FLOATING POINT ARITHMETIC IN FUTURE SUPERCOMPUTERS

David H. Bailey,[1] Horst D. Simon,[2] John T. Barton,[1] and Martin J. Fouts[3]

[1]NASA Ames Research Center, Mail Stop 285, Moffett Field, California 94035.

[2]Boeing Computer Services (mailing address, NASA Ames Research Center).

[3]Intergraph Corp., 2400 Geng Road, Palo Alto, California 94303.

The format and functional design of floating point hardware in a supercomputer tend to persist a very long time because of the great resistance of both hardware engineers and customers to any change in such a basic aspect of the system. For example, the hexadecimal floating point format of the IBM 370 series has not changed for over 20 years, and the floating point format on the new CRAY Y-MP is identical to that of the original CRAY-1. Furthermore, numeric idiosyncrasies present in the original design of these two systems are generally still present in their successor systems today. Thus, it is crucial that issues relating to this topic be carefully considered by any manufacturer that has an opportunity to start from a reasonably clean slate, since it is likely that the basic design will still be in place 20 years from now.

Certainly the first consideration in the floating point design of a supercomputer is performance. No degree of numerical fidelity or application convenience can compensate for a design that results in a serious performance degradation for the basic floating point operations. Nonetheless, within the constraints of high performance, we feel that certain features of a floating point design can significantly enhance its usability. The principal issues to be considered in such a design are

word size (32 bit, 64 bit, etc.),

hardware support for extended precision,

format (Cray, CDC, IBM, VAX, IEEE, etc.), and

accuracy characteristics (including the issue of true division versus reciprocal approximation).

In this article the above issues are discussed from the perspective of the Numerical Aerodynamic Simulation (NAS) Systems Division at NASA Ames Research Center.

WORD SIZE

Historically many systems, both large and small, have supported both 32-bit and 64-bit floating point arithmetic. Looking into the future, however, we do not see that 32-bit arithmetic is either essential or desirable. Particularly as we move to TFLOPS class systems, the usual 32-bit format, with only 24 mantissa bits, will be completely inadequate. Although there may be some applications that are large enough to require an extremely high-performance computer, and yet numerically stable enough that 32-bit accuracy is sufficient, we do not feel that this limited class of problems justifies the additional hardware cost. We feel that the cost of whatever hardware such features would require would be better spent on other improvements.

HARDWARE SUPPORT FOR EXTENDED PRECISION

Sixty-four-bit floating point arithmetic, with at least 48 bits of mantissa, seems to be satisfactory at present and for the next 5 years or so. Looking forward to the year 2000 and beyond, however, we are concerned that 64-bit arithmetic will not be adequate for some applications. While we do not yet feel that it is time to insist on a 128-bit format as the default precision, an increasingly compelling case can be made for some hardware support to facilitate fast extended precision operations on future

The International Journal of Supercomputer Applications, Volume 3, No. 3, Fall 1989, pp. 86–90.

systems. With such a facility, at least some critical calculations in an application program could be done using the Fortran DOUBLE PRECISION data type without the substantial performance reduction that results on many of today's supercomputers.

One study that indicates such a requirement is by Marshal L. Merriam, a fluid dynamicist at NASA Ames ("Precision requirements for supercomputers," unpublished). His study concludes that for a model three-dimensional Poisson PDE problem, the numerical round-off error will exceed errors due to the discretization of the grid unless the number of mantissa bits exceeds the number of bits in the array size by at least a factor of 4/3 or so. He concludes that the 48 mantissa bits typical of today's supercomputers will be inadequate once problems about 1,000 times the size of today's large problems (64 million words) are attempted.

A similar situation appears to loom even closer for the finite element calculations typical of both structural codes and newer fluid dynamics codes. For example, scientists at the MacNeal-Schwendler Corporation who support the MSC/NASTRAN package have observed numerical difficulties when increasing the size of their models not too far beyond the size of what is considered "large" today. In tests solving a static problem for models with about 130,000 degrees of freedom, the residual for the computed solution was about 10^{-3}, as compared with a residual of 10^{-8} for a similar model with about 65,000 degrees of freedom (L. Komzsik, MacNeal-Schwendler Corporation, personal communication). With a residual this large, the accuracy of the solution is questionable.

One way to support extended precision is simply to provide hardware instructions for a full 128-bit floating point data format (i.e., with either 12 or 16 exponent bits, and the remaining 112 or 116 bits as mantissa). This approach has already been taken by at least one Japanese manufacturer (NEC, Inc.), although these operations are at present only provided in scalar mode.

Another approach for extended precision is merely to provide some enhancements to the normal single precision operations that would enable extended precision to be achieved from software, using only a few hardware instructions. For example, the hardware multiply instruction could be changed from the current "truncated" pyramid design as in Cray machines to optionally produce the entire double-word result. Curiously, such hardware features were present in earlier high-speed scientific computers, notably the CDC 7600 series. Unfortunately, such features were never incorporated in Cray systems, and while the ETA line has retained some of these features in hardware, the ETA Fortran compiler has never been upgraded to support them.

One compelling reason to seriously consider the IEEE floating point standard is that it allows double precision operations to be performed with only about 10 instructions. A slowdown of a factor of 10 to perform double precision is acceptable. In any event, it is far superior to the factor of 80 or so penalty that must be paid on today's Cray computers.

While it is doubtful that 128-bit precision will prove inadequate for the main body of scientific computations any time in the foreseeable future, there are a few interesting scientific problems that even now require very high precision. One of these is public key cryptography, which requires repeated multiplications of numbers over 200 digits in size. This application may become important if these public key cryptosystems start to be routinely used in password verification, for example. Other applications of this sort include studies in mathematical number theory and in computational chaos (D. H. Bailey, "Numerical results on the relations between fundamental constants," *Math. Comp.*, in press). While such applications are at present neither sufficiently numerous nor of high enough priority to justify special hardware, it is worthwhile to note that hardware features that facilitate fast double precision arithmetic would also permit multiple precision arithmetic to be performed more efficiently.

FORMAT

We feel that the best format for floating point arithmetic is the IEEE 64-bit standard, with 11 exponent bits, 52 mantissa bits (actually 53 counting the "hidden bit"), and a sign bit (*IEEE standard for binary floating point numbers,* ANSI/IEEE Standard 754-1985, IEEE, New York, 1985). There are several reasons for this. First, having this format on a supercomputer would greatly facilitate distributed applications, such as the remote interactive particle tracer, which was written here at NAS. Scientists using the NAS system are enthusiastic about such combinations of workstation graphics and supercomputer computation and wish to pursue other such possibilities. All workstation manufacturers and virtually all mini-supercomputer vendors now employ this standard format, leaving only the established mainframe vendors with different formats. As a result, costly format conversion is required whenever data are communicated between a supercomputer and any other system. In addition, the widespread availability of high-performance workstations and mini-supercomputers has encouraged users to develop and debug their codes on these systems and then move them to mainframe supercomputers for production runs. Such code movement is sometimes problematical when the floating point formats are not the same.

There are some who have worried

about the limited exponent range of the IEEE 64-bit format (11 bits as compared with 14 bits on Cray systems). However, virtually none of the NAS users normally works with wide range data, and almost all would welcome the additional mantissa bits in exchange. Furthermore, it has been the authors' experience that even for those few applications (such as probability calculations or multiple precision calculations) which involve wide exponent ranges, occasionally even the Cray exponent range proves inadequate, and such applications eventually have to be rewritten to employ some sort of scaling. Besides, programmers of such applications have learned that without such scaling, their codes are not transportable to other computers.

ACCURACY CHARACTERISTICS

In addition to compatibility with other computer systems, a principal reason for preferring the IEEE 64-bit standard is its excellent numerical stability. First of all, there are four more mantissa bits than on Cray systems (five if the hidden bit is counted). Second, the IEEE standard requires that the results of all operations be correct to the nearest bit compared to the result using infinite precision arithmetic. This is substantially more accurate than on current supercomputers, where results (especially quotients) are typically accurate only to about 46 bits.

It must be acknowledged that even with the IEEE standard, some anomalies are unavoidable, such as when a program tries to compare two floating point numbers that theoretically should be equal. Indeed, programmers of any scientific computer must be warned never to write code that depends on complete accuracy in results. Nonetheless, the IEEE design is far superior in this regard to the Cray design, whose numerical deficiencies have been noted before (W. Kahan, "Handouts for floating point lectures," unpublished). The principal deficiency of the Cray design is its inaccurate divide, which of course results from the fact that there is no true divide operation, only a reciprocal approximation followed by multiplication. The addition/subtraction and multiplication operations have also been faulted in these studies. For example, no guard or sticky bit is employed in the add/subtract operation, and a truncated pyramid scheme is used for multiplication.

An example of the impact of the relatively inaccurate Cray arithmetic can be seen in some recent NASTRAN structural analysis computations. For the dynamic analysis of large structures, it is critical to distinguish between rigid body modes (i.e., exact zero eigenvalues of the model) and the first nonzero mode. On machines like a VAX, with a 53-bit mantissa and relatively clean numerical characteristics, such computed eigenvalues are of the order of 10^{-12}. Because of the shorter mantissa and the less accurate arithmetic, the same computed eigenvalues are of the order of 10^{-6} on the CRAY X-MP. This creates a real difficulty for the engineer, since these values are sometimes difficult to distinguish from the first truly nonzero modes of the structure, which can be of this magnitude (L. Komzsik, personal communication).

The most frequently noted numerical anomaly in the Cray systems is the fact that the quotient of two exactly equal numbers is not guaranteed to be 1. This anomaly is perhaps unavoidable in any system that lacks a true divide operation. However, consider the following example, which was run on our CRAY-2 using the latest version of the CFT-77 compiler:

```
    PROGRAM TEST
    READ (5, 1) X, Y
1   FORMAT (Z16)
    Z = AMOD (X, Y)
    WRITE (6, 2) X, Y, Z
2   FORMAT (1PE25.15)
    STOP
    END
Input:
4009F9FFFFFFFFFF
4009FA0000000000
Output:
     4.999999999999982E+02
     5.000000000000000E+02
   -1.818989403545856E-12
```

The result of this AMOD operation is a negative number! Although this anomaly was identified quite some time ago, Cray has not yet rectified it. It could be fixed by modifying the compiler to optionally add a few instructions to the code generated for an AMOD reference.

It may be true that the strict accuracy requirement of the IEEE standard may be difficult to achieve in a vector supercomputer design without a significant sacrifice in performance. However, the success of such firms as Weitek, MIPS, and Intel in implementing high-performance versions of the IEEE standard should be noted. In any event, we feel that the potential consequences of adopting a less accurate design must be fully understood by the designers before the final design is set in silicon.

One compromise in this regard is to implement at least one of the four rounding modes specified in the IEEE standard. A significant number of the numeric anomalies that can occur in floating point computation could be avoided if the rounding mode for the basic arithmetic operations, particularly addition/subtraction and multiplication, were consistent (such as always using R* rounding).

The issue of true division versus reciprocal approximation is a difficult one, since many of Cray's numeric anomalies result from this feature. Nonetheless, we acknowledge that it might not be possible to implement the full-accuracy divide operation specified by the IEEE standard without significant cost in real estate or performance.

Furthermore, we acknowledge that division operations are not executed very often in the fluid dynamics codes that are of central importance to NAS users. If reciprocal approximation is implemented instead of a true division, we would recommend that some care be taken, both in the hardware and compiler design, to minimize the anomalies that can result. For example, we recommend that the hardware and compiler design ensure that the results of intrinsics such as AMOD conform to the specifications of the Fortran 77 standard.

It is important to note that effective hardware support for 128-bit arithmetic could ameliorate some inaccuracy in the 64-bit operations. For example, if 128-bit floating point addition were available at low cost, users concerned about inaccuracy in 64-bit additions or subtractions could use the 128-bit operations in critical areas of their program and compare the results with the 64-bit mode. Also, it should be noted that if 128-bit floating point addition/subtraction and multiplication were available at low cost, an efficient 128-bit division operation could be obtained by first computing the 64-bit reciprocal of the divisor, after which one 128-bit Newton iteration, followed by a 128-bit multiplication by the dividend, should produce the 128-bit quotient to within 2 bits or so.

OTHER FEATURES

It should be noted that the IEEE standard specifies that a square root operation, also accurate to the nearest bit, must be provided. In addition, both the CRAY-2 and CRAY Y-MP designs feature hardware instructions to facilitate the extraction of square roots. However, we do not feel that the square root operation deserves such attention. None of the current scientific applications running on the NAS system critically relies on high-speed square roots, for example. Given that square roots can be computed even in vector mode with only about 12 standard hardware operations, we would not object if no hardware of any sort were devoted to this operation on a future system. We recommend that the real estate required for a square root be devoted to other features that we have mentioned.

The IEEE standard specifies certain special numbers, including overflow, underflow, not-a-number, and a series of denormalized numbers (i.e., numbers less than the smallest normalized floating point number). We do not see a compelling case for the inclusion of denormalized numbers. However, we do feel that it is important that a floating point design allow for overflow, underflow, and not-a-number. It would be sufficient to trap to an error-handling routine whenever a program attempts to compute with one of these numbers.

There are many other fine details in the IEEE standard. While we do not regard any of these details as regrettable, we recognize that in a supercomputer some of these might not be worth their hardware cost. However, it must be emphasized that many of these features were included for a purpose, and they should not be casually discarded. For example, the specifications of the REM function permit a completely reasonable Fortran AMOD function to be implemented. Also, the suggested software implementation of 128-bit floating point arithmetic relies on control of the rounding mode. Thus, if any of these ancillary features are omitted or significantly altered, it is essential to verify that such alterations will not result in unacceptable numerical anomalies or in unacceptably slow performance for such features as extended precision.

One possible compromise for a system that for whatever reasons cannot implement the full IEEE standard in hardware is to provide it in software. This is analogous to the bounds-checking feature of many Fortran compilers, which for efficiency reasons is usually disabled, but which can be invoked whenever one wishes to check full compliance with the Fortran standard.

One issue that has not yet been mentioned is the question of integer arithmetic. We do not see a requirement for high-performance integer arithmetic. We have found the current Cray design satisfactory in this regard, even though it lacks such operations as an integer divide. It has been our experience that if an application really requires high-performance arithmetic on whole numbers, this can be better done by assigning data to floating point variables and using the ordinary floating point operations.

CONCLUSION

The ideal floating point design for a future supercomputer would be a complete, certified implementation of the IEEE 64-bit floating point standard. However, we recognize that this attention to accuracy may have a performance cost, such as an increased CPU cycle time due to the additional logic required. Nonetheless, we are of the opinion that future systems must include some improvements over the current Cray design.

In summary, the features that we feel the most important to include in a future supercomputer floating point design are the following:

1. The 64-bit IEEE floating point format, with 11 exponent bits, 52 mantissa bits (53 including the hidden bit), and one sign bit.
2. Hardware support for reasonably fast double precision (128-bit) arithmetic.
3. Improvements in numeric accuracy over the current Cray design.
4. Provision for special overflow, underflow, and not-a-number

values, combined with a predictable trap system when programs attempt to compute with these numbers.

5. Provision for correct handling (perhaps through software) of such intrinsics as AMOD, particularly if reciprocal approximation is used in the place of a true divide operation.

Although we do not pretend to fully understand the performance tradeoffs of these features, it is our belief that these can be incorporated into a design without a significant performance sacrifice. In any event, we hope that this discussion will highlight these issues and lead to supercomputer designs that offer both high performance and improved numeric accuracy.

ACKNOWLEDGMENT

This work was performed under funding from the Numerical Aerodynamic Simulation (NAS) Systems Division at NASA Ames Research Center.

BIOGRAPHIES

David H. Bailey received a B.S. in mathematics from Brigham Young University and a Ph.D. in mathematics from Stanford University. In 1984, he joined the Numerical Aerodynamic Simulation (NAS) Systems Division at NASA Ames Research Center. Currently he serves as the technical monitor of the RIACS research group in advanced algorithms and architectures.

Horst D. Simon received his B.S. in mathematics from the TU Berlin, West Germany, in 1978, and his Ph.D. in mathematics from the University of California, Berkeley, in 1982. In 1983 he joined Boeing Computer Services as a research analyst. In 1987 he accepted an assignment for Boeing at NASA Ames Research Center, pursuing algorithm research in support of the NAS Systems Division. Jointly with colleagues at Boeing and Cray, Dr. Simon was awarded the 1988 Gordon Bell Award for practical parallel processing research, for work leading to 1.56 GFLOPS performance on a general sparse matrix factorization.

John Barton received his Ph.D. in mathematics from the University of California, Berkeley, in 1979. Since then he has worked at NASA Ames Research Center. After several years of research in computational fluid dynamics, in 1984 he joined the NAS Systems Division at Ames, where currently he is the manager of the High Speed Processor Group, with responsibility for selection of future supercomputers for NAS.

Martin Fouts was previously with the NAS Systems Division at NASA Ames, where he was responsible for research and development of interactive operating systems for supercomputers. He is currently manager of operating system development for the advanced processor division of Intergraph Corp.