Extending the Range of C-XSC: Some Tools and Applications for the use in Parallel and other Environments

Markus Grimmer[1]

CETEQ GmbH & Co. KG
42119 Wuppertal, Lise-Meitner-Str. 5-9, Germany
markus.grimmer@math.uni-wuppertal.de

Abstract. We present some examples of extensions for C-XSC that have been developed lately. Among these are extensions that give access to further hardware and software environments as well as applications making use of these possibilities.
The first area of extension is C-XSC usage in parallel environments. An MPI package for C-XSC data types allows to easily use C-XSC in parallel programs without bothering about the internal structure of data types. Different versions of parallel verified linear system solvers based on the package are now available. An application making use of these and further extensions is a parallel verified Fredholm integral equation system solver. Some results are given to demonstrate the reduction of computation time and, at the same time, the accuracy gain that can be obtained using the increased computation power.
Another possibility to extend the range of C-XSC is to make results available for further computation in different software environments as, for example, computer algebra packages. An example of this is presented for the Maple interval package intpakX.

Keywords. C-XSC, Integral Equations, Interval Arithmetic, Maple, MPI, Parallel Environment, Verified Linear System Solver.

1 Introduction

C-XSC is a well-known C++ class library for scientific computation. It is one important member of the family of XSC languages, libraries and compilers that cover a wide range of computational tools and methods. It is thus of special value not only to maintain and update all these elements but also to extend the current range towards new areas that could not be used with XSC languages and applications before or where XSC languages offered only limited support.

One of these areas is parallel computation. Of course, C-XSC can be used in parallel environments if only the very data elements of C-XSC types are used, but using C-XSC *objects* in parallel communication in a straightforward way hasn't been possible in the past.

In this article, we present an extension for C-XSC that makes it quite more comfortable to use C-XSC in parallel environments with the parallel communication interface MPI, together with some parallel applications that make use of it.

Besides the usage of C-XSC in special environments it is also helpful to be able to export results from C-XSC programs to other computing environments as, for example, Computer Algebra packages. An example for this is presented as well.

1.1 C-XSC: A short overview

First of all, we want to give an overview of C-XSC and some of its existing older and lately developed extensions and applications.

The C-XSC library [1] [2] itself consists of the following elements:

Interval Data Types First of all and most importantly, C-XSC offers a range of data types for interval computation:

– Basic Data Types: `real`, `interval`, `complex`, `cinterval`
– Vector and Matrix Types: Types for vector and matrix computation for either of the basic types
– Staggered Multiple Precision Arithmetic: Multiple precision arithmetic types allow for the use of n-fold precision using a vector of basic type elements to represent a staggered number.
– Dotprecision Types: These allow for the exact representation of scalar product expressions, i.e. sums of products of basic type numbers.

Arithmetic For the mentioned data types, not only basic arithmetic operations are available, C-XSC also offers one of the most extensive sets of standard functions: Among these are also less well-known functions as the Gaussian error function or compound functions as $\sqrt{1-x^2}$ and more.

Toolbox The formerly separate C-XSC Toolbox [3] is now an integrated part of the library. It contains a number of numerical algorithms with result verification, among these

– Serial Linear system solvers
– Global Optimization algortihms

and more.

Extensions Extensions for C-XSC can be divided into different categories:

– Arithmetic Extensions
– C-XSC applications
– Technical Extensions

Arithmetic Extensions Examples of arithmetic extenisons are *Taylor Arithmetic* and *Hansen Arithmetic*. The C-XSC *Taylor Arithmetic* [4] [5] offers

– One- and multidimensional Interval Taylor Arithmetic types and functions
– Real and complex Interval Taylor Arithmetic types and functions
– Real staggered (multiple precision) Interval Taylor Arithmetic
– A complete set of standard functions

Lately added have been

– Improved function implementations for special expressions (as $\sqrt{1-x^2}$)
– Algorithms for Gaussian error function *erf* and complementary error function *erfc*

An example for the use of the C-XSC Taylor Arithmetic in an application will be given in the next sections.

The C-XSC *Hansen Arithmetic* which represents a generalized interval arithmetic has also been developed lately in the course of the development of a parametric interval linear system solver [6] and is a further useful arithmetic extension of C-XSC.

In addtion, an improved staggered correction arithmetic with enhanced accuracy and very wide exponent range is now available [7].

C-XSC applications and Technical Extensions But there are not only arithmetic extensions of C-XSC. On the one hand, there are C-XSC applications beyond the Toolbox algorithms, as, for example,

– New and especially parallel linear system solvers
– Linear Fredholm Integral Equation Solvers
– Parametric linear system solvers [6]

We will have a look at the first two of these below.

On the other hand, it is technical extensions that allow for the development of applications in the first place. An example of this kind is the MPI extension for C-XSC data types that is presented in the following section.

2 C-XSC in Parallel Environments

In this section, we present a package for C-XSC computation in parallel environments.

In distributed memory parallel environments it is neccessary to communicate data between processes. MPI [8] is the most common interface for this kind of task. It offers a variety of functions for message passing and related tasks. Since MPI implementations operate on basic data types only, it is not possible to communicate class objects in a straightforward way. For a class library like C-XSC, additional packages for integration of MPI and the class library are necessary.

MPI communication with user defined data types can be done in different ways:

- Direct application of existing routines, applied to single data elements of objects
- Usage of the MPI data packing/unpacking mechanism
- Definition of MPI data types with the type definition mechanism

These approaches have advantages and disadvantages.

Direct application of existing routines, naturally, is quite always possible. Unfortunately, MPI routines cannot be applied to the data object in question since MPI only knows to handle elements or arrays of elements of a number of basic types. This is inconvenient, and it leaves the handling of class internal structures to the user or application developer. Moreover, communication will be time consuming, since all data elements are handled in separate communication calls.

MPI offers two strategies to make communication of data more convenient. The first of these is the *packing/unpacking* mechanism. It is the more versatile of the two since it allows to pack, then send, receive and finally unpack every kind of data you like.

The second strategy is the definition of new MPI data types. There is a collection of routines for data type definition in MPI making it possible to virtually assemble data by defining a so called *type map* and giving it a name, but unfortunately, there are limitations. Namely, no dynamically allocated data can be incorporated into the new type, and lenghts of any data structures of variable size have to be known in advance to develop the new type, which is not the case in C-XSC and typically not useful for the definition of a general interface.

Hence, the package implemented for C-XSC data types uses the first of the two strategies for types that incorporate dynamic memory allocation and/or array-like structures like vector and matrix types.

It uses data type definition for the basic C-XSC data types `real`, `interval`, `complex` and `cinterval`, and includes packing/unpacking routines and communication routines as template functions for the types derived from these. All C-XSC types are included in the package:

- Basic types: `real`, `interval`, `complex`, `cinterval`
- Vector and Matrix data types for the basic types
- Staggered (= multiple precision) data types
- Dotprecision ("accumulator") data types

Regarding MPI communication, the following routines are covered:

- `MPI_Pack`, `MPI_Unpack`
- Point-to-point communication:
  - `MPI_Send`, `MPI_Bsend`, `MPI_Ssend`, `MPI_Rsend`
  - `MPI_Isend`, `MPI_Ibsend`, `MPI_Irsend`, `MPI_Issend`
  - `MPI_Recv`
- Collective Communication:
  - `MPI_Bcast`

A number of additional features is included as well:

- Communication functions for submatrices, i.e. rows, columns and proper submatrices; submatrices can also be used as full matrices on either communication node, i.e. a submatrix from a sending node can be stored as a full matrix on the receiving side.
- Communication functions for one and two-dimensional real C-XSC Taylor Arithmetic Types (see above)
- Communication functions for the STL vector type

In a general interface or package, it is not possible to offer general versions of further collective communication routines, since data subdivision for gather/scatter processes can be done in various ways, and the decision how to subdivide and distribute data has to be left to the author of the application.

To analyze the performance of the newly developed functions, tests have been done, and we want to give the following diagrams for illustration.

The tests were carried out in the parallel test environment ALiCEnext [9] in Wuppertal:

- 1024 1.8 GHz AMD Opteron 64 Bit Processors on 512 nodes
- LINPACK max. performance 2083 GFlops

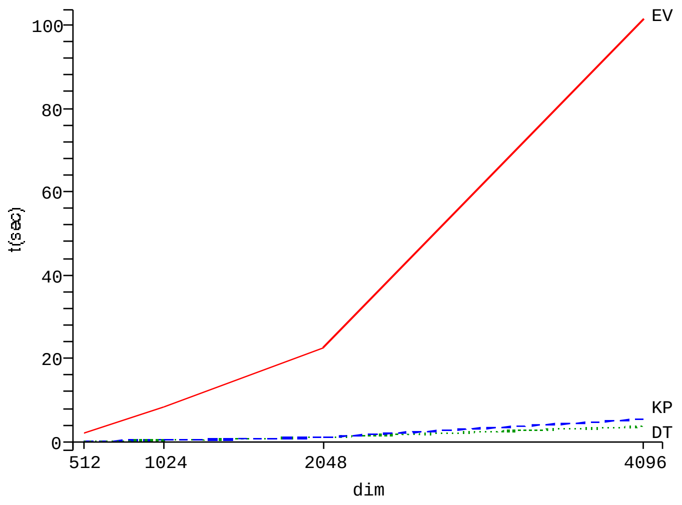

Fig. 1. EV: Elementwise communication; KP: Communication package; DT: Data type definition for different fixed dimensions

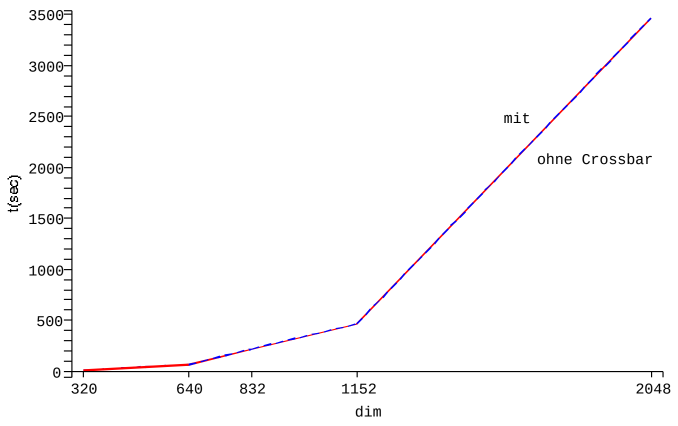

Fig. 2. Tree vs. Crossbar network (sample problem)

The first diagram (see fig. 1) shows communication times for the circulation of C-XSC matrices of different sizes through 32 nodes. The test was carried out on randomly chosen nodes, so that the given times are only example times. The depicted ratio between the three methods, however, was found to be rather independent of the choice of nodes.

We secondly tested two different node topologies: In one case, the nodes were connected as a tree network, in the other case a crossbar switch was activated on the same machine. In order to find out if a typical example problem would be influenced by the different choice of topology, we computed the solution for different problem sizes and found that there was no significant change in the measured times (see fig. 2).

3 Applications using C-XSC and MPI

In this section, we present a few applications that make use of the communication package described in the previous section.

An important type of application to use a parallel communication package are parallel linear system solvers.

In C-XSC, different interval linear system solvers are available now, building upon each other:

1. The base is the C-XSC Toolbox interval linear system solver, which is a serial solver for real input data
2. On stage 2, there is the serial interval linear system solver by Hölbig [10] including the second verification step with double precision computation of the approximate inverse as proposed by Rump [11]
3. The first parallel interval linear system solver developed in [12] builds upon the serial solver by Hölbig and allows for interval input data

4. Just recently developed there is now a fast interval linear system solver package by Zimmer [13] including fast serial and parallel solvers for various input formats using BLAS and LAPACK and the error free transformation dot product proposed by Rump, Oishi et al. [14]

Linear system solvers are applications of their own, but typically, they can also be found as subparts of more complex applications. As a C-XSC application using the parallel interval linear system solver 3 from above a verified integral equation system solver for linear Fredholm integral equation systems of the second kind is presented in the following paragraphs.

We will shortly describe the solver first and then give some results from the actual C-XSC implementation in a high performance computing environment.

A system of Fredholm Integral Equations of the second kind is given by

$$y^i(s) - \lambda \sum_{j=1}^{N} \int_{\alpha_j}^{\beta_j} k^{ij}(s,t) y^j(t)\ dt = g^i(s) \tag{1}$$

for $i = 1...N$ with continuous kernel functions k^{ij} on $[\alpha_i, \beta_i] \times [\alpha_j, \beta_j]$ with

$$\alpha_i := a + \frac{i-1}{N}(b-a), \quad \beta_i := a + \frac{i}{N}(b-a), i, j = 1...N.$$

as well as continuous g^i on $[\alpha_i, \beta_i]$ and unknown result functions $y^i,\ i = 1...N$.

With

$$\begin{aligned} \mathcal{Y} &:= (y^1, ..., y^N)^T \\ \mathcal{G} &:= (g^1, ..., g^N)^T \\ \mathcal{K} &:= (k^{ij})_{i,j=1...N} \end{aligned}$$

we get

$$\mathcal{Y}(s) - \lambda \int_a^b \mathcal{K}(s,t) \mathcal{Y}(t)\ dt \ = \ \mathcal{G}(s). \tag{2}$$

The system of integral equations can thus be written in the same way as a single integral equation. For this reason, a solution method for systems of integral equations can be derived from a corresponding solution method for single equations.

If a kernel κ has a representation

$$\kappa(s,t) = \sum_{m=1}^{T} a_m(s) b_m(t)$$

with continuous functions $a_m, b_m, m = 1...T$, it is called *degenerate kernel of order* T, so that matrices $\mathcal{A}_m$, $\mathcal{B}_m$, $m = 0..T$ (with some suitable $T \in I\!N$) for degenerate kernels can be additionally introduced for the above matrix notation.

The kernel of a linear Fredholm integral equation of the second kind can be represented as

$$K = K_{\mathfrak{E}} + K_{\mathfrak{N}}$$

with a degenerate part $K_{\mathfrak{E}}$ and a non-degenerate part $K_{\mathfrak{N}}$ (see, e.g., [15]). According matrix denotations $\mathcal{K}_{\mathfrak{N}}$, $\mathcal{K}_{\mathfrak{E}}$ can be introduced componentwise.

The solution methods are described in detail in [12], [16]and [17] and shall not be discussed here. We only want to give the solution algorithm to indicate that the most relevant parts of the algorithm can be efficiently parallelized (see method 1).

Method 1: Integral Equation System Solution Method

```
For $j := 1...N, n := 1...2T$:
    Compute integrals of basic monomials $(t - t_0^j)^n$.
Carry out the iteration
    $\mathcal{F}^0 := \mathcal{G}$;   $\mathcal{F}^{i+1} := \mathcal{G} + \mathcal{K}_{\mathfrak{N}}\mathcal{F}^i$,   $i := 0, 1, ...$
until $\mathcal{F} := \mathcal{F}^{i+1} \subseteq \mathcal{F}^i$ (or abort).

For $m := 0...T$:                                                    (PAR)
    For $j := 1...N$:
        Carry out the iteration
            $\mathcal{C}_m^{j,0} := \mathcal{A}_m^j$;   $\mathcal{C}_m^{j,i+1} := \mathcal{A}_m^j + \mathcal{K}_{\mathfrak{N}}\mathcal{C}_m^{j,i}$
            $i := 0, 1, ...$
        until $\mathcal{C}_m^j := \mathcal{C}_m^{j,i+1} \subseteq \mathcal{C}_m^{j,i}$  (or abort).

Compute the entries                                                  (PAR)
  – $\mathcal{M} := (\mathcal{M}_{ij})_{i,j=0...T}, \mathcal{M}_{ij} := \int_a^b \mathcal{B}_i(t)\mathcal{C}_j(t)dt$
  – $\mathcal{R} := (\mathcal{R}_i)_{i=0...T}, \mathcal{R}_i := \int_a^b \mathcal{B}_i(t)\mathcal{F}(t)dt$
of the interval linear system for the final application of the method for degenerate kernels.

Solve the interval linear system                                     (PAR)
    $(\mathcal{I} - \lambda\mathcal{M})\mathcal{X} = \mathcal{R}$.

Compute the solution function $\mathcal{Y} := \mathcal{G} + \lambda\sum_{m=0}^{T} \mathcal{C}_m\mathcal{X}_m$.
```

The highlighted parts of the algorithm were parallelized, the remaining parts being of minor importance since they only have a small share of the method's overall complexity. We can see that one of the parallel parts is given by the application of a parallel interval linear system solver.

The actual implementation of the solver was designed to solve single integral equations by splitting them up into integral equation systems as indicated above.

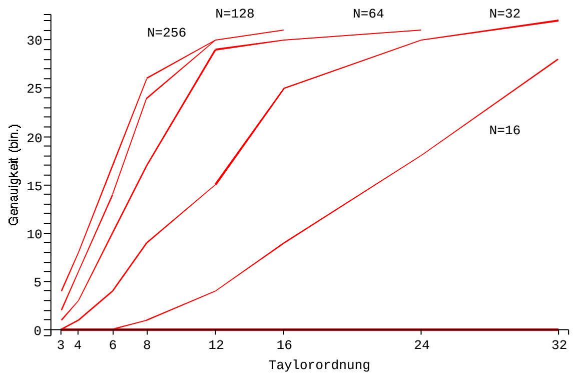

Fig. 3. Accuracy Gain for growing Taylor order and different system sizes

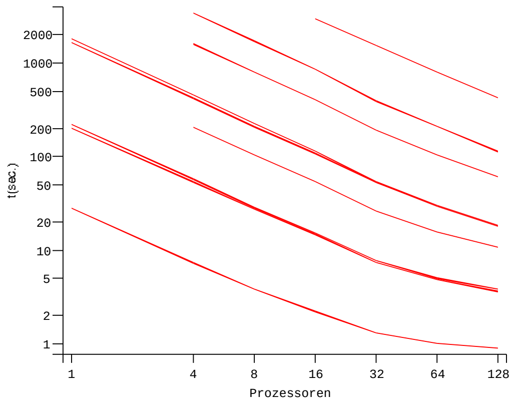

Fig. 4. Computation time for increasing number of processors (different parameter values for a sample problem)

Let us point out which elements of the solver make it a relevant example for a parallel C-XSC application using the presented extensions:

– We solve a functional problem with functions in one and two variables which are represented in interval Taylor arithmetic
– As a matrix problem with possibly high matrix dimensions, the method is time consuming and thus a good candidate to be parallelized for increased efficency.
– The solution reduces to an interval linear system so that an interval linear system solver can be applied

We now want to give some results for this integral equation system solver, taking into account time/parallelization and accuracy issues.

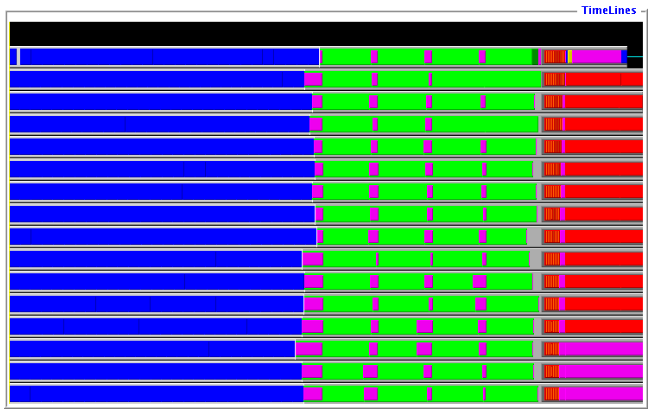

Fig. 5. Example: Taylor order 3, System order 128, 16 procs., computation time $\sim$ 2 min. - Time shares: Fixed Point Iterations (Blue) - LS Computation (Green) - LSS (Red) - Comm.+Idle (Magenta)

In figure 3 result accuracy is plotted against Taylor order for different dimensions of the example integral equation system derived by splitting up the integral equation

$$y(s) - \int_{-1}^{1} st^2 - s^2t \; y(t) \; dt = (s - \frac{1}{2}) \cdot erf(sin(9s)).$$

It is clearly visible that solutions become more accurate with increasing dimension of the integral equation system, especially in cases where low system size does not yet yield reasonable results.

Parallelization is considered in figure 4, giving example timings for different parameter combinations and up to 128 processors. Linear speedup could be achieved for most of the relevant parameter combinations.

We also analyzed time shares of different parts of the program:

– Iteration phase
– Computation of the entries of the linear system
– Linear system solution (including approximate matrix inversion and further matrix operations, e.g. matrix multiplications)

Figures 5 and 6 show the time shares of the different parts of the algorithm for different (medium sized) parameter combinations.

It is observable that time shares differ significantly for different parameter values due to the complexity of the parts of the algorithm. The greatest time share is used by either the iteration part or the linear system assembly, but not by linear system solution. In general, computation time is evenly distributed over the processors and idle times are kept within bounds.

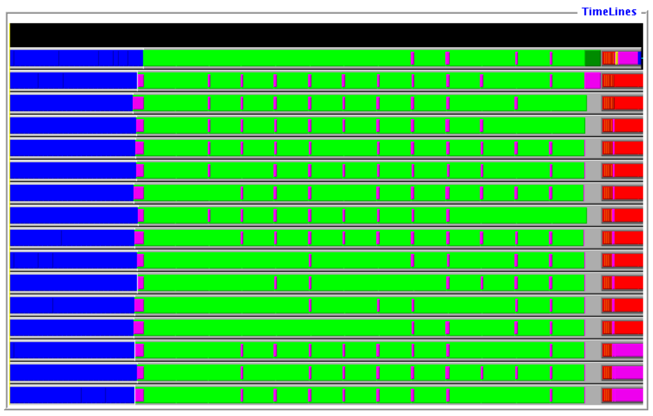

Fig. 6. Example: Taylor order 12, System order 32, 16 procs., computation time $\sim$ 2 min. - Time shares: Fixed Point Iterations (Blue) - LS Computation (Green) - LSS (Red) - Comm.+Idle (Magenta)

4 Exporting C-XSC Results to other Environments

It is not only important to extend the range of a library like C-XSC. It is also valuable to be able to export results to other environments since those typically offer further possibilities. For example, Computer Algebra Packages (with suitable interval facilites) allow for symbolical computation with result functions from a C-XSC integral equation solver, and they also allow for visualization of results.

As an example for C-XSC and Maple, a Maple interface has been included in the above integral equation application. It exports solution functions as Maple code and range enclosures of the solution functions for visualization in Maple.

Interval arithmetic in Maple is provided by the Maple package intpakX [18].

Example Consider the integral equation

$$\begin{aligned} y(s) - \frac{1}{2}\int_0^1 (s+1)e^{-st}y(t)\ dt \\ = e^{-x} - \frac{1}{2} + \frac{1}{2}e^{-x+1}, \qquad s,t \in [0,1] \end{aligned}$$

which, according to Kress [19], has the analytic solution

$$y : s \to e^{-s}.$$

The Taylor coefficients of the first component of the solution can be computed as

```
[ 0.969232610773108516, 0.969235565177278713]
[-0.969233189716268040,-0.969233131674795078]
```

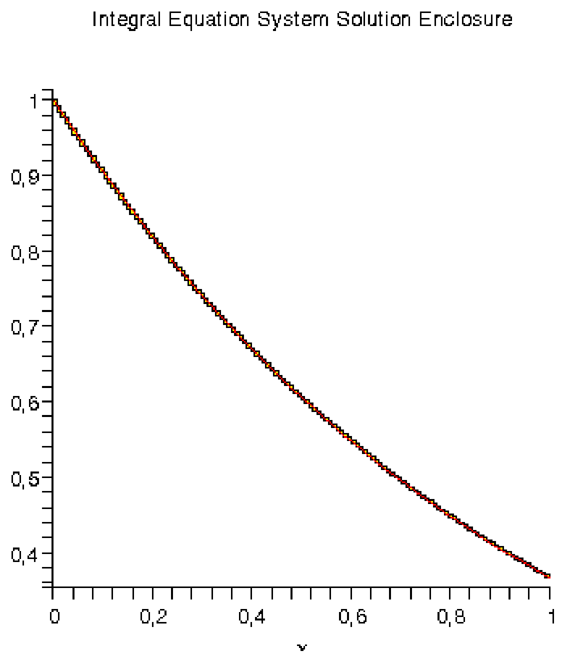

Fig. 7. Maple visualization of integral equation solution

```
 [ 0.484616598544399534, 0.484616637435440978]
 [-0.161554269639638399,-0.161550720767527217]
 [ 0.038892044738204872, 0.041956101920650240]
```

The function output in Maple code is

```
IGLSys_Lsg[0] := x ->
[ 0.969232610773108516,   0.969235565177278713]
&+ ( [-0.969233189716268040,-0.969233131674795078]
&* ( ( x
&- [ 0.031250000000000000, 0.031250000000000000] )
   &intpower 1 ) )
&+ ( [ 0.484616598544399534, 0.484616637435440978]
&* ( ( x
&- [ 0.031250000000000000, 0.031250000000000000] )
   &intpower 2 ) )
&+ ( [-0.161554269639638399,-0.161550720767527217]
&* ( ( x
&- [ 0.031250000000000000, 0.031250000000000000] )
   &intpower 3 ) )
&+ ( [ 0.038892044738204872, 0.041956101920650240]
&* ( ( x
&- [ 0.031250000000000000, 0.031250000000000000] )
   &intpower 4 ) ) ;
```

The solution can be visualized in Maple as shown in figure 7.

5 Conclusions

The extensions presented in the above sections allow for the successful implementation of parallel applications using C-XSC and MPI, so that parallel environments as important computing environments for solving time consuming problems can be easily accessed with C-XSC. They contribute to making the XSC languages a valuable and still growing framework for scientific computing. Moreover, examples show how C-XSC can be connected to other environments with different focus. Work is going on to provide further efficient C-XSC problem solution applications and further arithmetical and technical C-XSC extensions to give access to further computing environments.

References

1. C-XSC: C-XSC Download. (http://www.math.uni-wuppertal.de/wrswt/xsc/cxsc_new.html)
2. Hofschuster, W., Krämer, W.: C-XSC 2.0 – a C++ class library for extended scientific computing. In Alt, R., Frommer, A., Kearfott, B., Luther, W., eds.: Numerical Software with Result Verification, Springer Lecture Notes in Computer Science 2991 (2004) 15–35
3. Hammer, R., Hocks, M., Kulisch, U., Ratz, D.: C++ Toolbox for Verified Computing: Basic Numerical Problems. Springer, Berlin, Heidelberg, New York (1995)
4. Blomquist, F., Hofschuster, W., Krämer, W.: Real and complex Taylor Arithmetic in C-XSC. Preprint BUW-WRSWT 2005/4, University of Wuppertal (2005) http://www.math.uni-wuppertal.de/wrswt/literatur/lit_wrswt.html.
5. Bräuer, M.C.: Berechnungsmethoden für Ableitungen und Steigungen und deren Realisierung in C-XSC. Master's thesis, University of Karlsruhe (1999)
6. El-Owny, H.: Verified Solution of Parametric Interval Linear Systems. PhD thesis, University of Wuppertal (2007)
7. Blomquist, F., Hofschuster, W., Krämer, W.: Real and Complex Staggered (Interval) Arithmetics with Wide Exponent Range (in German). Preprint BUW-WRSWT 2008/1, University of Wuppertal (2008) http://www.math.uni-wuppertal.de/wrswt/literatur/lit_wrswt.html.
8. Message Passing Interface Forum: MPI: A message passing interface standard. Library specification, University of Tennessee (1993-1995)
9. ALiCEnext: Alicenext information. (http://www.alicenext.uni-wuppertal.de)
10. Hölbig, C., Krämer, W.: Selfverifying solvers for dense systems of linear equations realized in C-XSC. Preprint BUW-WRSWT 2003/1, University of Wuppertal (2003) http://www.math.uni-wuppertal.de/wrswt/literatur/lit_wrswt.html.
11. Rump, S.: Kleine Fehlerschranken bei Matrixproblemen. PhD thesis, University of Karlsruhe (1980)
12. Grimmer, M.: Selbstverifizierende mathematische Softwarewerkzeuge im High Performance Computing. PhD thesis, University of Wuppertal (2007)
13. Zimmer, M.: Laufzeiteffiziente, parallele Löser für lineare Intervallgleichungssysteme in C-XSC. Master's thesis, University of Wuppertal (2007)

14. Ogita, T., Rump, S., Oishi, S.: Accurate sum and dot product. SIAM Journal on Scientific Computing **26(6)** (2005) 1955–1988
15. Heuser, H.: Funktionalanalysis. Teubner, Stuttgart (1975)
16. Klein, W.: Zur Einschließung der Lösung von linearen und nichtlinearen Fredholmschen Integralgleichungssystemen zweiter Art. PhD thesis, University of Karlsruhe (1990)
17. Klein, W.: Enclosure methods for linear and nonlinear systems of Fredholm integral equations of the second kind. In Adams, E., Kulisch, U., eds.: Scientific computing with automatic result verification, Boston, Academic Press (1993)
18. Grimmer, M.: Interval Arithmetic in Maple with intpakX. In: PAMM - Proceedings in Applied Mathematics and Mechanics. Number Vol. 2, Nr. 1, Wiley-InterScience (2003) 442–443
19. Kress, R.: Linear Integral Equations. Springer, Berlin, Heidelberg (1989)