Efficient Concurrency-Bug Detection Across Inputs

随着多核技术的普及，多线程软件也越来越流行，然而多线程软件的一个很重要的问题也随之发生，造成巨大的损失，也就是 concurrency bug，解决这个问题，需要高效的 detection 技术以及测试技术。这基于 input 来发现暴露 bug 的拘束在两个方面提出了挑战：（1）能够触发 bug 的 input（输入）；（2）操作的次序问题，同一个 input 里的操作，如果以不一样的顺序执行，会有不一样的结果。

in-house 的技术通常包括以下几个步骤：（1）input 的设计；（2）Bug Detection；（3）Bug Validation。

上述的步骤，已经有很多研究在进行改进和优化了，对于第一个步骤，很多技术已经能够自动的生成代码覆盖率很高的 input，无论是对于单进程还是多进程程序；对于第二个步骤，很多工具已经能够检测出不少类型的交叉存取的 bug，例如 data race，原子性违背等等；对于第三个步骤，也有各种 schemes 能够高效的执行验证受到怀疑的 interleavings

然而这些方法都没有改变第二步和第三步的基础实现，这导致原本就存在的性能损耗没有得到解决，，因此需要高性能的更好的技术 。

本文章的方法通过 input 的集合来对 bug 进行检测和测试，这个方法减少了不同 input 里重复的分析，也减少了多余的 bug report，从而提高了性能。

依据：现有的很多技术，都存在着冗余的现象，同样的 interleave 会在不同的输入被发现以及检测。

- Identifying the inefficiency of existing concurrency-bug detectors on a set of inputs：
  对于不同的系统，设计不同的 test input，主要针对 data race 和单个变量的 atomicity violation 检测
  检测的步骤：To detect these two types of bugs, we first use a run-time tool implemented in Pin to collect per-thread execution traces of global/heap memory accesses and synchronization operations. We then analyze traces to detect bugs. Both detectors were implemented and used in our previous work

  能够识别 C/C++ 程序中的 pthread mutex (un)lock, pthread create, pthread join, and the barrier macro in SPLASH2 benchmarks，但是无法识别自定义的同步操作，这会导致误报。

  在不同的输入中，race 和 atomicity violation 会重叠，这会导致很大的损耗，只要能够识别出这些重叠的部分，就能够节省很多开销。

- Proposing a new interleaving-coverage metric, Concurrent Function Pair (CFP), to guide concurrency-bug detection, as well as a carefully designed algorithm to efficiently measure this metric.
  Metric 的设计需要遵循这两个原则:
  1. Characterization Accuracy: it has to characterize the interleaving space with decent accuracy, so that it can guide concurrency-bug detection to reduce redundant analysis without missing many bugs.
  2. Measurement Complexity: it has to be relatively cheap to measure. Otherwise, its measurement cost would out- weigh its benefit in concurrency-bug detection.

  检测两个函数是不是可以并发的方法：看一个函数的入口能不能夹在另一个函数的入口和出口之间。
  通过分析 run-time trace 来计算 CFP：trace 中有两个属性：entrance/exit 和同步操作，计算氛围两个步骤：（1）对于每一个函数的入口和出口，计算保护他们需要用到的锁集和时间戳，得到函数实例；（2）通过上一个步骤拿到的数据，得到并发的所有函数对。
  但是这种方法在有很多函数调用的长期跑着的程序中，会消耗很长的时间进行分析，因此需要进行优化。
  优化一：跳过只是用了栈上变量的函数；优化二：跳过继承了 caller 函数的并发函数的函数，因为它会与 caller 的所有并发函数并发。但是优化带来了问题：（1）基于"循环必须至少走一次"这样一个想法，会产生

误报，因为一些并发的函数对并不会平行执行，所幸的是这种情况出现的概率比较低，而且并不会造成 bug detection 中的误报问题，所以可以忽略；（2）只基于一些常用的操作，并不支持自定义的同步操作，因此会产生误报

- Designing and implementing a new data-race and atomicity-violation detection framework based on CFP.
  CFP-Guided Bug Detection：
- Compute the CFP of each input and get the aggregated CFP of the whole input set：
  用静态分析得到和 caller 函数有相同的并发性质的函数，计算每个 input 的 CFP，归总 CFP。
- Select inputs and select functions for each selected input：
  用贪婪算法去找到覆盖最多总 CFP 的 input 集合
- Apply a race detector or an atomicity-violation detector to selected functions under each selected input.
  对现有的算法做了一些小修改
  在误报和性能方面都比原来的技术好，在有限的资源下，会有比较少的漏报，但是在资源很多的情况之下，会产生很多漏报，因为某些 bug 只有在特定的输入才会暴露，而 CFP 的 bug detection 针对每一对函数对只会选取少量的 input；同时，有时候，在 atomicity-violation detection 环节 atomicity violation 中的 atomic region 中包含了超过一个函数的代码，这之中带来的 bug 是 CFP 无法检测到的。

CFP 评价：在可能漏掉 bug 的前提下，大大提高了 bug detection 的性能，在资源有限的情况之下是一个好的方案，但是在现实企业生产中，可能因为精力有限无法完全执行，但是其中的 metric 对多线程测试能够带来很好的启发作用