Interval Matrix Multiplication Using Fast Low-precision Arithmetic on GPU

Katsuhisa Ozaki (Shibaura Institute of Technology)
Takeshi Ogita (Tokyo Woman's Christian University)
Daichi Mukunoki (R-CCS, Riken)

the 9th International Workshop
on Reliable Engineering Computing (REC2021)
" Risk and Uncertainty in Engineering Computations "
May 20th, 2021

Introduction and Notations

- The concept of an interval and its arithmetic, (e.g. proposed by Sunaga and Moore), has been widely applied to scientific problems.
- Uncertainness is controlled by an interval (width of interval).
- The topic is interval matrix multiplication.

- $\mathbb{F}_p$: set of p-bit floating-point numbers (IEEE 754)
  - p is omitted for general discussions
- $\mathbb{IR}$: set of real intervals
- $\mathbb{IF}$: set of intervals with floating-point numbers
- $\mathtt{fl}_p(\cdot)$: computed result by p-bit floating-point arithmetic
  - $\mathtt{fl}(\cdot)$: nearest, $\mathtt{fl}_\triangle(\cdot)$: upward, $\mathtt{fl}_\triangledown(\cdot)$: downward
- u_p: unit roundoff,
  $u_{64} = 2^{-53}$ (binary64), $u_{32} = 2^{-24}$ (binary32)

Interval and Interval Arithmetic

Inf-Sup form of Interval:

$$[a,b] = \{x \in \mathbb{R} \mid a \le x \le b,\ a,b \in \mathbb{R}\} \in \mathbb{IR}$$

$$[a,b] = \{x \in \mathbb{R} \mid a \le x \le b,\ a,b \in \mathbb{F}\} \in \mathbb{IF}$$

Mid-Rad form of Interval:

$$\langle c,r\rangle = \{x \in \mathbb{R} \mid c-r \le x \le c+r,\ c,r \in \mathbb{R}\} \in \mathbb{IR}$$

$$\langle c,r\rangle = \{x \in \mathbb{R} \mid c-r \le x \le c+r,\ c,r \in \mathbb{F}\} \in \mathbb{IF}$$

For $\mathbf{a} = [\underline{a}, \overline{a}] \in \mathbb{IF}$ and $\mathbf{b} = [\underline{b}, \overline{b}] \in \mathbb{IF}$,

$$\mathbf{a} + \mathbf{b} = [\underline{a} + \underline{b}, \overline{a} + \overline{b}] \subset [\mathtt{fl}_\triangledown(\underline{a} + \underline{b}), \mathtt{fl}_\vartriangle(\overline{a} + \overline{b})]$$

$$\begin{aligned}
\mathbf{a} \cdot \mathbf{b} &= [t_d, t_u] \subset [t'_d, t'_u], \\
t_d &= [\min(\underline{ab}, \underline{a}\overline{b}, \overline{a}\underline{b}, \overline{ab})], \\
t'_d &= [\min(\mathrm{fl}_\triangledown(\underline{ab}), \mathrm{fl}_\triangledown(\underline{a}\overline{b}), \mathrm{fl}_\triangledown(\overline{a}\underline{b}), \mathrm{fl}_\triangledown(\overline{ab}))], \\
t_u &= [\max(\underline{ab}, \underline{a}\overline{b}, \overline{a}\underline{b}, \overline{ab})], \\
t'_u &= [\max(\mathrm{fl}_\vartriangle(\underline{ab}), \mathrm{fl}_\vartriangle(\underline{a}\overline{b}), \mathrm{fl}_\vartriangle(\overline{a}\underline{b}), \mathrm{fl}_\vartriangle(\overline{ab}))].
\end{aligned}$$

Why BLAS Based ?

```
for i=1:n
    for j=1:n
        for k=1:n
            c[i][j] += a[i][k] * b[k][j];
```

Visual Studio (C), $n = 5,000$, CPU: i7-8560U.

triple loop: 1049 sec, compile option Ox: 95 sec,

loop change $(i \to k \to j)$ 26 sec,

dgemm (in Intel MKL): 2.08 sec

Interval Matrix Multiplication

For $\langle A_m, A_r \rangle,\ A_m, A_r \in \mathbb{F}^{m\times n}$ and $\langle B_m, B_r \rangle,\ B_m, B_r \in \mathbb{F}^{n\times p}$

$$\langle A_m, A_r \rangle \cdot \langle B_m, B_r \rangle \subset \langle A_m B_m,\ |A_m|B_r + A_r(|B_m| + B_r) \rangle$$

$$A_m B_m \in [\mathrm{fl}_\triangledown(A_m B_m),\ \mathrm{fl}_\vartriangle(A_m B_m)]$$

S. M. Rump. Fast and parallel interval arithmetic, BIT Numerical Mathematics, 39(3), 539–560, 1999. ⇒ Two matrix multiplications are necessary

T. Ogita, S. Oishi: Fast inclusion of interval matrix multiplication, Reliable Computing, 11(3), 191–205, 2005. ⇒ no matrix multiplication

Interval Matrix Multiplication

For $\langle A_m, A_r\rangle,\ A_m, A_r \in \mathbb{F}^{m\times n}$ and $\langle B_m, B_r\rangle,\ B_m, B_r \in \mathbb{F}^{n\times p}$

$$\langle A_m, A_r\rangle \cdot \langle B_m, B_r\rangle \subset \langle \mathrm{fl}(A_m B_m), T_2\rangle,$$

$$T_2 = |A_m|(nu|B_m| + B_r) + A_r(|B_m| + B_r).$$

K. Ozaki, T. Ogita, S. M. Rump, S. Oishi: Fast algorithms for floating-point interval matrix multiplication. Journal of Computational and Applied Mathematics, 236 (7), 1795–1814, 2012.

K. Ozaki, T. Ogita, F. Bunger, S. Oishi: Accelerating interval matrix multiplication by mixed precision arithmetic, Nonlinear Theory and its Applications, IEICE, 6 (3), 364–376, 2015. ⇒ Two matrix multiplications with low-precision

Graphics Processing Unit (GPU)

GPU is excellent resource of scientific computing.

Table 1: Theoretical Peak Performance (TFLOPS) of several GPUs by NVIDIA

GPU	FP32	FP64
RTX2070	7.5	0.23
RTX5000	11.2	0.34
V100	14.0	7.0
A6000	38.7	1.25
A100 (TC)	19.5	19.5

Graphics Processing Unit (GPU)

For floating-point matrices A and B,

$$\mathrm{fl}_{\nabla}(AB) \leq AB \leq \mathrm{fl}_{\Delta}(AB), \quad AB \in [\mathrm{fl}_{\nabla}(AB), \mathrm{fl}_{\Delta}(AB)]$$

is often used for interval matrix multiplication.

However, it cannot be applied for built-in libraries such as cuBLAS.

For the scalars, we can compute $x + y$ by

fadd_[rn, rz, ru, rd] (x, y).

The goal

- We exploit fast low-precision arithmetic on GPU: e.g., RTX2070, RTX5000, and A6000
  - price is cheaper compared to GPU (high performance FP64)
- using cuBLAS (with rounding-to nearest mode)
- fast (computing time) and tight intervals

The radius (blue color) is computed by the same way.

Error-Free Transformation of Matrix Multiplication

For $A \in \mathbb{F}^{m\times n}$, $B \in \mathbb{F}^{n\times p}$, the matrices are divided such that

$$A = A^{(1)} + A^{(2)} + \cdots + A^{(k)}, \ \ B = B^{(1)} + B^{(2)} + \cdots + B^{(\ell)},$$

in order to satisfy

$$A^{(i)}B^{(j)} = \mathtt{fl}\left(A^{(i)}B^{(j)}\right).$$

Then,

$$AB = \sum_{i=1}^{k}\sum_{j=1}^{\ell} \mathtt{fl}\left(A^{(i)}B^{(j)}\right).$$

The first paper:

K. Ozaki, T. Ogita, S. Oishi, S. M. Rump: Error-Free Transformation of Matrix Multiplication by Using Fast Routines of Matrix Multiplication and its Applications, Numerical Algorithms, 59(1), 95–118, 2012.

Recent paper using GPU:

D. Mukunoki, K. Ozaki, T. Ogita, T. Imamura: DGEMM Using Tensor Cores, and Its Accurate and Reproducible Versions, Lecture Notes in Computer Science, 12151, 2020, 230-248.

Error-Free Transformation of Matrix Multiplication

For $A_m \in \mathbb{F}_{64}^{m\times n}$, $B_m \in \mathbb{F}_{64}^{n\times p}$, we find s and t such that

$$\begin{aligned} A_m &= A_m^{(1)} + A_m^{(2)} + \cdots + A_m^{(s)} + \cdots + A_m^{(k)}, \\ B_m &= B_m^{(1)} + B_m^{(2)} + \cdots + B_m^{(t)} + \cdots + B_m^{(\ell)}, \\ A^{(i)} &\in \mathbb{F}_{32}^{m\times n}, \quad B^{(j)} \in \mathbb{F}_{32}^{n\times p}, \quad |\underline{A}_m^{(s+1)}| \lesssim A_r, \quad |\underline{B}_m^{(t+1)}| \lesssim B_r \end{aligned}$$

$$\mathbf{A} = \langle A_m, A_r \rangle \subset \left\langle \sum_{i=1}^{s} A_m^{(i)},\ A_r + |\sum_{i=s+1}^{k} A_m^{(i)}| \right\rangle,$$

$$\mathbf{B} = \langle B_m, B_r \rangle \subset \left\langle \sum_{j=1}^{t} B_m^{(j)},\ B_r + |\sum_{j=t+1}^{\ell} B_m^{(j)}| \right\rangle$$

The mid-point is enclosed by

$$\sum_{i=1}^{s} \sum_{j=1}^{t} A_m^{(i)} B_m^{(j)} \in \left\langle \sum_{i+j \le \min(s,t)+1} A_m^{(i)} B_m^{(j)},\ \sum_{i+j > \min(s,t)+1} |A_m^{(i)}||B_m^{(j)}| \right\rangle$$

For a product of matrices with non-negative entries

$(A = |A| \in \mathbb{R}^{m \times n},\ B = |B| \in \mathbb{R}^{n \times p})$

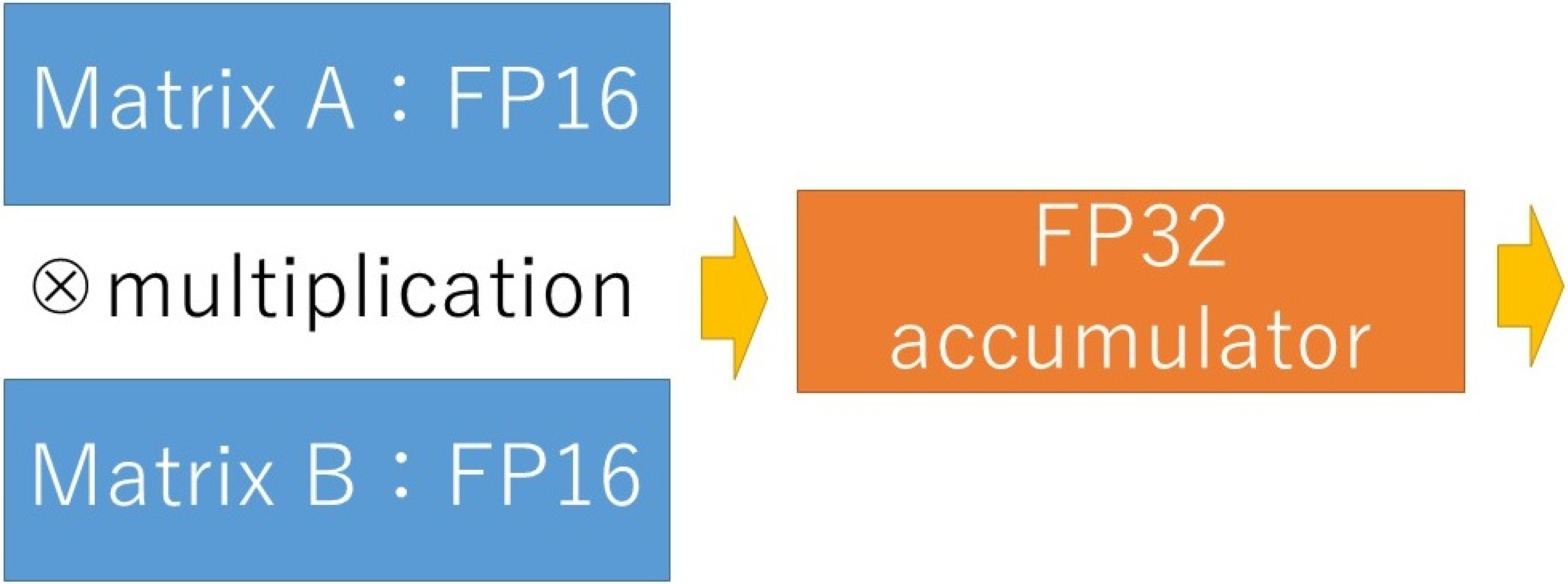
$$
\begin{aligned}
g_j &= \max_i |a_{ij}|, \quad h_i = \max_j |b_{ij}|, \\
e &= (1, \ldots, 1)^T \in \mathbb{R}^m, \quad f = (1, \ldots, 1)^T \in \mathbb{R}^p
\end{aligned}
$$

Then,

$$AB \le \min\left(e(g^T B), (Ah) f^T\right)$$

T. Ogita, S. Oishi: Fast Inclusion of Interval Matrix Multiplication. Reliable Comput. 11, 191–205 (2005).

Tensor Core



The performance is 2 - 8 times faster than native FP32.

$$\sum_{i+j\le\min(s,t)+1} A_m^{(i)}B_m^{(j)} = \sum_{i+j\le\min(s,t)+1} \mathtt{fl}_{32}\left(A_m^{(i)}B_m^{(j)}\right)$$

$$= \sum_{i+j\le\min(s,t)+1} P^{(i)^{-1}}\mathtt{fl}_{32tc}\left((P^{(i)}A_m^{(i)})(B_m^{(j)}Q^{(j)})\right)Q^{(j)^{-1}},$$

where $P^{(i)}$ and $Q^{(j)}$ are diagonal matrices such that

$$P^{(i)}A_m^{(i)} \in \mathbb{F}_{16}^{m\times n}, \quad B_m^{(j)}Q^{(j)} \in \mathbb{F}_{16}^{n\times p}.$$

$$\sum_{i+j \le \min(s,t)+1} A_m^{(i)} B_m^{(j)} = \sum_{i=1}^{w} T^{(i)}, \quad T^{(i)} \in \mathbb{F}_{32}^{m \times p}$$

$$S^{(i)} := \mathrm{fl}_{64}\left(S^{(i-1)} + T^{(i)}\right), \quad S^{(0)} = \mathbf{O}$$

$$R^{(i)} := \mathrm{fl}_{64}\left((1 + 2u_{64})(R^{(i-1)} + u_{64}|S^{(i)}|)\right), \quad R^{(0)} = \mathbf{O}$$

for $i = 1, 2, \ldots$ (no underflow is assumed). Then

$$\sum_{i+j \le \min(s,t)+1} A_m^{(i)} B_m^{(j)} \in \langle S^{(w)},\ R^{(w)} \rangle.$$

The technique multiplying $(1 + 2u_{64})$ is introduced in INTLIB.

Numerical Examples

Interval matrices $\langle A_m, A_r \rangle$ and $\langle B_m, B_r \rangle$ were generated using MATLAB R2021a as follows

$$
\begin{aligned}
A_m &= \texttt{randn}(n); \\
A_r &= c_1 * \texttt{rand}(n). * \texttt{abs}(A_m); \\
B_m &= \texttt{randn}(n); \\
B_r &= c_2 * \texttt{rand}(n). * \texttt{abs}(B_m);
\end{aligned}
$$

Benchmark for GPU

Table 2: Performance (TFLOPS) on several GPUs by NVIDIA ($n = 10,000$)

GPU	FP16 (TC*)	FP32	FP64
RTX2070	15.1	7.20	0.24
RTX5000	14.8	7.19	0.40
V100	21.0	12.7	5.90
A6000	55.1	15.4	0.56
A100	124	15.5	13.8

We checked the performance of matrix multiplication using MATLAB R2021a.

CUDA version 11.3, *Including diagonal scaling

Numerical Examples

M1: Rump: BIT Numerical Mathematics, 1999 (for comparison)

M2: Ozaki et al.: Journal of Computational and Applied Mathematics, 2012, One binary64 and two binary32 matrix multiplications

We call cublasGemmEx in cuBLAS from MATLAB R2021a.

M2

For $\langle A_m, A_r \rangle,\ A_m, A_r \in \mathbb{F}^{m\times n}$ and $\langle B_m, B_r \rangle,\ B_m, B_r \in \mathbb{F}^{n\times p}$

$$\langle A_m, A_r \rangle \cdot \langle B_m, B_r \rangle \subset \langle \mathrm{fl}_{64}(A_m B_m), T_2 \rangle,$$

$$T_2 = |A_m|(nu|B_m| + B_r) + A_r(|B_m| + B_r).$$

One binary64 matrix multiplication

Two binary32 matrix multiplications

Numerical Examples

Table 3: Computing time (sec), $n = 10,000$

$c_1 = c_2$	M2	Proposed	Speedup
10^{-13}	4.74	4.09	1.15
10^{-10}	4.75	3.49	1.36
10^{-07}	4.74	2.61	1.81
10^{-04}	4.74	2.03	2.33

MATLAB R2021a, RTX A6000, CUDA Toolkit 11.3, including data transfer time (CPU $\leftrightarrow$ GPU)

Numerical Examples

Table 4: Computing time (sec), $n = 10,000$

$c_1 = c_2$	M2	Proposed	Speedup
10^{-13}	4.43	3.36	1.31
10^{-10}	4.44	2.44	1.81
10^{-07}	4.47	2.05	2.17
10^{-04}	4.48	1.42	3.14

MATLAB R2021a, RTX A6000, CUDA Toolkit 11.3, excluding data transfer time (CPU $\leftrightarrow$ GPU)

Numerical Examples

Table 5: Maximum of computed radii for $n = 10,000, c_1 = c_2$

$c_1 = c_2$	M1	M2	Proposed
10^{-13}	6.93e-10	8.25e-09	8.87e-10
10^{-10}	6.87e-07	6.94e-07	6.98e-07
10^{-07}	6.91e-04	6.91e-04	7.38e-04
10^{-04}	6.86e-01	6.87e-01	9.04e-01

Future Work

- Extension to product of three matrices
- Verified numerical computations for eigen problems
- Memory-reduced implementation
- Improvement of coding (complete code for CUDA)

Conclusion

- We focused on interval matrix multiplication using GPU.
- We exploit fast low-precision arithmetic for interval matrix multiplication.
- Acceleration is from 1.1 to 3.1.

Thank you for your attention!

Additional Data

Numerical Examples

Table 6: Computing time (sec), $n = 15,000$

$c_1 = c_2$	M2	Proposed	Speedup
1e-13	14.9	11.5	1.29
1e-10	14.8	9.97	1.49
1e-07	14.7	7.16	2.05
1e-04	14.8	5.24	2.83

MATLAB R2021a, RTX A6000, CUDA Toolkit 11.3, including data transfer time (CPU $\leftrightarrow$ GPU)

Numerical Examples

Table 7: Computing time (sec), $n = 15,000$

$c_1 = c_2$	M2	Proposed	Speedup
1e-13	14.2	10.0	1.42
1e-10	14.3	7.18	2.00
1e-07	14.4	5.95	2.42
1e-04	14.4	4.00	3.59

MATLAB R2021a, RTX A6000, CUDA Toolkit 11.3, excluding data transfer time (CPU $\leftrightarrow$ GPU)

Numerical Examples

Table 8: Maximum of computed radii for $n = 15,000, c_1 = c_2$

c_1	M1	M2	Proposed
1e-13	1.02e-09	1.78e-08	1.36e-09
1e-10	1.01e-06	1.03e-06	1.03e-06
1e-07	1.01e-03	1.01e-03	1.10e-03
1e-04	1.01e+00	1.02e+00	1.35e+00