1. **Title**  
Scalable dynamic race prediction

2. **Applicant(s)**  
Vector Fabrics  
Vonderweg 22  
5616 RM Eindhoven  
The Netherlands  
http://www.vectorfabrics.com

Contact person: Stefan Holdermans  
Phone: +31 40 820 09 60  
E-mail: stefan@vectorfabrics.com

3. **Company information**  
Vector Fabrics helps programmers write software for multicore and many-core processors. Its multicore programming tools analyse the dynamic behaviour of programs, detect multithreading bugs, and enable parallelism even where concurrent execution is nonobvious.

Vector Fabrics licenses its tools and technology towards software vendors, OEMs, and semiconductor vendors. Its tools have been deployed to optimise web-browser engines, physics engines, game engines, video codecs, and image-processing applications.

4. **Problem**  
Among the most hard-to-debug defects in multithreaded programs are data races.

According to the most common definition, a data race occurs if

- two operations access the same memory location from different threads of execution,  
- at least one of these operations updates the memory at that location, and  
- the two accesses are not separated by any synchronisation events between the two threads.

This notion is captured more formally by Lamport (1978) in his definition of the so-called happens-before relation (HB).

Following the definition given above, data races appear in executions of programs rather than in programs themselves. A program admitting one or more racy executions is said to have a race condition or a race hazard. Hence, race conditions are attributes of programs, while actual races are attributes of executions. As a result, a single program with a race condition may have multiple executions, some of which do exhibit data races, while others do not.

A dynamic race detector observes the execution of a program and reports whether or not the execution exhibited a data race. For a race detector to be useful in practice, it is important that it never produces a false positive, i.e., that it never reports a race for an execution that did not exhibit one. Note that an HB-based race detector, in general, reports different results for different executions of the same program.

HB-based race detectors are typically implemented in terms of so-called vector clocks (Mattern, 1989; e.g., Pozniansky and Schuster, 2003). Vector clocks allow for a very precise tracking of the happens-before relation, but their naive use is very expensive in terms of both time and memory behaviour. As a result, most HB-based race detectors do not scale well.

The state of the art focuses on avoiding the full generality of vector clocks in order to make HB-based race detectors more scalable (Flanagan and Freund, 2009) and, more recently, on variations on and extensions of the HB-based approach in which, in addition to detecting data races that actually occur in the execution under analysis, predictions are made about which other races may occur in a set of executions that is obtained by conservatively reordering the events of the analyzed execution (Said et al., 2011; Smaragdakis et al., 2012; Huang et al., 2014).

Unsurprisingly, keeping an HB-based analysis scalable is even more challenging for a race predictor than it is for a mere race detector. The aim of this project is therefore to devise an approach to predictive dynamic race analysis that at the same time

1. based on a single execution of a program, maximises the number of identified race hazards in the program,  
2. scales well enough to be able to analyse large real-world applications, and  
3. never produces false positives.
REFERENCES


FURTHER READING


5. Input
Dynamic race analyses operate on abstract execution traces that result from recording relevant events (such as memory accesses and operations on locks) during the execution of a program. Vector Fabrics will provide a collection of example traces that may drive the design of a new race-analysis algorithm.