Accelaration of MuPAT on MATLAB/Scilab Using Parallel Processing

Hotaka Yagi
(Tokyo University of Science, Japan)
Joint work with
Emiko Ishiwata
(Tokyo University of Science, Japan)
Hidehiko Hasegawa
(University of Tsukuba, Japan)

March 7 2018

Outline

What is MuPAT ?

- Multiple Precision Arithmetic Toolbox
- Multi precision arithmetic on Matlab[1]/Scilab[2]
  - avoid round-off errors, cancellations, information loss
- Interactive environment (Matlab/Scilab)
- Ease of Use
  - We can use MuPAT any environment easily

[1] Mupat on Matlab: Implementation of high-precision arithmetic onto MATLAB (in Japanese)
[2] Mupat on Scilab: https://atoms.scilab.org/toolboxes/DD_QD

Key features of MuPAT

- The same operator (+, -, *, /) can be used for double, DD, and QD arithmetic

➡ We can write a code easily

- Double, DD, and QD arithmetic can be used at the same time, and also mixed precision arithmetic is available
- It is independent of any hardware and operating systems : Only depends on Matlab/Scilab

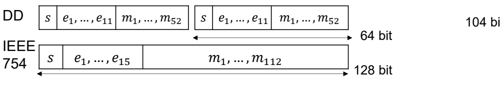
DD arithmetic

DD number α is represented in combination with only 2 double precision numbers α_0 and α_1 as below

$$\alpha = \alpha_0 + \alpha_1$$

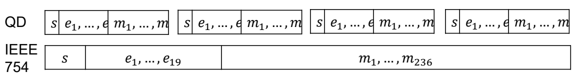
- α_0 *is rounded value of* α
- α_1 *is rounded value of* $\alpha - \alpha_0$

α_0 and α_1 satisfy
$$|\alpha_1| \le \frac{1}{2} ulp(\alpha_0)$$



QD number γ is also represented in the same way *as follows*

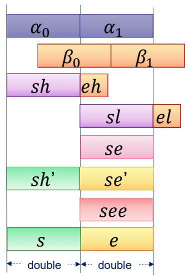
$$\gamma = \alpha_0 + \alpha_1 + \alpha_2 + \alpha_3$$



Addition algorithm of DD arithmetic

$\alpha, \beta \in DD, s, e, \alpha_0, \beta_0, \alpha_1, \beta_1 \in Double$

$[s, e] = Addition(\alpha, \beta)$
  $[sh, eh] = twoSum(\alpha_0, \beta_0);$
  $[sl, el] = twoSum(\alpha_1, \beta_1);$
  $se = eh + sl;$
  $[sh', se'] = fastTwoSum(sh, se);$
  $see = se' + el;$
  $[s, e] = fastTwoSum(sh', see);$
end



twoSum
```
[s, e] = twoSum(a, b)
        s = a + b;
        v = s - a;
        e = (a - (s - v)) + (b - v);
```

fastTwoSum
```
[s, e] = fastTwoSum(a, b)
        s = a + b;
        e = b - (s - a);
```

Multiplication algorithm of DDarithmetic

$\alpha, \beta \in DD, p, e, \alpha_0, \beta_0, \alpha_1, \beta_1 \in Double$

$[p, e] = Multiplication(\alpha, \beta)$
$\quad [p', e'] = twoProd(\alpha_0, \beta_0);$
$\quad e' = e + \alpha_0 * \beta_1;$
$\quad e'' = e' + \alpha_1 * \beta_0;$
$\quad [p, e] = fastTwoSum(p', e'');$
end

twoProd

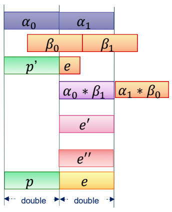
```
[p, e] = twoProd(a, b)
        p = a * b;
        [ah, al] = split(a);
        [bh, bl] = split(b);
        e= (ah * bh - p) + ah * bl
             + al * bh + al * bl;
```

split

```
[h, l] = split(a)
        t = (2^27+1) + a;
        h = t - (t - a);
        l = a - h;
```

fastTwoSum

```
[s, e] = fastTwoSum(a, b)
        s = a + b;
        e = b - (s - a);
```



7

Number of double precision operations

		Add-Subtract	Multiplication	Division	Sum
DD	Add-subtract	11	0	0	11
	Multiplication	15	9	0	24
	Division	17	8	2	27
QD	Add-subtract	91	0	0	91
	Multiplication	163	46	0	209
	Division	713	88	5	806

8

Problems

- Heavy:
  The number of double precision operations for DD and QD require from 10 to 1,000 times
- Difficulty:
  We cannot reduce the number of double precision operations, because the order of computations must be kept!

9

Performance gain of Parallelization

- FMA : **All** (Scalar, Vector, Matrix) , Multiplication only, not to high, Max 2 times
- SIMD: Vector and Matrix, 4 double precision numbers, Max 4 times
- OpenMP: Vector and Matrix, for loop, Max **# of cores**

10

Comparisons

- Matlab:
  Pure Matlab code
- Mex :
  C function outside Matlab
  Same operator as Matlab
  Matrix structure based on Matlab/Scilab
  Reducing calling overhead by large size of data
- Mex+OMP:
  C function outside Matlab, and accelarated by **OpenMP**
  Same operator as Matlab
  Matrix structure based on Matlab/Scilab
  Reducing calling overhead by large size of data

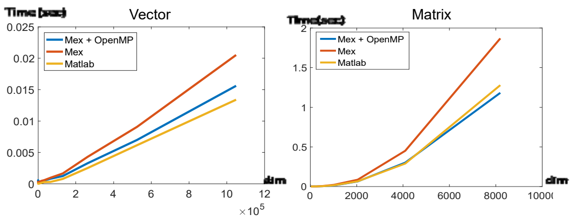
11

Environment of experiments

- MacBook Pro
  Processor : 2.5 GHz Intel Core i7 (2 cores)
  Memory : 16 GB
  OS: High Sierra version v10.13.1
- Matlab version Matlab_R2017a

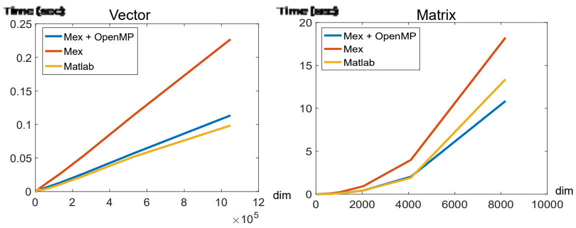
12

DD addition (Vector and Matrix)



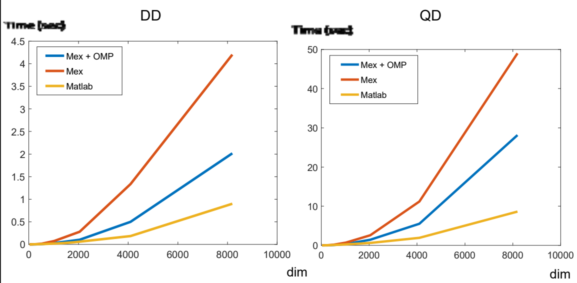
Time(sec)	Function/Dim	2^5	2^8	2^11	2^13	2^18	2^20
Vector	Mex+OMP	2.2e-04	2.2e-04	2.2e-04	2.7e-04	3.3e-3	1.6e-2
	Matlab	2.2e-05	2.3e-05	3.0e-05	7.6e-05	2.2e-3	1.2e-2
Matrix	Mex+OMP	2.6e-04	8.0e-04	6.4e-2	1.18	—	—
	Matlab	3.2e-05	3.2e-04	6.6e-2	1.28	—	—

QD addition (Vector and Matrix)

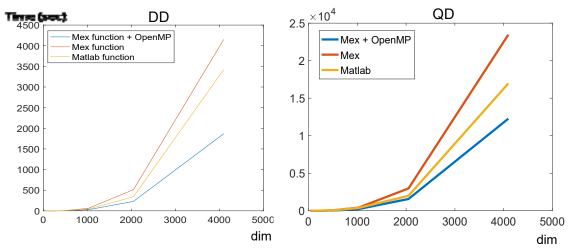


Time(sec)	Function/Dim	2^5	2^8	2^11	2^13	2^18	2^20
Vector	Mex+OMP	4.5e-04	4.7e-04	5.2e-04	9.2e-04	2.7e-2	1.1e-1
	Matlab	8.9e-05	9.3e-05	1.3e-04	4.3e-04	2.4e-2	9.9e-2
Matrix	Mex+OMP	4.9e-04	7.4e-3	4.2e-1	9.14	—	—
	Matlab	1.1e-04	3.4e-3	4.2e-1	13.38	—	—

Matrix-Vector multiplication (DD and QD)



Matrix-Matrix multiplication (DD and QD)



Computation time of DD and QD multiplication

Matrix-Vector

Time (sec)	Function /Dim	2^5	2^6	2^7	2^8	2^9	2^10	2^11	2^12
DD	Mex+OMP	3.2e-4	3.9e-4	1.0e-3	2.2e-3	9.9e-3	4.0e-2	4.8e-1	2.0
	Matlab	5.6e-4	1.1e-3	2.1e-3	3.3e-3	7.5e-3	6.0e-2	1.9e-1	0.9
QD	Mex+OMP	5.2e-4	8.8e-4	6.0e-3	1.5e-2	6.1e-2	2.3e-1	9.3e-1	23.5
	Matlab	1.5e-1	6.2e-1	2.16e-2	4.6e-2	9.9e-2	2.3e-1	6.7e-1	8.6

Matrix-Matrix

Time (sec)	Function /Dim	2^5	2^6	2^7	2^8	2^9	2^10	2^11	2^12
DD	Mex+OMP	7.4e-4	3.2e-3	4.5e-2	3.2e-1	2.8	21.4	158.4	1206.9
	Matlab	1.0e-2	4.2e-2	1.9e-1	9.5e-1	5.7	43.2	341.6	3424.1
QD	Mex+OMP	4.3e-3	2.0e-2	4.3e-1	3.4	26.6	204.0	1550.2	12264.5
	Matlab	1.5e-1	6.2e-1	2.8	12.4	70.5	350.8	1984.9	16954.4

Winners

- Addition (Vector and Matrix)
  Matlab
- Matrix-Vector Multiplication
  Matlab
- Matrix-Matrix Multiplication
  OpeMP

Computation time of CG methods

Matrix name : ex3
Dimension : 1821
Condition number : 1.683779e+10
(The University of Florida Sparse Matrix Collection)

Initial vector: $x_0 = (0,0,\ldots,0)^T$
Exact solution: $x^* = (1,1,\ldots,1)^T$
Stopping criterion: $|r_k|_2 < 10^{-12}|r_0|_2$
Max iteration: 5000

	Double	DD	QD
Matlab function (time)	3.20	303.79	5.12e+03
MEX +OpenMP (time)	—	603.28(0.50)	3.75e+03 (1.36)
residuals	4.57e-02	2.69e-04	1.84e-07
Iteration number	5000	5000	5000

19

Computation time for each iteration

Matrix ex3 (N=1821)

		Mat-Vec	Inner Product	Vec Addition	Scalor
# of used		1	3	3	3
DD	Mex+OMP	1.1e-1	3.3e-4	2.2e-4	7.1e-4
	Matlab	4.8e-2(0.43)	4.2e-3(12.7)	2.9e-5(0.13)	2.7e-4(0.38)
QD	Mex+OMP	7.3e-1	7.8e-4	5.1e-4	1.1e-3
	Matlab	5.9e-1(0.81)	1.66e-1(212)	1.2e-4(0.23)	5.8e-4(0.52)

20

Summary : tentative

- We applied OpenMP to accelarate MUPAT on Matlab.
  - Only Matrix multiplication is accelarated.
  - Vector operations using Matlab functions are fast enough.
  - OpenMP is not bad on a small laptop PC (small # of cores).
- Tune for more speed-ups.
  - Is it possible to apply OpenMP to Matlab function?
- To reduce # of operations Sparse data structure is needed.

21

References

[1] Hidehiko Hasegawa: Implementation of high-precision arithmetic onto MATLAB (Japanese)
[2] Mupat: https://atoms.scilab.org/toolboxes/DD_QD
[3] T.J.Dekker , Aa floating-point technique for extending the available precision, Numerische Mathematik, 18:224-242 (1971)
[4] D.E.Knuth, The Art of Computer Programming: Seminumerical Algorithms, volume 2 Addison Wesley, Reading, Massachusetts (1981)
[5] Matlab, https://jp.mathworks.com/products/matlab.html
[6] OpenMP, http://www.openmp.org/
[7] The University of Florida Sparse Matrix Collection http://www.cise.ufl.edu.research/sparse/matrices/

22