Scientific Programming 15 (2007) 189–190
IOS Press

Book Review

Writing Scientific Software – A Guide to Good Style, by Suely Oliveira & David Stewart, Cambridge University Press, 2006, ISBN 0-521-67595-2

As a computer scientist working in the world of "real" (physical) scientists, I often feel stuck in some sort of purgatory between million-line FORTRAN codes and the (currently in vogue) elegance of Python, Java or Ruby. It is refreshing to discover a book on coding style written by authors who obviously understand the application side of scientific software. Throughout the book, I must admit, I kept checking references and wanting to add footnotes to the content.

Suely Oliveira (CS department) and David Stewart (Department of Mathematics) from the University of Iowa have a deep understanding of many of the complexities underlying software in the scientific domain. This topic is incredibly deep, including the obvious scientific simulation, image manipulation and even video games. The authors in this text take a broad approach, yet delve in deep in several (mostly mathematical) areas.

The book is divided into five parts:

I) Numerical Software,
II) Developing Software,
III) Efficiency in Time, Efficiency in Memory,
IV) Tools and
V) Design Examples.

Each topic is a text in itself (typically in computer science), which can make the field rather daunting for people without a CS background. The book approaches the topic from the perspective of a scientist taking the next step into software. To take them down this path, the authors cover a broad range of topics yet with sufficient examples and references to do them justice.

Part I: Numerical Software, starts by describing issues around precision and limits introduced through computational methods. This short section reveals the bitter reality of numerical calculations and ways to minimize negative effects. The section ends with a few short examples of just how much damage these little gremlins of precision can be (famously disastrous, of course).

Part II: Developing Software, covers some of the basic computer science motherhood and apple pie. Topics include system architecture, design principles, data structure and designing for test. This is the largest section of the book, but appropriately so for an audience more likely to be familiar with test tubes or laser optics than MMU's, ALU's and the like. The level of coverage is just enough to whet the appetite of those interested in what the buzzwords are about. The computer scientist in me finds this broad topic to be of critical importance and I feel the need to add a pointer here to a wonderful educational resource available online called Software Carpentry (http://www.scipy.org disclaimer: I was associated with this work when it started in the mid-1990's).

The next section (Part III: Efficiency in Time, Efficiency in Memory) discusses issues relating to performance. I read this with mixed feelings as part of me wanted to scream out that optimization should be done as a latter step and/or when the application is under performing. Of course this isn't an absolute and it is important to be aware of algorithmic design issues and the "Big O" nomenclature (which, unfortunately just showed up in the text and was not appropriately introduced). The section ends with a touch on the ever-evil memory bug and things to be done to avoid facing it.

Part IV: Tools, is the topic I expect to be of most leverage for the reader. There is a wealth of scientific software available thanks to the research community within the US, and elsewhere in the world. The importance of these resources cannot be over emphasized as the reader is very likely to find either an existing solution to use or a close solution that can be modified (thanks to the amazing power of Open Source Software). Unfortunately this section is also the smallest of the book. I would have liked to have seen references to where to go to find existing software resources

ISSN 1058-9244/07/$17.00

(http://www.sourceforge.net is a good place to start, and of course http://www.Google.com).

The last section (Part V: Design Examples) is where the authors put some meat on the bone for the reader. Here they provide short examples, or tutorials on a few key design patterns. It is likely that the intended reader will be most familiar with the application space, and this section of the book would be most appealing. I read it with interest, as the techniques are in common use by those I work with.

The book itself is not without bugs. There are the occasional typo's and use of terminology without introducing it. My biggest complaint is that the authors did not address the issues of parallelism. As multi-core systems have completely overtaken, it seems an oversight to not address this topic. I strongly feel the authors should have spoken up and provided their opinions, advice and references to where interested readers could go to learn more.

Despite these minor complaints I do think the authors did a great job and should be commended for their efforts. Writing scientific software is an exciting and rewarding activity and also a great career for those trained (or training) in the sciences and interested in the topic. As is widely publicized, computational simulation is rapidly becoming accepted as a peer to theory and experiment in the quest for knowledge. Writing scientific software is key to enhancing our understanding of things. Guides to good style are a valuable resource to help us along that path.

Brent Gorda
Lawrence Livermore National Laboratory
Livermore, CA, USA