Efficient Datarace Detection in a Structured Programming Language

Trudy Firestone, Ganesh Gopalakrishnan (Advisor)

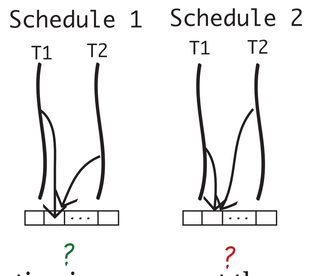
THE UNIVERSITY OF UTAH

CENTER FOR PARALLEL COMPUTING @ UTAH

This research was supported by NSF REU Award 1321642

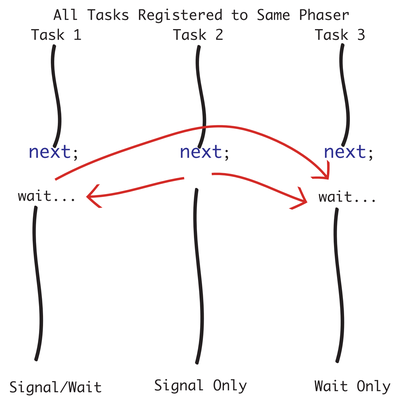
Background

- Many programmers struggle with parallel programming due to its often nondeterministic nature
- Data races (such as the one shown to the right) are bugs that can easily go unseen as they may pass many traditonal tests
- Data Race:
  - Two threads access the same location in memory at the same time
  - At least one of the threads performs a write on the location
- In general, data race detection is an NP-hard problem [2]
- Languages like HJ (e.g., Cilk and X10) are structured to provide support for parallel programming–making race detection significantly simpler [3]



Focus: Phasers

- Provide point-to-point synchronization [1]
- Registered on one or more async tasks
- Can be registered in 3 different modes
  - Signal/Wait
  - Wait Only
  - Signal Only
- Can additionally include a statement to be executed once by one of the tasks in the barrier formed by `next;`



Algorithm

Determine if two steps may happen in parallel:

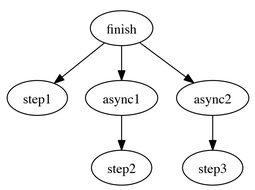
```
Traverse up the DPST to the two steps' LCA {
  During the traversal keep a count of each
     phaser's signals and waits (barriers);
}
if (steps cannot happen in parallel according to
              the Cave et al. algorithm){
  if(tasks have no phasers registered or different
                 phasers registered) {
    Steps may happen in parallel;
  }
  for (each phaser that both tasks are registered
                  on) {
    if ((number of signals between step1 and LCA >=
                     the number of waits between
                     step2 and LCA) or (number of
                     signals between step2 and LCA
                     >= the number of waits between
                     step1 and LCA)) {
            Steps may happen in parallel;
    }
  }
    Steps cannot happen in parallel given the same
                     input;
}
else {
  Steps cannot happen in parallel given the same
                     input;
}
```

Data Race Detection in HJ

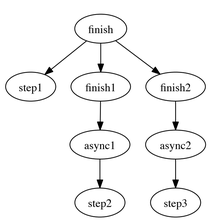
- Dynamic Program Structure Tree (DPST) [3]
  - Contains a node for each step that has no parallel constructs
  - A node for each async and finish construct
  - The root of the tree is always the finish node that encloses the main method
  - Each time an async is encountered a new branch splits off
- Determining Dynamic May Happen in Parallel (DMHP) [3]
  - Find the least common anscestor (LCA) of two steps
  - Find the left child of the LCA that is also an anscestor of one of the steps
  - If that child is an async, the steps may happen in parallel

```
public static void main (String[] args)
{
     AnyObject = new AnyObject(20);
     async {AnyObject.field = 21;}
     async {AnyObject.field = 22;}
}
```

The DPST on the right demonstrates a data race: step2 and step3 can occur concurrently and they write to the same memory location.



The below DPST shows the lack of a data race in the accompanying code. While steps 2 and 3 still write to the same memory location, they cannot occur concurrently.

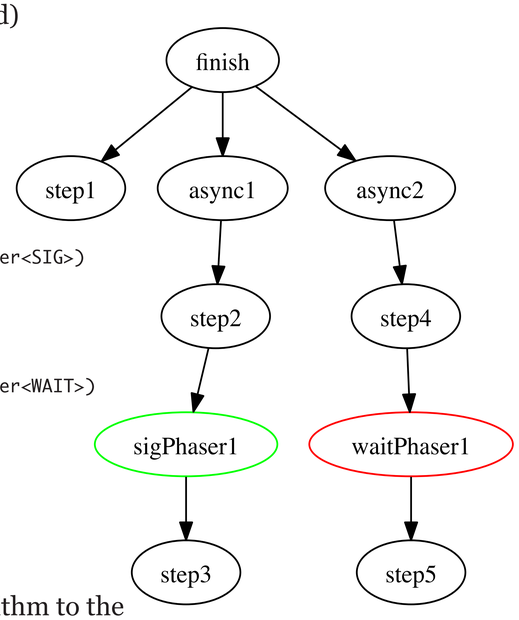



```
public static void main (String[] args)
{
     AnyObject = new AnyObject(20);
     finish
     {
          async{AnyObject.field = 21;}
     }
     finish
     {
          async{AnyObject.field =22;}
     }
}
```

Contributions

- Modified DPST to account for phaser constructs
- Created algorithm to work with modified DPST to detect Data Races
  - Follows the original HJ algorithm (with some additional record keeping)
  - Then adds additional support for phasers (by changing the DMHP method)

```
finish
{
   phaser = new
phaser<SIG_WAIT>();
   //step1
   async phased (phaser<SIG>)
   {
      //step2
       next;
      //step3
   }
   async phased (phaser<WAIT>)
   {
      //step4
       next;
      //step5
   }
}
```



Following the algorithm to the upper right:

- step1 and step2 cannot happen in parallel
- step2 and step4 may happen parallel
- step3 and step4 may happen parallel
- step2 and step5 may not happen in parallel
- step 3 and step5 may happen in parallel

Further Work

- Implement the modified DPST and algorithm by adding it to HJ's source (i.e. modify ESPbags.java and RaceDetInstrumentor.java in the HJ source files)
- Test the modified algorithms against benchmarks as done by Cave et al.

References

[1] Cave , 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.