A Note on Solving Problem 7 of the SIAM 100-Digit Challenge Using C-XSC

Kolberg[1], M., Krämer[2], W., and Zimmer[3], M.

Abstract: C-XSC is a powerful C++ class library which simplifies the development of selfverifying numerical software. But C-XSC is not only a development tool, it also provides a lot of predefined highly accurate routines to compute reliable bounds for the solution to standard numerical problems.

In this note we discuss the usage of a reliable linear system solver to compute the solution of problem 7 of the SIAM 100-digit challenge [23, 24]. To get the result we have to solve a $20\,000 \times 20\,000$ system of linear equations using interval computations. To perform this task we run our software on the advanced Linux cluster engine ALiCEnext located at the University of Wuppertal and on the high performance computer HP XC6000 at the computing center of the University of Karlsruhe.

The main purpose of this note is to demonstrate the power/weakness of our approach to solve linear interval systems with a large dense system matrix using C-XSC and to get feedback from other research groups all over the world concerned with the topic described. We are very much interested to see comparisons concerning different methods/algorithms, timings, memory consumptions, and different hardware/software environments. It should be easy to adapt our main routine (see Section 3 below) to other programming languages, and different computing environments. Changing just one variable allows the generation of arbitrary large system matrices making it easy to do sound (reproducible and comparable) timings and to check for the largest possible system size that can be handled successfully by a specific package/environment.

Key words: C-XSC, reliable computing, 100-digit challenge, reliable linear system solver, high performance computing, large dense linear systems

1 Introduction

Up to now there are no special linear system solvers for sparse system matrices available in C-XSC [21, 8] (but see e.g. [15, 11] for some first approaches). Thus we will solve the 7. Problem of the Siam 100-digit Challenge (see the next section) for dimensions 20.000 and 50.000 with brute force using our parallel linear system solver for dense matrices [25, 12, 5]. In case of dimension $n = 20.000$ the matrix has 400.000.000 and in case of $n = 50.000$ it has 2.500.000.000 entries. We do not store all these entries

[1]Faculdade de Informática Pontificia Universidade Catolica do Rio Grande do Sul
[2]Bergische Universität Wuppertal, Wissenschaftliches Rechnen/Softwaretechnologie
[3]Bergische Universität Wuppertal, Wissenschaftliches Rechnen/Softwaretechnologie

Dagstuhl Seminar Proceedings 08021
Numerical Validation in Current Hardware Architectures
http://drops.dagstuhl.de/opus/volltexte/2008/1447

on one (master) processor. Each processor generates a matrix `MyA`, which stores only his part of the matrix, with the help of the function `generateElementOfA` (a pointer to this function is the first parameter when calling our new solver `LSS`). Matrices are stored according to the two-dimensional block cyclic distribution used by ScaLAPACK [17]. Internally the solver creates an approximate inverse R of A and also computes an enclosure (an interval matrix [C]) for $I - R \times A$. Because the solver does not exploit the sparse structure of the problem these matrices will be dense. Thus a lot of memory is needed. Nevertheless, using e.g. the cluster computer ALiCEnext at the University of Wuppertal or the high performance computer HP XC6000 at the computing center of the University of Karlsruhe we are able to solve the linear system $Ax = e_1$, where $e_1 = (1, 0 \ldots 0)^T$ denotes the first unit vector. Then, the first component of the solution vector x gives an enclosure of the element at position $(1, 1)$ of A^{-1}, which is needed to solve the 7. problem of the challenge mentioned.

In this note we do not give any details on the algorithm used to solve linear systems with interval matrices. We refer the reader to [22, 7]. The parallelization is described in [25, 12]. A different ansatz (some kind of master-slave) for the parallelization has been used in [5, 16, 4] storing the complete system matrix (and/or some auxiliary matrices) on a single node (memory bottleneck). The new solver [25, 12] overcomes this difficulty using a two dimensional block cyclic distribution of the matrices. It is based on BLAS, MPI [9, 6], ScaLAPACK [17], so called error free transformations [2, 10, 20, 19, 18] and allows very large matrices.

2 The 7. Problem of the Siam 100-digit Challenge

Here comes the original problem [23]: Let $A = (a_{ij})$ be the $20\,000 \times 20\,000$ matrix whose entries are zero everywhere except for the primes 2, 3, 5, 7, ..., 224737 along the main diagonal and the number 1 in all the positions a_{ij} with $|i - j| = 1, 2, 4, 8, \ldots, 16384$. What is the (1, 1) entry of A^{-1}?

Later on we also solve this problem with dimension $n = 50.000$. In this case the primes from 2 to 611953 are the diagonal entries and the number 1 is in all the positions $|i - j| = 1, 2, 4, 8, ..., 32768$.

Please note that we are only interested in an accurate enclosure of the true result within an interval with floating point numbers as bounds (i.e. roughly 15 decimal digits accuracy).

3 Source Code to Create the System Matrix and to Call the Parallel Solver

In this section we reproduce the source code of the program to create the matrix and right hand side. It calls the parallel solver `LSS` for dense linear interval systems available in C-XSC [25, 12].

```
/*
```

```
* Solving problem 7 of the SIAM 100-digit challenge using C-XSC
*
* The problem: A=a_ij is the 20000x20000 matrix (in this program,
* the dimension can be altered) whose entries are zero everywhere except
* for the primes 2,3,5,7,... along the main diagonal and the number 1 in
* all position a_ij with |i-j|=1,2,4,8,...
* What is the entry (1,1) of the inverse of A?
*
* Here this problem is solved by computing the solution of the linear system
* A*x=b, where b is the first unit vector, with a fast parallel dense linear
* system solver using C-XSC. (Note: Since A is a sparse matrix, using a
* dense solver is not very efficient).
*
* Author: Michael Zimmer
*
*/

//Headerfiles for output streams(logfile)
#include <iostream>
#include <fstream>
#include <sstream>
//Headerfile for MPI functions
#include "mpi.h"
//Headerfile for time mesurements
#include "sys/time.h"
//Headerfile for the solver
#include "lss_par.hpp"

using namespace cxsc;
using namespace std;

//Dimension of the interval matrix A
const int dim = 20000;
//All primes between 1 and dim
int primes[dim];
//Directory for logfile of each node
const string datadir = "/data/mzimmer";
//Directory to which all logfiles will be copied at the end
const string targetdir = "/home/mzimmer/Upload/bin/output";

//Get current timer value in s
double getTime() {
   struct timeval _tp;
   gettimeofday(&_tp,0);
   return _tp.tv_sec + _tp.tv_usec / 1000000.0;
}
```

```
//Logarithm of base 10
real log10(real x) {
  return ln(x) / ln(real(10.0));
}

//Logarithm of base 2
real log2(real x) {
  return ln(x) / ln(real(2.0));
}

//Determines if an integer n is a prime
bool isPrime(int n) {
  if(n == 2)
    return true;
  else if(n < 2  ||  n%2 == 0)
    return false;

  bool prime = true;

  for(int i=3 ; i<=(int)sqrt((double)n) ; i++) {
    if(n%i == 0) {
      prime = false;
      break;
    }
  }

  return prime;
}

//Generates all primes between 1 and dim and stores them in the
//array primes
void generatePrimes() {
  int p = 0;
  int i = 0;

  while(p < dim) {
    i++;
    if(isPrime(i)) {
      primes[p] = i;
      p++;
    }
  }
}

//Determines if an integer n is a power of two
```

```
bool isPowTwo(int n) { //works well in the range were we use it
  real l = log2(n);
  return l == (int)_double(l);
}

//Function defining elements of A. The reference parameter r
//will be overwritten with the value of element (i,j) of matrix
//A.
void generateElementOfA(int i, int j, real& r) {
  if(i==j)
    r = primes[i-1];
  else if(isPowTwo(cxsc::abs(i-j)))
    r = 1.0;
  else
    r = 0.0;
}

//Computes an approximate of the average number of
//exact digits (the number of digits equal in the
//infimum and the supremum)
real computeNumberOfExactDigits(ivector &x) {
  dotprecision accu(0);

  for(int i=1 ; i<=VecLen(x) ; i++) {
    real d = diam(x[i]);
    if(d == 0) {
      accu += 1e16;
    } else {
      accu += abs(mid(x[i])) / d;
    }
  }

  return abs(log10(rnd(accu)/VecLen(x)));
}


int main (int argc, char** argv) {
  //MPI process ID
  int myId;
  //Number of processes
  int np;
  //Dimension of the matrix A
  int m=dim, n=dim;
  //Right hand side of the system
  rvector b(m);
  //Computed solution
```

```
ivector x(n);
//Precision to be used for computation of dot products
//(K-fold working precision)
int K=2;
//Number of threads used for computation of dot products
int threads=0;
//Variables for error codes
int comErr, err;
//Blocksize for ScaLAPACK
int nb=256;
//Variables for the measurement of the time needed
double start, stop;

//Initialise MPI
MPI_Init(&argc,&argv);

//Determine the number of processes
MPI_Comm_size(MPI_COMM_WORLD, &np);

//Determine own process ID
MPI_Comm_rank(MPI_COMM_WORLD, &myId);

//Prepare output of process data into a logfile
ostringstream s;
s << datadir << "/" << "output" << myId << ".txt";
ofstream out(s.str().c_str());

//Write some information to the logfile
out << SetPrecision(23,15) << Scientific;   // Output format
out << "K=" << K << endl;
out << "m=" << m << endl;
out << "n=" << n << endl;

//Generate all needed prime numbers
generatePrimes();

//Right hand side is the first unit vector. Will be broadcast
//by process 0 to the other processes
if(myId==0) {
  b = 0.0;
  b[1] = 1.0;
}

//Start the solver
//
//Parameters are:
```

```
//generateElementOfA: A pointer to a function which generates
//                     Element (i,j) of the matrix A
//b: Right hand side of the system
//x: Solution of the system (will be overwritten by the solver)
//m: Number of rows of matrix A
//n: Number of columns of matrix A
//np: Number of available processes
//myId: Own process id
//nb: Block size for ScaLAPACK
//err: General error variable
//comErr: Communication-error variable
//out: Output stream to write status messages to
//K: Precision for dot products
//threads: Number of threads used for dot products
//LSS_ONLY_PART_ONE: A pre-defined constant determining that only
//                   part one of the solver should be executed
//                   (other possibilities: LSS_ONLY_PART_TWO, LSS_BOTH_PARTS)

start = getTime();

LSS(generateElementOfA,
    b, x, m, n, np, myId, nb, err, comErr,
    out, K, threads, LSS_ONLY_PART_ONE);

stop = getTime();

//Write overall time needed to logfile
out << endl << "Overall time needed:  " << stop-start << "s" << endl;

//Write result or error message to logfile
if(myId == 0) {
  if (!err) {
      out << "Average number of exact digits: "
  << computeNumberOfExactDigits(x) << endl;
      out << "Result: " << x[1] << endl;
  } else {
      out << "Error: " << LinSolveErrMsg(err) << endl;
  }
}

//Close the logfile and move it to the targetdir
out.close();
string syscmd_mvres= "mv "+ s.str() + " " + targetdir;
system(syscmd_mvres.c_str());

//End the program
```

```
  MPI_Finalize();

  return 0;
}
```

4 Parallel Computer Systems Used

Our software has been developed and tested on two very different high performance cluster computers at the Universities of Wuppertal and Karlsruhe, on which also the final computations have been done.

4.1 The system ALiCEnext at the University of Wuppertal

The following information is taken from
`http://www.alicenext.uni-wuppertal.de/hardware.htm`

ALiCEnext consists of 1024 AMD-Opteron processors as one would find in many standard desktop PCs. However, in ALiCEnext these processors are not located in usual PC cases but in towers with highly integrated technology (blade technology). They are connected by a double communication network, which allows super fast data exchange. Super-computers with this type of architecture are called clusters.

The Blades

1024 processors are distributed on 512 blades. On each of these blades are:

- 2 x AMD Opteron 1.8 GHz processors
- 2 x 250 GB harddiscs
- 2 x 1024 MByte RAM
- 6 x Gigabit-Ethernet connections

The Network

The network is a combination of gigabit-ethernet and 2-d tori. The additional 2-d torus is needed for computations in lattice gauge theory, because here mainly data from immediate neighbours is required by each process.

On this machine, the Intel Compiler 9.0 and the AMD Core Math Library (containing optimized implementations of BLAS, LAPACK and ScaLAPACK) were used. Our selfverifying solver for linear interval systems was first developed on this machine [25].

4.2 The system HP XC6000 at the University of Karlsruhe

The following information is taken from
`http://www.rz.uni-karlsruhe.de/ssck/4141.php`

Configuration of High Performance Computer HP XC6000

Nodes of HP XC6000 may have different roles. In Karlsruhe there are

- 2 login nodes each with 8 processors,
- 101 compute nodes each with 2 processors,
- 10 compute nodes each with 8 processors,
- a few service nodes for resource management etc.,
- 8 file server nodes each with 2 processors with a netto capacity of 10 TB and
- a fast Quadrics QsNet II Interconnect.

The HP XC6000 is a distributed memory parallel computer with 128 nodes all in all; 108 nodes consist of 2 Intel Itanium2 processors with a frequency of 1.5 GHz and 12 nodes consist of 8 Intel Itanium2 processors with a frequency of 1.6 GHz and all nodes own local memory, local disks and network adapters. Thus the theoretical peak performance of the system is 1.9 TFLOPS. The main memory above all compute node is about 2 TB. Additionally special nodes are attached as file servers to the cluster to support a fast and scalable parallel filesystem. All nodes are connected to the Quadrics QsNet II interconnect that shows a high bandwidth of more than 800 MB/s and a low latency.

The basic operating system on each node is HP XC Linux for High Performance Computing (HPC), a Linux implementation which is compatible to RedHat AS 4.0. On top of this operating system a set of open source as well as proprietary software components constitute the XC software environment enabling the efficient usage of the parallel computer.

As global filesystem the scalable, parallel filesystem Lustre is used within the XC cluster. By the usage of several Lustre Object Storage Target (OST) servers and Meta Data Servers (MDS) both high scalability and redundancy are reached in case of malfunction of single servers. At present 10 TB disk space is available within the global filesystem. Besides each node of HP XC6000 is equipped with local disks for temporary data.

More detailled description of the nodes: 108 nodes each with 2 Intel Itanium2 processors and 12 GB main memory and 146 GB local disk space; 12 nodes each with 8 Intel Itanium2 processors and 64 GB main memory and about 500 GB local disk space; 8 2-way file server nodes based on Xeon processors with attached disks of the overall size of 10 TB.

The operating system on each node is HP XC Linux for High Performance Computing. The C++ compiler is icc 10.0 and theversion of the Intel Math Kernel Library (which contains optimized implementations of BLAS, LAPACK and ScaLAPACK) is 10.0.011.

5 Interval Results and some Timing Information

Let us first discuss the results obtained for the original problem ($n = 20.000$). We find the interval enclosure

$A_{11}^{-1} = 0.725078346268401167468687719251160968869180594479508957878164769\ldots \in$ `[0.7250783462684010, 0.7250783462684012]`. This enclosure was reproduced in all cases on both machines even when using different numbers of processors. The value of A_{11}^{-1} is known exactly as a rational number in which the numerator and denominator each have 97389 decimal digits, see [24] or

`http://www-m3.ma.tum.de/m3old/bornemann/challengebook/Chapter7/sol7_numerator.txt`

and

`http://www-m3.ma.tum.de/m3old/bornemann/challengebook/Chapter7/sol7_denominator.txt`

On AliceNext, solving the problem using np= 20 processes took about 1800 seconds and using np=50 processes, it took about 870 sec. On the super computer HP XC6000 in Karlsruhe the corresponding timings are 620 and 280 seconds, respectively.

Problem with dimension $n = 50.000$: In this case we found the interval enclosure $A_{11}^{-1} \in$ `[ 7.250799024199662E-001, 7.250799024199664E-001]`. Again the computed result does not depend on the number of processors used.

On the high performance computer in Karlsruhe the timings using 50 and 100 processes are 3603 and 1958 sec, respectively.

The following two excerpts are taken from the logfiles produced in Wuppertal and Karlsruhe, respectively when executing the linear system solver in case of a system matrix with dimension $n = 20.000$ and using 50 processes. Comparing these data gives deeper insight in the main contributions to the overall computing times. These main contributions are (as expected) on both machines

- computing the approximate inverse R,
- and computing the enclosure [C] of $I - R \times A$.

There is a significant difference comparing the ratio of the time to compute R to the time to compute [C] on the different machines. This is most likely due to different levels of optimization of the used ScaLAPACK libraries but further investigations have to be done.
Here are the details concerning AliceNext:

```
K=2
Number of rows m=20000
Number of columns n=20000
Starting LSS...
Time for data distribution and initialization: 27.6366
Starting matrix inversion
Inversion finished
Time for inversion to compute matrix R: 430.164
Starting defect iteration...
 1. iteration step
 2. iteration step
Time for defect iteration: 3.85521
Computing [z]
```

```
Time for computation of [z]: 3.58533
Computing [C]
Time for computation of [C]: 402.01
Verifying...
 1. verification step
Time for verification: 2.12393
LSS finished.

Overall time needed:  869.747s
Average number of exact digits:  1.581288580850802E+001
Result: A^(-1)[1,1] lies in [ 7.250783462684010E-001, 7.250783462684012E-001]
```

The corresponding data concerning HP XC6000 are:

```
K=2
Number of rows m=20000
Number of columns n=20000
Starting LSS...
Time for data distribution and initialization: 23.6741
Starting matrix inversion
Inversion finished
Time for inversion to compute matrix R: 94.8271
Starting defect iteration...
 1. iteration step
 2. iteration step
Time for defect iteration: 3.40674
Computing [z]
Time for computation of [z]: 4.42973
Computing [C]
Time for computation of [C]: 149.153
Verifying...
 1. verification step
Time for verification: 2.34285
LSS finished.

Overall time needed:  278.179s
Average number of exact digits:  1.581288580850802E+001
Result: A^(-1)[1,1] lies in [ 7.250783462684010E-001, 7.250783462684012E-001]
```

6 Conclusion

To solve the Hundred-Dollar, Hundred-Digit Challenge Problem 7 is not very hard when using our parallel C-XSC solver. We get bounds for the correct mathematical results. The correctness of these bounds is proved automatically by the computer using (machine) interval calculations. Our open source parallel C-XSC solver for linear interval systems is a very powerful tool. It is also able to handle system matrices and right hand sides with

real or even with complex interval entries. The numerical results are always reliable. To the authors knowledge there is no other software available which is able to solve such large dense interval linear systems (please inform us if you are aware of other packages!). Of course, a lot of additional work has to be done to create algorithms and to implement corresponding software packages which are able to exploit sparsity. This is a very challenging task.

We will extend this note with some additional material concerning the solution of large ill conditioned dense matrices in the near future.

Acknowledgement

We want to thank Gerd Bohlender, Rudolf Lohner and Holger Obermaier for helping us to get our programs executed on the parallel machine at the Computing Center of Karlsruhe University.

References

[1] C. Hölbig, and W. Krämer. Selfverifying solvers for dense systems of linear equations realized in C-XSC. *BUW-WRSWT 2003/1*, 2003.

[2] T.J. Dekker. A floating-point technique for extending the available precision. *Numerische Mathematik*, 18(3):224–242, 1971.

[3] E. Anderson, E. and Bai, Z. and Bischof, C. and Blackford, S. and Demmel, J. and Dongarra, J. and Du Croz, J. and Greenbaum, A. and Hammarling, S. and McKenney, A. and Sorensen, D. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

[4] M. Kolberg, G. Bohlender, and D. Claudio. Improving the Performance of a Verified Linear System Solver Using Optimized Libraries and Parallel Computation. In 8^{th} *VECPAR - International Meeting on High Performance Computing for Computational Science*, Toulouse, France, 2008. To appear.

[5] M. Grimmer. *Selbstverifizierende Mathematische Softwarewerkzeuge im High Performance Computing. Konzeption, Entwicklung und Analyse am Beispiel der parallelen verifizierten Lösung linearer Fredholmscher Integralgleichungen zweiter Art.* Logos Verlag, 2007.

[6] M. Grimmer and W. Krämer. An MPI extension for verified numerical computations in parallel environments. *in: Arabnia et al. (editors): Proceedings to Int. Conf. on Scientific Computing (CSC'07, Worldcomp 07) Las Vegas*, pages 111–117, 2007.

[7] R. Hammer, M. Hocks, U. Kulisch, and D. Ratz. *Numerical Toolbox for Verified Computing I: Basic Numerical Problems*. SpringerVerlag, 1993.

[8] W. Hofschuster and W. Krämer. C-XSC 2.0: A C++ library for extended scientific computing. *in: R. Alt et al. (editors): Numerical Software with Result Verification, Lecture Notes in Computer Science, Springer Heidelberg, Volume 2991/20040*, pages 15–35, 2004.

[9] George Em Karniadakis and Robert M. Kirby II. *Parallel Scientific Computing in C++ and MPI*. Cambridge University Press, 2003.

[10] Donald E. Knuth. *The Art of Computer Programming Vol.2 - Seminumerical Algorithms, Third Edition*. Addison-Wesley, 1997.

[11] C. Hölbig, W. Krämer and D. Tiarajú. An accurate and efficient selfverifying solver for systems with banded coefficient matrix. *In: Parallel Computing: Software Technology, Algorithms, Architectures and Applications. 1 ed. Elsevier Science B.V. Amsterdam*, pages 283–290, 2004.

[12] W. Krämer and M. Zimmer. Fast (parallel) dense linear interval systems solving in C-XSC using error free transformations and BLAS. *To be published, Springer Lecture Notes in Computer Science*, 2008.

[13] U. Kulisch. *Computer Arithmetic and Validity*. de Gruyter, 2008.

[14] U.W. Kulisch and W.L. Miranker. The arithmetic of the digital computer: A new approach. *SIAM Rev.*, 28(1):1–40, 1986.

[15] W. Krämer, U. Kulisch and R. Lohner. *Numerical Toolbox for Verified Computing II: Advanced Numerical Problems*. SpringerVerlag, 2006.

[16] M. Kolberg, L.F. Fernandes, and D.Claudio. Dense Linear System: A Parallel Selfverified Solver. *International Journal of Parallel Programming, Springer Netherlands*, 2007.

[17] L.S. Blackford,, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997.

[18] N.Yamanaka, T.Ogita, S.M.Rump, and S.Oishi. A parallel algorithm of accurate dot product. `http://www.ti3.tu-harburg.de/paper/rump/RuZi07.pdf`.

[19] Takeshi Ogita, Siegfried M. Rump, and Shin'ichi Oishi. Accurate sum and dot product. *SIAM Journal on Scientific Computing*, 26(6):1955–1988, 2005.

[20] G. Bohlender, W. Walter, P. Kornerup, and D.W. Matula. Semantics for exact floating point operations. *Proceedings, 10th IEEE Symposium on Computer Arithmetic, IEEE*, pages 26–28, 1991.

[21] R. Klatte, U. Kulisch, A. Wiethoff, C. Lawo, M. Rauch. *C-XSC - A C++ Class Library for Extended Scientifix Computing.* SpringerVerlag, 1993.

[22] S.M. Rump. *Kleine Fehlerschranken bei Matrixproblemen.* PhD thesis, Universität Karlsruhe, 1980.

[23] N. Trefethen. *A Hundred-Dollar, Hundred-Digit Challenge.* SIAM, 2002.

[24] F. Bornemann, D. Laurie, S. Wagon, J. Waldvogel. *The SIAM 100-Digit Challenge - A Study in High-Accuracy Numerical Computing.* SIAM, 2004.

[25] M. Zimmer. Laufzeiteffiziente, parallele Löser für lineare Intervallgleichungssysteme in C-XSC. Master's thesis, Universität Wuppertal, 2007.