MuPAT User Guide on MATLAB

Hotaka Yagi

2019 Apr 9

Contents

1 Introduction

The multiple precision arithmetic environment MuPAT [3], which is a multiple precision arithmetic environment that can be handled on pseudo-quadruple-precision numbers (double-double) [1] and pseudo-double-precision numbers (quad-double) [2], implemented on Matlab. It is prepared for each OS independently (now only Mac). MuPAT allows you to easily handle the above high-precision operations interactively using the MATLAB command window, and with almost same code when it was double precision. Furthermore, MuPAT is applied parallel processing to vector and matrix operations. As the parallel processing features, three kinds of FMA, AVX2 and OpenMP are used, and high speed processing is possible. For more information on parallel processing, please refer to the Parallel Processing section of the User Guide.

2 Install

After downloading MuPAT from the home page or downloading from GitHub, save MuPAT in a directory that is easy to work, such as under the MATLAB directory.

2.1 Syetem requirement

MuPAT is fully functional with the following environment.
After MATLAB R2017a
CPU: After Haswell (intel), AMD

2.2 How to start MuPAT

1. Start MATLAB
2. Move to MuPAT directory under MATLAB
3. Enter the following command in 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 and FMA and AVX2 are set off)
Example)
startMuPAT (2, 1, 1): Compute with 2 threads, AVX2 enabled, FMA enabled.
startMuPAT (4, 0, 1): Compute with 4 threads, AVX2 disenabled, FMA enabled.
startMuPAT(1,0,0): Compute with 1 threads, AVX2 disenabled, FMA disenabled (serial computing)

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

3 Usage

MuPAT can perform double-double (DD) operation that means pseudo-quadruple-precision operation and quad-double (QD) operation that means pseudo-quadruple-precision operation using the same operator as double precision. This is because we defined new types in MATLAB and overloaded the operators. DD has two double variables of hi (upper part) and lo (lower part), and QD has four double variables of hh, hl, lh and ll from the highest part to the lowest part respectively.

3.1 Definition of variables

All numbers in MATLAB are treated as doubles. In MuPAT, a new data type called DD type is defined to obtain the number of pseudo quadruple precision operations, and a new data type called QD type is defined

to obtain the number of pseudo 8 double precision operations. With these types, you can handle scalars, vectors, matrix operations, etc. with commands similar to double type. You can declare DD type and QD type variables with the following function.

>> DD(a, b)
DD type requires 0 to 2 arguments. As in DD(), the value 0 is assigned to the upper part and the lower part if there are zero arguments. As in DD (a), a is assigned to the upper part if only one argument is input, and the lower part is automatically assigned the value 0. When two arguments are input, the input values are assigned to a and b respectively. The types of inputs a and b, whether scalars, vectors or matrices, are irrelevant as long as the types of a and b match.

>> QD(a, b, c, d)
QD type requires 0 to 4 arguments. As in QD (), if there are zero arguments, the value 0 is assigned from the highest part to the lowest part. As in QD (a) if only one argument is input, a is assinged to the highest part, and the others are automatically assigned the value 0. As in QD (a, b), if two arguments are input, the value 0 is in the lower two places. As in QD (a, b, c), if there are 3 arguments, the value in the lowest part is 0. When 4 arguments are input, the input values in a, b, c and d are assigned to from the highest part to the lowest part respectively.
More over, if DD type variable A is substituted to QD type as QD (A), upper part and lower part of value DD type variable A allocated to upper two places of QD, and values after lower two places are automatically 0. The type of inputs in the QD type when scalar, vector, or matrix are irrelevant as long as the input types of a, b, c, d match.

The values of upper part hi and lower part lo of DD variable A can be referred to by A. hi and A. lo, respectively. QD variable B can also be referred to the highest part to lowest part by B. hh, B. hl, B. lh, B.ll, respectively.
Example)
```
>> a = DD(1, 0)
a =
hi: 1
lo: 0
>> b = QD(1, 0, 0, 0)
b =
hh: 1
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 Input/Output

MuPAT has the following output functions. This function can only use scalar variables.

・ddprint(a)

Display DD type variable as a character string in 31 decimal places.

・qdprint(a)

Display QD type variable as a character string in 63 decimal places.

3.3 Type conversion

MuPAT can convert data types with the following function.

・double(a)

Convert DD and QD type variable a to double type.

・DD(a)

Convert Double and QD type variable a to DD type.

・QD(a)

Convert Double or DD type variable a to QD type.

3.4 Write or extract elements in array

・A(i, j) = b

Write DD type variable b to (i, j) element of DD type variable A.

・A(i, j)

Extract the (i, j) element of variable A of DD type.

```
>> A = [1,2,3; 4,5,6];
>> AA = DD(A);
>> AA(2, 3)
ans =
hi: 6
lo: 0
>> AA(1, :)
ans =
hi: [1 2 3]
lo: [0 0 0]
>> AA(2, 2) = 10;
>> AA.hi
ans =
1  2 3
4 10 6
>> AA.lo
ans =
0 0 0
0 0 0
```

3.5 Arithmetic operator

3.5.1 Four arithmetic operations

You can use the four arithmetic operators (+,-, *, /) in DD and QD. Division operator is only supported in scalars.

Example1)

Compute $1 + 10^{-20}$ using Double type and DD type.

```
>> 1 + 1.0E-20
ans =
1
```

```
>> e = DD(1);
>> f = DD(1.0E-20);
>> e + f
ans =
hi: 1
lo: 1.0000e-20
```

Example2)
Compute 1 ÷ 3 = 0.333 ... using Double type and DD type.

```
>> format long
>> f = 1/3
f =
0.333333333333333
>> g = DD(1) / DD(3)
g =
hi: 0.333333333333333
lo: 1.850371707708594e-17
>> ddprint(g)
ans =
' 3.33333333333333333333333333333333E-1'
```

※ If the command window is small, part of the output may be omitted.

3.5.2 Other arithmetic operators

You can use dot arithmetic (. *, ./) in DD type and QD type as well as Double type.
Example3)
Compute element-wise multiplication using a 2x2 DD matrix.

```
>> A = [1,2,3; 4,5,6];
>> B = [2,2,2; 5,5,5];
>> AA = DD(A);
>> BB = DD(B);
>> CC = AA .* BB;
>> CC.hi
ans =
2   4   6
20 25 30
>> CC.lo
ans =
0 0 0
0 0 0
```

Example4)
Compute element-wise division using a 2x2 DD matrix.

```
>> A = [1,2,3; 4,5,6];
>> B = [3,3,3; 3,3,3];
>> AA = DD(A);
>> BB = DD(B);
>> CC = AA ./ BB;
>> CC.hi
ans =
0.333333333333333 0.666666666666667 1.000000000000000
1.333333333333333 1.666666666666667 2.000000000000000
>> CC.lo
ans =
1.0e-16 *
```

0.185037170770859 0.370074341541719 0
0.740148683083438 -0.740148683083438 0

3.5.3 Mixed type operation

Addition of double and DD types, double and QD types, and DD and QD types can also be computed using the " +" operator. This means that mixed arithmetic operations can be computed in the same way. When performing high-precision arithmetic with mixed precision, the calculation result is automatically converted to the higher precision type.

3.6 Relational operator

Like double type , DD and QD type can be used logical operators $(=, \sim=, <, >, \leq, \geq)$.
Example5) Relationship between 1/7 stored in double type and 1/7 stored in DD type and QD type

```
>> format long
>> a = 1/7;
>> b = 1/DD(7);
>> c = 1/QD(7);
>> disp(a)
hi: 0.1429
lo: 7.9302e-18
>> ddprint( b )
ans =
' 1.42857142857142857142857142857142857E-1'
>> disp(c)
hh: 0.1429
hl: 7.9302e-18
lh: 4.4021e-34
ll: 2.4437e-50
>> qdprint(c)
ans =
' 1.428571428571428571428571428571428571428571428571428571428571428571143E-1'
>> a == b
ans =
0
>> a < b
ans =
1
>> a ~= c
ans =
1
>> a == b.hi
ans =
1
```

3.7 Logical operator

Like double type, DD and QD types can use logical operators $(\&, |, \sim)$.
Example6) Logical operator in DD type

```
>>A = [1,0,3; 4,5,0];
>> B = [0,0,3;4,0,6];
>> AA=DD(A);
>> BB=DD(B);
>> AA & BB
```

ans =
0 0 1
1 0 0
>> AA | BB
ans =
1 0 1
1 1 1
>> ∼AA
ans =
0 1 0
0 0 1

3.8 Other operators

・A'
Return the transposed matrix of A.
・[A, B]
Connect A and B horizontally.
・[A; B]
Connect A and B vertically.
Example7) Horizontal connection
>> A = [1, 2, 3];
>> B = [4, 5, 6];
>> AA = DD(A);
>> BB = DD(B);
>> [AA, BB]
ans =
hi: [1 2 3 4 5 6]
lo: [0 0 0 0 0 0]
Example8) Vertical connection
>> A = [1, 2, 3];
>> B = [4, 5, 6];
>> AA = DD(A);
>> BB = DD(B);
>> CC = [AA ; BB];
>> CC.hi
ans =
1 2 3
4 5 6
>> CC.lo
ans =
0 0 0
0 0 0

3.9 Compatibility with MATLAB functions

You can use some MATLAB functions with DD and QD types.

3.9.1 Functions that can be used like MATLAB functions

・[m, n] = size(a)
Returns the row length m of a and the column length n.
・abs(a)
Returns the absolute value of a.
・sqrt(a)

Returns the square root of a.
・dot(a, b)
Returns the inner product of a and b.
他の関数
・ddpow(a, n).
Returns the nth power of a.
・ddrand(m, n)
Returns a DD pseudo-random matrix of type m × n.
・qdrand(m, n)
Returns a QD pseudo-random matrix of type m × n.
・ddeye(m, n)
Returns a DD-type identity matrix of m × n type.
・qdeye(m, n)
Returns a QD-type identity matrix of m × n type.

4 Parallel processing

There are three parallel processing methods used by MuPAT: FMA, AVX2, and OpenMP. To enable or disable these parallel processes, execute the command as follows.
>> startMuPAT(n1, n2, n3)
Specify as arguments n1 (the number of OpenMP threads), n2 (AVX2 on / off) and n3 (FMA on / off). If you do not specify it, the features not specified automatically will be turned off.
The argument n1 is assumed to be a nonnegative input, and n2 and n3 are assumed to be 0 (off) or 1 (on).
You can also set like following.
>> fmaon
>> fmaoff
>> avxon
>> avxoff
>> omp(n1)
Here, n1 in omp is assumed to be a nonnegative input.

4.1 FMA

A floating point multiply-add operation is performed in one step via a single rounding FMA (fused multiply-add) instruction[4, 5]. We applied twoprod_fma algorithm[2] that computes the exact product of two floating point numbers. Therefore, the internal algorithm 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. When array length is not a multiple of SIMD register length, the leftovers must be handled without AVX2.

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. You can not set the environment variable to change the number of threads in MuPAT.

4.4 Processing that has been accelerated

MuPAT uses the above three types of parallelization. As a result, the execution time will be shorter if all three types are used together. The following table shows the 48 types of operations targeted for acceleration.

Table 4.1: Functions and how to call the methods that currently support acceleration

	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							

※ Sparse matrix form is not supported.

For example, in a 4-core machine equipped with intel core i77820 HQ 2.9 GHz, the inner product of order 4,192,000 can be accelerated 6.6 times with DD-DD, and computed in about 0.005 seconds. With QD-QD, the inner product can be accelerated 16.4 times and computed in about 0.016 seconds.

The matrix-vector multiplication of order 2,048 can be accelerated 7.9 times with DD-DD, and can be computed in about 0.0034 seconds. With QD-QD, the matrix-vector multiplication can be accelerated 15.5 times and computed in about 0.016 seconds.

The matrix-matrix multiplication of order 500 can be accelerated 19.1 times with DD-DD, and can be computed in about 0.043 seconds. With QD-QD, the matrix-matrix multiplication can be accelerated 17.1 times and computed in about 0.46 seconds.

4.5 Remarks

Be aware of the required memory size for the problem you are dealing with on your machine. MATLAB's mex function may kill MATLAB if the requested memory size is somewhat larger than the memory size of your PC. It also depends on the number of threads used. Furthermore, when parallelization is used, the computation order changes from that of serial computing, and due to rounding errors, the computation results may not match completely as compared with seiral computing.

References

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

[2] 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).

[3] 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).

[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: Folder structure

The folder structure of MuPAT is described below.

- MuPAT
  - README.md
  - @DD
    - DD.m
  - @QD
    - QD.m
  - @Calc
    - Calc.m
  - functions
    - useful
      - ddeye.m
      - ddrand.m
      - qdeye.m
      - qdrand.m
      - avxoff.m
      - avxon.m
      - fmaoff.m
      - fmaon.m
      - omp.m
    - maincode
      - mupat.c
      - mupat.h
      - mupat.o
      - set of C functions to build MEX file (48 files)
      - set of MEX files (48 files)