Accurate Matrix Multiplication on Binary128 using Ozaki Scheme

Daichi Mukunoki[1], Katsuhisa Ozaki[2], Takeshi Ogita[3], Toshiyuki Imamura[1]
[1]RIKEN Center for Computational Science, [2]Shibaura Institute of Technology, [3]Tokyo Woman's Christian University

Acknowledgement: This research was supported by the Japan Society for the Promotion of Science (JSPS) KAKENHI Grant Number 19K20286.
This research used computational resources of the Cygnus supercomputer provided by Multidisciplinary Cooperative Research Program in Center for Computational Sciences, University of Tsukuba.

Introduction

Binary128 (FP128)

- The quadruple-precision floating-point format with 113-bit significand and 15-bit exponent defined in IEEE 754-2008
- Currently, not available on most processors such as x86 and ARM in hardware
- Software implementation (emulation) is available on GNU and Intel compilers, but they take a large performance overhead compared with binary64 in hardware
- The need for quadruple-precision is limited at present but certainly exists in some applications such as SDP [1]. We expect that providing fast implementation invokes new applications

Overview of this work

- A fast accurate matrix multiplication method on binary128 matrices on CPUs using Ozaki scheme [2]
- Ozaki scheme enables one to perform the most of the computations using DGEMM; a good performance can be obtained easily on many-core processors

Related work

- Double-double (DD) arithmetic [3]: a quadruple-precision arithmetic method that can be built upon binary64 arithmetic, with 106-bit significand and 11-bit exponent; this is faster than binary128 emulation at arithmetic operation level
- MPLAPACK [4]: a multi-precision BLAS/LAPACK; the high-precision arithmetic can be performed using binary128 (GCC/ICC), QD [5] (with double-double), MPFR [6], etc.

Tab. Floating-point formats

Format	Exponent	Mantissa	Decimal digits (approx.)
binary32 (FP32)	8	24	7.23
binary64 (FP64)	11	53	15.95
binary128 (FP128)	15	113	34.02
double-double	11	106	31.91

Methodology

Ozaki scheme

- We utilize the Ozaki scheme [2], which is an accurate matrix-multiplication algorithm based on the error-free transformation for dot-product / matrix-multiplication
- Ozaki scheme transforms a matrix multiplication to the element-wise summation of several error-free matrix multiplications, which can be performed using standard floating-point operations without rounding-errors
- Our previous work "DGEMM using Tensor Cores" [7] extended the original Ozaki scheme to utilize low-precision GEMM inside

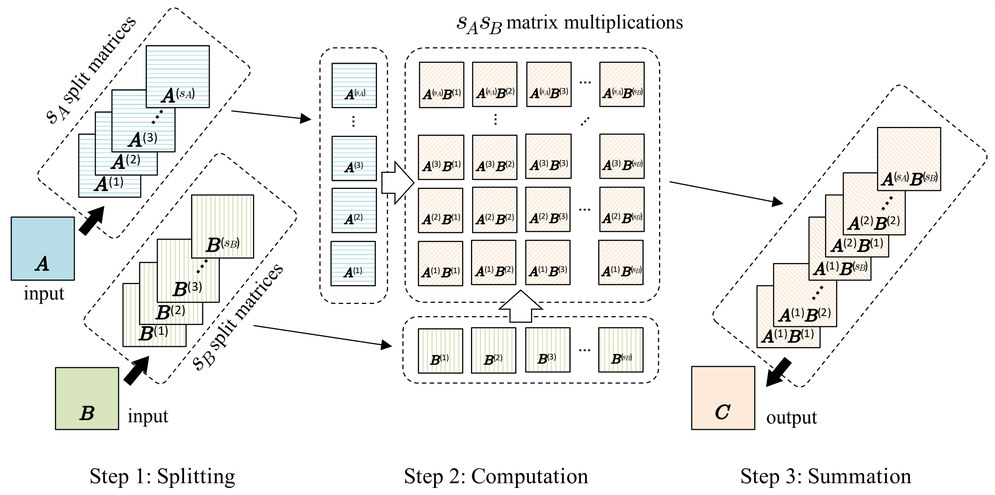


Fig. Schematic of matrix multiplication using Ozaki scheme

For $\boldsymbol{x}^T\boldsymbol{y}$ ($\boldsymbol{x},\boldsymbol{y} \in \mathbb{F}_{\mathtt{b128}}{}^n$, $\mathbb{F}_{\mathtt{b128}}{}^n$ is the set of binary128)

1. Splitting (element-wise)

$$\boldsymbol{x} = \sum_{p=1}^{s_x} 2^{c^{(p)}} \underline{\boldsymbol{x}}^{(p)} \qquad \boldsymbol{y} = \sum_{q=1}^{s_y} 2^{c^{(q)}} \underline{\boldsymbol{y}}^{(q)}$$

2. Computation & Summation

$$\boldsymbol{x}^T\boldsymbol{y} = \sum_{p=1}^{s_x}\sum_{q=1}^{s_y} 2^{c_x{}^{(p)}+c_y{}^{(q)}} \underline{\boldsymbol{x}}^{(p)T}\underline{\boldsymbol{y}}^{(q)}$$

Exponent (2byte-integer); Significand (binary64)

$$\underline{\boldsymbol{x}}^{(p)T}\underline{\boldsymbol{y}}^{(q)} = \mathtt{fl}_{\mathtt{b64}}\left(\underline{\boldsymbol{x}}^{(p)T}\underline{\boldsymbol{y}}^{(q)}\right)$$

Error-free: no rounding-error occurs on binary64

$$\rho := \left\lceil \frac{-\log_2 \mathtt{u}_{\mathtt{b128}} + \log_2 \mathtt{u}_{\mathtt{b64}} + \log_2 n}{2} \right\rceil$$

Recursively performed until $c^{(p)}=0$

$$c^{(p)} := \left\lceil \mathtt{fl}_{\mathtt{b128}}\left(\log_2\left(\max_{1\le i\le n}\left|\boldsymbol{x}^{(p)}{}_i\right|\right)\right)\right\rceil^{\dagger}$$

$$\sigma := 2^{\rho+c^{(p)}}$$

$$\underline{\boldsymbol{x}}'{}_i := \mathtt{fl}_{\mathtt{b128}}\left(\left(\boldsymbol{x}^{(p)}{}_i + \sigma\right) - \sigma\right)$$

$$\boldsymbol{x}^{(p+1)}{}_i := \mathtt{fl}_{\mathtt{b128}}\left(\boldsymbol{x}^{(p)}{}_i - \underline{\boldsymbol{x}}'{}_i\right)$$

$$\underline{\boldsymbol{x}}^{(p)}{}_i := \mathtt{fl}_{\mathtt{b128}}\left(2^{-c^{(p)}}\underline{\boldsymbol{x}}'{}_i\right)$$

$^{\dagger}\, c^{(p)} := 0$ when $\max_{1\le i\le n}|\boldsymbol{x}^{(p)}{}_i| = 0$

Fig. Ozaki scheme for inner-product

Optimizations for binary128

- The implementation is based on our previous work [7], but the following optimizations have been applied for improving the performance on binary128 computations: they try to avoid slow binary128 operations as much as possible (note: they are applicable only when input matrices fit into the binary64's exponent range. The other cases need to be computed without them)
- Optimization-1: fast splitting using binary64. The input binary128 matrices are first split into three binary64 matrices before applying to the Ozaki scheme, and those binary64 matrices are split using the Ozaki scheme for binary64.
- Optimization-2: fast summation using binary64: the results of matrix multiplications of split matrices (obtained in binary64) are summed by TwoSum [8] using three bins and then converted to binary128

Evaluation

Environment

- CPU: Intel Xeon Gold 6126 (Skylake, 12 cores, 2.6GHz) x 2 sockets (24-cores in total)
- RAM: DDR4-2666 192GB (255.9GB/s)
- Execution: 24 threads, with "numactl --localalloc"
- Compilation: g++ (GCC) 8.3.1 20190311 with MKL 19.0.5, "-O3 -fp-model source -fprotect-parens -qopenmp -xCORE-AVX2 -mtune=skylake-avx512"

Experimental Setup

- A matrix multiplication: C=AB (α=1.0 and β=0.0 in GEMM that computes C=αAB+βC)
- Input matrices are initialized with pseudo uniform random numbers of +/- [1,10^R) as *the performance depends on the absolute value range of the input matrices* (it decides the number of split matrices and GEMMs)

Comparison

- **Oz-b128**: our implementation based on Ozaki scheme
- **MM-b128**: MPLAPACK's binary128 GEMM (ICC's binary128 emulation)
- **MM-dd**: MPLAPACK's double-double GEMM (QD's double-double)
- Note: MPLAPACK's GEMM routines perform a classic matrix multiplication using high-precision arithmetic with OpenMP

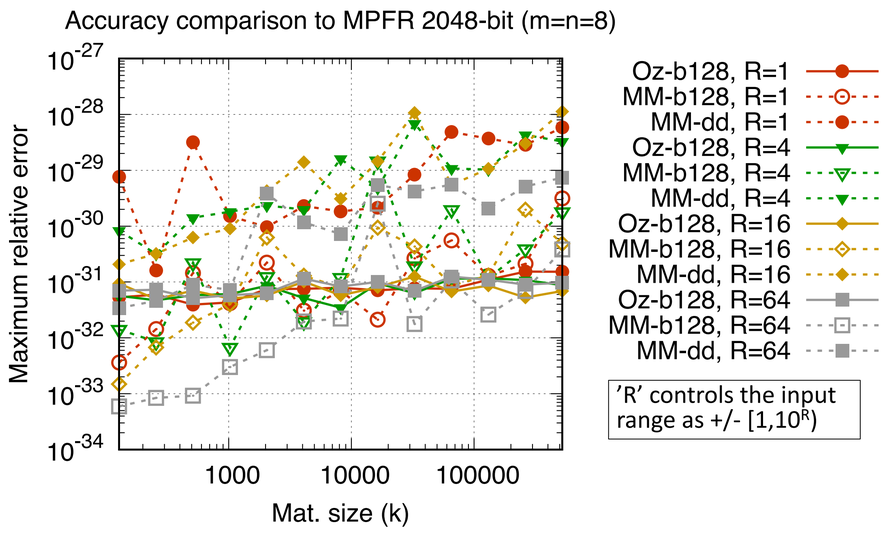


Fig. Accuracy evaluation

- Oz-b128 does not increase the error even when matrix size increases while others increase by following the error-bound of inner-product

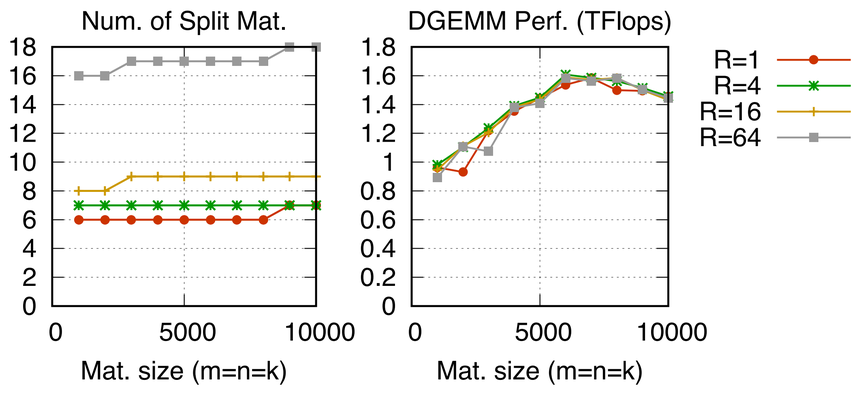


Fig. Performance analyses

- Performance of Oz-b128 depends on the number of split matrices (and GEMMs accordingly) and the DGEMM performance

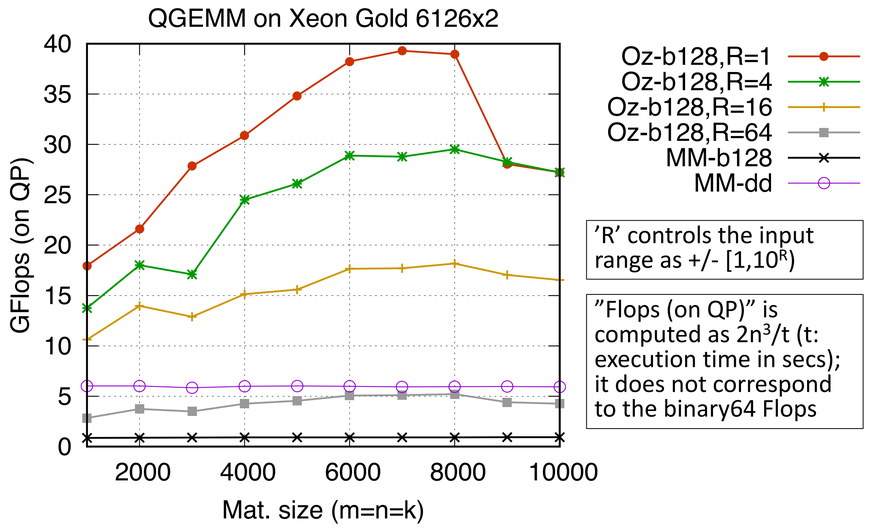


Fig. Performance evaluation

- Oz-b128 outperforms MPLAPACK's binary128 GEMM and double-double GEMM in most cases

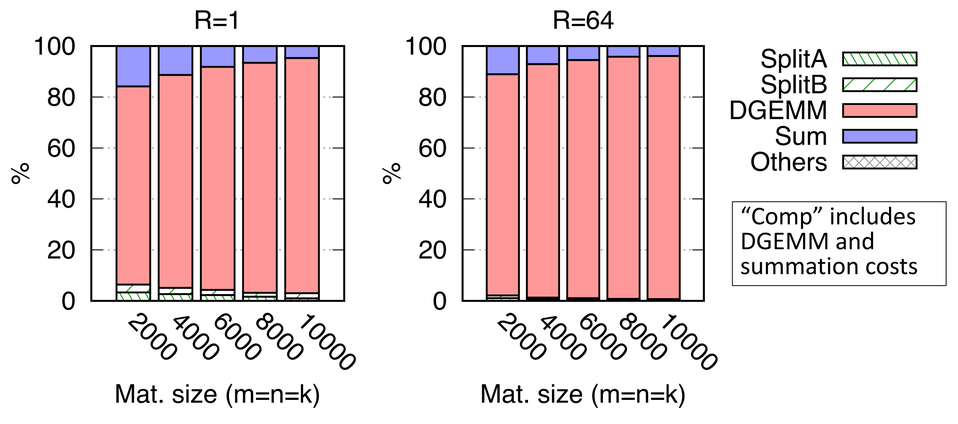
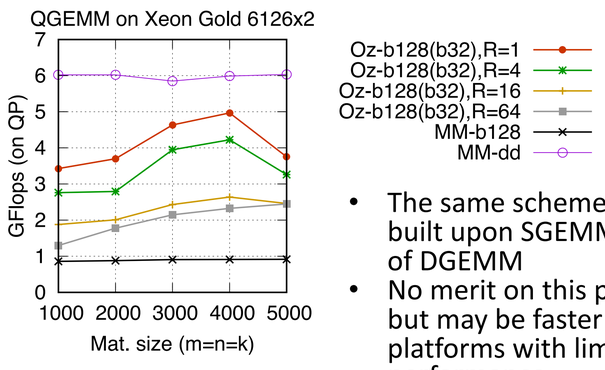


Fig. Execution time breakdown

- The matrix splitting cost is negligible, and the DGEMM cost is dominant in the total execution time

Extension



- The same scheme can be built upon SGEMM instead of DGEMM
- No merit on this platform, but may be faster on platforms with limited FP64 performance

Conclusion

- An accurate matrix multiplication on binary128 matrices using Ozaki scheme, which enables one to compute binary128 matrices using DGEMM: easy to achieve a good performance on many-cores with SIMD
- Performance is input-dependent: performance decreases as the absolute range of input matrices increase
- Faster than MPLAPACK's binary128-GEMM and double-double-GEMM (but double-double has a potential to achieve better performance with SIMD optimizations [9])
- Code is available as Accurate and Reproducible BLAS "OzBLAS"
  https://www.r-ccs.riken.jp/labs/lpnctrt/projects/ozblas/

References

[1] M. Nakata, "A numerical evaluation of highly accurate multiple-precision arithmetic version of semidefinite programming solver: SDPA-GMP, -QD and -DD", Proc. International Symposium on Computer-Aided Control System Design, 2010
[2] K. Ozaki, T. Ogita, S. Oishi, S. M. Rump, "Error-free transformations of matrix multiplication by using fast routines of matrix multiplication and its applications," Numerical Algorithms, Vol. 59, No. 1, 2012.
[3] T. J. Dekker, "A floating-point technique for extending the available precision", Numer. Math. 18, 1971.
[4] M. Nakata, "The MPACK; Multiple precision arithmetic BLAS (MBLAS) and LAPACK (MLAPACK)", http://mplapack.sourceforge.net.
[5] Y. Hida, X. S. Li, D. H. Bailey, "Library for Double-Double and Quad-Double Arithmetic", Tech. Rep., NERSC Division, Lawrence Berkeley National Laboratory, 2007.
[6] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, P. Zimmermann, "MPFR: A multiple-precision binary floating-point library with correct rounding", ACM Trans. Math. Software 33, 2, 2007.
[7] D. Mukunoki, K. Ozaki, T. Ogita, T. Imamura, "DGEMM using Tensor Cores, and Its Accurate and Reproducible Versions," Proc. ISC High Performance 2020, LNCS, Vol. 12151, 2020.
[8] D.E. Knuth, "The Art of Computer Programming Vol. 2 Seminumerical Algorithms", Addison-Wesley, 1969.
[9] T. Kouya, "Acceleration of multiple precision matrix multiplication based on multi-component floating-point arithmetic using AVX2", arXiv:2101.06584v1, 2021.