User's Guide for MuPAT on MATLAB

Hotaka Yagi

Apr 24 2019

Contents

1 Introduction

The multiple precision arithmetic toolbox MuPAT[1] implemented on Matlab can be used pseudo-quadruple-precision numbers (double-double)[2] and pseudo-octuple-precision numbers (quad-double)[3]. It is prepared for each OS independently (currently only MacOS). MuPAT allows uers to easily handle the above high-precision operations interactively using the MATLAB command window, and with same MATLAB codes. Furthermore, MuPAT is applied parallel processing to vector and matrix operations. As the parallel processing features, FMA, AVX2 and OpenMP are used. For more information on parallel processing, please refer to the Parallel processing section in this User Guide.

1.1 Syetem requirement

MuPAT is fully functional with the following environment.
MATLAB R2017a or more later
CPU: intel Haswell or more later， amd processer is suveying now...

2 How to start MuPAT

2.1 Install

After downloading zip file of MuPAT from the web page https://www.ed.tus.ac.jp/1419521/or from GitHub https://github.com/HotakaYagi/MuPAT, unzip and move MuPAT to a directory that is easy to work, such as under the MATLAB directory.

2.2 Start-up

1. Start MATLAB
2. Move current directory to MuPAT
3. Enter the following command to the MATLAB command window
>>startMuPAT(n1, n2, n3)

※ MuPAT can select thread number, AVX2 on / off, FMA on / off as input arguments. (No arguments means the number of threads = 1, FMA, and AVX2 are set off)

Table 2.1: The input arguments for startMuPAT

	the number of threads (n1)	AVX2 (n2)	FMA (n3)
Speed up ON	$\{n \mid n \in \mathbb{N}, n > 1\}$	1	1
Speed up OFF	1(serial) or No arguments	0 or No arguments	0 or No arguments

Ex.
startMuPAT: Compute with 1 threads, AVX2 disenabled, FMA disenabled (serial computing)
startMuPAT (2, 1, 1): Compute with 2 threads, AVX2 enabled, FMA enabled.

After completing the above steps, enter the following command to the MATLAB command window, and if the following result is output, MATLAB is ready to use MuPAT.
>>a = DD(1)
a =
hi: 1
lo: 0

2.3 Remarks

Be aware of the required memory size for the input you are dealing with on your machine. MATLAB's mex function may kill MATLAB if the requested memory size is somewhat larger. Furthermore, in parallel

computing, the computation order differs from the serial computing. The computation results may not match completely as seiral computing, because they have different rounding errors.

3 Usage

In MuPAT, we can use quadruple and octuple precision arithmetics almost the same way as double precision arithmetic, because MATLAB can define new data type and apply overloading for operators and some MATLAB funcitons. The difference with ordinary operation in MATLAB is that the declaration of variab of type DD and QD (section 3.1) (or conversion (section 3.2)), the ways of input and output (section 3.3), and some of DD and QD function (Section 3.4). First, I will describe the operation that are different from MATLAB, and then describe that can be performed by the same operation as MATLAB.

3.1 Variable definition

All numbers in MATLAB are treated as doubles. In MuPAT, we define new data types named DD and QD to contain double-double and quad-double numbers. These type require two or four double precision numbers. type DD has two double numbers of hi (upper part) and lo (lower part), and type QD has four double numbers of hh, hl, lh and ll from the highest part to the lowest part respectively. The upper part of DD variable A can be referenced with A.hi, and the lower part of A is with A.lo. QD variable B can be referenced with B.hh, B.hl, B.lh, and B.ll from the highest part to the lowest part respectively.

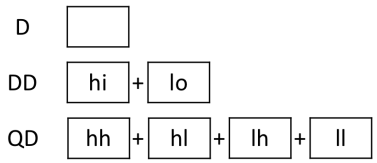

Figure 3.1: Images of type DD and type QD

To define type DD, you need to use the following function. The DD function need up to two arguments. The type of arguments does not matter.
>> DD(a, b)
· DD(), if there is no argument, the value 0 is assigned to the upper part and the lower part.
· DD (a), if only one argument is input, a is assigned to the upper part and the lower part is assigned the value 0.
· When two arguments are input, the input values are assigned to upper part and lower part respectively.

To define type QD, you need to use the following function. The QD function need up to four arguments. The type of arguments does not matter.
>> QD(a, b, c, d)
· QD (), if there is no argument, the value 0 is assigned to from the highest part to the lowest part.
· QD (a), if only one argument is input, a is assinged to the highest part, and the value 0 is assigned to the others.
· QD (a, b), if two arguments are input, a and b are assigned to the upper two places, and the value 0 is assigned to the lower two places.
· QD (a, b, c), if there are three arguments, a, b and c are assigned to the upper three places, and value 0 is assigined to the lowest part.
· When four arguments are input, the input values in a, b, c and d are assigned to from the highest part to the lowest part respectively.

The components of DD and QD are substituted in descending (renormalized) order.
Ex.
>> a = DD(1, 0)

```
a =
hi: 1
lo: 0
>> b = QD(3.14, 0, 0, 0)
b =
hh: 3.14
hl: 0
lh: 0
ll: 0
>> A = [1, 2, 3; 4, 5, 6];
>> AA = DD(A)
AA =
hi: [2x3 double]
lo: [2x3 double]
>> AA.hi
ans =
1 2 3
4 5 6
>> AA.lo
ans =
0 0 0
0 0 0
```

3.2 Type conversion

MuPAT can convert variables of data types (double a, DD b, QD c) with the following function.

Table 3.1: The way to type conversion

from\to	D	DD	QD
D		DD(a)	QD(a)
DD	double(b)		QD(b)
QD	double(c)	DD(c)	

Ex.
To convert type DD b into double,
>>double(b)
To convert double a into type QD,
>>QD(a)

3.3 Input/Output

MuPAT has the following input and output functions. This function can only use scalar variables.
· ddinput(s)
Converts a numeric string s into numeric value of type DD.
· qdinput(s)
Converts a numeric string s into numeric value of type QD.
· ddprint(a)
Display the variable of type DD as a character string in 31 decimal places.
· qdprint(a)
Display the variable of type QD as a character string in 63 decimal places.

3.4 Other DD and QD function

A random DD and QD numbers are generated by the MATLAB function. The upper part and the lower part in the DD are assigned random numbers and are normalized to one value. In the same way, from the highest to the lowest part of QD numbers are assigned random numbers and normalized to one value.
・ddrand(m, n)
Returns a pseudorandom DD matrix that size is m by n.
・qdrand(m, n)
Returns a pseudorandom QD matrix that size is m by n.
・ddeye(m, n)
Returns an identity matrix of type DD that size is m by n.
・qdeye(m, n)
Returns an identity matrix of type QD that size is m by n.

3.5 Arithmetic operators

3.5.1 Four arithmetic operators

You can use the four arithmetic operators (+, -, *, /) in type DD and QD. Division operator is only supported in scalars. When performing mixed-precision operations, the computation results are stored in the type of higher-precision. It is like an implicit type conversion between 'int' and 'double' on usual programming language.
Ex. 1
Compute $1 + 10^{-20}$ using double and DD.
>> d = 1 + 1.0E-20
d =
1
>> e = DD(1, 1.0E-20);
e =
hi: 1
lo: 1.0000e-20
>> e - d
ans =
hi: 1.0000e-20
lo: 0

Ex. 2
Compute $1 \div 3 = 0.333$ ... using double, DD, and QD.
>> format long
>> disp(1/3);
0.333333333333333
>> ddprint(DD(1)/3);
' 3.333333333333333333333333333333E-1'
>> qdprint(QD(1)/3);
'3.333333333333333333333333333333333333333333333333333333333333333E-1'
※ If the command window is small, some of the displayed output values may be omitted.

3.5.2 Dot operators

You can use dot arithmetic (. *, ./) in type DD and type QD as well as MATLAB.
Ex. 3 Compute element-wise multiplication using a 2x2 DD matrix.
>> A = DD([1.1, 1.2, 1.3]);
>> B = DD([1.1, 1.2, 1.3]);
>> C = A .* B

hi: [1.2100 1.4400 1.6900]
lo: [8.8818e-18 -5.3291e-17 -5.3291e-17]
Ex. 4
Compute element-wise division using a 2x2 DD matrix.
>> A = DD([1.1, 1.2, 1.3]);
>> B = DD([1.1, 1.2, 1.3]);
>> C = A ./ B
ans =
hi: [1 1 1]
lo: [0 0 0]

3.6 Relational operators

We can use the same operator $(=, \sim=, <, >, \leq, \geq)$ for DD and QD as double. These operators return logical value 0 or 1.
Ex. 5 The relationship between 1/7 stored in double and 1/7 stored in DD and QD.
>> a = 1/7;
>> b = 1/DD(7);
>> c = 1/QD(7);
>> a == b
ans =
logical
0
>> a < b
ans =
logical
1
>> a ~= c
ans =
logical
1
>> a == b.hi
ans =
logical
1

3.7 Logical operators

We can use the same operator $(\&, |, \sim)$ for DD and QD as double. These operators return logical value 0 or 1.
Ex. 6 Logical operators example
>> A = DD([1,0,3; 4,5,0]);
>> B = DD([0,0,3;4,0,6]);
>> A & B
ans =
2x3 logical array
0 0 1
1 0 0
>> A | B
ans =
2x3 logical array
1 0 1
1 1 1

>> ∼A
ans =
2x3 logical array
0 1 0
0 0 1

3.8 Element insertion and extraction

· A(i, j) = b
Insert the variable b to the (i, j) element of array A. The assignment is not reflected if the precision on the left side is higher than the right side.
Ex.
>> A = DD([1,2,3; 4,5,6]);
>> A(2, 3)
ans =
hi: 6
lo: 0
>> A(2, 2) = 10;
>> A.hi
ans =
1 2 3
4 10 6
>> A.lo
ans =
0 0 0
0 0 0
>> A(2, 2) = QD(1); This is error because A is DD array although right side is QD number.

3.9 Other MATLAB functions

3.9.1 supported functions

· A'
Return the transposed matrix of A.
· [A, B]
Concatenates B horizontally to the end of A when A and B have compatible sizes.
· [A; B]
Concatenates B vertically to the end of A when A and B have compatible sizes.
· [m, n] = size(a)
Return the row length m and the column length n of a.
· abs(a)
Return the absolute value of a.
· sqrt(a)
Return the square root of a.
· dot(a, b)
Return the inner product of a and b.
· a^n
Returns the value of the scalar a raised to the nth power.
· norm(a,n)
Return the nth norm of a. The second argument can be 1, 2, inf or fro.

3.9.2 unsupported functions

Functions other than those described in Section 3.9.1, and functions such as numerical methods and matrix decomposition are not supported. If you call a non-supported MATLAB function, it looks as below.

Ex.
>> a=ddrand(5);
>> qr(a)
Undefined function 'qr' for input arguments of type 'DD'.

3.10 The difference from MuPAT on Scilab

Regarding the operation, when declaring type DD and type QD in the Scilab version, they were described as dd and qd in small letters, but in MATLAB version it is necessary to call functions in capital letter. In the Scilab version when converting types, for example, it was necessary to call the d2dd() function when you want to convert from double precision to DD, and the d2qd() function when you want to convert from double precision to type QD. However, in the MATLAB version, you can convert just do assign a variable to the type you want to convert. For more information, please refer to the section on type conversion in the User Guide. In addition, it was necessary to call the getHi () function in Scilab version when extracting double precision elements such as upper part of type DD and type QD. However, in MATLAB version, just do write a. hi or b.hh in code can extract double-precision elements.

For functions, the sparse matrix form supported by Scilab is not currently supported by MATLAB. Also, the MATLAB version does not support applications such as numerical methods and matrix decomposition. As a function not found in the Scilab version, the MATLAB version uses vector processing to speed up vector calculations and matrix calculations using parallel processing. Please refer to the item of parallel processing for details.

4 Parallel processing

To enable or disable FMA, AVX2, and OpenMP, enter the command as follows to the MATLAB command window.
>> startMuPAT(n1, n2, n3)
Please specify the arguments n1 (the number of OpenMP threads), n2 (AVX2 on / off) and n3 (FMA on / off). If you do not specify the input arguments, the parallel processing features not specified will be turned off.
The argument n1 can be a nonnegative input, and n2 and n3 can be 0 (off) or 1 (on).
You can also set on / off as following by entering to the MATLAB command window.
>> fmaon
>> fmaoff
>> avxon
>> avxoff
>> omp(n1)
When you check which parallelization is enabled or disabled, you can enter the following command to the MATLAB command window.
>> statusMuPAT

4.1 FMA

A floating point multiply-add operation ($x = a \times b + c$) is performed in one step via a single rounding FMA (fused multiply-add) instruction[4, 5]. As a result, the number of multiply-add operations are reduced up to two times. Multiply-subtraction is also supported. If turn on FMA, twoprod_fma algorithm[3] for DD and QD algorithm using FMA is called. Therefore, the internal algorithm in multiply operation differs depending on whether you use FMA or not.

4.2 AVX2

Since four double precision data are processed simultaneously with the AVX2 (Advanced Vector Extensions)[4, 5] instruction, we can expect the computation time would become four times faster.
Ex.

>> startMuPAT(1, 0, 0)
>> a = qdrand(500);
>> b = qdrand(500);
>> tic; a * b; toc;
Elapsed time is 8.374103 seconds.
>> avxon
>> tic; a * b; toc;
Elapsed time is 2.227658 seconds.

4.3 OpenMP

Since OpenMP[6] enables thread level parallelism within a shared memory, we can expect the computation time to decrease by a multiple of the number of cores. The number of threads indicates how many parallel to compute. The maximum number of threads varies depending on the CPU. The number of threads is not equal to the speedup rate. You can select from 1 to the maximum number of threads. It is usually better to set the number of threads = the number of cores.

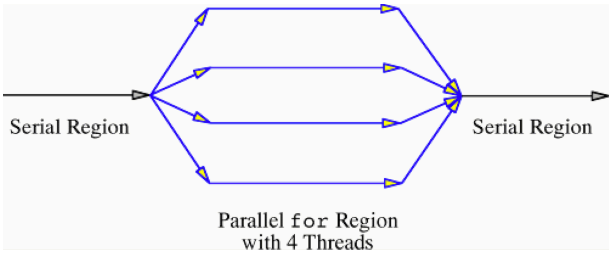

Figure 4.1: Image of multi threads using OpenMP

Ex.
>> startMuPAT(1, 0, 0)
>> a = ddrand(500);
>> b = ddrand(500);
>> tic; a * b; toc;
Elapsed time is 0.814772 seconds.
>> omp(2)
set the number of threads = 2
>> tic; a * b; toc;
Elapsed time is 0.449846 seconds.

4.4 Acceleration-supported operations

With MuPAT, using these parallelization, if the data is large enough and the operation count is large, for example, on a 4-core machine, it can be computed 16 to 17 times faster. In the table, 'D-DD' means that a operation x∘y is performed with double precision variable x and DD variable y, and 'DD-D' mean that a operation y∘x is performed with DD variable y and double precision variable x. The following table shows the 48 of operations are parallelized.
※ Sparse matrix form is not supported.

Table 4.1: Parallelization supported functions and how to call them

	D-DD	D-QD	DD-D	DD-DD	DD-QD	QD-D	QD-DD	QD-QD
Vector (matrix) addition	x, y: vector or matrix → x + y or x - y							
Scalar multiplication	a: scaler, x: vector or matrix → a * x or x * a							
Inner product	x, y: vector → dot(x, y) or x' * y							
Matrix-vector multiplication	A: matrix, b: vector → A * b							
Transposed matrix-vector multiplication	A: matrix, b: vector → tmv(A, b)							
Matrix-matrix multiplication	A, B: matrix → A * B							

References

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

[2] D. H. Bailey, High-Precision Floating-point arithmetic in scientific computation, Computing in Science and Engineering, Vol.7(3), pp.54-61

[3] Y. Hida, X. S. Li and D. H. Baily, Quad-double arithmetic: algorithms, implementation and application, Technical Report LBNL-46996, Lawrence Berkeley National Laboratory, Berkeley (2000).

[4] Intel: Intrinsics Guide, available from https://software.intel.com/sites/landingpage/IntrinsicsGuide/

[5] Intel: 64 and IA-32 Architectures Optimization Reference Manual, https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual

[6] OpenMP : http://www.openmp.org/

[7] Matlab Documentation: https://jp.mathworks.com/help/matlab/ref/mex.html

Appendix

A: Contents of toolbox

The folder structure of MuPAT is described below.

- MuPAT
  - README.md
  - useguideJap.pdf
  - useguideEng.pdf
  - @DD
    - DD.m
  - @QD
    - QD.m
  - @Calc
    - Calc.m
  - src
    - ddeye.m
    - ddrand.m
    - qdeye.m
    - qdrand.m
    - avxoff.m
    - avxon.m
    - fmaoff.m
    - fmaon.m
    - omp.m
    - statusMuPAT.m
    - mupat.c
    - mupat.h
    - .....(set of C program)
    - .....(set of MEX files)