Kernighan, B. W. & Pike, R. (1984) *The Unix Programming Environment (Prentice-Hall Software Series)*. Prentice Hall, New York.

Kernighan, B. W. & Ritchie, D. M. (1988) *The C Programming Language*, 2nd ed. Prentice Hall PTR, New York.

Thompson, S. (1999) *Haskell: The Craft of Functional Programming*, 2nd ed. Addison Wesley, London.

Von Sydow, B. (2000) Review of Hudak (2000). *J. Funct. Prog*, **10**(5), 501–508.

Programming Erlang – Software for a Concurrent World by Joe Armstrong, Pragmatic Bookshelf, 2007, p. 536. ISBN-10: 193435600X.
doi:10.1017/S0956796809007163

Context

A funny thing happened on the microprocessors' race to faster and faster clock speeds; somewhere along the way, the speeds and feeds hit a brick wall and the only way out is lateral – more cores rather than faster CPUs. It started with hyper-threading and progressed to dual cores and to multi-core CPUs. While this paradigm shift solved the hardware challenges, a new bottleneck arose: in the software! Multi-threaded programming, which was mainly for leveraging I/O waits and SMPs could not scale to moderately massive parallelism; while the domain complexity of concurrent programming itself is interesting, the additional accidental complexity of threads and murexes and semaphores in traditional programming just does not cut it.

The answer it seems is 'Pluralitas non est ponenda sine neccesitate', i.e. the singularity of immutability and statelessness of functional languages rather than the pluralities (of mutexes, semaphes, spin locks and code locking): a manifestation of Ockham's Razor! As a result, more and more folks are looking towards functional programming as the silver bullet for providing the software version of Moore's Law! In fact a timely article in Dr. Dobb's Journal (Swaine 2008) wonders whether 'functional programming is on the verge of becoming a must-have skill'! Erlang is at the forefront of this revolution – an article in ACM Queue (Larson 2008) is of the opinion that '… designed for concurrency from the ground up, the Erlang language can be a valuable tool to help solve concurrent problems'.

In short Joe Armstrong's book *Programming Erlang – Software for a concurrent world* is at the right place and at the right time! As the title implies, the focus of the book is more on concurrency than functional programming but functional programming is the essential backdrop for the functionality required.

Overview

This is not a traditional programming book – rather I consider this to be a systems book, and that comes from the fact that it is difficult to separate 'Erlang the language' from 'Erlang the system'. The book follows the architecture of the Erlang system, which makes perfect sense, especially as the author is one of the originators of the language.

The book is pretty thick – 20 chapters and 5 appendices, coming out at over 500 pages. The subject is deep but the style makes it interesting and easy to comprehend. I felt that the book ended very fast! The book has a logical progression and is logically divided into four parts:

- Part 1 covers the important concepts of the language with enough syntax to get started, and ends with an overview of compiling and running Erlang Programs.

- Part 2 dives into bigger topics like concurrency and distributed programming plus files and sockets.
- Part 3 is where the Erlang system concepts are detailed including OTP and databases.
- The book ends with Part 4, which talks about multi-core programming.

On the whole the book is laid out well. I felt that this is not a book for the casual reader, both because of the depth of the subject as well as the breadth of understanding required to ‘grok’ Erlang. In my observation, Erlang is slightly difficult to get around for OO programmers. I have Cobol and Pascal background so was easier, still was corrupted by years of OO programming.

One should read the book fully to get the benefit – some of the concepts become very clear at the later chapters – an artefact of the language-as-a-system characteristic of Erlang. There are other Erlang books in the pipeline, and it will be interesting to see how the authors are handling this in those books.

Half way through the book, I found it helpful to read through the Erlang language reference – just to get a feel for the full syntax. Erlang is very small language, which is one of its beauties. Another must-read is the History Of Erlang paper (Armstrong 2007) as well as the presentations. The history of Erlang is fascinating and helps to broaden understanding while reading the book.

A few detailed observations from my reading of the book are as follows:

- The Bit syntax is explained very well. It could have helped me for a time sync protocol implementation (IEEE1588v2) with 48-byte manipulations.
- No surprise – the chapters on concurrency are the strongest.
- One area it does not cover is the cloud computing, in which Erlang very strong – for example AmazonDB is written in Erlang and the new cloud infrastructure project Verteba is also built using Erlang.
- The chapter on ets/dts key/value stores is timely, as that is very common in internet applications.

Conclusion

Overall, this is an exceptionally good book on a very relevant language/system. I would suggest prereading the couple of papers – the Erlang syntax as well as history of Erlang before reading this book. Then lots of things would make sense from the start of reading this book.

One interesting artefact of the immutability of data in Erlang is that handling traditional data structures like linked lists. A companion book that would be of interest to the readers of this book is ‘Purely functional data structures’. In addition to Erlang, the other FP languages under which have concurrency primitives include Scala, F# and Clojure. I look forward to reading the book ‘Programming Scala: Tackle Multi-Core Complexity on the Java Virtual Machine’ which follows the concurrency paradigm covered in the Erlang book. Scala is a mixture of OO and functional programming and has many of the Erlang patterns like gen_server. Clojure might be of interest to readers because of its origins in LISP.

There are concerns that functional concurrency – pure or part of an OO model – itself might not solve the massive concurrency problem; with programs running on more than 16 cores potentially hitting the memory wall. It is apt to quote the epilogue which says it all – ‘… large monolithic programs are becoming dinosaurs … the future has multiple cores and … distributed processors … .

References

Armstrong, Joe (2007) History of Erlang, *Proceedings of the third ACM SIGPLAN conference on History of programming languages*, ACM Press. http://www.cs.chalmers.se/Cs/Grundutb/Kurser/ppxt/HT2007/general/languages/armstrong-erlang_history.pdf

Larson, Jim (2008) *Erlang for Concurrent Programming*, ACM Queue, 1 September 2008. http://acmqueue.com/modules.php?name=Content&pa=showpage&pid=556 Last accessed 10 February 2009.

Stokes, Jon (2008) Analysis: more than 16 cores may well be pointless, *Ars Technica*, 7 December 2008. http://arstechnica.com/news.ars/post/20081207-analysis-more-than-16-cores-may-well-be-pointless.html Last accessed 10 February 2009.

Swaine, Michael (2008) It's Time to Get Good at Functional Programming - Is it Finally Functional Programming's Turn?, *Dr.Dobb's Journal*, 3 December 2008.

Krishna Sankar
Cisco Systems, USA