Book reviews

Industrial Strength Parallel Computing: Programming Massively Parallel Processors, by Alice E. Koniges, Academic Press / Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999. ISBN: 1558605401

Industrial Strength Parallel Computing is an industrial strength book. It has 597 pages, with a Preface, 25 Chapters, an Appendix, a Glossary, an Index and a Contributors section with a paragraph about each contributor. 17 of the 25 chapters are case studies of making numerical applications run on large scale parallel computers. Each chapter has its own bibliography. The work reported here is the result of the Parallel Applications Technology Project, supported by Los Alamos National Laboratory, Lawrence Livermore National Laboratory, and Cray Research, Inc. Who should read this book? Anyone who plans to port a program from a sequential machine to a large scale parallel processor will benefit from the experiences described here. Also, anyone who wants to quickly survey the range of work done on parallel machines will find this book an authoritative source. The writing is consistently good technical writing, the editors and the authors have done a good job of presentation.

So what does 'Industrial Strength' mean? One has an image, I suppose, of a large factory with many busy sections, with railroads supplying resources and removing products. In the case of computers then, industrial strength describes the largest computers available, with modern networks playing the role of the railroads. The authors clearly mean industrial strength in the sense of very large capacity, but also, I think, in the sense of cost-effective, dependable, and known to be workable. In the end, a meaning of routinely available, a normal way of production, a choice easily made, is also intended. So the thesis to be demonstrated is that large-scale parallel computing is ready for everyday use, as any other segment of computing is used everyday. The authors pitch the benefits to be gained as if trying to attract capital investment for a proposed, perhaps novel, factory.

The first six chapters are a review of the state-of-the-art, starting with Flynn's Taxonomy. Then SMP, DMP and ccNUMA systems are described. The authors settle on the term MPP to describe the parallel hardware being used (largely, but not exclusively, Cray T3D and T3E systems). The issues of software methods, message passing, Cray's shmem library, HPF, threads, and various mixed models, are discussed. Software tools, such as Cray's Apprentice tool, and debuggers, are then described. Since single processor performance is an important scale factor in parallel computation, a chapter describes techniques for achieving high performance with a single processor. The last of the first six chapters contains a discussion of parallel scheduling issues, necessary to integrate parallel computing with the rest of computing, especially in an age of networks where the MPP is just another server. Gang scheduling is described as the preferred scheduling strategy. The authors' use of the term MPP avoids getting stuck splitting hairs among ccNUMA, DMP with single processor nodes, DMP with multiprocessor nodes, and any other architecture one may imagine. The authors settle on the "scaled problem" (that is, twice the problem size is compared when twice the number of processors are used) as the appropriate measure of parallel speed-up.

The heart of the book is the 17 case studies. Topics range from environmental studies, to petroleum reservoir management, fluid dynamics, plasma studies, solid state physics simulations, radar reflectivity simulations, nuclear magnetic resonance analysis, molecular dynamics, genetic studies, and image processing. So a wide range topics is discussed, and therefore a wide range of mathematical methods are employed. A helpful table inside the front cover and repeated elsewhere coordinates model, methods, tools and techniques. Most chapters begin with a description of the mathematical background, and how the mathematics leads to the computational methods used, a few chapters leave the mathematics to the references. Then the implementation is described, as are the results achieved. Attention is paid to the scaling properties of the program. The descriptions are detailed enough to allow a worker to benefit when trying to decide how to parallelize an application in the same or in another field.

I'll choose two case histories to illustrate the 17 presented in the book. My first choice is *Ocean Modeling and Visualization,* Chapter 7. The code was ported from sequential processors to the parallel processor using the Apprentice tool. The authors used compiler options and optimized libraries to optimize the code for a single processor of the parallel processor. They replaced if and where constructs with masked operations (an addend added conditionally is instead always added after being multiplied by 0 or 1 conditionally) to keep the processor's pipelines from being disrupted by jumps. The authors note that vector-vector (BLAS1) routines, which their program currently uses, could be replaced with matrix-vector or matrix-matrix (BLAS2 or BLAS3) routines to give better cache re-use and a higher computational rate per processor. Nevertheless, the per processor speed on the scaled problem is nearly constant up to 256 processors, to a rate of 3.55 Gflops. The authors also discuss the tools used to visualize the results. There are 21 entries in the bibliography for this chapter. This is "Industrial Strength" parallel processing in the sense of being large-scale problem solving, and very necessary for modern society.

The second case history I'll choose is Functional Magnetic Resonance Imaging Dataset Analysis, Chapter 21. MRI is used to image the head and brain. Slice after slice is taken, a spiral really. Each slice must be rendered independently. Then, since the head in question is likely to have moved slightly while the scan is taken, the rendered slices must be re-aligned. Then the slices can be assembled into a three dimensional image of the brain. This is previously done by a sequence of programs running on high-end workstations. The computation required several hours. Thus, the patient had left the building before the images were available. The programs were run via a shell script as a sequence of JCL. These calculations were moved to an MPP. There was an attempt to demonstrate this during the Supercomputing '96 conference. The use of the MPP allows the results to be available in minutes rather than hours. Flexible scheduling of the MPP is required, a patient is never ready at the exact second originally scheduled, and the patient can't be held for too long afterward. The rapid turn-around provided by the MPP allows feedback, concentration on a region of interest, or other adjustments to be made while the patient is still in the scanning device. This is an improvement of quality, not merely computational rate or other quantity. Furthermore, given modern networks, the patient may be, say, in Los Angeles, the MPP may be in Pittsburgh (where it actually was for this work), and the specialist may be in Boston, who can then communicate with the attending physicians in Los Angeles, completing the feedback circuit. There are 21 references in the chapter bibliography. Beyond considerations of scale, I believe this is what the authors mean by "Industrial Strength" parallel computing. It's a valuable part of (what ought to be) the routine way of doing business. The benefits to patients are obvious.

The last two chapters are a summary of the lessons learned from the case studies. Some of the parallelization efforts ported an existing code, others started from the equations making a new application especially for the MPP. Finally, the authors make some predictions regarding the future of parallel computing. The penultimate chapter has the only sentence with which I really take issue. The sentence is the question "Should the basic language of the code ... Fortran 77 be updated to ... Fortran 90/95 ... ?". I must complain. Any Fortran 77 program is a valid Fortran 95 program. The language lawyer who objects may be reminded that compilers are not refusing to recognize the very few features actually deleted from the standard. The question is, rather, which features of Fortran 95 (or really, these days, Fortran 2000) should be used. I can think of two categories of features. One is features which provide similar functionality to that provided by a Fortran 77 feature but are simply better. Free format over fixed format, or the kind mechanism over "*n" notation, or modules over common blocks, are in this category, certainly. The second set of features are those features which standardize abilities provided with many compilers, but each with a different spelling. Memory management is an example. Various spellings of "malloc", or "heap_alloc", or whatever, are supported as extensions by various Fortran 77 compilers. Surely, the Fortran 95 standard spelling of "allocate" is to be preferred. In the ultimate chapter, the authors should be commended for daring to make predictions. Split/C, UPC, and Co-Array Fortran are improvements over message passing, perhaps mention here will encourage their wider availability. Wider availability, of course, will in turn, encourage greater use.

The sole appendix shows one way of combining message passing with threading, which will be useful on distributed processors where each node is a shared memory system. MPI is combined with pthreads to do so. The example code is written in C, rather than C++ or Fortran (either of which could have used encapsulation and overloading), so all the gory details are visible. The authors do mention that they have a Fortran binding for pthreads, but it's not part of the standard. Today, Hanson et al. have a portable Fortran binding (ACM-TOMS, 9/2002, #821) to pthreads, and Fortran 2000 has its *Interoperability with C* features, so there are more possibilities. But the "Fortran can't use pthreads" canard needs no emphasizing, and I would have appreciated different views of the multifaceted topic of threading while passing messages. An example where OpenMP provides the threading would have helped here, and perhaps would have been less complex. Maybe I'm complaining too much, this is a topic easily made very complicated, and a simple example is not unwelcome.

Is the author's thesis demonstrated? Overall, I think so. Certainly, the case studies are all successes. Is parallel computing all that difficult? Well, perhaps it is. Obviously, if one must decide between porting an existing, working code, on the one hand, and starting from scratch, on the other, there's considerable effort involved. But this book shows a wide variety of work, on a wide variety of ranges of length scales, being parallelized successfully. These applications scale up to several tens or to several hundreds of processors. Most of the authors of the case studies describe future efforts which will benefit their applications further. On the other hand, probably because it's still the only widely available scheme that scales satisfactorily, we're still passing messages (after all these years!). So to this extent, at least, the glass is half full or half empty.

Perhaps "industrial" use will spur the drive for a better paradigm. A wider range of applications will surely motivate the search for a more general paradigm than, say, HPF, or something easier to use than message passing. Maybe that more general scheme is Co-Array Fortran. But what the authors do clearly demonstrate that what was once "bleeding edge" experimentation is now merely leading edge work, with a well established body of practice available for guidance. That alone makes this book a valuable read, and I certainly enjoyed, and was educated by, reading it. I am convinced that large scale parallel computing is ready to be a part of the usual way of doing business wherever it's needed. And there are needs for it. These days, that means being a compute server on a network. That makes parallel computing "Industrial Strength" by any definition.

Dan Nagel
Purple Sage Computing Solutions Inc.
USA

Java Number Cruncher: The Java Programmer's Guide to Numerical Computing, by Ronald Mak, Prentice-Hill PTR, Upper Saddle River, NJ, USA, 2003. ISBN: 0130460419.

Java for numerical computing? Are you kidding!? Until recently, such a combination would indeed draw well-deserved howls of laughter from serious practitioners of numerical computing. The huge overhead of Java's traditional interpreted-code implementation has resulted in run times that have been typically at least an order of magnitude higher than traditional languages such as Fortran-90 or C/C++. As a result, Java has been a non-starter for large, numerically intensive scientific computation.

But times are changing. Partly because of the extensive interest in Java programming in the business and Internet world, "just-in-time" compilers and the like have been developed that result in much faster execution times, nearly competitive with C and Fortran code in many cases. As a result, computational scientists are now seriously looking at potential uses of Java for scientific computing. But who is going to write scientific Java? Perhaps some numerical scientists will become proficient in Java. But maybe we should also try to teach Java programmers the basics of numerical computing. Indeed, given the increasing scarcity of computer science graduates who are also trained in numerical analysis, perhaps this is a more realistic route to take.

Ronald Mak's book is very timely in this regard. It is targeted directly at beginning (or even not-so-beginning) Java programmers who would like to become familiar with numerical computing. It does not pretend to be an traditional course in numerical analysis, which quite frankly many present-day computer science undergraduates avoid like the plague. Rather, it teaches Java programmers what they need to know to be numerically literate, so as to be equipped to take on serious technical computing tasks when needed.

The book starts out by describing in detail the IEEE-754 floating-point standard, both single and double formats. The author first drives home the point that IEEE arithmetic is *not* the same as the real number system – for example, there is potential for significant loss of accuracy when two nearby floating-point values are subtracted. The book continues with topics such as the potential for difficulties when a large number of floating-point values of different sizes are summed, finding roots of equations using basic iterative techniques such as Newton's iteration, finding interpolating and data-fitting polynomials, and linear regression.

Chapter 7, for example, discusses the trapezoidal rule and Simpson's rule for integration. It also presents techniques for numerical solutions to differential equations, including Euler's method and the Runge-Kutta scheme. Again, the restraint that the author exercises here is remarkable. I'm sure that most numerical analysts writing such a book could not resist the temptation to include here a furtive write-up of their favorite advanced techniques, say for numerical quadrature. Space could also have been devoted to detailed discussion of techniques for numerical solutions of 2-D and 3-D partial differential equations. Instead, Mak continues to stick to his formula of providing a very detailed and readable account of basic numerical methods.

Beginning in Chapter 9, the author discusses matrix computations. Here, as in some previous chapters, the author provides a Java software package, which in this case is for matrix operations. With this facility, the author can focus on the concepts of matrix computation rather than on the detailed mechanics of carrying out such computations. Issues such as matrix condition numbers are discussed in Chapter 11. Even here, the focus is on concepts rather than on theorems, proofs or advanced implementation techniques.

Part IV (beginning with Chapter 12), entitled "The Joys of Computation", starts out by saying "Numerical computation isn't all work and no play". In these chapters the author gives several examples of computations using a "BigNumber" and a "BigDecimal" package, which perform high-precision integer and floating-point computation, respectively. Some of us would choose to differ with Mak's characterization of this material as "play", since in recent years numerous important mathematical and scientific results have been obtained using such high-precision computations. Besides, much of our Internet commerce relies on security schemes based on high-precision arithmetic. In any event, this material is actually quite well written. The author covers topics such as large integer factorizations, computing mathematical constants and functions to high precision and fractals.

In summary, this book would make an excellent undergraduate course in numerical computing, suitable for a wide range of students in computer science, physical science and engineering. It is also very well suited to professional Java programmers who would like to become more familiar with the world of scientific computing. It has a practical, down-to-earth approach that avoids exotic material, opting instead for a thorough and understandable coverage of basic material. It includes (and in fact relies on) software available from a website.

Many students or other readers, after completing Mak's book, will continue their careers with a greater appreciation of the issues and techniques of numerical computing, although perhaps they will not specialize in this arena. Others may find the topic sufficiently engaging that they will pursue more serious coursework and career paths in scientific computing. Either way, the computational science community will be enriched as a result.

David H. Bailey
Lawrence Berkeley National Laboratory
Berkeley
CA 94720
USA

Four Colors Suffice: How the Map Problem Was Solved by Robin Wilson, Princeton University Press, USA, 2002. ISBN: 0691115338.

Robin Wilson's *Four Colors Suffice: How the Map Problem Was Solved* is a popularization of the history and proof of the four-color theorem. A coloring of a map is an association of a color to each "country" or "region" of that map, so that bordering countries have different colors. In 1852, Francis Guthrie first voiced the hypothesis that four-colors are enough to color any planar graph. (Colloquially, this can be understood as the number of different colors of ink a mapmaker might need to keep in stock.) Trying to prove the truth of this conjecture was a Siren's song for many distinguished mathematicians for the next century and a quarter.

A bit of formalization clarifies the problem. Guthrie must have been concerned with countries that share not just a common point but a border segment. Otherwise, a pie-wedge map would require arbitrary many colors. Similarly, countries must be assumed to be contiguous. Allowing disconnected empires, like pre-Bangladesh Pakistan, can require an unlimited mapmaker's palette. We are, of course, concerned with maps on a plane or a sphere – donuts and more complex topological spaces require more colors.

Wilson provides a facile ménage of history and mathematics. He demonstrates the projection equivalence of polyhedra and planar maps, and explains how Euler's formula (the number of regions, including the exterior region, plus the number of vertices is two more than the number of edges, or $R + V = E + 2$) can be derived.

This leads to a proof that every planar map must have at least one country with five or fewer neighbors. Focusing on cubic maps (ones where exactly three edges meet at each vertex, because for coloring, these are the hardest maps), he presents the critical counting theorem: $4C_2 + 3C_3 + 2C_4 + C_5 - 0C_6 - 1C_7 - 2C_8 - 3C_9 - \ldots = 12$, where C_k is the number of countries with k sides. This formula has some surprising ramifications, including the fact that a cubic map with no regions with four or fewer neighbors must have at least twelve pentagons, a result most familiar in the twelve pentagons of a soccer ball.

Guthrie passed his hypothesis to his brother Frederick Guthrie and from there to Frederick's professor, Augustus De Morgan (perhaps most famous for De Morgan's laws) and on to the algebraist Arthur Cayley. Cayley introduced the idea of the *minimal criminal*: the smallest map needing five colors.

In general, proofs of the four-color theorem were *reductio ad absurdum* with respect to the minimal criminal: one shows certain properties of any minimal criminal and then shows that the four-coloring of some smaller map (as the criminal is minimal and all smaller maps are four-colorable) enables the four-coloring of the minimal criminal. One can easily show that the minimal criminal cannot contain a digon (two-sided region), or triangle, because such a region can be removed, the reduced map four-colored, and then removed digon or triangle reinstated without requiring an additional color.

Cayley couldn't prove the theorem but presented it to the London Mathematical Society in 1878. A year later, Alfred Bray Kempe, a member of the society, published a proof of the theorem. Kempe's proof introduced the important notion of "unavoidable configurations." For example, the counting theorem shows that every minimal criminal must be in at least one of four unavoidable configurations: having as its simplest polygon a digon, a triangle, a quadrilateral or a pentagon. Kempe showed that each of these configurations could be colored with four colors. The digon and triangle are trivial. For the quadrilateral, he introduced the *Kempe* chain: a subgraph of alternating opposite side colors between one pair of opposite sides. Using the Kempe chain allows recoloring an enclosed subgraph to free a color for the quadrilateral. He applied the same argument twice to show how to color maps whose smallest region is a pentagon.

Kempe's proof was widely accepted, at least for eleven years, until Percy Heawood inconveniently presented a graph for which Kempe's algorithm failed. The problem with applying the Kempe-chain method twice was that the second application does not necessarily preserve the conditions that enabled the first. However, Kempe's notion of dividing the space of maps into sets of unavoidable configurations was fundamental to the subsequent work on the problem.

Kempe also introduced the notion of *reducible configuration*: an arrangement of countries that cannot occur in a minimal criminal. As the above arguments show, a triangle is a reducible configuration. The focus of the subsequent work on the four color theorem was to find an unavoidable set of reducible configurations – that is, a set of subgraphs such that every cubic graph was in one of the sets but that every such set contained a reducible configuration. Hence, none of these sets could contain a minimal criminal.

Wilson's next few chapters are devoted to discussing the incremental progress on the theorem, small results that contributed to the ultimate proof but signaled the frustration of those who worked on the problem along the way. The four-color theorem is simply stated, easily understood, and the proof techniques can be understood by a bright high-school student. It was a continual source of distress to mathematicians in the first three quarters of the twentieth century that the actual proof was so elusive. Critical steps in this progression included the notions of discharging and rings. *Discharging* starts by assigning each country a number called a "charge." For example, we might give a country with k sides a charge of $6 - k$. By the counting theorem, the total charge of the graph is thus 12. We now progress to discharging: moving the charges around the map, preserving the total charge. For example, we could move 1/5 of the charge from each pentagon to its neighbors. Using this tehnique, Wilson explains Paul Wernicke's result that a digon, a triangle, a quadrilateral, two touching pentagons and a pentagon touching a hexagon are an unavoidable set. Eventually, unavoidable sets with thousands of elements were constructed. If it could only be shown that each element of such a set was reducible, the theorem would be proven, but for obvious reasons, mid-twentieth century mathematicians didn't traffic in proofs with thousands of cases.

Wilson also presents the key results of George Birkhoff, who generalized Kempe's work on chains to the idea of *rings* of countries that divide the map in two. By considering every possible coloring of the ring, it is often possible to show that the countries within the ring can be four-colored. Since the ring and countries outside the ring are smaller than a presumed minimal criminal, such a graph cannot be a minimal criminal.

Processing power increased. With the help of the high-speed computers of the mid-1970s, checking many cases became conceivable. Kenneth Appel and Wolfgang Haken were the first to complete the proof, relying on showing that a set of 1,936 graphs represented an unavoidable set of reducible configurations. They eventually reduced this to a set of 1,405. Their proof, as published, had 100 pages of summary, 100 pages of detail, 700 additional pages of back-up and required 1000 hours of computer time on the fastest computers. (Today, running their proof might take an hour on a fast personal computer.)

There is no evidence the four-color theorem has ever had any implications for actual mapmaking, but the century and a quarter of work on proving the theorem were foundational in the development of graph theory.

Wilson's book supports several different readings. On one hand, even a non-mathematician can come away with proof techniques in graph theory, though admittedly, a book on graph coloring would have been easier for the non-mathematician to understand if it hadn't been printed in monotone.

The book can also be read as drama: the valiant struggle of the mathematicians to slay the four-color dragon, until at last the true princes succeeded. Unfortunately for the storyteller, the history of the four-color theorem has little sex and no bloodshed, so Wilson is reduced to providing a catalog of eccentricities in an attempt to prove what wild and crazy guys those mathematicians are. I, for one, tired of hearing snippets such as how one professor always brought his dog to class, while another's wife had to color maps on her honeymoon. However, such factoids may make the work more accessible to the lay reader.

The critical philosophical issue exposed by the work of Appel and Haken is a fundamental issue in the aesthetics of mathematics: Is proof an exercise in deductive logic or a social process? Many mathematicians were unhappy with an unfathomable, unverifiable computer program as a major part of a proof. Wilson quotes an anonymous mathematician,

> "In my view, such a solution does not belong to mathematical science at all."

Wilson also reports,

> "Haken's son Armin, by then a graduate student at the University of California, Berkeley, gave a lecture on the four-colour problem . . . At the end, the audience split into two groups: the over-forties could not be convinced that a proof by computer was correct, while the under-forties could not be convinced that a proof containing 700 pages of hand-calculations could be correct."

A nineteenth century mathematician would have surely understood proof as social process; the path from Russell and Whitehead through Gödel to Appel and Haken makes the former more the current view. These days, most now seem to regard computers as mathematical tools, as appropriate to use in the proof process as pencils, and understand not only the distinction between truth and communication, but the necessity for both.

Robert E. Filman
Research Institute for Advanced Computer Science/NASA Ames Research Center