Review of Software and the Concurrency Revolution by Herb Sutter and James Larus

Riley Wood

March 29, 2016

Key Ideas

The main goal of this paper is to describe the necessary trajectory for programming languages and compilers given the recent trend toward multiprocessor systems. Sutter and Larus highlight several key shortcomings in how concurrenct is exposed to programmers today; they also summarize existing approaches to these problems as well as suggest what they believe to be the right solutions. They describe how the global nature of synchronization locks defies modularity and makes it hard to blackbox functionality, because a function that secretly uses a lock could cause deadlock to occur. Locks also have no easy way of being coupled with the data they protect - it is simply up to the programmer to remember how locks and data are correlated. At the time of writing, programming languages were also not capable of catching the new programming errors that concurrency introduces, such as deadlock, livelock, data races, and others.

Review

Reading through the article, I see that some of the problems addressed have been addressed since the article was published. Other problems have yet to be solved. After reading over the different categories of approaches to concurrency in programming languages, I wonder how LabView would be categorized in terms of its concurrency. LabView is essentially parallelized by default. Operations are executed as soon as their inputs are available, and you can have many concurrent data paths all execute at once. A visual programming approach makes it easier to think about concurrency, since you can see all of the datapaths at once. Also, on the subject of catching concurrency errors, languages like Rust can catch such errors at compile time. Rust allows data to be bound to only one identifier (aka no making copies of pointers) as a way of avoiding data races. This is called its ownership system; ownership of data can belong to only one binding at a time. Ultimately, this should help programmers feel more comfortable embracing concurrency knowing that the language itself will keep them safe from difficult-to-find concurrency errors.

Conclusions