Reliable Computing, 5, 311?? (1999)
© 1999 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

The Interval-Enhanced GNU Fortran Compiler

MICHAEL J. SCHULTE, VITALY ZELOV, AND AHMET AKKAS mschulte@eecs.lehigh.edu
EECS Dept., Lehigh University, Bethlehem, PA 18015, USA

JAMES CRAIG BURLEY burley@gnu.org
Free Software Foundation, 59 Temple Place, Suite 330, Boston, MA 02111, USA

Received October 31, 1998; Revised March 1, 1999

Editor: Tibor Csendes

Abstract. Compiler support for intervals as intrinsic data types is essential for promoting the development and wide-spread use of interval software. It also plays an important role in encouraging the development of hardware support for interval arithmetic. This paper describes modifications made to the GNU Fortran Compiler to provide support for interval arithmetic. These modifications are based on a recently proposed Fortran 77 Interval Arithmetic Specification, which provides a standard for supporting interval arithmetic in Fortran. This paper also describes the design of the compiler's interval runtime libraries and the methodology used to test the compiler. The compiler and runtime libraries are designed to be portable to platforms that support the IEEE 754 floating point standard.

Keywords: Fortran, interval arithmetic, compiler, runtime libraries, validate, containment.

1. Introduction

Interval arithmetic provides an efficient method for performing operations on intervals of real numbers [1]. With interval arithmetic, each interval is represented by its lower and upper endpoints. More formally, an interval $X \equiv [\underline{x}, \overline{x}]$ is defined as

$$X \equiv [\underline{x}, \overline{x}] \equiv \{x \in \mathbb{R} | \underline{x} \leq x \leq \overline{x}\}$$

For example, the interval $X = [1.23, 1.24]$ is a closed interval that includes all values greater than or equal to 1.23 and less than or equal to 1.24.

Although the concept of interval arithmetic is relatively straightforward, it provides a powerful mechanism for bounding the results of floating point computations. As demonstrated in [1] and [2], interval arithmetic provides a practical method for bounding errors in numerical computations including roundoff errors, approximation errors, and errors due to inexact inputs. In addition, interval arithmetic has been used to develop validated algorithms for solving problems in several areas. For example, interval arithmetic has been used to provide validated solutions in global optimization, finding roots of functions, solving systems of linear and nonlinear equations, performing numerical differentiation and integration, and solving systems of ordinary and partial differential equations. These and other applications for interval arithmetic are presented in [2], [3], [4], [5], [6].

Because of its ability to provide validated solutions, several software tools have been developed to support interval arithmetic. These include interval arithmetic libraries, [7], [8], [9], language extensions with support for interval arithmetic [10], [11], [12], and interval application software [4], [13], [14]. Although these tools give programmers access to interval arithmetic, they do not conform to a specified standard.

As noted in [15], the lack of a specified standard for interval arithmetic has the following disadvantages:

- Resources are unnecessarily expended to redevelop tools for interval arithmetic.
- Diverse semantics in programming interval computations inhibit multi-person development of interval application software.
- Interval arithmetic packages written in high-level languages seldom take advantage of machine hardware, which may lead to poor runtime performance.
- Interval arithmetic packages designed on one platform may not be portable to other platforms.

Furthermore, current interval arithmetic software packages do not always guarantee containment. As noted in [16], several packages for interval arithmetic produce incorrect results due to incorrect handling of arithmetic exceptions. Some of these packages also depend on the underlying platform's built-in libraries when performing interval mathematical functions and interval input and output. Consequently, the correctness and tightness of the interval routines are dependent on the accuracy of the built-in libraries. This accuracy is often difficult to determine and can vary on different platforms [17]. Furthermore, previous software packages for interval arithmetic sometimes cause unexpected behavior. For example, with certain interval arithmetic libraries, the machine floating point rounding mode can be changed as a side-effect of performing interval arithmetic operations.

To overcome these difficulties, a standard for supporting interval arithmetic in Fortran was recently proposed [15]. This proposal specifies support for interval data types, interval arithmetic operations, interval relations, interval versions of mathematical functions, and interval I/O. The Fortran 77 Interval Arithmetic Specification builds upon [15] to provide a complete standard for supporting intervals in Fortran [18]. It adds several new operators and intrinsic functions, which are defined in [20]. It also defines a set of extended real intervals and their internal representation for IEEE 754 compliant processors. The set of extended real intervals is closed with respect to interval arithmetic operations and interval enclosures of real functions [19]. The Fortran 77 Interval Arithmetic Specification also gives algorithms for correct nonstop (i.e., without interrupts for arithmetic exceptions) handling of several interval intrinsics and operations, discusses interval expression optimization and mixed-mode evaluation, and gives several examples to illustrate the correct implementation of the specification.

To ease the burden of writing interval code and to help provide a standard for interval arithmetic, the GNU Fortran Compiler has been enhanced to support intervals as an intrinsic data type. The interval-enhanced GNU Fortran Compiler provides support for interval data types, constants, arithmetic operations, set operations, relations, special functions, mathematical functions, conversion functions, and I/O.

Support for interval arithmetic in the GNU Fortran Compiler is based on the Fortran 77 Interval Arithmetic Specification [18]. To support interval arithmetic on the GNU Fortran Compiler, the front-end of the compiler was modified and an extensive interval runtime library was developed. The interval-enhanced compiler and the routines in the interval runtime library guarantee containment, produce relatively sharp intervals, and are designed to be portable to platforms that support the IEEE 754 floating point standard. The arithmetic operations, interval special functions, and interval I/O produce minimum width intervals. Interval versions of mathematical functions can produce wider intervals.

The contents of the remainder of this paper is as follows. Section 2 gives an overview of the GNU Fortran Compiler. Section 3 describes the Fortran 77 Interval Arithmetic Specification and its implementation on the interval-enhanced GNU Fortran compiler. Section 4 discusses changes made to the compiler to provide support for interval arithmetic. Section 5 describes the interval runtime libraries that are used by the GNU Fortran compiler to support interval arithmetic. Section 6 discusses the method used for testing the compiler and the routines in the runtime library. Section 7 gives conclusions. A preliminary version of the interval-enhanced compiler is available from http://www.eecs.lehigh.edu/˜mschulte/compiler/code.

2. The GNU Fortran Compiler

This section gives an overview of the GNU Fortran Compiler. The GNU Fortran Compiler, also known as g77, is a publically available Fortran compiler that is designed to run on a variety of platforms. To promote software reuse, it interfaces with other GNU compilers and tools, such as the GNU C compiler (gcc), the GNU C++ compiler (g++), and the GNU debugger (gdb).

Similar to other Fortran compilers, the g77 compiler translates Fortran programs to machine code. To perform this translation, the g77 compiler

1. Reads a program written in Fortran
2. Checks the program for errors
3. Translates the Fortran program to an intermediate form, which includes calls to the runtime libraries
4. Runs the GNU code generator, which performs optimizations and converts the intermediate form to assembly language
5. Runs the GNU assembler, which converts the assembly language to machine code

The g77 compiler also has support for debugging, program linking, and error and warning diagnostics. Command line arguments can be used to specify various options including output control, Fortran dialect, warning messages, debugging support, optimization levels, preprocessing, directory locations, code generation, and environment variables. The GNU Fortran Compiler utilizes four main components: a modified version of the gcc compiler, the g77 interface, the libf2c runtime libraries, and the Fortran f771 compiler.

The gcc compiler is more than just a compiler for C. Based on command-line options and the names given for files on the command line, gcc determines which actions to perform, including preprocessing, compiling, assembling, and linking. In a GNU Fortran installation, gcc recognizes Fortran source files by names ending in .f or .F For these files, it uses the f771 Fortran compiler to perform compilation.

The g77 interface is essentially just a front-end for the gcc compiler. Because of this, g77 can compile and link programs and source files written in other languages, such as C and C++. Fortran programmers will normally use g77 instead of gcc, because g77 knows how to specify the libraries that need to be linked with Fortran programs. Two libraries that are automatically linked are the Fortran library, libf2c, and the math library, lm.

The libf2c runtime library contains machine code needed to support capabilities of the Fortran language that are not directly provided by the machine code generated by the g77 compilation phase. This library includes procedures needed by Fortran programs while they are running. For example, while machine code generated by g77 performs additions, subtractions, and multiplications, it does not perform I/O or compute the trigonometric functions. Instead, Fortran statements that perform these operations are converted by the f771 compiler into function calls in the machine code. When run, the function calls in the machine code invoke functions in the libf2c runtime library.

The f771 compiler is a combination of two rather large pieces of code. One piece is called the GNU Back End (GBE), which knows how to generate fast code for a wide variety of processors. The same GBE is used by the C, C++, and Fortran compiler programs. The GBE also generates some warnings, such as those for references to undefined variables. A typical distribution of g77 contains patch files for the GBE that allow it to process Fortran code correctly and efficiently. The other piece of f771, called the Fortran Front End (FFE), is the majority of what is unique about GNU Fortran. The FFE knows how to interpret Fortran programs to determine what they are intending to do, and then communicate this knowledge to the GBE for the actual compilation of the programs. The FFE is responsible for diagnosing incorrect usage of the language in the programs it processes, and is also responsible for generating most of the warnings about questionable constructs.

3. Interval Arithmetic Enhancements to the GNU Fortran Compiler

The interval-enhanced version of the g77 compiler, referred to as g77-i, is based on the Fortran 77 Interval Arithmetic Specification. This section gives a summary of the interval-enhancements implemented in the g77-i compiler. Further details can be found in the Fortran 77 Interval Arithmetic Specification [18].

The g77-i compiler supports the set of extended real intervals defined in [18] and [19]. The set of extended real intervals includes the set of real intervals (i.e. intervals of the form $X \equiv [\underline{x}, \overline{x}]$, where $\underline{x}$ and $\overline{x}$ are real numbers and $\underline{x} \leq \overline{x}$), intervals with infinite endpoints, and the empty interval. The empty interval is an interval that contains no real numbers, and can be produced by inputting an empty interval, intersecting disjoint intervals, or evaluating an interval function strictly outside of its domain of definition. Results from [19] and [20] show that the set of extended real intervals is closed with respect to the interval arithmetic operations and interval enclosures of real functions.

The g77-i compiler represents intervals using their endpoints, which are either single precision or double precision IEEE 754 floating point numbers [21]. Empty intervals are represented by using a quiet not-a-number (NaN) for each interval endpoint. Double precision intervals, which are the default, are defined using either INTERVAL or INTERVAL*16. Single precision intervals are defined using INTERVAL*8. The number after the '*' corresponds to the total number of bytes used for both interval endpoints.

Interval constants may be specified using one of two formats: $[x]$ or $[\underline{x}, \overline{x}]$, where x, $\underline{x}$, and $\overline{x}$ are real or integer constants. An interval constant that uses the $[x]$ format is the same as the interval constant $[x, x]$. For example, the interval constants [1.23] and [1.23, 1.23] are equivalent. When an interval constant is converted, the internal interval contains the interval constant, regardless of the value of or number of digits in either interval endpoint. Invalid interval constants, which have a lower interval endpoint that is greater than the upper interval endpoint, result in a compile time error. When converting interval constants, the g77-i compiler always produces the minimum-width intervals that guarantee containment.

The interval arithmetic operations supported by g77-i include addition, subtraction, multiplication, and division. If either operand is the empty interval then the result is the empty interval. Unlike some interval arithmetic libraries, which can change the rounding mode as a side-effect of performing interval arithmetic operations, the interval arithmetic operations in g77-i save and then later restore the rounding mode. They also produce minimum-width intervals that guarantee containment. There are also two versions of the interval power operator: one computes the interval power of an interval base and the other computes the integer power of an interval base.

The interval set intrinsics include ISEMTPY, which returns TRUE if an interval is empty, and INF and SUP, which return the infimum and supremum of an interval, respectively. The interval set operations include .IX., which returns the intersection of two intervals, and .IH., which returns the hull of two intervals (i.e., the smallest interval that encloses both intervals). The intersection of two disjoint intervals returns the empty interval. There are also several special interval functions, which include MID, WID, MIG, MAG, ABS, MAX, MIN, NDIGITS, and INT. Definitions for each of these special functions are given in the Fortran 77 Interval Arithmetic Specification [18].

Several interval relation operations, which return logical values of TRUE or FALSE, are also supported by g77-i. These relation operations are classified as set relations, certainly relations, or possibly relations. The set relations treat intervals as sets of real numbers. The certainly relations are true if and only if the relation is true for every value in both intervals. The possibly relations are true if the relation is true for any value in both intervals.

For each real mathematical function in Fortran, a corresponding interval mathematical function is defined. Results from [19] and [20] show that the interval mathematical functions are closed with respect to the extended interval system. This is accomplished by intersecting the interval input arguments with the domain over which the function is defined. For example, SQRT($[-2, 4]$) returns $[0, 2]$ and SQRT($[-4, -2]$) returns the empty interval. If any input argument to an interval mathematical function is the empty interval, then the result is the empty interval.

Three interval conversion functions, INTERVAL, DINTERVAL, and SINTERVAL, are defined to provide explicit conversion to interval types from integer, real, and interval types. These functions typically take either one or two integer or real arguments and produce an interval result. They can also take one interval argument and produce an interval result. The INTERVAL and DINTERVAL functions both produce double precision intervals, while the SINTERVAL function produces single precision intervals.

The g77-i compiler also supports formatted and list-directed interval I/O, as described in the Fortran 77 Interval Arithmetic Specification. With formatted interval I/O, the user specifies the total field width, w, the number of significant digits, d, and (optionally) the number of exponent digits e. The VE, VF, and VG edit descriptors support standard formatted interval I/O. These descriptors use Fortran's F, E, and G edit descriptors for the interval endpoints, while ensuring that the interval result is the sharpest interval that guarantees containment. The Y edit descriptor supports formatted single number interval I/O [22]. With single number interval I/O, intervals are input and output using a single decimal number. List-directed interval I/O allows intervals to be input and output without specifying the format. If the intervals are read from or written to a binary file, list-directed interval I/O avoids roundoff errors due to conversions. If intervals are read from or written to a text file, the compiler chooses suitable values for the output format, and roundoff errors due to conversions may occur.

Much of the syntax used in g77-i is similar to the ACRITH-XSC extensions to Fortran [11]. ACRITH-XSC is actually more powerful than g77-i because it provides additional features, such as dynamic arrays, operator overloading, dot product expressions, accurate vector and matrix computations, and complex intervals. The main benefits of g77-i are that it is publically available, it is portable to a wide range of platforms, it can be used in conjunction with other GNU compilers and tools, and it conforms to the Fortran 77 Interval Arithmetic Specification.

4. Modifying the GNU Fortran Compiler to Support Intervals

To support interval arithmetic, several modifications were made to g77. These modifications provide support for the new interval data types, interval constants, interval I/O and format statements, interval operations, and interval intrinsics.

For the new interval data types, diagnostic messages were added, internal Fortran front-end (FFE) representations of the types were created, routines for the internal GNU back-end (GBE) translations of the types were developed, and initializations of internal FFE and GBE interval type information were provided. Also, code to recognize the INTERVAL keyword as a statement or modifier and code to assign internal interval type information based on parsed keywords was added.

To support interval constants, diagnostic messages for interval constants, the ability to recognize '[' and ']' as lexemes, and code to parse such constants were added. Also, the internal representations and a new expression-parsing context for interval constants were developed. In addition, routines to build intermediate representation entries and to translate FFE interval constants to their GBE form were designed. There were also several features added to support interval constant arrays including the ability to extract an element of a constant array, create new constant arrays, copy between constant arrays, copy between a constant array and a constant scalar, and put a value in an element of a constant array.

To support interval I/O, new diagnostic messages for the interval format specifiers were added, and code to parse the format specifiers and to convert the format specifiers to their runtime form was developed. It was also necessary to provide data type information to the I/O runtime library to allow the I/O routines to distinguish between real, complex, and interval data.

To support interval arithmetic operations, new runtime routines were defined, and the code that determines whether the operands are unary or binary operators was modified to support intervals. Also, code that translates the FFE representations of such operations to their GBE counterparts was added, and the ability to define runtime routines as operating on or returning interval values was provided. These modifications allow the FFE to translate interval operations in Fortran to calls to the interval runtime library.

For interval intrinsics, the specifications of several of the intrinsics' interfaces were modified to allow intervals. Code to compile intrinsic operations to appropriate GBE representations was also added. Additionally, the ability to define an intrinsic as operating on, or returning, interval values was provided. Similar to the interval arithmetic operations, the interval intrinsics are implemented by converting the Fortran intrinsics to calls to the interval runtime library.

5. Interval Runtime Libraries

Much of the work to add intervals to g77 focused on developing efficient interval runtime library routines. These routines were added to the existing g77 runtime libraries. The g77 runtime directory is composed of three sub-directories: libF77, libI77, and libU77. The libF77 directory contains the non-I/0 support routines needed by the GNU Fortran Compiler, such as routines for mathematical and string functions. The libI77 directory contains support routines for Fortran I/O and file management. The libU77 directory contains support routines for UNIX system function calls. All g77 runtime library routines are written in C.

To incorporate the routines for interval arithmetic into the existing g77 runtime libraries, the non-I/O routines were added to the libF77 directory and the I/O routines were added to the libI77 directory. Support routines for changing rounding modes, using floating point numbers, and initializing intervals are accessible from both of these directories via symbolic links. Names for the interval routines in the runtime libraries are derived from the names given in the Fortran 77 Interval

Arithmetic Specification [18], and function definitions are designed to be similar to those currently used in the runtime libraries.

The routines in the runtime libraries are designed to support IEEE single and double precision intervals in either big endian or little endian format. They also are designed to run on several different platforms including Sun SPARC, DEC Alpha, SGI MIPS, IBM RS/6000, Intel x86 and Motorola 68k. So far, they have been tested primarily on Sun Ultras running Solaris and Intel Pentiums running Linux. Later, they will be ported to a wider range of platforms. To provide tight intervals, the interval arithmetic routines use the underlying hardware's built-in IEEE rounding modes to produce minimum-width intervals. Changing the rounding modes is performed through platform specific function calls.

Header files and support routines were developed to assist in adding interval arithmetic to the GNU Fortran Compiler. These files include definitions for the interval data types (interval.h), definitions for various floating point values and parameters (fp.h), and support for directed rounding (round.h). Definitions and routines defined in these files are used internally by the other routines in the interval runtime libraries.

The non-I/O interval routines include those that support interval arithmetic operations, set operations, relations, and special and mathematical functions. There are two versions of each routine: one for single precision intervals and a second for double precision intervals. In the function definitions, the names of the single precision routines are prefixed with a *u* and the names of the double precision routines are prefixed with a *v*. For example, single and double precision interval addition routines are called u_add and v_add, respectively.

As an example of the interval arithmetic routines, code for the v_add subroutine is shown in Figure 1. The interval addition routine takes as input arguments pointers to the intervals, a and b, and outputs the interval resx. Initially, the current IEEE rounding mode is saved and the rounding mode is set to round towards plus infinity. Next, the interval endpoints of the sum are computed. At the end of the routine, the rounding mode is restored to its original value and the result is returned.

To reduce the number of rounding mode changes, sign symmetry is employed when performing interval arithmetic operations [16]. With this technique, both interval endpoints are computed with the rounding mode set to round towards positive infinity. To compute the lower interval endpoint, the additive inverse of the lower interval endpoint is computed and then negated. The upper interval endpoint is computed in the usual fashion. On most machines, in which changing the rounding mode takes several cycles and stalls the floating point pipeline, the use of sign symmetry results in an overall performance improvement.

Routines for the interval mathematical functions are based on the Fast Interval Library [23]. This library is written in ANSI-C and uses fast table look-up algorithms to evaluate the interval mathematical functions. To comply with the Fortran 77 Interval Arithmetic Specification, these routine were modified to support empty intervals, intervals with infinite endpoints, and intervals over which the function contains singularities.

```
#include "round.h"
#include "interval.h"
interval v_add(interval *a, interval *b) {
   interval resx;                 /* Interval sum */
   RND_MODE_TYPE round_mode;      /* Variable for IEEE rounding mode */
   round_mode = GET_RND_SET_UP(); /* Save rounding mode and set to round up*/
   resx.inf = -(-a->inf - b->inf); /* Compute lower endpoint of interval */
   resx.sup = a->sup + b->sup;    /* Compute upper endpoint of interval */
   SET_ROUND(round_mode);         /* Restore the IEEE rounding mode */
   return resx;                   /* Return the result */
}
```

Figure 1. Runtime Routine for Interval Addition

The interval I/O routines are designed to interface directly with the existing I/O routines in the libI77 runtime directory. This allows them to reuse much of the code that is already available for I/O of real and complex numbers in Fortran. The routines developed for interval I/O include several interval Fortran specific routines that are designed for providing interval I/O in the VF, VE, VG, and Y formats. In addition, there are a number of support routines for performing conversion between character strings and intervals. The support routines were developed based on routines for floating point input [24] and floating point output [25]. These routines were first modified to provide correctly rounded input and output, and then used to provide proper input and output of unformatted and formatted intervals.

6. Testing the Runtime Libraries and Compiler

One of the benefits of interval arithmetic algorithms is that they supply validated solutions. Therefore, it is extremely important that the underlying software used to perform the interval algorithms is reliable. To help ensure this, extensive testing of the interval compiler and runtime libraries has been performed. For each interval operation or intrinsic, interactive tests, special case tests, identity tests and random tests were performed.

Interactive tests are typically run on the routines just after they are developed. Interactive test routines allow the user to specify the input values to the interval routine. The test routine reads in the values, performs the specified function, and outputs the result. Interactive test routines are designed to catch more obvious bugs and to give the user the ability to fairly quickly test potential problem cases.

After performing interactive tests, special cases are tested to ensure that they produce correct results. This is typically done by reading input intervals from one file, writing the results to a second file, and then performing manual checks to ensure that correct results are produced. Examples of test cases include testing division by zero, multiplying very large or very small intervals together, taking the midpoint of an infinite interval, and taking the intersection of disjoint intervals.

Naturally, the types of special case tests performed depend on the function being tested.

After ensuring that special cases work correctly, a large number of identity tests are run. Examples of identity tests include $X + Y = Y + X$, $X + X = 2 * X$, and $X - Y = X + -(Y)$. Several identity tests were supplied by Michael Parks of Sun Microsystems. For many of the identities, outward rounding prevents identical results from being produced. In these cases, it is often required to allow the results to differ by a unit in the last place and handle cases which lead to overflow or underflow separately. For example, $(X + Y) + Z = X + (Y + Z)$ does not always produce identical results for the left and right hand sides of the equation. Identity tests are run for both random inputs and values input from a user-specified file.

After running identity tests, a large number of random tests are performed. The random test routines are designed to test the algorithm for a very large number of pseudo-random inputs. To determine if the correct results are produced, results from these routines are compared with results from other interval software packages.

After running large numbers of random tests, the g77-i compiler was used to develop complete interval programs. Examples of interval programs that have been developed using g77-i include programs for global optimization, solving systems of linear equations, finding roots of functions, and performing vector and matrix computations. The g77-i compiler has also been used to compile interval code that is used by the GlobSol software package [26]. Replacing calls to the intlibf90 interval software package by calls to code compiled using g77-i resulted in a significant performance improvement and sharper interval results.

7. Conclusions

A preliminary version of the g77-i compiler has been completed and tested. This version of the compiler satisfies the requirements given in the Fortran 77 Interval Arithmetic Specification and provides an efficient tool for developing interval software in Fortran. A fast interval runtime library is used to improve the speed of interval code while guaranteeing containment and providing sharp rounding of the interval endpoints. The g77-i compiler has been extensively tested and has been used to develop several interval application programs. Similar interval-enhancements are being made to the GNU C and C++ compilers, and to versions of Sun Microsystem's Fortran compilers.

Acknowledgments

The authors are thankful to G. William Walster and Dimitry Chiriaev for their comments and advice on the design of the interval-enhanced GNU Fortran compiler. They are also grateful to Sun Microsystems for funding this project, and to the anonymous reviewers for their valuable comments.

References

1. R. E. Moore, *Interval Analysis.* Prentice Hall, 1966.
2. G. Alefeld and J. Herzberger, *Introduction to Interval Computations.* Academic Press, 1983.
3. R. E. Moore, *Methods and Applications of Interval Analysis, SIAM Studies in Applied Mathematics*, SIAM, 1979.
4. R. Hammer, M. Hocks, U. Kulisch, and D. Ratz, *C++ Toolbox for Verified Computing.* Springer-Verlag, 1995.
5. G. F. Corliss, "Industrial Applications of Interval Techniques," in *Computer Arithmetic and Self-Validating Numerical Methods* (C. Ullrich, ed.), Academic Press, pp. 91–113, 1990.
6. E. Hansen, *Global Optimization Using Interval Analysis.* Marcel Dekker, 1992.
7. R. B. Kearfott *et al.*, "Algorithm 737: INTLIB: A Portable FORTRAN 77 Interval Standard Function Library," *ACM Transactions on Mathematical Software*, vol. 20, pp. 447–459, 1994.
8. O. Knüppel, "PROFIL/BIAS - A Fast Interval Library," *Computing*, vol. 53, pp. 277–288, 1994.
9. R. B. Kearfott, "A FORTRAN 90 Environment for Research and Prototyping of Enclosure Algorithms for Nonlinear Equations and Global Optimization," *ACM Transactions on Mathematical Software*, vol. 21, no. 1, pp. 63–78, March, 1995.
10. R. Klatte *et al.*, *C-XSC: A C++ Class Library for Extended Scientific Computing.* Springer-Verlag, 1993.
11. W. V. Walter, "ACRITH-XSC: A Fortran-like Language for Verified Scientific Computing," in *Scientific Computing with Automatic Result Verification* (E. Adams and U. Kulisch, eds.), Academic Press, pp. 45– 70, 1993.
12. J. S. Ely, "The VPI Software Package for Variable Precision Interval Arithmetic," *Interval Computations*, vol. 2, pp. 135–153, 1993.
13. R. B. Kearfott and M. Novoa, "INTBIS, A Portable Interval Newton Bisection Package," *ACM Transactions on Mathematical Software*, vol. 16, pp. 152–157, 1990.
14. R. B. Kearfott, *Rigerous Global Search: Continuous Problems*, Kluwer Academic Publishers, Dordchet, 1997.
15. R. B. Kearfott *et al. A Specific Proposal for Interval Arithmetic in Fortran*, 1996. Available at http://interval.usl.edu/F90/f96-pro.asc.
16. D. Priest, *Handling IEEE 754 Invalid Operation Exceptions in Real Interval Arithmetic*, Manuscript, 1997.
17. D. Priest, "Fast Table-Driven Algorithms for Interval Elementary Functions," *Proceedings of the 13th Symposium on Computer Arithmetic*, pp. 168-174, July, 1997.
18. D. Chiriaev and G. W. Walster, *Fortran 77 Interval Arithmetic Specification*, 1997. Available at http://www.mscs.mu.edu/˜globsol/Papers/spec.ps.
19. G. W. Walster, "The Extended Real Interval System," 1997. Available at http://www.mscs.mu.edu/˜globsol/Papers/extended_intervals.ps.
20. G. W. Walster and E. R. Hansen, *Interval Algebra, Composite Functions and Dependence in Compilers*, submitted to *Reliable Computing*, 1998. Available at http://www.mscs.mu.edu/˜globsol/Papers/composite.ps.
21. ANSI/IEEE 754-1985 Standard for Binary Floating-Point Arithmetic, Institute of Electrical and Electronics Engineers, New York, 1985.
22. M. J. Schulte, V. A. Zelov, G. W. Walster, and D. Chiriaev, "Single-Number Interval I/O," *Proceedings of the International Symposium on Scientific Computing, Computer Arithmetic, and Validated Numerics* (T. Csendes ed.), Budapest, Hungary, September, 1998.
23. W. Hofschuster and W. Kraemer, "A Fast Interval Library," 1997. Available at ftp://iamk4515.mathematik.uni-karlsruhe.de/pub/iwrmm/software.
24. D. M. Gay, "Correctly Rounded Binary-Decimal and Decimal-Binary Conversions," Numerical Analysis Manuscript 90-10, AT&T Bell Laboratories, 1990.
25. R. G. Burger and R. K. Dybvig, "Printing Floating Point Numbers Quickly and Accurately," *Sigplan Notices*, vol. 31, no. 5, pp. 108-116, 1996.
26. G. F. Corliss, "Rigerous Global Search: Industrial Applications," *Proceedings of the International Symposium on Scientific Computing, Computer Arithmetic, and Validated Numerics*, (T. Csendes ed.), Budapest, Hungary, September, 1998.