Accurate BLAS Implementations: OzBLAS and BLAS-DOT2

LSPANC2020January, January 30, 2020, R-CCS
Daichi Mukunoki (R-CCS)

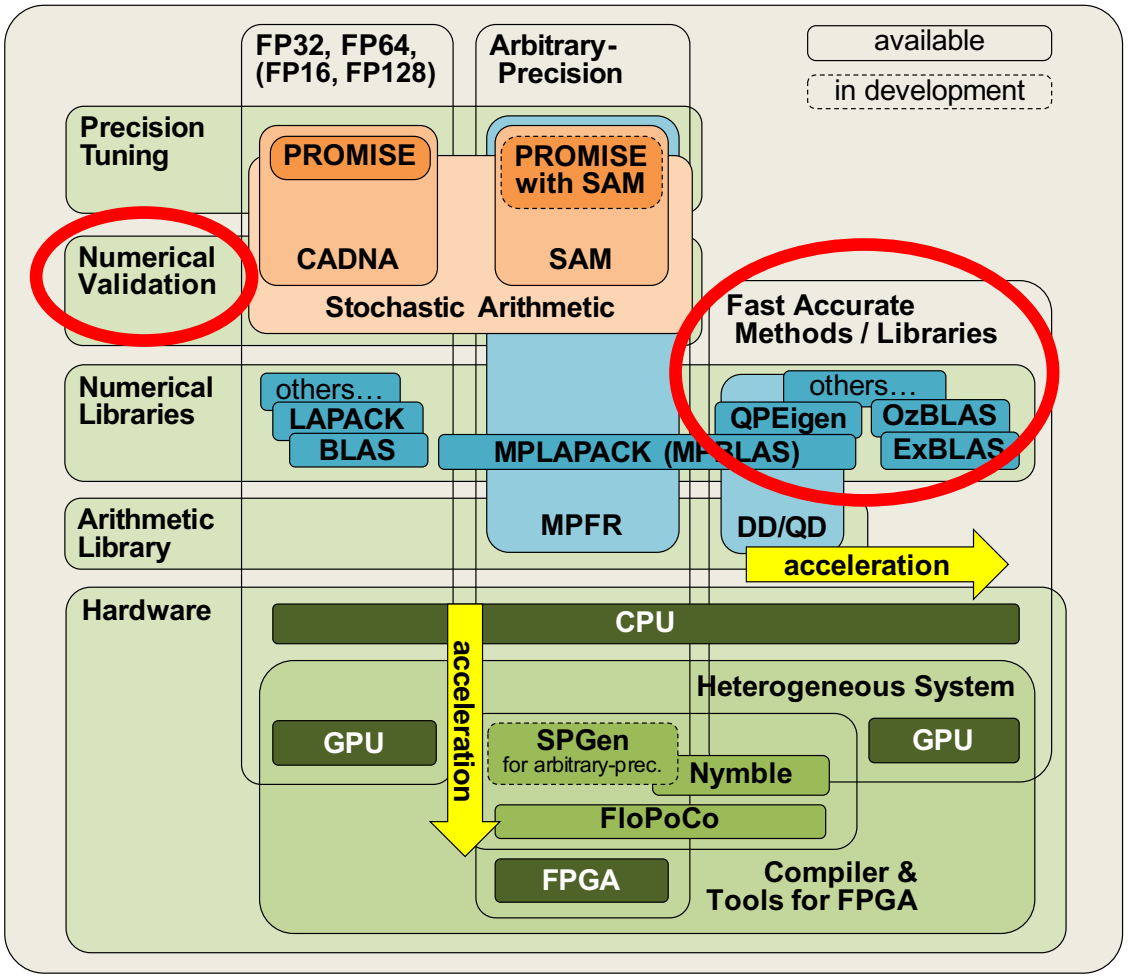
Research Collaborators:
Takeshi Ogita (Tokyo Woman's Christian University)
Katsuhisa Ozaki (Shibaura Institute of Technology)

Acknowledgement:

This research was partially supported by MEXT as "Exploratory Issue on Post-K computer" (Development of verified numerical computations and super high performance computing environment for extreme researches) and the Japan Society for the Promotion of Science (JSPS) KAKENHI Grant Number 19K20286.

Fast ~~Accurate~~ Multi-Precision Numerical Library

- **Needs in Minimal-Precision Computing**
  1. Fast computation of arbitrary-precision (incl. accurate) result on CPU/GPU
  2. Accurate numerical validation (or verification)



Accurate BLAS Implementations

Examples of accurate BLAS implementations

	Platform	Data Precision	Computation Precision
BLAS (netlib)	CPU, GPU	FP32/64	FP32/64
XBLAS (netlib)	CPU	FP32/64	FP32/64/2xFP64
BLAS-DOT2 (TWCU)	GPU	FP64	2xFP64
ReproBLAS (Berkeley)	CPU	FP64	Tunable
OzBLAS (TWCU/RIKEN)	CPU, GPU	FP64	Tunable (correct-rounding)
ExBLAS (Sorbonne U.)	CPU, GPU	FP64	Correct rounding
QPBLAS (JAEA/RIKEN)	CPU, GPU	DD	DD
MBLAS / MPLAPACK (M. Nakata)	CPU	FP64/DD/ QD,MPFR	FP64/DD/QD/MPFR
…			

Red: our work

Accurate BLAS Implementations

Examples of accurate BLAS implementations

	Platform	Data Precision	Computation Precision
BLAS (netlib)	CPU, GPU	FP32/64	FP32/64
XBLAS (netlib)	CPU	FP32/64	FP32/64/2xFP64
BLAS-DOT2 (TWCU)	GPU	FP64	2xFP64
ReproBLAS (Berkeley)	CPU	FP64	Tunable
OzBLAS (TWCU/RIKEN)	CPU, GPU	FP64	Tunable (correct-rounding)
ExBLAS (Sorbonne U.)	CPU, GPU	FP64	Correct rounding
QPBLAS (JAEA/RIKEN)	CPU, GPU	DD	DD
MBLAS / MPLAPACK (M. Nakata)	CPU	FP64/DD/ QD,MPFR	FP64/DD/QD/MPFR
…			

Red: our work

BLAS-DOT2

BLAS-DOT2 [1]

- Input/output: FP64, computation: 2x FP64 (same as XBLAS)
- Based on a 2-fold precision inner-product algorithm: “Dot2” [2]
  - Based on error-free addition & multiplication, similar to double-double
  - Accuracy depends on the condition number
- Theoretical performance overhead compared to cuBLAS FP64 routines
  - DOT & GEMV: no overhead, GEMM: 11x slower

[1] Daichi Mukunoki, Takeshi Ogita: Performance and Energy Consumption of Accurate and Mixed-precision Linear Algebra Kernels on GPUs, Journal of Computational and Applied Mathematics, Vol. 372, p. 112701, 2020.
[2] T. Ogita, S. M. Rump, S. Oishi, Accurate Sum and Dot Product, SIAM J. Scientific Computing 26 (2005) 1955–1988.

Code: http://www.math.twcu.ac.jp/ogita/post-k/results.html

Accurate Inner-Product: Dot2 [Ogita et al. 2005]

Notation:

- fl (…): computation performed with floating-point arithmetic (in our case FP64)
- fma (…): computation performed with fused multiply-add
- ulp (a): unit in the last place

Accurate inner-product "Dot2"

[Ogita et al. 2005]: $r = x^{\mathrm{T}}y,\ r, x, y \in F^n$

function r = Dot2 (x, y)
  $[p, s]$ = TwoProd (x_0, y_0)
  for $i = 1 : n\text{-}1$
    $[h, r]$ = TwoProd (x_i, y_i)
    $[p, q]$ = TwoSum (p, h)
    s = fl ($s + (q + r)$)
  r = fl ($p + s$)
end function

Accurate Inner-Product: Dot2 [Ogita et al. 2005]

Notation:
- fl (…): computation performed with floating-point arithmetic (in our case FP64)
- fma (…): computation performed with fused multiply-add
- ulp (a): unit in the last place

Error-free transformation for addition

[Knuth 1969]: $s + e = a + b,\ s = \text{fl}(a + b)$

```
function [s, e] = TwoSum (a, b)
    s = fl (a + b)
    v = fl (s - a)
    e = fl (a - (s - v)) + (b - v))
end function
```

Error-free transformation for multiplication

[Karp et al. 1997]: $p + e = a \times b,\ p = \text{fl}(a \times b)$

```
function [p, e] = TwoProd (a, b)
    p = fl (a×b)
    e = fma (a×b - p)
end function
```

Accurate inner-product "Dot2"

[Ogita et al. 2005]: $r = x^{\mathrm{T}}y,\ r, x, y \in F^n$

```
function r = Dot2 (x, y)
    [p, s] = TwoProd (x_0, y_0)
    for i = 1 : n-1
        [h, r] = TwoProd (x_i, y_i)
        [p, q] = TwoSum (p, h)
        s = fl (s + (q + r))
    r = fl (p + s)
end function
```

Accurate Inner-Product: Dot2 [Ogita et al. 2005]

Notation:
- fl (…): computation performed with floating-point arithmetic (in our case FP64)
- fma (…): computation performed with fused multiply-add
- ulp (a): unit in the last place

Error-free transformation for addition
[Knuth 1969]: $s + e = a + b$, $s = \text{fl}(a + b)$

```
function [s, e] = TwoSum (a, b)
    s = fl (a + b)
    v = fl (s - a)
    e = fl (a - (s - v)) + (b - v))
end function
```

Error-free transformation for multiplication
[Karp et al. 1997]: $p + e = a \times b$, $p = \text{fl}(a \times b)$

```
function [p, e] = TwoProd (a, b)
    p = fl (a×b)
    e = fma (a×b - p)
end function
```

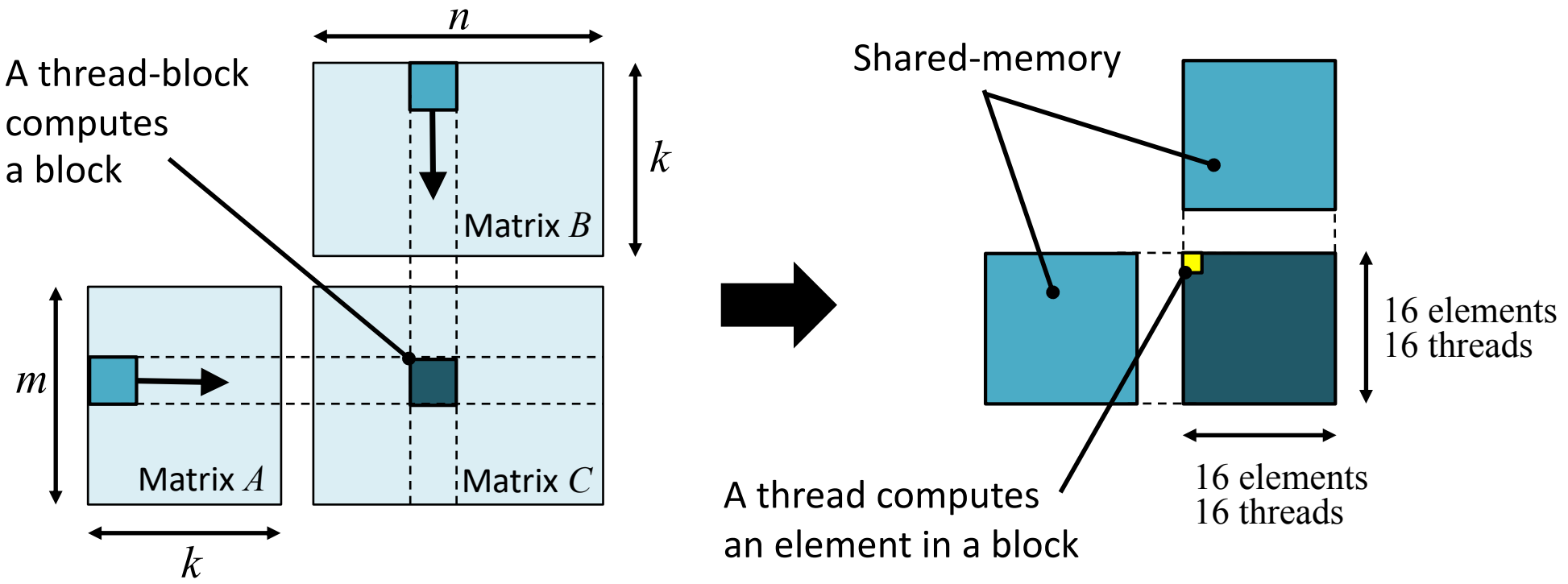
11 instructions
(11x overhead than FP64 DOT using FMA)

Accurate inner-product "Dot2"
[Ogita et al. 2005]: $r = x^{\mathrm{T}}y$, $r, x, y \in F^n$

```
function r = Dot2 (x, y)
    [p, s] = TwoProd (x_0, y_0)
    for i = 1 : n-1
        [h, r] = TwoProd (x_i, y_i)
        [p, q] = TwoSum (p, h)
        s = fl (s + (q + r))
    r = fl (p + s)
end function
```

Implementation

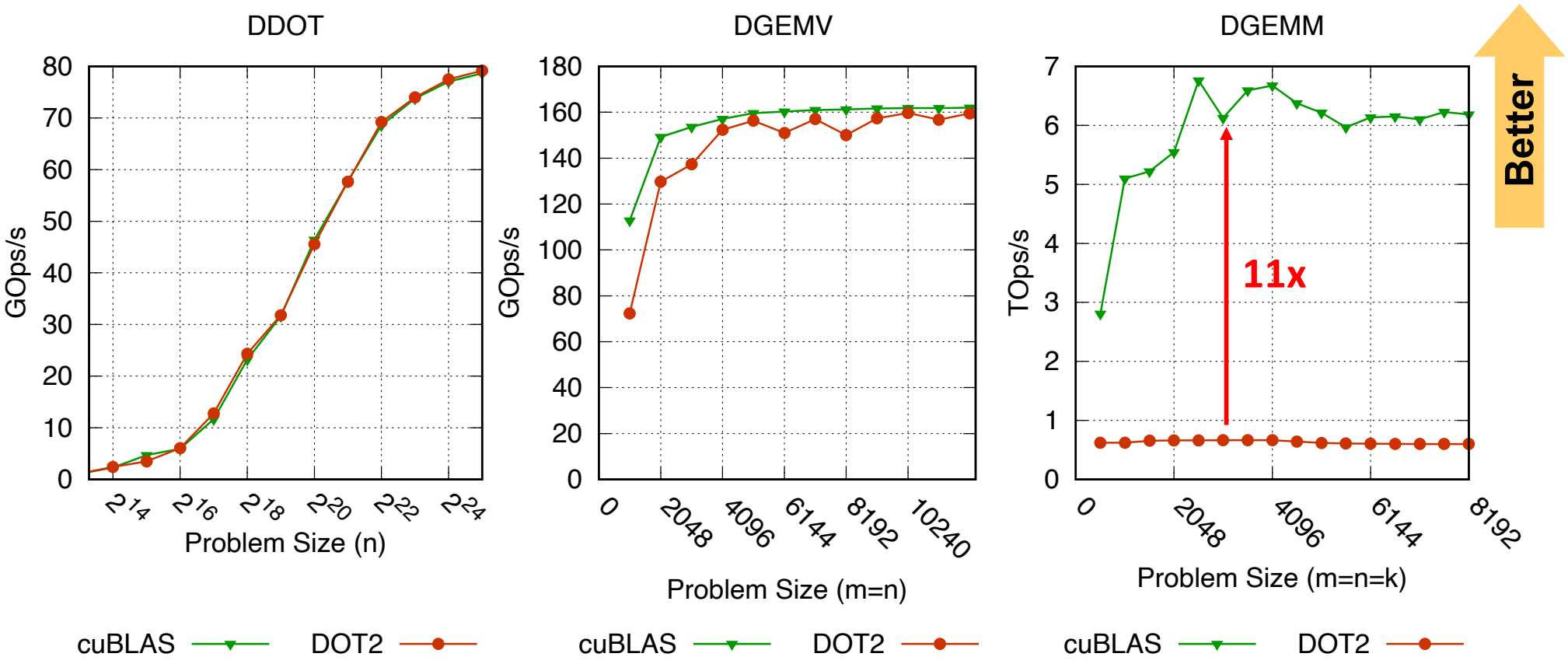
- **Matrix multiplication (GEMM) on CUDA**
  - Each thread computes an element of matrix C using Dot2
  - 2D thread-block and 2D grid utilizing 2D data locality on matrix
  - Shared-memory blocking
  - To achieve nearly the theoretical peak performance is not so hard thanks to the high compute-intensity (11x higher than double)



Performance on Titan V

Throughput (Ops/s) on Titan V (Volta), CUDA 10.0

Ops: number of mathematical computations of the BLAS operation



Execution time overhead of Dot2 v.s. cuBLAS FP64 routine:

- DOT & GEMV: Almost no overhead as memory-bound
- GEMM: 11x slower as compute-bound

Accurate BLAS Implementations

Examples of accurate BLAS implementations

	Platform	Data Precision	Computation Precision
BLAS (netlib)	CPU, GPU	FP32/64	FP32/64
XBLAS (netlib)	CPU	FP32/64	FP32/64/2xFP64
BLAS-DOT2 (TWCU)	GPU	FP64	2xFP64
ReproBLAS (Berkeley)	CPU	FP64	Tunable
OzBLAS (TWCU/RIKEN)	CPU, GPU	FP64	Tunable (correct-rounding)
ExBLAS (Sorbonne U.)	CPU, GPU	FP64	Correct rounding
QPBLAS (JAEA/RIKEN)	CPU, GPU	DD	DD
MBLAS / MPLAPACK (M. Nakata)	CPU	FP64/DD/ QD,MPFR	FP64/DD/QD/MPFR
…			

Red: our work

OzBLAS

OzBLAS [3]

- Input/output: FP64, computation: tunable (including correct-rounding)
- Based on the error-free transformation for dot/mat-mul: “Ozaki scheme” [3]
  - Accuracy depends on the range of the absolute values and the dimension (inner-product direction) of the input
- Full-scratch implementation is not needed: the kernel computation (most time-consuming part) can be performed using standard FP64 BLAS
  - High performance and low development cost
- Speed compared to cuBLAS depends on the input
  - At least more than 4x (on GEMM) and 8x (on DOT & GEMV)

[3] D. Mukunoki, T. Ogita, K. Ozaki: Reproducible BLAS Routines with Tunable Accuracy Using Ozaki Scheme for Many-core Architectures, Proc. PPAM2019, 2019 (accepted).
[4] 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. Numer. Algorithms 59(1), 95–118 (2012)
Code: http://www.math.twcu.ac.jp/ogita/post-k/results.html

Error-Free Transformation for Dot/Mat-mul: Ozaki Scheme [Ozaki et al. 2012]

DOT ($x^{\mathrm{T}}y$, $x, y \in F^n$) with Ozaki scheme:

(1) Input vectors x, y are split into several vectors:

$$x = x^{(1)} + x^{(2)} + \cdots + x^{(p)} + \cdots + x^{(sx)}$$

$$y = y^{(1)} + y^{(2)} + \cdots + y^{(q)} + \cdots + y^{(sy)} \qquad x^{(p)},\ y^{(q)} \in F^n$$

Here, the splitting is performed to meet the following properties:

1. $(x^{(p)})^{\mathrm{T}}y^{(q)} = \mathrm{fl}\left((x^{(p)})^{\mathrm{T}}y^{(q)}\right)$ for any p and q
2. $x^{(p)}(i)$ and $y^{(q)}(j)$ are non-zeros, $|x^{(p)}(i)| \geqq |x^{(p+1)}(i)|$, $|y^{(q)}(j)| \geqq |y^{(q+1)}(j)|$
   - The number of split vectors required to achieve correct-rounding depends on the range of the absolute values and length of the input vectors

Error-Free Transformation for Dot/Mat-mul: Ozaki Scheme [Ozaki et al. 2012]

DOT ($x^{\mathrm{T}}y$, $x, y \in F^n$) with Ozaki scheme:

(1) Input vectors x, y are split into several vectors:

$$x = x^{(1)} + x^{(2)} + \cdots + x^{(p)} + \cdots + x^{(sx)}$$
$$y = y^{(1)} + y^{(2)} + \cdots + y^{(q)} + \cdots + y^{(sy)} \qquad x^{(p)},\ y^{(q)} \in F^n$$

Here, the splitting is performed to meet the following properties:

1. $(x^{(p)})^{\mathrm{T}}y^{(q)} = \mathrm{fl}\left((x^{(p)})^{\mathrm{T}}y^{(q)}\right)$ for any p and q
2. $x^{(p)}(i)$ and $y^{(q)}(j)$ are non-zeros, $|x^{(p)}(i)| \geqq |x^{(p+1)}(i)|$, $|y^{(q)}(j)| \geqq |y^{(q+1)}(j)|$
   - The number of split vectors required to achieve correct-rounding depends on the range of the absolute values and length of the input vectors

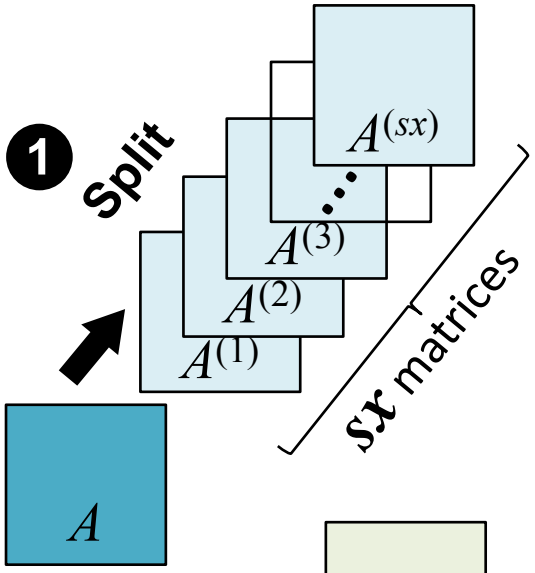
(2) $x^{\mathrm{T}}y$ is transformed to a summation of several dot-products

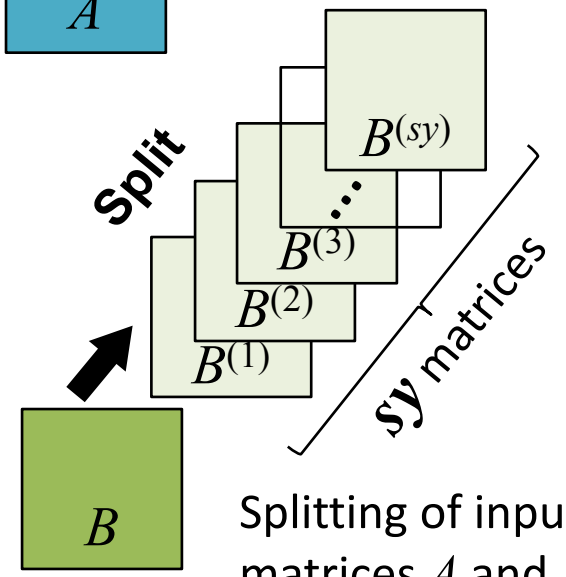
$$\begin{aligned} x^{\mathrm{T}}y = {} & (x^{(1)})^{\mathrm{T}}y^{(1)} + (x^{(1)})^{\mathrm{T}}y^{(2)} + (x^{(1)})^{\mathrm{T}}y^{(3)} + \cdots + (x^{(1)})^{\mathrm{T}}y^{(sy)} \\ & + (x^{(2)})^{\mathrm{T}}y^{(1)} + (x^{(2)})^{\mathrm{T}}y^{(2)} + (x^{(2)})^{\mathrm{T}}y^{(3)} + \cdots + (x^{(2)})^{\mathrm{T}}y^{(sy)} \\ & + (x^{(3)})^{\mathrm{T}}y^{(1)} + (x^{(3)})^{\mathrm{T}}y^{(2)} + (x^{(3)})^{\mathrm{T}}y^{(3)} + \cdots + (x^{(3)})^{\mathrm{T}}y^{(sy)} \\ & + \cdots \\ & + (x^{(sx)})^{\mathrm{T}}y^{(1)} + (x^{(sx)})^{\mathrm{T}}y^{(2)} + (x^{(sx)})^{\mathrm{T}}y^{(3)} + \cdots + (x^{(sx)})^{\mathrm{T}}y^{(sy)} \end{aligned}$$

Those parts can be computed using FP64 operation (DDOT)

With a correctly-rounded summation (e.g. *NearSum* [Rump et al. 2005]), the final result achieves correct-rounding.

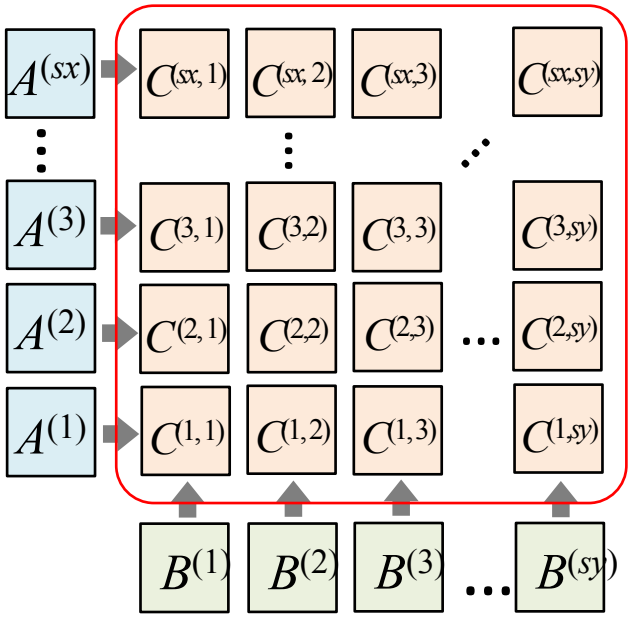
GEMM with Ozaki scheme





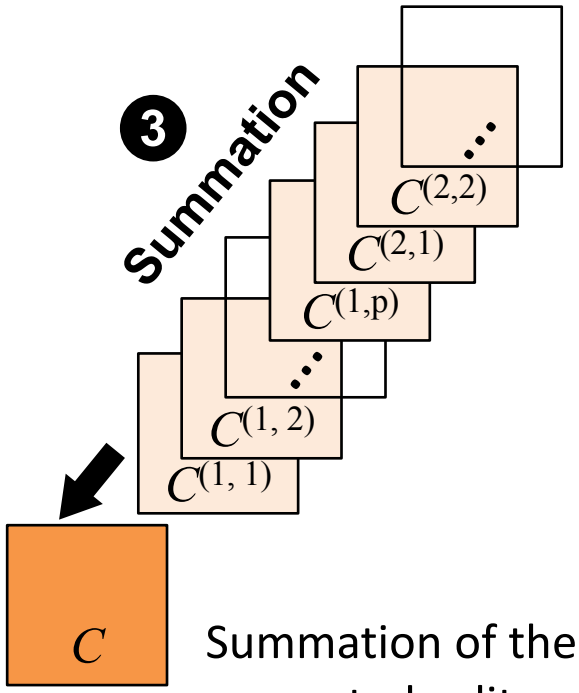
Splitting of input matrices A and B

2 Matrix Multiplications



Multiplications of the split matrices by DGEMM

Those GEMMs can be computed using standard DGEMMs



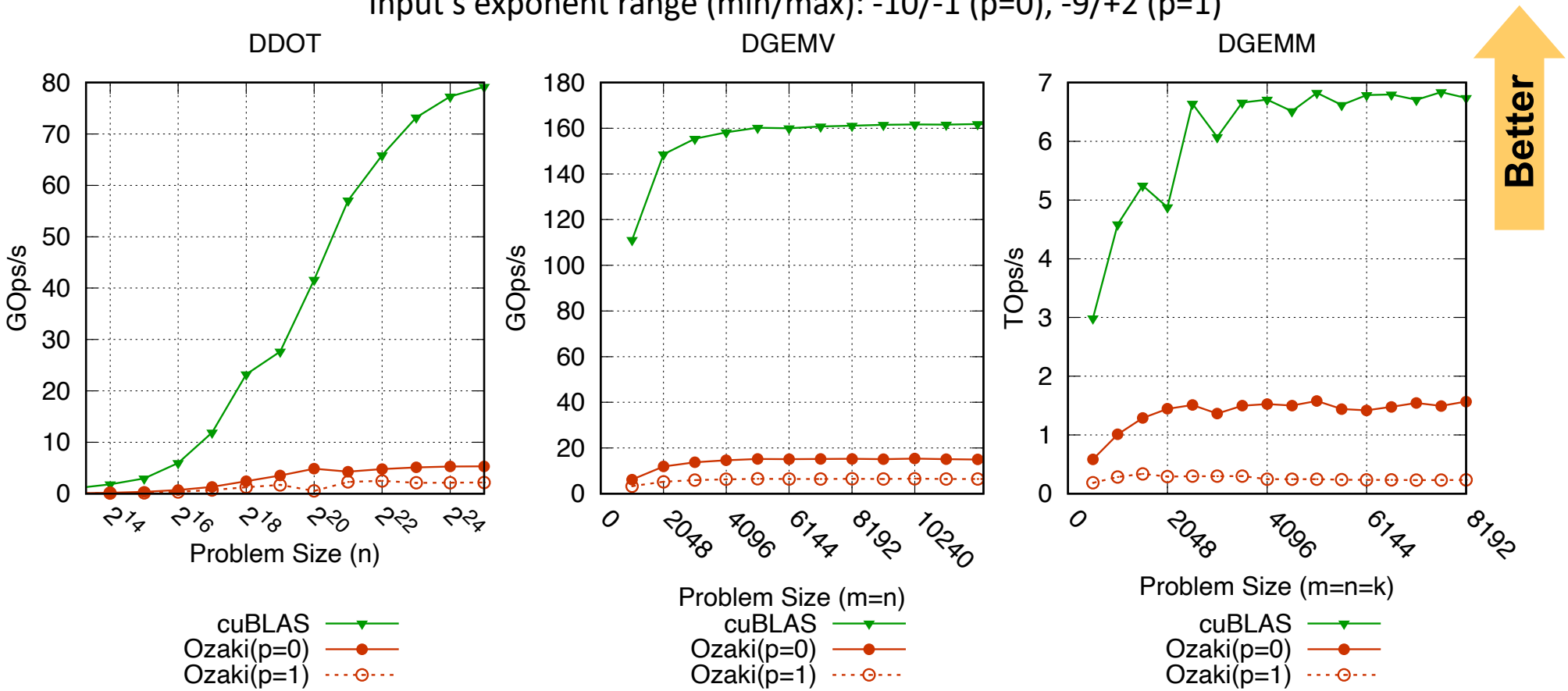
Summation of the computed split matrices by NearSum

Performance on Titan V

Throughput (Ops/s) of Correctly-Rounded Operation on Titan V (Volta), CUDA 10.1

Ops: number of mathematical computations of the BLAS operation

Input's exponent range (min/max): -10/-1 (p=0), -9/+2 (p=1)



Execution time overhead v.s. cuBLAS FP64 routine depends on the input
(the range of the absolute values of the input vectors/matrices)
e.g., more than 10x on DOT & GEMV and 4x on GEMM in above cases at minimum

Accuracy (tunable-accuracy version)

Maximum relative error to MPFR (2048-bit)

(GEMM, $m=n=k=1000$, input: $(\text{rand}-0.5)\times\exp(\phi\times\text{randn})$)

Input (min/max exp)	ϕ=1 (-07/+01)	ϕ=4 (-10/+07)	ϕ=7 (-17/+13)	ϕ=10 (-24/+19)
cuBLAS (double)	6.14E-10	2.95E-11	2.22E-10	2.27E-10
OzBLAS (d=2, fast)	4.75E-05	3.32E-02	3.92E+01	5.55E+06
OzBLAS (d=2)	1.98E-06	4.61E-03	4.11E+01	5.52E+06
OzBLAS (d=3, fast)	6.28E-12	1.80E-08	2.08E-04	8.99E+00
OzBLAS (d=3)	5.07E-13	1.17E-09	4.42E-05	6.82E+00
OzBLAS (d=4, fast)	0	7.42E-15	2.87E-10	3.65E-04
OzBLAS (d=4)	0	8.25E-16	7.79E-11	2.78E-04
OzBLAS (d=6, fast)	0	0	0	4.37E-16
OzBLAS (d=6)	0	0	0	0

†The results were compared after rounded to double-precision

- Accuracy depends on (1) the range of the absolute values in the inputs, (2) the dimension (for dot-product-wise), (3) the number of split-matrices (d)
- Ozaki scheme can be less accurate than cublasDgemm (shown in red)

Reproducibility

Results with various implementations

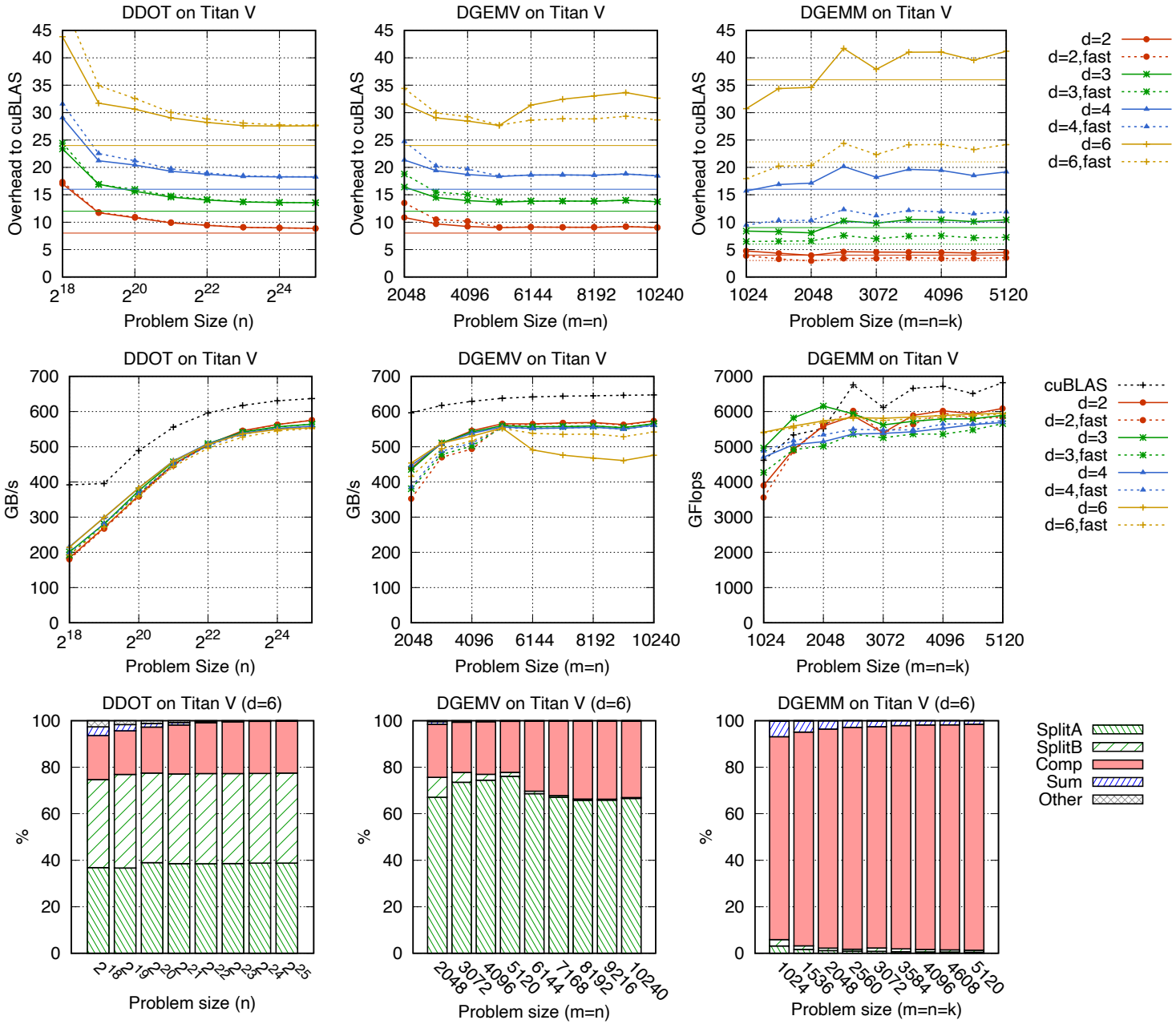
(DOT, n=10000)

	CPU (Xeon E5-2623v4)	GPU (Titan V)
MPFR (2048-bit)	0x1.33b6d7b84f15cp+7	N/A
cuBLAS (double)	N/A	0x1.33b6d7b84f15bp+7
MKL (double)	0x1.33b6d7b84f15ep+7	N/A
OzBLAS (d=1)	0x1.33b4bbaep+7	0x1.33b4bbaep+7
OzBLAS (d=2)	0x1.33b6d7b83efffp+7	0x1.33b6d7b83efffp+7
OzBLAS (d=3)	0x1.33b6d7b84f15cp+7	0x1.33b6d7b84f15cp+7

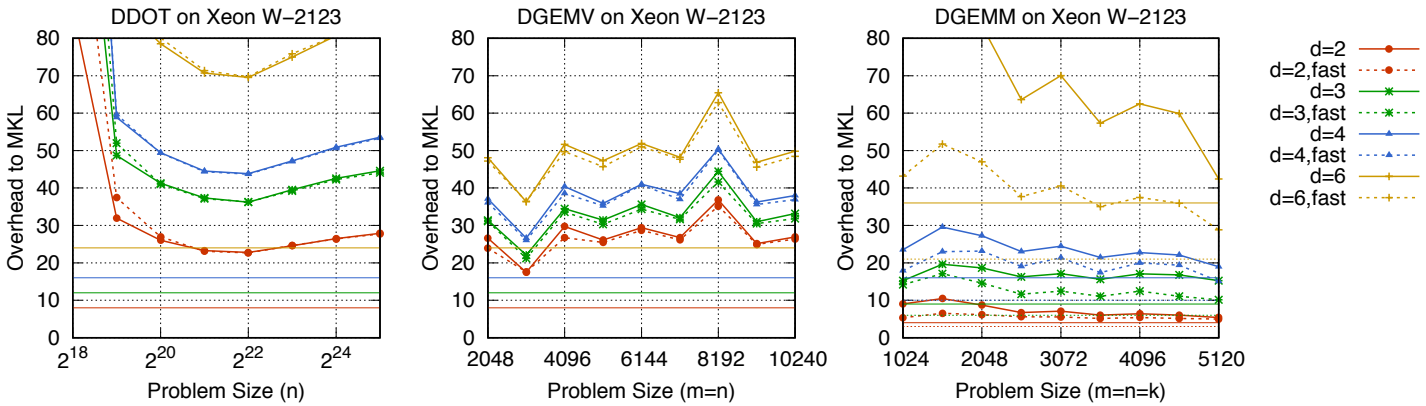
OzBLAS:

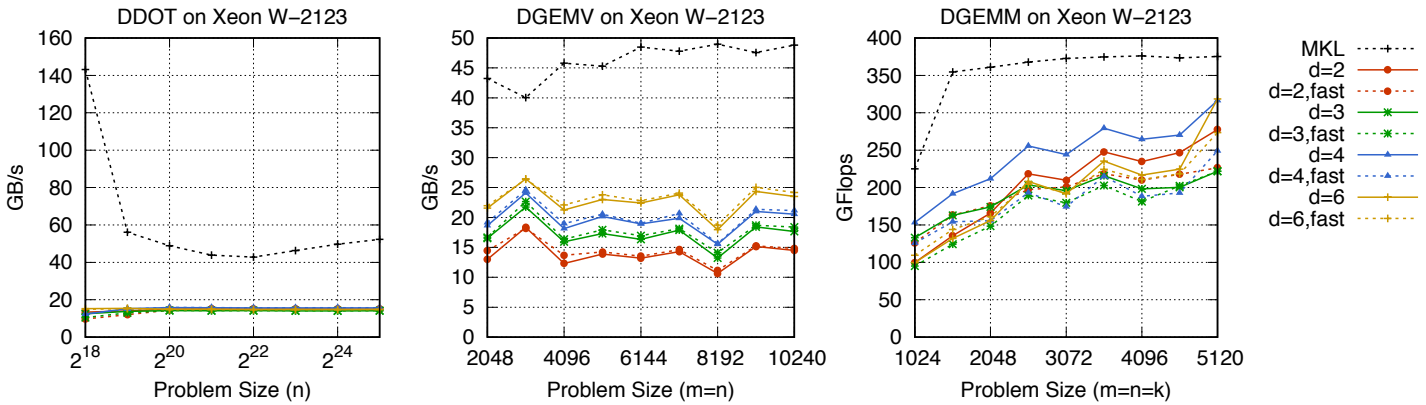
- CPU version relies on MKL and GPU version relies on cuBLAS
- when d<3, the results are not correct but reproducible between CPU and GPU

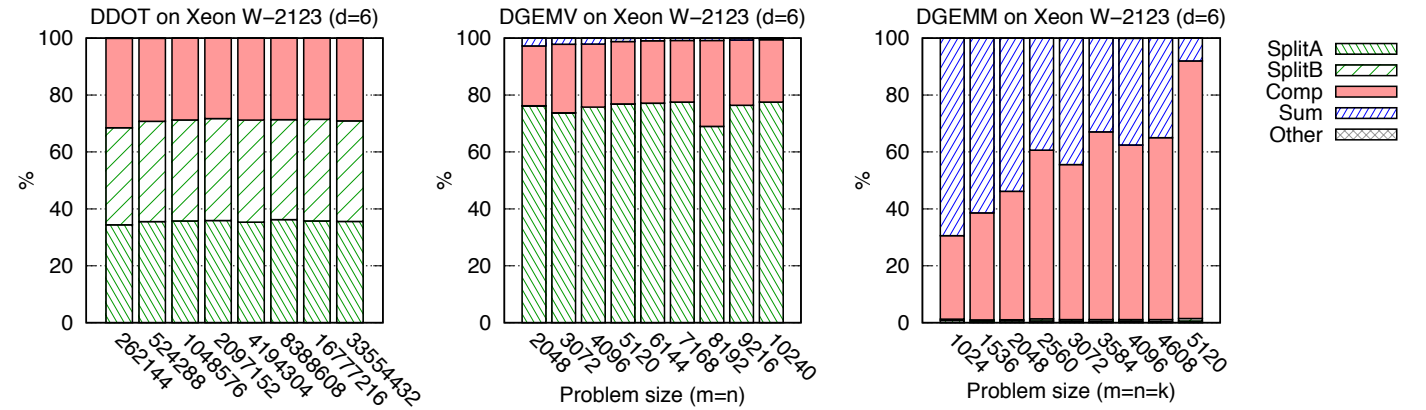
cuOzBLAS (NVIDIA Titan V)



OzBLAS (Intel Xeon W-2123)



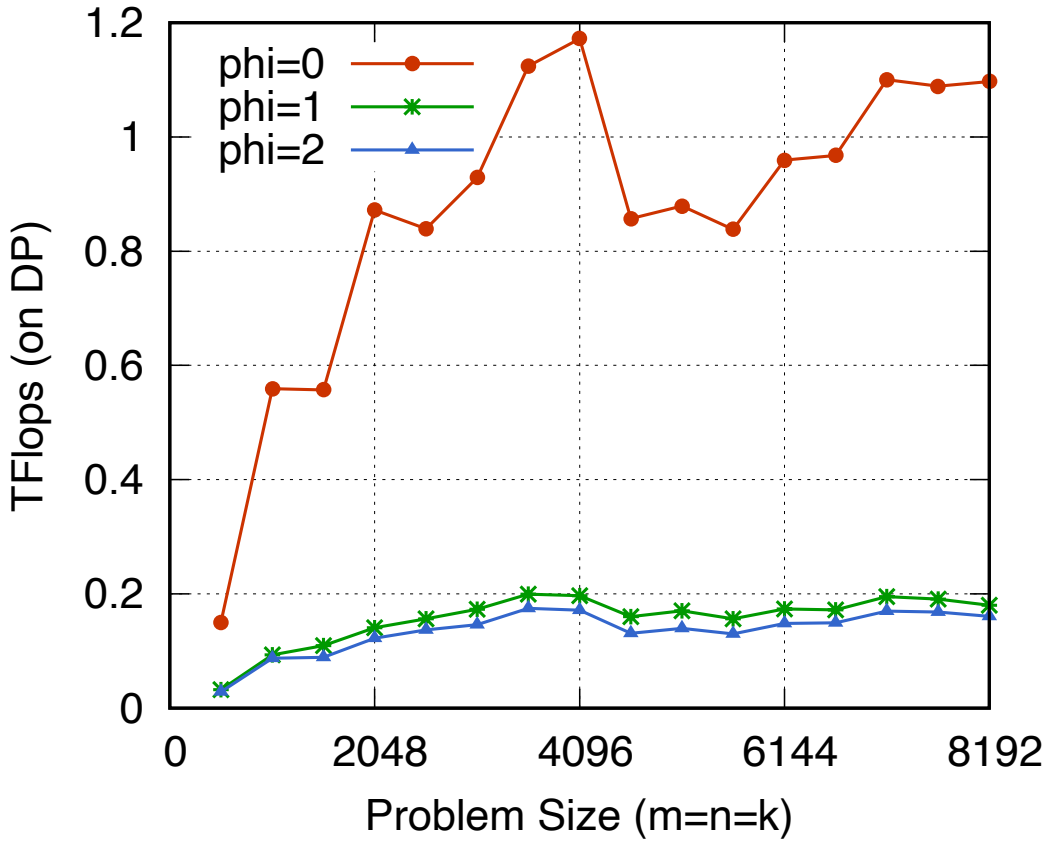




Accurate DGEMM using Tensor Cores

Throughput (TFlops on FP64) on Tesla V100 (Volta), CUDA 10.1

Input's exponent range (min/max): -10/-1 (p=0), -9/+2 (p=1), -9/+4 (p=2)



Tesla V100
- 7 TFlops on FP64
- 113 TFlops TC

With Ozaki scheme, accurate DGEMM can be built upon cublasGemmEx using Tensor Cores !! [5] (a paper incl. some extensions will be published soon…)

[5] D. Mukunoki, K. Ozaki, T. Ogita: Accurate DGEMM using Tensor Cores, HPC Asia Poster session, Jan. 2020
http://sighpc.ipsj.or.jp/HPCAsia2020/hpcasia2020_poster_abstracts/poster_abstract_09.pdf

Conclusion

Summary

- Two accurate BLAS implementations: OzBLAS and BLAS-DOT2

1. **BLAS-DOT2 [1]:** 2-fold precision for inner product with Dot2 algorithm
2. **OzBLAS [3]:** Tunable accuracy (incl. correct-rounding) with Ozaki scheme

- Both are interface compatible with standard BLAS for FP64 data, but the computation is performed with an accurate method
- Code: http://www.math.twcu.ac.jp/ogita/post-k/results.html

Future work

- Full-set implementation
- For high-precision inputs:
  - BLAS-Dot2 -> BLAS-DD (already available as MBLAS and QPBLAS)
  - Ozaki scheme for binary128 and more
- Demo of the use in minimal-precision computing