Modifications to Expression Evaluation in C-XSC

Gerd Bohlender, Mariana Lüderitz Kolberg, and
Dalcidio Moraes Claudio

Preprint 2005/5

Wissenschaftliches Rechnen/
Softwaretechnologie

Impressum

Herausgeber:	Prof. Dr. W. Krämer, Dr. W. Hofschuster
	Wissenschaftliches Rechnen/Softwaretechnologie
	Fachbereich C (Mathematik und Naturwissenschaften)
	Bergische Universität Wuppertal
	Gaußstr. 20
	D-42097 Wuppertal

Internet-Zugriff

Die Berichte sind in elektronischer Form erhältlich über die World Wide Web Seiten

`http://www.math.uni-wuppertal.de/wrswt/literatur.html`

Autoren-Kontaktadressen

Gerd Bohlender
Universität Karlsruhe
D-76128 Karlsruhe, Germany
E-mail: `bohlender@math.uka.de`

Mariana Lüderitz Kolberg
PPGCC - Pontifícia Universidade Católica do Rio Grande do Sul
Porto Alegre
Brazil
E-mail: `mkolberg@inf.pucrs.br`

Dalcidio Moraes Claudio
Pontifícia Universidade Católica do Rio Grande do Sul
Porto Alegre
Brazil
E-mail: `dalcidio@inf.pucrs.br`

Modifications to Expression Evaluation in C-XSC

Gerd Bohlender

Universität Karlsruhe, Germany

Mariana Lüderitz Kolberg

PPGCC - Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, Brazil

Dalcidio Moraes Claudio

Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, Brazil

Abstract

The C++ interval library C-XSC has been developed to provide high precision and self-validated results. In this paper, several modifications and improvements of expression evaluation in C-XSC are proposed:

Accurate expressions (in the style of Pascal-XSC) are implemented by means of a pre-processor and an overloaded library.

Several modifications are suggested for improving the performance of the library. These improvements include support for parallel processing and faster interval operations by means of a special representation.

Key words: Scientific Computing, C-XSC, interval arithmetic, accurate expressions, parallel processing.

* This work was partly supported by project Probral of DAAD

Email addresses: `bohlender@math.uka.de` (Gerd Bohlender), `mkolberg@inf.pucrs.br` (Mariana Lüderitz Kolberg), `dalcidio@inf.pucrs.br` (Dalcidio Moraes Claudio).

1 Introduction

In the recent years, many tools have been developed for interval arithmetic and computing with verified results, e.g. the XSC languages and libraries [8,9,13,5]. Some implementation characteristics of these tools are:

- directed rounding;
- optimal scalar product;
- data type *dotprecision* (used to store intermediate values of scalar products in full accuracy, without rounding errors);
- special arithmetic for complex and interval data types;
- special arithmetic for high accuracy matrix and vector data types;
- high precision arithmetic for all data types;
- optimal expression evaluation.

In particular, C-XSC is a C++ class library which is intended for the development of numerical algorithms with interval arithmetic, providing highly accurate and automatically verified results. Version 2.0 which was recently released works with all standard compatible C++ compilers [5,6]. Source code and documentation of C-XSC can be obtained free from the website *xsc.de*.

In this paper, we discuss two subjects for further improvement of the C-XSC library:

- Simpler usage by means of accurate expressions
- Improvement of performance

2 Accurate Expressions

Accurate expressions are composed of subexpressions which involve the accumulation of products. To better define their syntax, we present the following grammar that defines the operations which can be evaluated in these expressions:

$$X \leftarrow a \,|\, -a \,|\, a \cdot b \,|\, -a \cdot b \,|\, X + a \,|\, X - a \,|\, X + a \cdot b \,|\, X - a \cdot b$$

where a and b belong to one of the spaces of scientific computation represented in C-XSC, e.g. vectors or matrices of real numbers, complex numbers, or intervals.

Suppose, that the following expression should be computed with high accuracy: $y_0 = b - A * x_1 - A * x_0$, where y_0, b, x_1, x_0 are real vectors and A is a real matrix (data type *rvector* and *rmatrix*, respectively). In Pascal-XSC, this expression can be programmed with nearly no modifications. In C-XSC, however, a long and error prone sequence of operations is required, as shown in the following program part:

```
for (i=Lb(A,1) ; i<=Ub(A,1) ; i++) {
     accu=b[i];
     accumulate(accu,-A[Row(i)],x1);
     accumulate(accu,-A[Row(i)],x0);
     y0[i]=rnd(accu);
}
```

To resolve this problem, accurate expressions in C-XSC have to be made available in a similar way like in Pascal-XSC, i.e. in a mathematical notation, prefixed by a symbol for high accuracy and an optional rounding symbol.

2.1 Implementation

An important factor has to be sonsidered: in contrast to Pascal-XSC, C-XSC is not an extension of a language, but a library, written in the programming language C++. To implement these expressions in C-XSC we propose a new environment that will provide a direct evaluation of accurate expressions in C-XSC. This environment is based on a pre-processor and a library, which together implement the routines for the evaluation of accurate expressions.

The library, written in C++, uses the C-XSC data types, including new routines for the direct evaluation of accurate expression. The following operations were implemented:

- Construction of a new data type for vectors (example: *dpvector* - vector of *dotprecision* elements) and matrices (example: *dpimatrix* - matrix of *idotprecision* elements) with elements of the following data types *dotprecision, idotprecision, cdotprecision and cidotprecision.* This includes methods to construct, destruct and copy such special variables.
- Redefinition of the assignment operator.
- Addition and subtraction of different combinations of the C-XSC data types and *dotprecision.*
- Multiplication of C-XSC data types.
- Monadic subtraction for C-XSC data types.
- We also implemented some methods for directly rounding the created variables (e.g. *dpmatrix*) to the corrsponding C-XSC data types (e.g. *rmatrix*).

The pre-processor takes as input an extension of the programming language C++ with the symbols that identify the accurate expression and its rounding mode: #* (round to nearest), #> (round up), #< (round down), ## (round to interval, interval vector, interval matrix, etc.), # (do not round at all, store with full precision). As output it generates a C++ program with function calls to an extended C-XSC library. That means that a program using the new definition of accurate expression has first to be sent to the pre-processor to generate the code that will be accepted by the normal C++ compiler. This pre-processor only makes a mechanic transformation of certain symbols to function calls that are defined in the library. Therefore, some restrictions are necessary to make it possible to write a pre-processor instead of a full compiler.

Our pre-processor reads a text file with a program written in C++ with some extensions, for example some special symbols. The pre-processor changes these symbols to a pre-defined new set of symbols and stores the modified text in an output file which may then be compiled by a normal C++ compiler. Due to the use of function overloading, we do not need to know the data types of the operands in an accurate expression, and it is possible to make a mechanical transformation of the occurring operators to function calls. Table 1 shows some examples of the transformations that the pre-processor executes. Lower case variables are vectors, upper case variables are matrices. We do not indicate the data type of the components, because the change is the same for the different data types.

Expression	Transformation
x = #*(A*B)	dprnd(dpmul(A,B))
x = #>(A*b-C*d)	dprndup(dpsub(dpmul(A,b),dpmul(C,d)))
x = #*(A*B+C*D-A)	dprnd(dpsub(dpadd(dpmul(A,B),dpmul(C,D)),A))
x = #<(a*b-c*d+e*f)	dprnddown(dpadd(dpsub(dpmul(a,b),dpmul(c,d)),dpmul(e,f)))

Table 1
Examples of transformations

The implemented pre-processor is powerful enough to handle most accurate expressions, but it does not analyze the semantics or syntax of a C++ program.

2.2 Test Results

We made some comparative tests between the execution time of the code that evaluates the accurate expressions in a direct way (new implementation) and the execution time of the code that evaluates the accurate expression using just C-XSC (code that the user would have to write using the specific functions, loops and the special C-XSC data types). The result shows that the execution time using our library and pre-processor is bigger. On the other hand, the factor of this difference changes according to the operation. Tests were performed e.g. for the addition, subtraction and multiplication of square matrices.

The multiplication is the operation with the best result. In spite of multiplication being slower than addition and subtraction, the quotient between the versions was the smallest, around 1.1. We observe also that the quotient of execution times for big dimensions does not grow as compared with small dimensions.

In addition and subtraction of matrices, a larger increase in execution time can be observed. Depending on the dimensions of matrices, a factor of 5 - 10 was observed. These effects are explained by the following reasons which are caused by limitations of the preprocessor:

- The necessity of creating one or more variables (scalars, vectors or matrices) of type *dotprecision*, which requires more time for memory management, especially in the case of big matrices.
- The operations that used to be evaluated element by element, using only one *dotprecision* variable in C-XSC, now are evaluated using matrices of *dotprecision* variables. This increases the execution time.
- Results are returned by means of a copy constructor which involves some overhead for the creation and copying of objects.

These results were published in [10–12].

3 Performance of C-XSC

Performance of C-XSC is not as good as pure C++. There are several reasons for this:

- The runtime system (inherited from Pascal-XSC) is very flexible, but not ideally adapted to IEEE 754 arithmetic.
- Interval operations require directed roundings, this may cost additional runtime.
- Scalar products are computed with maximum accuracy; software implementations are time consuming.
- Operations in higher numerical spaces (e.g. matrix and vector operations) are always executed with maximum accuracy, using the software scalar product.
- C-XSC is modular: well structured, but many function calls, no optimizations in e.g. matrix operations.

Some suggested improvements are:

- Replace the runtime system with a small improved library optimized for IEEE 754 arithmetic
- Improve operations in higher numerical spaces (e.g. matrix and vector operations); provide these with maximum accuracy, using the software scalar product, or with reduced accuracy, but verified interval inclusion
- Implement optimized version for parallel processors
- Reduce overhead for directed roundings
- Provide different versions of scalar product [1,14] with maximum accuracy, possibly using different algorithms, or with reduced accuracy, but verified interval inclusion

3.1 Version for Parallel Processors

For the efficient implementation of C-XSC on parallel computers, special MPI functions for the following C-XSC data types were developed:

- dotprecision types
- data types interval, complex, complex interval
- matrices and vectors

Implementation was on a labtec cluster at UFRGS, Porto Alegre [7]. Tests were performed with

- matrix multiplications
- scalar products
- linear systems (full and band matrices)
- conjugate gradient methods

3.2 Implementation of Interval Operations in C-XSC

In interval operations, lower bounds have always to be rounded down, and upper bounds have to be rounded up, resp. Therefore, in a straight forward implementation, the rounding mode has to be switched twice per interval operation.

On many common processors (e.g. Intel Pentium), switching the rounding mode is a very expensive operation which may require up to 10 times as much execution time as a floating-point operation. Therefore, the rounding mode should be switched as infrequently as possible.

In interval vector- and matrix operations, switching the rounding mode can be avoided by modifying the sequence of operations:

- set the rounding mode to “round down”,
- compute all lower bounds of all components,
- set the rounding mode to “round up”,
- compute all upper bounds.

Here, the rounding mode is switched only twice instead of $2 * dim$ times. This method is applied e.g. in INTLAB [15], however it can only be applied in vector and matrix operations, not in simple interval operations.

The formula $\nabla(a) = -\Delta(-a)$ can be used, to replace the switching of rounding modes by negations. Negation usually requires much less overhead than switching the rounding modes. The number of additional negations can be minimized by storing intervals in a special representation: instead of infimum and supremum, an interval is stored in the form infimum and negated supremum:

```
class interval {
        double inf, nsup; // infimum and negated supremum
        ...
}
```

This negation has to be taken care of in all component references and arithmetic operations [2].

The overhead (in terms of additional negations) is between 0 and 2 negations per interval operation. On the average, the overhead is about 10% .

This implementation is based on several assumptions: in particular, the rounding mode must be preserved between operations, and the logical sequence of operations must be preserved.

4 Conclusion and Future Work

Some extensions and optimizations for expression handling in C-XSC were proposed.

The aim to make the handling of accurate expressions in C-XSC as simple as in Pascal-XSC, using a mathematical notation, has been achieved. However, for some operations the performance is not quite as good as in pure C-XSC.

With respect to performance optimizations, work is in progress. Parts are implemented in the context of a cooperation project Probral of DAAD with partners in Germany and Brazil. Cooperation with more partners would be desirable.

References

[1] Bohlender, G. : What Do We Need Beyond IEEE Arithmetic? Pages 1-32 in: Ullrich, Ch.: Computer Arithmetic and Self-Validating Numerical Methods. Academic Press, San Diego, 1990.

[2] Bohlender, G.: Faster Interval Computations Through Minimized Switching of Rounding Modes. Porto Alegre, 2002.

[3] Hammer, R.; Hocks, M.; Kulisch, U.; Ratz, D. : Numerical Toolbox for Verified Computing I - Basic Numerical Problems. Springer-Verlag, Berlin, 1993.

[4] Hammer, R.; Hocks, M.; Kulisch, U.; Ratz, D. : C++ Toolbox for Verified Computing I - Basic Numerical Problems. Springer-Verlag, Berlin, 1995.

[5] Hofschuster, W.; Krämer, W.; Wedner, S.; Wiethoff, A.: C-XSC 2.0 A C++ Class Library for Extended Scientific Computing. Universität Wuppertal, Preprint BUGHW - WRSWT 2001/1, 2001.

[6] Hofschuster, W.; Krämer, W.: C-XSC 2.0: A C++ Library for Extended Scientific Computing. Numerical Software with Result Verification, Lecture Notes in Computer Science, Volume 2991/2004, Springer-Verlag, Heidelberg, pp. 15 - 35, (2004).

[7] Hölbig, C.A.; Kolberg, M.L.; Morandi Jr., P.S.; Alcalde, B.F.K.; Diverio, T.A.; Claudio, D.M.: Solvers with High Accuracy. ICCAM, Leuven, 2004.

[8] Klatte, R.; Kulisch, U.; Neaga, M.; Ratz, D.; Ullrich, Ch.: Pascal-XSC - Language Reference With Examples, Springer-Verlag, Berlin, 1992.

[9] Klatte, R.; Kulisch, U.; Lawo, C.; Rauch, R.; Wiethoff, A.: C-XSC - A C++ Class Library for Extended Scientific Computing. Springer-Verlag, Berlin, 1993.

[10] Kolberg, M. L.; Hölbig, C. A.; Bohlender, G.; Claudio, D. M.: Accurate Expressions in C-XSC: a new Format. In: Eleventh International Congress on Computation and Applied Mathematics, Leuven, 2004. Abstract of ICCAM 2004.

[11] Kolberg, M. L.; Hölbig, C. A.; Bohlender, G.; Claudio, D. M.: New Accurate Expressions in C-XSC. In: GAMM - 75th Annual Meeting, Dresden, 2004. Proceedings of annual meeting. p.218-218.

[12] Kolberg, M. L.; Hölbig, C. A.; Bohlender, G.; Claudio, D. M.: New Accurate Expressions in C-XSC. Proceedings In Applied Mathematics And Mechanics, Berlin, 2004, not yet published.

[13] Krämer, W.; Kulisch, U.; Lohner, R.: Numerial Toolbox for verified Computing II: Advanced Numerical Problems. Berlin, Springer-Verlag, 1996, 385p.

[14] Ogita, T.; Rump, S.M.; Oishi, S.: Accurate Sum and Dot Product. scan 2004, Fukuoka, 2004.

[15] Rump, S.M.: INTLAB, INTerval LABoratory. `www.ti3.tu-harburg.de/~rump/intlab/`.