

# Implementation of highly optimized multiple precision BLAS: Strassen vs. Ozaki scheme

Tomonori Kouya

Shizuoka Institute of Science and Technology

2023-08-21(Mon)

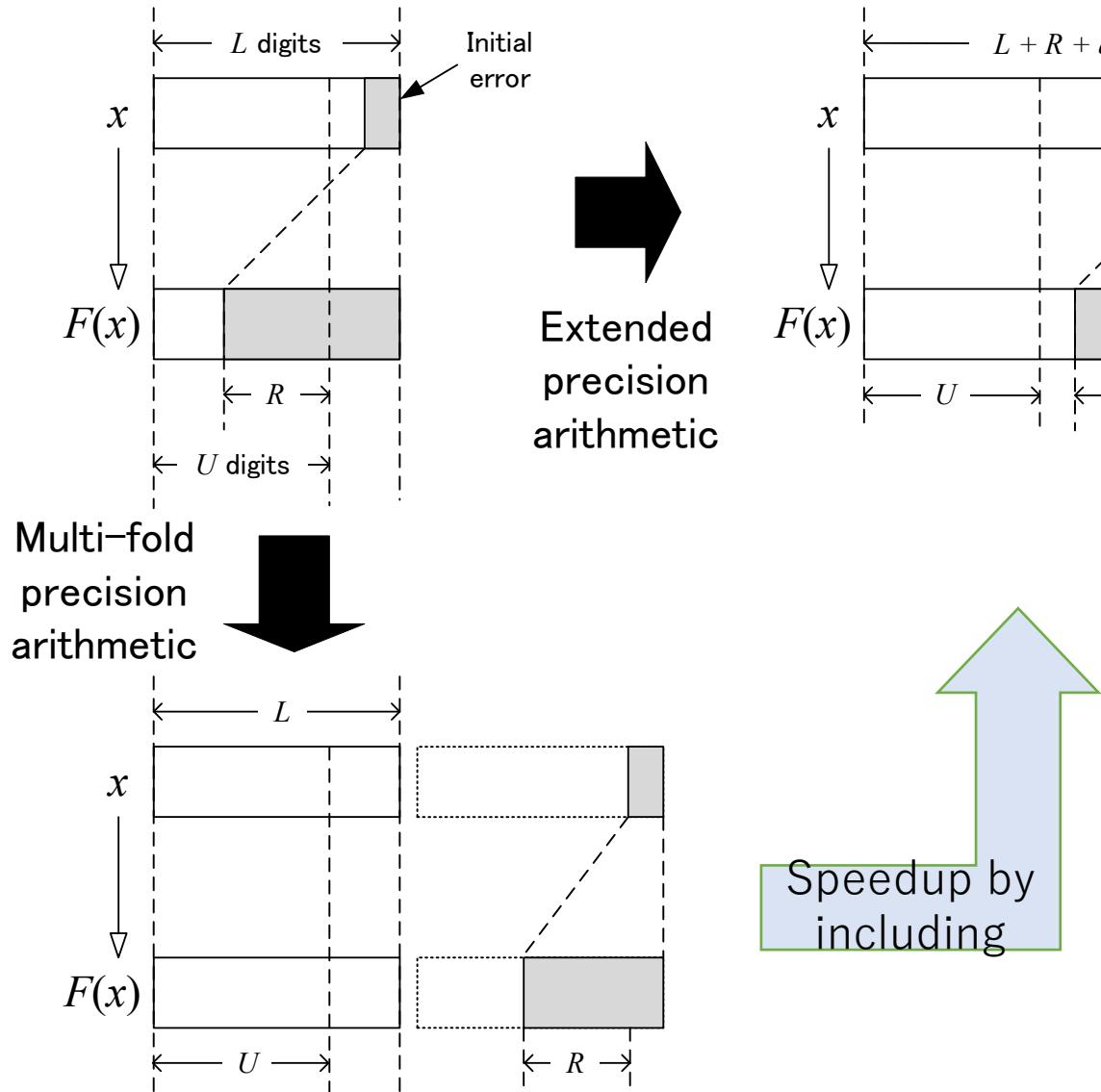
[01060] Exploring Arithmetic and Data Representation  
Beyond the Standard in HPC

@ICIAM2023

# Abstract

- Purpose
- BNCFmatmul's software layer
- Optimized multiple precision (MP) matrix multiplication
- Application to Optimization of LU decomposition
- Complex GEMM based on optimized MP GEMM
- Conclusion and future work

# Why do we need extended-precision numerical library?

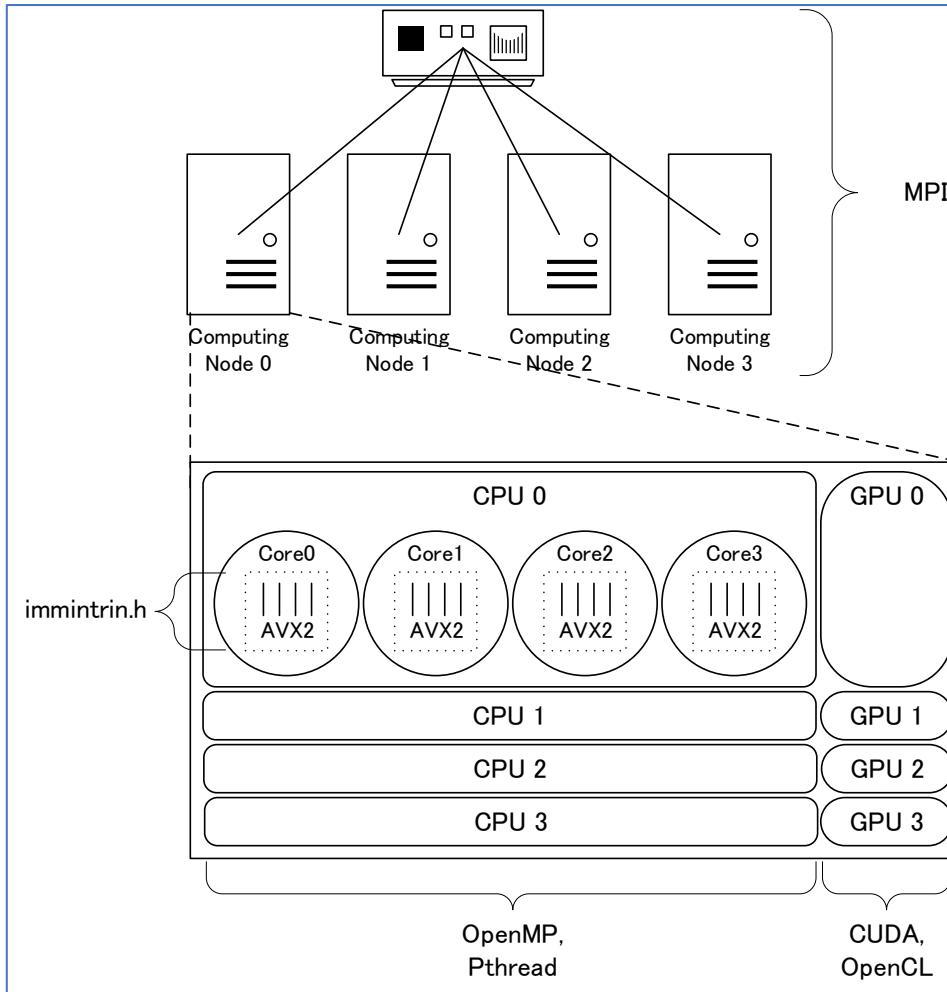


- **Extended-precision arithmetic :** Reduce the effect of initial errors due to rounding and errors introduced during calculation by increasing the number of digits in the mantissa part.
- **Multi-fold precision arithmetic :** Reduce the process of renormalization by combining optimized numerical modules based on binary16, 32 or 64 arithmetic

→ Both ways of arithmetic are necessary  
Ex: Extrapolation method for solving ill-conditioned ODEs (ARITH2019)

# Categorization of optimization techniques for HPC

## Architecture-oriented speedup techniques



## Optimization using computing algorithm

### Long-digit real multiplication

- Karatsuba method
- Toom-Cook method
- FFT

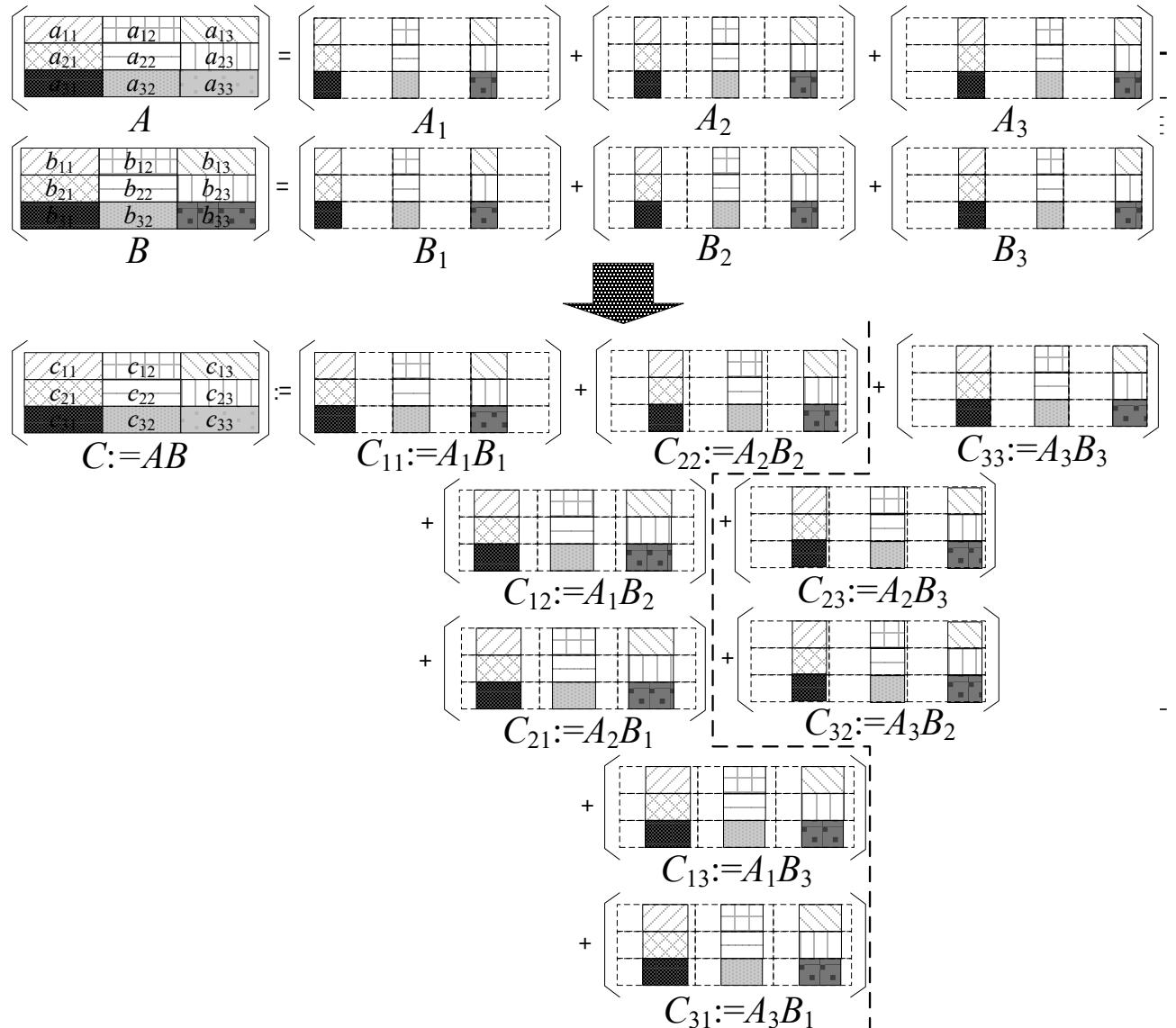
### Complex multiplication

- 3M method

### Matrix multiplication

- Blocking(Tiling) algorithm
- Strassen and Winograd
- Ozaki scheme (New!)

# Matrix algorithms: Strassen and Ozaki scheme



**Algorithm 2** Ozaki scheme for multiple-precision matrix multiplication

**Input:**  $A \in \mathbb{F}_{bL}^{m \times l}, B \in \mathbb{F}_{bL}^{l \times n}$

**Output:**  $C \in \mathbb{F}_{bL}^{m \times n}$

$A^{(S)} := A, B^{(S)} := B : A^{(S)} \in \mathbb{F}_{bS}^{m \times l}, B^{(S)} \in \mathbb{F}_{bS}^{l \times n}$

$\mathbf{e} := [1 \ 1 \ \dots \ 1]^T \in \mathbb{F}_{bS}^l$

$\alpha := 1$

**while**  $\alpha < D$  **do**

$\mu_A := [\max_{1 \leq p \leq l} |(A^{(S)})_{ip}|]_{i=1,2,\dots,m} \in \mathbb{F}_{bS}^m$

$\mu_B := [\max_{1 \leq q \leq l} |(B^{(S)})_{qj}|]_{j=1,2,\dots,n} \in \mathbb{F}_{bS}^n$

$\tau_A := [2^{\lceil \log_2((\mu_A)_i) \rceil + \lceil (S + \log_2(l))/2 \rceil}]_{i=1,2,\dots,m} \in \mathbb{F}_{bS}^m$

$\tau_B := [2^{\lceil \log_2((\mu_B)_j) \rceil + \lceil (S + \log_2(l))/2 \rceil}]_{j=1,2,\dots,n} \in \mathbb{F}_{bS}^n$

$S_A := \tau_A \mathbf{e}^T$

$S_B := \mathbf{e} \tau_B^T$

$A_\alpha := (A^{(S)} + S_A) - S_A : A_\alpha \in \mathbb{F}_{bS}^{m \times l}$

$B_\alpha := (B^{(S)} + S_B) - S_B : B_\alpha \in \mathbb{F}_{bS}^{l \times n}$

$A := A - A_\alpha, B := B - B_\alpha : L\text{-bit FP arithmetic}$

$A^{(S)} := A, B^{(S)} := B$

$\alpha := \alpha + 1$

**end while**

$A_D := A^{(S)}, B_D := B^{(S)}$

$C := O$

**for**  $\alpha = 1, 2, \dots, D$  **do**

**for**  $\beta = 1, 2, \dots, D - \alpha + 1$  **do**

$C_{\alpha\beta} := A_\alpha B_\beta$

**end for**

$C := C + \sum_{\beta=1}^{D-\alpha+1} C_{\alpha\beta} : L\text{-bit FP arithmetic}$

**end for**

# Purpose: Optimization of multiple-precision matrix multiplication and its application

xGEMM	CPU				GPU	
Opt.Method	None	AVX2	OpenMP	Ozaki Scheme	CUDA	Ozaki Scheme
DS	?	?	?	?	Mukunoki	Mukunoki
TS	?	?	?	Utsugiri	Utsugiri	Utsugiri
QS	?	?	?	?	?	?
IEEE754 Binary128	MPBLAS	?	MPBLAS	Mukunoki	Mukunoki	Mukunoki
DD	MPBLAS, BNCmatmul	Lis, MuPAT BNCmatmul	Lis, MuPAT, MPBLAS, BNCmatmul	Utsugiri	Mukunoki	Mukunoki
TD	BNCmatmul	BNCmatmul	BNCmatmul	Utsugiri	Utsugiri	Utsugiri
QD	MuPAT, MPBLAS, BNCmatmul	BNCmatmul	MuPAT, MPBLAS, BNCmatmul	Utsugiri	?	?
MPFR	MPBLAS, BNCmatmul	?	MPBLAS, BNCmatmul	Utsugiri	CUMP	?

MPLAPACK/MPBLAS <https://github.com/mahonakata/mplapack>  
 MuPAT Yagi, H et.al(2020)  
 Lis <https://www.ssisc.org/lis/>  
 CUMP <https://github.com/skystar0227/CUMP>  
 BNCmatmul <https://na-inet.jp/na/bnc/>  
 Mukunoki Mukunoki, D et.al.(2021)  
 Utugiri Utsugiri(2021-2023)

- Completed implementation of optimized DD, TD, QD and MPFR matrix multiplication (xGEMM)
- Found the effectiveness of Ozaki scheme
- Confirmed the effectiveness of Ozaki scheme to LU decomposition

# BNCmatmul User's Guide

Tomonori Kouya

<https://github.com/tkouya/bncmatmul>

Version 0.21: May 31, 2023

## 1.7 History of Version and Todo list

### ■History of Version

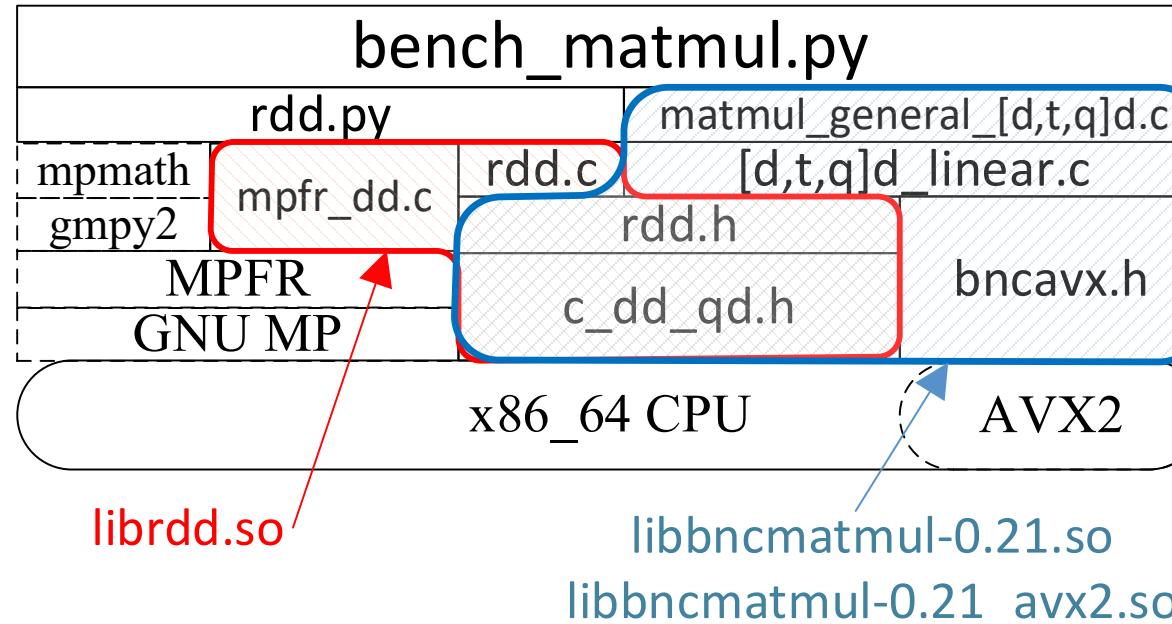
Version 2.0: 2016-06-08 Firstly opening sources at <https://na-inet.jp/na/bnc/bncmatmul-0.2.tar.bz2>, which provides only arbitrary matrix multiplication based on MPFR.

Version 2.1: 2023-05-31 Secondly opening sources including DD, TD, QD and MPFR precision real BLAS functions at <https://github.com/tkouya/bncmatmul/blob/main/bncmatmul-0.21.tar.bz2>.

### ■Todo list

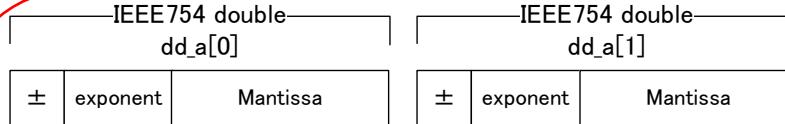
1. Appending complex BLAS functions with DD, TD, QD, and MPFR(MPC)
2. Appending complex LU decomposition and related functions
3. Appending sparse matrix-vector multiplication
4. Showing more sample sources in this manual using BNCmatmul and MPLAPACK/BLAS

# BNCmatmul's software layer



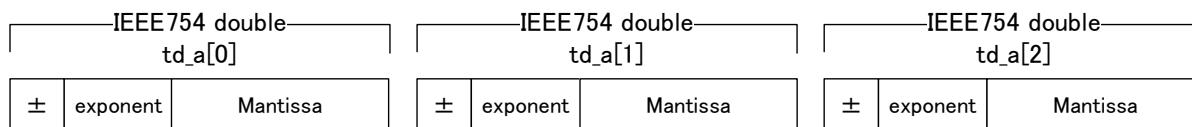
- All FP computation is supported on CPU.
- Limited functionaries but faster than MPLAPACK/MPBLAS
- Multi-component-way fxed-precision arithmetic is constructed with original ANSI C codes : Double, DD, TD, and QD
- Multi-digit-way arbitrary-precision arithmetic is based on MPFR/GMP
- Serial multiple-precision computation is runnable on Python ecosystem.

# Our targeting FP precision: DD, TD, QD and MPFR

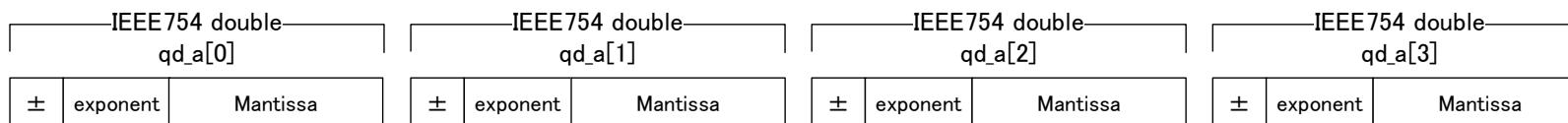


Accelerated with AVX2

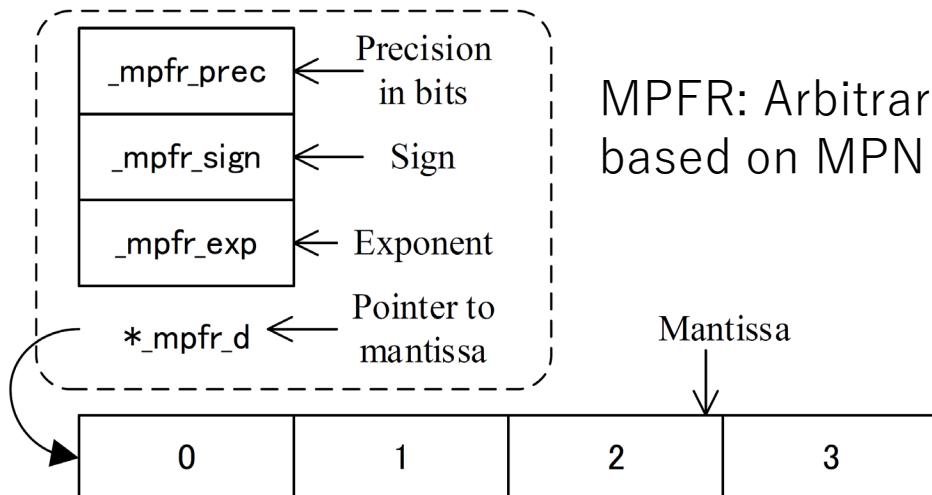
Double-double(DD)



Triple-double(TD)



Quadruple-double(QD)



MPFR: Arbitrary precision FP arithmetic  
based on MPN kernel of GNU MP

MPBLAS in MPLAPACK: MPREAL C++ class  
library calls MPFR functions  
BNCmatmul: Directly calls MPFR  
functions → about 10% better performed

# List of categorized functions in BNCFmatmul

Technique		None	SIMD		Parallelization			
		256bits AVX2		512bits AVX-512		OpenMP		MPI
Real	BLAS 1	(F) D DD TD QD MPFR	D DD TD QD	D DD TD QD	D DD TD QD	D DD TD QD	D DD TD QD MPFR	MPFR
	BLAS 2	(F) D DD TD QD MPFR	D DD TD QD	D DD TD QD	D DD TD QD	D DD TD QD	D DD TD QD MPFR	MPFR
	BLAS 3	Strassen (D) DD TD QD MPFR	D DD TD QD	D DD TD QD	D DD TD QD	D DD TD QD	D DD TD QD MPFR	
	Ozaki scheme	(D) DD TD QD MPFR						
	LU	(F) D DD TD QD MPFR	D DD TD QD	D DD TD QD	D DD TD QD	D DD TD QD	D DD TD QD MPFR	
	SpMV		MPFR					
Complex	BLAS 1	(D) DD TD QD MPFR	D DD TD QD	D DD TD QD	D DD TD QD	D DD TD QD		
	BLAS 2	(D) DD TD QD MPFR	D DD TD QD	D DD TD QD	D DD TD QD	D DD TD QD		
	BLAS 3	Strassen (D) DD TD QD MPFR	D DD TD QD	D DD TD QD	D DD TD QD	D DD TD QD		
	Ozaki scheme	(D) DD TD QD MPFR						
	LU		MPFR					
	SpMV							


**BNCmatmul Version 2.1**

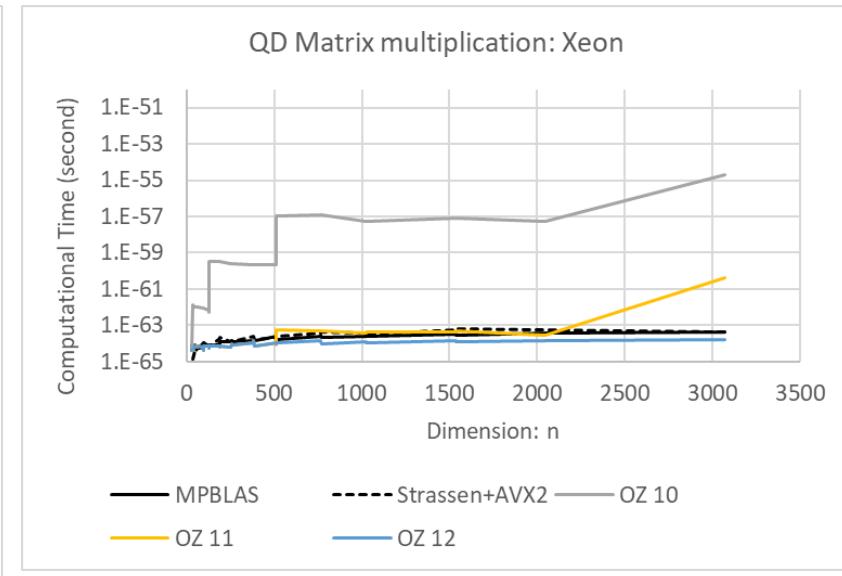
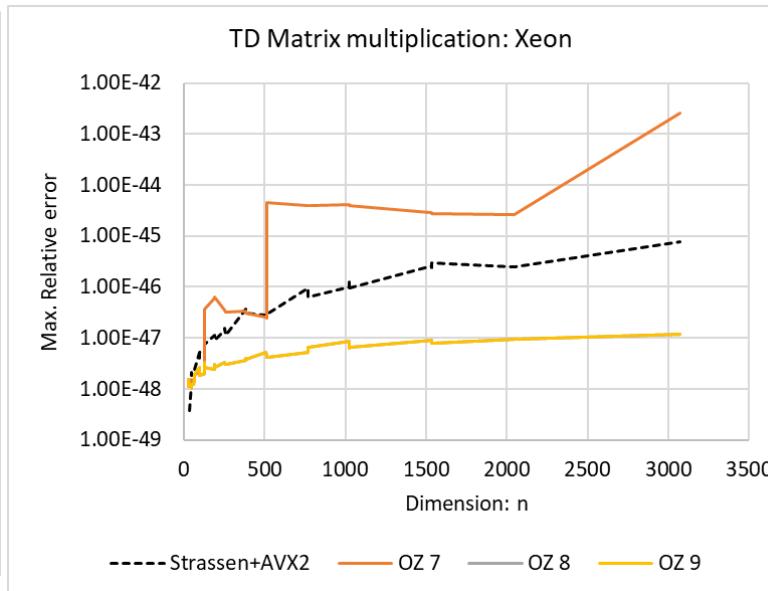
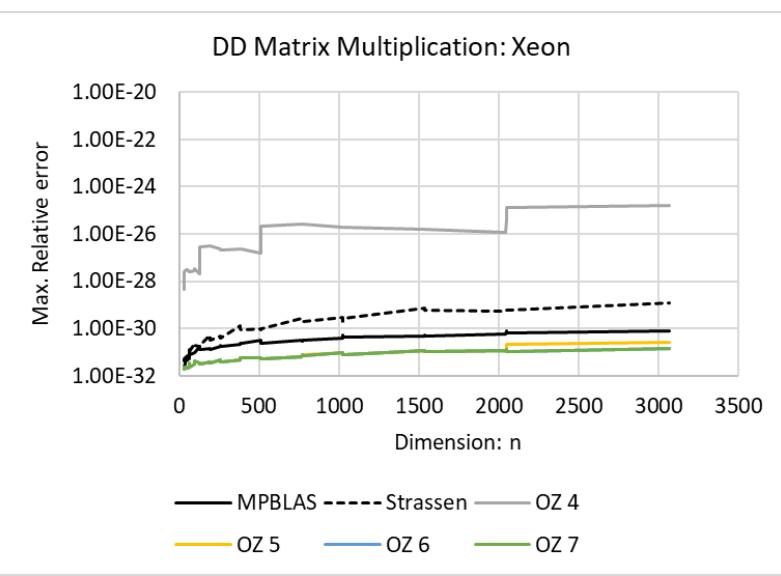
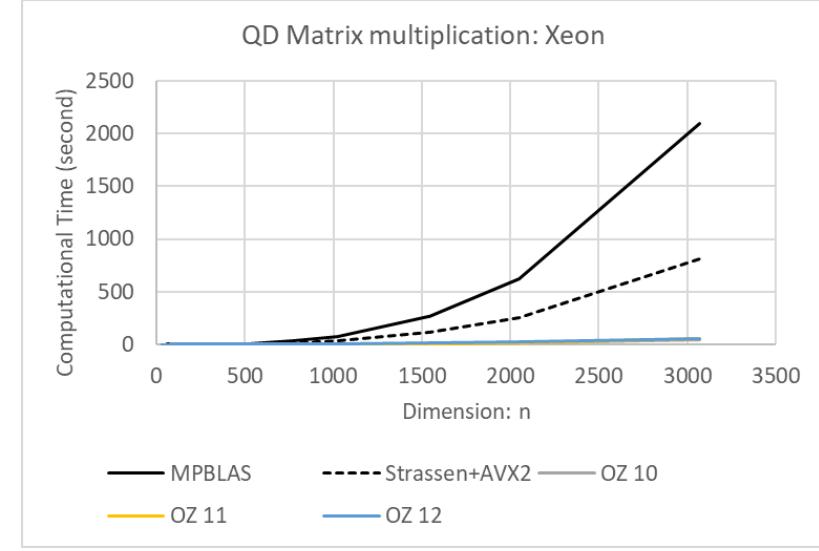
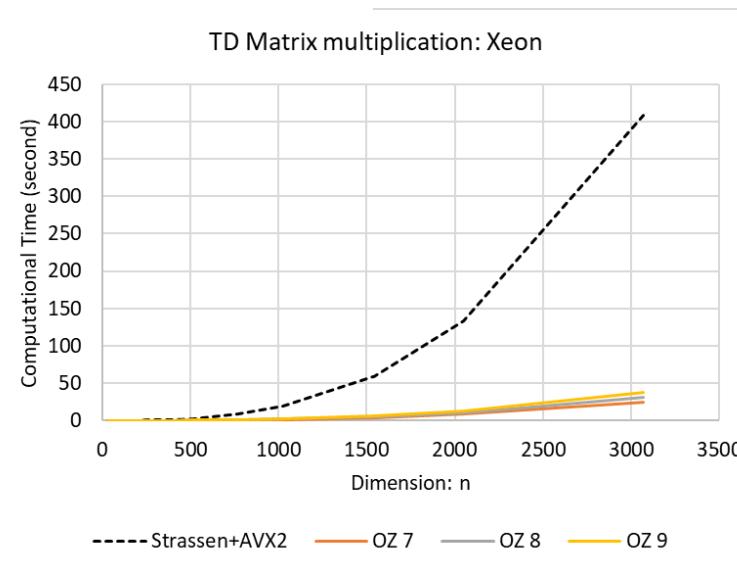
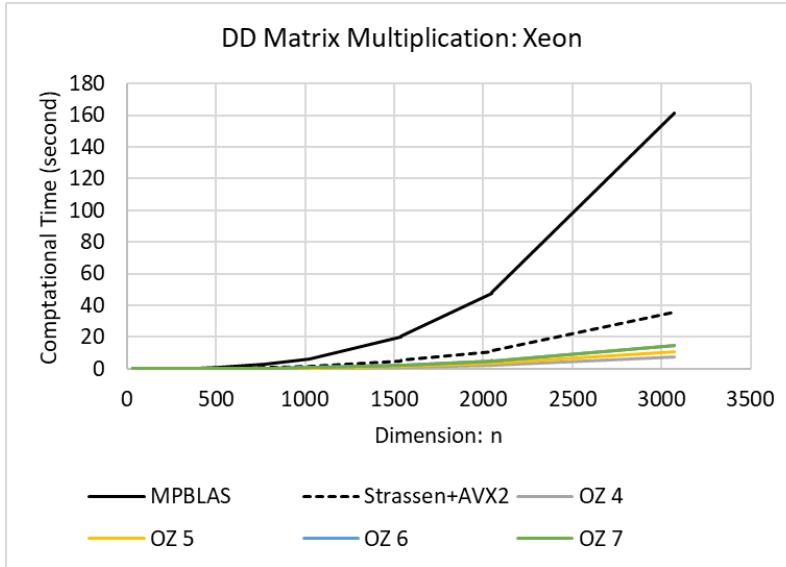

**BNCmatmul Version 2.2**

F IEEE754 binary32  
Single precision(24bits)     
 D IEEE754 binary64  
Double precision(53bits)     
 DD Double-double(106bits)     
 TD Triple-double(159bits)     
 QD Quadruple-double(212bits)     
 MPFR MPFR/GMP  
Arbitrary precision

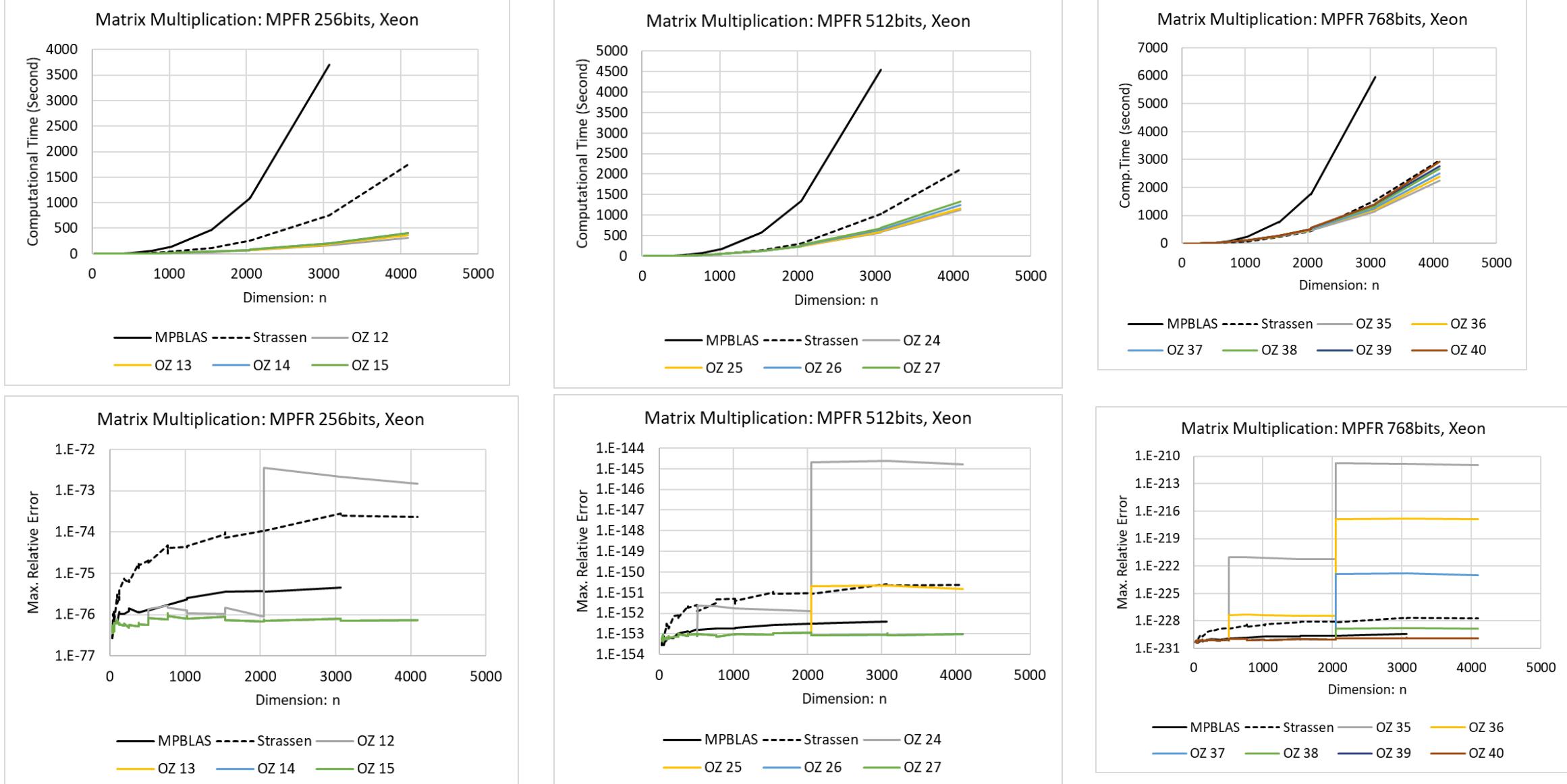
# Computing environment for benchmark tests

- [Xeon] Intel Xeon W-2295 3.0GHz 18 cores, Ubuntu 20.04.3 LTS, Intel Compiler version 2021.5.0, MPLAPACK 1.0.1, MPFR 4.1.0
- [C++(icpc), C(icc)] -O3 -fp-model precise -qopenmp -axCORE-AVX2 -march=skylake -mtune=skylake -mcpu=skylake

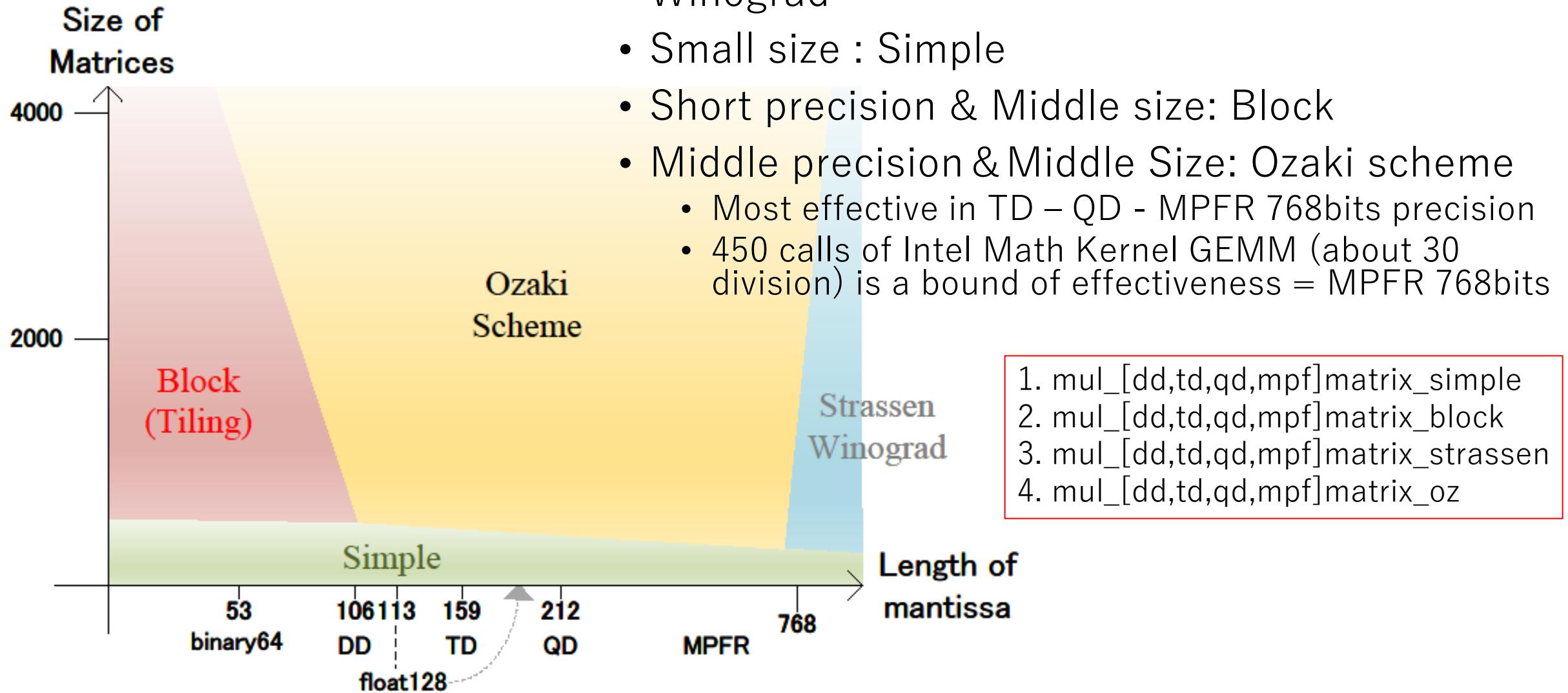
# DD, TD, QD on Xeon



# MPFR 256, 512, 768bits on Xeon

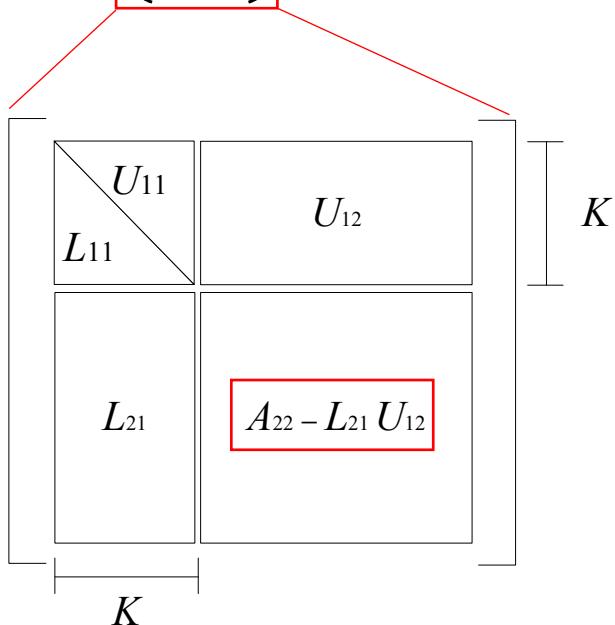


# A quick summary of optimization techniques for multiple precision matrix multiplication



# Test linear system of equations

$$(LU)\mathbf{x} = \mathbf{b}$$



As an application of optimized matrix multiplication, we implemented LU decomposition in the direct method for various benchmark tests, including the Top500, and measured its utility on Xeon. The corresponding  $n$ -dimensional linear system of equation is:

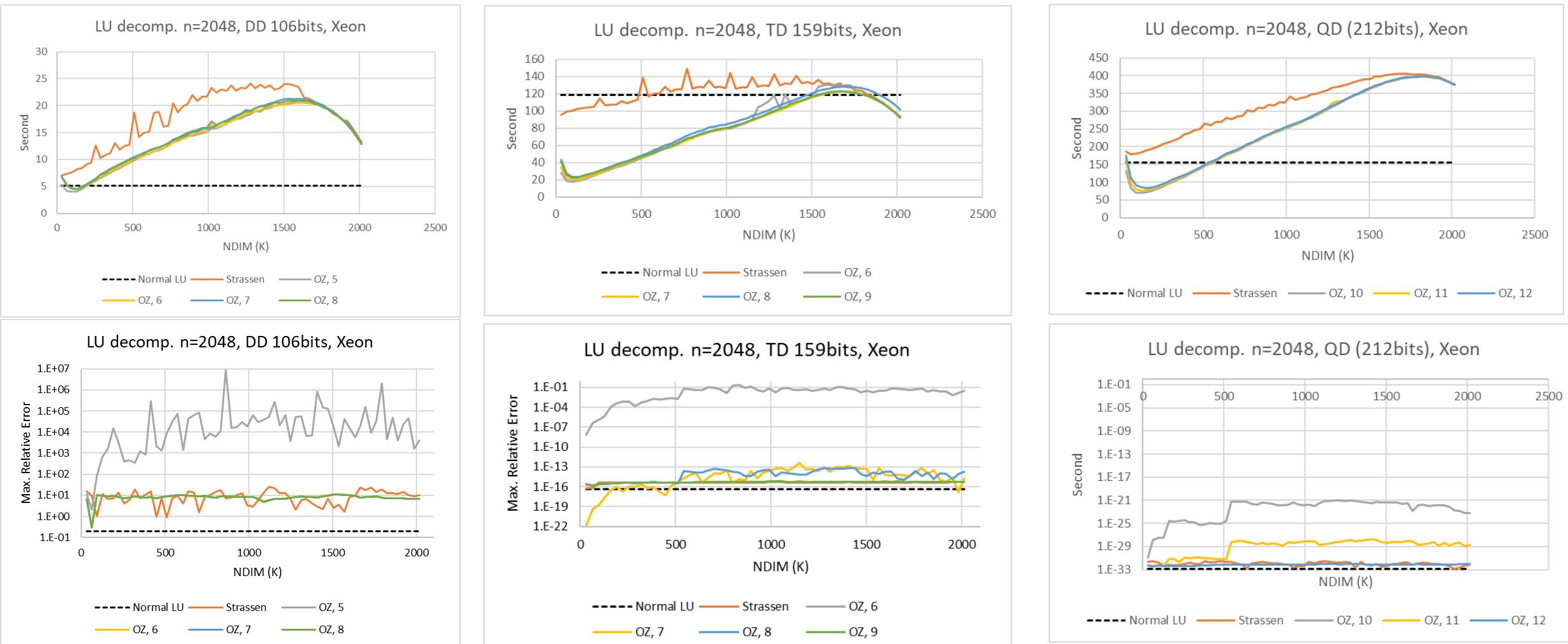
$$A\mathbf{x} = \mathbf{b}, \quad (2)$$

where  $A \in \mathbb{R}^{n \times n}$ ,  $\mathbf{x} \in \mathbb{R}^n$ , and  $\mathbf{b} \in \mathbb{R}^n$  become ill-conditional as follows:

**A:** The diagonal matrix is  $D = \text{diag}[d_1 \cdots d_n]$ , where  $d_i := 10^{-26(i-1)/n}$ . The random matrix is  $R \in \mathbb{R}^{n \times n}$ ; then,  $A$  is calculated as  $A := RDR^{-1}$  using the mpmath Python library. Therefore, the condition number of  $A$  is  $\kappa_2(A) = \|A\|_2\|A^{-1}\|_2 = 10^{26(n-1)/n}$ , which requires super-DD precision arithmetic to gain accuracy.

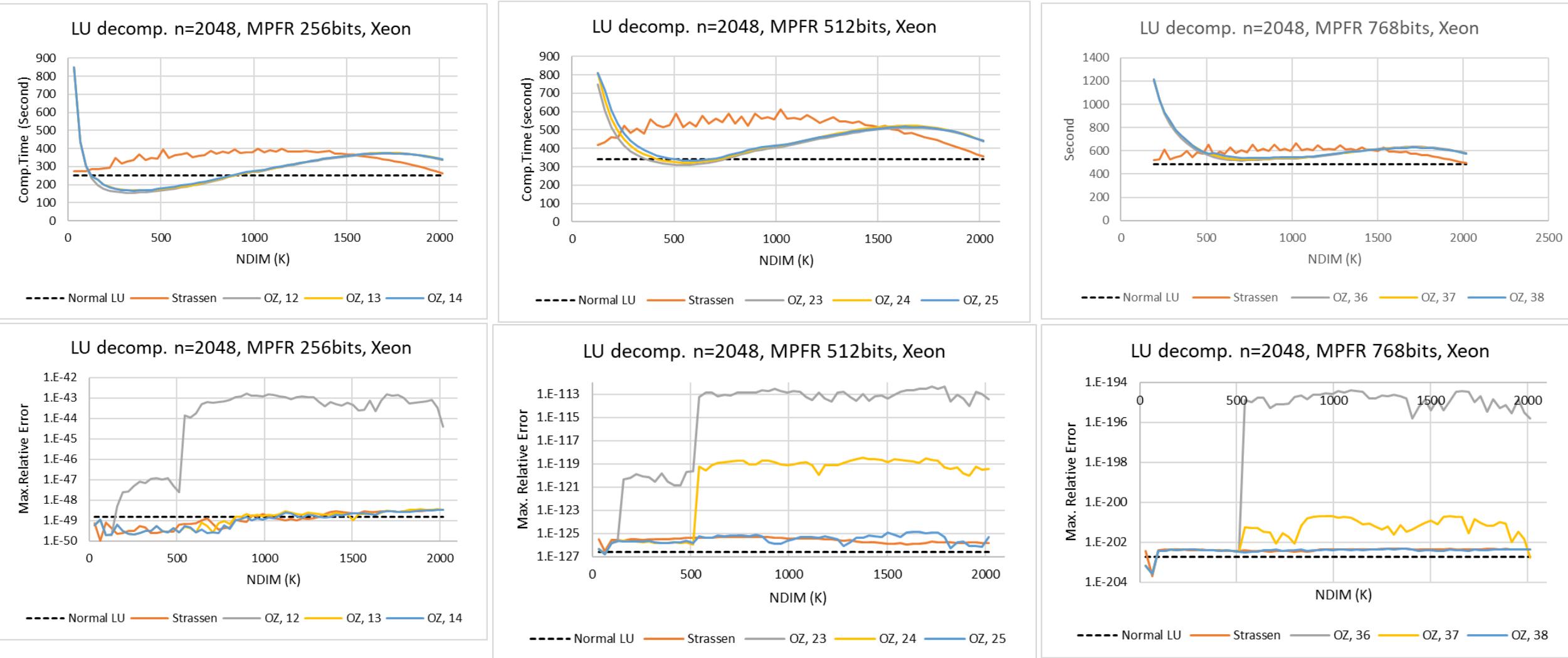
**x, b:** After setting  $\mathbf{x} = [0 \ 1 \ \cdots \ n-1]^T$ ,  $\mathbf{b} := A\mathbf{x}$  is calculated using mpmath.

# DD, TD, QD LU on Xeon



- Reduced the effectiveness due to rectangle matrix multiplication used in LU decomposition
- Trend of optimization is similar with square matrix multiplication

# MPFR 256, 512, 768 bits on Xeon



- LU w/ Ozaki scheme is not faster than row-wised LU decomposition over 768bit-precision MPFR arithmetic

# Conclusion: Ozaki scheme is effective for middle-precision LU decomposition

Prec.	Method	$K$	Second	Rel.Err.
DD 106bits	Rgetrf	N/A	15.8	2.4E – 01
	Normal LU	1	5.1	1.9E – 01
	Strassen+AVX2	32	7.4	1.8E + 01
	OZ 6	128	4.5	9.4E + 01
TD 159bits	Normal LU	1	118.6	3.9E – 17
	Strassen+AVX2	32	95.6	5.9E – 17
	OZ 7	96	19.8	1.7E – 17
	OZ 12	160	83.2	5.8E – 33
QD 212bits	Rgetrf	N/A	207.4	3.8E – 34
	Normal LU	1	155.3	1.4E – 33
	Strassen+AVX2	64	180.8	1.5E – 32
	OZ 12	160	83.2	5.8E – 33

Prec.	Method	$K$	Second	Rel.Err.
MPFR 256 bits	Rgetrf	N/A	398.6	2.1E – 50
	Normal LU	1	250.0	1.5E – 49
	Strassen	96	273.7	8.3E – 50
	OZ 13	320	167.6	3.2E – 50
MPFR 512 bits	Rgetrf	N/A	492.3	2.5E – 127
	Normal LU	1	341.1	2.6E – 127
	Strassen	32	390.4	3.1E – 126
	OZ 25	576	333.4	4.1E – 126
MPFR 768 bits	Rgetrf	N/A	627.8	5.1E – 205
	Normal LU	1	481.4	1.9E – 203
	Strassen	32	491.1	3.6E – 203
	OZ 38	736	536.6	3.7E – 203

# Complex GEMM: 4M and 3M methods

- Multiplication of Complex matrices

$$A \in \mathbb{C}^{m \times l}, B \in \mathbb{C}^{l \times n} \Rightarrow AB \in \mathbb{C}^{m \times n}$$

- 4M method : 4 real multiplications, 2 real additions

$$AB = \{\text{Re}(A)\text{Re}(B) - \text{Im}(A)\text{Im}(B)\} + \{\text{Re}(A)\text{Im}(B) - \text{Im}(A)\text{Re}(B)\} \cdot i \in \mathbb{C}^{m \times n}$$

- 3M method : 3 real multiplications, 5 real additions

$$T_1 = \text{Re}(A)\text{Re}(B) \in \mathbb{R}^{m \times n}, T_2 = \text{Im}(a)\text{Im}(b) \in \mathbb{R}^{m \times n}$$

$$AB = (T_1 - T_2) + \{(\text{Re}(A) + \text{Im}(A))(\text{Re}(B) + \text{Im}(B)) - T_1 - T_2\} \cdot i \in \mathbb{C}^{m \times n}$$

## Implementation of 3M method

1. MPC library <https://www.multipleprecision.org/mpc/>
2. Bini: MPsolver GNU MP's mpf-based 3M method
3. BLIS: 3M and 4M CGEMM based on DGEMM

# Benchmark test of CGEMM(1/3)

$$A, B \in \mathbb{C}^{n \times n} \quad \longrightarrow \quad C := AB$$

Real and imaginary parts of each element

$$(ru - 0.5) \times \exp(rn)$$

[CPU and OS] Intel Xeon W-2295 3.0GHz 18 cores, Ubuntu 20.04.3 LTS

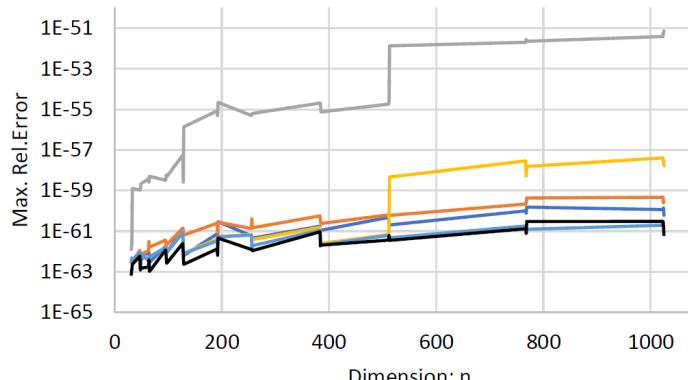
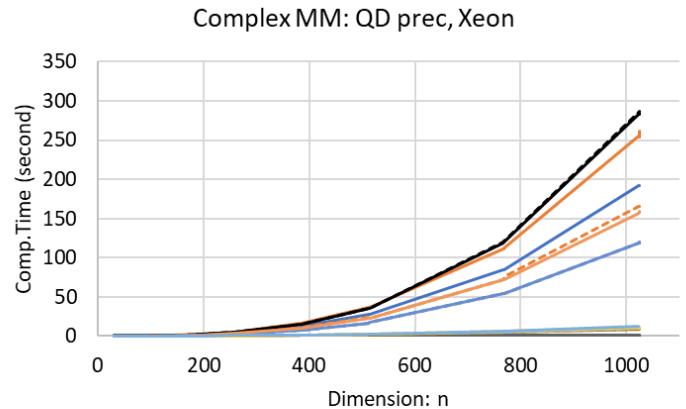
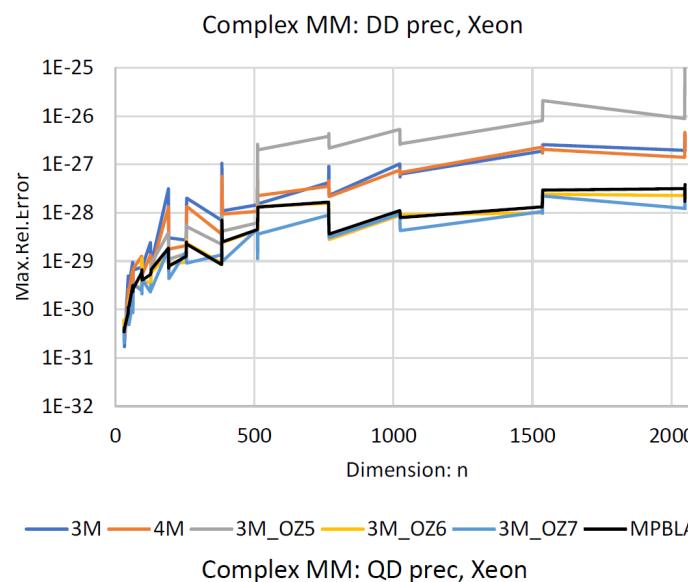
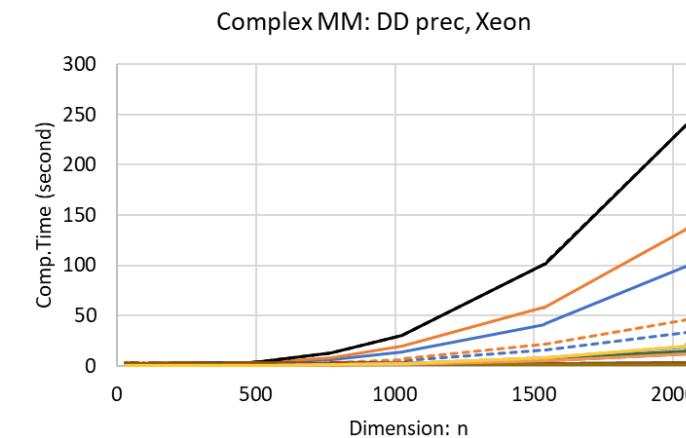
[C/C++] Intel Compiler version 2021.5.0

[Compiler Option] -O3 -std=c++11 -fp-model precise

[with AVX2] -axCORE-AVX2 -march=skylake -mtune=skylake mcpu=skylake

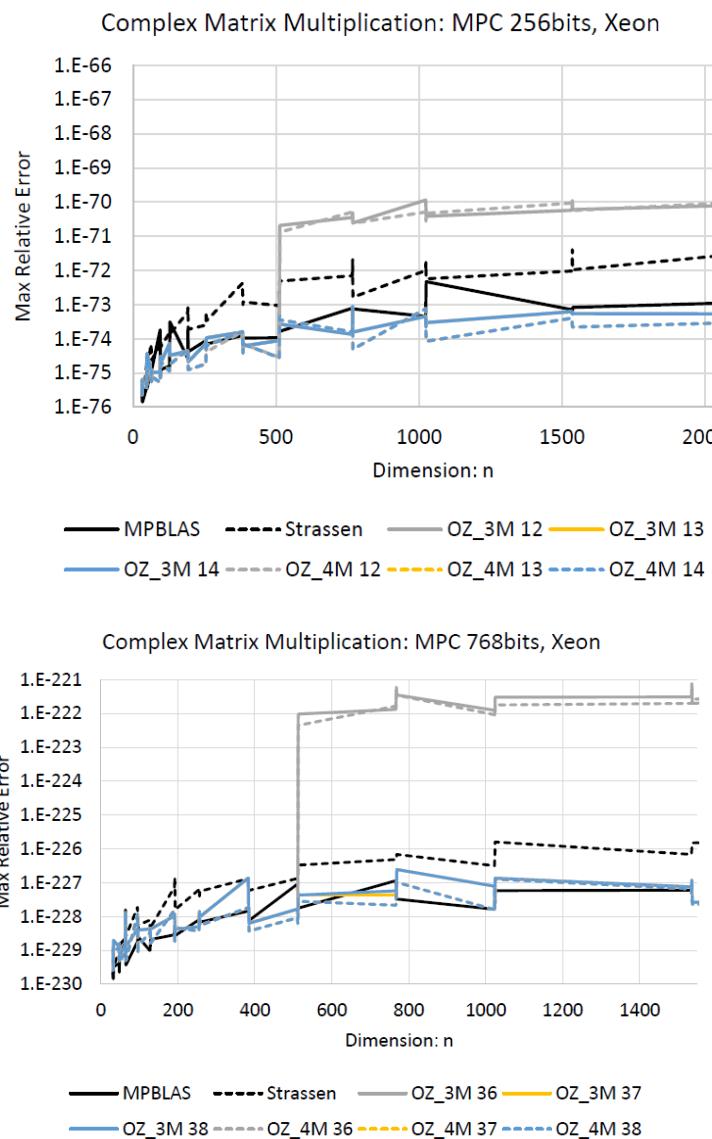
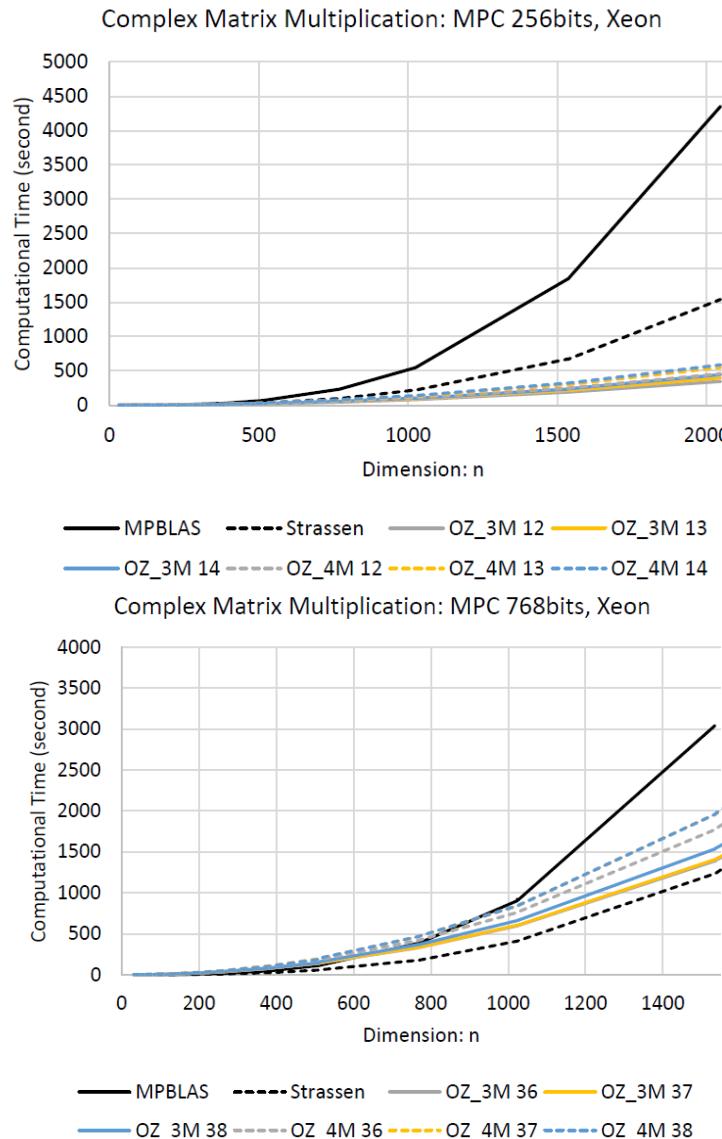
[MPLAPACK] 2.0.1, GNU MP 6.2.1, MPFR 4.1.0, MPC 1.2.1

# Benchmark test of CGEMM(2/3): DD and QD prec.



- Ozaki scheme(DD, QD + IMKL DGEMM)
  - DD prec. → 6 divisions
  - QD prec. → 12 divisions
- There is no difference of accuracy between 4M and 3M methods

# Benchmark test of CGEMM(3/3): 256bits and 768bits



- MPBLAS and Strassen are built on MPC
- 3M Ozaki scheme is made of IMKL DGEMM
- 256bits → 13 divisions → Ozaki scheme is the fastest
- 768bits → 37 divisions → Strassen is the fastest
- The same result as on real GEMM benchmark

# Conclusion and future work

## Conclusion

- Ozaki scheme is usually more efficient than Strassen in middle-precision (From DD to QD and 512bit-length MPFR).
- For complex GEMM, 3M method based on optimized real GEMM is better than 4M method

## Future work

- Benchmark test with real and complex GEMMs requiring more number of divisions of matrices
- Implementation of complex LU decomposition
- Implementation of optimized sparse matrix-vector multiplication (SpMV)
- Application to solve ill-conditioned algebraic equations