Paper Reading 1

Aniket Sanghi (170110)

29th October 2020

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

1 Paper 1 - Critique

Summary

The paper presents an OpenMP thread-centric, dynamic data race detector - **ARCHER** which exploits OpenMP's structured parallelism and modified state-of-the-art race detectors (TSan) to accurately and with high precision detect data races in OpenMP applications. ARCHER also proved its practicality and scalability by detecting races(even benign) in real-world HPC codes (e.g. HYDRA).

ARCHER detects data races by combining well-layered modular static and dynamic analysis stages. It uses static analysis to help classify the code into *guaranteed race free* and *potentially racy* categories, and then feed the potentially racy code region to on-the-fly race detection algorithms in dynamic analysis giving a good speed up and precision boost. In the static analysis phase, it applies data dependence analysis (using a tool called Polly) and sequential code detection to detect and blacklist the guaranteed race-free regions so obtained. The blacklists are communicated to the modified TSan which then exploits them to enhance precision and improve running time in the dynamic analysis phase where TSan detects races during runtime by analyzing loads/stores information.

ARCHER is evaluated in comparison with Intel@Inspector XE on the OmpSCR Benchmark suite and ARCHER not only gives high precision and accuracy, but it also detects some previously-unknown races in the suite. It also had low memory overhead and runtime on almost all cases (with few exceptions) as compared to Intel@Inspector XE. It also outperformed Intel@Inspector XE on the AMG2014 case study with a bigger codebase of around 75000 lines. ARCHER also resolved a real-world code crash in a project named HYDRA by detecting races in the code.

Key Insights & Strengths

- **Easy to understand**, they have explained the idea, approach, and evaluation in very descriptive and diagrammatic format. The approach has been explained with ease using diagrammatic representation and by focusing on only the major sections. The evaluation section beautifully depicts all results in carefully plotted graphs, performance plots and major issues have been explained in simple understandable words.
- **Innovative idea**, the idea of combining static and dynamic analysis is very innovative. The idea of separating guaranteed race-free regions beforehand helped them speedUp and enhance precision, basically by reducing a lot of workload from the dynamic analysis part.
- **Real-life value**, they not only evaluated their tool on benchmarks but also on real-world problems. They gained a lot of trust in its practicality and scalability by solving the real-world HYDRA race crash problem.
- **Thorough evaluation**, they have evaluated the detector's performance thoroughly on all the properties provided by ARCHER as they tested by removing static analysis, removing parallel blacklisting, removing serial blacklisting, and even using un-modified TSan. They thoroughly evaluated each algorithm/optimization they proposed on various metrics and manually confirmed each new-race encountered.

Weaknesses

- **Missing Details**, Paper hasn't explained various parts of the approach in detail. Like it doesn't talk much on the Sequential code section detector or about what exact modification was applied to TSan. Some pseudo-codes expressing details would have been very beneficial in reviewing the approach better.
- **Suboptimal Algorithm**, the approach is prone to give false negatives in some scenarios, since they used TSan as their dynamic race detector which is thread-centric by design. It can give false negatives in scenarios like when 2 logically concurrent operations are performed by the same thread, TSan will call it okay but in some other scenario, these operations could have raced with each other.
- **High space overhead**, TSan hasn't been modified from its memory handling perspective. TSan uses shadow memory for variables and doesn't selectively deallocate memory, for instance, based on whether they are live beyond a certain point. This leads to high memory overhead which could have been minimized further.
- **Not generic**, ARCHER isn't generic enough to work well with the new OpenMP versions. It is limited by its thread-centric design and also on its capabilities to use OMPT callback routines to correctly detect races in codes with various new pragmas like task-wait etc.

Unsolved Problems & Potential future work

- **Reduce Memory Overhead**, Modify TSan's memory management by actively tracking and deallocating shadow memory of variables (particularly large arrays) when there is no further use for their memory location. Currently, since it doesn't deallocate shadow memory for them, the overall memory overhead is quite high than optimal (that could be achieved)
- **Extend support for OpenMP 5** Currently ARCHER only supports pragmas and features till OpenMP version 3.1 only. Extension to the current features of explicit tasking and all have to be made.
- **Using Task-centric detector**, Currently, it uses a thread-centric design of TSan because of which we can get false negatives in some scenarios as explained in the weakness section above, we can instead use a data race detector which is task-centric by design to avoid false negatives and be more generic.
- **Static Analysis algorithm improvement**, it currently classifies data into two categories of race-free or potentially racy, fine grained advanced techniques could be applied in each of these to detect race-free sub-regions within them to improve performance even further.

2 Paper 2 - Critique

Summary

The paper presents a new OpenMP task-centric on-the-fly data race detector **ROMP**. It is a hybrid data race detector that maintains happens-before orderings, locksets, and memory access history to reason about the data race in a code. It is the first of its type that uses a task-centric scheme to detect races rather than thread-centric, i.e. it can even detect races where the two conflicting accesses were performed by the same thread.

ROMP reasons about happens-before orderings between memory accesses by using two strategies. For structured parallelism, it employs a hybrid data-race detector which is a modified version of *All-Sets* algorithm, with memory pruning modified so as to make it race-transitivity condition free. Then for unstructured parallelism like for explicit task construct, it first formulates a task graph model from the code where each node is given a unique label containing task-specific information regarding various OpenMP constructs, and then these are used to reason about concurrency in the ROMPs hybrid data race detection algorithm.

ROMP tracks the data environment by maintaining a Shadow Memory which contains the access history for each memory address. The edit-access to each memory location is constraint by using queue lock and lock contention is used as a quicker way to report race for many scenarios. It also handles the various synchronization constructs of OpenMP by exploiting the callbacks (Metadata) received from OMPT at the entry/exit/execution of the various synchronization directives. It also had added few ROMP specific callbacks to OMPT in OpenMP 5.0.

ROMP has been evaluated with all other state-of-the-art race detectors of the time and it proved to be the most accurate and precise mainly because of its task-centric design. It also had a 2.5x lower memory overhead as compared to then the best OpenMP data race detector ARCHER.

Key Insights & Strengths

- **Self-contained paper**, the paper contains almost all details from the mathematical formulation of the problem to a detailed description of the algorithm along with pseudo codes which makes the paper self-contained enough for any person outside the field of study to understand and review the paper.
- **New idea**, ROMP went with a task-centric design even though most of the then state-of-the-art data race detectors were thread-centric, and eventually got successful in devising a more precise detector with it.
- **Well Engineered**, the algorithm has been very optimally modified taking care of reducing memory overhead wherever possible like by deciding what and what not to store in the memory access history. Also, they used lock contention in Shadow Memory very optimally in detecting a possible data race directly and reducing the memory overhead and slow down further.
- **Parallel Detection**, most of the dynamic data race detectors require a sequential execution of the code to detect all possible races but in ROMP it allows a parallel execution of the code, thus giving better performance.
- **Coverage**, the algorithm covers all of the OpenMP constructs(except SIMD) including the various new synchronization constructs as well, the support of which weren't present in many other state-of-the-art detectors.

Weaknesses

- **Suboptimal Algorithm**, ROMP maintains an access history of read-only data as well which is a waste of memory as read-only data can never give race conditions. This further degrades the performance in shadow memory management leading to high memory overheads and suboptimal runtime.
- **False Negatives**, as seen from the evaluation, in case of static scheduling of DataRaceBenchmark codes 5-8, ROMP gave false negatives due to an incomplete logical task label.
- **Numerical evaluation**, they have used more tabular data to provide us with evaluation metrics instead of performance plots that are more easy to compare and judge accordingly. Histograms, line plots help understand the statistics better and compare results more precisely.

Unsolved Problems & Potential future work

- **Static Analysis**, the algorithm stores more than necessary data for highly shared variables in the shadow memory it maintains. It further degrades because of the queue lock for all these to make their entries. This can be improved if we perform static analysis before the dynamic analysis and limit the access history recorded.
- **Read-only**, ROMP maintains an access history of read-only data also, which is a waste of memory as read-only data can never lead to a data race. This can also be resolved by using static analysis.
- **SIMD support**, Currently ROMP doesn't support the SIMD construct of OpenMP, primarily because after vectorization it gets hard to detect the racy regions. We can extend support for this which will not be straight-forward and may involve stricter compiler directives to be used while running the race-detector.