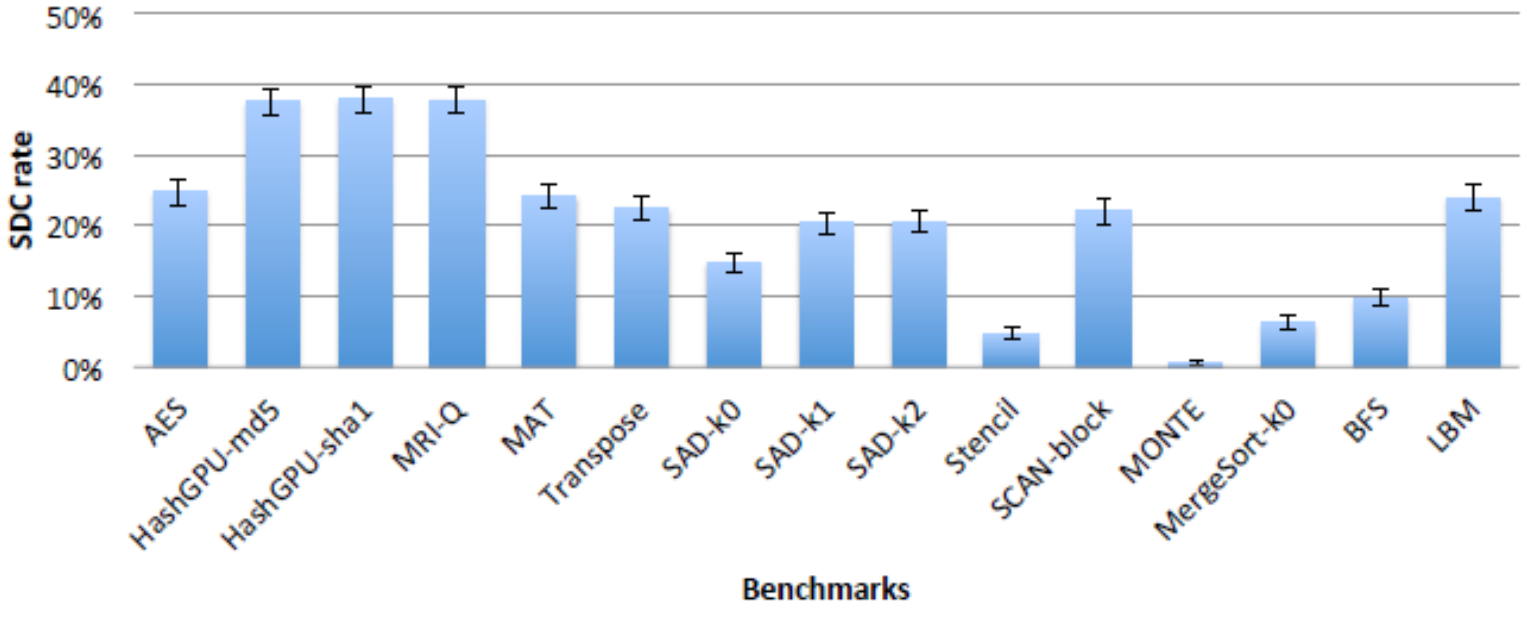
```
gropler@fusco:~/GPU-Injector-master$ python profiler.py
Traceback (most recent call last):
  File "profiler.py", line 216, in <module>
    main()
  File "profiler.py", line 214, in main
    profiler(configure.binary_path,0,trial)
  File "profiler.py", line 68, in profiler
    cuda_gdb_p.expect(CUDA_GDB_EXPECT)
  File "/usr/lib/python2.7/dist-packages/pexpect/__init__.py", line 1418, in expect
    timeout, searchwindowsize)
  File "/usr/lib/python2.7/dist-packages/pexpect/__init__.py", line 1433, in expect_list
    timeout, searchwindowsize)
  File "/usr/lib/python2.7/dist-packages/pexpect/__init__.py", line 1535, in expect_loop
    raise TIMEOUT(str(err) + '\n' + str(self))
pexpect.TIMEOUT: Timeout exceeded.
<pexpect.spawn object at 0x7f6f4de805d0>
version: 3.1
command: /usr/local/cuda/bin/cuda-gdb
args: ['/usr/local/cuda/bin/cuda-gdb', '/home/gropler']
searcher: <pexpect.searcher_re object at 0x7f6f4de80610>
buffer (last 100 chars): 'loaded.  Use the "file" command.\r\nMake breakpoint pending on future shared                    or [n]) '
before (last 100 chars): 'loaded.  Use the "file" command.\r\nMake breakpoint pending on future shared                    or [n]) '
after: <class 'pexpect.TIMEOUT'>
match: None
match_index: None
exitstatus: None
flag_eof: False
pid: 807
child_fd: 4
closed: False
timeout: 30
delimiter: <class 'pexpect.EOF'>
logfile: None
logfile_read: None
logfile_send: None
maxread: 1000000
ignorecase: False
searchwindowsize: None
delaybeforesend: 0.05
delayafterclose: 0.1
```

Demo

Sorry, I really tried my best.

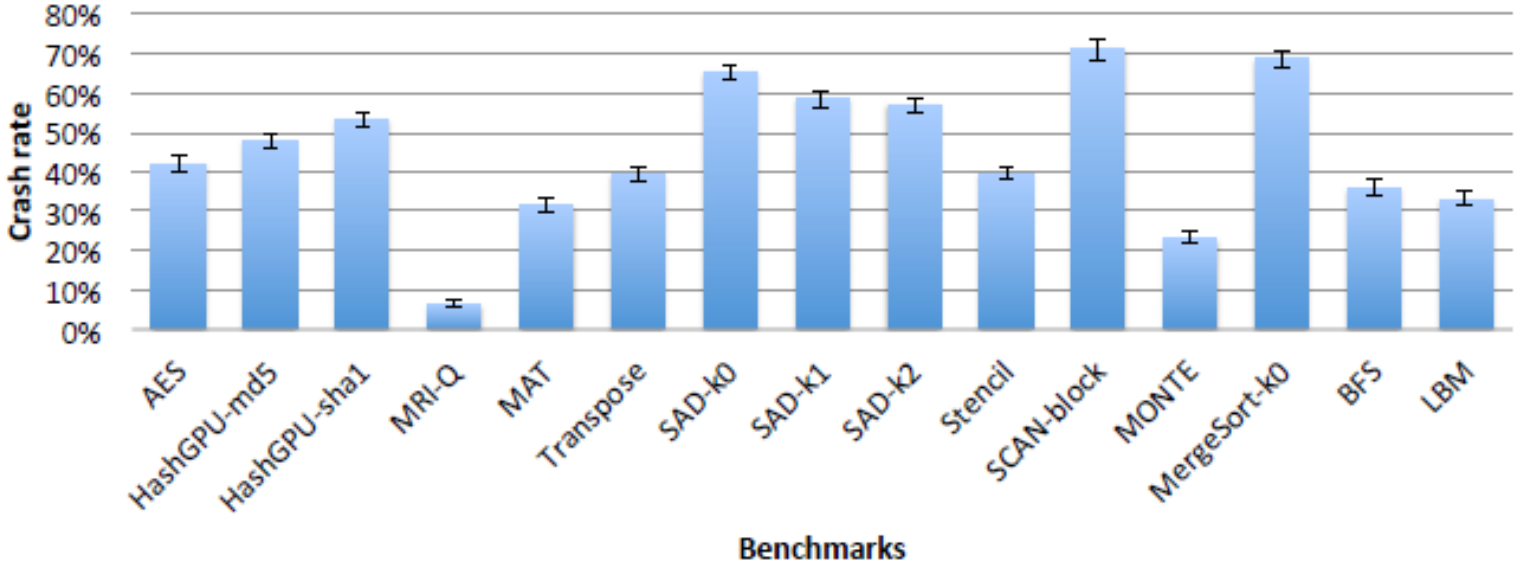
Results: SDCs

SDC rate varies across different benchmarks.



Results: Crashes

Crashes are a form of error detection performed by the GPU.



Results: Hangs

Hang: Timeouts, infinite loops

Uniformly lower than 1%.

Discussion: Heuristics

Thread partitioning into groups,

then profiling and fault injection based on most popular groups.

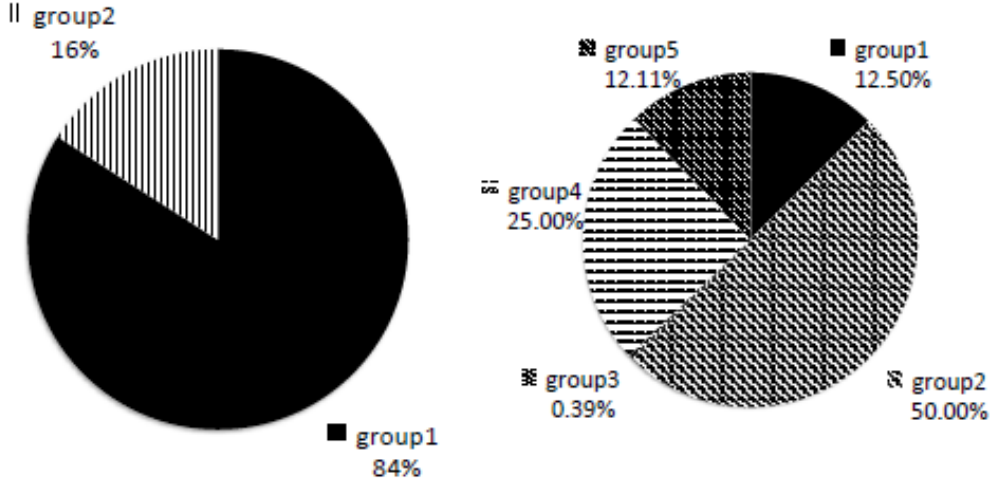


Fig. 2: Percentage of number of threads in each group to the total number of thread. *Left*: LBM *Right*: Monte Carlo

Discussion: Heuristics

Thread partitioning into groups,

then profiling and fault injection based on most popular groups.

TABLE I: The group identification process leads to classifying the benchmarks in three categories.

Category	Benchmarks	Groups	Groups to profile	% threads in picked groups
Category I	AES, MRI-Q, MAT, MergeSort-k0, Transpose	1	1	100%
Category II	SCAN, Stencil, Monte Carlo, SAD, LBM, HashGPU	2 - 10	1 - 4	95% - 100%
Categroy III	BFS	79	2	>60%

Discussion: Heuristics

Limit number of loop iterations to 64.

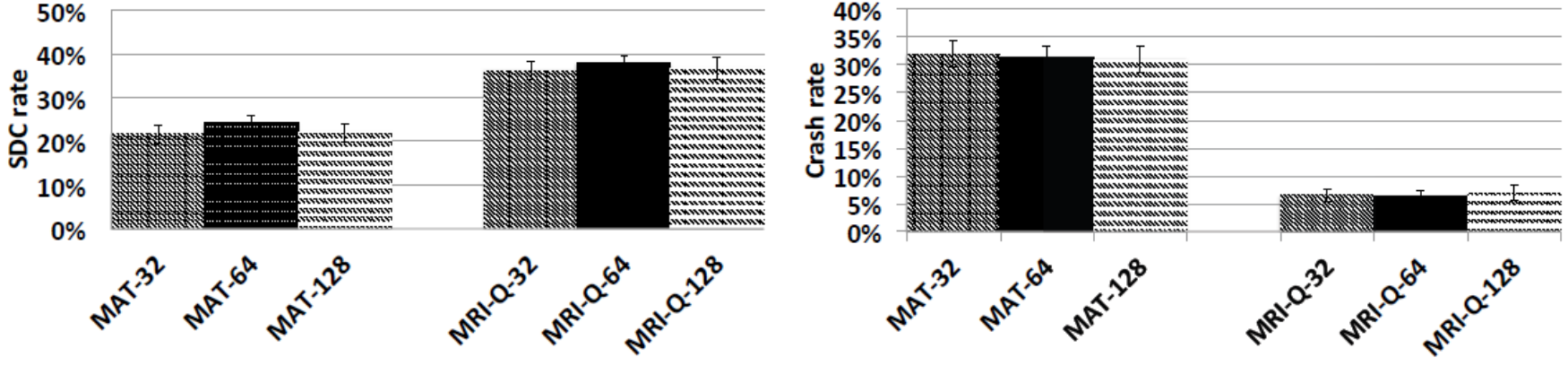


Fig. 6: Comparison of SDC and crash rate for different iteration threshold. *Left:* SDC. *Right:* Crash

Discussion: Heuristics

Fault is considered unactivated, if not seen activated within an activation window of 1600 dynamic instructions.

How often was this window exceeded?

- 36 cases in MAT
- 29 cases in MRI-Q
- (…)

… in thousands of runs!

Summary: GPU-Qin

Trigger mechanism:

- Execution-driven
- Location-based

Injection time:

- During runtime

Injection level:

- Intermediate code representation
- Instruction level (assembly-language level using GPU-based debugger)

Summary: GPU-Qin

Intended use cases:

- Transient hardware faults
- Single bit flips

Fault Coverage:

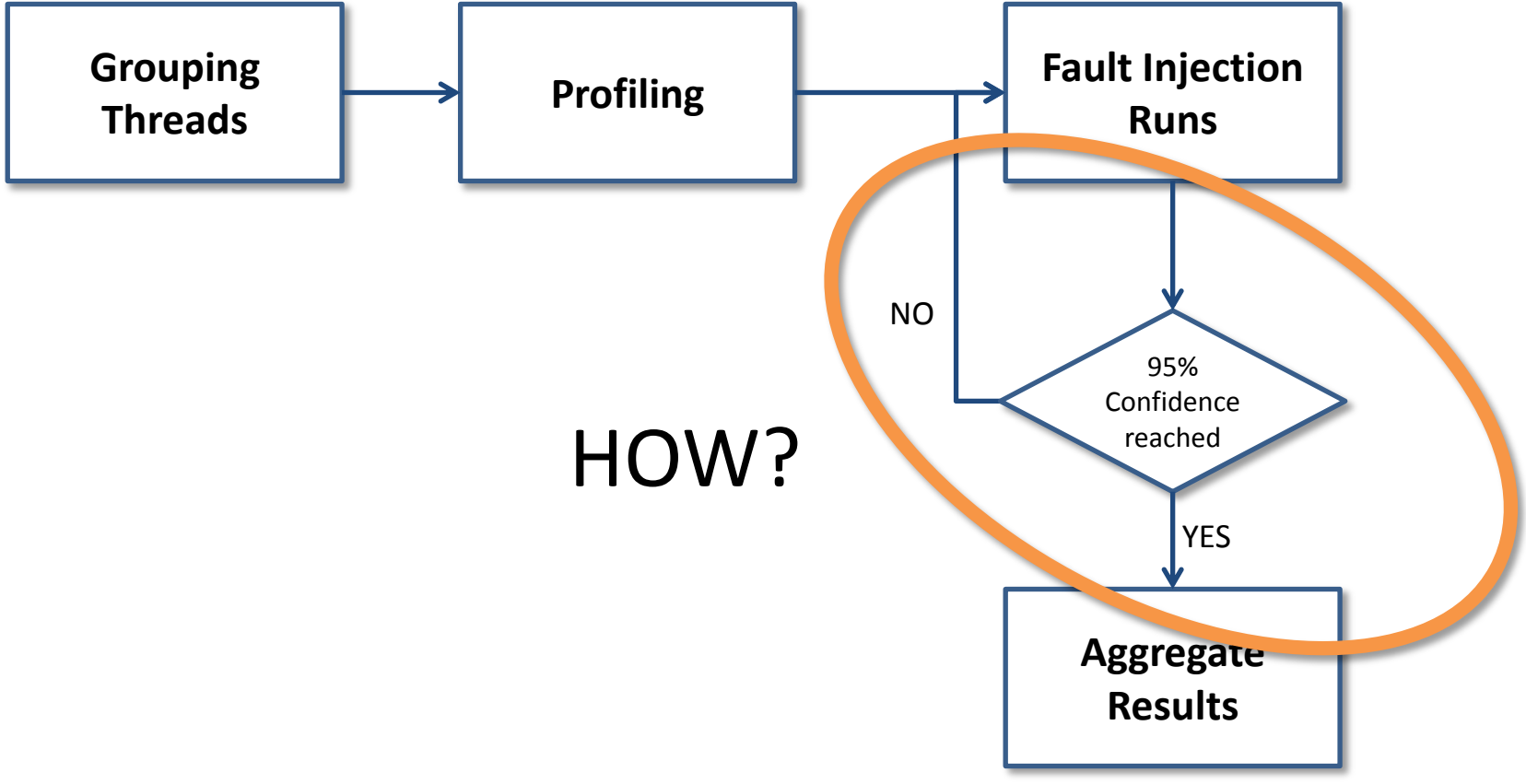
- Multi bit flips at locations “protected” by ECC , considered “No cost to extend to multiple-bit flip”, but is not evaluated yet.

Other use cases:

- Permanent hardware faults
- Over-heated components

→ GPU Stress test

Discussion: Open Questions?



HOW?

Appendix

Sources and Further Research

GPU-Qin project homepage and related ressources:

http://netsyslab.ece.ubc.ca/wiki/index.php/FTGPU

GPGPU-Sim tool used in first phase („Grouping“)

http://www.gpgpu-sim.org/

Understanding the parallelism of GPUs

http://renderingpipeline.com/2012/11/understanding-the-parallelism-of-gpus/

Sources and Further Research

Pay for extensive HDMI-cables to have less pixel errors?

http://www.expertreviews.co.uk/tvs-entertainment/7976/expensive-hdmi-cables-make-no-difference-the-absolute-proof/page/0/1

Found a single bit flip!

https://blogs.oracle.com/ksplice/entry/attack_of_the_cosmic_rays1

GPU Stress Test

http://www.geeks3d.com/gputest/

Elektrotechnische und physikalische Ursachen für transiente Hardwarefehler

http://ess.cs.tu-dortmund.de/Teaching/WS2012/SFt/Downloads/ausarbeitungen/Marc_Spohr.pdf

Sources: Images

http://cdn2.expertreviews.co.uk/sites/expertreviews/files/styles/insert_main_image/public/images/dir_335/er_photo_167680.png?itok=JANLYyfv

http://memeguy.com/photos/images/yesterday-was-the-first-day-of-linear-algebra-this-was-how-the-class-ended-80154.jpg

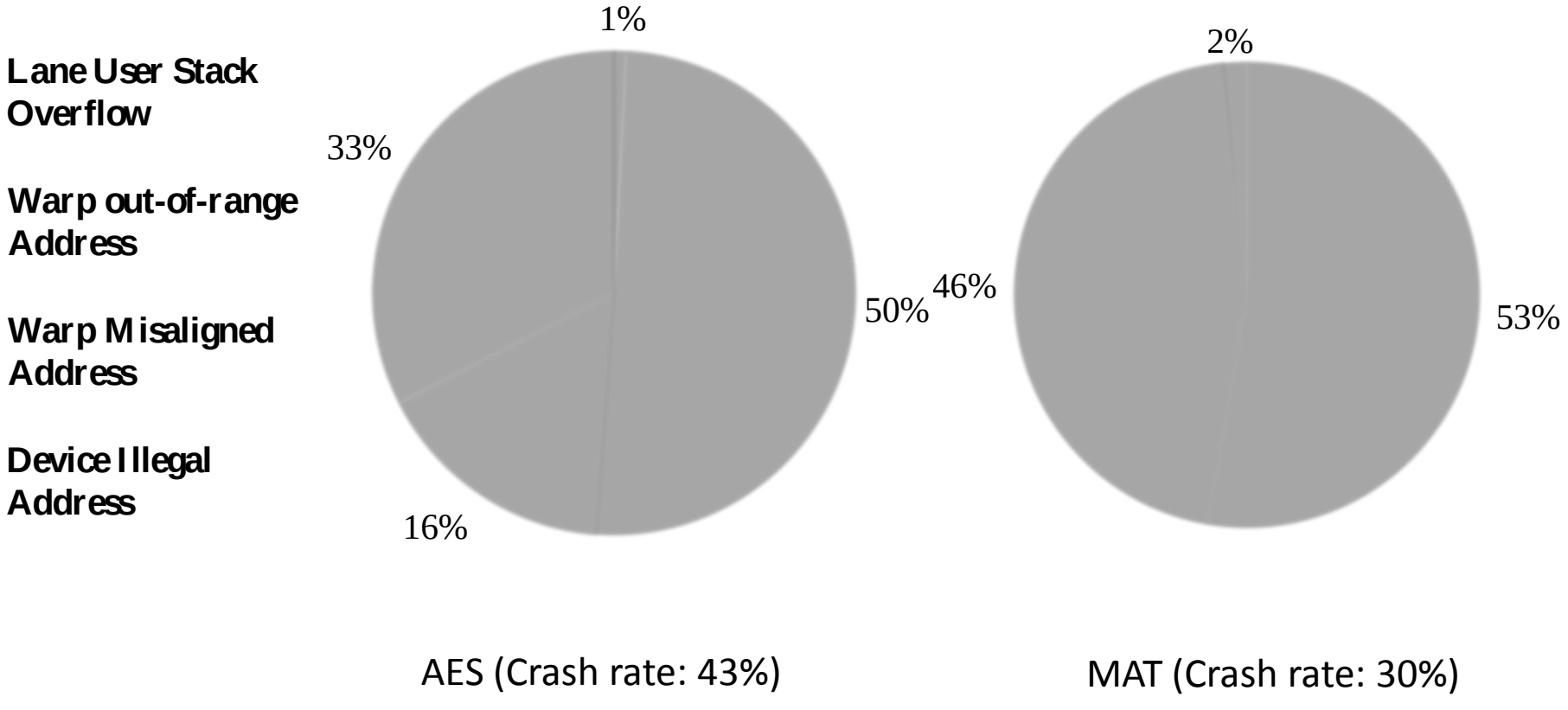
http://frontiersmag.wustl.edu/wp-content/uploads/2015/02/DNA_finger_large_CTAG1.jpg

http://www.chipsetc.com/uploads/1/2/4/4/1244189/1364845_orig.jpg?307

http://1.bp.blogspot.com/-tz38lMri9-Y/T74L9mjV9UI/AAAAAAAAARc/g3KIlSnw-3s/s1600/bitflip.jpg

http://i2.kym-cdn.com/entries/icons/original/000/002/862/Re2idh_c.jpg

Hardware Exceptions (Crashes)



AES (Crash rate: 43%)

MAT (Crash rate: 30%)

Results: Crashes

Crashes are a form of error detection performed by the GPU.

TABLE IV: Description of CUDA hardware exceptions

Exception type	Descrption
Lane user stack overflow	Occurs when a thread exceeds its stack memory limit
Warp out-of-range address	Occurs when a thread within a warp accesses an out-of-bounds local or shared memory address
Warp misaligned address	Occurs when a thread within a warp accesses an incorrectly aligned local or shared memory address
Device illegal address	Occurs when a thread accesses an out-of-bounds global memory address

Crash latency – measure the fault propagation