Parallelization of DD and QD high-precision arithmetic operations

Hidehiko Hasegawa[†], hasegawa@slis.tsukuba.ac.jp
Hotaka Yagi[*], 1419521@ed.tus.ac.jp
Emiko Ishiwata[*], ishiwata@rs.tus.ac.jp
[†]University of Tsukuba, Japan, [*]Tokyo University of Science, Japan

Introduction

- High precision arithmetics are effective for reducing rounding error.
- Especially, an interactive tool is needed. Our team developed MuPAT [1] (Multiple Precision Arithmetic Toolbox) for MATLAB using DD and QD algorithms.
- DD (double-double) and QD (quad-double) are easy to implement on conventional processors, but 10 to 600 double-precision floating-point operations are required for each DD and QD operations.
- Parallel processing by using AVX2 and OpenMP can reduce their computation time to a practical level.

DD and QD numbers

DD number α is represented in combination with 2 double-precision numbers α_0 and α_1:

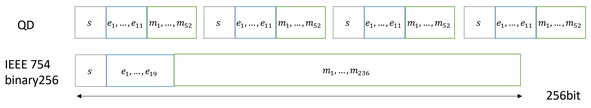
$$\alpha = \alpha_0 + \alpha_1$$ (almost 31 decimal digits)

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 β is represented in combination with 4 double-precision numbers $\beta_0, \beta_1, \beta_2$, and β_3 :

$$\beta = \beta_0 + \beta_1 + \beta_2 + \beta_3$$ (almost 63 decimal digits)



Basic algorithm for DD arithmetic [2]

	$c = DD\ addition\ (a,b)$	$c = DD\ multiplication\ (a,b)$
1.	$s = a_{hi} \oplus b_{hi}$	$p = a_{hi} \otimes b_{hi}$
2.	$v = s \ominus a_{hi}$	$e = fl(a_{hi} \times b_{hi} - p)$
3.	$eh = a_{hi} \ominus (s \ominus v)$	$e = fl(a_{hi} \times b_{lo} + e)$
4.	$eh = eh \oplus (b_{hi} \ominus v)$	$e = fl(a_{lo} \times b_{hi} + e)$
5.	$eh = eh \oplus (a_{lo} \oplus b_{lo})$	$c_{hi} = p \oplus e$
6.	$c_{hi} = s \oplus eh$	$c_{lo} = e \ominus (c_{hi} \ominus p)$
7.	$c_{lo} = eh \ominus (c_{hi} \ominus s)$	

$\oplus, \ominus, \otimes$: double-precision floating-point operations
$fl(a \times b \pm c)$: double-precision FMA

Parallelizing inner loop using AVX2

AVX2 instructions can process 4 double-precision data in one unit of time.

- AVX2 can be used with FMA by using Intel Intrinsic Instructions.
- Using AVX2 vector-load and vector-store instructions is key.
- Accessing memory continuously is essential for use AVX2.
- Summing the data in the SIMD register is a problem.
- When the dimension is not a multiple of 4, the leftovers must be handled without AVX2.

Parallelizing outer loop using OpenMP

OpenMP allows thread-level parallelism on shared memory for a multicore environment.

Each thread is a separate process with its own instructions and data.

- A loop is parallelized by putting a pragma directive above the loop.
- Parallelizing outer loop to give more workload for thread.
- Summing up thread local variable into master thread is a problem.

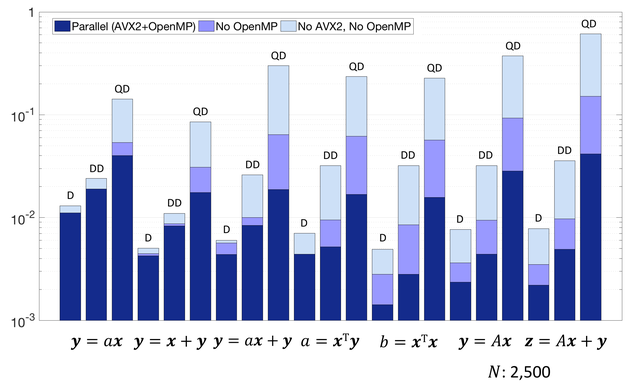
The numnber of double-precision operations

		+, -	*	FMA	/	total
DD	Addition, Subtraction	11	0	0	0	11
	Multiplication	3	1	3	0	7
	Division	6	1	2	2	11
QD	Addition, Subtraction	91	0	0	0	91
	Multiplication	106	6	10	0	123
	Division	549	48	3	5	605

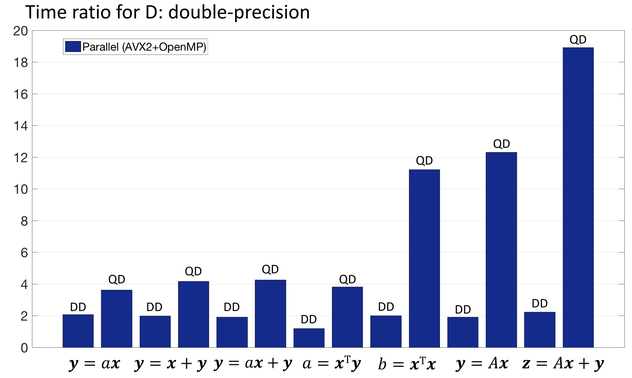
Comparison of parallelization

CPU: Intel Core i7 7820HQ, 2.9 GHz processor, 4 cores
Memory: LPDDR-2133
Peak performance of No AVX2, No OpenMP: 5.8 Gflops/sec
Peak performance of No OpenMP: 23.2 Gflops/sec
Peak performance of parallel : 92.8 Gflops/sec

Computation time [sec] N: 4,092,000



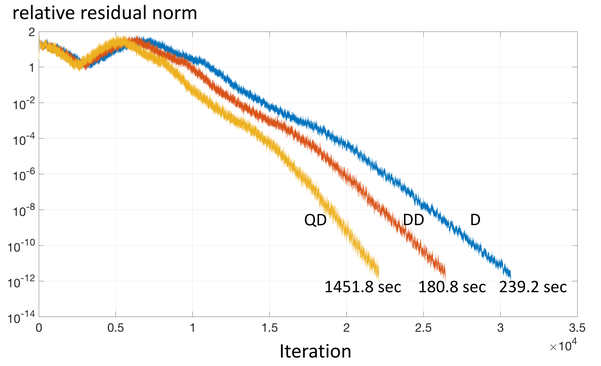
Time ratio for D: double-precision



		Memory Requirement	Memory Refreence	Flops	Oper-ational Intensity	Serial		AVX2		AVX2 + OpenMP	
						msec	Gflops /sec	msec	Gflops /sec	msec	Gflops /sec
$y = ax$	D	$16N$	$16N$	N	0.063	11.8	0.035	10.0	0.41	11.1	0.37
	DD	$32N$	$32N$	$7N$	0.22	24.0	1.19	19.0	1.51	23.0	1.25
	QD	$64N$	$64N$	$122N$	1.91	141.8	3.52	53.7	9.31	40.2	12.44
$y = x + y$	D	$16N$	$24N$	N	0.042	5.0	0.82	4.5	0.90	4.2	0.97
	DD	$32N$	$48N$	$11N$	0.23	11.0	4.10	8.7	5.18	8.3	5.43
	QD	$64N$	$96N$	$91N$	0.95	85.5	4.36	30.8	12.11	17.5	21.26
$y = ax + y$	D	$16N$	$24N$	$2N$	0.083	6.0	1.36	5.6	1.46	4.4	1.86
	DD	$32N$	$48N$	$18N$	0.38	26.0	2.84	10.0	7.37	8.4	8.78
	QD	$64N$	$96N$	$213N$	2.22	299.8	2.91	64.0	13.63	18.7	46.64
$a = x^{\mathrm{T}}y$	D	$16N$	$16N$	$2N$	0.13	6.8	1.20	4.2	1.95	4.4	1.86
	DD	$32N$	$32N$	$18N$	0.56	32.0	2.30	9.5	7.76	5.2	14.18
	QD	$64N$	$64N$	$213N$	3.33	235.0	3.71	61.8	14.12	16.8	51.88
$y = Ax$	D	$8N^2 + 16N$	$8N^2 + 16N$	$2N^2$	0.25	7.7	1.62	3.6	3.47	2.3	5.43
	DD	$16N^2 + 32N$	$16N^2 + 32N$	$18N^2$	1.13	32.0	3.52	9.4	11.97	4.4	25.57
	QD	$32N^2 + 64N$	$32N^2 + 64N$	$213N^2$	6.66	375.1	3.55	93.0	14.32	28.3	47.08

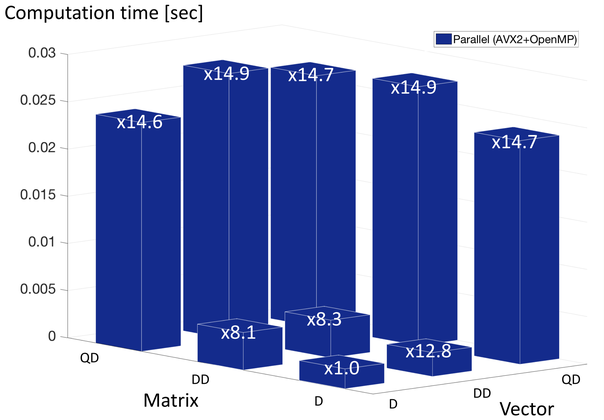
Convergence history of CG

matrix: bcsstk15 (Double)
N: 3948
condition number: 6.5e+09



Mixed-precision matrix vector product

Computation time [sec]



Summary

- DD and QD high precision arithmetic operations are accelerated on a laptop computer.
- A computation time of DD arithmetic operations is almost twice of double-precision operations by parallelization.
- High precision arithmetic can be use interactively and easily on MATLAB by using MuPAT.

URL of MuPAT

References

[1] S. Kikkawa, T. Saito, E. Ishiwata, and H. Hasegawa. 2013. Development and acceleration of multiple precision arithmetic toolbox MuPAT for Scilab. JSIAM Letters 5 (2013), 9-12.
[2] Y. Hida, X. S. Li, and D. H. Bailey. 2000. Quad-Double Arithmetic: Algorithms, Implementation, and Application. Technical Report LBNL-46996