User's Guide for MuPAT on MATLAB

version 1.0.0

Hotaka Yagi

April 30 2019

Contents

1 Introduction

The multiple precision arithmetic toolbox MuPAT implemented on Scilab[1] and MATLAB[2] can be used pseudo-quadruple-precision numbers (`double-double`)[3] and pseudo-octuple-precision numbers (`quad-double`)[4]. We can easily handle the above high-precision operations interactively using the Scilab and MATLAB command window, and with same codes. Furthermore, MuPAT on MATLAB is applied parallel processing to vector and matrix operations. As the parallel processing features, FMA, AVX2 and OpenMP are used. MuPAT on MATLAB is prepared for MacOS.

1.1 System requirements

MuPAT is available in the following environment.

MacOS 10.13 High Sierra or later
MATLAB R2017a or later
CPU: Intel Haswell or later, AMD processor is surveying now...

2 How to start MuPAT

2.1 Install

After download zip file of MuPAT from the web page [https://www.ed.tus.ac.jp/1419521/], unzip and move MuPAT to a directory that is easy to work, such as under the MATLAB directory.

2.2 Start-up

1. Launch MATLAB
2. Move current directory to MuPAT
3. Enter the following command to the MATLAB command window
```
>> startMuPAT(n1, n2, n3)
```
4. enter the following command to the MATLAB command window. When the following result is output, MATLAB is ready to use MuPAT.
```
>> a = DD(1)
a =
hi:  1
lo:  0
```

MuPAT can select the number of threads, valid or invalid of AVX2, valid or invalid of FMA as input arguments. (No arguments means the number of threads = 1, FMA, and AVX2 are set off)

Ex.
`startMuPAT`: Single-threading, invalid of AVX2, and invalid of FMA (serial computing)
`startMuPAT(2, 1, 1)`: run 2 threads, valid of AVX2, and vaild of FMA

2.3 Remarks

Although it is not a problem specific to MuPAT, MATLAB MEX function may kill MATLAB if the requested memory size is large. Furthermore, the computation order differs from the serial computing and parallel computing. It means they have different rounding errors. As a result, the computation results can not match.

3 Usage

In MuPAT, we can use quadruple and octuple precision arithmetics in the same way as double precision arithmetic, because MATLAB can define new data type and apply overloading for operators and some MATLAB functions. The special command in MuPAT is following.

- The declaration or conversion of variables of type `DD` and `QD` (section 3.1 and section 3.2)
- The ways of input and output (section 3.3)
- Some of `DD` and `QD` function (Section 3.4)

3.1 Variable definition

We define new data types named `DD` and `QD` to contain double-double and quad-double numbers. These types require two or four double precision numbers. The type `DD` has two double numbers of `hi` (upper part) and `lo` (lower part) for quadruple precision, and type `QD` has four double numbers of `hh` (the highest part), `hl`, `lh` and `ll` (the lowest part) for octuple precision. The upper part of `DD` variable `A` can be referenced with `A.hi`, and the lower 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.

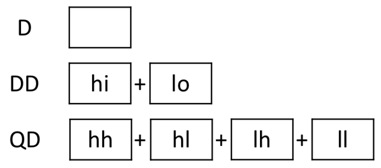

Figure 3.1: Images of type `DD` and type `QD`

To define type `DD` variable `A`, use DD function as follows. The `DD` function need up to two `double` precision value or variables as arguments. One type `DD` or type `QD` value or variable is also possible for argument (type conversion).

```
>> A = 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`.
- `DD(a, b)`, if two arguments are input, the input values are assigned to upper part and lower part respectively.

To define type `QD` variable `B`, use QD function as follows. The `QD` function need up to four `double` precision value or variables as arguments. One type `DD` or type `QD` value or variable is also possible for argument (type conversion).

```
>> B = 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 assigned 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 assigned to the lowest part.
- `QD(a, b, c, d)`, if 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:  3.1400
hl:  0
lh:  0
ll:  0
>> A = DD([1, 2, 3; 4, 5, 6], 1e-18 * [7, 8, 9; 10, 11, 12])
A =
hi:  [2x3 double]
lo:  [2x3 double]
>> A.hi
ans =
1 2 3
4 5 6
>> A.lo
ans =
1.0e-16 *
0.0700  0.0800  0.0900
0.1000  0.1100  0.1200
>> ddprint(A(2, 2)) % to output A(2, 2) of type DD matrix
'5.0000000000000000110000000000000E0'
```

3.2 Type conversion

We can convert the variables of data types (`double`, `DD`, or `QD`) into different data types (`double`, `DD`, or `QD`) with the following functions. When the type of arguments (scaler, vector or matrix) of `a` and `b` is different, please see section 3.8.

Table 3.1: The way to type conversion

from\to	`type double`	`type DD`	`type QD`
`type double`		`DD(a)`	`QD(a)`
`type DD`	`double(b)`		`QD(b)`
`type QD`	`double(c)`	`DD(c)`	

Ex.
To convert variable `b` of type `DD` into type `double`:
```
>> double(b)
```
To convert variable `a` type `double` into type `QD`:
```
>> QD(a)
```

3.3 Input/Output

MuPAT has the following input and output functions. These functions 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 functions

A random DD and QD numbers are generated by the MATLAB function `rand()`. The upper part and the lower part in the DD are assigned random numbers. Then, merge to one value. In the same way, from the highest to the lowest part of QD numbers are assigned random numbers and merged 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

We can use the same operators (`+`, `-`, `*`, `/`) in type `DD` and `QD`, and mixed precision arithmetic. Division operator is only supported in scalars. The computation result will be converted to larger type.

Ex. 1
```
>> d = 1 + 1.0E-20 % addition in double precision
d =
1
>> e = DD(1, 1.0E-20); % the number of type DD
e =
hi:  1
lo:  1.0000e-20
>> e - d % mixed precision arithmetic
ans =
hi:  1.0000e-20
lo:  0
```

Ex. 2
```
>> format long
>> disp(1/3); % double precision division
0.333333333333333
>> ddprint(DD(1)/3); % DD division
' 3.333333333333333333333333333333E-1'
>> qdprint(QD(1)/3); % QD division
'3.33333333333333333333333333333333333333333333333333333333333333E-1'
```

※ If the command window is small, some of the displayed output values may be omitted.

3.5.2 Dot operators

We can use the same dot operatiors (`.*, ./`) for type `DD` and `QD`.

Ex. 3
```
>> A = QD(ones(3,1));
>> B = 3 * A;
>> C = A ./ B;
>> ddprint(C)
'3.33333333333333333333333333333333333333333333333333333333333333E-1'
```

```
'3.33333333333333333333333333333333333333333333333333333333333333E-1'
'3.33333333333333333333333333333333333333333333333333333333333333E-1'
>> D = B .* C;
>> ddprint(D)
1.00000000000000000000000000000000000000000000000000000000000000E0'
1.00000000000000000000000000000000000000000000000000000000000000E0'
1.00000000000000000000000000000000000000000000000000000000000000E0'
```

3.6 Relational operators

We can use the same relational operators ($=, \sim=, <, >, \leq, \geq$) for type `DD` and `QD`.

Ex. 4

```
>> a = 1/7;
>> b = 1/DD(7);
>> a == b
ans =
logical
0
>> a == b.hi % the upper part is same
ans =
logical
1
>> a < b % the lower part of double precision number is 0
ans =
logical
1
```

3.7 Logical operators

We can use the same logical operators ($\&, |, \sim$) for type `DD` and `QD`.

Ex. 5

```
>> A = DD([1,0,3;4,5,0], 1e-18 * [7,8,9;0,0,12]);
>> B = DD([0,0,3;4,0,6], 1e-18 * [0,8,9;0,0,12]);
>> 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 precision of the left side must be the same

as or higher than that of the right side. If not, you will get an error.

Ex.
```
>> A = DD([1,2,3; 4,5,6], 1e-18 * [7,8,9;10,11,12]);
>> A(2, 3)
ans =
hi:  6
lo:  1.2000e-17
>> A(2, 2) = 10;
>> A.hi
ans =
1  2 3
4 10 6
>> A.lo
ans =
1.0e-16 *
0.0700 0.0800 0.0900
0.1000      0 0.1200
>> A(2, 2) = QD(1); % This comand is error because left side is type DD but right side is
type QD.
error:
You should cast DD to QD.
```

3.9 Other MATLAB functions

These MATLAB functions are enable to use in type DD and QD.

- `A’`
  Return the transposed matrix of `A`.
- `[A, B]`
  Concatenates `B` horizontally to the end of `A` (The type of result is cast to larger type) .
- `[A; B]`
  Concatenates `B` vertically to the end of `A` (The type of result is cast to larger type) .
- `[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 ofssssss `a`.
- `dot(x, y)`
  Return the inner product of vectors `x` and `y`.
- `tmv(A, b)`
  Return the fast transposed matrix vector multiplication of `A’` and `b`.
- `a^n`
  Returns the value of the scalar `a` raised to the `n`th power.
- `norm(a,n)`
  Return the nth norm of `a`. The second argument can be `1`, `2`, `inf` or `fro`.

When you call an unsupported function, the following error is displayed.

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

3.10 The difference from MuPAT on Scilab

When declaring type `DD` and type `QD` in the Scilab, they were described as dd and qd in small letters, but in MATLAB, it is necessary to call functions in capital letter. In Scilab, when converting types, it was necessary to call the `d2dd()` function when you want to convert from `double` to `DD`, and the `d2qd()` function when you want to convert from `double` to type `QD`. However, in MATLAB, you can convert just do assign a variable to the type you want to convert (see section 3.2). 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` elements. The sparse matrix form and the applications such as numerical methods and matrix decomposition are not supported in MATLAB.

4 Parallel processing

To be valid or invalid of FMA, AVX2, and OpenMP, enter the `startMuPAT` command to the MATLAB command window.

```
>> startMuPAT(n1, n2, n3)
```

Please specify the arguments `n1` (the number of threads), `n2` (valid is 1 or invalid is 0 of AVX2) and `n3` (valid is 1 or invalid is 0 of FMA). If you do not specify the input arguments, that will be invalid.
We can also set valid or invalid as following by entering to the MATLAB command window.

```
>> fmaon
>> fmaoff
>> avxon
>> avxoff
>> omp(n1) % n1 is the number of threads
```

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` × `b + c`) is performed in one step via a single rounding FMA (fused multiply-add) instruction[5, 6]. As a result, the number of multiply-add operations are reduced up to two times. If we set valid of FMA, `twoprod_fma`[4] routine is called for DD and QD algorithm. 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)[5, 6] instruction, we can expect the computation time would become four times faster.

4.3 OpenMP

Since OpenMP[7] 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 maximum number of threads varies depending on the CPU.

4.4 Acceleration-supported operations

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 $\rightarrow$ `x + y` or `x - y`							
Scalar multiplication	`a`: scaler, `x`: vector or matrix $\rightarrow$ `a * x` or `x * a`							
Inner product	`x`, `y`: vector $\rightarrow$ `dot(x, y)` or `x' * y`							
Matrix-vector multiplication	`A`: matrix, `b`: vector $\rightarrow$ `A * b`							
Transposed matrix-vector multiplication	`A`: matrix, `b`: vector $\rightarrow$ `tmv(A, b)`							
Matrix-matrix multiplication	`A`, `B`: matrix $\rightarrow$ `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] H. Yagi, E. Ishiwata, and H. Hasegawa, Acceleration of interactive multi-precision arithmetic toolbox MuPAT using parallel processing, Forum on Information Technology, Vol.1, pp.43-48 (2018).

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

[4] 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).

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

[6] 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

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

[8] 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)
```