Making Threads More Reliabile and Secure through Stable Multithreading

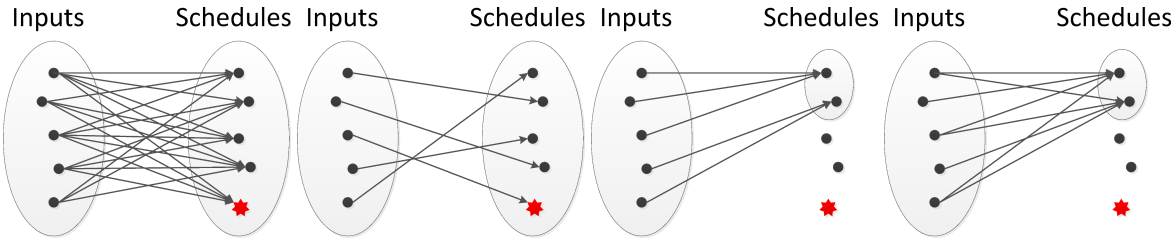
Authors

Multithreaded programs have become pervasive due to the accelerating computational demand and the rise of multi-core hardware. Unfortunately, despite much research and engineering effort, these programs are still notoriously difficult to get right. They are plagued with concurrency bugs, which can easily cause severe program behaviors such as wrong outputs, program crashes, and security breaches.

Our research [7] reveals that a key reason of this difficulty is that multithreaded programs have too many possible thread interleavings, or schedules. Even given only a single input, a program may run into excessive schedules, depending on factors such as hardware timing and OS scheduling. Considering all inputs, the number of schedules is even much greater. Finding a buggy schedule in this huge schedule set is like finding a needle in a haystack, which aggravates understanding, testing, and analyzing of programs. For instance, testing is ineffective because the schedules tested in the lab may not be the ones run in the field.

To reduce the number of schedules for all inputs, we have studied the relation between inputs and schedules of real-world programs, and made a surprising discovery: many programs require only a small set of schedules to efficiently process a wide range of inputs. Leveraging this discovery, we have proposed the idea of stable multithreading (StableMT) [5] that reuses each schedule on a wide range of inputs. StableMT conceptually maps all inputs to a greatly reduced set of schedules, drastically shrinking the haystack, making the needles much easier to find. StableMT can greatly benefit understanding multithreaded programs and many reliability techniques, including testing, debugging, replication, and verification. For instance, testing schedules in such a much smaller schedule set becomes a lot more effective.

StableMT is complementary to another idea called deterministic multi-threading (DMT) that enforces the same schedule on each input. Our research [7] has shown that although DMT is useful, it is not as useful as commonly perceived, because a DMT system can map slightly changed inputs to very different schedules, which can easily cause very different program behaviors. Furthermore, the number of schedules for all inputs in DMT can still be extremely large, which is the key problem that StableMT mitigates.



(a) *Traditional.* (b) *Deterministic.* (c) *Stable (deterministic).* (d) *Stable (nondeterministic).*

Figure 1: Different multithreading approaches. Red stars represent buggy schedules.

Figure 1 shows different multithreading approaches in a conceptual level. Traditional multithreading (Figure 1a) is a conceptual many-to-many mapping between inputs and schedules. DMT (Figure 1b) may map each input to an arbitrary schedule, reducing programs' robustness on input perturbations. StableMT (Figure 1c and Figure 1d) reduces the total set of schedules for all inputs (represented by the shrunk ellipses), increasing robustness and improving reliability. StableMT is complementary to DMT: a StableMT system can be deterministic (Figure 1c) or nondeterministic (Figure 1d). Notably, StableMT has attracted the research communitys interest since it was initially proposed by us [5], and most subsequent systems are both StableMT and DMT.

To realize StableMT, we have built three systems, TERN [5], PEREGRINE [4], and PARROT [3], with each addressing a distinct challenge. Moreover, to justify the benefits of StableMT, we have applied StableMT to several reliability and security topics, including improving precision of static analysis [6], detecting security rule violations [2], and building transparent state machine replication service for general multithreaded programs [1].

References

[1] H. Cui, R. Gu, C. Liu, and J. Yang. Paxos made transparent. In *Proceedings of the 25th ACM Symposium on Operating Systems Principles (SOSP '15)*, Oct. 2015.

[2] H. Cui, G. Hu, J. Wu, and J. Yang. Verifying systems rules using rule-directed symbolic execution. In *Eighteenth International Conference on Architecture Support for Programming Languages and Operating Systems (ASPLOS '13)*, 2013.

[3] H. Cui, J. Simsa, Y.-H. Lin, H. Li, B. Blum, X. Xu, J. Yang, G. A. Gibson, and R. E. Bryant. Parrot: a practical runtime for deterministic, stable, and reliable threads. In *Proceedings of the 24th ACM Symposium on Operating Systems Principles (SOSP '13)*, Nov. 2013.

[4] H. Cui, J. Wu, J. Gallagher, H. Guo, and J. Yang. Efficient deterministic multithreading through schedule relaxation. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP '11)*, pages 337–351, Oct. 2011.

[5] H. Cui, J. Wu, C.-C. Tsai, and J. Yang. Stable deterministic multithreading through schedule memoization. In *Proceedings of the Ninth Symposium on Operating Systems Design and Implementation (OSDI '10)*, Oct. 2010.

[6] J. Wu, Y. Tang, G. Hu, H. Cui, and J. Yang. Sound and precise analysis of parallel programs through schedule specialization. In *Proceedings of the ACM SIGPLAN 2012 Conference on Programming Language Design and Implementation (PLDI '12)*, pages 205–216, June 2012.

[7] J. Yang, H. Cui, J. Wu, Y. Tang, and G. Hu. Determinism is not enough: Making parallel programs reliable with stable multithreading. *Communications of the ACM*, 2014.