

JHPCN2020年度採択課題(jh200008-NAH)

# Developing Accuracy Assured High Performance Numerical Libraries for Eigenproblems

名古屋大学情報基盤センター

片桐孝洋

第12回 自動チューニング技術の現状と応用に関するシンポジウム(ATTA2020)

日時: 2020年12月25日(金) 13:30~14:00

場所:オンライン開催

# 2020年度 JHPCN採択課題

- タイトル
  - Developing Accuracy Assured High Performance Numerical Libraries for Eigenproblems
- 期間
  - 2020年4月～2021年3月 (1年間)
  - 3年計画の2年目
- 構成
  - 副代表：
    - Weichung Wang (National Taiwan U.),  
Takeshi Ogita (Tokyo Women's U.)
  - 主な構成員：
    - K. Nakajima (U. of Tokyo), S. Ohshima (Nagoya U.),  
K. Ozaki (Shibaura I.T.), O. Marques (LBNL),  
F-N. Hwang (National Central U.)

# Interdisciplinary Research

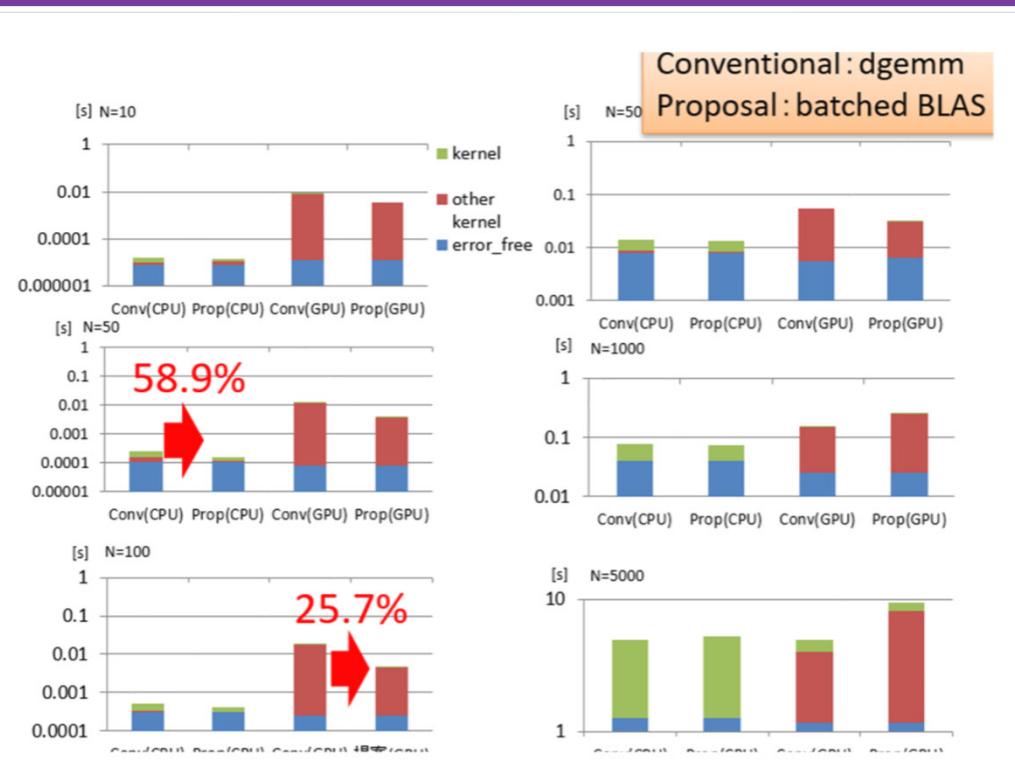
# Project Overview

Researchers for  
Accuracy Assurance  
Problems

Researchers for  
High-performance  
Computing

Researchers for Eigenproblems

## (2) Developing Assured Libraries to CPU and GPU



## (1) Assured algorithm

**Algorithm 2** RefSyEvCL: Refinement of approximate eigenvectors of a real symmetric matrix with clustered eigenvalues. Higher-precision arithmetic is required for all the computations except line 11 and the computations of  $\|\cdot\|_2$ .

```

Input:  $A, \widehat{X} \in \mathbb{R}^{n \times n}$  with  $A = A^T$ 
Output:  $X' \in \mathbb{R}^{n \times n}$ 
1: function  $X' \leftarrow \text{RefSyEvCL}(A, \widehat{X})$ 
2:    $[X', \tilde{D}, \tilde{E}, \omega] \leftarrow \text{RefSyEv}(A, \widehat{X})$                                 ▷ Apply Algorithm 1 to  $A$  and  $\widehat{X}$ .
3:   if  $\|\tilde{E}\|_2 \geq 1$ , return, end if                                         ▷ Improvement cannot be expected.
4:   Determine the index sets  $\mathcal{J}_k$ ,  $k = 1, \dots, n_{\mathcal{J}}$ , as in (27) for eigenvalue clusters using
    $\tilde{D} = \text{diag}(\tilde{\lambda}_i)$  and  $\omega$ .                                         ▷  $n_{\mathcal{J}}$ : The number of clusters.
5:   for  $k \leftarrow 1, 2, \dots, n_{\mathcal{J}}$  do                                         ▷ Refine eigenvectors for each cluster.
6:      $V_k \leftarrow X'(:, \mathcal{J}_k)$                                               ▷ Pick out  $V_k \in \mathbb{R}^{n \times n_k}$  where  $n_k := |\mathcal{J}_k|$ .
7:      $\mu_k \leftarrow (\min_{i \in \mathcal{J}_k} \tilde{\lambda}_i + \max_{i \in \mathcal{J}_k} \tilde{\lambda}_i)/2$           ▷ Determine the shift constant  $\mu_k$ .
8:      $A_k \leftarrow A - \mu_k I$                                                  ▷ Shift  $A$  for separating clustered eigenvalues.
9:      $C_k \leftarrow V_k^T A_k V_k$                                               ▷ Compute  $C_k \in \mathbb{R}^{n_k \times n_k}$ .
10:     $T_k \leftarrow \text{fl}(C_k)$                                                  ▷ Conversion from higher-precision to ordinary precision.
11:     $[W_k, \sim] \leftarrow \text{eig}(T_k)$                                          ▷ Compute eigenvectors of  $C_k$  in ordinary precision.
12:     $\nu \leftarrow 1$ ;  $V_k^{(1)} \leftarrow V_k \cdot W_k$ 
13:    repeat
14:       $[V_k^{(\nu+1)}, \sim, \tilde{E}_k] \leftarrow \text{RefSyEv}(A_k, V_k^{(\nu)})$           ▷ Apply Alg. 1 to  $A_k$  and  $V_k^{(\nu)}$ .
15:      if  $\|\tilde{E}_k\|_2 \geq 1$ , return, end if                                         ▷ Improvement cannot be expected.
16:       $\nu \leftarrow \nu + 1$ 
17:    until  $\|\tilde{E}_k\|_2 \leq \|\tilde{E}\|_2$ 
18:     $X'(:, \mathcal{J}_k) \leftarrow V_k^{(\nu)}$                                          ▷ Update the columns of  $X'$  corresponding to  $\mathcal{J}_k$ .
19:  end for
20: end function

```



## Adaptation of Supercomputing

Fig. 1 : Batched BLAS implementation for Ozaki Method on a CPU and a GPU (Results from FY2019)

# UNC-HPCライブラリ

- ポスト「京」萌芽的課題1 基礎科学のフロンティア—極限への挑戦—「極限の探究に資する精度保証付き数値計算学の展開と超高性能計算環境の創成」（代表：荻田武史 東京女子大学教授）～2019で開発
- 精度が保証された計算結果を実用的に得られるような超高性能計算環境を構築
- 以下のライブラリのソースコードを公開中
  1. OzBLAS: Accurate and Reproducible BLAS based on Ozaki scheme (尾崎スキームに基づく高精度基本線形計算ライブラリ) [for PC, GPU]
  2. GEMMTC: GEMM using Tensor Cores (Tensorコアを用いた行列積プログラム) [for GPU]
  3. DHPMM\_F for GPU: High-precision Matrix Multiplication with Faithful Rounding (高精度行列積プログラム) [for GPU]
  4. PDDOTK: K-fold Precision Dot Product (分散並列版高精度内積計算プログラム) [for PC, FX100]
  5. BLAS-DOT2: Higher-precision BLAS based on Dot2 (高精度内積計算アルゴリズムDot2に基づく基本線形計算ライブラリ) [for GPU]
  6. LINSYS\_VR: Verified Solution of Linear Systems with Directed Rounding (連立一次方程式に対する精度保証プログラム(丸めモードの変更機能付き)) [for K Computer, FX100]
  7. LINSYS\_V: Verified Solution of Linear Systems (連立一次方程式に対する精度保証プログラム) [for PC, K Computer, FX100]
  8. DHPMM\_F: High-precision Matrix Multiplication with Faithful Rounding (高精度行列積プログラム) [for PC, K Computer, FX100]

HP : <http://www.math.twcu.ac.jp/ogita/post-k/index.html>

Verified Numerical Computations  
**UNC - HPC**  
High-Performance Computing

ポスト「京」萌芽的課題1  
基礎科学のフロンティア — 極限への挑戦  
極限の探究に資する精度保証付き数値計算学の展開と超高性能計算環境の創成

トップページ Top 課題概要 Abstract 研究体制 System 研究内容 Contents 成果公開 Results

リンク

- トップページ
- 課題概要
- 研究体制
- 研究内容
- 成果公開

課題概要

本研究課題の目的は、ポスト「京」において、精度が保証された計算結果を実用的に得られるような超高性能計算環境を構築することです。具体的には、ポスト「京」で実行される様々な数値シミュレーションにおいて、数値計算による計算誤差の問題が解消されることによって、シミュレーションサイエンスの品質を向上させ、さらに想定外の現象が発生する可能性を低減することが可能となります。すなわち、本研究課題の遂行は、人が安心して生活できる社会基盤の構築に直結します。

たとえば、産業界における製品開発や非破壊検査等のための計算工学シミュレーションや、地震や津波などの災害シミュレーションは、日本に限らず世界中で盛んに行われていますが、計算誤差の観点から高精度なシミュレーションの実現をしている例は甚わずです。これは、計算機によって問題の近似的な解を得ることよりも、その近似解の検算のほうはるかに困難かつ計算資源を必要とする、と考えられており、それが高性能計算分野の常識であるからです。実際、これは1990年代までは事実でしたが、本研究グループが2000年代から切り拓いてきた高速で実用的な精度保証付き数値計算法やエラーフリー変換に基づく高精度数値計算法をベースとして、今、ポスト「京」によって、この常識を打ち破る時期が到来しています。すなわち、ポスト「京」において、高速性と高精度性を融合した超高性能計算環境の創成を世界に先駆けて達成することは、スーパーコンピュータに質的転換をもたらし、我が国の高い科学技術力を国内外に示すことになります。

精度 (高信頼性)  
エネルギー効率 (省エネ)  
速度 (高速性)  
Top500, Graph500, ...

性能 = 速度 × 精度

大規模数値計算を行う高性能計算分野においては、問題の大規模化に伴って計算誤差が累積しやすくなり、これが今後、大きな問題となってきます。たとえば、計算機上で標準的に用いられる32ビットの浮動小数点演算では、100万次元程度の密行列系線形問題に対して数値計算を行うと、問題自身は比較的の良条件で解きやすい問題であったとしても、誤差解析の結果から理論的には1桁も正しくないような計算結果が得られることが分かっています。また、計算誤差の単なる累積だけでなく、問題の困難さが条件数として

# Aim of This Study

To establish accuracy assured numerical computing, we focus on the following three topics:

1. Developing an accuracy assured numerical libraries for eigenproblems;
2. Development of high-performance implementation and auto-tuning technology for the developed accuracy assured numerical library;
3. Discussing an extension to non-liner problems based on obtained knowledge of accuracy assured algorithms.

芝浦工業大学 尾崎 克久 教授  
東京女子大 萩田 武史 教授

# 連立一次方程式の精度保証 (FY2019の成果)

# 残差反復

- 高精度内積計算(疑似4倍精度)を用いた  
残差反復で大規模連立一次方程式の真の解は  
数値計算で求まるか？の検証実験
  - 175万元の連立一次方程式に対して2500ノードを使用
  - LU分解を用いて近似解を求め、残差反復を行ったとき
  - 1回目の残差ノルム: 4.019007e-14
  - 2回目の残差ノルム: 0.000000e+00
- となり、2回の残差反復により**真の解**自体が得られた。
- 大規模計算においても、高精度内積を用いた残差反復法の有用性を検証できた。

# 連立一次方程式

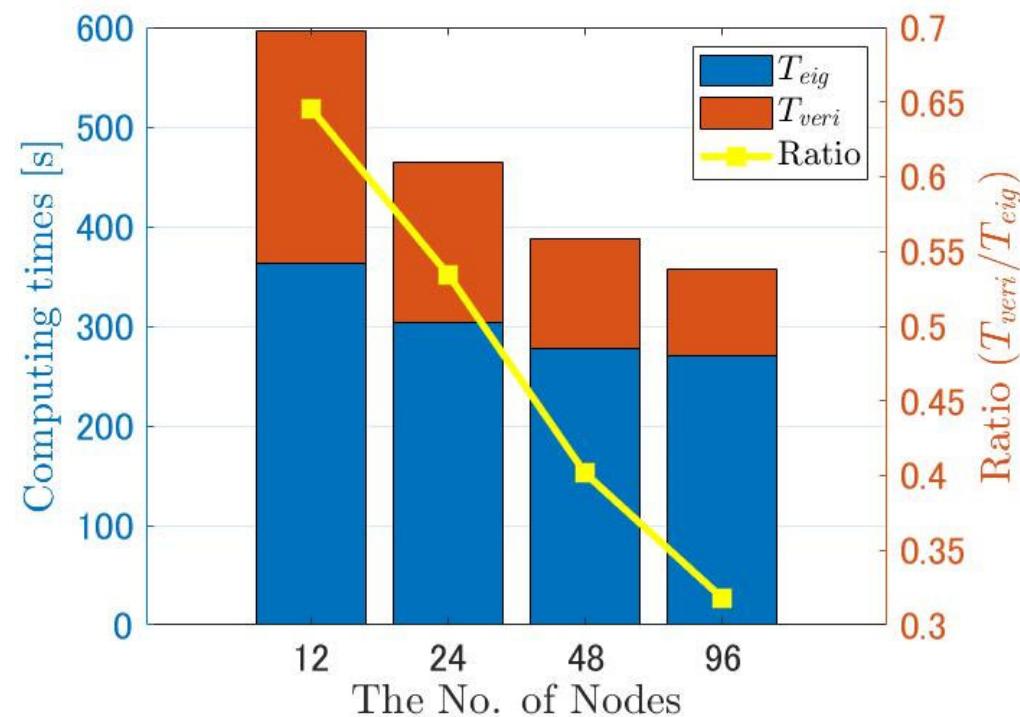
- ・ 連立一次方程式の精度保証付き数値計算
- ・ 残差反復により近似解の精度を改善し、精度保証を行った
- ・ 真の解がおよそ $(1, 1, \dots, 1)^T$ の場合、  
名古屋大学Fujitsu PRIMEHPC FX100  
**(2020年3月末退役済)**を2500ノード使用で
  - 100万元の問題、誤差の上限が $1.111484e-16$
  - 150万元の問題、誤差の上限が $1.113360e-16$   
(ブロック計算を用いてメモリを節約)
- ・ となり、**ほぼ最終ビットまで近似解が正しいことを保証**

理化学研究所 計算科学研究センター 寺尾 剛史 特別研究員  
芝浦工業大学 尾崎 克久 教授  
東京女子大 萩田 武史 教授

# **標準固有値問題の精度保証 (FY2019の成果)**

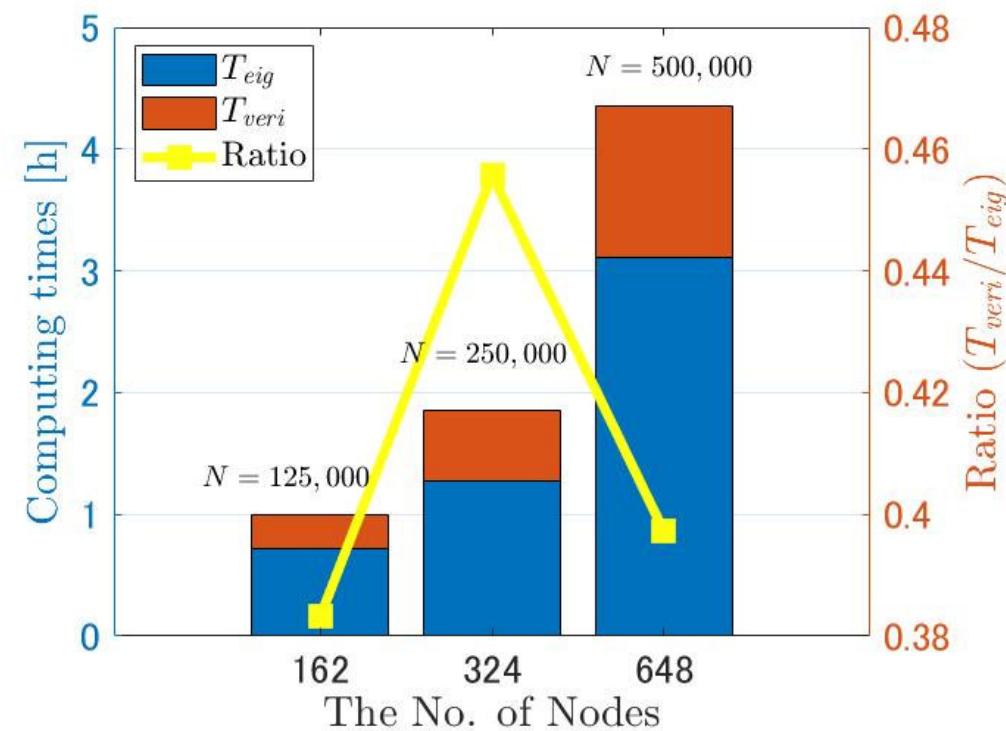
# 数値実験結果(計算性能)

- 名古屋大学FX100  
※2020年3月退役済み
- $n = 50,000$
- PDSYEVD (ScaLAPACK)
- テスト行列: 各要素が  $[0, 1]$  区間の一様分布な 実対称行列
- 大規模並列環境にて 高い強スケーラビリティを 持つ
- ScaLAPACKの計算より 高速に精度保証が可能
- その他の固有値ソルバにも応用 が可能



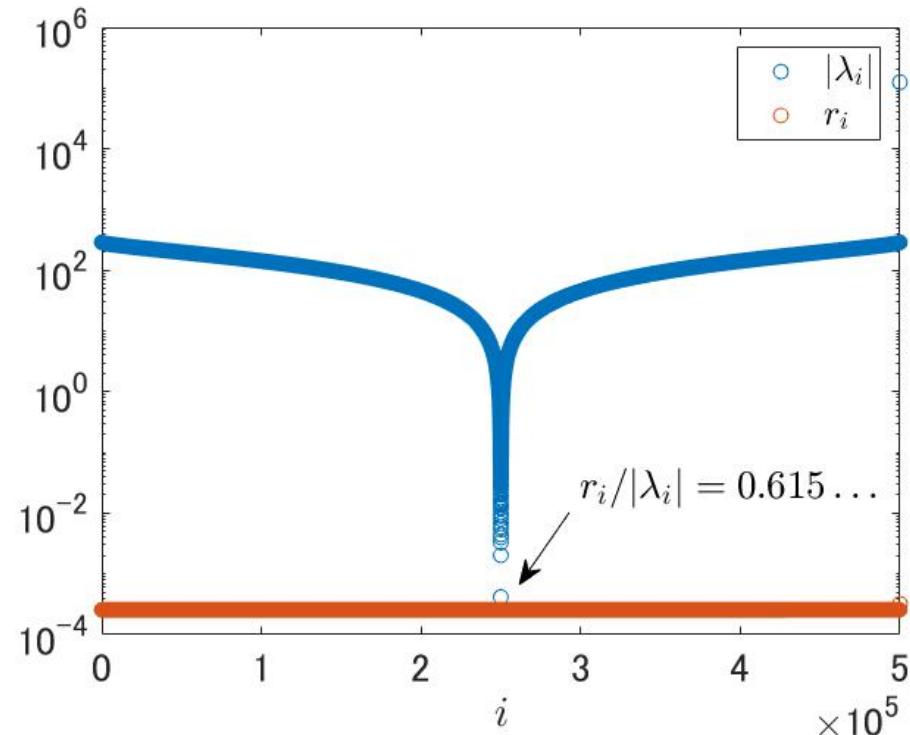
# 数値実験結果（計算性能）

- 名古屋大学FX100  
※2020年3月退役済み
- $n = 125,000 \sim 500,000$
- PDSYEVD (ScalAPACK)
- テスト行列：各要素が  $[0, 1]$  区間の一様分布な実対称行列
- 大規模密行列に対して 4~5割程度の 計算時間で精度保証が可能である。



# 数値実験結果(精度)

- 名古屋大学FX100  
※2020年3月退役済み
- $n = 500,000$
- PDSYEVD (ScaLAPACK)
- $\lambda_i : i$ 番目に小さい近似固有値
- $r_i$  : 精度保証で得られた  $\lambda_i$  の誤差上限
- テスト行列 : 各要素が  $[0, 1]$  区間の一様分布な対称行列
- 50万元の乱数行列に対して、相対誤差の上限は、絶対値最小の固有値に対して6割強である。
- よって、重複固有値の非存在を証明できた



名古屋大学 大島 聰史 準教授  
片桐 孝洋 教授

# 高精度行列-行列積(尾崎の方法) のGPU実装

7th International Workshop on Large-scale HPC Application Modernization  
(one of co-located workshop of CANDAR'20)  
2020.11.25 (onlineで発表)  
In Proc. of CANDARW2020, IEEE Xplore, (2020)

# Performance Evaluation of Accurate Matrix–Matrix Multiplication on GPU Using Sparse Matrix Multiplications

---

Fumiya Ishiguro<sup>1</sup>, Takahiro Katagiri<sup>2</sup>, Satoshi Ohshima<sup>2</sup>,  
Toru Nagai<sup>2</sup>

1: Graduate School of Informatics, Nagoya University

2: Information Technology Center, Nagoya University

# Background, objective, and motivation of our work

- High-accuracy and low-accuracy calculations are one of the important calculation techniques which can solve large-scale and complicated calculations.
- Some studies have investigated the accuracy assurance of BLAS and LAPACK.
  - These studies have used mixed procedure computations and arbitrary digit computations.
  - Most of BLAS and LAPACK libraries tend to place less emphasis on the accuracy of the computation results.
- Why high-accuracy and assured calculations are not widespread?
- => TIME
- We consider to calculate it on GPU and shorten the time.
- This work will be helpful for large-scale and complicated calculations on current and future generation computers. (➡ topics of interest of this workshop)

In particular, we will focus on the following topics of interest, but not limited to:

- Programming models, languages and frameworks for facilitating HPC software evolution and refactoring.
- Algorithms and implementation methodologies for future-generation computing systems, including manycores and accelerators (GPUs, Xeon Phi, etc).
- Automatic performance tuning techniques, runtime systems and domain-specific languages for hiding the complexity of underlying system architectures.
- Practices and experiences on porting of legacy applications and libraries.

# High-precision matrix-matrix multiplication algorithm

- Our target calculation: assured matrix-matrix multiplication (MMM) method proposed by Ozaki et al.
  - hereinafter, we refer to this MMM method as **the Ozaki method**
- Overview of the Ozaki method:

consider  $C = AB$

- $A$ : a matrix of size  $m * l$
- $B$ : a matrix of size  $l * n$
- $C$ : a matrix of size  $m * n$

## Step1: error-free transformation

$$A = A^{(1)} + A^{(2)} + A^{(3)} + \dots + A^{(p)}$$

$$B = B^{(1)} + B^{(2)} + B^{(3)} + \dots + B^{(q)}$$

The elements in the matrices with lower indices are given with a higher number of digits.

## Step2: individual MMM

$$\begin{aligned} AB &= (A^{(1)} + A^{(2)} + \dots + A^{(p)}) (B^{(1)} + B^{(2)} + \dots + B^{(q)}) \\ &= \underline{\underline{A^{(1)}}} \underline{\underline{B^{(1)}}} + \underline{\underline{A^{(1)}}} \underline{\underline{B^{(2)}}} + \underline{\underline{A^{(2)}}} \underline{\underline{B^{(1)}}} + \dots + \underline{\underline{A^{(p)}}} \underline{\underline{B^{(q)}}} \end{aligned}$$

## Step3: Accurate Sum

$$\begin{aligned} fl(A^{(i)} B^{(j)}) &= A^{(i)} B^{(j)} \text{ for } 1 \leq i \leq p, 1 \leq j \leq q. \\ &= fl(A^{(1)} B^{(1)}) + fl(A^{(1)} B^{(2)}) + fl(A^{(2)} B^{(1)}) + \dots \\ &\quad + fl(A^{(p)} B^{(q)}) \\ &= C_1 + C_2 + \dots + C_{pq} \end{aligned}$$

$fl$  is a floating-point arithmetic with rounding to the nearest

# Example of $C=AB$ by double precision (causes some errors)

$$A = \begin{bmatrix} 2^0 + 2^{-30} & 2^{-20} + 2^{-40} & 2^{-10} + 2^{-15} \\ 2^{-10} + 2^{-30} & 2^{-15} + 2^{-35} & 2^1 + 2^{-15} \\ 2^{-5} + 2^{-25} & 2^0 + 2^{-30} & 2^5 + 2^{-30} \end{bmatrix}$$

$$B = \begin{bmatrix} 2^0 + 2^{-30} & 2^{-35} + 2^{-60} & 2^5 + 2^{-15} \\ 2^{-5} + 2^{-10} & 2^{-30} + 2^{-40} & 2^{-10} + 2^{-30} \\ 2^{-3} + 2^{-15} & 2^{-40} + 2^{-50} & 2^{-8} + 2^{-17} \end{bmatrix}$$

$$\begin{aligned} C &= A B \\ &\text{first row of } A * \text{first column of } B \\ &= (2^0 + 2^{-30})(2^0 + 2^{-30}) + (2^{-20} + 2^{-40})(2^{-5} + 2^{-10}) + \\ &\quad (2^{-10} + 2^{-15})(2^{-3} + 2^{-15}) \end{aligned}$$



$$2^0 + 2^{-29} + 2^{-29} + 2^{-60}$$

loss of trailing digits may be happened

# Example of the Ozaki method

## Step1: error-free transformation

$$A = \begin{bmatrix} 2^0 + 2^{-30} & 2^{-20} + 2^{-40} & 2^{-10} + 2^{-15} \\ 2^{-10} + 2^{-30} & 2^{-15} + 2^{-35} & 2^1 + 2^{-15} \\ 2^{-5} + 2^{-25} & 2^0 + 2^{-30} & 2^5 + 2^{-30} \end{bmatrix}$$



$$B = \begin{bmatrix} 2^0 + 2^{-30} & 2^{-35} + 2^{-60} & 2^5 + 2^{-15} \\ 2^{-5} + 2^{-10} & 2^{-30} + 2^{-40} & 2^{-10} + 2^{-30} \\ 2^{-3} + 2^{-15} & 2^{-40} + 2^{-50} & 2^{-8} + 2^{-17} \end{bmatrix}$$



$$A^{(1)} = \begin{bmatrix} 2^0 & 2^{-20} & 2^{-10} + 2^{-15} \\ 2^{-10} & 2^{-15} & 2^1 + 2^{-15} \\ 2^{-5} & 2^0 & 2^5 \end{bmatrix}$$

$$B^{(1)} = \begin{bmatrix} 2^0 & 2^{-35} & 2^5 + 2^{-15} \\ 2^{-5} + 2^{-10} & 2^{-30} + 2^{-40} & 2^{-10} \\ 2^{-3} + 2^{-15} & 2^{-40} + 2^{-50} & 2^{-8} + 2^{-17} \end{bmatrix}$$

$$A^{(2)} = \begin{bmatrix} 2^{-30} & 2^{-40} & 0 \\ 2^{-30} & 2^{-35} & 0 \\ 2^{-25} & 2^{-30} & 2^{-30} \end{bmatrix}$$

$$B^{(2)} = \begin{bmatrix} 2^{-30} & 2^{-60} & 0 \\ 0 & 0 & 2^{-30} \\ 0 & 0 & 0 \end{bmatrix}$$

## Step2: individual MMM,

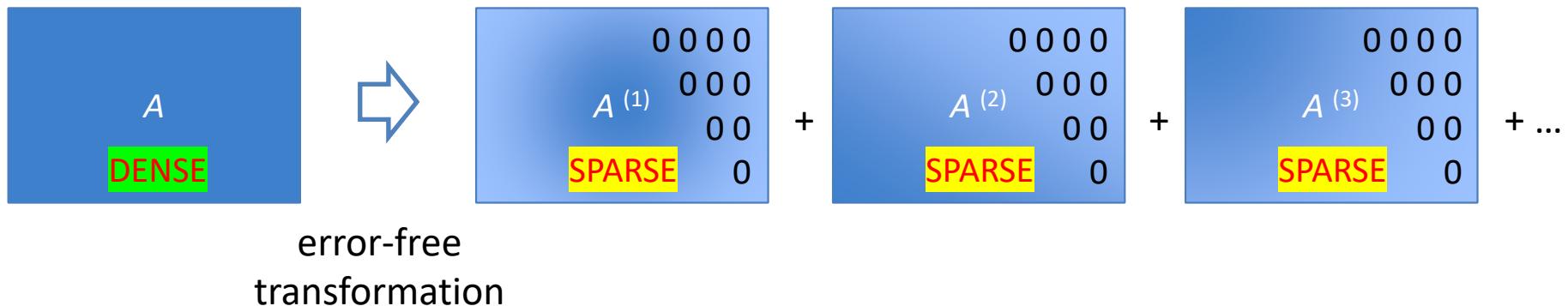
## Step3: Accurate Sum

$$C = A^{(1)} B^{(1)} + A^{(2)} B^{(1)} + A^{(1)} B^{(2)} + A^{(2)} B^{(2)}$$

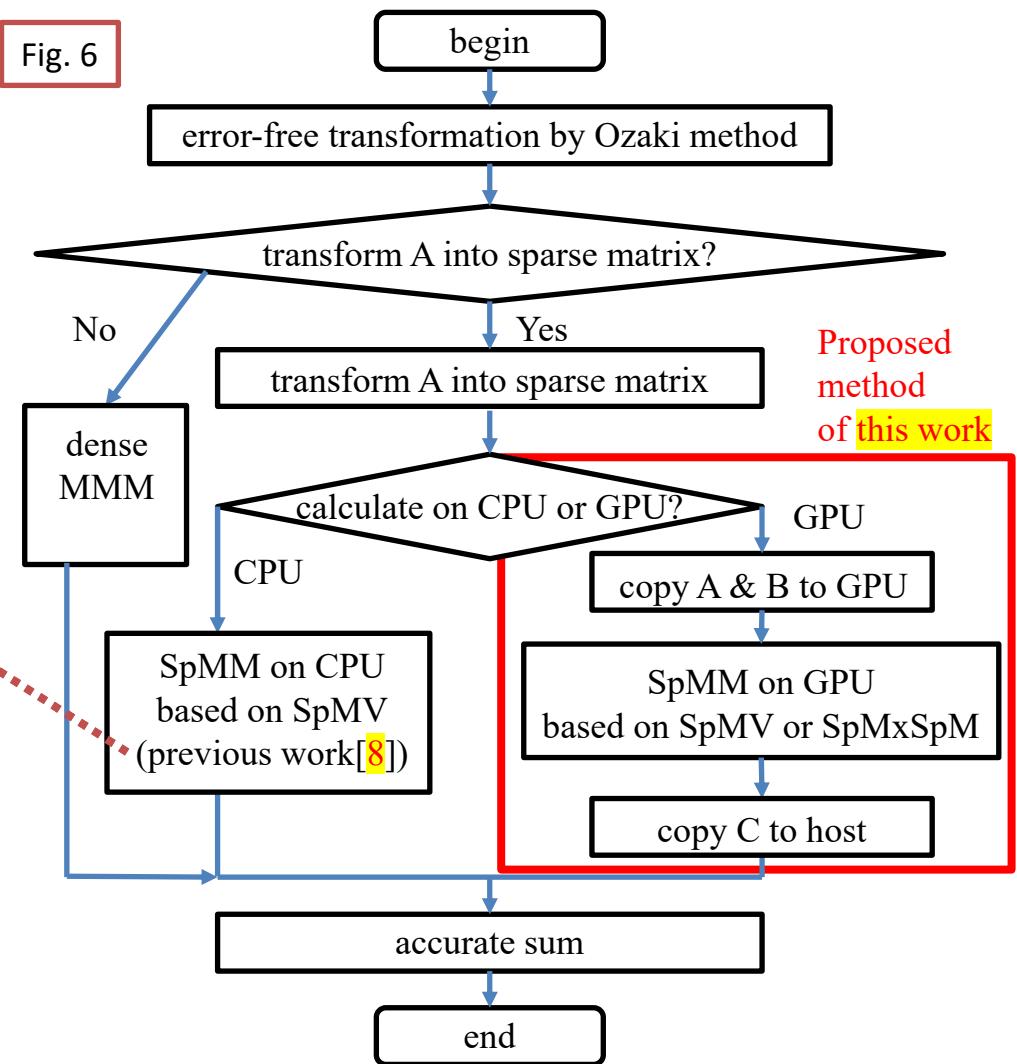
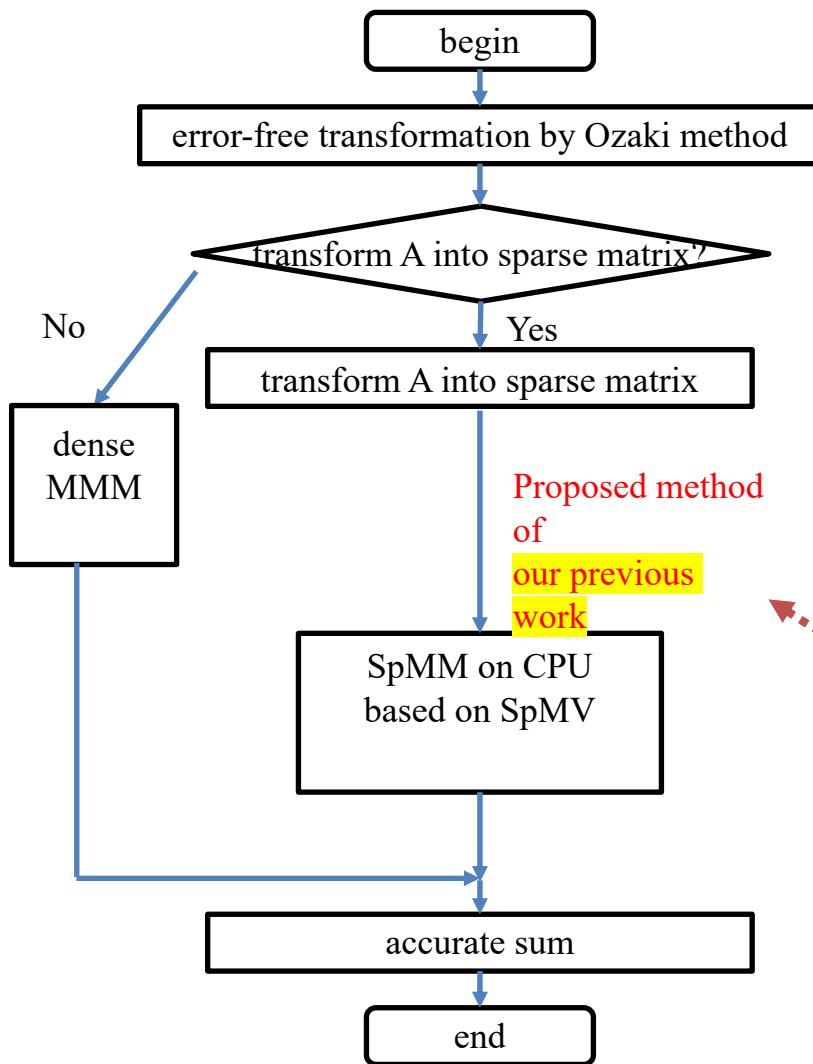
- each calculation is executed in the 53bit range
- no round-off error, no loss of trailing digits

# Previous work and Proposed method

- When the value ranges of the input matrix elements are large, error-free transformation generates many sparse matrices.



- In this case, many double precision general matrix - matrix multiplication (dgemm) are performed. Transforming dense matrices into sparse matrices and **performing sparse matrix operations will require shorter calculation time than dense matrix operations.**
- Therefore, our previous work proposed to transform the target matrices into sparse matrices and calculate sparse matrix computations **on CPU**.
  - Considering the performance, sparse matrix - vector multiplications (SpMV) are used.
- In this study, we propose to calculate these sparse matrix computations **on GPU**.



# Sparse matrix formats

- CRS format

$$A = \begin{bmatrix} a_1^1 & b_2^2 & c_3^3 & 0 \\ 0 & 0 & 0 & d_4^4 \\ e_1^5 & 0 & 0 & f_4^6 \\ 0 & 0 & g_3^7 & 0 \end{bmatrix}$$

```
val    = [ a b c d e f g ]  
colind = [ 1 2 3 4 1 4 3 ]  
rowptr = [ 1 4 5 7 8 ]
```

- ELL format

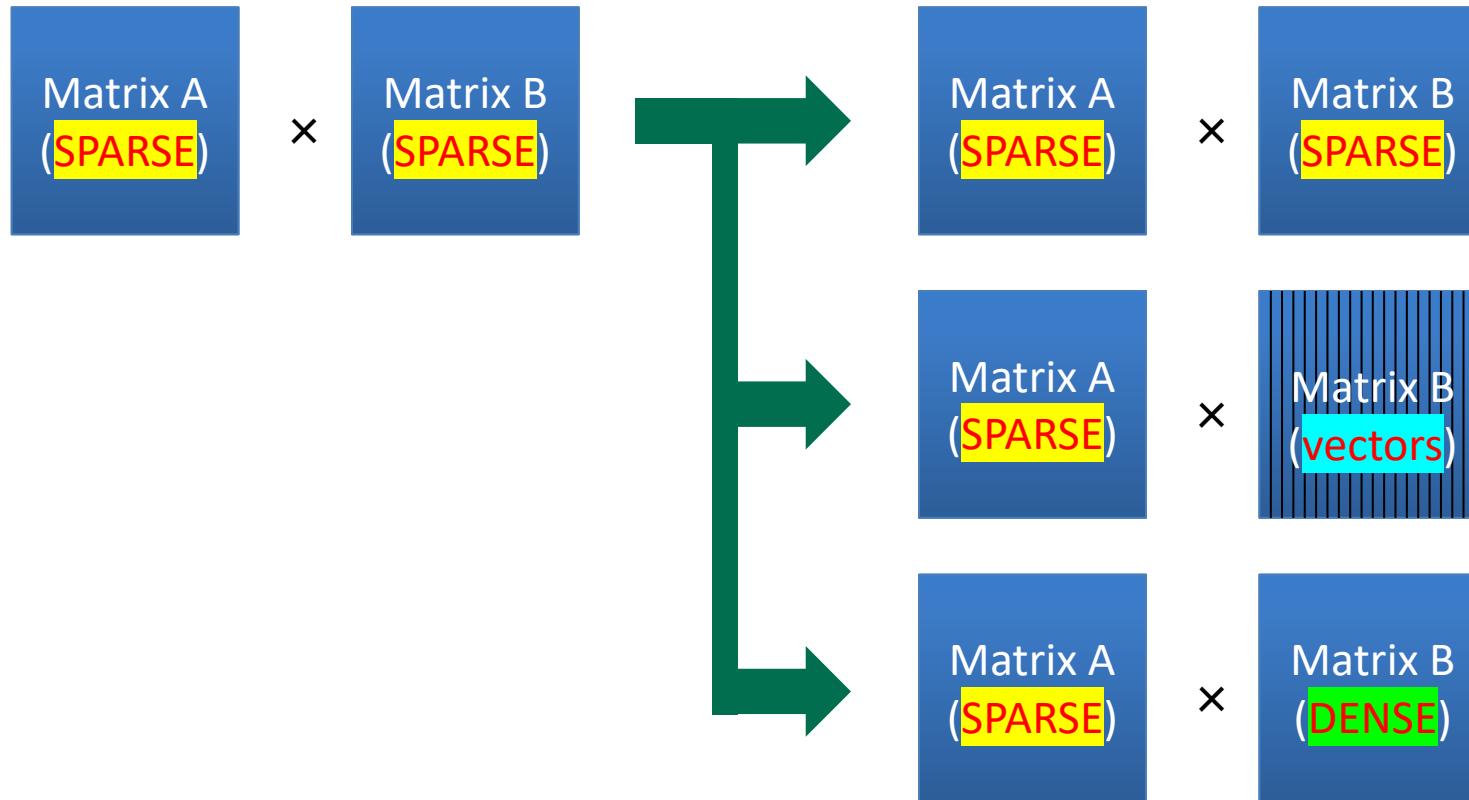
$$A = \begin{bmatrix} a_1^1 & b_2^2 & c_3^3 & 0 \\ 0 & 0 & 0 & d_4^4 \\ e_1^5 & 0 & 0 & f_4^6 \\ 0 & 0 & g_3^7 & 0 \end{bmatrix}$$



$$\begin{bmatrix} a_1 & b_2 & c_3 \\ d_4 & 0 & 0 \\ e_1 & f_4 & 0 \\ g_3 & 0 & 0 \end{bmatrix}$$

```
val    = [ a d e g b 0 f 0 c 0 0 0 ]  
colind = [ 1 4 1 3 2 -1 4 -1 3 -1 -1 -1 ]  
numcol = 3
```

# How to calculate Sparse Matrix A - Matrix B multiplication?



# Performance evaluation

- Evaluation environment
  - **Reedbush-H supercomputer**, 1 node
    - installed at the Information Technology Center, The University of Tokyo
    - each nodes has  
Intel Xeon E5-2695 v4 (**Broadwell-EP, 18cores**) \* 2 + **NVIDIA Tesla P100** \* 2
    - Intel C++ Compiler ver.18.1.163 + CUDA 9.2.148
- Test conditions
  - All matrix–matrix multiplications are performed with double precision.
  - On the CPU, 36 threads per node are used.
    - "export KMP AFFINITY=granularity=fine,compact,1,0" is specified in the job script for NUMA Affinity and the default values are chosen for NUMA policies.
  - On the GPU, cuSPARSE is used for matrix computations.
    - **SpMV** (Sparse Matrix \* Vector) / CRS format: **cusparseDcsrmv**
    - **SpMV** (Sparse Matrix \* Vector) / ELL format: **cusparseDhybm**
    - **SpMM** (Sparse Matrix \* Dense Matrix) / CRS format: **cusparseDcsrmm2**
    - **SpMxSpM** (Sparse Matrix \* Sparse Matrix) / CRS format: **cusparseDcsrmm**

# Target matrices

- A : an identity matrix with additional elements
  - additional elements are generated by a pseudo-random generator
- B :  $A^{-1}$  generated by the *dgetrf* and *dgetri* routines in LAPACK
- Matrix set1
  - The sparseness of A is 90%.
  - The sparseness of each sub-matrix A obtained from the Ozaki method is higher than 90%.
  - All sub-matrix A - sub-matrix B multiplications are calculated by SpMV.
  - Compared the execution time of MMM using the CRS and ELL, and on CPU and on GPU.  compares previous method and our proposed method
- Matrix set2
  - The sparseness of matrix B is one of the following seven types: 90%, 95%, 96%, 97%, 98%, 99%, and a zero matrix with only one element inserted (Max).
  - While matrix A is a dense matrix, stored in the CRS format.
  - Compared the execution time of MMM on GPU using SpMM and SpMxSpM.  compares two implementations in our proposed method
- For both matrix sets, matrix sizes of N (both height and width of original dense matrices) = 50, 100, 500, 1000, 5000, and 10000.

# Number of divisions in Error-free transformation

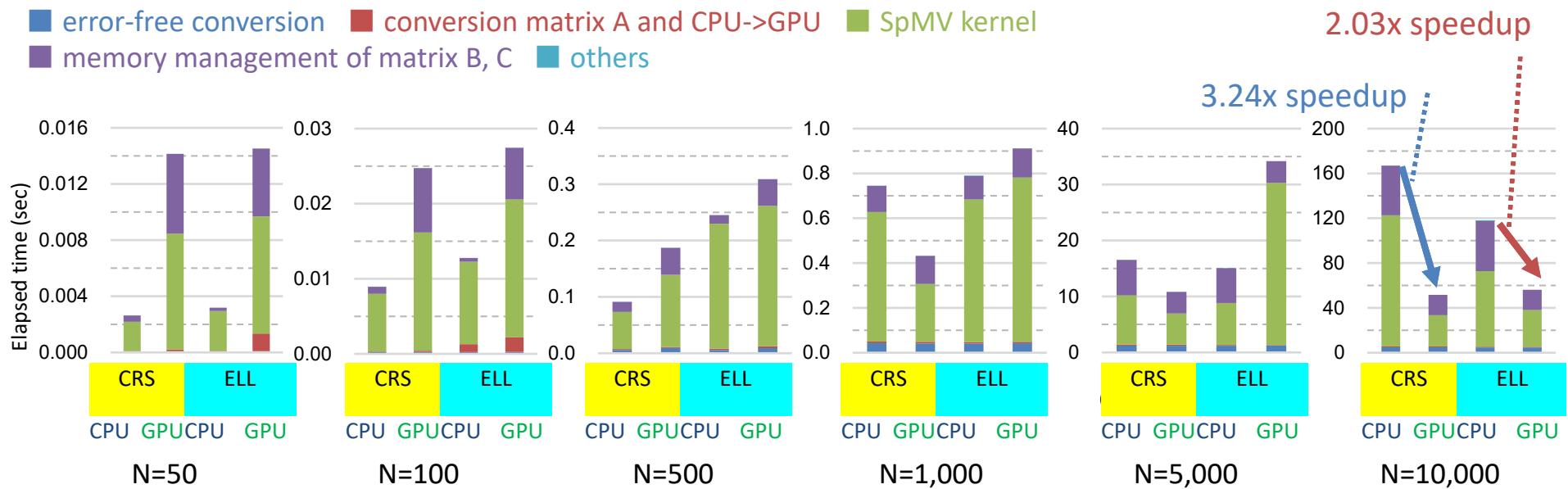
Number of matrix divisions  
by error-free transformation in Matrix set 1.

Size	50	100	500	1000	5000	10000
Ak	3	3	3	3	3	4
Bk	3	3	4	4	5	4

Number of matrix divisions  
by error-free transformation in Matrix set 2.

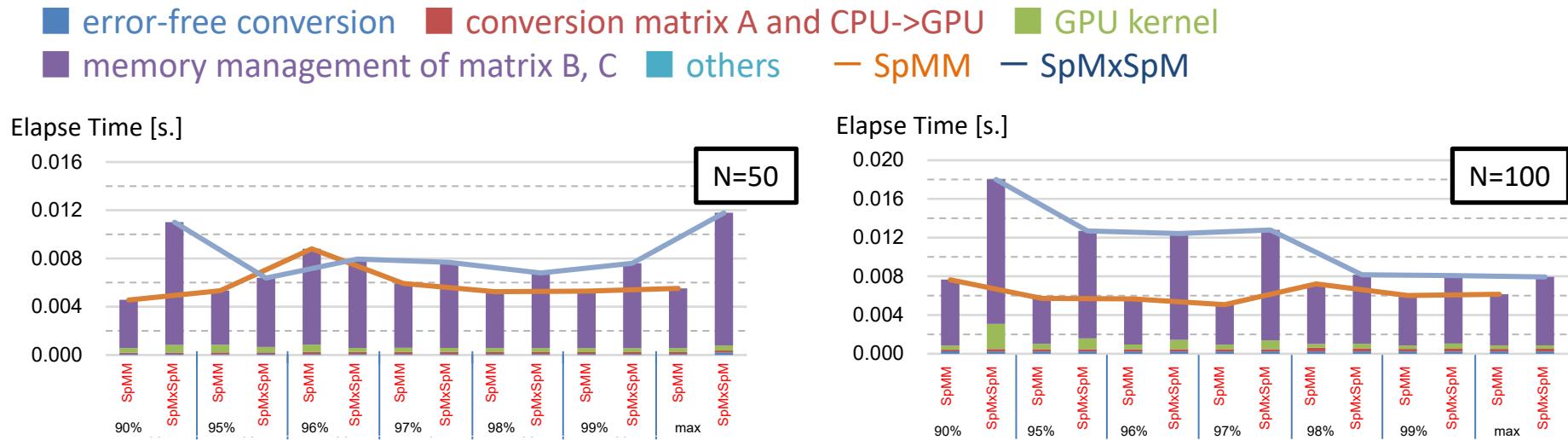
Sparseness \ Size	50	100	500	1000	5000	10000
90%	Ak	3	3	3	3	4
90%	Bk	3	3	4	4	5
95%	Ak	3	3	3	3	4
95%	Bk	3	3	4	4	5
96%	Ak	4	3	3	3	4
96%	Bk	2	3	4	4	5
97%	Ak	4	3	3	3	4
97%	Bk	2	3	4	4	5
98%	Ak	4	4	3	3	4
98%	Bk	2	2	4	4	5
99%	Ak	4	4	3	3	4
99%	Bk	2	2	4	4	5
Max	Ak	4	4	3	3	3
Max	Bk	2	2	3	3	4

# Result1: Execution time of accurate MMM using SpMV in CRS and ELL on CPU and GPU for Matrix set 1



- When N is small, CPU is faster than GPU mainly because data transfer between CPU and GPU is not negligible. Moreover, time of launching GPU kernel is included in GPU.
- CRS format: GPU is faster than CPU when N>=1,000
- ELL format: GPU is faster than CPU when N=10,000

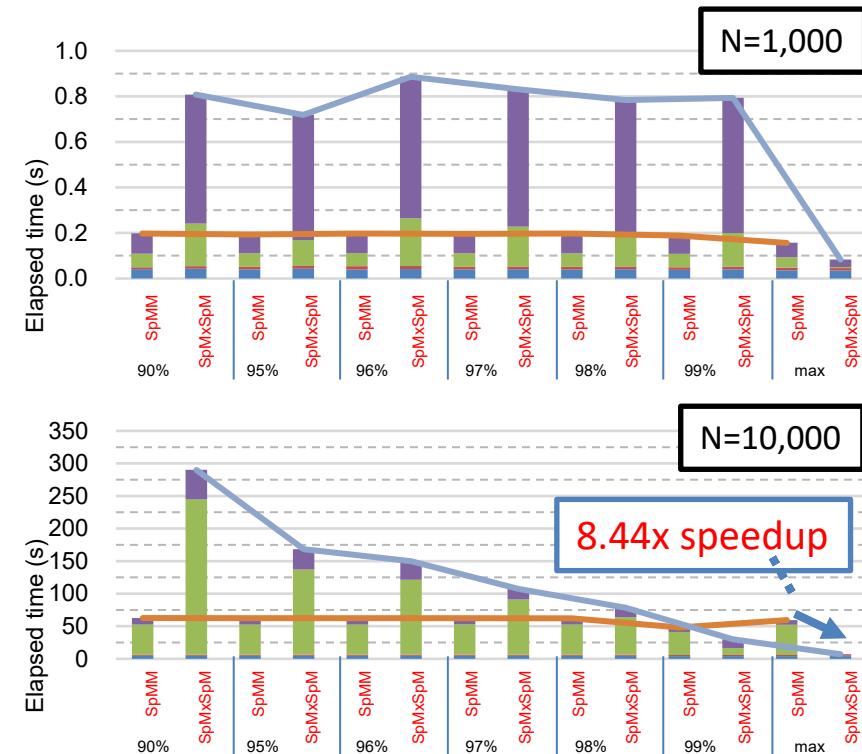
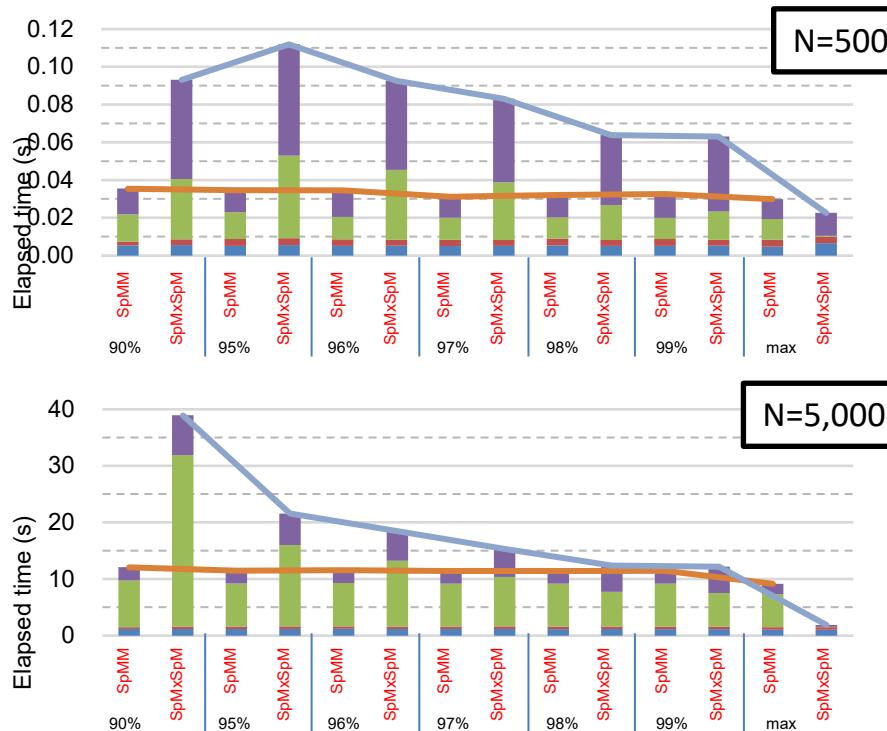
# Result2: Execution time of accurate MMM using SpMM and SpMxSpM in CRS on GPU for Matrix set 2 (1/2)



- **SpMM:** time and sparseness have few relationship because matrix B is dense matrix
- **SpMxSpM:** when N and sparseness increase, calculation time is relatively shortened.
- When N is small, execution times of SpMM is smaller than times of SpMxSpM.

# Result2: Execution time of accurate MMM using SpMM and SpMxSpM in CRS on GPU for Matrix set 2 (2/2)

█ error-free conversion    █ conversion matrix A and CPU->GPU    █ GPU kernel  
█ memory management of matrix B, C    █ others    — SpMM    — SpMxSpM



# Conclusion (1/2)

- We proposed to calculate sparse matrix calculations generated by the Ozaki method **on GPU**.
- Results
  - Compared with previous work (using CPU), performance improved by using GPU when the input matrix is large.
    - Accurate MMM using **SpMV in CRS on GPU** achieved **3.24x** higher performance than CPU.
    - Accurate MMM using **SpMV in ELL on GPU** achieved **2.03x** higher performance than CPU.
  - SpMxSpM is more effective than SpMM when Matrix B is highly sparse.
    - Accurate MMM **on GPU in CRS using SpMxSpM** achieved **8.44x** higher performance than using SpMM.

# Conclusion (2/2)

- Future work
  - The effectiveness depends on complicate conditions.  
(e.g. matrix format, matrix size, and the sparseness)
  - It's important to analyze the relationships between optimization parameters and performance.
- Acknowledgement
  - This work was supported by "Joint Usage/Research Center for Interdisciplinary Large-scale Information Infrastructures" in Japan (Project ID: jh200008-NAH) and JSPS KAKENHI Grant Number JP19H05662.