Efficient Datarace Detection in a Structured Programming Language

Trudy Firestone
School of Computing
University of Utah
50 S. Central Campus Drive
Salt Lake City, UT 84112
u0674851@utah.edu

Ganesh Gopalakrishnan (Advisor)
School of Computing
University of Utah
50 S. Central Campus Drive
Salt Lake City, UT 84112
ganesh@cs.utah.edu

ABSTRACT

Datarace detection is increasingly important in Computer Science as non-specialized computer scientists are required to write more parallel programs for better performance. Because dataraces are an extremely common bug, detection of these conflicts is a necessary debugging tool. Unfortunately discovering dataraces is computationally expensive. However, semantics guarantees provided by structured parallelism make it possible to create an efficient solution. To this end, we look to expand the race detector in the structured language, Habanero Java (HJ). Building upon HJ's current race detector, which supports async/finish constructs and isolated regions, this research theoretically extends HJ's algorithm to include the phaser construct. [1]

Categories and Subject Descriptors

D.1.3 [**Concurrent Programming**]: Parallel programming; D.2.3 [**Coding Tools and Techniques**]: Structured programming

General Terms

Race Detection; Datarace; Phasers

Keywords

Races; Debugging

1. BACKGROUND

Dataraces are a large debugging problem in the world of parallel computing. They can be easy to miss and evade normal means of debugging because they are dependent on the schedule and the number of threads. Besides the difficulty involved in creating an efficient datarace detector [2], there is also some confusion over what constitutes a datarace. Netzer and Miller succinctly and formally define dataraces as "nonatomic execution of critical sections" [2].

[1]This research was supported by NSF REU Award 1321642.

The HJ datarace detector described by Raman et al. leverages the power of the structured language to efficiently manage time and memory [3]. They describe an algorithm that makes use of a Dynamic Program Structure Tree (DPST) which allows the datarace detector to find out if two steps may happen in parallel with only a relatively small slowdown (a mean of 2.78X) [3]. The algorithm requires a constant amount of "shadow memory" for each memory location used in the program, a relatively small amount [3]. Furthermore, it reports no false dataraces—saving hours of wasted debugging time—and it reports every possible datarace for a given input [3]. While there are other structured parallel programming languages such as Cilk and X10, HJ is easily accessible due to its reliance on Java [3].

2. CONTRIBUTIONS

We seek to expand the detector to uncover dataraces when phasers are used. Phasers are an HJ construct that allow additional parallelization of a program by creating flexible barriers with multiple registration modes (signal/wait, wait-only, and signal-only) [1].

Phasers can be included in the DPST by simply adding an extra type of node. This node will represent the barrier operations associated with the HJ keyword *next*. With this extra node there are several language properties that will guarantee whether the steps may or may not execute in parallel. Following along with the algorithm presented by Raman et al., our algorithm must first determine if the leftmost child of the least common ancestor(LCA) that is an ancestor of one of the two steps is an async [3]. If this is not the case, the two steps will not be able to execute in parallel, and the algorithm is complete for those two steps. However, if the right child is an async, the two steps may be able to execute in parallel and further checking is required.

Once the above is confirmed, phasers come into play. The only part of the algorithm with substantial changes is the determination of whether or not two steps may happen in parallel. Phasers introduce extra synchronization restrictions. For instance steps that happen in two different tasks after a *next* statement may not be able to happen at the same time as those before it, depending on the registration. This research devises an algorithm for determining whether or not these barrier conditions are met.

3. FINAL REMARKS

In the future, we hope to add this algorithm to HJ's race detection system and explore the efficiency of this algorithm through benchmarks and optimizations.

4. REFERENCES

[1] Cavé, V., Zhao, J., Shirako, J., and Sarkar, V. Habanero-java: the new adventures of old x10. In *Proceedings of the 9th International Conference on Principles and Practice of Programming in Java* (2011), ACM, pp. 51–61.

[2] Netzer, R. H., and Miller, B. P. What are race conditions?: Some issues and formalizations. *ACM Letters on Programming Languages and Systems (LOPLAS) 1*, 1 (1992), 74–88.

[3] Raman, R., Zhao, J., Sarkar, V., Vechev, M., and Yahav, E. Scalable and precise dynamic datarace detection for structured parallelism. In *ACM SIGPLAN Notices* (2012), vol. 47, ACM, pp. 531–542.