Editorial

ROBERT G. BABB II
Oregon Graduate Institute, USA

RON H. PERROTT
The Queen's University of Belfast, UK

Welcome to the first issue of *Scientific Programming*. In the coming decade, as the world comes to rely more and more on programming to solve real-world engineering, scientific, and social problems, the importance of new languages, tools, environments, and compiler technology to support scientific programmers will increase rapidly. By focusing attention on practical aspects of this emerging technology, we hope that *Scientific Programming* will become mandatory reading not only for all researchers in this area, but for practicing scientific programmers as well.

The emergence of vector/parallel supercomputers has created a wealth of new challenges and opportunities for scientific programmers. Currently, however, reports on new developments in scientific programming are scattered across a wide variety of journals devoted primarily to broader subjects, or in difficult to obtain conference and workshop proceedings.

The main objectives of this *Scientific Programming* are to provide an international forum for

1) the dissemination of state-of-the-art information in the field of scientific programming,
2) the promotion of software technology transfer to the wider programming community.

International in scope, this journal brings together for the first time areas that until now have been thought of as distinct, and more closely related to their parent discipline (parallel processing, software engineering, compiler technology, specific application areas, etc.) than to scientific programming. Papers within these related disciplines will be chosen for publication only if they deal primarily with practical issues of programming of general interest to scientific programmers and scientific programming researchers.

We view *Scientific Programming* mainly as a "bridge" journal between modern computer science and software engineering technology and the world of scientific computing. Articles giving actual experience and results of use of new ideas, tools, and languages are particularly welcome. *Scientific Programming* provides a meeting ground for research in and practical experience with software engineering environments, tools, languages, and paradigms aimed specifically at supporting scientific and engineering computing. Coverage also includes vectorizing/parallelizing/optimizing compiler techniques to support emerging supercomputer architectures, as well as implementation techniques applicable across several areas of scientific programming. We do not intend to focus only on scientific programming issues on supercomputers or parallel processors. Several of today's high-speed RISC-based scientific workstations are faster than the "supercomputers" of just a few years ago, and ways to program scientific applications on workstations and networks of these workstations are also areas of special interest.

RATIONALE

For many years now, it has been apparent that substantial and innovative research and development into software technology has been, and is being performed, in computer science depart-

Received June 1992

Scientific Programming, Vol. 1, pp. 1–10 (1992)
CCC 1058-9244/92/010001-10$04.00

ments and institutes throughout the world; while scientists and engineers have accumulated direct and relevant experience in applying a wide range of computers to large scale and computationally demanding applications.

Gradually amongst computer scientists a consensus has been established as to what is required in a tool to implement particular software applications and when are the best circumstances in which to use a particular tool. A simple historical analogy is with the development of sequential programming languages; it took a decade to identify what constructs are the most suitable to promote the concept of abstract programming and what are the best ways of structuring the data to facilitate that concept. It took perhaps even longer for that technology to percolate into the wider community of professional programmers.

Since the conquest of sequential computing, the computer science community has turned its attention to other demanding and increasingly difficult aspects of computing, for example: how to build a software environment to help the user with the complete programming process, from specification through design etc. to debugging; how to handle the various types of parallelism that were emerging in commercial and experimental systems, etc. In the latter case each type of parallelism has given rise to different techniques and different primitives with which to master and control the potential power of parallel computing for the benefit of ever demanding applications.

At the same time as these computer science developments were taking place, application scientists were enlisting the help of the latest hardware machines and devices. However, the software tools that were being used, in most cases, left much to be desired. This resulted from a lack of cross fertilization and understanding between computer scientists and application scientists. Scientists appreciate the necessity of using the best tools in the advancement of their discipline—this has been a basic underlying principle in all branches of science. In addition, scientists and engineers have built up a substantial armory in techniques for the construction of applications.

Years of experience and experimentation have been accumulated in the design and implementation of software tools and techniques which can benefit the wider programming community. Only if this knowledge is transferred into the working environment of scientists and engineers and utilized in the promotion of their disciplines will success and progress in many application areas be possible. At the same time scientists and engineers have accumulated substantial and relevant experience which can benefit the work of the computer science community. An interaction and exchange of experiences between these two groups would be to their mutual advantage. *Scientific Programming* will provide a forum for this to take place.

TYPES OF CONTRIBUTIONS SOLICITED

Full-length Research Papers

We are looking for readable, high-quality papers that address the key issues underlying current research and development related to programming support for science and engineering. For example, a paper on theoretical advances in scientific compiler technology would probably be more suitable for a journal in computer science or engineering, while a paper on embodying that technology in a compiler, or practical experience reports on use of that technology would be quite appropriate for *Scientific Programming.*

Full-length Practical Experience Papers

We are interested in publishing papers based on "real" experiences in all areas of scientific programming technology, especially papers that attempt to draw wider conclusions from those experiences. If specific performance results are presented, guidelines such as those shown in Figure 1 should be taken into consideration.

Full-length Tutorial Papers

Scientific Programming has already developed areas of considerable technical complexity within it. Even the developers and implementers of the relevant technology can no longer be expected to maintain expertise and awareness of the state-of-the-art in all of these areas. Hence, we would like to encourage submission of in-depth, accessible tutorials within the scope of *Scientific Programming.*

Short Communications

- Letters to the Editor

  These will not be peer-reviewed, but we will attempt to allow relevant authors a chance to respond so that their ideas can appear in the same issue as the original letter.
- Publication, Software Tool, and Programming Environment Reviews

1. If results are presented for a well-known benchmark, comparative figures should be truly comparable, and the rules for the particular benchmark should be followed.

2. Only actual performance rates should be presented, not projections or extrapolations.

3. Comparative performance figures should be based on comparable levels of tuning.

4. Direct comparisons of run times are preferred to comparisons of megaflops rates or the like.

5. Megaflops figures should not be presented for any comparative purpose unless they are computed from consistent flop counts, preferably flop counts based on the best serial algorithm.

6. If speedup figures are presented, the single processor rate should be based on a reasonably well-tuned program without multiprocessing constructs (and overhead).

7. Any ancillary information that would significantly affect the interpretation of performance results should be fully disclosed.

8. Due to the natural prominence of abstracts, figures and tables, special care should be taken to insure that these items are not misleading, even if presented alone.

9. Enough details on the hardware, software, system environment, language, algorithms, datatypes, programming techniques, tuning, and timing techniques should be presented that others could reproduce the performance results presented.

FIGURE 1. Bailey's guidelines for reporting performance results.

These reviews will be done primarily to give early, wide visibility into interesting new ideas and software systems, rather than rigorous competitive product analyses. More details on these are given below in the contribution by Eugene Miya, associate editor for software and publication reviews.

- Short Tutorial Papers

Many topics within scientific programming are narrow enough to be covered in concise tutorial form.

Retrospective Papers

Due partly to the previous lack of a suitable forum for publication, a number of research reports have come to our attention that have been written in the past five to ten years but were never submitted for journal publication. Although the specific computer systems and language technology involved may have become somewhat "dated", we intend to publish these "historic" papers occasionally, partly because they still contain valuable practical lessons, and also they can provide some perspective on areas where significant advances have been made in scientific programming technology.

ORGANIZATION

Reviewers and Authors

Any scientific journal relies for its success on the good will and hard work of authors willing to take the time to write up their results in a readable fashion, and on reviewers that will help not only with leads for good potential paper submissions, but also with the in-depth review and revision cycle that can lead to the publication of very high quality papers. Our general policy for reviews is that the relevant associate editor will contact only potential reviewers likely to have more than a passing interest in the topic of the paper (or publication or software package), and then come to a mutual agreement on a schedule for returning the review results.

The ultimate success of any enterprise such as this journal will depend on the efforts and contributions of a great many people. If the help we have gotten so far is any indication, then *Scientific Programming* has a good chance to be successful in promoting research, development, and dissemination of scientific programming technology.

Editorial Advisory Board

In addition to the help of the associate editors, a number of people active in the area of scientific programming around the world have accepted our invitation to serve in an editorial advisory capacity for the journal.

Duties expected of members of the editorial advisory board of *Scientific Programming* include:

1) Keep a lookout for groups/individuals who have done or are doing interesting work in scientific programming in their geographical area (country, etc.) as well as in their special fields of interest/expertise anywhere in the world.
2) Where appropriate, encourage them to write up the aspects of their work of most interest to the readers of *Scientific Programming*. In general, this means that the practical aspects and lessons learned should be more prominent in the paper than purely theoretical concerns.
3) We aim to have very broad, worldwide coverage of the work in this field. In cases where the technical message of a particular paper would be obscured primarily by problems with English, they will assist the author(s) with this aspect prior to getting the paper into the standard review process.

The list of editorial advisory board members is given on Cover 2 of this journal. We appreciate very much their willingness to help with the journal.

Associate Editors

In order to help with identifying, reviewing, and revising appropriate papers for *Scientific Programming*, we have been very fortunate to be able to enlist the assistance of the following associate editors:

- Jim McGraw (Languages and Paradigms)
  Lawrence Livermore National Laboratory, USA
- David Callahan (Environments and Tools)
  Tera Computer Company, USA
- Bo Kågstrom (Techniques and Experiences)
  University of Umeå, Sweden
- Hans Zima (Compiler Technology)
  University of Vienna, Austria
- Eugene Miya (Software and Publication Reviews)
  NASA Ames Research Center, USA

The associate editors are primarily responsible for managing the review and revision process. They will rely on members of the editorial advisory board for help with identifying and encouraging high-quality paper submissions, as well as for help with finding qualified reviewers with expertise in particular areas. The associate editors will recommend acceptance/rejection of papers whose review cycle they manage. Papers can be submitted initially to one of the co-editors, or to an appropriate associate editor.

Each of the associate editors has contributed a brief position paper to this inaugural issue describing their subject area, and giving their thoughts on the kinds of papers they would like to see published within their respective areas in future issues.

Languages and Paradigms for Scientific Programming

JIM MCGRAW
Lawrence Livermore National Laboratory, USA

This new journal provides a common publication forum to be shared by computational scientists and computer scientists for the interchange of information of mutual benefit. Within the subarea of languages and paradigms, I believe that both groups have much to offer in terms of valuable

lessons learned, intermediate results on new research directions, and new ideas for making scientific programming a more effective computational strategy for advancing science in general. Computational scientists bring to the table a deep understanding of the problem domain and the most effective algorithms for correctly solving those problems in an efficient manner. Computer scientists bring an understanding of linguistic techniques for expressing algorithms in ways that permit the most effective use of particular machine features. My goal as associate editor for languages and paradigms is to publish practical papers that contribute to enhancing understanding of how to best express scientific algorithms. This goal statement contains several key phrases that deserve elaboration.

The phrase "practical papers" emphasizes my interest in seeing the results of real studies, results that can be put into practice by readers. If I have one criticism of the research community from which I have come (computer science), it would be that we tend to generate far too many "new" language ideas (which are more often minor variations on an existing theme) and fail to make a substantial test of those ideas to demonstrate clearly that the new ideas will have a substantial impact on the development of new applications or the long-term maintenance of existing ones. In this sense, I would give the most weight to papers that have analyzed new or existing language ideas (and/or paradigms) against a substantial set of real test problems (i.e., at least one). This is a place where computational scientists can be of enormous help. What are the key algorithmic techniques for future applications? Where are the areas of code development and maintenance in scientific computing that cause the greatest number of programmer errors and computer inefficiencies? I believe that our journal should encourage submission of papers that authoritatively and analytically set out the current and future needs for scientific programming. A key component of such papers could be defining objective measures of success.

The second key phrase I identified in my goal statement stresses "enhancing understanding." In this context, I want to emphasize a charge that I plan to give to all referees who get papers from me to review. If a referee recommends that a paper be accepted, I want to understand clearly what that referee believes readers will gain by reading the article. In this sense, a paper discussing how the author implemented algorithm ABC on XYZ computer using MNO language or paradigm and produced 95% speedup on 1024 processors will not be (in general) enough to get a paper accepted. Within the area of languages and paradigms, we need to focus on the process by which the results were achieved. How long did it take to develop the code? What were the biggest stumbling blocks to expressing the algorithms? If converting from a previous code version, how much of the original code remained untouched? In what ways did the language either enhance or inhibit the efficient translation of the algorithm into high performance execution on the machine? These questions are often unanswered in papers I see published today—mainly because they are hard questions to answer. However, these are also the kinds of questions that can substantially enhance understanding on the part of readers.

The last key phrase in my goal statement points to one of the most difficult aspects of this area of study—demonstrating ways to "best express scientific algorithms." Criteria for "best" can be a challenging and complex issue in itself. Options include: easiest to program, fastest execution time, highest degree of portability without code modification, conciseness in expressing algorithms, and fastest total development time (just to name a few). Depending on the work context, any and/or all of these criteria can be important. What I would most like to see in accepted papers is that a specific set of criteria have been selected and carefully measured as a part of the analysis. In doing the analysis, the authors should be able to explain the results (including performance anomalies) without resorting to guesswork or opinions. In looking at identifying options for claiming "the best" performance, researchers are often drawn to the use of making comparisons. In this context, reviewers will be looking to determine if a true and fair comparison is being made, or if a weak strawman opponent has been selected. Needless to say, I favor comparisons of the former type.

The preceding discussion has focused primarily on a description of the characteristics I would like to see in the papers published. Following are some specific topic areas that I would definitely like to see published in our journal.

High on my list of interesting topics for this subject area would be experiences reported by computational scientists using new styles of programming languages and paradigms. I would tend to keep a broad definition of "new" in this context—basically anything that is not generic Fortran or C.

Of particular interest would be experiences with paradigms such as: object-oriented, data parallel, functional, message passing, and shared-memory parallel programming. In my mind, the important aspect would be to get some concrete data on the successes and failures in using new techniques. Papers that are very subjective in their likes and/or dislikes of features are NOT what I have in mind.

Another important topic within this subject area evaluates the ability of various languages and paradigms to exploit effectively a variety of different computer architectures with "minimal" (or ideally "no") changes to the source code. Based on my experience, it seems that to get peak performance out of a particular architecture, programmers need to fine-tune the underlying algorithm. The important question for most application people is how close can you get to peak without all of those careful changes? What amount of development energy is needed to get those "best" algorithms? These questions are of particular importance when moving among different styles of machines, like the Cray 2, CM 5, Paragon, BBN Butterfly, NCube 2, and Kendall Square.

I realize in all of my examples so far, I have stressed parallel computing. Clearly this is an aspect of scientific computing that is of current importance and growing interest. However, we want to encourage submissions beyond this specific domain. Other types of languages and paradigms may have a dramatic impact on the ability of application developers to build and maintain high-quality scientific software in the future. Data abstraction, object-oriented programming, and high-level application area-specific languages only begin the list of possibilities. New language ideas and their use on scientific applications would be welcome submissions. I would also look favorably on efforts to evolve existing languages like Fortran and C toward newer programming paradigms. For example, what kind of performance could someone get by writing in Fortran, but using a "nearly" functional subset of the language?

Another topic area for which I would like to see greater coverage is that of negative results. I believe we learn more from our mistakes than our successes. Yet, as researchers, we are often discouraged from drawing attention to failures. We can provide a valuable service to practitioners by warning them away from paths that produce poor results. The key point in considering such papers will be the simple rule—is the negative result clearly explained and sufficiently documented to correctly warn off others from encountering the same problem. I will not make it a requirement that the paper include the "right" answer, or even a "better" solution, for the paper to be accepted for publication.

The one last point I wish to include relates to reporting performance results. David Bailey has published a humorous, but pointed article entitled "Twelve Ways to Fool the Masses When Giving Performance Results on Parallel Computers" [1]. The thrust of this article is that without meticulous care and thorough and fair analysis, the numbers reported in journals can be worse than useless. They can be seriously misleading. Based on my readings in the literature, we have too many examples of sloppy performance experiments that lead to erroneous and/or misleading conclusions. I believe David's comments apply equally to the analysis of programming languages and paradigms. Our goal in this journal is to provide material that the common practitioners of scientific programming will find useful and beneficial. We can only meet this criterion if authors and reviewers carefully examine the results reported to make sure that claims of success are warranted. I encourage everyone to read David Bailey's article to raise their awareness of this subtle and often very tricky issue. A summary of the guidelines is shown in Figure 1.

I see this journal as an excellent opportunity for an exchange of ideas among computational science and computer science practitioners for their mutual benefit. Results that have value and applicability to a broad section of readership will have the highest priority. Within the area of languages and paradigms, the field of scientific programming field is exploding with new and innovative approaches to expressing critical applications in an effective and efficient manner. I encourage all of you who have results consistent with the goals set out in this article to consider writing them up as a paper submission to *Scientific Programming*.

Scientific Programming Environments and Tools

DAVID CALLAHAN
Tera Computer, USA

In the area of programming environments and tools, *Scientific Programming* offers a unique opportunity for software researchers to describe recent advances in programming support to an audience that can both build on the research and benefit from the software products directly. My goal as the associate editor for this area is to publish practical papers that describe software tools that automate or simplify some aspect of scientific program development.

A practical paper should include either enabling algorithm development, interesting implementation experience, or experience based on field use by computational scientists. Hybrid papers are of course encouraged.

An algorithm or theory paper must provide new results on the computational properties of some programming task. This can include bounds on asymptotic complexity or new algorithms for some standard problem. Such work must be cast as enabling research for tool development. These papers will be of interest to other developers who may incorporate the work into their own products. These papers will also be of interest to computation scientists so that they can develop an understanding of what is possible and what is plausible to expect in the near future from scientific software tools. Theory papers might also address meta-problems, such as how to evaluate tools or characterize their effectiveness.

An implementation experience paper should address design goals and their ramifications in the implementation of the tool. Particular attention should be paid to tradeoffs in the implementation and their impact on performance and functionality. These papers should emphasize those aspects of the design that were the most troubling, thus providing direction for future research and suggesting changes in programming practice that might alleviate conflicting requirements.

A tool use experience paper should objectively measure the effectiveness of a tool. In general, papers which provide only simple anecdotal evidence that a tool or environment was used are not acceptable. Papers must go on to analyze the strengths and weaknesses of a tool in some setting and attempt to objectively gauge its utility. Objective data on how a tool is used in the field is vital to advancing the state-of-the-art. Which features are used and which are not; why were features not used; what were the productivity gains; what was the measured performance change in applications and job mixes—this information is crucial input to the next generation of tools. In this context, scientific programmers can provide invaluable feedback to tool developers to help explain these data. Computer scientists are often removed from their client users and can lose touch with the daily problems that consume the bulk of scientific program development time.

The focus on scientific programming must be maintained. Papers that would be more appropriate to "mainstream" journals on programming practice should be submitted there. Papers which are particularly motivated by the needs of scientific programmers should be submitted here. What are these needs? I have no specific answer and so each paper which may be borderline should motivate why the tools are of particular interest to scientific programmers.

I am excited about the opportunity presented by the journal to advance the state-of-the-art of scientific programming and to enhance the productivity of scientific programmers. Recent advances in architecture, particularly massively parallel systems, have made critical the role of software tools and programming environments. Through vehicles like this journal we can combine diverse talents to help overcome this crisis.

Scientific Programming Techniques and Experiences

BO KÅGSTRÖM
University of Umeå, Sweden

This subarea focuses on different techniques employed in, and experiences obtained from, advanced software implementations on the large spectrum of today's advanced and high-performance (parallel) computer architectures. Techniques that in some sense are scientific, i.e., general and will influence the future design of software, are of most interest. For example, techniques that apply to different problem classes (non-numerical as well as numerical) and/or focus on architectural considerations, e.g., highlighting the efficient use of hierarchical memory systems of advanced computer architectures. The aim is to develop general and good techniques that can transfer to a design and implementation paradigm. The scope of this subarea is also on experiences of implementing different techniques on advanced computer architectures. Here, the focus should be on the use of techniques on computers rather than on the exposition of results peculiar to a specific research problem. Performance measuring, evaluation, simulation and modeling relating to implemented techniques and experiences to their use are also interesting topics.

We strive for original papers of high scientific quality, e.g., techniques and experiences should be put into a general context and compared with other techniques and experiences published in scientific journals. However, we are also interested in publishing short communications on techniques and experiences that are of general and immediate interest. It is also clear that many topics, etc. will not clearly and easily split into the four subareas of the journal (a paper may cover several subareas). I think it is important that we maintain a "scientific focus" (as stressed in the title of the journal). However, I believe that an opening also exists for short communications which could have a less "scientific focus" if they are of sufficient immediate interest.

Scientific Programming Compiler Technology

HANS ZIMA
University of Vienna, Austria

Given below is a list of the main topic areas that I believe are of the current greatest interest within this subfield of *Scientific Programming:*

1) Compiler Technology for New Languages/ New Language Features
   - Procedural Languages
   - Parallel Language Extensions
     Examples: C*, Data Parallel C, Vienna Fortran, Fortran D, High Performance Fortran (HPF)
   - Functional Languages
   - Very High-Level Languages
     — Domain-Specific Languages and Sys-

tems (DEQSOL,SUSPENSE,ALPAL, etc.)
— Object-Oriented Languages
— Logic Programming Languages

2) Compiler Technology for Current and Novel Computer Architectures
   • Superscalar architectures
   • Parallel architectures
     — SIMD
     — MIMD Shared Memory
     — MIMD Distributed Memory
     — MIMD Virtual Shared Memory
   • Distributed Systems
   • Systolic Systems
   • Vector Architectures
3) Compiler Technology for Support Environments and Tools
   • Restructuring Systems
   • Performance Analysis and Design Tools
   • Measurement
   • Debugging and Tracing
   • Interactive System Support
4) Intelligent Compiler Support
   • Heuristics
   • Pattern Matching Applications
   • Knowledge-based Systems
   • Expert Systems Techniques
   • Interactive Systems
5) Other Topics
   • Compiler Generators
   • Intermediate Languages
   • Rule-based Compiling
   • Analysis and Optimization Techniques

Where is the area heading? In my opinion, the major directions in which compiler technology (this assumes a rather broad concept of "compiler" in the sense in which this term has been used in recent years—clearly, there is an overlap with the languages and tools areas) is heading will be mainly determined by the objective to make architectures transparent and move programming to a higher level. As a consequence, major developments will be in the fields of

1) Algorithm Development—in particular for machine-independent and machine-specific optimizations.
2) Tool Development—in particular for performance analysis, graphics support, and high-level debugging.
3) Intelligent Support Systems—in particular providing automatic decision making capabilities and intelligent assistance.
4) Very High-Level Languages.

Scientific Programming Software and Publication Reviews

EUGENE MIYA
NASA Ames Research Center, USA

I believe:

The average scientific programmer does not have a lot of time.

> "Who has time to read journals?"—
> Forrest Baskett, ASPLOS, IV

The average scientific programmer may:

1) not know a great deal about computer science oriented things:

   e.g., dead lock, Petri nets, Byzantine Agreements, assumptions of consistency and atomicity, numerical stability, critical sections, etc.

2) have a slightly distorted view of computers:

   that all computers have the same instruction set, that commands found on one system can be found on other systems (ha!)

The average scientific programmer has exposure to many different kinds of terminology: concurrent, parallel, multiprocess. They wish to keep their learning of another discipline to a minimum to get the job done.

Our audience for *Scientific Programming*, we hope, will consist of people from widely varying backgrounds: acousticians, biologists, chemists, engineers, mathematicians, physicists, programmers, zoologists, as well as computer scientists. They may or may not know acronyms like ADI, DAGs, NUMI, DMMP, NURB's, etc. We will find programmers who have spent their lives doing scientific programming only on Crays or only on PCs. They may know only old languages, or only very new languages: Fortran, assembly, and BASIC or ML, Linda, and SISAL.

Hands on Imperative

I believe in an emphasis on the "hands on imperative" be it books or software reviews or references. The reader needs an "experience" with a book or package. They want the "experienced" to steer them around things which should be avoided.

We need to help that programmer find his or her way through the morass of confusing information about scientific programming. I would almost suggest a "Dear Abby"–type of column. Almost.

What does a scientific programmer want to know?

Software Reviews

The information and format I want is:

Title
Detailed Summary
Body of the review
The detailed summary will be the most important part (I want "one liners", or 3–4 liners for addresses):

1) Cost?
2) Availability? Address? Demo available?
3) Platforms: hardware? operating system? languages? Does it work on shared AND distributed memory?

3a) Is it real (physical) or simulated (vaporware) [TBA dates]?

4) Restrictions: licensing: site or individual, copying, etc.
5) Usability? Other "-abilities"?
6) "Gotchas" and other surprises?
7) Environmental considerations: lots of disk required? Memory? Files? Special permissions required? Protections?
8) Performance, timing, documentation, help, crashes?

The qualitative descriptive text (review) follows.

Book and Other Publication Reviews

1) Contents: either a table of contents by chapters or parts or sections
2) Publisher information: who, cost, etc.
3) Is it useful?
   a) What are the issues?
   b) Does it get to the point?
   c) Does it have lots of theory?
   d) Lots of math?
   e) What kind of book: survey, practical, theoretical?
4) Interesting special features? (Does it come with software for instance?)
5) Problems with the text (Errors, Controversial statements).
6) Additional comments and conclusion.

REFERENCES

[1] D. H. Bailey, "Twelve ways to fool the masses when giving performance results on parallel computers," *Supercomput. Rev.*, August 1991, pp. 54–55. Also published in *Supercomputer*, Sept. 1991, pp. 4–7.