# Foundations of Empirical Memory Consistency Testing

Jake Kirkham, Tyler Sorenson, Esin Tureci, Margaret Martonosi

Presented by
Akshay Gopalakrishnan

McGill University

July 19, 2021

# Introduction

- Testing semantics of Shared Memory Consistency w.r.t. their parent Hardware.
- Testing done using various parameters forcing the hardware to proc Relaxed Behaviors.
- Testing done on the OpenCL memory consistency model running on various GPU/CPU bases.
- Bugs detected hint towards compiler bugs (in this paper that of Intel x86)/ hardware bugs hinting towards misrepresentation of the behaviors of Hardware..

# Memory Consistency Models

- Semantics defining the possible values a memory read can give in execution of concurrent programs.
- Required to write concurrent programs using shared memory concurrency.
- Our intuitive reasoning of concurrent programs rely on Sequential Consistency.

*The outcome of an execution of a program is equivalent to the same program running in a uniprocessor setting (interleaving semantics).*

# Why Empirical Testing as opposed to Formal Proofs?

- Formal proofs regarding memory consistency (compiler mappings and memory model implementation) are done manually.
- Compiler technology is ever evolving, and manual efforts needed to account for new parts of languages (which influence concurrent programs).
- Hardware proofs (checking implementation of memory model w.r.t. hardware behaviors) have been applied only to small systems or toy models.
- Existing proofs have sometimes also been shown to be incorrect (mainly through counter examples).
- Empirical testing has shown promise to identify bugs in compilers and microarchitectural features.
- Litmus tests run for a million iterations gives much confidence towards validation of the system w.r.t. semantics defined.

# Previous Testing Framework Deficiencies

- Reproducibility - Often difficult to reproduce the same set of bugs in the same setting (require more fine grained setting).
- False Negatives - Does not guarantee lack of bugs (its confidence can be enhanced using some measures of probability).
- Redundant testing time - Often over testing is done (this can be managed using probabilistic guarantees) .

# Proposed Testing framework with Parameters

- Stressing and Testing.
- Barrier and Occupancy.
- Id Shuffling and Oversubscription.

# Memory Parameters: Stressing and Testing Parameter

- Memory is divided into two parts : Stressing and Testing.
- Testing memory - Memory using which our target program operates/ executes.
- Stressing memory - Memory not touched by the target program.

It is called stressing memory because for testing, we would employ programs to heavily use the stressing memory section. This would influence the role of caches in the execution. Also, it would give us insight on how other physical parameters (temperature, power consumption, etc.) would affect.

# Concurrency Parameters: Barrier and Occupancy

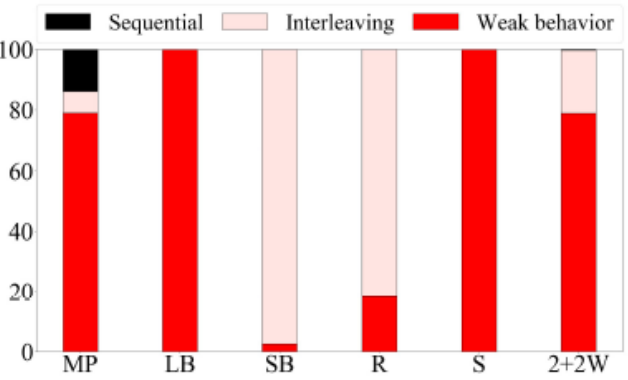
- In any multi-core experiment, it could be that each thread is spawned for execution or assigned to cores in an uneven manner.
- It could even result in the whole execution being sequential in time.
- This would prevent any relaxed behaviors from being observed.
- Hence, a barrier is used to ensure uniform spawning of threads for them to possibly exhibit relaxed behaviors.

# Affinity Parameters: Id Shuffling and Oversubscription

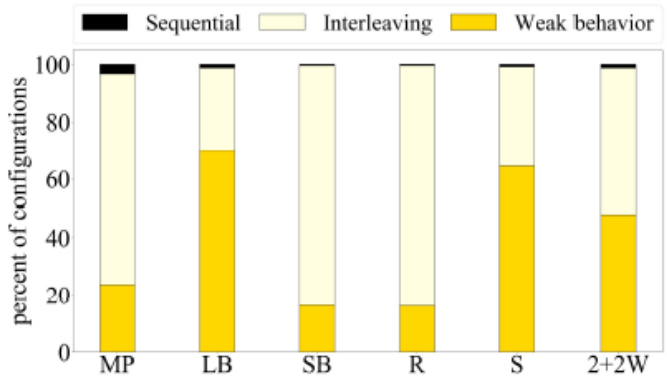
- We have many software threads but limited hardware threads.
- Manually assigning these threads to that of GPU will prevent us from seeing relaxed behaviors due to other hardware components (eg: Shared Cache).
- Randomized assignment works better for the purpose of experiment.

# Sequential, Interleaving and Relaxed Observations

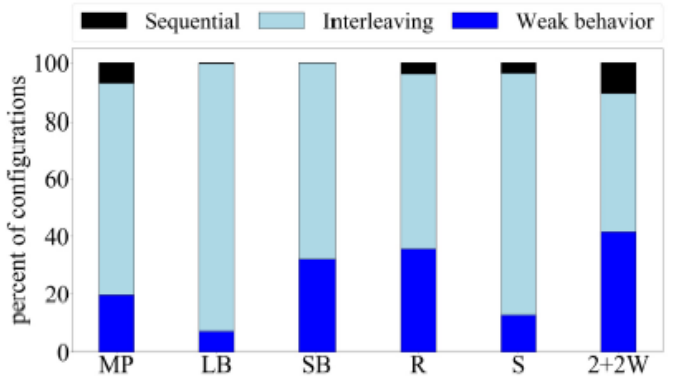
| Execution | SB | LB | MP |
|---|---|---|---|
| 1 2 3 4 | (L1==0 && L2==1) | (L1==0 && L2==1) | (L1==1 && L2==1) |
| 3 4 1 2 | (L1==1 && L2==0) | (L1==1 && L2==0) | (L1==0 && L2==0) |
| 1 3 2 4 | (L1==1 && L2==1) | (L1==0 && L2==0) | (L1==0 && L2==1) |
| 1 3 4 2 | (L1==1 && L2==1) | (L1==0 && L2==0) | (L1==0 && L2==1) |
| 3 1 4 2 | (L1==1 && L2==1) | (L1==0 && L2==0) | (L1==0 && L2==1) |
| 3 1 2 4 | (L1==1 && L2==1) | (L1==0 && L2==0) | (L1==0 && L2==1) |
| *Relaxed Behavior* | (L1==0 && L2==0) | (L1==1 && L2==1) | (L1==1 && L2==0) |




(a) VEGA

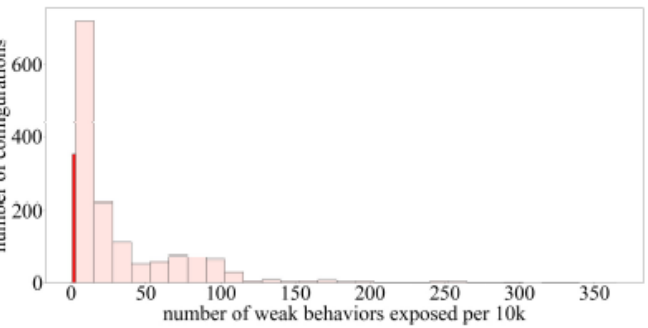



(b) QUADRO

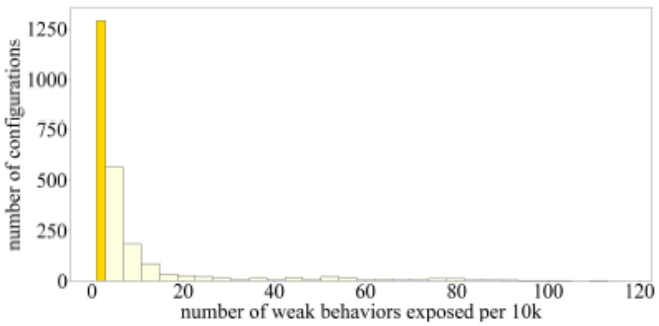



(c) IRIS

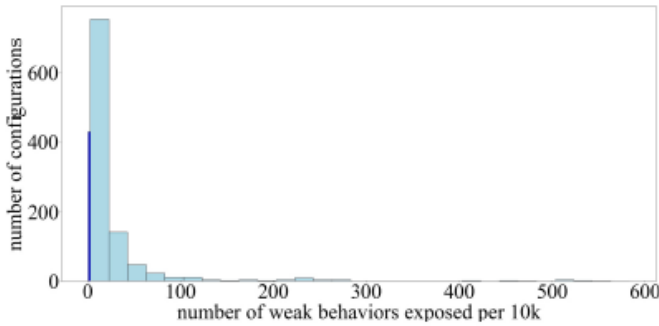
# Rate of relaxed Behaviors observed w.r.t. Parameters




(a) VEGA




(b) QUADRO

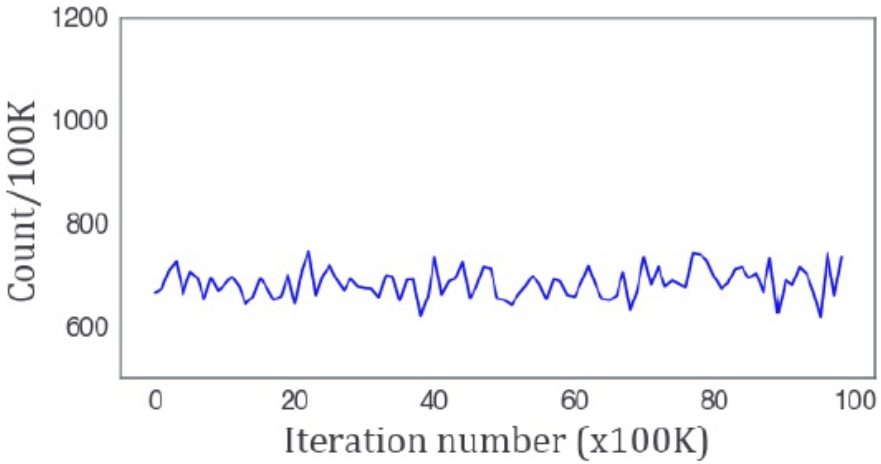



(c) IRIS

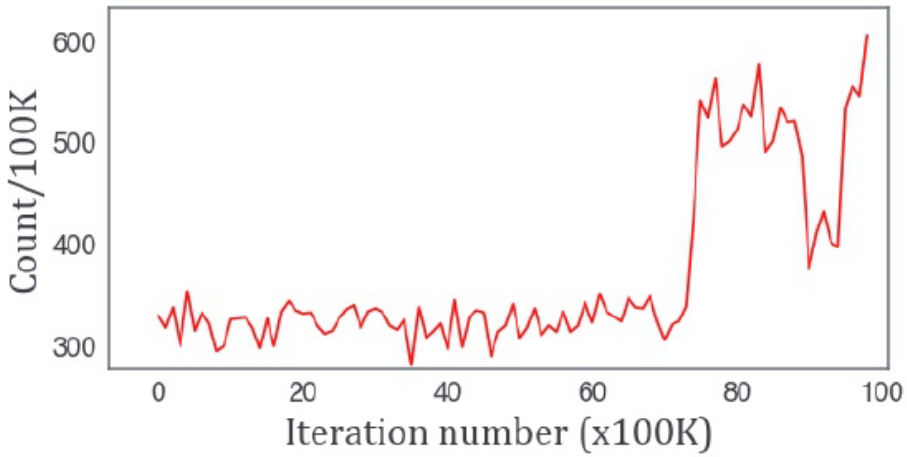
- Note that the rate of decrease is exponential, indicating there are very few configurations showing more relaxed behaviors.
- This greatly influences the factor of reproducibility.

# Stationary Relaxed Behaviors

- Often relaxed behaviors will vary as the time passes by testing the same litmus test.
- This is due to external factors (heat/power consumption) that may influence the GPU/CPU to resort to other measures.
- To check stationary factor of relaxed behaviors, choose a parameter and just keep running litmus tests based on it.
- See whether the same set of relaxed behaviors persist over time, with negligible variance.




(a) Iris




(b) Vega

# Litmus Test Tuning (LTT)

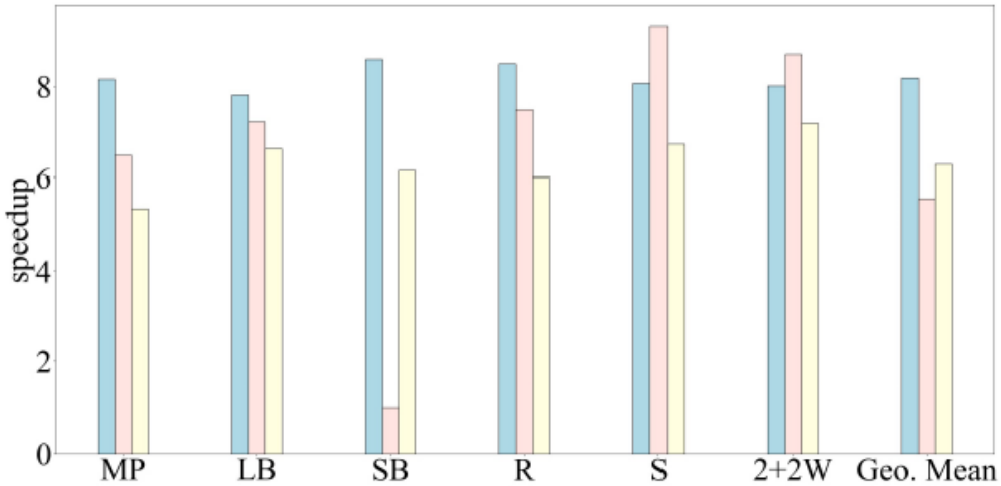
- Best litmus tests ?
- How many relaxed behaviors a single litmus test proc?
- Can a litmus test proc the same relaxed behaviors on different hardwares (if they allow it)?
- For this, litmus test tuning is required.

Broadly three types of tuning

- Data peeking.
- Portable parameter testing.
- Cross correlations among litmus tests.

# LTT 1: Data Peeking

- If in two tests, one clearly shows more promise of relaxed behaviors.
- The lesser promising can be discarded.
- This can save power and testing time.
- Misc: Heat dissipation reduced.



# LTT 2: Portability based parameters

There are 3 ways we can divide our approach, assume we have N GPUs and M tests:

- Global - Across all GPUs, a single parameter which give most relaxed behaviors in all of them.
- Per-GPU - Each GPU will spur our representative parameter sets (N).
- Per-test - Each parameter per litmus test that works well for all GPUs (M).
- Per-GPU/test - For each GPU, each test, with best parameters (N * M).

# LTT 3: Cross Test Correlations

- Two tests trigger the same behavior in question.
- Eg: *SB* and *R* will have read-after-write sequence, both of whose relaxed behavior will involve violating this order.
- More beneficial to discard one case.
- Metric for this can be correlations matrix.

(a) Vega

| $R^2$ | MP | LB | SB | R | S | TPTW |
|---|---|---|---|---|---|---|
| MP | | 0.05 | 0.09 | 0 | 0.01 | 0.01 |
| LB | | | 0.01 | 0.19 | 0.61 | 0.13 |
| SB | | | | 0.03 | 0.01 | 0 |
| R | | | | | 0.02 | 0.12 |
| S | | | | | | 0.04 |
| SUM | 0.16 | 0.99 | 0.13 | 0.36 | 0.69 | 0.3 |

(b) Quadro

| $R^2$ | MP | LB | SB | R | S | TPTW |
|---|---|---|---|---|---|---|
| MP | | 0.034 | 0.126 | 0.089 | 0.041 | 0.029 |
| LB | | | 0.029 | 0.033 | 0.787 | 0.618 |
| SB | | | | 0.149 | 0.035 | 0.021 |
| R | | | | | 0.037 | 0.027 |
| S | | | | | | 0.636 |
| SUM | 0.208 | 0.977 | 0.235 | 0.218 | 1 | 0.866 |

# Conformance and Mutation Testing

- Basically, we want to see what elements (semantics and hardware component wise) triggers the relaxed behaviors.
- Conformance testing using Minimal forbidden litmus tests (tests whose accesses, when relaxed even one of them will result in a relaxed behavior)
- The authors found a bug in Intel Iris doing this type of testing.
- Mutation testing is using an implementation of the semantics and then creating a set where each element represents on implementation minus one semantic requirement.
- The authors noted that the Nvdia GPU does not show many relaxed behaviors using mutation testing.

# Conclusion

A more careful empirical testing of memory models (H/W) reveals to us several aspects of the system that play a role in relaxed behaviors:

- Different Hardware components may affect the testing results.
- Reproducibility, False Negatives and Redundancy.
- Stress on components not involved in the memory.
- Setting barriers, affinities towards threads of litmus tests.
- Litmus test tuning is helps reduce power consumption, heat dissipation, time and processing power.
- Tuning can also result in redundancy removal of litmus tests.
- Conformance testing reveals possible bugs in hardware(eg: Intel Iris).
- Mutation testing helps observe which hardware is more affinity for weak behaviors.

# Thank you

Questions?