Book Reviews

MPI – The Complete Reference, Vol. 1, The MPI Core, 2nd ed., Scientific and Engineering Computation Series, by Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker and Jack Dongarra, MIT Press, Cambridge, MA, USA, 1998. ISBN: 0 262 69215 5

MPI – The Complete Reference, Vol. 2, The MPI Extensions, 2nd ed., William Gropp, Steven Huss-Lederman, Andrew Lumsdaine, Ewing Lusk, Bill Nitzberg, William Saphir and Marc Snir, Scientific and Engineering Computation Series, MIT Press, Cambridge, MA, USA, 1998. ISBN: 0 262 57123 4

MPI – The Complete Reference eponymously stakes its claim to being a complete reference. Volume 1 discusses "The MPI Core" (meaning MPI-1 but using MPI-2 names) and has 9 chapters, a bibliography, and 426 pages including the index; Volume 2 discusses "The MPI Extensions" (meaning features new with MPI-2) and has 9 chapters, a bibliography, and 344 pages including the index. The index in each volume includes entries in the other. Both volumes contain a constants index and a function index. The authors are members of the MPI Forum, other members of the Forum are mentioned in the Preface. The authors clearly state that these books are not a tutorial but rather than being a first book on using MPI, these books are intended as a resource for the experienced MPI user. And indeed, every function, including constants associated with use, is described. The bindings for C, C++, the original Fortran bindings and the modern Fortran bindings are included.

So why would one read these books? The message passing beginner has tutorials available. The experienced programmer wanting the definitive MPI specification may fetch it from the MPI Forum's web site (http://www.mpi-forum.org). Have the authors given the reader more than the definitive specification? If successful, the intended audience would be the programmer who will make a significant investment in software using MPI and who wants to make good use of the full extent of MPI in doing so. The Preface emphasizes that the book is not "a gentle introduction" nor a tutorial. And, yes, every nook and cranny of MPI is explored systematically. In fact, I found these books so complete that I found it difficult to review these books without, somehow, reviewing MPI as well. That indicates to me how well the authors succeeded.

So let's begin with the Preface of Volume 1, and see what's here. The starting point is a chronology of the MPI Forum, with a discussion of what was, and was not, included in MPI-1. A subset of Volume 1 suitable for a first read is described. Differences between the first edition and the second edition, and what's in MPI-2 (here that means, in Volume 2), are then sketched. The acknowledgments include some of the funding sources of the MPI Forum, and earlier efforts towards standardizing parallelism are acknowledged.

The Introduction discusses the conventions used throughout both volumes, as well as conventions adopted by MPI itself for naming procedures and constants. There is enough discussion of language (C/C++ and Fortran) binding issues to make the following chapters clear, but a much more extensive discussion of language-specific issues is presented in Chapter 8 near the end of Volume 2. Since message passing is, conceptually at least, a processes-to-process scheme, the basics involving processes, error handling and the interaction of MPI with other communication schemes (signals, Posix and other interprocess communications) are explained.

Chapter 2 discusses the basics of MPI communications. The basic transaction is the point-to-point (that is, process-to-process, node-to-node) copy of a contiguous block, called a buffer, of data. Not surprisingly, a number of basic issues arise immediately. The authors pay suitable attention to these basics: How much data is transferred (how large is the buffer)? Is the transfer between similar processor architectures or different processor architectures (representation format and endian)? Will the transferring processes wait (a blocking transfer) or not (a nonblocking transfer) during the transfer? And it is here that one is directly confronted by the very low level nature of the message passing paradigm: where any assembly language has a simple copy instruction (for example, “mov”) to transfer data, message passing generally requires two calls, one marking the transmission and one marking the reception, to transfer data (but we’ll see “one-sided transfers” in Volume 2).

It is here in Chapter 2 that the discussion turns briefly to what MPI means by type and how type matching is done between the sender and receiver. The issues of blocking versus nonblocking, and how to query the status and then cancel a nonblocking transfer are examined. One then moves to esoterica such as multiple completions, null requests and persistent requests.

Chapter 3 fully describes the MPI mechanisms for constructing multi-strided arrays, which MPI calls a datatype. These constructs allow “holes” between elements, and so they go beyond the contiguous blocks discussed in Chapter 2. The Fortran programmer can skim this quickly while the C and C++ programmer must learn how, at a very low level, to construct what a Fortran programmer can do with a few keystrokes of array reference. Nevertheless, the examples include Fortran cases to show the effects of array indexes starting at one rather than zero (of course, Fortran allows arbitrary lower bounds of array indexes including zero). The full create, commit, and use sequence is shown. The formulas describing the precise data locations are given, so one may check the exact locations referenced. It is complex to be sure but that is the nature of low-level detail. The rules for type matching are described. The various utility functions associated with datatypes are shown, as are conversion between C and Fortran array conventions (row-wise versus column-wise). MPI’s pack and unpack functions are described (again, Fortran has its own intrinsic pack and unpack).

Chapter 4 covers the collective communication capabilities of MPI. These include broadcasts and reductions. While these operations could be explicitly done by the user’s program, the higher-level functions are a welcome convenience. The rationale for the stricter matching rules of collective communications calls is explained and there is a brief introduction to the basic idea of the MPI communicator. The collective communications proper, beginning with barriers, are discussed. The distinctions among broadcast, scatter, gather, alltoall and allgather are made clear. Then the variants of the routines, for various data layouts, are discussed. The reduce operations are discussed, with the same thoroughness. Finally, a detailed discussion of the semantics of collective communications is presented.

Chapter 5 discusses MPI communicators, which the reader has been accepting without too much explanation until this point. Simply, communicators are named sub-groups of processors. The programmer may create them as needed. For example, the writer of a library for use by MPI programs may want a private group in order for the library procedures to communicate without treading on the caller’s namespace. A communicator provides a way to communicate within a group. Management of groups and communicators is fully discussed, as are intracommunicators (to communicate within a group) and, briefly, intercommunicators (to communicate between groups). Each process has a rank within each group to which it belongs. The heavy discussion of intercommunicators is left until later.

Chapter 6 discusses topologies. An MPI topology is a logical shape of the set of processors executing the MPI program. This logical layout is distinct from the supporting hardware, although the implementation may use the logical layout as a hint to the system when assigning processes to processors. A program might find a rectangular scheme, where each processor has north, east, south and west neighbors, for example, to be the most convenient. The MPI facilities for defining and grids and graphs are explained. Attention is given to the mapping between rank (within the communicator) and the coordinates (within the topology).

Chapter 7 discusses the MPI environment, including startup and shutdown issues. Both external (for example, command line) and internal (initialization and finalization procedures) are discussed. The standard does not require a command line startup means, but the mpiexec command is required if one is supported. Within the program, the required initialization and finalization procedures are described in detail, together with synchronization requirements for their use and the associated inquiry procedures. Version, environment, timing and error reporting facilities are also covered in this chapter.

Chapter 8 discusses MPI profiling. Each MPI entry point has associated with it an alternative entry point of the same name (in a different library) which enables profiling of MPI programs. Thus, any MPI program may be profiled easily. This profiling is standardized because it is unreasonable to expect suppliers of MPI profiling tools to have access to the source code for every different implementation of MPI (especially the vendor-specific highly optimized versions where the profiling tool may add the most value).

Chapter 9 discusses some miscellaneous issues, including the size of MPI (why is MPI so big?) and gives advice on writing portable MPI programs. The answer to the “why is MPI so big?” question, of course, reduces to a desire to include a wide range of functionality, while also providing ease of use in straightforward situations. As is usually the case, portability is ensured by writing standard-conforming programs. These books take care to inform the reader of the nuances of the MPI standard.

Moving on to Volume 2, Chapter 1 repeats some of the conventions stated in Chapter 1 of Volume 1. It also explains the motivation for increasing the functionality (after the "why is MPI so big" discussion) with the introduction of MPI-2. To remain competitive, MPI-2 acquired dynamic process management available with PVM, one-sided transfers available with Cray's shmem library, a parallel file system available with IBM and Intel systems and other experimental systems, bindings for C++ (rather than C-only) and modern (rather than archaic-only) Fortran, and definition of some ambiguities and provision of some conveniences sought by programmers. The plan of the second volume is set.

Chapter 2 discusses some miscellany, including MPI interoperating with threads, interlanguage interoperability, some status issues, and MPI's built-in memory management. The interaction of MPI and threads is especially interesting because while MPI may be used in an environment supporting multithreaded programs, the basic fact of MPI's process-based perspective remains. Thus, one approach for the programmer is to allow only one thread per process to call MPI at all. In fact, only one thread may make some MPI calls. With the growing popularity of multiprocessor nodes, this is a welcome discussion.

Chapter 3 discusses process management issues. The goal is to provide a process model which applies to a wide range of implementations while avoiding responsibilities of host operating systems. Determinism and MPI-1 compatibility are requirements. Thus, the targets include MPPs managed by a batch queuing system, networks of workstations with various management schemes, and even large SMPs. This work was written too early to discuss how MPI's process model will interact with grids (but see Chapter 6 below). Presumably, some accommodation will be made (perhaps with MPI-3?), at least when grids have stabilized at bit more than at present.

Chapter 4 discusses "one sided" communication. Cray's shmem library enables one-sided communication, adding similar capability to MPI widens the range of application which may be efficiently supported (and some programmers find one-sided to be simpler than matching sends and receives, even if synchronization issues remain). But how to support one-sided transfers across an arbitrary network with heterogeneous processors? The MPI solution is to define a *window* of memory and restrict the transfers to offsets within the window. While a bit cumbersome, the model appears to be able to be implemented on a wide variety of processors without too much operating system support. The model is fully explained here.

Chapter 5 discusses communication between processes designated by different communicators. Volume one discussed communication within one communicator (*intracommunicator* actions) while only briefly mentioning communication issues between communicators. Here, the discussion broadens to more thoroughly include actions among processes identified by different communicators (*intercommunicator* actions). These actions include broadcasts, reductions, scans and barriers.

Chapter 6 discusses ways of extending MPI, including debugger and profiling support. No specification can anticipate the ways in which users and vendors will want to extend it, the MPI Forum has the foresight to provide hooks to provide standard means of extension of the MPI specification. These include access to the value of otherwise opaque constants and other information internal to the state of the MPI system running on each processor within an MPI program.

Chapter 7 discusses input/output operations from within MPI programs. While MPI does not require that language input/output be available on all processes within an MPI program, it probably is. So what does MPI mean by *parallel I/O*? It refers to input/output operations performed in parallel by several processes on the same file. This requires some definitions, if only to cover all languages in use (C/C++/Fortran). The most interesting feature, to me, is the info object which is helpful when trying to optimize the input/output operations. It consists of hints-only specifiers which may be ignored if not supported. Thus, the hints may not change the semantics of the input/output operations and may be extended if a vendor has particular needs unanticipated by the MPI specification.

Chapter 8 discusses language binding issues in detail, including the C++ binding, and archaic Fortran (Fortran 66/77) and modern Fortran (Fortran 90/95/2003) bindings. Unfortunately, the C++ binding came prior to the C++ 98 standard and does not take advantage of templates (which were brought into the C++ standardization process very late before release of the C++ 98 standard). However, the binding, the issues faced and decisions made are explained clearly. An effort to take full advantage of modern Fortran features seems not to have been made which is harder to understand. Fortran 95 was the reigning standard when MPI-2 appeared, Fortran 90 having been available for several years (today, Fortran 2003 is the current standard, compilers are now implementing the new features of Fortran 2003 steadily). A binding for modern Fortran which takes better advantage of its modern features would make these features, and the optimizations they enable, available to message passing programmers. But again, the issues faced and decisions made are explained clearly.

Chapter 9 is a summary. It contains a few notes and comments. The quotes about this generation being the first and last generation of message passing programmers may or may not be true, given the longevity of the client-server paradigm and the coming of grid computing. (On the other hand, Fortran seems poised to include co-arrays in the just-now-commencing Fortran 2008 standard.)

I am glad to write that the authors successfully gloss the MPI specification, providing insights into every detail of MPI. Aside from the sheer completeness of these books taken together, I found their greatest assets are the discussions contained in the *rationales, advice to users, and advice to implementers*. These paragraphs provide explanatory context for the dry, succinct prose, the necessarily cover-all-cases mode of expression of the specification itself. For example, why does MPI not allow a process to read from a buffer once it has been set for transmission? That shouldn't hurt, should it? The buffer may be unmapped from the virtual memory of the transmitting process, in order to be available to a DMA port or other independent I/O processor. Even an experienced message passing programmer may not have thought of that issue, at least if not present on the platforms in use. I found insights such as this to be helpful when trying to understand message passing as implemented by MPI.

So message passing has moved to MPI-2 as the most popular way to go, with various other schemes being left progressively further behind. After so much time, it's amazing, at least to me, that no new paradigm beyond message passing has become widespread for massively parallel programming. Yet I recently read [1] how message passing is considered fundamental both to physics and to computer science. Perhaps message passing is here to stay for longer than otherwise imagined, perhaps a higher level of expression will be found. But the basic efficiency of message passing indicates that even that future higher level paradigm will likely be implemented via message passing, at least where the network is distinct from the processors. And until that higher level of expression is popular, MPI appears to have cornered the market as the message passing library of choice. With documentation of the high quality of these books available, MPI can more easily continue in that role. I will certainly keep both volumes of *MPI – The Complete Reference* handy by my workstation.

Dan Nagle
Purple Sage Software Inc.
USA

Reference

[1] M. Mézard, Passing Messages Between Disciplines, *Science* **301** (2003), 1685–1686.

Parallel I/O for High Performance Computing by John May, Morgan Kaufmann Publishers, San Francisco, CA, USA. 2000. ISBN: 1-55860664-5

There is much to like about John May's book, Parallel I/O for High Performance Computing. First of all, it is a book about that most neglected area of computer life, I/O, especially I/O to storage devices.

Then there is its exceptionally broad coverage of the subject. After the introduction, the book starts with a chapter on the low level of storage devices and methods used to interconnect them. Then, it marches, chapter by chapter, up the abstraction hierarchy. Chapter three is devoted to file systems, mostly based on Unix like semantics. Section four of this chapter contains the real justification for the book. It explains in clear and concise language, why the "traditional" mechanisms used for I/O in more typical systems aren't sufficient for high-performance scientific workloads.

Chapter four is a slight, but appropriate, detour from the hierarchy and discusses the various access patterns used by high-performance applications and ways to optimize their I/O performance. Then we go back to ascending the hierarchy with chapter five on low-level software I/O interfaces such as HPF and MPI-I/O. Next up the hierarchy is chapter six, on scientific data libraries such as NetCDF and HDF. Then one more detour, this time into special purpose I/O techniques such as out of memory computations (what to do when your intermediate results don't fit into available memory), and I/O for checkpoints. The eighth and final chapter, at the highest abstraction level, focuses on metadata (both directory data and other information such as file descriptions and formatting details), database management systems, and a bit on knowledge discovery in very large data files.

Given the wide range of topics in a relatively short book (305 pages of main text), the coverage of each topic is necessarily shallow. Compensating for this are suggestions for further reading at the end of each chapter, as well as a 16 page glossary and a 174 entry annotated bibliography.

Another likable aspect of this book is the writing style, which is clear and direct. The author adds information that compares alternative approaches, giving the advantages and disadvantages of each, and indicating which ones are primarily of historical interest (at least at the time of writing, see below). This makes the book much more than what it could have been; a mere boring compendium of software capabilities and specifications.

Inevitably, there are a few things that could be improved. The necessarily brief coverage of each topic can lead to misleading impressions. In addition, most of the material is aimed at distributed memory computer systems, such as clusters. While this does highlight the areas where parallel I/O is most difficult, it results in giving short shrift to the problems encountered with I/O for high performance computing even on shared memory systems. There are a few factual errors, but these aren't significant and could be corrected with the deeper study required from someone really working in this area. The chapter on scientific data libraries contains several five page long code listings that are more tedious than enlightening, but even here, the key comments are highlighted in a bold font, giving the reader the opportunity to get the gist of what is going on without reading the whole listing.

One other issue that is due to its wide coverage, is that the book is more exposed to change than most other books as a result from ongoing technological developments. For example, a book on file systems design would have to keep up with developments only in that field but this book must keep up with at least the major developments in the whole range of fields that it covers. The book's age (copyright 2000) shows up quite clearly in some technology areas. Examples of newer technologies not covered in the book include serial interconnects for disk drives and the growth of switched fibre channel storage area networks (indeed it doesn't mention fibre channel wire speeds faster than one gigabit per second.). Problems can also show up in other ways. For example, the discussion of networked attached storage focuses on the Network Attached Secure Disks (NASD) work. However, this work has, so far, not resulted in successful products. The flip side is that the book ignores the network attached storage products of such companies as Network Appliances and EMC as stand alone file servers. All of this means that readers would benefit substantially from an updated second edition that would include new developments and perhaps drops some of the topics that have gone out of favor.

There are at least two potential audiences for this book. One is people coming to high performance computing from a subject matter specific background such as physics or meteorology. These people will benefit from the discussions of the basics of I/O as well as from exposure to the I/O related issues in their new field. The second is people who have a background in computer science and want to broaden their horizon by learning more about this specialized area. People in this group could probably skip the first 70 or so pages as the material covered there would be a review of material covered in basic computer architecture or OS classes. However, they will benefit immensely from the rest of the book, as that material is generally not covered anywhere else.

Overall, this is a very useful introduction to a field that has far too few books like it. The style is easy to follow and it imparts a basic understanding of the key issues, without becoming boring or too deeply technical. A reader certainly won't learn all he needs to know about the field from this book, but any serious user of any of the techniques will have the references he needs to find out more.

Stephen Fuld
USA

Linux for Non-Geeks, Rickford Grant (Book + 2 CDs), No Starch Press, San Francisco, USA, March 2004. ISBN: 1 59327 034 8

The title is amusing, but the subtitle of Rickford Grant's book is more informative: "A hands-on, project-based, take-it-slow guidebook". As such, it includes everything that you will need to experiment with Linux.

As the title says, this is not a book for those who know and love Linux already (that is to say, geeks). It is heavily oriented towards those of us who come from

Windows and Mac desktops, and want to know what all the fuss being made over Linux is about.

Firstly, in case anyone hasn't been paying attention recently, Linux (or GNU/Linux as it should more properly be known) is a family of generally free computer operating systems. I say free because, since the kernel, essential support structures, development tools and productivity sets are nearly all free of restrictive copyright. I say family because, since the components are nearly all free, a wide variety of people and organisations have developed Linux distributions that reflect particular ideas of how a system should function, how it should be maintained, and how much flexibility, utility, or security it should offer to the user. There are also different distributions that focus on providing systems specialised for use as servers, routers, graphics servers, database servers, multimedia desktops, office desktops, or even network clients. A growing number of these different distributions are available in the form of live-CDs. The way these work is that you boot your computer from the CD, and instead of going through the process of installing the system to your hard disk and configuring everything step by step, the system loads into memory and auto-detects all hardware on-the-fly.

Despite all the differences listed, Linux distributions are more similar than they are different, so that expertise gained with this book's system and projects will transfer well to other Linux systems. The particular flavour of Linux chosen for distribution with Linux for Non-Geeks is Fedora Core, which is a product of Red Hat, the biggest company in the global Linux market.

Linux for Non-Geeks consists of a sequence of projects that will take you through the process of understanding what Linux is, how to install it using the CDs included with the book, how to set it up for your preferences, and then how to learn to do everything that you might reasonably want do using a desktop computer. Anyone who comes to this book with experience of using Windows or Macintosh desktop systems will have no trouble following the projects, but will find many interesting new differences from what they are already accustomed to.

To be more thorough in my description, the material covered in the course of the book includes:

– Understanding the differences between Linux and other systems, and the advantages of Fedora Core Linux (the flavour of Linux provided on the enclosed discs).
– Installing and configuring Fedora Core, user accounts, and the Gnome desktop manager.
– Setting up hardware such as printers, cameras, scanners and network interfaces.
– Managing and updating the system using the provided system management utilities.
– Setting up Internet applications including browser, email, FTP and IRC programs.
– Accessing the vast repositories of Linux software available on the Internet.
– Managing data and archiving it using Linux tools and CD burning packages.
– Using the included office suite (OpenOffice) to best advantage.
– Loading and running multimedia including DVD movies and music on CDs, on file as MP3s, or via streaming media form the Internet.
– Coping with multiple languages, which is a particular strength area for Linux.
– Installing and managing extra fonts for both printing and onscreen use.
– Exploring the power of the command line.

At this level, the only areas where the author might usefully have included some extra coverage are in understanding Linux shell scripts and in selecting and experimenting with the vast choice of available development tools, but it has be allowed that this is, at most, a minor quibble. One area that is thoroughly glossed over in the book is the subject of device drivers and how one makes devices work with Linux. The is some justification for choosing to skip lightly over this subject, since the issues that arise are highly technical, more suitable for genuine Linux geeks than the designated audience of non-geeks. Another justification is that Linux device support is remarkably good for most standard devices, and the drivers are loaded automatically and silently so that things work correctly. Coping with the situation when a device fails to work is somewhat more technically detailed than is appropriate for the level of the book, although I felt that the author provided good enough links to further resources that any non-geeks who wanted to become geeks could do so by starting with the recommended reading given at the end of the book.

While the style of the book is informal and gently humorous (but not annoyingly so) the pedagogical approach of the author is rigorous and thorough. Although it may seem so upon casually flipping through the book, this is not a basic recipe book that tells how to use particular pieces of software. No indeed, this is a book that teaches the kind of questions to ask and how to answer them. This is based upon the best principles of Linux (and Unix systems in general), one of which is that there is generally more than one right way to do things. That, by the way, is one of the appeals of Linux to the scientific mind, that flexible thinking is possible, and may very well explain why Linux feels so refreshing after being stuck with the other desktop alternatives (no names mentioned!) for too long.

An important point worth mentioning is that using Linux for Non-Geeks as a way to experience and learn about Linux does not require you to scrape your computer clean of Windows, if you already have it and aren't sure about being ready to commit fully to Linux yet. Rickford Grant describes quite clearly in the book how to go about making Linux share your computer with Windows. Being a curious and dedicated reviewer, I adopted the harder path of setting up a dual-booting Windows/Linux system according to the instructions given in the chapter on installation, appropriately called Making Commitments.

I came out of that chapter with a choice each boot time of starting up Linux or Windows, and that's the way it still is, but the Windows boot option is hardly ever used these days. Besides, Linux provides transparent access to all the data I have stored under Windows, together with programs for dealing with most of that data. I can handle Microsoft Office documents using OpenOffice, which is installed by default with Fedora Core, I can play stored music and watch stored movies with a variety of supplied programs, I can develop and compile code written in Fortran, C, C++, Java, Perl, Tcl/Tk and many others, and support for doing so is there straight "out of the box" as it were.

What if you want or need a program that isn't set up when you install your Fedora Core system? Well, the book has the answer to that as well. Chapter 10 is a step-by-step approach to making your system indefinitely upgradable by use of the automated package management tools. You open the package management program (Synaptic is the one chosen for use in the book), search by keyword or name for the program you need, and click to mark it for installation when you find it. When you've found all the packages you need, click the install button and the packages are downloaded, installed, and ready to run. Since this is Linux, you don't even need to reboot before you can run them.

This review was entirely written using OpenOffice Writer running on the Fedora Core Linux system provided with the book. It is only fair to point out that the review would have been finished sooner if I hadn't followed the instructions given in the book for installing the hopelessly addictive Frozen-Bubble.

As an introduction to Linux and an explanation of what all the present fuss is about, Linux for Non-Geeks provides the gentlest possible entry to the world of Linux, provides enough handholding to get the reader through the initial mysteries of a new operating system environment, convinces the reader that there is still more worth exploring, and leaves them with enough experience to go solo. As you can probably tell by now, I give the book a strong commendation and the author a gold star.

Mark Madsen
Geneva, Switzerland

About the Reviewer

Mark Madsen is a consultant specialising in Internet issues. His published research spans cosmology, mathematics, biology, philosophy, and computer science. His current role is as Principal of his consulting organisation Manige Solutions (www.manige.co.uk). Mark presently divides his time between Cambridge and Geneva.