Development and acceleration of multiple precision arithmetic toolbox MuPAT for Scilab

Satoko Kikkawa\textsuperscript{1}, Tsubasa Saito\textsuperscript{1}, Emiko Ishiwata\textsuperscript{2} and Hidehiko Hasegawa\textsuperscript{3}

\textsuperscript{1}Graduate School of Science, Tokyo University of Science, 1-3 Kagurazaka, Shinjuku-ku, Tokyo 162-8601, Japan
\textsuperscript{2}Tokyo University of Science, 1-3 Kagurazaka, Shinjuku-ku, Tokyo 162-8601, Japan
\textsuperscript{3}University of Tsukuba, 1-2 Kasuga, Tsukuba-shi, Ibaraki 305-8550, Japan
E-mail j1411605@ed.tus.ac.jp
Received

Abstract

MuPAT enables the users to easily treat quadruple and octuple precision arithmetic as well as double precision arithmetic on Scilab. Using external C routines, we have also developed a high speed implementation MuPAT\textsubscript{c} for Windows, Mac OS, and Linux. MuPAT\textsubscript{c} reduced the computation time especially for all octuple precision arithmetic and inner product of quadruple precision arithmetic. MuPAT\textsubscript{c} can run 90~1200 times faster than MuPAT. We applied three different precisions to tridiagonalization by Lanczos method and confirmed that a high precision arithmetic was essential for Lanczos method to get accurate eigenvalues of real symmetric matrices.

Keywords Quad-Double, octuple precision arithmetic, Scilab, Lanczos tridiagonalization

Research Activity Group Algorithms for Matrix / Eigenvalue Problems and their Applications

1. Introduction

To analyze errors for construction of new numerical algorithms, easily usable high precision arithmetic is important for end users.

We have developed a quadruple and octuple precision arithmetic toolbox named MuPAT [1] (Multiple Precision Arithmetic Toolbox). Using MuPAT, the users can use the same operators and functions to double, quadruple and octuple precision numbers and mixed precision arithmetic is also available. MuPAT includes all the functions of a quadruple precision arithmetic toolbox QuPAT, which was proposed by Saito et al. [2].

To enable the users to use three different precision arithmetics at the same time, formulas should be expressed without numerical data types, and change of data types should be done dynamically. For these purpose, Scilab [3], a free and open source numerical software, was chosen to implement MuPAT. MuPAT works independently on any hardwares and operating systems since MuPAT is using pure Scilab functions. An interpreted language Scilab incurs a big overhead, but it is possible to accelerate its operations by using external routines written in C or Fortran.

We use DD [4] arithmetic for quadruple precision arithmetic and QD [5] arithmetic for octuple precision arithmetic. QD arithmetic needs tens or hundreds of double precision operations for an octuple precision operation, then it consumes hundreds or thousands of time compared to the double precision operation. To accelerate QD and DD operations, we have implemented MuPAT\textsubscript{c} using external C routines on Scilab. MuPAT\textsubscript{c} can also accelerate matrix or vector operations frequently used in numerical analysis. The computation speed of MuPAT\textsubscript{c} is 90~1200 times faster than that of MuPAT.

To confirm the effectiveness of MuPAT\textsubscript{c} and MuPAT, we applied double, quadruple and octuple precision arithmetics to eigenvalue computation. Lanczos method is often used for tridiagonalization but is known to lose orthogonality because of roundoff error. We compared 9 combinations of three different precision arithmetics for tridiagonalization by Lanczos method and eigenvalue computation by shifted QR method. We could get accurate eigenvalues only when we used octuple precision arithmetic for tridiagonalization, so it becomes clear that high precision arithmetic is essential for Lanczos method.

2. Features of DD and QD arithmetics

DD (Double-Double) is the way to represent a quadruple precision number with two double precision numbers and QD (Quad-Double) is the way to represent an octuple precision number with four double precision numbers.

QD number $A$ is represented as stated below with $a_0, a_1, a_2, a_3$ where $a_0, a_1, a_2, a_3$ are double precision numbers.

$$A = a_0 + a_1 + a_2 + a_3,$$

with

$$|a_{i+1}| \leq \frac{1}{2}\text{ulp}(a_i), \quad i = 0, 1, 2,$$

where ulp stands for ‘unit in the last place’. A DD number is 31 decimal digits and a QD number is 63 decimal...
3. MuPAT

We developed a quadruple and octuple precision arithmetic toolbox MuPAT (Multiple Precision Arithmetic Toolbox). MuPAT enables the users to use double, quadruple, octuple precision arithmetics with the same operators or functions, and mixed precision arithmetic and a partial use of different precision arithmetics are possible. We used DD arithmetic for quadruple precision arithmetic and QD arithmetic for octuple precision arithmetic.

3.1 Features of MuPAT

For MuPAT, we defined two types ‘dd’ and ‘qd’ as quadruple and octuple precision numbers using Scilab function ‘tlist’ and the data type ‘constant’ which is a double precision number in Scilab. Constant can treat matrices and vectors as well as scalar values, and dd and qd have the same feature. To make the same operators and functions available among double, quadruple and octuple precision numbers, we applied overloading to arithmetic operators and functions for dd and qd. For instance, the addition of two numbers can be done by inputting \(a+b\) whether the types of them are constant, dd or qd. An operation among dd numbers or a mixed precision operation among constant and dd numbers returns the result as dd. An operation among qd numbers or among constant, dd and qd numbers returns the result as qd. In MuPAT, the same functions as constant can be used even if the arguments are dd or qd. The function returns the result of the same precision type as the arguments. MuPAT includes the following functions: zeros for generating a zero matrix; eye for generating a unit matrix; rand for generating a random matrix; norm for returning a norm; lu for LU decomposition; qr for QR decomposition. \(A^T\) for transposition of \(A\) and insertion of matrix elements can be done in the same way as constant. Every function of MuPAT is written by Scilab language in a ‘sci’ file, and error processing and computation processing are done on it.

To use a high precision arithmetic in MuPAT, only a modification to definition of high precision numbers is needed. MuPAT is as easy for programming as Scilab. MuPAT is independent on any hardwares and operating systems because it is implemented using Scilab functions.

In Fig. 1, the relationship between Scilab and MuPAT is shown.

4. Acceleration of DD and QD arithmetics on Scilab

Table 1 shows that QD arithmetic needs tremendous double precision operations. Especially, one QD division requires 649 double precision operations, so the computation time requires hundreds times greater than double precision arithmetic on Scilab. To accelerate QD and DD arithmetic operations, we prepared external routines written by C language. These MuPAT functions achieved high-speed processing however they depend on a hardware and an operating system.

4.1 Implementation method

In Scilab, numeric data is treated as a matrix. The external routines are designed to be passed the initial address and the size of the data so that as many as operations can be done at the call.

The implementation method is shown in Fig. 2. The external C routine carries out DD arithmetic or QD arithmetic and the sci file is rewritten to call the external C routine. We implemented the external C routine to be passed its arguments by pointer. In Fig. 2, the arguments of the Scilab function \(a_0, a_1, a_2, a_3, m, n\) are passed by pointer \(*a_0, *a_1, *a_2, *a_3, *m, *n\) to the external C routine. The external C routine can treat not only scalar but also matrix or vector arithmetics since the type of \(a_0, a_1, a_2, a_3\) is constant.

We developed MuPAT.c for Windows, Mac OS and Linux. The required files are different on each operating system. On Windows, the sci files including calling functions and dll files including compiled code for arithmetics are required. On Mac OS and Linux, the sci files and .c files including code for arithmetics are required. C programs are compiled when the toolbox is built and

| Table 1. Number of double precision arithmetic operations |
|----------------|----------------|----------------|----------------|
|               |    add & sub |      mul |      div |   total |
| DD             |            |          |          |     |
| add & sub      |     11    |     0   |     0   |   11  |
| mul            |     15    |     9   |     0   |   24  |
| div            |     17    |     8   |     2   |   27  |
| QD             |            |          |          |     |
| add & sub      |     91    |     0   |     0   |   91  |
| mul            |    171    |    46   |     0   |  217  |
| div            |    579    |    66   |     4   |  649  |
function c = %qd_a_qd int qd_a_qd(arguments)
whose dimension is equal to 10
(iii) Inner product
(iv) Matrix-vector product

three trials.
(iv) repeatedly 10

version 5.3.3 running on Windows 7. We executed (i) to
out on Intel Core i5 2.5GHz, 4GB memory and Scilab

dynamic link library is created, and then C programs
get linked to Scilab functions. We use Microsoft Visual
C++ 2010 to compile code.

4.2 Computation time of MuPAT and MuPAT_c
Table 2 shows the computation time of MuPAT and
MuPAT_c in seconds and the ratio of time required for
DD and QD arithmetic to time required for double pre-
cision arithmetic on Scilab. All experiments are carried
out on Intel Core i5 2.5GHz, 4GB memory and Scilab
version 5.3.3 running on Windows 7. We executed (i) to
(iv) repeatedly 10^4 times. Each result is the average over
three trials.

(i) Scalar addition, subtraction, multiplication, and
division
(ii) Vector addition
(iii) Inner product
(iv) Matrix-vector product
whose dimension is equal to 10^8

In the case of QD basic arithmetics, the computation
time in MuPAT is 241×2317 times greater than that for
double precision arithmetic, and that in MuPAT_c is 14
×24 times greater than that for double precision arith-
metic. The computation time for QD division is 2317
times greater than that for double precision arithmetic
and it becomes 24 times in MuPAT_c.

Using MuPAT_c, the computation time of inner pro-
duct is improved from 9859 times to 52 times greater than
that for double precision arithmetic for DD, and from
270273 times to 223 times for QD. The computation
time of Matrix-vector product for DD is 28 times and
that for QD is 278 times greater than that for double
even in MuPAT.

MuPAT_c can run 90–1200 times faster than Mu-
PAT. MuPAT_c is implemented to reduce the computa-
tion time efficiently when more operations are executed
at one call. Therefore, QD division and inner product
can significantly reduce the computation time since they
need tremendous double precision operations.

To confirm a calling overhead, we compared (a) and
(b) for DD and QD arithmetics.

(a) Scalar addition : (x + y) repeats 10^6 times
(b) Vector addition : x + y whose dimension is equal to
10^6

In the case of DD, it took 25.9 seconds for (a) and
0.10 seconds for (b), then a calling overhead is about
2.6 × 10^{-5} seconds. In the case of QD, it took 32.5 sec-
onds for (a) and 0.31 seconds for (b), then a calling over-
head is about 3.6 × 10^{-5} seconds. The users should be
encouraged to use matrix or vector operations rather
than use scalar operations to avoid calling overheads.

5. Arithmetic precision for tridiagonal-
ization by Lanczos method

To verify the effectiveness of fast high precision arith-
metics, we applied DD and QD arithmetics to eigenvalue
computation for a real symmetric n×n matrix A.

To compute the eigenvalues of a real symmetric ma-
trix A, the matrix A is tridiagonalized to an equivalent
tridiagonal matrix, then the eigenvalues of the tridiago-
nal matrix are computed. Lanczos method can construct
an equivalent tridiagonal matrix as generating orthogo-
nal bases one after another. Lanczos method is useful for
a large sparse matrix since it isn’t necessary to modify
the matrix A, but roundoff error causes Lanczos vectors
to lose orthogonality [6].

In this section, we analyzed arithmetic precision for
tridiagonalization by Lanczos method and eigenvalue
computation by shifted QR method for the tridiagonal
matrix. Table 3 shows 9 combinations of using double,
quadruple and octuple precision arithmetics for each of
the tridiagonalization and the eigenvalue computation
where D, Q and O stands for double, quadruple and
octuple precision arithmetics respectively. We assumed
that the true solution is equal to the computation result
produced by the function ‘Eigenvalues’ of Mathematica.

![Table 2. Computation time in seconds; the ratio is between parentheses](image)

<table>
<thead>
<tr>
<th></th>
<th>x + y (s)</th>
<th>(i)</th>
<th>x y (s)</th>
<th>(ii)</th>
<th>x + y (s)</th>
<th>(iii)</th>
<th>xy (s)</th>
<th>(iv)</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>0.022</td>
<td>(1)</td>
<td>0.017</td>
<td>(1)</td>
<td>0.01</td>
<td>(1)</td>
<td>0.02</td>
<td>(1)</td>
</tr>
<tr>
<td>DD</td>
<td>MuPAT</td>
<td>0.22</td>
<td>(10.4)</td>
<td></td>
<td>0.27</td>
<td>(12.1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MuPAT_c</td>
<td>0.41</td>
<td>(19.7)</td>
<td></td>
<td>0.31</td>
<td>(18.2)</td>
<td>0.66</td>
<td>(66.0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.43</td>
<td>(25.6)</td>
<td></td>
<td>0.69</td>
<td>(69.0)</td>
<td>197.18</td>
<td>(9859.0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.66</td>
<td>(66.0)</td>
<td></td>
<td>1.05</td>
<td>(52.5)</td>
<td>778.91</td>
<td>(18.5)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.69</td>
<td>(69.0)</td>
<td></td>
<td></td>
<td></td>
<td>11716.7</td>
<td>(278.8)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.31</td>
<td>(241.5)</td>
<td>4.46</td>
<td>(223.0)</td>
<td></td>
<td>4339.42</td>
<td>(105.6)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.01</td>
<td>(1)</td>
<td>0.02</td>
<td>(1)</td>
<td>42.04</td>
<td>(1)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>8.39</td>
<td>(533.0)</td>
<td>39.39</td>
<td>(2317.6)</td>
<td></td>
<td>5405.47</td>
<td>(270273.5)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.39</td>
<td>(25.6)</td>
<td></td>
<td>3.26</td>
<td>(23.6)</td>
<td>2.36</td>
<td>(10.4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.94</td>
<td>(236.0)</td>
<td></td>
<td>3.94</td>
<td>(236.0)</td>
<td>3.94</td>
<td>(236.0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4.66</td>
<td>(69.1)</td>
<td></td>
<td>4.66</td>
<td>(69.1)</td>
<td>4.66</td>
<td>(69.1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.97</td>
<td>(197.2)</td>
<td></td>
<td>5.97</td>
<td>(197.2)</td>
<td>5.97</td>
<td>(197.2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>39.39</td>
<td>(2317.6)</td>
<td></td>
<td>39.39</td>
<td>(2317.6)</td>
<td>39.39</td>
<td>(2317.6)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>7.39</td>
<td>(739.0)</td>
<td></td>
<td>7.39</td>
<td>(739.0)</td>
<td>7.39</td>
<td>(739.0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11716.7</td>
<td>(278.8)</td>
<td>11716.7</td>
<td>(278.8)</td>
<td>11716.7</td>
<td>(278.8)</td>
<td></td>
</tr>
</tbody>
</table>

![Fig. 2. Argument types of each routine](image)
5.1 Comparison of arithmetic precision

We tested ‘bcsstk02’ and ‘nos4’ from MatrixMarket (http://math.nist.gov/MatrixMarket/) with the initial vector \( v = (0, 1, 0, \ldots, 0)^T \) for Lanczos method. For bcsstk02, the dimension is 66, the number of nonzero entries is 2211, and the condition number is \( 4.3 \times 10^4 \). For nos4, they are 100, 347, 1.6 \times 10^3 respectively. Matrices have no multiple eigenvalues.

First, we computed

\[
\max_{1 \leq i \leq n} |\lambda_i - \bar{\lambda}_i| \quad (\bar{\lambda} = O_D, O_Q, O_0)
\]

where \( \lambda^{O_D}, \lambda^{O_Q}, \lambda^{O_0} \) and \( \bar{\lambda} \) represent the eigenvalues of \( O_D, O_Q, O_0 \) and the true solution respectively. The error of each result was \( 9.8 \times 10^{-11} \) for bcsstk02 and \( 2.0 \times 10^{-15} \) for nos4 at the maximum and there was not much difference among \( \lambda^{O_D}, \lambda^{O_Q}, \lambda^{O_0} \). There was not also much difference among \( \lambda^{D_D}, \lambda^{Q_Q}, \lambda^{D_D} \) or among \( \lambda^{Q_D}, \lambda^{O_0}, \lambda^{O_Q} \). This means that using a high precision arithmetic for eigenvalue computation does not affect to the final results.

Next, we fixed the arithmetic precision to \( D \) for eigenvalue computation and compared \( O_D, Q_D, D_D, D_D \) changing the precision for tridiagonalization by Lanczos method. Fig. 3 and Fig. 4 illustrate the difference between \( \lambda \) and \( \bar{\lambda} \). (\( \lambda, \bar{\lambda} \)) are plotted where \( \lambda \) is the result of each \( O_D, Q_D, D_D, D_D \). The closer the point is to the dotted line \( \lambda = \bar{\lambda} \), the more accurate \( \lambda \) is.

For bcsstk02 and nos4, the absolute error of \( \lambda^{O_D} \) was \( 9.8 \times 10^{-11} \) and \( 1.0 \times 10^{-15} \) at the maximum, so every eigenvalue of \( O_D \) was nearly equal to \( \bar{\lambda} \). Using \( D_D \) or \( Q_D \), some multiple eigenvalues appeared.

Accurate eigenvalues were obtained by using octuple precision arithmetic for tridiagonalization by Lanczos method however they couldn’t be obtained by using double or quadruple precision arithmetic. It is important to use higher precision, especially octuple precision arithmetic for tridiagonalization by Lanczos method in cases of bcsstk02 and nos4. On the other hand, double precision arithmetic is enough for computing eigenvalues of a tridiagonal matrix since a high precision arithmetic had no effect to the results. Comparison of the Lanczos method with reorthogonalization in double precision and simple Lanczos process in higher precision is one of our future works.

6. Conclusion

We have developed a Multiple Precision Arithmetic Toolbox MuPAT on Scilab. In MuPAT, quadruple and octuple precision arithmetics can be treated as well as double precision arithmetic.

MuPATc is a high-speed version using external routines written by C language. To reduce overheads, external C routines are implemented to be passed matrix or vector values by pointer so that one call can carry out as many operations as possible. MuPATc reduces the computation time for QD arithmetics from \( 241\sim 270 \) to \( 14\sim 236 \) times greater than that for double precision arithmetic, and the computation time for DD arithmetics from \( 10\sim 9859 \) to \( 12\sim 69 \) times.

We analyzed arithmetic precision for tridiagonalization using MuPATc. We compared 9 combinations of using double, quadruple, and octuple precision arithmetics for tridiagonalization by Lanczos method and eigenvalue computation by shifted QR method. Accurate eigenvalues were obtained only by using octuple precision arithmetic for tridiagonalization. It becomes clear that a higher precision arithmetic is essential for tridiagonalization by Lanczos method.

To use MuPAT and MuPATc, only a modification to definition of high precision numbers is required. MuPAT and MuPATc are efficient toolboxes for mixed precision arithmetic, thus they should be important for numerical analysis.

References