The Elixir Programming Language

Erlang is a programming language [1] that was developed by Ericsson in the late 1980's, and open-sourced in 1998. Erlang was designed to support concurrency in the core language, in addition to other related features, such as fault-tolerance and distributed computing over many nodes. The main purpose of concurrency was initially to handle network centric tasks, such as routing and processing phone calls. Erlang uses shared-nothing message passing and tends toward the functional side-effect-free paradigm. In general Erlang follows the Actor model of concurrent computation.

Erlang is distributed as OTP (Open Telecom Platform), which includes both the language implementation and a set of libraries and tools that make use of Erlang's concurrency. For example, OTP includes libraries implementing a common protocol for communicating between nodes, a distributed database, and many standard language libraries for common datastructures.

Like Java, Erlang uses a bytecode virtual machine, normalizing the underlying hardware and OS implementations. Having a separate and well-defined bytecode layer allows for alternate languages to exist in the same runtime, and one such language, Elixir, was recently created to run on the Erlang VM.

Elixir aims to maintain all the benefits of Erlang while switching to a more consistent and simple set of language syntax and semantics. Relying heavily on macros, Elixir promotes metaprogramming as a way to simplify solution implementations. The syntax of Elixir is inspired by Ruby, though the semantics are quite different. For example, Elixir is not object oriented, instead relying on modules to group related operations. In one article the author said “Elixir is what would happen if Erlang, Clojure, and Ruby somehow had a baby and it wasn’t an accident.” [2]

Erlang doesn't appear to be a popular choice for scientific computing, though there are some references to the advantages offered by having concurrency built-in [3][4]. One interesting proof-of-concepts combines Erlang with CUDA, though the project has since been abandoned [5]. Elixir is able to call any Erlang library function with zero overhead, so it inherits these advantages. The additional advantages that Elixir brings might be enough to increase the Erlang VM usage in scientific and other parallel computation applications.

As seen with the popularity of (Google's) Go language, built-in concurrency constructs can allow everyday applications to juggle many events and thus take advantage of multiple CPU cores and multiple nodes [6]. Being able to naturally implement concurrent and parallel algorithms without having to deal directly with locking and other low-level mechanisms (for the most part) can increase the likeliness of these advanced algorithms being implemented at all.

Over the last year, Elixir has caught the attention of an increasing number of developers [7]. At two different developer conferences that I attended, one on Ruby and one on Javascript, I heard many people mention that they were experimenting with Elixir. It's difficult to tell how much this trend will continue, but as the number of cores increase it seems inevitable that developers will need to take advantage of these for some tasks.


