CS 610 Semester 2020–2021-I: Paper Reading

Hirak Mondal

Roll Number - 20111022
hirakm20@iitk.ac.in

Paper 1: ARCHER

Introduction

Parallel Computing has revolutionized the computer industry. The OpenMP API plays a major role in on-node parallelism, but with parallelism there comes risk of data races, which harms the correctness and performance of the program. But present day data race checkers for OpenMP have high overheads and also generates many false positives. In this paper "ARCHER: Effectively Spotting Data Races in Large OpenMP Applications" by Gopalakrishnan et al the authors propose "Archer" a new data race checker that they have claimed to achieve high accuracy and low overheads. While this paper produced significant results showing Archer's capabilities there are limitations related to its memory overhead which has further scope of improvement.

Paper Summary

Archer, the data race checker is built on a pre-exisiting tool named ThreadSanitizer (TSan). Archer exploits OpenMP's structured parallelism so that it can write LLVM passes through which we can detect guaranteed sequential regions within OpenMP and it also suppress parallel loops from race checking by black listing accesses within parallelizable loops by static accesses. The static Analysis passes of Archer helps to classify the OpenMP code regions in 2 major classes: guaranteed race free and potentially racy. And the Dynamic Analysis checks only the potentially racy OpenMP region of code. Archer creates two types of Blacklists in static analysis side, one is Parallel Blacklist where it blacklists the data dependencies and another is Sequential Blacklist, where it blacklists the sequential code regions within an OpenMP code segment. On Dynamic side it uses customised version of TSan to detect data races at run-time. Moreover OpenMP parallelism is typically realized through a PThread Based run-time library. Unmodified TSan gives large number false positives in these cases. So we can pass only one thread at a time inside the critical section to avoid data races. This technique successfully eliminates all false positives. So Archer along

with modified TSan produces a selectively instrumented binary. When tested on OmpSCR Benchmark suite, the researchers observed that Archer detected all the documented races and more interestingly it documented six new previously undocumented races, which when manually verified were found to be true. Whereas its counterpart like IntelInspector XE suffers from accuracy and precision loss. To sum it up, Archer outperforms IntelInspecteor XE across all configuration in most of the Benchmarks. One important observation that the researchers have made is that Archer performs slightly better with static analysis support and it is overall 15% more faster on average. Moreover it was observed that Archer also reduces the memory overheads compared to its counterparts. So Archer with its low run time overheads, high accuracy and precision is proposed in this paper as the best Data race checker for OpenMP that has been developed till this time of writing the paper.

Key insights/Strengths

- **Low Run-Time Overhead, High Accuracy and Precision**

The main strength of Archer is that it provides us with low run time overheads, high accuracy and precision as claimed by the Authors and also demonstrated through various tests using Benchmarks. Run time overheads is a major concern for programs as it exploits machine resources in a negative manner. So reducing run time overhead is definitely a good achievement. On the other hand, high accuracy and high precision as proven from all the benchmark tests can be a major drive force to choose this product over the others that are available in the market. High accuracy and precision provides the user with reliability, so this is also an great achievement that "Archer" has achieved.

- **Elimination of False Positives**

The main problem with the present data race checkers was with False Positives has now been eliminated completely with the help of Archer, which is another key feature that promotes "Archer" over the others. This feature is most important as it helps to increase the accuracy and precision

- **Multiple Benchmark Tests**

Use of different benchmarks like OmpSCR, AMG 2013 for evaluation strengthens the claims of the paper and it was tested on HYDRA as a real world example. Testing software on various benchmarks help us to spot if there is any bug in the software as well as helps us to get a clear view of the accuracy

- **Speed-UP Achieved**

One important observation that the researchers have made is that Archer performs slightly better with static analysis support and it is overall 15% more faster on average. Speed-Up is one of the most important feature as it decreases the runtime, and makes the program faster.

Weakness

- **High Memory Overhead**

Though Archer reduces the memory overhead than all its counterparts, but still its memory footprint appears unnecessarily large, which is because the TSan's runtime shadow memory allocation policy, which Archer inherits unmodified. Larger Memory footprint increases the memory occupied the program and thus consuming more space which is a bad thing.

- **False Negatives**

Moreover papers on the related domain like "Dynamic Data Race Detection for OpenMP Programs" by GU et.al points it out that false negatives is also a issue with Archer which the authors have not mentioned in this paper. False negatives occur when conflicting accesses to shared variables are logically concurrent but are mapped to the same implementation thread. False negatives decreases the accuracy of the program and present of false negatives in such a critical software is not at all desirable.

- **No Detailed Information About Concurrency and Synchronization**

Adding to this, Archer does not keep detailed or any information about concurrency and synchronization. Since different parallel programming models have different primitives for concurrency and synchronization, detailed knowledge about a programming model's primitives is necessary to develop a precise data race detector for it.

- **Not Independent**

Another downside of Archer is that it heavily relies on support for race detection in the LLVM OpneMP run-time and thus is not supported by all the versions of OpenMP.

Unsolved problems/Potential future work

One of the improvement that is needed in Archer is to eliminate the false negatives that it is generating. And along with it we need to take care of the memory overhead as well. One of the main idea that I propose will help in detecting data races is to use Machine Learning, to learn from hundreds and thousands of data samples which areas in the code are more prone to generate data races, and what are the statistical characteristics that they express. We may use a clustering algorithm to cluster the codes which presents same kind of data races into a single cluster and improve Archer based on a specific cluster at a time. This method may overcome the problem with false negatives. To overcome the shadow memory overhead for an initialized array, we can copy the array to another memory location and can access the array with pointer variables. Thus TSan now don't have to create the entire memory shadow. Further research on Archer to overcome this deficiencies is necessary to provide the impact of the solutions proposed.

Paper 2: ROMP

Introduction

Parallel Computing has revolutionized the computer industry. The OpenMP API plays a major role in on-node parallelism, but with parallelism there comes risk of data races, which harms the correctness and performance of the program. But present day data race checkers for OpenMP have high overheads and also generates many false positives and false negatives. In this paper "Dynamic Data Race Detection for OpenMP Programs" by Yizi Gu and John Mellor-Crummey the authors propose "ROMP" a tool for detecting data races in execution of scalable parallel applications that employ OpenMP for node level parallelism. While this paper produced significant results showing ROMP's capabilities but there are limitations related to it as well.

Paper Summary

ROMP is a hybrid data race detector that tracks access,access orderings and mutual exclusion. ROMP detects data races with respect to concurrency rather than implementation threads, unlike other OpenMP race detectors. The authors claims that building a precise race detector needs detailed information about concurrency and synchronization, and determining only the false positives as shown by the previous work cannot be termed as a precise race detector if it dont have these features present. Data Race checkers for OpenMP that are available in the market employs thread level schemes for labelling concurrency and checking data races, which are not able to avoid false negatives. ROMP tries to solve this issue by performing in what the authors have mentioned as 'on-the-fly' mode of race detection for OpenMP that tracks logically concurrent OpenMP tast intervals instead of thread level concurrency. Moreover the hybrid data race detection of ROMP combines happens-before analysis and lock set analysis. To achieve powerful hybrid race detection ROMP employs a variant of All-Sets algorithm, adapted to support data race detection during an parallel execution of an OpenMP program. The major difference between ROMP and and All-Sets is that ROMP cannot use pruning criteria used by All-Sets because all sets assumes pseudo transitivity of the parallel relation between memory access events. When tested on Benchmark Suite "indirectaccess2—3—4_orig-*.c" it showed that ROMP detects all the data races whereas Archer and SWORD[8] yields false negatives. The main point to note is that ROMP's algorithm is based on tasking and does not suffer from the False Negative problem of thread based data race detection algorithm. All the results of the report demonstrates the high effectiveness of ROMP achieved due to the innovative task labelling mechanism which analyzes logical concurrency between tasks. ROMP even decreases the space overhead as compared to its counterparts. So ROMP with its high precision and accuracy is proposed in this paper as the best Data race checker

for OpenMP that has been developed till this time of writing the paper.

Key insights/Strengths

- **Not Dependent on LLVM OpenMP run-time for race detection**

Unlike Archer which relies on LLVM OpenMP run-time for race detection, ROMP lives on top of OMPT callbacs and so it works with all OpenMP run-time library that supports OpenMP 5.0 standard.

- **Improved Precision than other softwares available in the market**

ROMP improves precision over Archer by reasoning about logical concurrency.

- **Reduced Space Overhead**

For some benchmarks like OmpSCR ROMPs space overhead is 2.5x smaller than that of Archer. The less space overhead, the better it is. So its a good feature of ROMP

- **Precise Detection of False Negatives**

Moreover ROMP is also taking care of the false negatives which was being skipped by Archer and other data race detectors. Taking care of false negatives increases the precision and accuracy of ROMP. So its a great feature to have.

- **More Focused on Task Level than thread level concurrency**

The main strength of ROMP is that it tracks logically concurrent OpenMP task intervals instead of thread level concurrency.

- **Dynamic Race Detection thereby decreasing run time space**

ROMP is a Dynamic data race detector thereby it uses 'on-the-fly' race detection mechanism, which minimizes the extra space needed at the runtime to check for races. As stated earlier, the less space it takes, the better the software is. So it is definitely something which adds to the advantage of ROMP

Weakness

- **NO SIMD Support**

ROMP doesn't support SIMD as mentioned by the authors. So Single Instruction Multiple Data are not of any use to ROMP.

- **NO thorough checking for dynamically loaded shared libraries**

Moreover ROMP does not apply thorough checking to all dynamically loaded shared libraries used by an application.

- **Slow Program execution due to the use of multiple threads**

Another drawback that is present in ROMP is that widely shared data that are concurrently accessed by Multiple threads which slows the program execution as ROMP protects the shadow memory slots using mutual exclusion.

- **Keeping Track of Read-Only data**

Adding to the weakness is that, ROMP even maintains access histories for read-only data, but read only data can never cause race condition. So it is just an overhead.

Unsolved problems/Potential future work

The access histories for widely shared data contain more entries than necessary to detect representative races. This problem may be solved or can at least can be partially solved if we use 'Feature Selection' algorithms of Machine Learning to cut out the the entries which are more relevant to provide us with information. We can also use "information gain" and "entropy" to ensure which data are more important to us and which are not, thereby pruning them. Another point is that ROMP is also maintaining the access histories for read-only data as well, for which race condition is impossible. So we can stop considering read only data to improve the performance of the software.