Misleading Performance Reporting in the Supercomputing Field

David H. Bailey

RNR Technical Report RNR-92-005
December 1, 1992

Abstract

In a previous humorous note, I outlined twelve ways in which performance figures for scientific supercomputers can be distorted. In this paper, the problem of potentially misleading performance reporting is discussed in detail. Included are some examples that have appeared in recent published scientific papers. This paper also includes some proposed guidelines for reporting performance, the adoption of which would raise the level of professionalism and reduce the level of confusion in the field of supercomputing.

The author is with the Numerical Aerodynamic Simulation (NAS) Systems Division at NASA Ames Research Center, Moffett Field, CA 94035.

1. Introduction

Many readers may have read my previous article "Twelve Ways to Fool the Masses When Giving Performance Reports on Parallel Computers" [5]. The attention that this article received frankly has been surprising [11]. Evidently it has struck a responsive chord among many professionals in the field who share my concerns. The following is a very brief summary of the "Twelve Ways":

1. Quote only 32-bit performance results, not 64-bit results, and compare your 32-bit results with others' 64-bit results.

2. Present inner kernel performance figures as the performance of the entire application.

3. Quietly employ assembly code and other low-level language constructs, and compare your assembly-coded results with others' Fortran or C implementations.

4. Scale up the problem size with the number of processors, but don't clearly disclose this fact.

5. Quote performance results linearly projected to a full system.

6. Compare your results against scalar, unoptimized, single processor code on Crays.

7. Compare with an old code on an obsolete system.

8. Base megaflops operation counts on the parallel implementation instead of on the best sequential implementation.

9. Quote performance in terms of processor utilization, parallel speedups or megaflops per dollar (peak megaflops, not sustained).

10. Mutilate the algorithm used in the parallel implementation to match the architecture. In other words, employ algorithms that are numerically inefficient in order to exhibit artificially high megaflops rates.

11. Measure parallel run times on a dedicated system, but measure conventional run times in a busy environment.

12. If all else fails, show pretty pictures and animated videos, and don't talk about performance.

Some readers of the "Twelve Ways" have inquired whether the article was intended to be a criticism of parallel supercomputer vendors or of the technology of parallel computing in general. It was not. This misunderstanding may have been arisen from the fact that in the copy published in *Supercomputing Review*, the word "scientists" in an introductory paragraph was changed by the editors to "vendors" for unknown reasons. Instead, I wrote the article out of concern that the field of parallel supercomputing may lose credibility unless *scientists* themselves are more circumspect in reporting performance and in reviewing the work of their colleagues.

The "Twelve Ways" article jocularly suggests deliberate attempts by scientists to mislead their audiences. Clearly few, if any, who have employed one of these "techniques" have done so with such dark motives. But this sort of material still looks unprofessional and, if nothing else, has resulted in a great deal of confusion in the supercomputing field.

2. The Pressure for Reporting High Performance

Perhaps it was inevitable that this problem would arise in the field of parallel scientific computing. For many years, parallel computers were almost exclusively the province of theoretical computer science. In the mid 1980s, commercial parallel systems became available, but their performance ratings were a far cry from those of commercially available vector supercomputers at the time. In the late 1980s, commercial parallel systems were finally marketed with peak performance rates that equaled and even surpassed that of vector supercomputers. In response to these developments, a number of laboratories acquired these systems, and their scientists began to implement serious, full-scale applications. Inevitably, one of the first questions asked of these researchers was "How does your parallel computer compare with a conventional supercomputer on this application?"

Thus, since about 1988 scientists programming the parallel systems have been under pressure to exhibit performance rates comparable to or exceeding those of conventional supercomputers. This pressure has increased in the last year or two as these same groups of scientists have been called upon by their respective managements to justify the multimillion dollar price tags that serious parallel supercomputers now command. In many cases, aggressive long-range performance goals have been set by high-level managers that will require rapid progress for years to come. Looming budget cuts at large scientific laboratories have added additional pressure. At the very least, researchers feel obligated to highlight experience with certain applications that are naturally well-suited for highly parallel systems and thus achieve high rates, and to downplay more challenging applications that for various reasons do not yet achieve high rates.

When this external pressure is added to the natural human tendency of scientists to be exuberant about their own work, it should come as little surprise that some have presented sloppy and potentially misleading performance material in papers and conference presentations. And since the reviewers of these papers are themselves in many cases caught up in the excitement of this new technology, it should not be surprising that they have tended to be somewhat permissive with questionable aspects of these papers.

Clearly the field of supercomputing in general and parallel computing in particular does not do itself a favor by condoning inflated performance reports, whatever are the motives of those involved. In addition to fundamental issues of ethics and scientific accuracy, there is a real possibility that our field could suffer a serious loss of credibility if, for example, certain instances are given prominent media coverage. At the very least, there is the possibility that laboratory managers and even applications scientists will sour on parallel computing, much in the same way that the expansive claims and promises of the artificial intelligence field in early years has led to the present widespread skepticism.

There is another reason to uphold a high level of forthrightness and clarity in performance reporting. In order for highly parallel computers to achieve widespread acceptance in the scientific computing marketplace, it is essential that they deliver superior performance on a broad range of important scientific and engineering applications. Thus it is my opinion, shared by others in the field, that the best way to insure that future parallel scientific computers will be successful is to provide early feedback to manufacturers regarding their weaknesses. Once the reasons for less-than-expected performance rates on certain problems are identified, then vendors can incorporate the required improvements, both software and hardware, in the next generation.

In this paper, I will present a number of examples of questionable performance reporting that have appeared in published papers during the past few years. I have restricted this study only to articles that have recently appeared in refereed scientific journals and conference proceedings. I have not included press releases, marketing literature, technical reports, verbal presentations or papers I have read only as a referee.

My only purpose in citing these examples is to provide concrete instances of the performance issues in question. I do not wish for these citations to be misconstrued as criticisms of individual scientists, laboratories or supercomputer manufacturers. This is because I personally do not believe this problem is restricted to a handful of authors, laboratories and vendors, but that many of us in the field must share blame (see section 8). For this reason I have decided to take the unusual step of not including detailed references for these papers, in order not to cause the authors undue embarrassment. These references are available, however, to reviewers or others with a legitimate need to know. I suggest that readers who learn the identities of these authors use this information with discretion.

3. Plots

Plots can be effective vehicles to present technical information, particularly in verbal presentations. Further, plots are in many cases all that high-level managers (such as those with authority for computer acquisitions) have time to digest. Unfortunately, plots can also mislead an audience, especially if prepared carelessly or if presented without important qualifying information.

Figure 1 is a reconstruction of the final performance plot from a paper describing a defense application [13]. The plot compares timings of the authors' nCUBE/10 code with timings of a comparable code running on a Cray X-MP/416. The plot appears to indicate an impressive performance advantage for the nCUBE system on all problem sizes except a small region at the far left.

However, examination of the raw data used for this plot, which is shown in Table 1, gives a different picture. First of all, except for the largest problem size (i.e. object count), all data points lie in the small region at the far left. In other words, most of the two curves shown are merely the linear connections of the next-to-last data points with the final points. Further, the Cray X-MP is actually faster than the nCUBE for all sizes

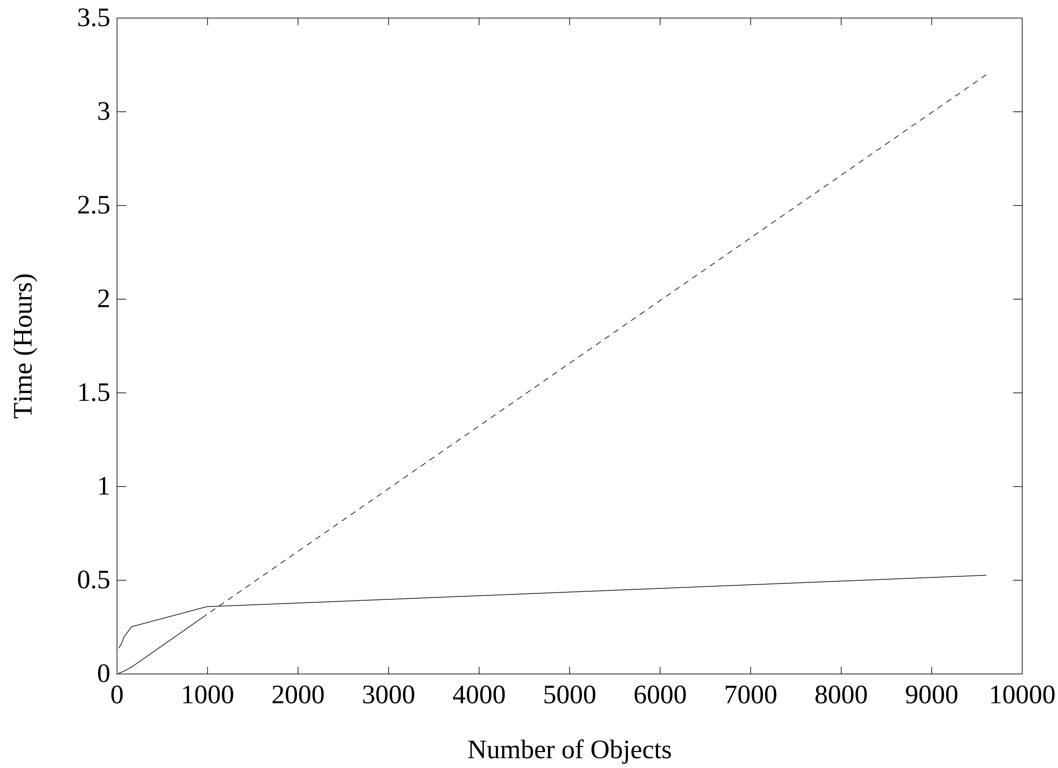

Figure 1: Timings of nCUBE/10 (lower) and Cray X-MP (upper) on a defense application

Total Objects	nCUBE Run Time	Cray X-MP Run Time
20	8:18	0:16
40	9:11	0:26
80	11:59	0:57
160	15:07	2:11
990	21:32	19:00
9600	31:36	*3:11:50

Table 1: Raw data for plot in Figure 1. * denotes estimate.

except for the largest problem size. My personal view, shared by several colleagues who have seen this graph, is that a logarithmic scale would have been more appropriate for this data.

Other difficulties are encountered when one reads the text accompanying this graph and table. First of all, the authors concede that the runs on the Cray X-MP/416 (a four processor system) were made on a single processor, and that "the Cray version of the code has not been optimized for the X-MP". The authors assert, however, that tuning "would not be expected to make a substantial difference".

Secondly, for the largest problem listed, the only one where the Cray fails to outperform the nCUBE, the Cray X-MP timing is by the authors' admission an estimate, an extrapolation based on a smaller run. In the paper, as in Figure 1, the Cray curve leading out to the last point is dashed, possibly intending to indicate that this is an estimate, but this feature is not explained in either the caption or the text. Some readers interpret this feature as merely an indication of the region where the nCUBE is faster.

To the authors' credit, they did include the raw data, and they did clearly acknowledge the fact that the Cray code is not fully optimized and that the last Cray timing is extrapolated. Thus it appears that the authors were entirely professional in the text of their paper. But the reader is left to wonder what fraction of the audience that has seen this plot fully appreciates the details behind it.

Figure 2 is a reconstruction of another performance plot from a paper describing a fluid dynamics application [14]. This plot compares timings of the author's codes running on a 64K CM-2 with those of comparable codes running on a one processor Cray X-MP. The two curves shown for each computer system represent a structured and an unstructured grid version of the code, respectively. As before, the plot appears to indicate a substantial performance advantage for the CM-2 for all problem sizes and both types of grids.

Once again, careful examination of the text accompanying this plot places these results in a different light. First of all, the author admits that his CM-2 results have been linearly extrapolated to a 64K system from a smaller system. The author then explains that the Cray version of the unstructured grid code is "unvectorized".

An additional difficulty with this plot can be seen by carefully examining the two Cray curves. In the original, as in Figure 2, these are precisely straight lines. Needless to say, it is exceedingly unlikely that a Cray code, scaled over nearly three orders magnitude in problem size, exhibits precisely linear timings. Thus one has to suspect that the two Cray "curves" are simply linear extrapolations from single data points. In summary, it appears that of all points on four curves in this plot, at most two points represent real timings.

The author of this article does not mention the precision of the data used in the CM-2 version, but from his description of the CM-2 as having 32-bit floating point hardware, it appears that the author is comparing a 32-bit CM-2 code with a 64-bit X-MP code.

4. Tuning

The previous two examples, in addition to their potentially misleading usage of graphical information, are examples of where performance comparisons are made based on some-

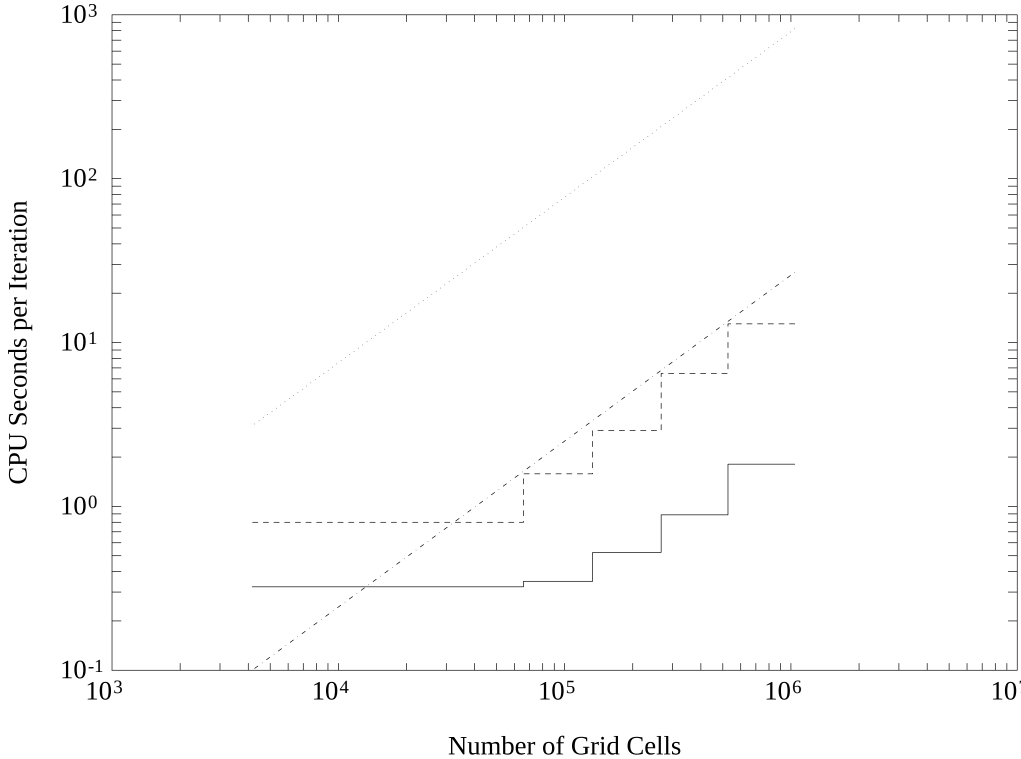

Figure 2: Timings of CM-2 (solid and dashes) and Cray X-MP (dash-dots and dots) on a fluid dynamics application

thing less than comparable tuning efforts. In some cases this may happen because the implementors of parallel codes are experts on a particular parallel system, but they do not have a great deal of experience programming the other system (usually a vector system) that they are comparing against.

For those of us who have significant experience programming both Cray-class vector multiprocessors and the various highly parallel systems, it is pretty clear which are easier to use at this point in time, both for initially implementing an application and for tuning to obtain full performance. Indeed, the difficulty of programming and tuning codes on highly parallel systems is currently an obstacle to more widespread usage. It has been my personal experience that even for applications that are a challenge to vectorize, it is still easier to program and tune them on a single processor of a Cray system than on a highly parallel distributed memory system. This assessment does not substantially change when one includes the additional effort required to utilize autotasking (multiprocessing) on Crays.

Thus one has to be skeptical of instances in the literature where an application has been ported and tuned on a highly parallel computer, usually requiring months of effort, and yet the corresponding Cray code exhibits poor performance, typical of a code that is not even vectorized, much less parallelized. One example of a performance comparison of this sort is in [15]. In this paper, the performance of a code fragment is listed as 18 megaflops on a Cray-2, but the translated code is claimed to run at 741 megaflops on the CM-2. First of all, one can question whether this is a fair comparison, since the Cray-2 performance was for a $16 \times 16 \times 32$ problem, whereas the CM-2 performance was for a $64 \times 64 \times 64$ problem. Also, as in the previous example, it is clear that the author is comparing a 32-bit code on the CM-2 with a 64-bit code on the Cray-2. Further, only one processor of the four-processor Cray-2 is being utilized.

The tuning problem in this paper is evident when one studies the Cray-2 Fortran code fragment that is the basis of this comparison. This code fragment (with minor changes) is shown in Figure 3. The performance of this code on the Cray-2 at NASA Ames is not as low as the author of this article reported — evidently the author's timing was based on an earlier version of the Cray-2 compiler. But it is not hard to see that without special compiler trickery, the performance of this code will be quite poor, since the inner loop vector length is only 16.

Figure 4 contains an equivalent fragment of code, but with the three dimensions of the various arrays collapsed to a single dimension. This tuned code runs on the NASA Ames Cray-2 at 160 megaflops. It is impossible to know how fast this tuned code would have run when the author wrote his article, but it is certain that it would have run much faster than 18 megaflops. It is regrettable that such material appeared in a published conference proceedings. Fortunately, however, the above-mentioned code fragment and performance comparison was omitted when the paper was subsequently republished as a journal article.

Another apparent example of this type of potentially misleading material can be seen in [16]. This paper compares the performance of a physics application running on several systems, including a 16K CM-2 and a Cray X-MP/14. A substantial effort was made to

```
      parameter (nx = 16, ny = 16, nz = 32)
      common /com/ a(nx,ny,nz), gv(nx,ny,nz), t(nx,ny,nz), d(nx,ny,nz),
     $  gr(nx,ny,nz), gi(nx,ny,nz)
c
      do 120 kz = 1, nz
        do 110 jy = 1, ny
          do 100 ix = 1, nx
            t(ix,jy,kz) = a(ix,jy,kz) / gv(ix,jy,kz)
            d(ix,jy,kz) = gr(ix,jy,kz) / gi(ix,jy,kz)
     $        / gv (ix,jy,kz)
 100      continue
 110    continue
 120  continue
```

Figure 3: Original Cray-2 code fragment

```
      parameter (nx = 16, ny = 16, nz = 32, nn = nx * ny * nz)
      common /com/ a1(nn), gv1(nn), t1(nn), d1(nn), gr1(nn), gi1(nn)
c
      do 100 i = 1, nn
        t1(i) = a1(i) / gv1(i)
        d1(i) = gr1(i) / (gi1(i) * gv1(i))
 100  continue
```

Figure 4: Tuned Cray-2 code fragment

tune the CM-2 code, including calls to low-level PARIS routines. The resulting CM-2 implementation runs at 126 megaflops, much faster than the 3.1 megaflops achieved on the Cray. The author explains that the reason for the poor performance on the Cray X-MP is that two key phases of this calculation "do not vectorize".

However, the reader is struck by the fact that both the Cray and the CM-2 can be thought of as SIMD processors. A code that has been implemented efficiently on a CM-2 should be directly translatable to a efficient vectorizable code on the Cray. An excellent demonstration of this principle can be found in [10], where three applications that had been programmed and tuned on the CM-2 were ported back to the Cray Y-MP with high performance.

Thus the reader of [16] is left to wonder why, if the CM-2 code for this application runs so well, did the author not try to adapt the CM-2 code to the X-MP? Since one of the author's CM-2 codes was written entirely in CM Fortran (i.e. Fortran-90 with directives), why did the author not try simply running this same code on the X-MP? After all, the Cray Fortran compiler now accepts many of the Fortran-90 array constructs.

5. Projections and Extrapolations

The practice of citing estimated and extrapolated performance results is, unfortunately, fairly widespread in the field. This may in part be an unintended consequence of limited research budgets at many research labs, where scientists often have to settle for scaled-down versions of highly parallel systems. As a result, researchers frequently cite performance results that are merely linear projections from much smaller systems, often without the slightest justification.

The practice of linearly extrapolating one's performance results to a larger system is doubly perplexing because the question of whether various computer designs and applications will "scale" is in fact an important topic of current research. It seems that many scientists using parallel computers are willing to assume as an established fact one of the most fundamental questions in the field!

We have already seen one instance of citing extrapolated results. Another example is [19], where the authors compare their defense application running on an nCUBE-2 with comparable codes running on a Cray Y-MP and a CM-2. Three tables of timings are included in this paper. Fortunately, all of the nCUBE-2 timings in the three tables are real timings. But out of a total of 33 figures listed for the Y-MP and CM-2, more than half (17) are merely projections or estimates. There does not appear to be any attempt to mislead the reader, since the authors indicate which figures in each table are projections by means of asterisks. Nonetheless, one is left to wonder about how reliable these comparisons are, and whether they will always be quoted with the appropriate disclaimer.

In most cases authors clearly disclose estimates and projections, but not always. In [17], the author gives performance results for his fluid dynamics code in a table at the end of the article. Timings are included for an 8K CM-2, a 16K CM-2 and a 64K CM-2. Curiously, the timings for the 64K system have parentheses around them, but nowhere in the text does the author state the meaning of these parentheses. However, by noting that this column of numbers is identical to the 16K numbers, shifted down by one, one has to conclude that the 64K numbers are merely linear projections from the 16K results.

Some authors have taken the practice of citing projections one step further. In [18] the author states in his abstract that his code runs "at the speed of a single processor of a Cray-2 on 1/4 of a CM-2". Some thirteen pages later, the author cites a timing on a Convex C210 and then states "experience indicates that for a wide range of problems, a C210 is about 1/4 the speed of a single processor Cray-2". No further mention is made of the Cray-2.

It is well known that for both the Convex C210 and the Cray-2, timings and megaflops rates can vary dramatically depending on the level of vectorization, inner loop vector lengths, compute-to-memory reference ratios, compiler features and other factors. Thus any blanket performance ratio such as 1/4 is rather dubious. But the most troubling item here is the fact that the author, in the abstract of his paper, implies a performance comparison with a Cray-2, even though he evidently has never run his code on a Cray-2.

6. Counting Flops

A common practice in the field of scientific computing is to cite performance rates in terms of millions of floating point operations per second (megaflops). For various reasons, some in the field have suggested that the practice of citing megaflops rates be abandoned. However, I am of the opinion that while direct timing comparisons are always preferred, megaflops rates may be cited if calculated and reported consistently.

Megaflops figures may of course be misleading, particularly on parallel computers. This confusion derives from the method used to determine the number of floating point operations (flops) performed. Many authors count the number of flops actually performed in their parallel implementations, a number usually obtained by analyzing the parallel source code. Megaflops figures computed in this manner may be used, for example, to indicate the extent to which the peak processing power of the computer system is being utilized.

However, parallel implementations almost always perform significantly more flops than serial implementations. For example, some calculations are merely repeated in each processor. Using the actual number of flops performed on a parallel computer thus results in megaflops rates that are inflated when compared to rates obtained from corresponding serial or vector computer implementations.

Another difficulty with basing megaflops rates on the actual parallel flop count is that this practice tacitly encourages scientists to employ numerically inefficient algorithms in their applications, algorithms often chosen mainly for the convenience of the particular architecture being used. It is easy to understand how such choices can be made, since it is widely accepted in the field that algorithmic changes are often necessary when porting a code to a parallel computer. But when this practice is carried too far, both the audience and the scientist may be mislead.

Because of the potential for misleading comparisons of megaflops figures, it is clear that a single standard flop count should be used when comparing rates for a given application. In my view, the most sensible flop count for this purpose is the minimal flop count — the value

Solver Algorithm	Floating Point Operations	CPU Time (Secs.)	Parallel megaflops	BPSA megaflops
Jacobi	3.82×10^{12}	2124	1800	1.00
Gauss-Seidel	1.21×10^{12}	885	1365	2.41
Least Squares	2.59×10^{11}	185	1400	11.51
Multigrid	2.13×10^{09}	6.7	318	318.00

Table 2: Parallel megaflops rates versus BPSA megaflops rates

based on an efficient implementation of the best practical serial algorithms. In this way, one is free to use an implementation with a higher flop count on a particular architecture if desired, but no extra credit is given for these extra operations when megaflops rates are computed. This standard also acts as a deterrent to the usage of numerically inefficient algorithms.

I have seen definite instances of inflated flop counts in papers I have read and in technical presentations I have attended, but it has been difficult to find clear-cut examples in published literature that are understandable to a general audience. Rather than cite examples of this type, I wish to cite instead some legitimate published results that emphasize the distinction between "parallel" megaflops rates (i.e. megaflops rates based on the actual number of flops performed on the parallel system) and what I will term "BPSA megaflops" (i.e. megaflops rates based on the flop count of the "best practical serial algorithms").

The first paper contains an interesting comparison of several different numerical schemes that can be used to solve a convection-diffusion problem [12]. Based on the authors' data, I have computed both parallel megaflops rates and BPSA megaflops rates for four of these schemes. These figures are shown in Table 2. The BPSA megaflops rates are based on the flop count of the multigrid algorithm.

When one looks at the column of parallel megaflops figures, it appears that the Jacobi scheme is the fastest, with a performance rate of 1800 megaflops. The other schemes are slower, and the multigrid scheme, at only 318 megaflops, is the slowest of all. However, when one examines the BPSA megaflops column, then a very different picture emerges: the Jacobi scheme is the worst, and the multigrid scheme is the best. Thus while the parallel megaflops figures may provide some useful information, it is clear that the BPSA megaflops figures are more meaningful when comparing computational performance.

Another article that emphasizes this same point is [8], where the authors of the Slalom benchmark describe Bjorstad and Boman's discovery. These two scientists found that a preconditioned conjugate gradient method could be used to produce the required solution of the Slalom benchmark in much fewer flops than the scheme previously used, albeit at a lower megaflops rate on many systems.

Admittedly, it may be a challenge to determine the minimal (i.e. BPSA) flop count for a given problem. However, at the least a scientist should be expected to analyze the source code of an efficient implementation on a serial or vector computer. Those with access to a NEC SX system or to a Cray X-MP/Y-MP system can take advantage of the hardware performance monitors present on these computers to obtain accurate flop counts. However, one must still be careful to insure that the code being measured by the hardware performance monitor employs the best available algorithms and is well optimized.

Along this line, perhaps those of us performing research in the area of numerical linear algebra should at some point reconsider the usage of classical formulas for flop counts in favor of flop counts based on implementations that employ Strassen's algorithm [2]. Strassen's algorithm is a scheme to multiply matrices that requires fewer floating point operations than the conventional scheme. It has been demonstrated that Strassen's algorithm is now practical and in fact produces real speedups for matrices with dimensions larger than about 128 [2]. Further, Strassen's algorithm can be employed to accelerate a variety of linear algebra calculations [3, 9] by substituting a Strassen-based matrix multiply routine for the conventional matrix multiply routine in a LAPACK [1] implementation. If a Strassen-based flop count were adopted for computing the megaflops rate in the solution of a $16,000 \times 16,000$ linear system, the resulting rate would have to be cut by roughly one-third from the usual reckoning.

In this vein, I myself must confess to citing potentially misleading performance figures in [3]. These articles include one processor Cray-2 and Y-MP performance rates for some Strassen matrix routines. Following established custom, my co-authors and I computed megaflops rates based on the classical flop count for matrix multiplication ($2n^3$). But since the Strassen routines can produce the matrix product in fewer flops, it could be argued that these megaflops figures are inflated.

7. Other Issues

Many authors report "speedup" figures for their parallel applications. Such figures indicate the degree to which the given application "scales" on a particular architecture. However, here also there is potential for the audience to be mislead, especially when speedup figures are based on inflated single processor timings.

For example, users of the Intel iPSC and other message passing systems often base speedup figures on a single node timing of the multiple node version of the program (for example, in [20]). When running on a single node, the multiple node program needlessly synchronizes with itself and passes messages to itself. These "messages" are handled quite rapidly, since the operating system recognizes that these are local transmissions. Nonetheless, a significant amount of overhead is still required, and it is not unusual for the single node run time to increase by 20 percent with the addition of message passing code. There is a similar potential for distortion when citing speedup figures for Cray multiprocessor vector systems. Clearly speedup figures should be based instead on the timing of a well optimized, purely single processor program (i.e. a program without unnecessary multiprocessor constructs).

Some authors present "scaled speedup" figures, first introduced in [7], where the problem size is scaled up with the number of processors. Such figures may be informative, but it is essential that authors who quote such figures clearly disclose the fact that they have scaled their problem size to match the processor count. It is also important that authors provide details of exactly how this scaling was done.

Another aspect of performance reporting that needs to be carefully analyzed is how the authors measure run time. Most of the scientists I have queried about this issue feel that elapsed wall clock time is the most reliable measure of run time, and that if possible it should be measured in a dedicated environment. By contrast, CPU time figures, such as those frequently quoted by users of Cray systems, may mask extra elapsed time required for input and output. Also, it is known that on the CM-2, "CM Busy Time" and "CM Elapsed Time" are quite different for some codes, even with no I/O and no other users sharing the partition. This can be seen in [4, 6].

One final aspect of performance reporting is the source of untold confusion in the supercomputing field: are the results for 32-bit or 64-bit floating point arithmetic? Since on many systems, 32-bit computational performance rates are nearly twice as high as 64-bit rates, there is a temptation for authors to quote only 32-bit results, to fail to disclose that rates are for 32-bit data, and to compare their 32-bit results with others' 64-bit results. It is clear that 32-bit/64-bit confusion is widespread in performance reporting, since we have seen several examples already.

In my view, quoting 32-bit performance rates is permissible so long as (1) this data type is clearly disclosed and (2) a brief statement is included explaining why this precision is sufficient. Along this line, it should be kept in mind that with new computer systems it is now possible to attempt much larger problems than before. As a result, numerical issues that previously were not significant now are significant, and some programmers are discovering to their dismay that higher precision is necessary to obtain meaningful results.

I suspect that in the majority of cases where the authors do not clearly state the data type, the results are indeed for 32-bit data. One example of this is [21], where in an otherwise excellent ten-page paper, the authors never state whether their impressive performance rates are for 32-bit or 64-bit calculations, at least not in any place where a reader would normally look for such information. That their results are indeed for 32-bit data can however be deduced by a careful reading of their section on memory bandwidth, where we read that operands are four bytes long.

8. Responsibility

It is most likely true that none of the authors cited above deliberately intended to mislead their audiences. After all, in most cases the potentially misleading aspects of these papers were evident only because of detailed information included in the text of the paper. It is also likely that in at least some cases, the authors' performance claims might be largely upheld if the full facts were known. Nonetheless, the overriding impression of these examples is that whatever the motives and actual facts may be, the material as presented generally gives the appearance of inflating the authors' performance results in comparison to other systems. Such material certainly has the potential to mislead an audience. And, at the very least, one can argue that these papers represent sloppy science.

Who is to blame? As I stated in section 2, it is not my opinion that the blame lies solely with the individual authors. Any scientist can write a paper that is not thoroughly sound — it is the duty of his or her colleagues and professional organizations to insure that questionable aspects of the paper are corrected before publication or public presentation. For example, in the above cited examples, these results were almost certainly presented to colleagues at the respective institutions, and in most cases the manuscripts were read by colleagues before the authors submitted them for outside publication. More significantly, these manuscripts were in all but four cases formally refereed by fellow professionals in the field. The four conference papers that were were not formally refereed were informally refereed by committee members of major conferences, who are generally prominent scientists in the supercomputer field. Thus whatever "blame" is to be assigned must be shared rather widely. Furthermore, how can these authors be accused of violating the "rules" when we have never established any standards for reporting performance?

9. Proposed Guidelines

Clearly this field needs a detailed set of guidelines for reporting supercomputer performance, guidelines which are formally adopted and widely disseminated to authors and reviewers. Virtually every field of science has found it necessary at some point to establish rigorous standards for the reporting of experimental results, and ours should be no exception. To that end, I propose the following. These guidelines focus on computational performance, since that is the topic of this paper and apparently the most frequent arena of confusion. However, it is hoped that the spirit of these guidelines will be followed by researchers reporting performance in other areas of supercomputers, such as in mass storage and local area networks.

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. For example, Linpack 1000 results should not be compared against Linpack 100 results.

2. Only actual performance rates should be presented, not projections or extrapolations. For example, performance rates should not be extrapolated to a full system from a scaled-down system. Comparing extrapolated figures with actual performance figures, such as by including both in the same table, is particularly inappropriate.

3. Comparative performance figures should be based on comparable levels of tuning. For example, results tuned with months of effort on one system should not be directly compared with results on a well-known vector system where no attempt has been made to even fully vectorize the code.

4. Direct comparisons of run times are preferred to comparisons of megaflops rates or the like. Whenever possible, timings should be true elapsed time-of-day measurements (this might not be possible in some "production" environments).

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

implementations of the best practical serial algorithms. One intent here is to discourage the usage of numerically inefficient algorithms, which may exhibit artificially high performance rates on a particular parallel system.

6. If speedup figures are presented, the single processor rate should be based on a reasonably well tuned program without multiprocessing constructs. If the problem size is scaled up with the number of processors, then the results should be clearly cited as "scaled speedup" figures, and details should be given explaining how the problem was scaled up in size.

7. Any ancillary information that would significantly affect the interpretation of the performance results should be fully disclosed. For example, if the results are for 32-bit rather than for 64-bit data, or if assembly-level coding was employed, or if only one processor of a conventional system is being used for comparison, these facts should be clearly stated.

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. For example, if significant performance claims are made in the abstract of the paper, any important qualifying information should also be included in the abstract.

9. Whenever possible, the following should be included in the text of the paper: the hardware, software and system environment; the language, algorithms, the datatypes and programming techniques employed; the nature and extent of tuning performed; and the basis for timings, flop counts and speedup figures. The goal here is to enable other scientists to accurately reproduce the performance results presented in the paper.

I am not presenting these guidelines as mandatory, inflexible requirements. Clearly in a fast-moving field such as ours, it would be unwise to do so. However, if a paper or presentation does present results that significantly deviate from these guidelines, I suggest that the author has an obligation to clearly state and justify these deviations.

10. Conclusions

The examples I have cited above are somewhat isolated in the literature, and I see no evidence that the problem of inflated performance reporting is out of control. However, clearly those of us in the parallel supercomputing field would be wise to arrest any tendency in this direction before we are faced with a significant credibility problem. As was mentioned above, scientists in many other disciplines have found it necessary to adopt rigorous standards for reporting experimental results, and ours should be no exception. It is my hope that this article, with the proposed guidelines in section 9, will stimulate awareness and dialogue on the subject and will eventually lead to consensus and formal standards in the field.

Acknowledgments

I wish to acknowledge the following persons who have provided valuable insight and suggestions: R. Babb, E. Barszcz, S. Chatterjee, R. Fatoohi, S. Hammond, A. Karp, C. Koebel, T. Lasinski, J. McGraw, J. Riganati, R. Schreiber, M. Simmons, H. Simon, V. Venkatakrishnan, S. Weeratunga, and M. Zosel.

References

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov and D. Sorenson, *The LAPACK Users' Guide*, SIAM, Philadelphia, 1992.

[2] D. H. Bailey, "Extra-High Speed Matrix Multiplication on the Cray-2", *SIAM Journal on Scientific and Statistical Computing*, vol. 9, no. 3, (May 1988), p. 603 – 607.

[3] D. H. Bailey, K. Lee, and H. D. Simon, "Using Strassen's Algorithm to Accelerate the Solution of Linear Systems", *Journal of Supercomputing*, vol. 4., no. 4 (Jan. 1991), p. 357 – 371.

[4] D. H. Bailey and P. O. Frederickson, "Performance Results for Two of the NAS Parallel Benchmarks", *Proceedings of Supercomputing '91*, IEEE, Los Alamitos, CA, 1991, p. 166 – 163.

[5] D. H. Bailey, "Twelve Ways to Fool the Masses When Giving Performance Results on Parallel Computers", *Supercomputing Review*, August 1991, p. 54 – 55. Also published in *Supercomputer*, September 1991, p. 4 – 7.

[6] M. Garbey and D. Levine, "Massively Parallel Computation of Conservation Laws", *Proceedings of the Fourth SIAM Conference on Parallel Processing for Scientific Computing*, SIAM, Philadelphia, PA 1990, p. 340 – 345.

[7] J. Gustafson, G. R. Montry and R. E. Benner, "Development of Parallel Methods for a 1024-Processor Hypercube", *SIAM Journal on Scientific and Statistical Computing*, vol. 9, no. 4 (July 1988), p. 609 – 638.

[8] J. Gustafson, D. Rover, S. Elbert and M. Carter, "SLALOM: Surviving Adolescence", *Supercomputing Review*, December 1991, p. 54 – 61.

[9] N. J. Higham, "Exploiting Fast Matrix Multiplication Within the Level 3 BLAS", *ACM Transactions on Mathematical Software*, vol. 16 (1990), p. 352 – 368.

[10] O. M. Lubeck, M. L. Simmons and H. J. Wasserman, "The Performance Realities of Massively Parallel Processors: A Case Study", *Proceedings of Supercomputing '92*, to appear.

[11] J. Markoff, "Measuring How Fast Computers Really Are", *New York Times*, September 22, 1991, p. 14F.

[12] J. N. Shadid and R. S. Tuminaro, "Iterative Methods for Nonsymmetric Systems on MIMD Machines", *Proceedings of the Fifth SIAM Conference on Parallel Processing for Scientific Computing*, SIAM, Philadelphia, 1992, to appear.

[13] Paper A.

[14] Paper B.

[15] Paper C.

[16] Paper D.

[17] Paper E.

[18] Paper F.

[19] Paper G.

[20] Paper H.

[21] Paper I.