Evaluating the Efficacy of Parallel Fuzzing

Sungho Lee, Ethan Oh

Fuzzing

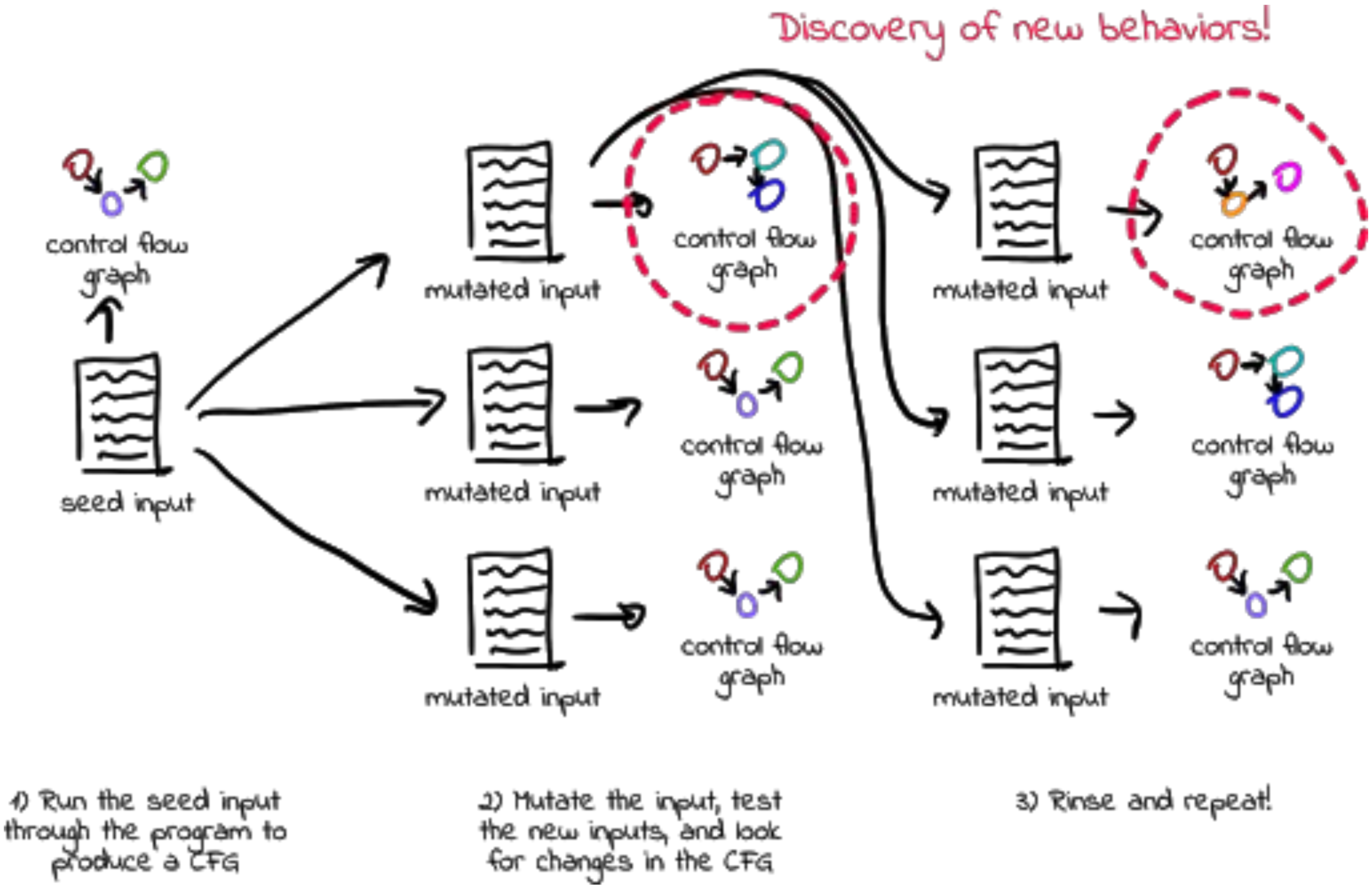
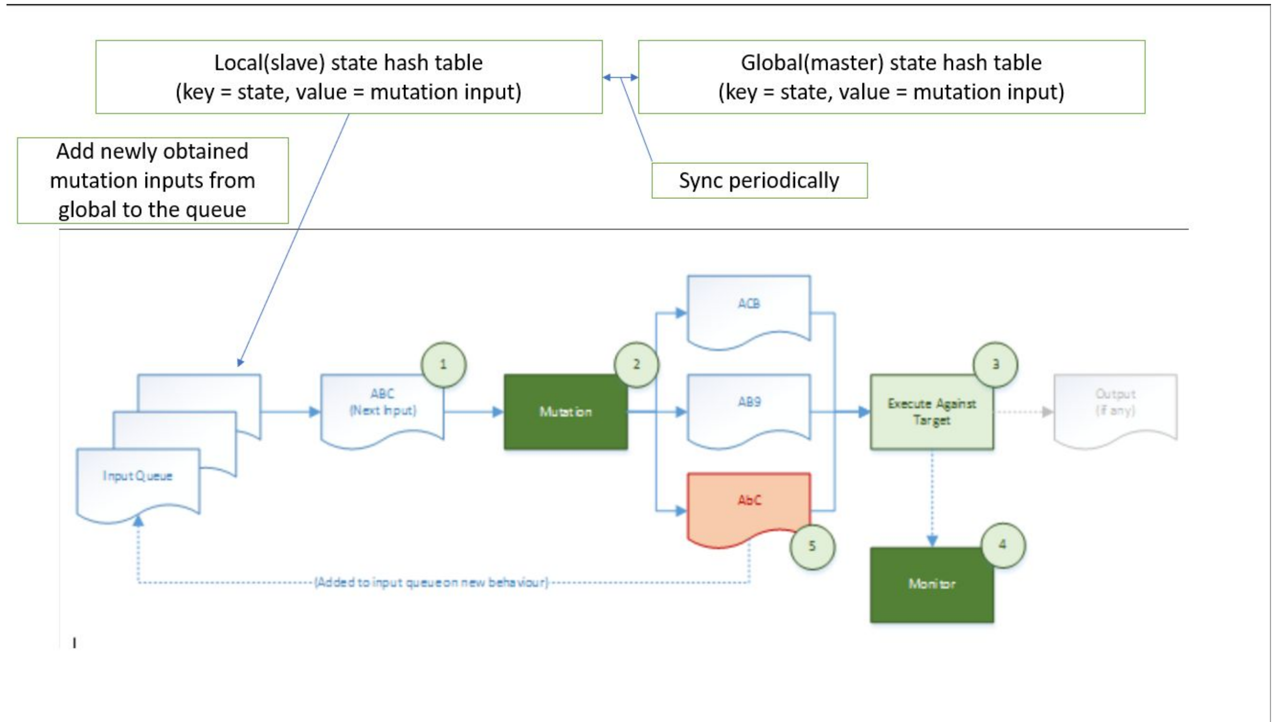


Image credit: https://blog.trailofbits.com/2020/10/22/lets-build-a-high-performance-fuzzer-with-gpus/

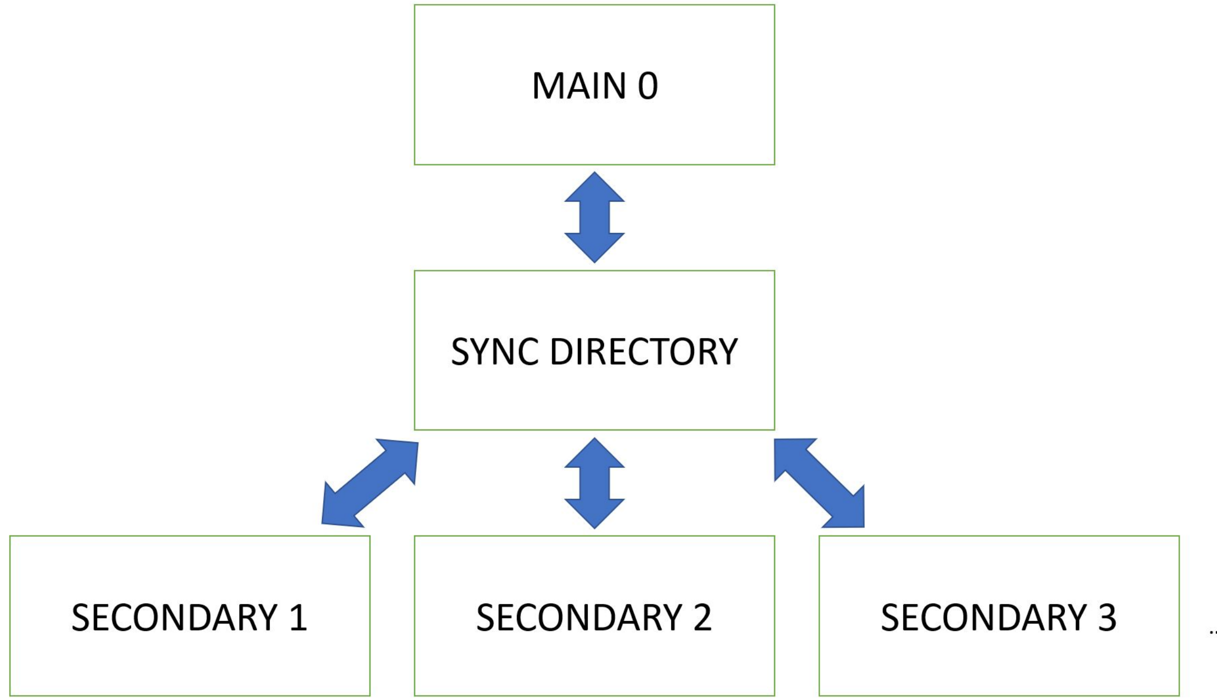
View of Single AFL++ Fuzzing Instance



AFL++ follows the scheme below:

1. Load an initial user-supplied input to the input queue
2. Take the next input out of the input queue
3. Attempt to trim the test case to maintain the measured behavior (1)
4. Mutate the input (1)
5. Run the input with the given program
6. Mutations that get a new state transition get added to input queue
7. **Only multi-thread : Periodically sync with main thread and add interesting inputs from main to the input queue** (1)
8. GOTO 2

Parallelism Explained



- Each Instance Classified as Main or Secondary Instance
- Each Instance Keeps Track of State Transitions Covered
- Synchronization
    - All synchronization done asynchronously
    - The Main Instance Scans all Secondary Fuzzers
    - Secondary Fuzzers scan only the Main Fuzzer
    - The topology above and the reduction of number of synchronizations help keep synchronization costs low

Testing

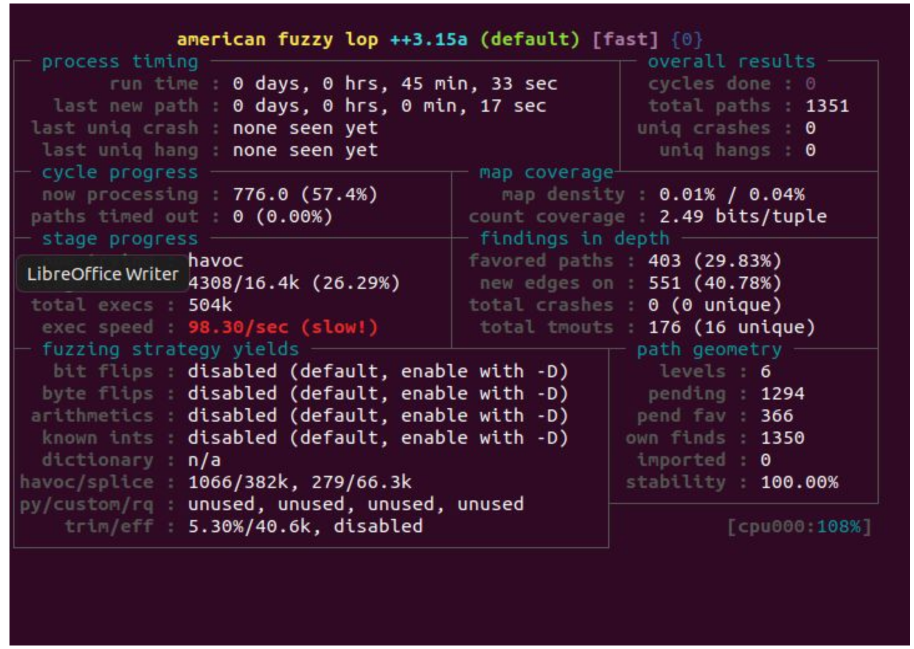
Specifications:

Ryzen 3600x CPU - 6 cores, 12 threads

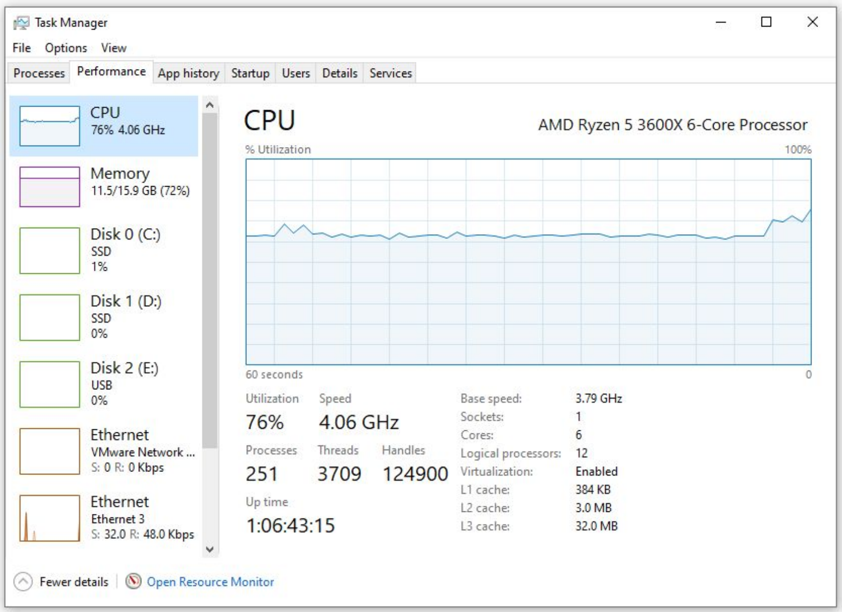
VMware running on Windows 10 settings:

- Ubuntu 20.04.03 LTS
- 4GB RAM
- CPU 6 cores, 2 threads each (gave all cores to the VM)

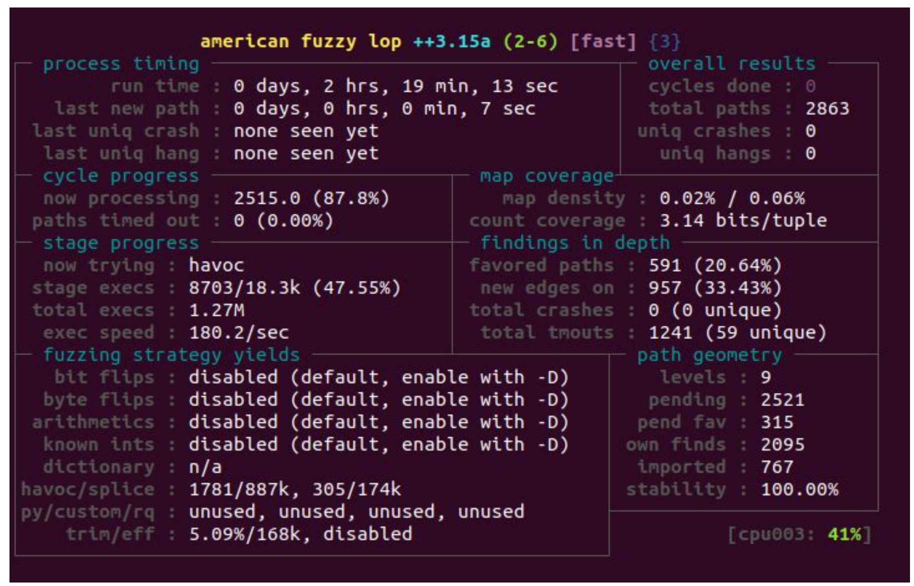
Program Under Test : Readelf
Initial Seed : /bin/ps



Single-thread execution

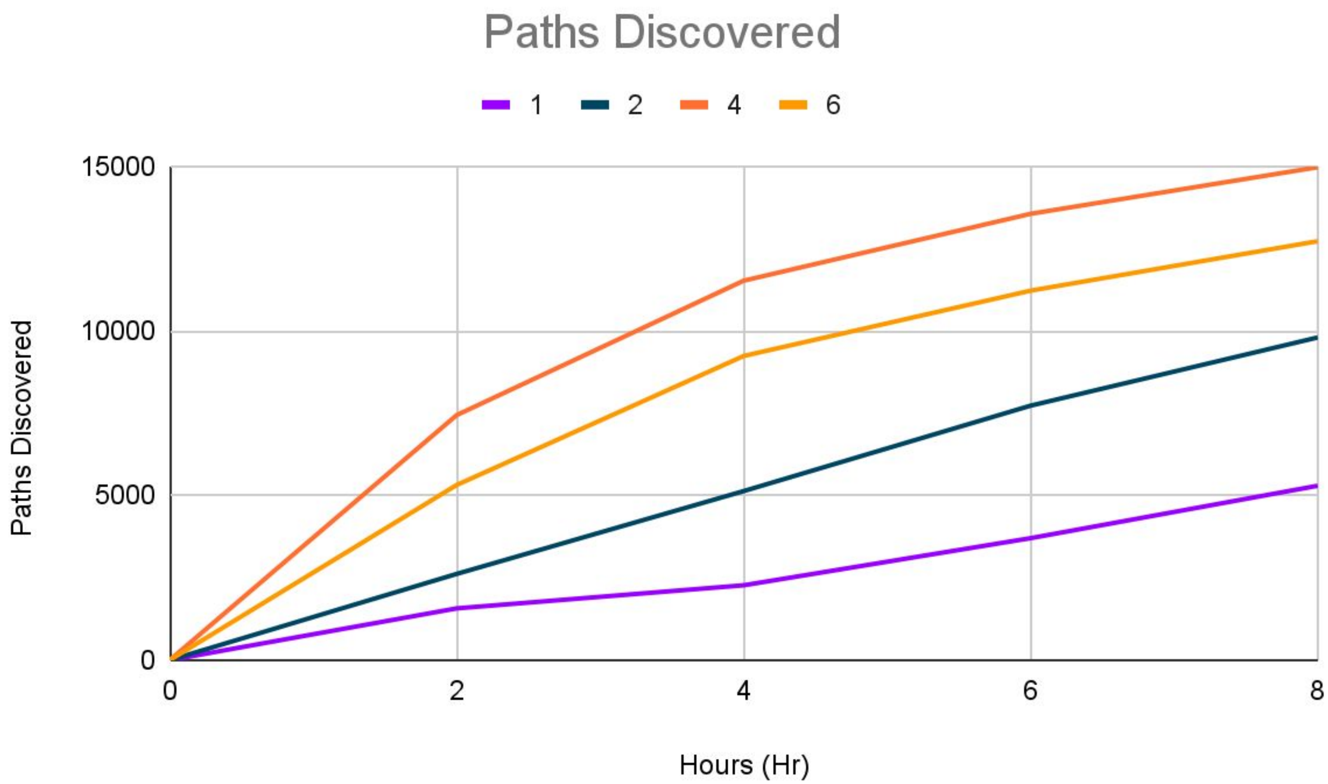


CPU Usage seen from Host Device



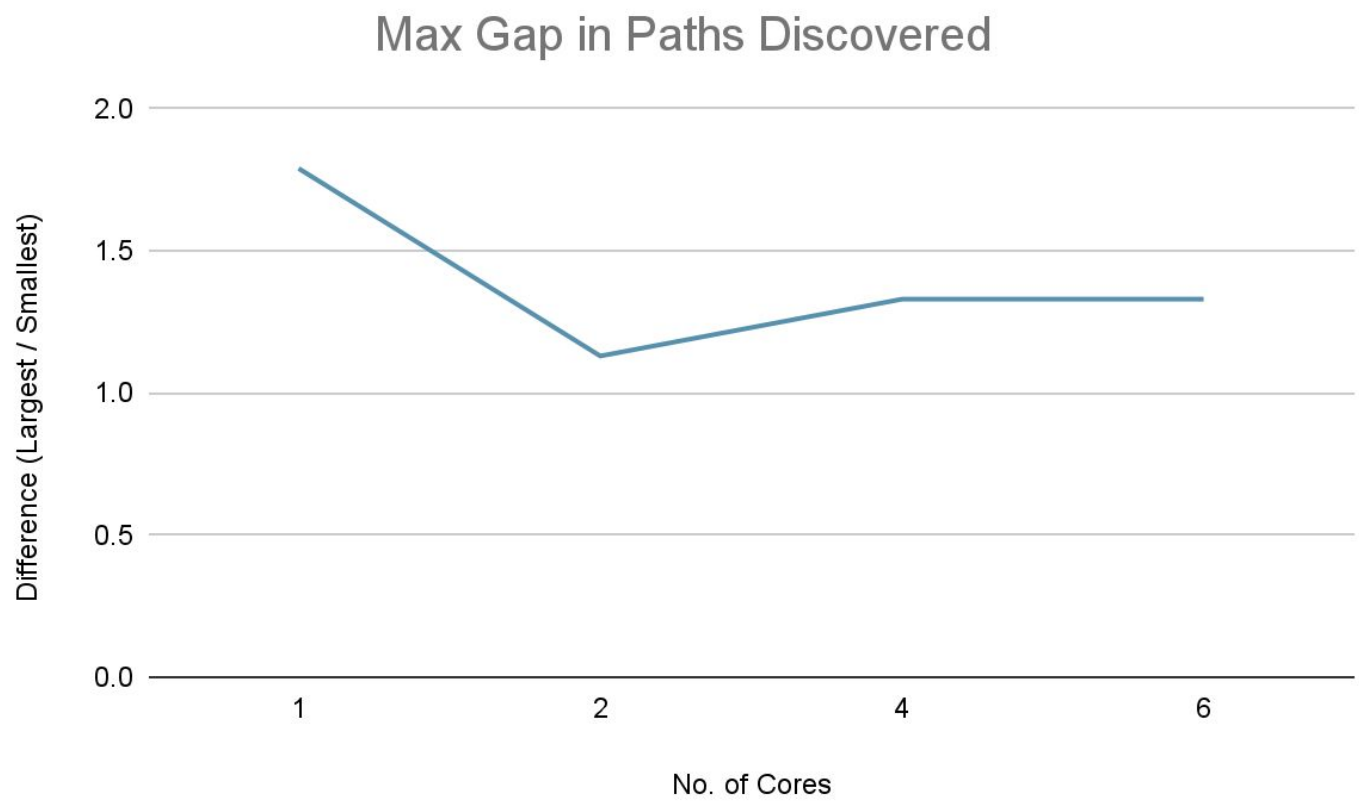
Multi-thread execution

Result 1 - Paths



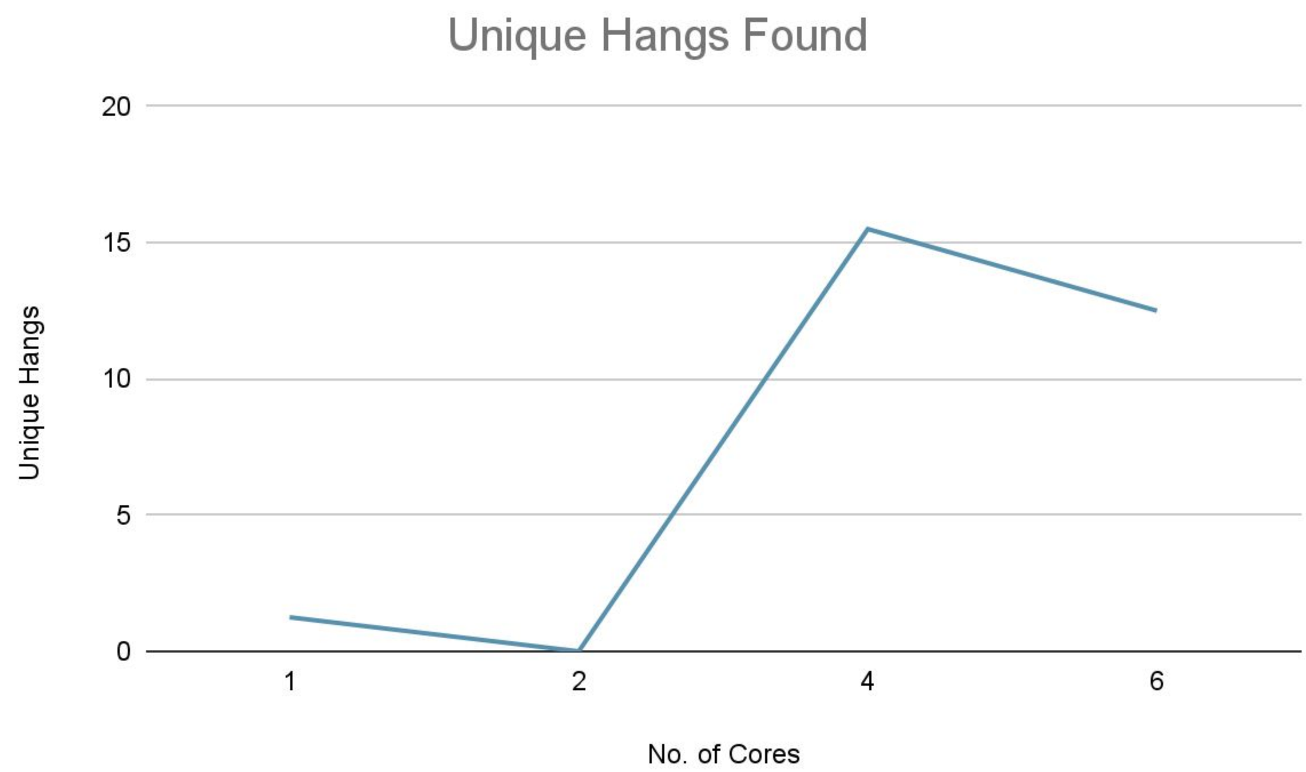
- Increase in performance from 1 core to 2, 4, 6 cores in all four time periods
- All multi-core fuzzing experienced the same trend of performance boost
    - (0 ~ 2) to (2 ~ 4) Hr
        - Successfully widened their gap with the single-core session, hitting the peak at around the 4 hour mark, hitting around linear speedup (2.25x / 5.07x / 4.06x).
    - (4 ~ 6) to (6 ~ 8)
        - Performed a bit worse, narrowing the gap between single core performance (1.85x / 2.82x / 2.40x)
- 4-core performed better than 6-core
    - Possibly resource allocation problems with the virtual machine for higher core counts.

Result 2 - Variety in No. of Paths



- Variety of the number of paths found between different fuzzing sessions
  - Single-Core
    - Showed a great difference between individual fuzzing sessions (1.73x)
  - Multi-Core
    - Stayed quite low (1.13x / 1.33x / 1.33x) for all multi-core sessions
- Multi-Core sessions shared interesting inputs, preventing fuzzing sessions from killing themselves
- Single-Core abruptly killed itself, causing large varieties between individual sessions.

Result 3 - Unique Hangs Found



- Increasing number of cores -> More Unique Hangs
- Search space widened from sharing inputs - no need to recalculate unfruitful input combinations
- Results
    - No unique hangs found for 2 cores
    - Super-linear performance boost (12.4x / 10x) with 4 and 6 cores

Conclusion

- Less than ideal speedups due to…
  - Different instances fuzzing the same inputs and states
  - No deterministic strategy to fuzz different program states on different instances of the fuzzer
- I/O not a bottleneck
- Less variability in results from run to run with more cores

Other Implementations

- PAFL[1]: leverage state information to distribute and confine each instance to a limited state space
- UniFuzz[2]: dynamic centralized input queue
- Enfuzz[3]: different grey-box fuzzers with global state and seed pool

Problems

- Original plan was to implement PAFL based on a paper, but turned out to be too difficult and out of scope
- Hardware limitations
- Fuzzing unstable with single-threaded instances

References

[1] J. Ye, B. Zhang, R. Li, C. Feng and C. Tang, "Program State Sensitive Parallel Fuzzing for Real World Software," in *IEEE Access*, vol. 7, pp. 42557-42564, 2019, doi: 10.1109/ACCESS.2019.2905744. https://ieeexplore.ieee.org/document/8668503
[2] Xu Zhou, Penfei Wang, Chenyifan Liu, Tai Yue, Yingying Liu, Congxi Song, Kai Lu and Qidi Yin. “UniFuzz: Optimizing Distributed Fuzzing via Dynamic Centralized Task Scheduling.” 2020. https://arxiv.org/pdf/2009.06124.pdf
[3] Chen, Yuanliang, Yu Jiang, Fuchen Ma, Jie Liang, Mingzhe Wang, Chijin Zhou, Xun Jiao and Zhuo Su. “EnFuzz: Ensemble Fuzzing with Seed Synchronization among Diverse Fuzzers.” USENIX Security Symposium (2019).
https://www.semanticscholar.org/paper/EnFuzz%3A-Ensemble-Fuzzing-with-Seed-Synchronization-Chen-Jiang/8754951ba8bbb42ff10e100fa853a5ce86af5ab1