Reliable Computing 2 (2) (1996), pp. 103–109

Errors in vector processing and the library libavi.a

Tiarajú A. Diverio, Úrsula A. Fernandes, and Dalcidio M. Claudio

In this paper, we describe the results of several tests that check the accuracy of numerical computation on the Cray supercomputer in vector and scalar modes. The known tests were modified to identify the critical point where roundings start causing problems. After describing the tests, we present an interval library called libavi.a. It was developed in Fortran 90 on the Cray Y–MP2E supercomputer of UFRGS–Brazil. This library makes interval mathematics accessible to the Cray supercomputers users. It works with real and complex intervals and intervals matrices and vectors. The library allows overloading of operators and functions. It is organized in four modules: real intervals, interval vectors and matrices, complex intervals, and linear algebra applications.

Погрешности при векторной обработке и библиотека libavi.a

Т. Диверио, У. Фернандес, Д. Клаудио

Проведено тестирование качества результатов нескольких численных расчетов на суперкомпьютере Cray в векторном и скалярном режимах. В тестах предусматривались изменения данных с целью определить критические точки, в которых округления приводят к проблемам при проведении вычислений. Представлена интервальная библиотека под названием libavi.a, разработанная на Фортране-90 на суперкомпьютере Cray Y–MP2E в Федеральном университете Риу-Гранди до Сул (Бразилия). Эта библиотека делает интервальную арифметику доступной пользователям компьютеров Cray. Она позволяет работать с вещественными и комплексными интервалами, а также интервальными матрицами и векторами. Допускается также совмещение идентификаторов функций и знаков операций. Библиотека состоит из четырех модулей: вещественные интервалы, интервальные векторы и матрицы, комплексные интервалы и приложения в линейной алгебре.

1. Introduction

In this paper, we describe the results of the tests that check the accuracy of numerical computations on a Cray supercomputer in vector and scalar modes. Some of these tests were originally proposed for other computers in Hammer's paper [4]. We made changes in the tests to identify the critical points where roundings start causing problems.

After describing the results of the test, we present an interval library called `libavi.a`. It was developed in Fortran 90 on the Cray Y–MP2E supercomputer of the *Universidade Federal do Rio Grande do Sul* (UFRGS–Brazil).

To analyze the numerical accuracy of the Cray supercomputer in evaluating expressions, numerical computations in scalar and vector mode were made. We used several expressions (found in papers of colleagues from Karlsruhe), for which some computations fail when they are processed in vector mode. The expressions were (sometimes modified and) processed on the Cray Y–MP2E. We then compare the results.

Situations were identified where the results of scalar mode and vector mode evaluations were different. For these expressions, we have tried to quantify the threshold values where the rounding errors start to interfere with the results. This study was done to promote the development of high accuracy high performance arithmetic on the Cray. A prototype of high performance arithmetic (`libavi.a`) was developed on a Cray Y–MP2E. It makes interval mathematics accessible to the users of the Cray supercomputer.

2. Motivation

This work was motivated by the problems that are encountered when numerical problems are solved on computers. The two major problems are: guaranteeing the *quality* of the result, and dealing with the *size* of the problem (often, an increase in problem size drastically increases the memory and time that are necessary to solve this problem):

Result quality—it refers to the result accuracy: how accurate and how reliable is the solution? It depends on how the machine stores and manipulates numerical data (internal implementation); on the stability of the numerical problem that we are solving; on error control techniques, and on several other factors.

Problem size—it refers to the size of data (e.g., number of or parameters) which must be stored (space-memory), and to the number of operations that must be run to solve the problem (time complexity measure).

3. Designed tests

We have designed several tests, and compiled and executed each test on the Cray Y–MP2E supercomputer in two modes: in scalar mode and in vector mode. The results obtained in scalar processing mode are, in general, different from the results of the vector processing mode. These differences do not result from data dependence, but from the floating-point arithmetic available on this machine. New test problems were developed [3] and non-stable problems were identified, such as long sums, cancelation of elements with opposite sign, sums of large and small values, and the computation of the real scalar product and of the interval scalar product.

All these situations were studied and the tests applied to determine the errors in the result. Several methods of minimizing the errors were tested, including the sorting of elements in long sums. (This idea is not applicable to vector computers: their main idea is to save time on computing, e.g., a scalar product $a_1b_1 + \cdots + a_nb_n$ by computing all products a_ib_i in parallel; additional sorting would require additional computation time that would eliminate the time speed-up gained by using vector computations.) In general, different modes of computation are very machine dependent, so the user who wants to improve the accuracy of the results must know how exactly elementary operations are implemented in each mode.

If the Proposal for Accurate Floating-Point Vector Arithmetic [1] was correctly implemented on the computer, we would not have these problems.

3.1. Long sums with cancelation–S_i

As a first example, consider the sums of the type $\sum_{i=N}^{-N}(16^i - 16^i) + \sum_{i=N}^{-N}(16^i - 16^i) + 1$. This sum is equal to 1. In the formula, the term 1 is added at the end, but in general, it can be

added to an arbitrary place in the sum; the index i indicates the term after which the value 1 is added. For example, S_0 means that the term "1" is added as a 0-th term (before all other terms in the sum); in the sum S_1, we add 1 after the first term (i.e., after 16^N), etc. Without 1, this sum has $4 \cdot (2N+1) = 8N+4$ terms, so, i can take $8N+5$ different values from 0 to $8N+4$:

$$\begin{aligned}
S_0 &= 1 + 16^N - 16^N + 16^{N-1} - + \cdots + 16^{-N} - 16^{-N} \\
&\quad + 16^N - 16^N + 16^{N-1} - + \cdots + 16^{-N} - 16^{-N}, \\
S_1 &= + 16^N + 1 - 16^N + 16^{N-1} - + \cdots + 16^{-N} - 16^{-N} \\
&\quad + 16^N - 16^N + 16^{N-1} - + \cdots + 16^{-N} - 16^{-N}, \\
&\cdots \\
S_{8N+4} &= + 16^N - 16^N + 16^{N-1} - + \cdots + 16^{-N} - 16^{-N} \\
&\quad + 16^N - 16^N + 16^{N-1} - + \cdots + 16^{-N} - 16^{-N} + 1.
\end{aligned}$$

Up to $N = 11$, all the values computed in scalar mode are equal to 1. For $N = 12$, the result is only equal to 1 from S_{52} on, for $N = 13$ from S_{58}, for $N = 14$ from S_{64}, and for $N = 15$ from S_{70}. These results can be explained by the mantissa size of the Cray, because when values of such different magnitudes are added, the smaller term is lost. The results start to be equal to 1 starting from some S_k (where in the example of the above table k is 154), because in this case, the value 1 is added to reasonably small values and is, therefore, not lost.

To demonstrate how the order in which the values are added influences the result, we have developed a test where positive and negative values are grouped in blocks in each of the S_i sums; for example, S_0 was re-grouped into the following sum:

$$\begin{aligned}
S_0 &= 1 + 16^N + 16^{N-1} + \cdots + 16^{-N} \\
&\quad - 16^N - 16^{N-1} - \cdots - 16^{-N} \\
&\quad + 16^N + 16^{N-1} + \cdots + 16^{-N} \\
&\quad - 16^N - 16^{N-1} - \cdots - 16^{-N}.
\end{aligned}$$

For $N = 29$, if we apply the scalar mode to thus regrouped sum, we get $S_0 = -0.314824\text{E}+21$, whereas in the previous tests (without re-grouping), we get $S_0 = 0$ in the scalar mode and $S_0 = 0.2951\text{E}+21$ in the vector mode (see Table 1).

Sum	Scalar mode	Vector mode
S0	0.000000E+00	0.2951479E+21
S23	0.000000E+00	0.1152922E+19
S119	0.000000E+00	0.0000000E+00
S122	0.000000E+00	−0.2951479E+21
S153	0.000000E+00	0.1000000E+01
S154	0.100000E+01	0.1000000E+01
S236	0.100000E+01	0.2951479E+21

Table 1. Results for $N = 29$

4. Library of interval routines–libavi.a

The library of interval routines called `libavi.a` was developed in Fortran 90. The name of this library means: `lib`—library and, `avi`—interval vector arithmetic; the ending `.a` is the library standard. It made interval mathematics accessible to the Cray Y–MP2E supercomputer users. It works with real and complex intervals and with interval matrices and vectors. It was designed to make possible the application of interval mathematics in supercomputers. 289 routines were developed in Fortran 90 (approximately 4900 lines of code).

The library allows the overloading of operators and functions. Functions such as sum, subtraction, multiplication, division and the relational symbols are overloaded, thus making programming easier.

The library `libavi.a` consists of four modules: `BASIC` module—real intervals; `MVI` module—interval vectors and matrices; `CI` module—complex intervals; `APLIC` module—linear algebra applications. The modules are described below.

`BASIC` **module—real intervals** (52 routines). Transformation functions (functions that transform two reals into an interval, and that make it possible to access the bounds of intervals), relational operations, operations with sets (namely, the union and intersection of intervals), arithmetic operations, basic functions, and input/output routines.

`CI` **module—complex intervals** (58 routines). Transformation functions, relational operations, operations with sets, arithmetic operations, elementary functions for complex intervals, and input/output routines.

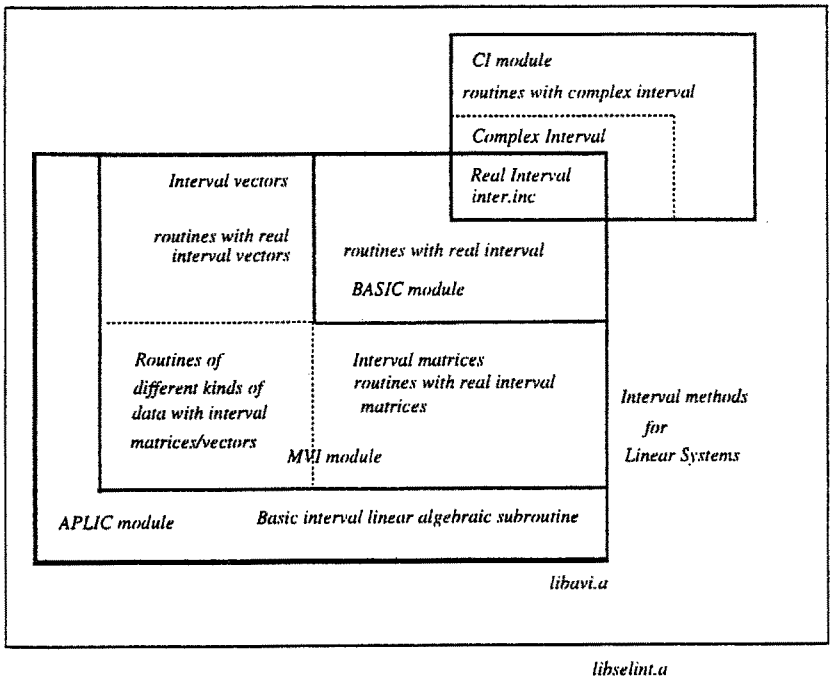

Figure 1. Hierarchy

MVI **module—interval vectors and matrices** (150 routines). Operations with real interval matrices, real interval vectors, and real interval matrices/vectors. Transformation functions, relational operations, operations with sets, elementary basic functions, predefined functions, and input/output routines. **Operations with different types of data.** Operations in which one of the operands is a real number, and the second operand is an interval; operations with real and interval vectors, operations with real and interval matrices, operations with intervals and interval vectors, and operations with intervals and interval matrices. For data of varying type, the scalar product is implemented using controlled rounding in the addition and in the subtraction (these controlled roundings are implemented in the basic module).

APLIC **module—Basic interval linear algebraic applications** (29 routines). The routines of the APLIC module are interval versions of standard linear algebra operations with vectors and matrices.

Some of these routines are:

Svsaxpy—Sum of a scalar multiple of an interval vector and of another interval vector.

Sgemv and sdgemv—These routines implement the interval residual calculus in single and double precision, using the BLAS routines with the values rounded up and down to generate the lower and the upper bound of the resulting interval.

Sgemm and sdgemm—These routines compute the result of the interval matrix operation $D = sC + rAB$ in single and double precision, using the BLAS routines.

Svdot and svddot—These routines are the interval versions of the scalar product available in the libavi.a. They use the BLAS routines for both bounds of the interval. They were compared to the routines of the *mvi* module to the interval scalar product.

Svmult—This routine has the best accuracy, because it has a built-in way to control the rounding error.

Other routines available in the APLIC module include routines for computing vector and matrix norms and condition numbers, interval Hilbert matrices, and inverse matrices.

The APLIC module was developed to show how interval computations can be used for linear algebra. The routines comprising this module help the users in solving their application problems. At present, further interval applications routines are being developed, including the routines for solving systems of linear equations and for numerical integration.

5. Conclusions and future work

Computation inaccuracy results from: the order in which the values are processed; the way operations with floating-point real numbers are implemented on the Cray; and on the representation of the data. To make Cray's numerical results more accurate (and thus, more reliable), we designed and implemented an interval library.

The first phase of this project consisted of the implementation of the basic interval arithmetic, which was composed of the four modules described in this paper.

The planned project has three further phases for the implementation of high accuracy and high performance arithmetic on the Cray Y–MP2E supercomputer; these phases are currently under development at the Computational Mathematics Group of UFRGS. They are:

Incorporation of high-precision arithmetic into the library libavi.a. On the Cray, the floating-point numbers are *not* represented according to the IEEE Standard for Binary Floating-Point Arithmetic (IEEE 754); therefore, we had to simulate directed rounding by

using special Fortran 90 functions which compute the downward and upward nearest numbers. This implementation does not exactly follow the definition of directed rounding (because it sometimes introduces an additional error), but it guarantees that the result is in the resulting interval.

The second part of this phase is computing the scalar product with high precision. We implemented and compared five routines that produce an interval containing the desired scalar product. The first two routines, `svdot` and `sddot` are implemented using the BLAS routines to compute the scalar product in single and double precision, respectively; direct roundings are then used to convert the results of these routines into intervals that contain the exact value. Routines `svmult` and `svdmult` compute an interval that contains the scalar product of two real vectors by multiplying their components, which are then accumulated; after these computations, the downward and upward directed roundings are applied. The fifth routine computes the scalar product of two interval vectors.

The two routines that used the BLAS library produced the worst results; this was expected, because in the multiplication and addition operations, the rounding errors were not controlled.

To incorporate these high precision routines into the library `libavi.a`, we must incorporate the following operations in software: downward and upward directed rounding, the basic four arithmetic operations with maximum accuracy, and the dot product with only one rounding.

Optimization, vectorization and performance analysis. The available Fortran compiler had its limitations of optimization and vectorization. Only now the `libavi.a` routines are being optimized and the vectorizable characteristics of the operations with vectors and matrices of intervals have been explored. We are currently trying to find the adequate performance measures and benchmark problems, so that we will be able to make meaningful comparisons of different algorithms and implementations. This activity constitutes the third phase of our project.

Development of applied interval libraries. One of the objectives of the design of the APLIC module was to enable the users to get guaranteed (interval) estimates for computations from linear algebra. To enable the users to solve more complicated applied problems, we are currently working on interval routines for solving algebraic equations, systems of linear equations, for numerical integration, and for several other numerical problems. (The resulting routines will automatically verify the results.) This activity corresponds to the fourth phase of our project.

All these three activities are being developed now. Several characteristics of the library `libavi.a` make our work easier:

Accuracy. In `libavi.a`, many routines are implemented using double precision; several BLAS routines (known to be of good accuracy) are also used.

Efficiency. The results of `libavi.a` are not only computed with guaranteed accuracy, but they are also computed with a reasonable speed.

Easy to use. Function identifiers and operator overloading enable the programmers to use standard mathematical notation (e.g., + to describe the sum of matrices, vectors, intervals, etc.). The description of interval matrices and interval vectors is based on dynamical arrays, so, a programmer can easily change the size of the matrix (vector) in the course of computations, without defining a new matrix. This feature is useful for linear algebra and thus, simplifies the use of the library `libavi.a`.

Optimized and vectorized routines. The routines from the library are written in such a

way that the compiler is able to automatically optimize and vectorize not only these routines, but also the programs that use these routines.

Modularity. The library consists of several reasonably independent modules. Each module contains operations with data of a certain interval data type.

Our ultimate goal is to combine Interval Arithmetic (as implemented in `libavi.a`) with High Performance Computing (i.e., methods that use parallelism to speed up computations); we call the desired combination High Performance Arithmetic. In particular, to achieve this goal, we are currently working on the best way to implement the optimal scalar product [1].

The `libavi.a` library was designed to produce results comparable in quality with Pascal–XSC [5, 6]; with this objective in mind, we compared the results of our library with the results obtained by using Pascal–XSC.

Additional information about the `libavi.a` library can be found in the library manuals, in T. A. Diverio's Ph.D. Thesis [2], and in the references therein.

References

[1] Bohlender, G., Cordes, D., Knöfel, A., Kulisch, U., Lohner, R., and Walter, W. V. *Proposal for accurate floating-point vector arithmetic.* In: Adams, E. and Kulisch, U. (eds) "Scientific Computing with Automatic Result Verification", Academic Press, Orlando, 1992, pp. 87–102.

[2] Diverio, T. A. *Uso efetivo da matemática intervalar em supercomputadores vetoriais.* Ph.D. thesis, Porto Alegre: CPGCC–UFRGS, 1995.

[3] Fernandes, U. A. L. and Diverio, T. A. *Fallas en los calculos utilizando procesamiento vetorial.* In: "CITA 95, Assuncion, 7–11, August 1995".

[4] Hammer, R. *How reliable is the arithmetic of vector computers?* In: Ullrich, Ch. (ed.) "Contributions to Computer Arithmetic and Self-Validating Numerical Methods", IMACS Annals on Computing and Applied Mathematics 7, J.C. Baltzer, 1990, pp. 467–482.

[5] Hammer, R., Hocks, M., Kulisch, U., and Ratz, D. *Numerical toolbox for verified computing I: basic numerical problems.* Springer-Verlag, Berlin, 1992.

[6] Klatte, R., Kulisch, U., Neaga, M., Ratz, D., and Ullrich, Ch. *Pascal–XSC. Language reference with examples.* Berlin, Springer-Verlag, 1992.

Received: October 26, 1995
Revised version: January 10, 1996

Universidade Federal do Rio Grande do Sul
Instituto de Informática
P.O. BOX 15064, 91501–970, Porto Alegre-RS
Brasil
E-mail: {diverio, ursula, dalcidio}@inf.ufrgs.br