Effectiveness of sparse data structure for double-double and quad-double arithmetics

Tsubasa Saito[1], Satoko Kikkawa[1], Emiko Ishiwata[2], and Hidehiko Hasegawa[3]

[1] Graduate School of Science, Tokyo University of Science, Japan
[2] Tokyo University of Science, Japan
[3] University of Tsukuba, Japan

Abstract. Double-double and quad-double arithmetics are effective tools to reduce the round-off errors in floating-point arithmetic. However, the dense data structure for high-precision numbers in MuPAT/Scilab requires large amounts of memory and a great deal of the computation time. We implemented sparse data types `ddsp` and `qdsp` for double-double and quad-double numbers. We showed that sparse data structure for high-precision arithmetic is practically useful for solving a system of ill-conditioned linear equation to improve the convergence and obtain the accurate result in smaller computation time.

Keywords:ill-conditioned matrix problem, sparse matrix, multiple precisions

1 Introduction

In floating-point arithmetic, we cannot avoid the computation errors. Therefore, for example, it is known that the iterative method for solving a system of ill-conditioned linear equation may not converge when double-precision arithmetic is used. Double-double and quad-double arithmetics facilitate the use of high-precision arithmetic on ordinary double-precision arithmetic environment. We have developed MuPAT[1, 2], which is 'Multiple Precision Arithmetic Toolbox' on Scilab[3], and have shown the effectiveness of double-double and quad-double arithmetics for ill-conditioned problems[4]. MuPAT has only dense data structures. Because of the large amount of memory and much more computation time, double-double and quad-double arithmetics cannot be applied for large matrices.

We developed sparse data structures for quadruple and octuple-precision arithmetics as a part of MuPAT. This implementation enables the users to treat large matrices with lower memory consumption and small computation time. We defined new data types for a sparse matrix which have double-double and quad-double numbers, and made it possible to use a combination of double, double-double, and quad-double arithmetics for both dense and sparse data structures.

We compared the memory consumption and the computation time of the matrix computations with sparse and dense data structures for double-double and quad-double arithmetics. We also showed that high-precision sparse data structure is practically useful for ill-conditioned matrices by applying double, double-double and quad-double arithmetics for the Biconjugate Gradient (BiCG) method.

2 Double-double and quad-double

Double-double and quad-double arithmetics were proposed for quasi-quadruple-precision and quasi-octuple-precision arithmetics by Hida et al.[5]. A double-double number is represented by two, and a quad-double number is represented by four, double-precision numbers. A double-double number $x_{(dd)}$ and a quad-double number $y_{(qd)}$ are represented by an unevaluated sum of double-precision numbers $x_0, x_1, y_0, y_1, y_2, y_3$ as follows:

$$x_{(dd)} = x_0 + x_1, \ \ y_{(qd)} = y_0 + y_1 + y_2 + y_3,$$

where $x_0, x_1, y_0, y_1, y_2, y_3$ satisfy the following inequalities:

$$|x_1| \le \frac{1}{2}\mathrm{ulp}(x_0), \ \ |y_{i+1}| \le \frac{1}{2}\mathrm{ulp}(y_i), \ \ i = 0, 1, 2,$$

where ulp stands for ‘units in the last place’. A double-double(quad-double) number has 31(63) significant decimal digits. They can be computed by using only double-precision arithmetic operations (see [5, 6] for details).

In Scilab, double-precision numbers are defined by the data type named `constant`. Scalars, vectors and matrices are treated in the same way as `constant`. In MuPAT, double-double and quad-double numbers are defined as data types named `dd` and `qd`, consisting of two or four `constant` data. We can use `constant`, `dd` and `qd` types at the same time, with the same operators ($+, -, *, /$) and the same functions such as `abs`, `sin` and `norm`.

3 Sparse data structure for MuPAT

MuPAT has only the dense data structures of the three data types `constant`, `dd`, and `qd`. Sparse data structure is important to reduce the memory consumption and the computation time. Especially using double-double and quad-double arithmetics, sparse data structures are more important because they require twice or four times memories compared with double-precision arithmetic, and also require much more computation time. We developed the sparse data structures for double-double and quad-double arithmetics with considering the following points;

- The same arithmetic operators ($+, -, *$) can be used among these data types.
- Operations for sparse and dense data in different precision numbers are available at the same time.

The users can compare problems in different precisions with lower memory consumption and small computation time.

3.1 Sparse data structure for double precision number

Sparse data structure stores non-zero entries with its row and column indices. In Scilab, the following matrix

$$a = \begin{pmatrix} 0 & 0 & 9 & 0 \\ 0 & 0 & 7 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 8 \end{pmatrix}$$

can be represented by a sparse data type `sparse` as follows.

```
  a  =
(    4,    4) sparse matrix
(    1,    3)        9.
(    2,    3)        7.
(    2,    4)        1.
(    3,    1)        1.
(    4,    4)        8.
```

The first line (4, 4) means the size of the matrix. The row and column indices and values of the matrix are stored after line 2. The entries are stored row-by-row. The same arithmetic operators $(+, -, *)$ for `constant` can be used for `sparse`, and mixed operations between `constant` and `sparse` are also allowed. The results of these binary operations become `constant` or `sparse` depending on the operations.

3.2 Sparse data structures for double-double and quad-double numbers

To treat high-precision arithmetic for sparse data structure, we defined two new sparse data types; one is `ddsp` for double-double numbers, and the other is `qdsp` for quad-double numbers. These data types are based on CCS (Compressed Column Storage) format, which contains some vectors; row index, column pointer and values. `ddsp` has two and `qdsp` has four value vectors to represent a double-double number and a quad-double number respectively. By these definitions of data types, MuPAT has six data types: `constant`, `dd`, `qd` for dense data and `sparse`, `ddsp`, `qdsp` for sparse data of double, double-double and quad-double numbers respectively.

3.3 Definition of matrix operators

Now we have three sparse data types `sparse`, `ddsp`, and `qdsp`. To enable the use of the same matrix operators $(+, -, *)$ for these data types, operator overloading was applied to perform arithmetic operations among every existing data types `constant`, `dd`, and `qd`, and sparse data types `sparse`, `ddsp`, and `qdsp`.

In many cases, the sparsity cannot be kept after sparse matrix operations. Especially for sparse matrix multiplication, the result tends to have many non-zero entries and become a dense matrix. Therefore, we should allocate memory space dynamically.

3.4 Functions for sparse matrix

Some functions for `sparse` are extended to `ddsp` and `qdsp`. For example, `full` for changing a sparse data type into a dense data type, and `nnz` for returning the number of non-zero entries, and so on can be used in the same syntax among `sparse`, `ddsp`, and `qdsp`. `A'` for transposition of `A` and insertion and extraction of matrix elements can be performed in the same syntax for all data types.

4 The memory consumption and the computation time

To confirm the effectiveness of a sparse matrix computation with using `ddsp` and `qdsp`, we compared the memory consumption and the computation time between sparse and dense data structures. All experiments were carried out on Intel Core i5 1.7GHz, 4GB memory and Scilab version 5.3.3 running on Mac OS X Lion.

4.1 Memory consumption

We prepared some 1000×1000 random sparse matrices with different sparsity for `constant`, `dd`, `qd`, `sparse`, `ddsp`, and `qdsp`. The sparsity patterns are random. Table 1 shows the sparsity and the memory consumption of each matrix. If the sparsity is less than 66% for double-double number or 80% for quad-double number, the memory consumptions of the sparse data structures are smaller than that of the dense data structures.

Table 1 : Memory consumption

Matrix		Memory (MB)					
	Sparsity	`constant`	`sparse`	`dd`	`ddsp`	`qd`	`qdsp`
A	1%	8.00	0.12	16.00	0.25	32.00	0.41
B	5%	8.00	0.60	16.00	1.21	32.00	2.01
C	10%	8.00	1.21	16.00	2.41	32.00	4.01
D	66%	8.00	7.92	16.00	15.85	32.00	26.40
E	80%	8.00	9.60	16.00	19.21	32.00	32.01

4.2 Matrix operations

Using the matrices in Table 1, we measured the following matrix operations.

- Matrix vector product $A\boldsymbol{x}$, $B\boldsymbol{x}$, $C\boldsymbol{x}$
- Matrix addition $A + B$, $B + C$, $C + A$
- Matrix multiplication AB, BC, CA

We executed each operation repeatedly 100 times. Table 2 and Table 3 show the results.

Matrix vector product The computation time of matrix vector product for `ddsp` is 141.5 times smaller than that of `dd` when the sparsity of the matrix is 1% and 13.0 times faster when the sparsity is 10%. The computation time for `qdsp` is 134.6 times smaller than that of `qd` when the sparsity is 1% and 14.0 times faster when the sparsity is 10%.

Matrix addition The computation time of matrix addition for `ddsp` is 10.0 times and 3.7 times smaller than that of `dd` when the sparsity is 6% and 15% respectively. The computation time for `qdsp` is 13.7 times and 5.5 times smaller than that of `qd` when the sparsity is 6% and 15% respectively. When the sparsity of the result is more than 66% for double-double number or 80% for quad-double number, the memory usage of sparse data types `ddsp` and `qdsp` are larger than that of dense data types `dd` and `qd`.

Table 2 : Results of matrix operations (Memory)

	Sparsity	Memory (MB)	
		ddsp	qdsp
$A\boldsymbol{x}$	-	-	-
$B\boldsymbol{x}$	-	-	-
$C\boldsymbol{x}$	-	-	-
$A+B$	6%	1.43	2.39
$C+A$	11%	2.63	4.39
$B+C$	15%	3.49	5.82
AB	40%	9.51	15.98
CA	63%	15.17	25.17
BC	99%	23.86	39.73

Table 3 : Results of matrix operations (Time)

	Time (sec.)					
	dd	ddsp	dd/ddsp	qd	qdsp	qd/qdsp
$A\boldsymbol{x}$	4.10	0.03	141.5	20.73	0.15	134.6
$B\boldsymbol{x}$	4.13	0.14	30.2	20.76	0.74	28.0
$C\boldsymbol{x}$	4.10	0.32	13.0	20.81	1.49	14.0
$A+B$	6.36	0.64	10.0	15.97	1.16	13.7
$C+A$	6.40	1.25	5.1	15.66	2.14	7.3
$B+C$	6.35	1.69	3.7	15.85	2.90	5.5
AB	2245.59	5.21	430.9	14909.50	12.27	1214.7
CA	2288.42	8.10	282.6	14964.51	20.84	718.1
BC	2282.17	16.54	138.0	14954.12	71.61	208.8

Matrix multiplication The sparsity may be increased in matrix multiplication. In case of quad-double arithmetic, the computation result of BC by using a dense data type `qd` requires 32MB memory. On the other hand, the result by using a sparse data type `qdsp` requires 40MB memory. However, the computation times using `qd` and `qdsp` are 14954.1 seconds and 71.6 seconds respectively. The computation time for `qdsp` is 208.8 times smaller than that of `qd`. In case of AB, AB keeps low sparsity, and the computation time of `ddsp` is 430.9 times and `qdsp` is 1214.7 times smaller than that of `dd` and `qd` respectively.

5 Using high-precision arithmetic with sparse data structure for ill-conditioned problems

We show the effectiveness of sparse data structure for high-precision arithmetic on Scilab by applying the Biconjugate Gradient (BiCG) method for ill-conditioned matrices. Theoretically, the BiCG method, which is one of the Krylov subspace method, converges after at most n iterations, where n is the dimension of the matrix[7]. However, in floating-point arithmetic, the norm of the residual may diverge and oscillates, and then the iteration process may not converge. Sometimes it may require more than n iterations.

The iteration was started with $\boldsymbol{x}_0 = \boldsymbol{0}$ and the right-hand side vector $\boldsymbol{b}$ was given by substituting the solution $\boldsymbol{x}^* = (1, 1, ..., 1)^\top$ into $\boldsymbol{b} = A\boldsymbol{x}^*$. Stopping criterion was $||\boldsymbol{r}_k||_2 \leq 10^{-12}\,||\boldsymbol{r}_0||_2$. The initial shadow residual was $\boldsymbol{r}_0^* = \boldsymbol{r}_0$. Iteration process was terminated at 10^4 iterations if it did not converge.

We took up four ill-conditioned test sparse matrices from [8]. These matrices are constructed double-precision numbers, then lower components of double-double and quad-double numbers are filled with zero. Condition numbers were obtained using the Scilab function `cond` in double-precision. Table 4 shows the list of test matrices.

Table 4 : Properties of test matrices

Matrix	Dimension	Non-zero	Sparsity	Condition number
west0497	497	1,721	0.70%	4.62×10^{11}
gre_1107	1,107	5,664	0.46%	3.19×10^{7}
tols2000	2,000	5,184	0.13%	5.99×10^{6}
sherman3	5,005	20,033	0.08%	3.49×10^{18}

Table 5 shows the results for double (D), double-double(DD), and quad-double (QD). 'Iterations' denotes the number of iterations required for convergence, 'Residual' denotes the relative residual norm $||\boldsymbol{r}||_2/||\boldsymbol{r}_0||_2$ and 'Error' denotes the relative error norm $||\boldsymbol{x} - \boldsymbol{x}^*||_\infty/||\boldsymbol{x}^*||_\infty$. `constant/sparse` is abbreviated to '`c/s`'. The values of 'Residual' and 'Error' were obtained by using sparse data structures.

Using double-precision arithmetic, the BiCG method did not converge for all matrices. Especially, for west0497 and gre_1107, the BiCG method converged by only using quad-double arithmetic. For sherman3, even if the BiCG method converged by using double-double arithmetic, the number of iteration became more than n. Using quad-double arithmetic, the convergence improved, and the number of iteration decreased and became less than n. High-precision arithmetic produces great improvement and enables us to obtain the accurate result that cannot be obtained by double-precision arithmetic.

However, using dense data types `dd` and `qd`, iteration process requires a great deal of the computation time. Sparse data types `ddsp` and `qdsp` can save the computation time. For sherman3, the computation time of `ddsp` is 680 times smaller than that of `dd`. Thus, high-precision sparse data structure provides more accurate results with practicable computation time . In case of dense data structure, sherman3 could not be stored by a quad-double number because of Out of Memory error. High-precision sparse data structure is also important in terms of the memory consumption.

An improvement of the accuracy by high-precision arithmetic depends on the problems and the methods. Although double-double and quad-double arithmetics do not perform well for all problems, high-precision sparse data structures surely increase the number of problems which can be solved accurately.

Table 5 : Computation results

	Matrix				Time (sec.)		
		Iterations	Residual	Error	`constant`	`sparse`	`c/s`
D	west0497	†	1.02e+02	3.70e+05	39.1	2.8	13.8
	gre_1107	†	6.97e+03	1.69e+04	278.7	4.1	68.7
	tols2000	†	8.06e+02	2.34e+06	998.6	4.7	211.7
	sherman3	†	1.73e-03	6.24e-01	6749.1	11.7	577.6
	Matrix				Time (sec.)		
		Iterations	Residual	Error	`dd`	`ddsp`	`dd/ddsp`
DD	west0497	†	2.18e-01	7.73e+02	303.7	15.0	20.2
	gre_1107	†	2.40e-01	9.08e-01	1828.9	21.2	86.2
	tols2000	1586	9.29e-13	3.55e-09	938.3	4.1	228.7
	sherman3	7696	9.98e-13	1.05e-13	31227.4	45.8	681.9
	Matrix				Time (sec.)		
		Iterations	Residual	Error	`qd`	`qdsp`	`qd/qdsp`
QD	west0497	2676	6.09e-13	3.50e-08	306.6	7.0	43.84
	gre_1107	3401	8.59e-13	3.05e-11	2136.2	17.6	121.3
	tols2000	1080	6.77e-13	1.96e-09	2342.8	7.1	328.5
	sherman3	4884	9.35e-13	1.73e-13	–	91.1	

† : More than 10^4 iterations, – : Out of Memory

6 Conclusion

We developed the sparse data structures for quadruple-precision and octuple-precision arithmetics in MuPAT/Scilab, and showed that high-precision sparse data structure is practicable for solving a system of ill-conditioned linear equation.

MuPAT covers all arithmetic operators for double, double-double, and quad-double numbers for both dense and sparse data structures. Using MuPAT, six data types are available at the same time and operations for mixed-precision and mixed data structure are also available by the same operators and functions. To use double-double and quad-double arithmetics with lower memory consumption and smaller computation time, only a modification to definition of numbers is needed.

The memory consumption of the sparse data types is smaller for a matrix whose sparsity is less than 66% for double-double number or 80% for quad-double number. In matrix vector product, the computation time of sparse data structures for a double-double number and a quad-double number are 141.5 times and 134.6 times smaller than that of dense data structures respectively, when the sparsity of the matrix is 1%. In matrix addition, the computation time of sparse data structure for a double-double number and a quad-double number are 10.0 times and 13.7 times smaller than that of dense data structures respectively, when the sparsity of the result is 6%. In matrix multiplication, even if the result becomes a dense matrix whose sparsity is more than 99% and need much memory, the computation time can be reduced.

We investigated the convergency of the BiCG method for ill-conditioned matrices. Double-double and quad-double arithmetics are crucial to improvement of the accuracy. However, dense data structures for double-double and quad-double numbers and arithmetics require large amounts of memory and considerably long computation time. For some situations, a matrix cannot be stored by a quad-double number because it requires four times as large memory as a double-precision number. High-precision sparse data structure facilitates the pragmatic problems of these restriction. Using sparse data structure, the computation time became 20 ~ 680 times smaller than using dense data structure. Sparse data structure for double-double and quad-double arithmetics is practicable way to improve the convergence for ill-conditioned matrices, and to increase the number of problems that can be solved.

References

[1] MuPAT, http://www.mi.kagu.tus.ac.jp/qupat.html

[2] S. Kikkawa, T. Saito, E. Ishiwata and H. Hasegawa, Development and acceleration of multiple precision arithmetic toolbox MuPAT for Scilab, JSIAM Letters, Vol. 5, 9-12(2013).

[3] Scilab, http://www.scilab.org/

[4] T. Saito, E. Ishiwata and H. Hasegawa, Analysis of the GCR method with mixed precision arithmetic using QuPAT, J. Comput. Sci., Vol. 3, 87-91(2012).

[5] Y. Hida, X. S. Li and D. H. Bailey, Quad-double arithmetic: Algorithms, Implementation, and application. Technical Report LBNL-46996, Lawrence Berkeley National Laboratory, Berkeley, CA 94720 (2000).

[6] T. J. Dekker, A Floating-Point Technique for Extending the Available Precision, Numer. Math. Vol. 18, 224-242(1971) .

[7] R. Barrett et al., Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition, SIAM, Philadelphia (1994).

[8] The University of Florida Sparse Matrix Collection, http://www.cise.ufl.edu/research/sparse/matrices/