Paper Review 1

Ayush Kumar (170195)

25 October 2020

I pledge on my honor that I have not given or received any unauthorized assistance.

1 ARCHER: Effectively Spotting Data Races in Large OpenMP Applications (Atzeni et al., 2016)

1.1 Summary

The paper investigates existing state of the art Data Race Checkers such as **ThreadSanitizer (TSan)** and **Intel®Inspector XE** used for checking concurrent accesses that can cause inconsistencies in parallel programs. It also introduces ARCHER, a new data race checker which supposedly not only outperforms the existing checkers in both the metrics runtime and memory, but is also able to find new concurrency bugs in existing scientific computing libraries which were never documented. It exploits OpenMP's structured parallelism (parallel regions demarcated using openmp pragmas) to perform static and dynamic analyses of the code more efficiently. It builds upon TSan's dynamic race checker by using its **Annotation API** to highlight implicit/explicit synchronization points (*barrier*, *critical*, *atomic*, *taskwait* etc.) which, as the paper claims, removed several false positives that were earlier reported by TSan.

ARCHER first performs static analysis which includes Sequential and Parallel Blacklisting. The former blacklists serial regions of the code i.e. code outside the openmp parallel pragmas, while the latter blacklists serial code contained in the parallel region such as code within loops carrying dependencies. This is identified using static analysis tools such as Polly. After this, ARCHER performs dynamic analysis using the customized version of TSan which offers finer granularity and can handle blacklisted code not necessarily demarcated by function boundaries. Finally, the paper evaluates ARCHER as well as well as TSan and Intel®Inspector XE using default and customized settings on different benchmarks such as OmpSCR, AMG2013 and real-world libraries such as HyPre and conclude that ARCHER performs the best. The paper claims that ARCHER achieves the highest accuracy and precision with an F1-score of 1.0.

1.2 Key Contributions

Some of the key contributions of the paper are as follows:

- One of the most important features of OpenMP is the use of pragmas to keep sequential regions of code separate from the parallel regions. The paper is one of the first to aptly utilize this to its advantage and is easily able to blacklist regions of code not involved in data races. As an instance, the use of synchronization pragmas such as *barrier*, *critcal*, *atomic*, and *taskwait* allows ARCHER to easily determine sequential parts inside a parallel region.
- The paper is able to capitulate on the existing data race checker TSan and the authors even modify it to support fine-grained blacklisting. Earlier, the Annotation API of TSan was only able to blacklist regions of code demarcated by function boundaries. With the new extension created by the authors, blacklisting can be performed at a finer-grained level of memory accesses. This extension could prove crucial to future works in this field.
- The evaluation performed by the paper is quite comprehensive and full-proof. Instead of simply presenting results obtained on standard benchmarks, the authors evaluated ARCHER on real-world scientific applications such as HYDRA which quickly showcases what an invaluable tool it can be to developers and testers of parallel applications. In some cases, it was even able to find new data races which were unknown till now, and to support their findings, the authors manually verified it.

1.3 Weaknesses

- The paper claims that ARCHER is able to find new data races in benchmarks such as OmpSCR and AMG2013 which were unreported till now. In this context, it is reasonable to investigate why existing data race checkers weren't able to find them. This might have uncovered some key findings that could have helped future application developers and well as data race checkers. The paper presents no specific explanation as to why these new data races had remained unreported.

- The paper does not address the problem imposed by optimizing compilers. More specifically, it was found that in particular cases a data race which seemed benign (such as multiple threads writing 0 to the same memory location) could prove erroneous due to optimizations performed by some compilers. To get around the problem, the authors simply disabled optimizations using the **-O0** flag, but this does not seem like a viable solution in the long run. The authors could have explored the exact cause and suggested improvisations.

1.4 Future Work

- Currently, the static analyser of ARCHER blacklists regions of code by classifying code segments into Sequential and Parallel categories. In future, the parallel category could be further subdivided using a more finer static analysis that could find sub-regions of code (in the parallel region) that are definitely race-free.

- Another future work could be exploring the cause of large memory footprints of ARCHER. The authors believe it is caused by TSan's runtime shadow memory allocation policy. But this needs to be verified, it could be resolved using a selective deallocation policy of deallocating array locations once they become stale.

- ARCHER does not support OpenMP 4 including features such as the new *device* construct which allows threads to run on GPUs. Even though at the time of writing this paper, no compiler fully supported the new OpenMP 4 constructs, current compilers have inbuilt support for them. In future work, ARCHER could be extended to handle them as well.

2 ROMP: Dynamic Data Race Detection for OpenMP Programs (Gu and Mellor-Crummey, 2018)

2.1 Summary

The paper introduces **ROMP**, a new on-the-fly Data Race Checker for OpenMP parallel programs. The main distinction between ROMP and other state of the art checkers is that it uses a novel **task-graph model** which focuses on concurrency of memory accesses using graph-reachability. The paper claims that this new design enables ROMP to avoid several false negatives by considering logical concurrencies among work sharing loop iterations performed by the same thread. This wouldn't be the case with thread-centric designs in other checkers such as ARCHER or TSan as they only argue or reason about data races among concurrent implicit/explicit tasks being performed by separate threads. The paper also claims that in comparison to others, ROMP is able to handle a broader range of OpenMP constructs including several synchronization and task-sharing constructs such as *ordered*, *section*, *taskwait* and *taskgroup*.

The task-graph model reduces the problem of finding concurrent memory accesses to finding a path between the corresponding task vertices. ROMP uses an extension of offset-span labelling for assigning a task label to each vertex in the directed acyclic task-graph. This task label is a sequence of label-segments containing the history of the task-creation nest and is obtained by appending a label-segment to the parent task at the time of spawning. These task labels are ultimately used by ROMP to check for concurrencies by finding the first differing label segments in two task labels and then reasoning using the information stored in them. The paper also briefly discusses how ROMP handles task-dependencies, reductions, critical sections and shared/private data. Finally, the paper gives a high-level overview of the implementation and then evaluates ROMP on benchmark programs such as DataRaceBench and OmpSCR and compares the results obtained with ARCHER which was found to be the most effective data race checker in another study.

2.2 Key Insights/Contribution

Some key insights from the paper are summarized as follows:

- The task-graph model is one of the most significant contributions of the paper. This approach is inspired from a recent paper by Agrawal et al. (2018) which proposes a DAG reachability based data race detection algorithm. ROMP augments this algorithm with access history pruning for parallel execution and an extension of offset-span labelling. A graph-based approach not only makes reasoning easier (considering our vast knowledge of graph algorithms) but also makes the approach scalable and amenable for modifications in future works.
- The authors worked with the OpenMP language committee to extend the OMPT tool interface for OpenMP 5 and added the necessary hooks/callbacks (implemented in ROMP) which are invoked by the OpenMP runtime. The addition of such callbacks makes it much easier to implement task based concurrency tracking for future OpenMP based data race checkers.
- The authors propose stricter definitions for metrics TP (True Positive), TN (True Negative), FP (False Positive) and FN (False Negative) claiming that the earlier definitions were more relaxed in the context of multiple runs of a single benchmark program. For instance, a race free program should be classified as TN only if no races are reported in all the runs. In contrast, earlier definitions could label the same program as TN even if it reports races in some runs and doesn't in other runs.
- For better evaluation of OpenMP directives, the paper also introduces 12 new benchmark programs containing synchronization constructs such as *ordered*, *section*, *taskwait* and *taskgroup* which were not present in earlier benchmarks and hence had remained untested.

2.3 Weaknesses

- ROMP doesn't support race detection for OpenMP SIMD constructs since it is an on-the-fly race detector and hence the original scalar code is not available to determine if a data race was ignored during possible vectorization of the scalar code.
- During evaluation and subsequent comparison of ROMP and ARCHER, the benchmark programs were compiled using the **-O2** optimization flag for clang, while as stated in (Atzeni et al., 2016), ARCHER runs best without compiler optimizations. Hence, this could prove to be an unfair evaluation scheme for ARCHER.

2.4 Future Work

- Currently, ROMP doesn't apply race checking of linked shared libraries. In other words, shared libraries included in the application are assumed to be race free. But this might not always be true, and a thorough checking scheme for dynamic libraries should be incorporated.
- As part of access history pruning, due to multiple memory accesses of the same address, multiple threads may race to inspect and update an access history. To prevent this race condition, ROMP employs an MCS queueing lock. If a memory address is widely read shared, a significant runtime overhead is caused due to contention of this queueing lock. Using a reader-writer lock or refining the pruning algorithm to reduce this overhead could be a subject of future work.

References

Agrawal, K., Devietti, J., Fineman, J. T., Lee, I.-T. A., Utterback, R., and Xu, C. (2018). Race detection and reachability in nearly series-parallel dags. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '18, page 156–171, USA. Society for Industrial and Applied Mathematics.

Atzeni, S., Gopalakrishnan, G., Rakamaric, Z., Ahn, D. H., Laguna, I., Schulz, M., Lee, G. L., Protze, J., and Müller, M. S. (2016). Archer: Effectively spotting data races in large openmp applications. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 53–62.

Gu, Y. and Mellor-Crummey, J. (2018). Dynamic data race detection for openmp programs. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 767–778.