CS636 - Analysis of Concurrent Programs
2020-2021-II: Paper Reading 1

Name: Sarthak Singhal Roll No.: 170635

1. Vindicator

Paper Summary

With the huge increase in parallel programs data races are a threat to the systems. The most used dynamic analysis is HB analysis for detecting data races, but HB only detects races based on a particular run i.e. it misses predictable races that can definitely occur in another execution. There are various sound predictive analysis techniques which can detect more races than HB but they can't scale well i.e. they can't analyze beyond a bounded window. There is an exception to this which is WCP(Weak causally precedes) analysis but it also misses many predictable races. This paper introduces a sound predictive approach, Vindicator, which finds more number of data races than the existing predictive approaches and also scales to full program executions i.e. can detect races whose accesses are far apart in the observed execution.

Vindicator consists of 2 components. The first is DC(Doesn't commute) analysis which covers all the predictable races but may also have some false races. The second is the VindicateRace algorithm that checks each DC-race whether it is true or not. DC relation is similar to WCP, just the difference is that DC only composes with PO and not HB making it a weaker relation than WCP leading to detecting more number of data races(which may be false though) than WCP. VindicateRace runs on each DC-race separately using a constraint graph made using DC eges to analyse whether the race is true or not. The initial graph lacks some contraints due to which we can get a wrong reordered trace. Hence, it adds edges in the graph to ensure that the race events are consecutive and the critical sections are ordered. If there is a cycle in the graph then it returns that it is a false race, else it tries to construct a reordered trace using a greedy algorithm. If the trace is successfully constructed then it is a true race, else the Vindicator doesn't know whether it is true or false race because it uses just a greedy algorithm and not try all the possible orders.

In the implementation, Vindicator together with DC analysis performs WC and HB analyses too to determine whether a DC-race is also WCP-race and HB-race. Vindicator is evaluated on DaCapo benchmarks. It detects races that are missed by WCP analysis. The evaluation also shows that DC-only races have larger event distances than HB-races and WC-only races. It has the highest coverage among all the sound predictive analyses that scale to full execution traces. Although, Vindicator has significant performance costs, but the extra time and memory requirements are worth the cost to detect new data races that are hard to detect due to scalability limits of other predictive analysis approaches.

Key insights/Strengths

- Most of the previous sound predictive analysis techniques don't scale well i.e. they can't analyse beyond a certain window of execution. But Vindicator can detect hard-to-detect races that can be even millions of events apart efficiently.
- WCP analysis scale well as opposed to above point but it misses many predictable races. The new idea of DC relation which is itself originated from WCP relation is a well thought approach that don't miss any predictable race. Even though it can give false races but that is handled by VindicateRace. So, this new approach of DC relation helps to increase coverage by a great extent.
- In the implementation Vindicator also performs WCP and HB analyses. This not only helps in providing a fair comparison of DC analysis but it also helps to check if a race is true or not for only DC-races which are not WCP-races which can help in runtime performance.
- DC analysis does "fast path" that identifies and skips redundant accesses to the same variable by the same thread without interleaving synchronization. This helps in reducing run-time overhead and also the size of the constraint graph. Also, the DC analysis merges adjacent PO-ordered nodes which also helps in reducing the size of graph which in turn improve VindicateRace's runtime.

Weaknesses

- Vindicator does not provide completeness as it can miss predictive data races. It uses a greedy algorithm in constructing a reordered trace but it can fail leading to missing a potential data race.
- Vindicator does not report if there are any predictable deadlocks while one of the previous analysis - WCP, reports both predictable races and predictable deadlocks.
- Each time vindicate race is called the additional constrains to the constraint graph are added from scratch. This can lead to a performance degradation.
- Vindicator has a significant performance overhead over FastTrack which is a commercially used detector. This happens because the DC analysis is far more complicated than the HB analysis.
- FastTrack optimizes HB using epoch and this was a prior work to this paper. Then also Vindicator uses vector clocks in its analysis which leads to performance overhead.

Unsolved problems/potential future work

- The algorithm used for constructing the reordered trace can be improved to take the Vindicator more towards completeness. Some more sophisticated graph based algorithms or ensemble of greedy and brute force approaches can be experimented.

- Necessary heuristics should be developed so that Vindicator can also report if the execution has any predictable deadlock.
- Instead of removing all the edges added to the constraint graph in the end of VindicateRace, the algorithm can be tuend so that the DC-races that are being analysed are reordered before starting calling VindicateRace for each of the DC-races. The DC-races should be reordered in such a way that the adjacent races are close to each other so that some of the additional constraints from execution of VindicateRace for previous race can be used in the graph of the current race. This can help in increasing the runtime performance.
- There are some predictive approaches that can detect races which can't be known using the observed execution. Vindicator can't detect those type of races. In addition to dynamic execution constraints, the control flow constraints can be added so that Vindicator can also detect those races. Although that approach can only be scaled to a bounded window but then also it can help in detecting some additional races that Vindicator is not able to detect.
- The use of vector clock in the DC analysis in Vindicator should be shifted to epoch to get a better performance.

2. SmartTrack

Paper Summary

With the huge increase in parallel programs data races are a threat to the systems. The most used dynamic analysis is HB analysis for detecting data races, but HB only detects races based on a particular run i.e. it misses predictable races that can definitely occur in another execution. There are various sound predictive analysis techniques like WCP(Weak causally precedes) and DC(Doesn't commute) which detect more races than HB analysis but at significantly higher performance cost than FastTrack-optimized HB analysis which is generally used in industry. This paper introduces SmartTrack which incorporates 3 optimizations to the predictive analysis: (1) introduce a weaker relation than DC, (2) use epoch and ownership optimizations from prior work, and (3) introduce optimizations in detecting Conflicting critical sections(CCSs). Using these optimizations, predictive analysis performs comparable to FastTrack in terms of overhead but also finds more number of data races as compared to it.

SmartTrack introduces a new relation WDC(Weak Doesn't Commute) which is same as DC except it removes one of its rule making it a weaker relation. This helps in removing the cost for doing the release-release ordering which requires complex queue operations at every synchronization operation. Due to this WDC will report more races(which will be false) than DC but evaluation shows that it does not report more races than DC analysis. Next SmartTrack applies epoch and ownership optimizations from FastTrack-Ownership(FTO) algorithm. Now, epochs are stored leading to better performance w.r.t both runtime and memory. SmartTrack introduces a new data type critical section(CS) list, which represents the logical times for the release of critical sections by a thread. This helps in representing CCSs in a manner which leads to cheaper logic than prior algorithms for predictive analysis. The insight they had for efficiently detecting CCSs is that they can unify the CCS metadata and the last access meta data for each variable. This is the main idea of the paper which creates a huge impact. The optimized DC and WDC analyses do not perform vindication to avoid the cost of constructing the constraint graph.

The evaluation is done on DaCapo benchmarks and RoadRunner framework is used for implementation. SmartTrack's optimizations improve the performance of all the 3 predictive analyses(WCP, DC, WDC) and even WDC achieves performance(in terms of time) very close to the state-of-the-art HB analysis. Both FTO and CCS optimizations contribute proportionate improvements to achieve predictive analysis with performance close to HB analysis. The predictive analysis finds more data races than HB analysis as they use weaker relations. These results show that the predictive analyses can be practical race detectors that are competitive with standard optimized HB data race detectors.

Key insights/Strengths

- Removing the DC rule which ensures release-release ordering is subtle as it helps to remove complex queue operations at every synchronization operation in turn reducing the execution

time. This approach would have given more data races(false races) than in DC analysis but the evaluation shows that no more data races than DC analysis are reported.

- The idea of borrowing the epoch and ownership optimizations from FTO algorithm is very beneficial. As now epochs are stored, this has decreased both the runtime and memory overheads. The geometric mean of slowdown in terms of runtime for WDC changes from 27X to 12X on using these optimizations. Similarly, the geometric mean of slowdown in terms of memory overhead for WDC changes from 31X to 11X on using these optimizations.
- SmartTrack WDC has a comparable runtime overhead(6.9X) with FastTrack(6.3X) and with that it can detect more number of data races than FastTrack because it uses a weaker relation. So, SmartTrack WDC can become an industry standard after making it comparable in memory overheads also.
- The CCS optimization is a great idea to for increasing the performance in terms of both runtime and memory. The geometric mean of slowdown in terms of runtime for WDC changes from 12X to 6.9X on using these optimizations over FTO optimizations. Similarly, the geometric mean of slowdown in terms of memory overhead for WDC changes from 11X to 6.2X on using these optimizations.
- The SmartTrack's optimizations are very generic such that they could be used on all the 3 predictive analyses given in the paper(WCP, DC, WDC) without much alteration. Also, at the same time they are effective for all the 3 predictive analyses in reducing the runtime and memory overheads.

Weaknesses

- The optimized DC and WDC analyses do not perform vindication to avoid the cost of constructing the constraint graph. They didn't get any false data race for the DaCapo benchmarks but that doesn't mean that there will be no false data races for other real world parallel softwares.
- The memory overhead for SmartTrack WDC is 6.2X while that of FastTrack is 4.9X which is 1.26 times better than SmartTrack WDC. The high memory overhead maybe due to the new analysis state datatype introduced in CCS optimizations. So, some heuristics can be applied to change the datatype so that the memory overhead of WDC also becomes comparable to FastTrack.
- The "ancillary" metadata is used to ensure sound tracking of DC as some CCS information is lost at writes to variables. The ancillary metadata is rarely used, but some programs perform a significant number of checks, which can degrade performance.

Unsolved problems/potential future work

- As the optimized DC and WDC analyses don't do vindication, it should be performed so any false data race is not reported if run on other parallel softwares. A way to do this can be to replay any execution that detects a new DC-race and construct constraint graph

and perform vindication during the replayed execution only. Also, the recent record & replay approaches add very low runtime overhead to record an execution. But as stated in the paper these tools target C/C++ programs while their implementation targets Java programs. So, these tools can be ported to Java so as to use them in the vindication.

- There are 2 recent partial order based analyses, strong-dependently-precedes(SDP) and weak-dependently-precedes(WDP), which have more precise notions of dependence than WCP and DC analyses respectively. But as they don't generally order write-write conflicting critical sections, they make it challenging to apply epoch and CCS optimizations. So, some heuristics should be figured out so that these optimizations or atleast some part of them can be applied to these more precise analyses.