More Accurate Computation for Double-Double Arithmetic without Additional Execution Time by Parallel Processing

Hotaka Yagi*
Tokyo University of Science
Shinjuku, Tokyo, Japan
1419521@ed.tus.ac.jp

Emiko Ishiwata
Tokyo University of Science
Shinjuku, Tokyo, Japan
ishiwata@rs.tus.ac.jp

Hidehiko Hasegawa
University of Tsukuba
Tsukuba, Ibaraki, Japan
hasegawa@slis.tsukuba.ac.jp

1 INTRODUCTION

While rounding errors are unavoidable in floating-point arithmetic, the use of high-precision arithmetic is effective. Our team developed *MuPAT*, an open-source interactive *Multiple Precision Arithmetic Toolbox* [3] for MATLAB and Scilab. MuPAT uses the DD (Double-Double) algorithm, which is based on a combination of double-precision arithmetic operations, and enables quasi quadruple-precision arithmetic. We accelerate DD operations by using AVX2 and OpenMP, and achieve higher performance for heavier DD operations. This poster shows more accurate computation can be performed without additional execution time based on the parallelization.

2 ACCELERATION OF DD ARITHMETIC

There are two implementations of addition ($a \oplus b$) in DD, called Cray-style and IEEE-style [1]. The Cray-style requires 11 double-precision floating-point operations, and only satisfies the weaker error bound ($a \oplus b = (1+\delta_1)a+(1+\delta_2)b$ with $|\delta_1|, |\delta_2| \leq \epsilon_{dd} = 2^{-105}$). The IEEE-style requires 20 double-precision floating-point operations, and satisfies a much strong error bound ($a \oplus b = (1+\delta)(a+b)$ with $|\delta| \leq 2\epsilon_{dd}$). The IEEE-style is accurate, but not widely used, because of its computation cost. We tried to reduce its execution times by using parallel processing. Since the order of computation in DD arithmetic cannot be changed, we considered processing multiple data simultaneously using data-level parallelism.

The DD multiplication algorithm utilizes FMA (Fused Multiply-Add) [2], which can perform a double-precision floating-point multiply-add operation in one step with a single rounding, and the rounding error is reduced. AVX2 (Advanced Vector Extensions 2) instructions [2] can process four double-precision data in one unit of time. The same arithmetic operations are applied to these four data. Thereby, the performance may be increased four-fold. OpenMP allows thread-level parallelism on shared memory for a multicore environment, so the performance may increase by the number of cores.

Performance [flops/sec] is defined as the number of double-precision floating-point operations [flops] / the execution time [sec]. The upper bound of performance is defined as min(computational performance, memory performance × operational intensity). The computational performance [flops/sec] is defined as the product of clock frequency for the CPU [Hz] and the number of flops which can be computed in one unit of time [flops/cycles]. Memory performance [bytes/sec] is defined as 8 bytes/cycles times the product of clock frequency for memory [Hz] and the number of channels. Operational intensity [flops/bytes] is defined as the number of double-precision floating-point operations [flops] / the number of memory references [bytes].

We used an Intel Core i7 7820HQ, 2.9 GHz processor, with LPDDR3-2133 memory. The peak computational performance is 92.8 Gflops/sec using AVX2 with four cores. The peak memory performance is 34.1 Gbytes/sec because there are two channels. We utilized computation offloading to call an outer C function with the MATLAB executable file.

Table 1: Operational intensity (O. I.) [flops/byte], execution time (serial/accelerated) [msec], and performance (measured/upper bound) [Gflops/sec].

	Cray-style			IEEE-style		
	O. I.	Time	Perf.	O. I.	Time	Perf.
$\boldsymbol{y} = \alpha\boldsymbol{x} + \boldsymbol{y}$	0.38	26/8.4	8.8/12.8	0.56	30/8.7	12.7/19.2
$\alpha = \boldsymbol{x}^T\boldsymbol{y}$	0.56	32/5.2	14.2/19.1	0.84	50/5.4	20.5/28.8
$\boldsymbol{y} = A\boldsymbol{x}$	1.13	32/4.4	25.6/38.5	1.69	53/4.3	39.2/57.2

In Table 1, "Time" refers to the execution time of serial/accelerated and "Perf." refers to the performance of measured / upper bound; N = 4,096,000 for $\boldsymbol{y} = \alpha\boldsymbol{x} + \boldsymbol{y}$ and $\alpha = \boldsymbol{x}^T\boldsymbol{y}$, and N = 2,500 for $\boldsymbol{y} = A\boldsymbol{x}$. If O. I. is below 2.72, when computational performance is equal to memory performance × operational intensity, then the upper bound of performance is limited by the memory performance. Therefore, most of the execution time is taken up by memory references, not computation. The execution times of $\boldsymbol{y} = \alpha\boldsymbol{x} + \boldsymbol{y}$, $\alpha = \boldsymbol{x}^T\boldsymbol{y}$, and $\boldsymbol{y} = A\boldsymbol{x}$ are almost the same in both the Cray-style and the IEEE-style. This means that the IEEE-style, which has higher computational cost and higher O. I., is much accelerated by parallel processing.

When operations are limited by memory performance and two operations have the same number of memory references, some additional computation can be processed in the same execution time. In this case, high performance means that more computation could be processed for the same amount of data. Thus, parallel processing processes a much larger workload and provides us more accurate results for the same time.

This research was supported by JSPS KAKENHI Grant Number JP17K00164.

REFERENCES

[1] Yozo Hida, Xiaoye S. Li, and David H. Bailey. 2000. Quad-Double Arithmetic: Algorithms, Implementation, and Application. Technical Report LBNL-46996.
[2] Intel. 2019. *Intel Intrinsics Guides*. Retrieved November 11 , 2019 from https://software.intel.com/sites/landingpage/IntrinsicsGuide/
[3] Satoko Kikkawa, Tsubasa Saito, Emiko Ishiwata, and Hidehiko Hasegawa. 2013. Development and acceleration of multiple precision arithmetic toolbox MuPAT for Scilab. *JSIAM Letters* 5 (2013), 9–12. https://doi.org/10.14495/jsiaml.5.9