5/13 13:00-13:50 (40+10)

尾崎スキームに基づく高精度かつ再現性のあるBLASルーチンの実装と自動チューニングの適用

2019年5月13日
第22回AT研究会オープンアカデミックセッション（ATOS22）

発表者：椋木 大地（RIKEN R-CCS）
共同研究者：荻田 武史（東京女子大），尾崎 克久（芝浦工大），
今村俊幸（RIKEN R-CCS）

謝辞：本研究の一部は，文部科学省ポスト「京」萌芽的課題
「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」
および科研費（16K16062, 19K20286, 19H04127）の助成による

はじめに（1/2）

■ 浮動小数点演算と計算精度

- 有限桁による演算，丸め誤差が生じる
- 丸め誤差が蓄積する大規模計算や，悪条件問題の計算などにおいて，倍精度では精度が不足する可能性がある

■ 浮動小数点演算の再現性（reproducibility）

- 演算順序が異なると計算結果が丸め誤差レベルで異なる可能性がある
- 再現性を損なう要因の例
  - ◎ 並列計算（並列数によって演算順序が変わることがある）
  - ◎ アトミック演算（演算順序は保証されない）
  - ◎ FMA命令（Fused Multiply-Add，a*b+cを1丸めで演算）

はじめに（2/2）

高精度・再現性の必要性

- 高精度な結果が必要，あるいは有意義な結果を得るのに高精度演算が必要
- デバッグ（計算結果が異なる理由の特定）
  - 例：並列計算，GPGPU黎明期におけるFMA
- 科学実験は再現可能であるべき
  - 再計算で結果が異なる場合はその理由が理論的に説明できるべきである
- エクサスケール時代に向けて
  - “Top Ten Exascale Research Challenges”, DOE Advanced Scientific Computing Advisory Subcommittee (ASCAC) Subcommittee Report, Feb 10, 2014（レポート）
  - Computational Reproducibility at Exascale 2018 (CRE2018)（ワークショップ）
  - Workshop on Batched, Reproducible, and Reduced Precision BLAS (2016, 2017)

関連研究（1/2）

高精度な内積計算

- 多倍長精度演算による方法
  - QD [Hida et al. 2001]，MPFR [Fousse et al. 2011] など
- 無誤差変換に基づく高精度内積手法
  - Dot2/DotK [Ogita et al. 2005]，AccDot [Rump et al. 2008]
- ロングアキュムレータ方式 [Kulisch 2011]
- その他（上記方法あるいはその組み合わせに基づく）
  - ReproBLAS [Ahrens et al. 2016]，ExBLAS [Iakymchuk et al. 2015] など

再現性のある内積計算

- 計算順序を固定する方法
  - Intel Conditional Numerical Reproducibility (CNR)，NVIDIA cuBLASのSYMV
- その他
  - ReproBLAS，ExBLAS，RARE-BLAS [Chohra et al. 2016]

関連研究（2/2）

尾崎スキーム [Ozaki et al. 2008]

- Level-3 BLASに基づく高精度行列積計算手法（原理上，微修正で再現性も保証可能）
- これまでの研究
  - メモリ量削減手法 [尾崎ら 2012]
  - スレッド並列化と自動チューニング [片桐ら 2012]，[市村ら 2012]
  - Batched BLASの適用 [石黒ら 2018]

既存手法と比べた尾崎スキームの利点

- 実装コストが低い
  - 計算に通常のBLAS実装を利用できるためフルスクラッチのBLAS実装が不要
  - Level 1-3の内積系統のルーチンをほぼ共通したコードで実装可能
- 高性能：土台となるBLAS実装が高性能であればその恩恵が得られる
- 精度を調節可能で再現性も保証可能

本研究の概要

概要

- 尾崎スキームによる高精度かつ再現性のあるDOT，GEMV，GEMM
- NVIDIA GPU向けのフルGPU実装（CPU版もあり）
- GPUのグローバルメモリ上にある倍精度データを計算し，倍精度で結果を返す（cuBLASの倍精度ルーチンとインタフェース互換）
- いくつかの最適化手法の実装（特にメモリ律速なルーチン向け）

既存BLASとの比較

	Standard BLAS	ReproBLAS	ExBLAS	OzBLAS
Reproducibility	No	Yes	Yes	Yes
Data precision	S, D, C, Z	S, D, C, Z	D	D（原理上他の精度も可能）
Accuracy	S, D	Tunable (incl. Correct-rounding)	Fixed (correct-rounding)	Tunable (incl. correct-rounding)
Routines	All	SUM, DOT, GEMV, GEMM	SUM, DOT, GEMV, GEMM, TRSV* (*not accurate)	DOT, GEMV, GEMM
Platform	CPU, GPU, etc.	CPU	CPU, GPU (OpenCL)	CPU (OpenMP), GPU (CUDA)
Parallelization	OpenMP, MPI, etc.	No	OpenCL	OpenMP, CUDA
Development cost	High (full scratch)	High (full scratch)	High (full scratch)	Low (built on standard BLAS)
Performance	High	?	?	?

*As of Nov 2018

尾崎スキーム

尾崎スキーム（1/6）

■ 尾崎スキーム [Ozaki et al. 2008] [1]

- 行列積の無誤差変換＋高精度総和に基づく，高精度行列積計算手法
- 行列積を，標準の浮動小数点演算で丸め誤差なしで計算できる，複数の行列積の総和として計算する
- 原理上，わずかな工夫で再現性も保証できる
- 本研究では高精度総和にcorrect-roundingな総和アルゴリズムNearSum[2]を使用

[1] K. Ozaki, T. Ogita, S. Oishi, and S. M. Rump, “Error-free transformations of matrix multiplication by using fast routines of matrix multiplication and its applications,” *Numer. Algorithms*, vol. 59, no. 1, pp. 95–118, 2012.
[2] S. Rump, T. Ogita, and S. Oishi, “Accurate floating-point summation part II: Sign, k-fold faithful and rounding to nearest,” *SIAM Journal on Scientific Computing*, vol. 31, no. 2, pp. 1269–1302, 2008.

尾崎スキーム（2/6）

Error-free transformation of inner-product ($x^{\mathrm{T}}y$)

Split x and y by split()

$$(x^{(1)}, \underline{x}^{(2)}) = \mathrm{split}(x)$$

$$(y^{(1)}, \underline{y}^{(2)}) = \mathrm{split}(y)$$

Then,

$$x = x^{(1)} + \underline{x}^{(2)}$$

$$y = y^{(1)} + \underline{y}^{(2)}$$

Then, $x^{\mathrm{T}}y$ is transformed to

$$x^{\mathrm{T}}y = (x^{(1)})^{\mathrm{T}}y^{(1)} + (x^{(1)})^{\mathrm{T}}\underline{y}^{(2)} + (\underline{x}^{(2)})^{\mathrm{T}}y^{(1)} + (\underline{x}^{(2)})^{\mathrm{T}}\underline{y}^{(2)}$$

Here, $(x^{(1)})^{\mathrm{T}}y^{(1)}$ is error-free

$$(x^{(1)})^{\mathrm{T}}y^{(1)} = \mathrm{fl}\,((x^{(1)})^{\mathrm{T}}y^{(1)})$$

Split algorithm ($x = x' + \underline{x}'$)

$x \in F^n$, **u**: the unit round-off

$(x', \underline{x}') = \mathrm{split}(x)$ {

$\rho = \lceil(\log_2(\mathbf{u}^{-1}) + \log_2(n + 1))/2\rceil$

$\tau = \lceil\log_2(\max_{1 \le i \le n} |x_i|)\rceil$

$\sigma = 2^{(\rho+\tau)}$

$x' = \mathrm{fl}((x + \sigma) - \sigma)$

$\underline{x}' = \mathrm{fl}(x - x')$

}

尾崎スキームの精度は，
(1)入力値の絶対値の範囲，
(2)ベクトル長に依存する

尾崎スキーム（3/6）

Accurate & reproducible inner-product ($x^{\mathrm{T}}y$)

The vectors can be split recursively until $\underline{x}^{(p)}$ and $\underline{y}^{(q)}$ become zero

$$x = x^{(1)} + x^{(2)} + x^{(3)} + \cdots + x^{(p\text{-}1)} + \underline{x}^{(p)}$$

$$y = y^{(1)} + y^{(2)} + y^{(3)} + \cdots + y^{(q\text{-}1)} + \underline{y}^{(q)}$$

Then, $x^{\mathrm{T}}y$ is transformed to

$$\begin{aligned} x^{\mathrm{T}}y = {} & (x^{(1)})^{\mathrm{T}}y^{(1)} + (x^{(1)})^{\mathrm{T}}y^{(2)} + (x^{(1)})^{\mathrm{T}}y^{(3)} + \cdots + (x^{(1)})^{\mathrm{T}}y^{(q\text{-}1)} \\ & + (x^{(2)})^{\mathrm{T}}y^{(1)} + (x^{(2)})^{\mathrm{T}}y^{(2)} + (x^{(2)})^{\mathrm{T}}y^{(3)} + \cdots + (x^{(2)})^{\mathrm{T}}y^{(q\text{-}1)} \\ & + (x^{(3)})^{\mathrm{T}}y^{(1)} + (x^{(3)})^{\mathrm{T}}y^{(2)} + (x^{(3)})^{\mathrm{T}}y^{(3)} + \cdots + (x^{(3)})^{\mathrm{T}}y^{(q\text{-}1)} \\ & + \cdots \\ & + (x^{(p\text{-}1)})^{\mathrm{T}}y^{(1)} + (x^{(p\text{-}1)})^{\mathrm{T}}y^{(2)} + (x^{(p\text{-}1)})^{\mathrm{T}}y^{(3)} + \cdots + (x^{(p\text{-}1)})^{\mathrm{T}}y^{(q\text{-}1)} \end{aligned}$$

無誤差で計算された分割ベクトル同士の積の総和を高精度かつ再現性のある手法（NearSum [Rump et al. 2008]など）で計算することにより，高精度かつ再現性のある結果が得られる

尾崎スキーム（4/6）

Accurate & reproducible inner-product ($x^{\mathrm{T}}y$)

The vectors can be split recursively until $\underline{x}^{(p)}$ and $\underline{y}^{(q)}$ become zero

$$x = x^{(1)} + x^{(2)} + x^{(3)} \cancel{+ \cdots + x^{(p-1)} + \underline{x}^{(p)}}$$

$$y = y^{(1)} + y^{(2)} + y^{(3)} \cancel{+ \cdots + y^{(q-1)} + \underline{y}^{(q)}}$$

$\boxed{d=3}$

Then, $x^{\mathrm{T}}y$ is transformed to

$$\begin{aligned} x^{\mathrm{T}}y \fallingdotseq\ & (x^{(1)})^{\mathrm{T}}y^{(1)} + (x^{(1)})^{\mathrm{T}}y^{(2)} + (x^{(1)})^{\mathrm{T}}y^{(3)} \cancel{+ \cdots + (x^{(1)})^{\mathrm{T}}y^{(q-1)}} \\ & + (x^{(2)})^{\mathrm{T}}y^{(1)} + (x^{(2)})^{\mathrm{T}}y^{(2)} + (x^{(2)})^{\mathrm{T}}y^{(3)} \cancel{+ \cdots + (x^{(2)})^{\mathrm{T}}y^{(q-1)}} \\ & + (x^{(3)})^{\mathrm{T}}y^{(1)} + (x^{(3)})^{\mathrm{T}}y^{(2)} + (x^{(3)})^{\mathrm{T}}y^{(3)} \cancel{+ \cdots + (x^{(3)})^{\mathrm{T}}y^{(q-1)}} \\ & \cancel{+ \cdots} \\ & \cancel{+ (x^{(p-1)})^{\mathrm{T}}y^{(1)} + (x^{(p-1)})^{\mathrm{T}}y^{(2)} + (x^{(p-1)})^{\mathrm{T}}y^{(3)} + \cdots + (x^{(p-1)})^{\mathrm{T}}y^{(q-1)}} \end{aligned}$$

計算に使用する分割ベクトル数（計算深度 d ）を調節することにより，計算精度を調節可能

尾崎スキーム（5/6）

Accurate & reproducible inner-product ($x^{\mathrm{T}}y$)

The vectors can be split recursively until $\underline{x}^{(p)}$ and $\underline{y}^{(q)}$ become zero

$$x = x^{(1)} + x^{(2)} + x^{(3)} \cancel{+ \cdots + x^{(p-1)} + \underline{x}^{(p)}}$$

$$y = y^{(1)} + y^{(2)} + y^{(3)} \cancel{+ \cdots + y^{(q-1)} + \underline{y}^{(q)}}$$

$\boxed{d=3}$

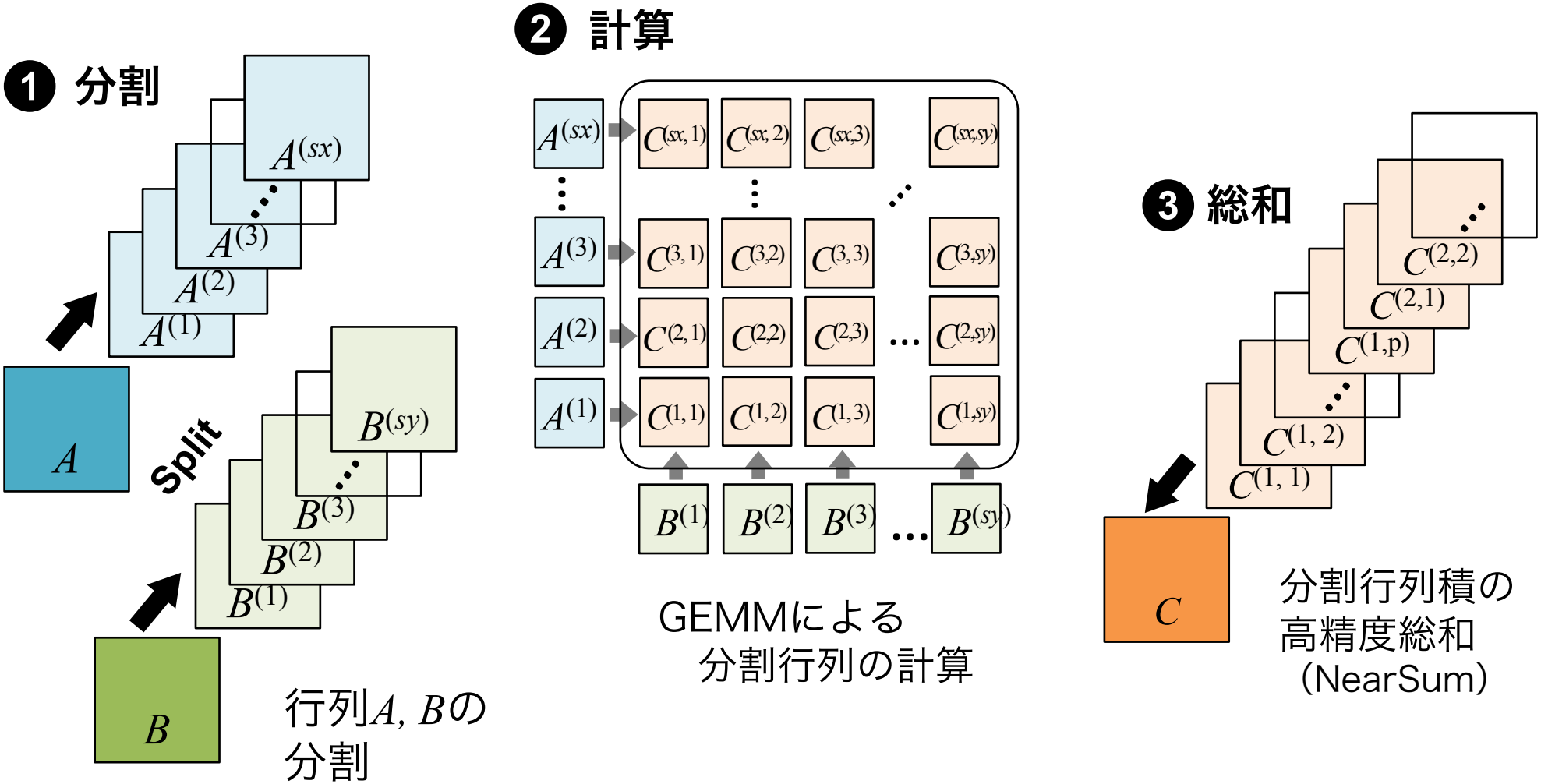
Then, $x^{\mathrm{T}}y$ is transformed to

$$\begin{aligned} x^{\mathrm{T}}y \fallingdotseq\ & (x^{(1)})^{\mathrm{T}}y^{(1)} + (x^{(1)})^{\mathrm{T}}y^{(2)} + (x^{(1)})^{\mathrm{T}}y^{(3)} \cancel{+ \cdots + (x^{(1)})^{\mathrm{T}}y^{(q-1)}} \\ & + (x^{(2)})^{\mathrm{T}}y^{(1)} + (x^{(2)})^{\mathrm{T}}y^{(2)} \cancel{+ (x^{(2)})^{\mathrm{T}}y^{(3)}} \cancel{+ \cdots + (x^{(2)})^{\mathrm{T}}y^{(q-1)}} \\ & + (x^{(3)})^{\mathrm{T}}y^{(1)} \cancel{+ (x^{(3)})^{\mathrm{T}}y^{(2)}} \cancel{+ (x^{(3)})^{\mathrm{T}}y^{(3)}} \cancel{+ \cdots + (x^{(3)})^{\mathrm{T}}y^{(q-1)}} \\ & \cancel{+ \cdots} \\ & \cancel{+ (x^{(p-1)})^{\mathrm{T}}y^{(1)} + (x^{(p-1)})^{\mathrm{T}}y^{(2)} + (x^{(p-1)})^{\mathrm{T}}y^{(3)} + \cdots + (x^{(p-1)})^{\mathrm{T}}y^{(q-1)}} \end{aligned}$$

計算深度 d が指定されたときに $(x^{(p)})^{\mathrm{T}}y^{(q)}$ の計算で $p+q > d+1$ を省略することにより，わずかな精度低下と引き換えに計算量を大きく削減できる (**fast mode**)

尾崎スキーム（6/6）

■ 尾崎スキームによる行列積計算の全体イメージ



実装・最適化

実装

概要

- 現時点ではDDOT, DGEMV, DGEMMのCPU・GPU（CUDA）版を実装
- 基本的には通常の倍精度BLASと互換
  - 一部のBLAS機能（転置モード，スカラ倍等）は未実装
  - 作業メモリ確保と精度指定を行う初期化ルーチンを提供

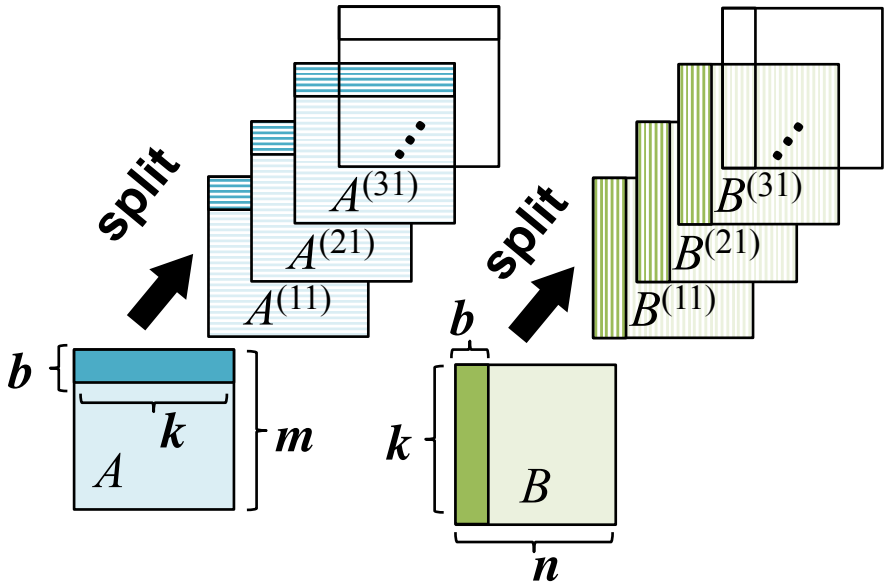
GPU実装

- 分割・計算・総和の3パートで構成
  - 分割・総和（NearSum）：それぞれCUDAカーネルを実装
  - 計算：cuBLASを使用
    - 性能最適化のため一部に独自実装のBLASルーチンを使用
- いくつかの実装最適化手法を適用
- 実装方法にGPU固有の話はほぼない（x86版もMKLベースで作れる）

最適化（1/4）

■ ブロッキングによる省メモリ化 [尾崎・荻田2012]

- 尾崎スキームは分割行列の格納のために大量のメモリを消費する
- 内積方向に短冊形にブロック化し，ブロックを逐次的に処理していくことにより，分割行列を保持するメモリを削減できる
  - ブロックサイズが小さすぎると行列積の実行効率が低下する恐れがある
  - ブロックサイズが1のとき行列積を内積で計算することと同等になり，最も省メモリであるが性能は最悪となる
- 現状では1万次元程度がメモリに載るよう適当なブロックサイズを設定しているが，メモリに載る最大ブロックサイズを自動決定することが望まれる



最適化（2/4）

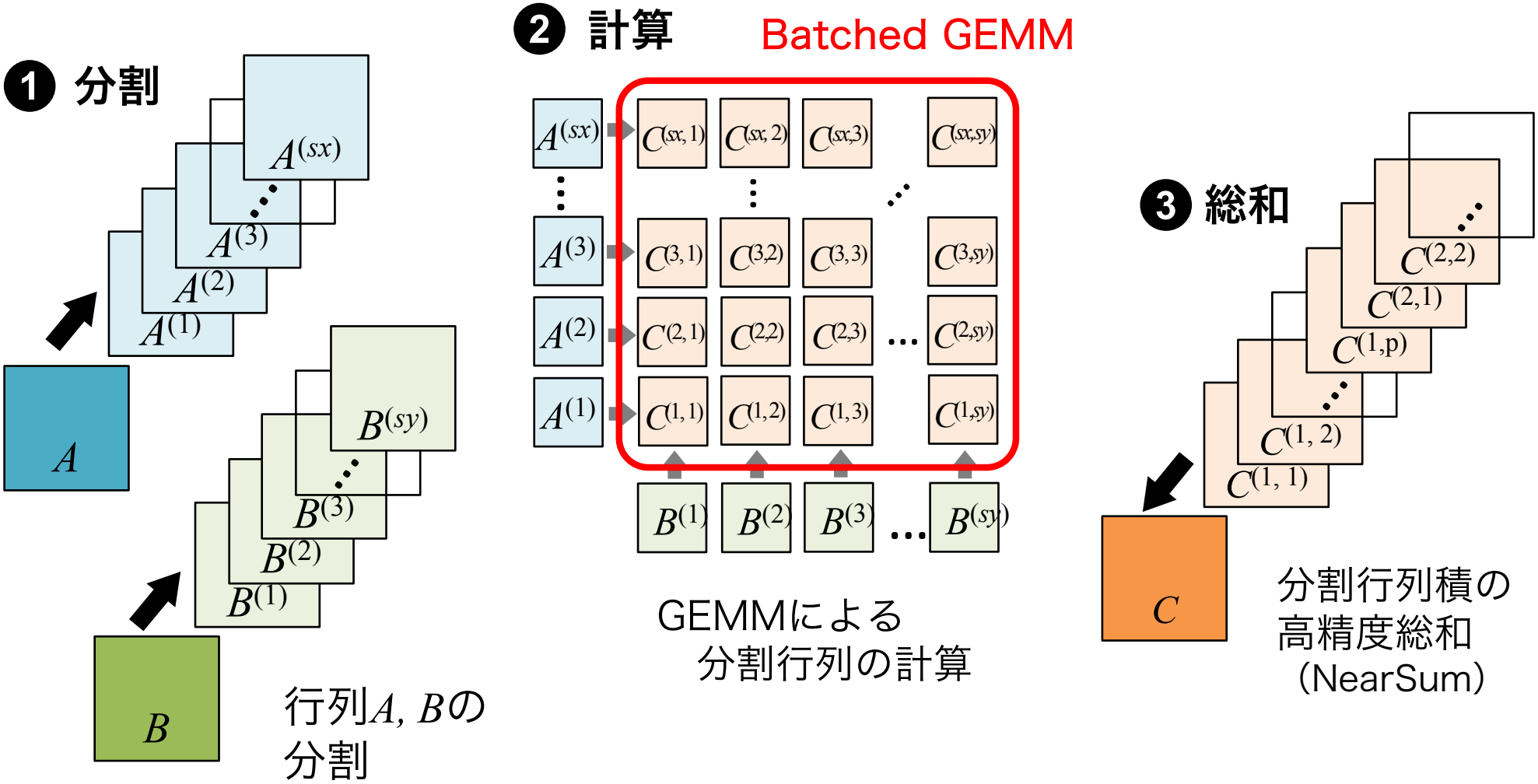
Batched BLAS

- 依存関係のない複数の同一BLAS計算を同時に処理するインタフェース
- メニーコアプロセッサにおいて，並列数に比べて小規模な計算を同時に処理することにより，演算器の実行効率向上を目指したもの
- Intel MKL，NVIDIA cuBLAS，Magma（テネシー大）などがGEMM等の一部ルーチンのバッチ版を提供

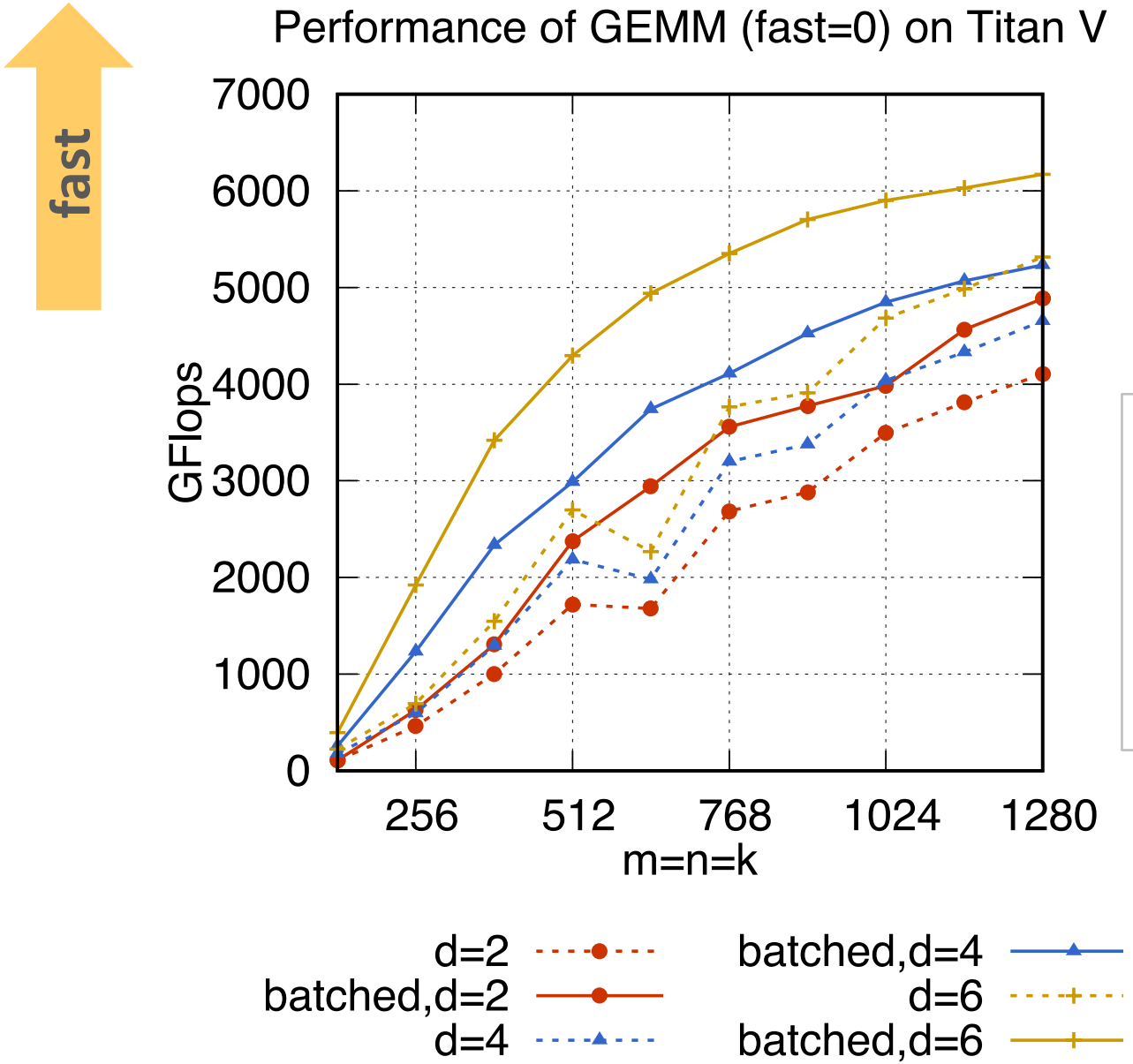
尾崎スキームへのbatched BLASの適用 [石黒ら2018]

- 分割ベクトル・行列の計算は依存関係がなく同時に計算可能
- GEMMにおいてはbatched GEMMで計算可能
- 問題サイズ・ブロックサイズが小さい時に有効

行列積におけるbatched GEMMの適用



GEMMにおけるbatchedGEMMの使用効果

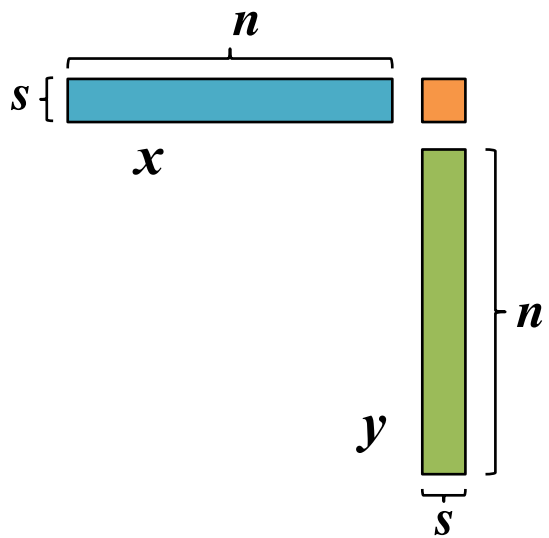


- GPU: NVIDIA Titan V (Volta architecture, 7449.6 GFlops on DP, 652.8 GB/s 12GB)
- CUDA 9.1 (cuBLAS 9.1), driver version: 390.67

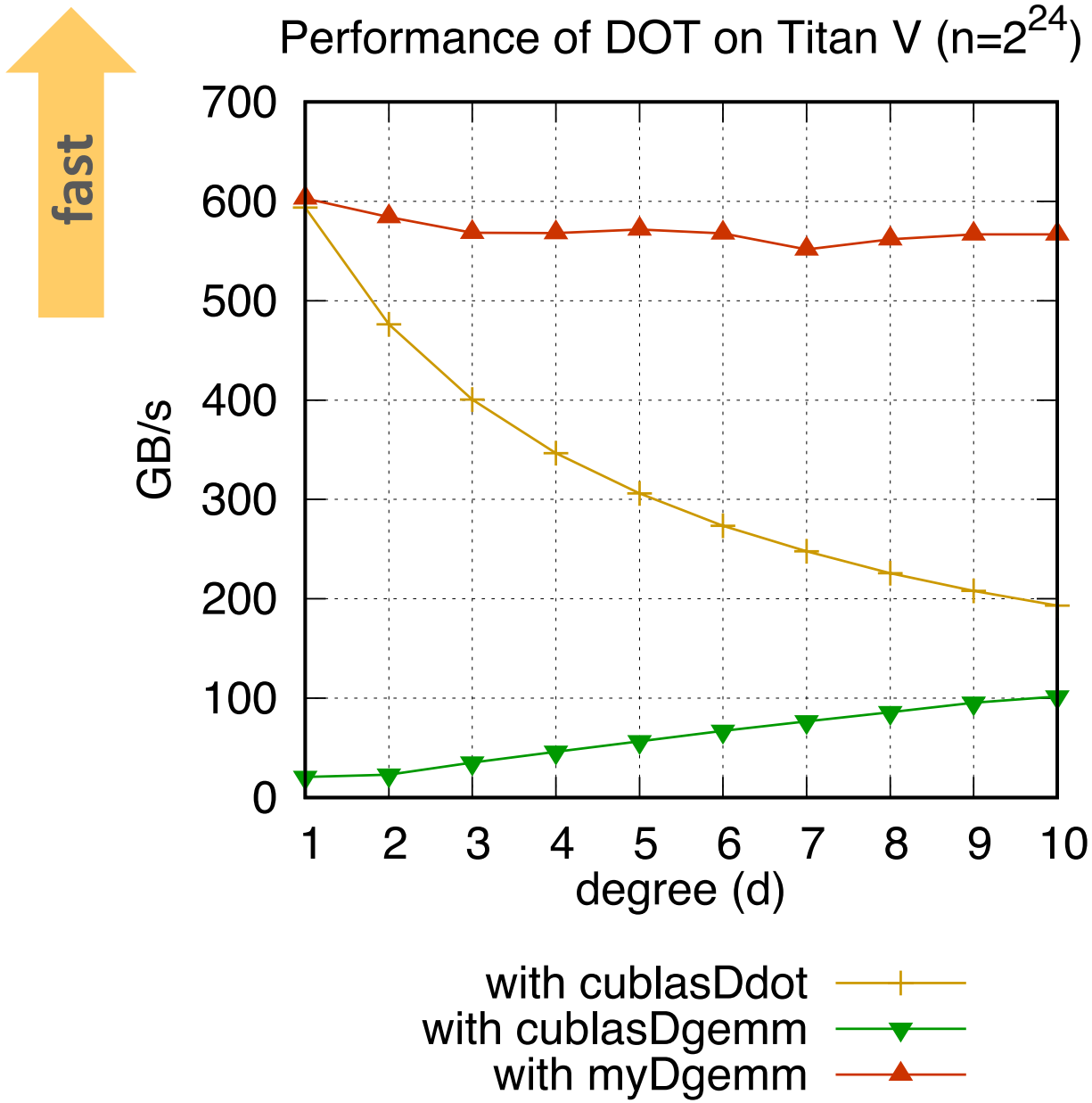
最適化（3/4）

DOT・GEMVのGEMMによる計算

- 複数の分割ベクトルを1つの行列として扱うことで，DOT・GEMVをGEMMで計算できる
  - 分割数が問題サイズに対して十分に小さければ，そのGEMMの性能はメモリ律速となり，データの再利用性の向上による性能向上が期待できる
- メニーコア向けのGEMMは行列の2次元のデータ並列性を用いて並列化されていることが多く，この用途ではデータ並列度が不足する場合がある
  - CuBLASのDGEMMはこの問題によりDOTの計算に用いると性能が著しく悪いため，本研究では独自に実装したGEMMを使用
  - 自動チューニングで計算に使用するDGEMM実装を切り替えることが望ましい



DOTにおけるGEMMの使用効果



- GPU: NVIDIA Titan V (Volta architecture, 7449.6 GFlops on DP, 652.8 GB/s 12GB)
- CUDA 9.1 (cuBLAS 9.1), driver version: 390.67

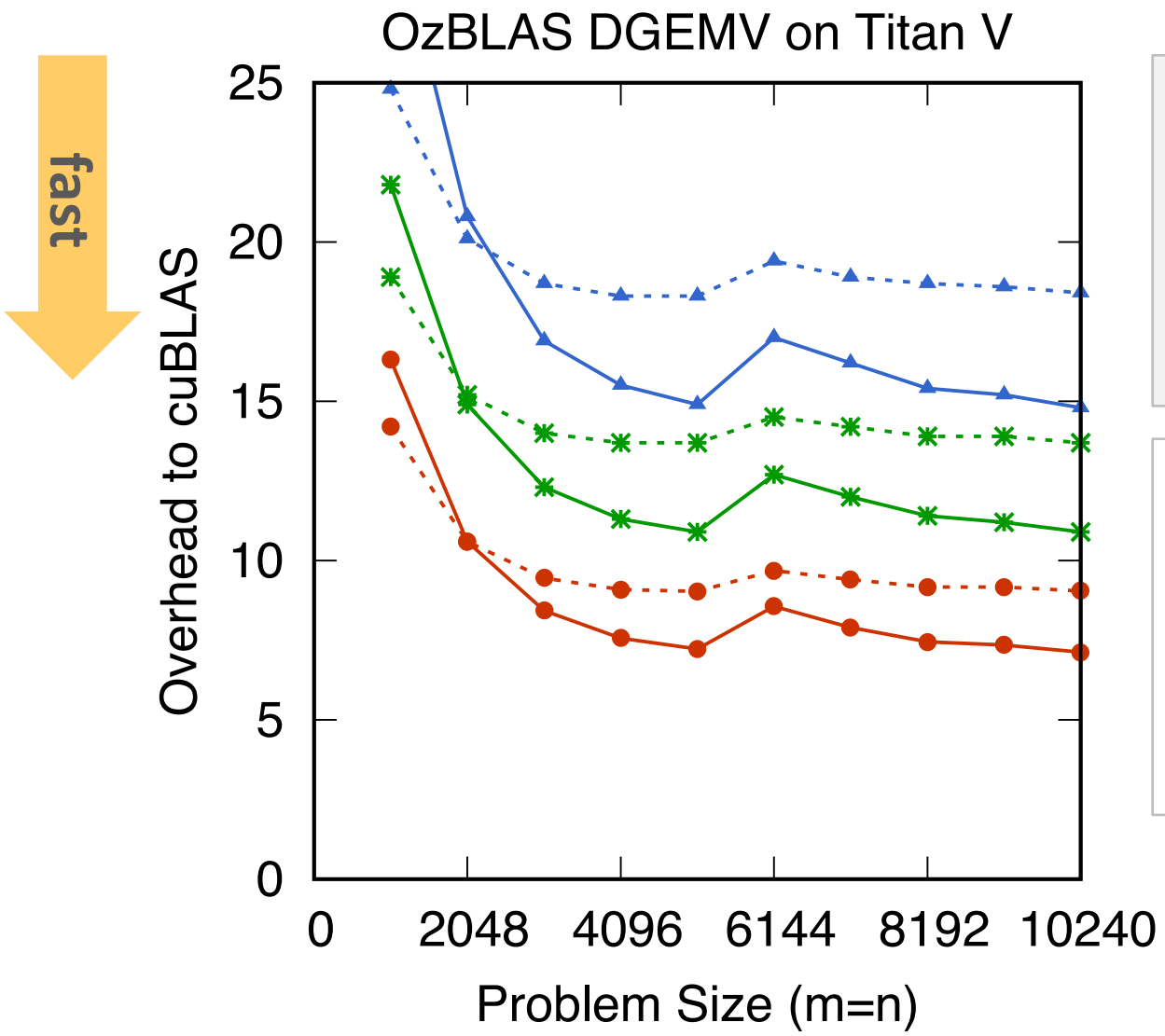
最適化（4/4）

■ 分割データのINT32表現

- 分割データは指数部をシフトすることで32ビット整数型に格納できる
  - ◎ 分割データの仮数部は多くとも26ビットが有効情報を持ち，指数部は分割対象のベクトルの最大値で決まる
  - ◎ 最大値が32ビット整数の最大値となるように指数部をシフトする
- 分割データへのメモリアクセス時間を半減できるため，メモリ律速な処理（DOT・GEMV）において高速化が期待できる
- ただしint32_tデータと指数部シフト量から元の値に復元して計算するためのBLASカーネルを自分で実装する必要がある
  - ➢ 尾崎スキームの大きな利点である既存BLASの活用ができなくなる

GEMVにおけるINT32表現の効果

※分割行列のみをInt32_tで格納し，int32_tをレジスタ上でdoubleに変換して計算



d=2
d=2,INT32
d=3
d=3,INT32
d=4
d=4,INT32

- GPU: NVIDIA Titan V (Volta architecture, 7449.6 GFlops on DP, 652.8 GB/s 12GB)
- CUDA 9.1 (cuBLAS 9.1), driver version: 390.67

評価実験

精度

MPFR 2048-bitと比較した最大相対誤差（doubleに丸めて比較）

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

Input (value range)	ϕ=1 (1.13E-07 ~ 5.23E+01)	ϕ=4 (4.07E-10 ~ 6.00E+07)	ϕ=7 (6.21E-17 ~ 6.88E+13)	ϕ=10 (9.46E-24 ~ 7.89E+19)
cuBLAS (double)	6.14E-10	2.95E-11	2.66E-10	4.14E-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	2.18E-16
OzBLAS (d=6)	0	0	0	0

- 精度は入力行列の要素の絶対値のレンジと計算深度 d（使用した分割行列個数）に依存する
- cuBLASのDGEMMよりも低精度になりうる（赤字）

理論性能

■ DOT・GEMV：通常のBLASに対する性能オーバーヘッド：$4d$ 倍

- 性能はメモリ律速：分割プロセスおよび分割行列の計算におけるベクトル参照回数（DOT）・行列参照回数（GEMV）で性能をモデル化

DOTにおけるベクトル参照回数

	Split	Comp	Total
Double	-	2	2
Ozaki	$6d$	$2d^2$	$2d^2+6d$
Ozaki (MM)	$6d$	$2d$	$8d$
Ozaki (MM+IN)	$5d$	d	$6d$

GEMVにおける行列参照回数

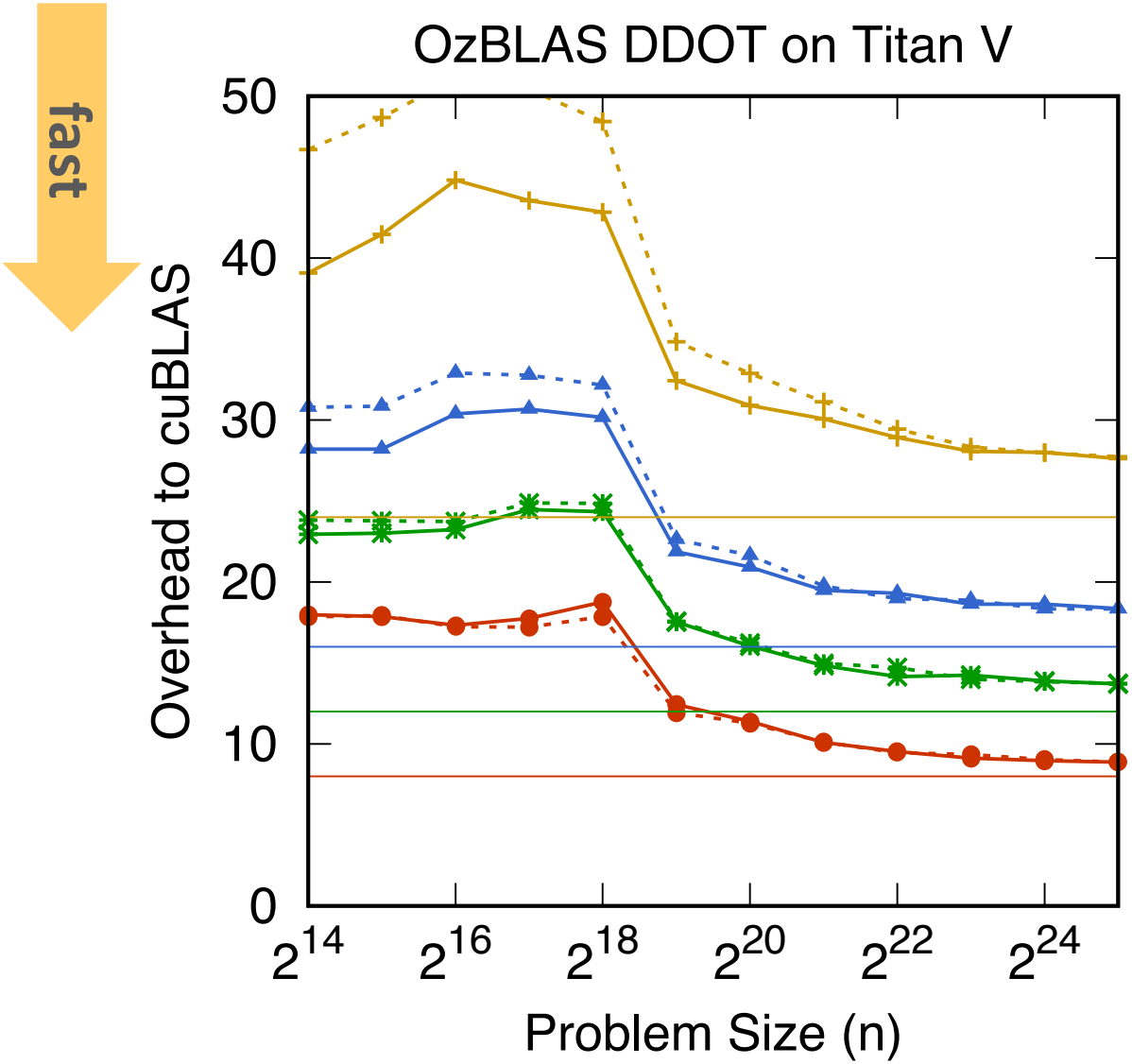
	Split	Comp	Total
Double	-	1	1
Ozaki	$3d$	d^2	d^2+3d
Ozaki (MM)	$3d$	d	$4d$
Ozaki (MM+IN)	$2.5d$	$0.5d$	$3d$

※ d：計算深度（分割行列の個数），”MM”：GEMMによる計算，”IN”：32ビット整数表現

■ GEMM：通常のBLASに対する性能オーバーヘッド：通常モード：d^2 倍，fastモード：$d(d+1)/2$ 倍

- 性能は演算律速：分割行列の計算回数（GEMMの回数）で性能をモデル化
- 計算回数：計算の簡略化を行わないと d^2 回，簡略化した場合：$d(d+1)/2$ 回

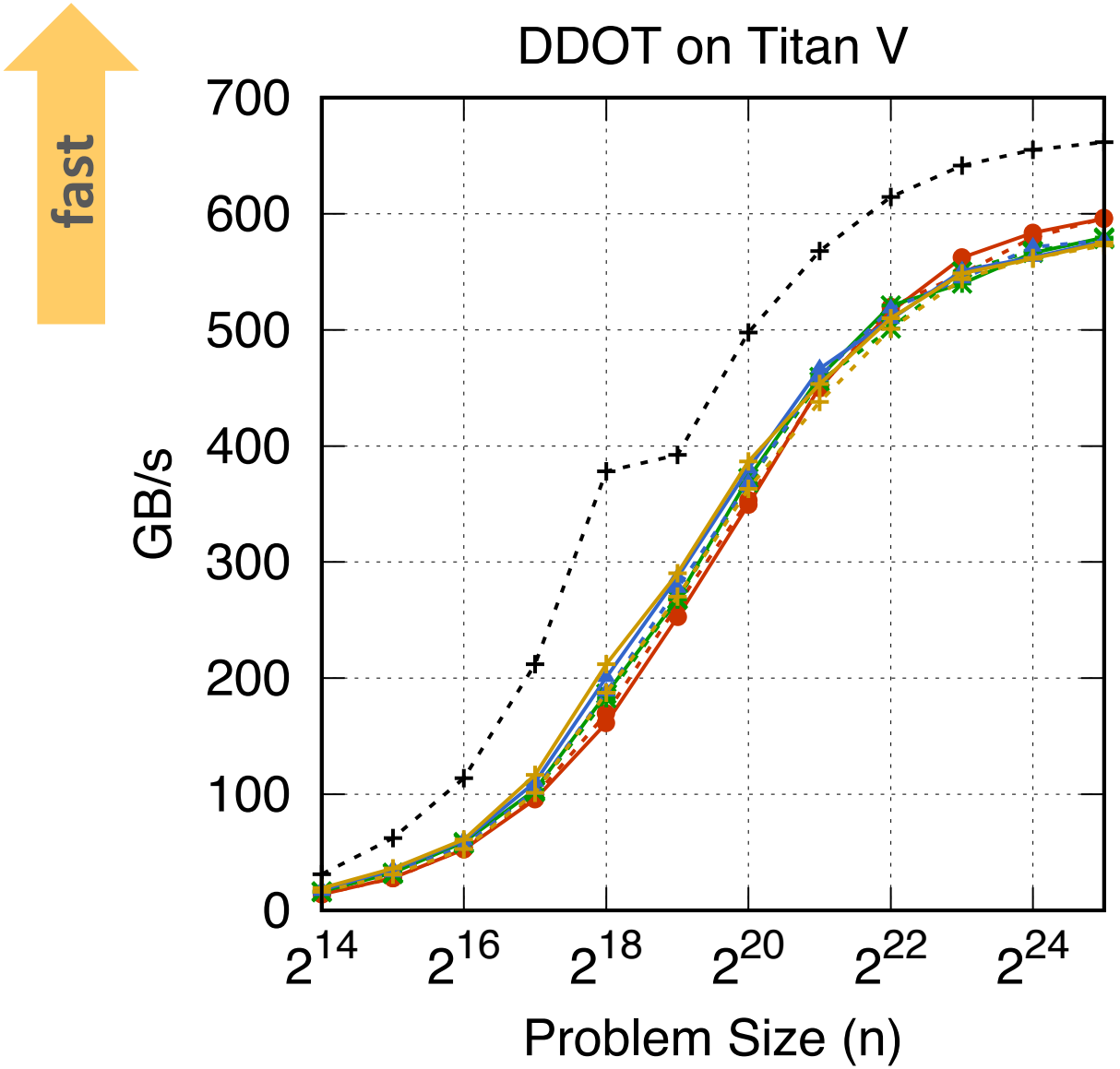
DOT – cublasDdotに対するオーバーヘッド

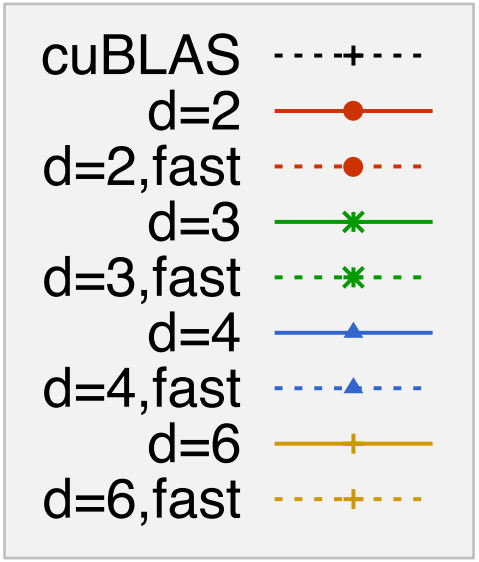


d=2
d=2,fast
d=3
d=3,fast
d=4
d=4,fast
d=6
d=6,fast
Straight lines: theoretical

- Input: ϕ=12
- GPU: NVIDIA Titan V (Volta architecture, 7449.6 GFlops on DP, 652.8 GB/s 12GB)
- CUDA 9.1 (cuBLAS 9.1), driver version: 390.67

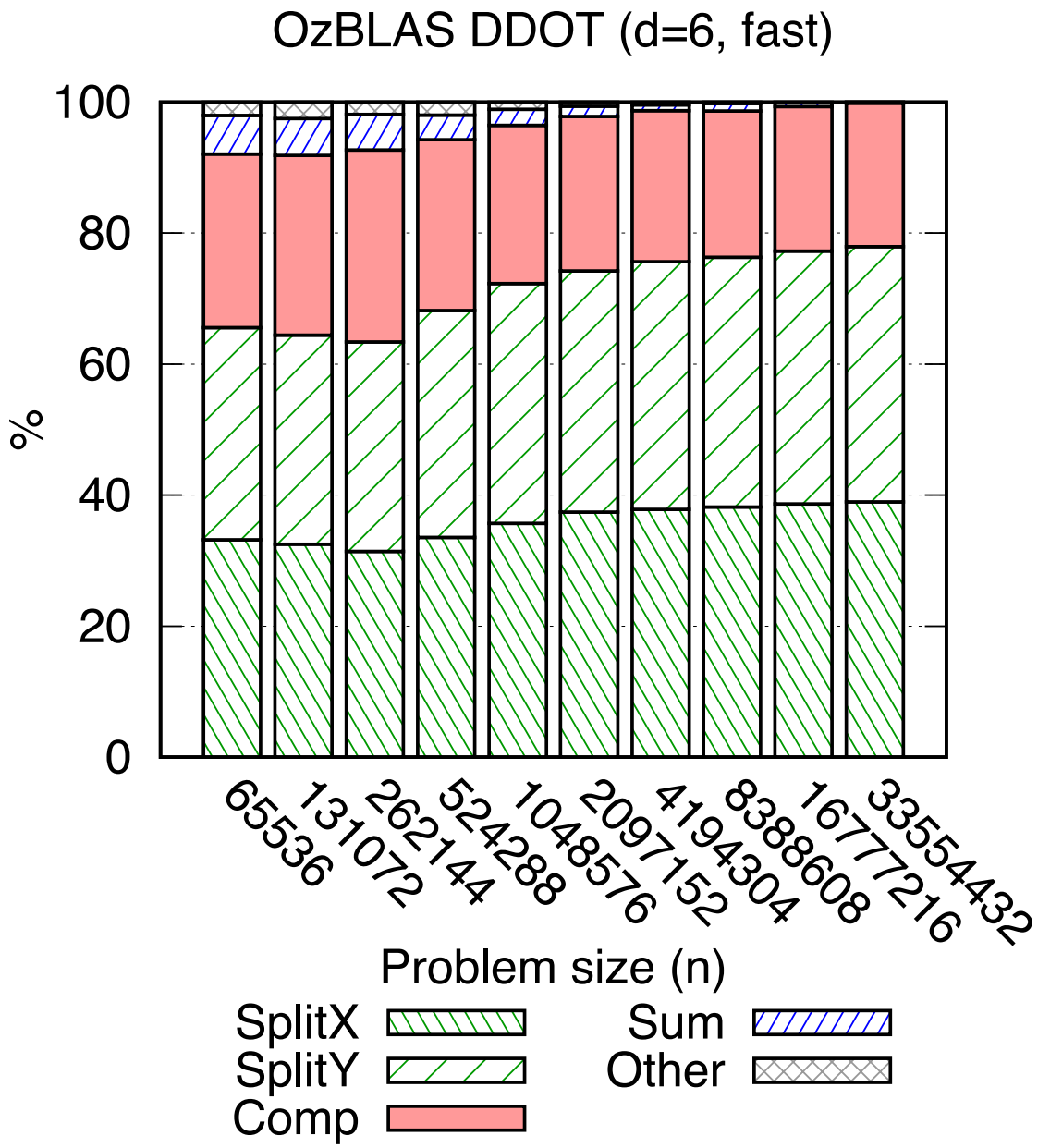
DOT – スループット（GB/s）



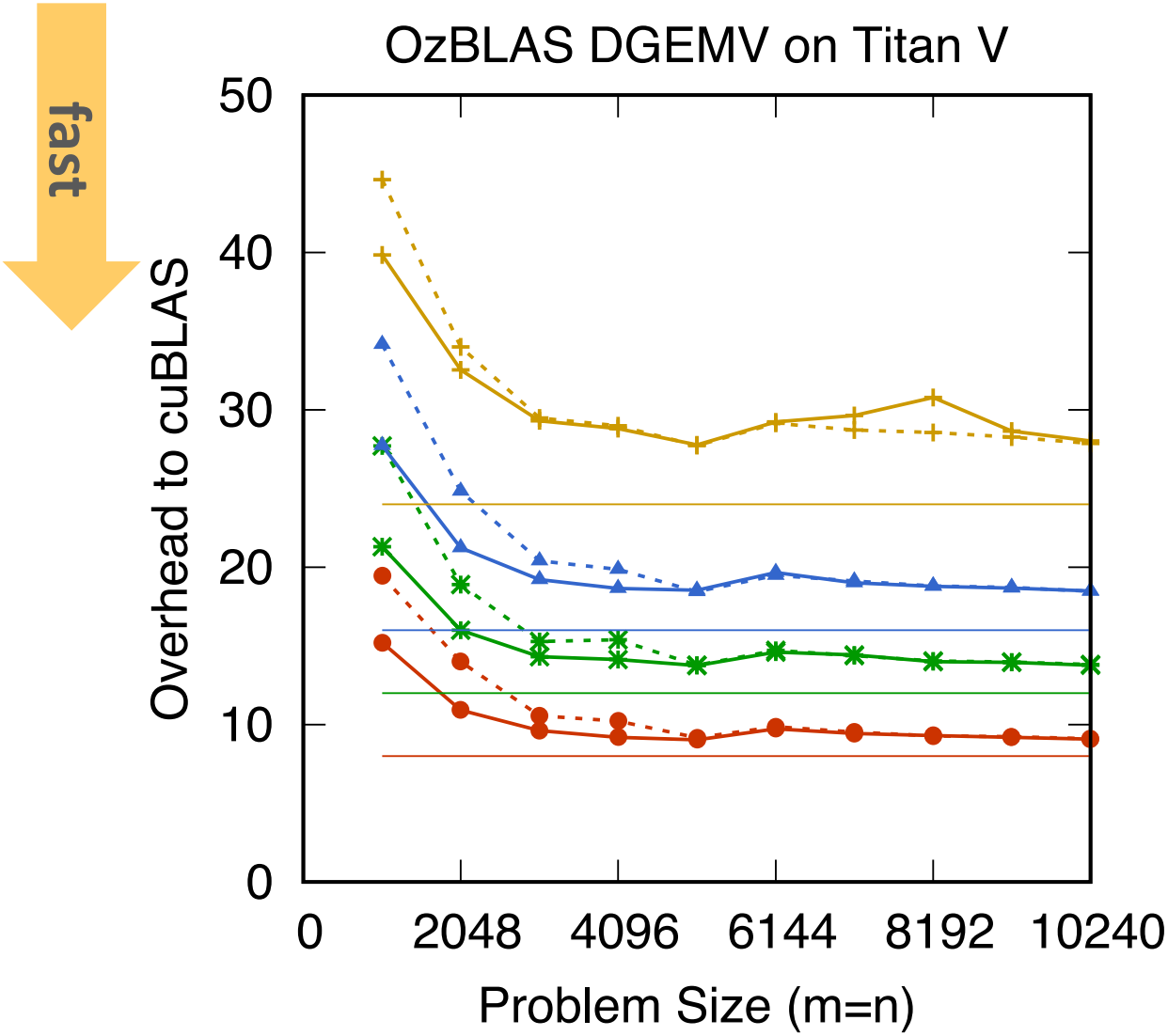


- Input: ϕ=12
- GPU: NVIDIA Titan V (Volta architecture, 7449.6 GFlops on DP, 652.8 GB/s 12GB)
- CUDA 9.1 (cuBLAS 9.1), driver version: 390.67

DOT – 性能内訳


OzBLAS DDOT (d=6, fast)
100
80
60
40
20
0
%
65536
131072
262144
524288
1048576
2097152
4194304
8388608
16777216
33554432
Problem size (n)
SplitX
SplitY
Comp
Sum
Other

GEMV – cublasDgemvに対するオーバーヘッド

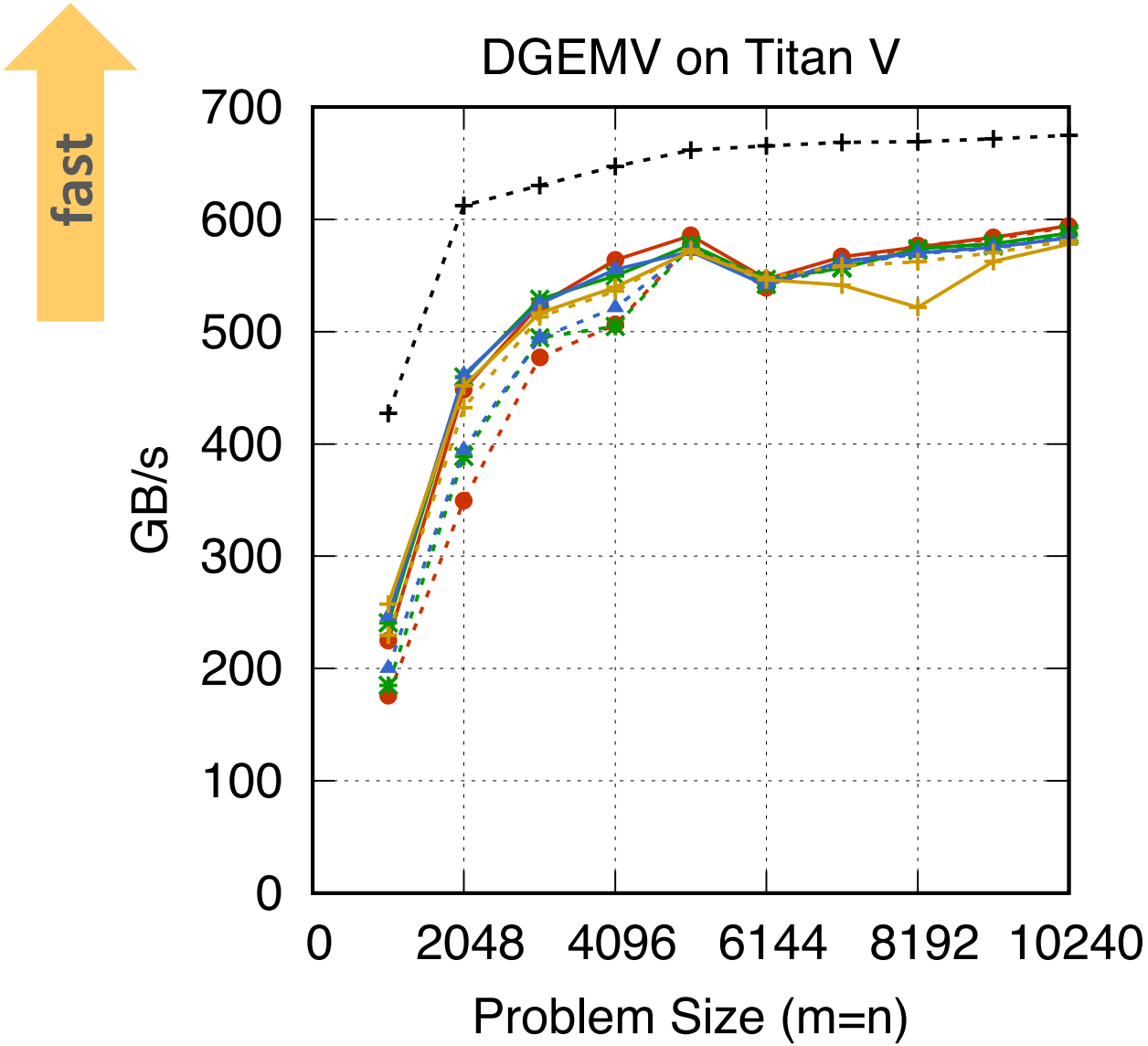


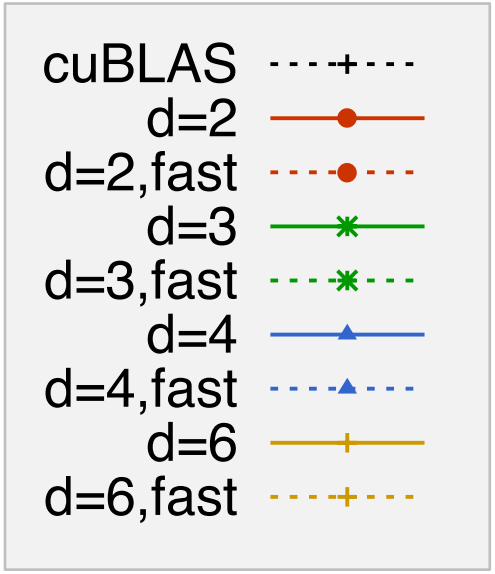
d=2
d=2,fast
d=3
d=3,fast
d=4
d=4,fast
d=6
d=6,fast

Straight lines: theoretical

- Input: ϕ=12
- Block-size: b=5120
- GPU: NVIDIA Titan V (Volta architecture, 7449.6 GFlops on DP, 652.8 GB/s 12GB)
- CUDA 9.1 (cuBLAS 9.1), driver version: 390.67

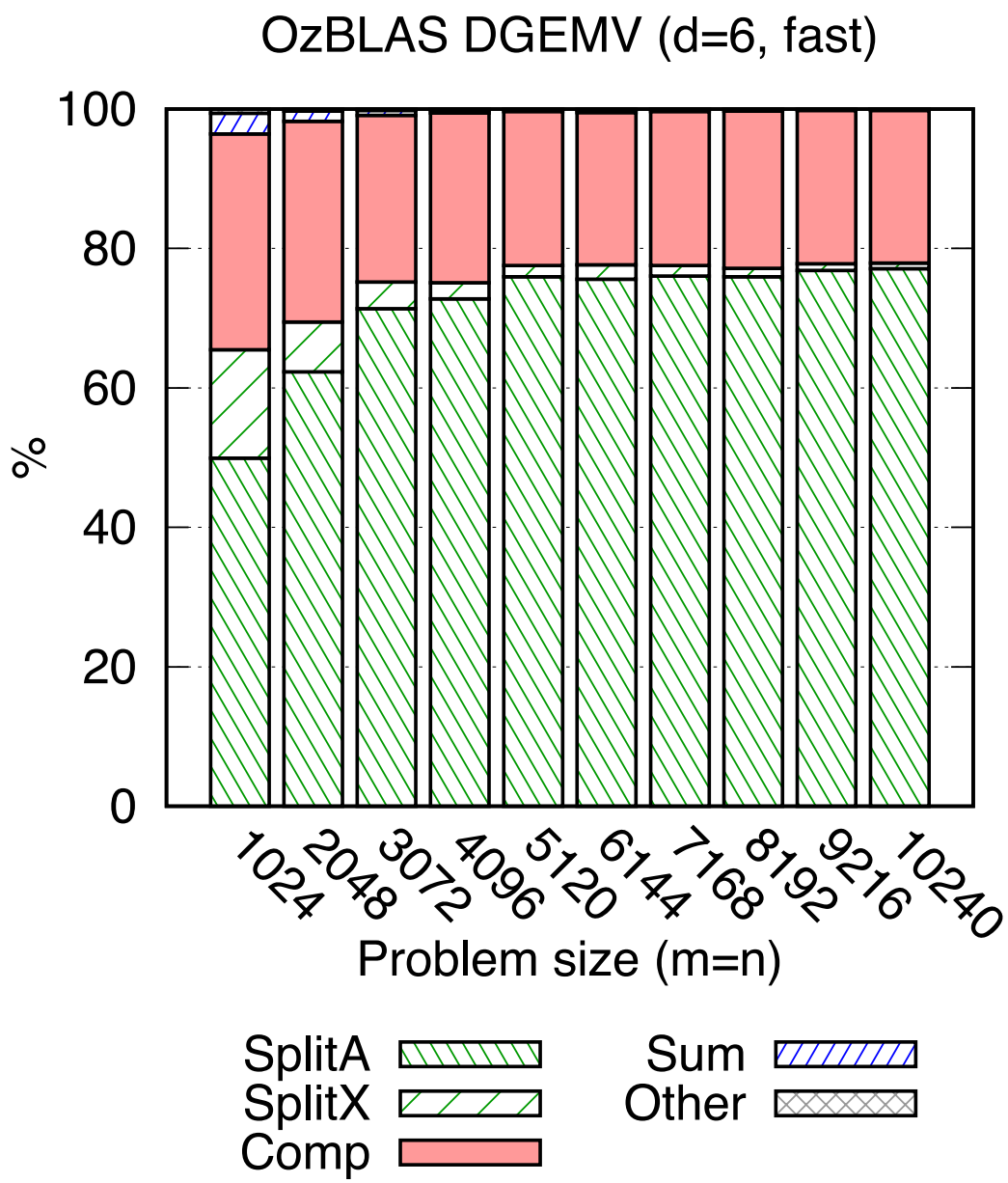
GEMV – スループット（GB/s）



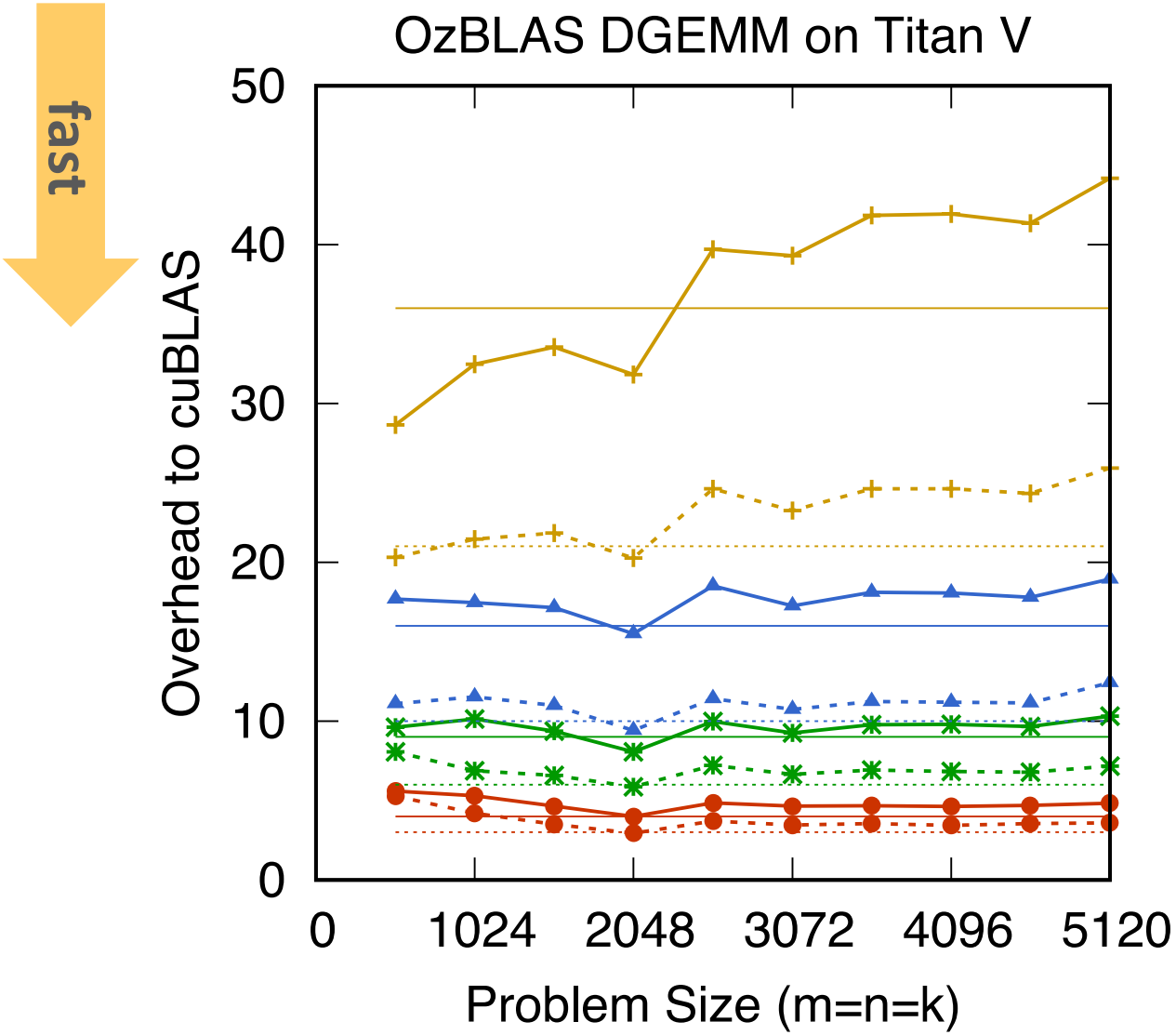


- Input: ϕ=12
- Block-size: b=5120
- GPU: NVIDIA Titan V (Volta architecture, 7449.6 GFlops on DP, 652.8 GB/s 12GB)
- CUDA 9.1 (cuBLAS 9.1), driver version: 390.67

GEMV – 性能内訳


OzBLAS DGEMV (d=6, fast)
100
80
60
40
20
0
%
1024
2048
3072
4096
5120
6144
7168
8192
9216
10240
Problem size (m=n)
SplitA
SplitX
Comp
Sum
Other

GEMM – cublasDgemmに対するオーバーヘッド

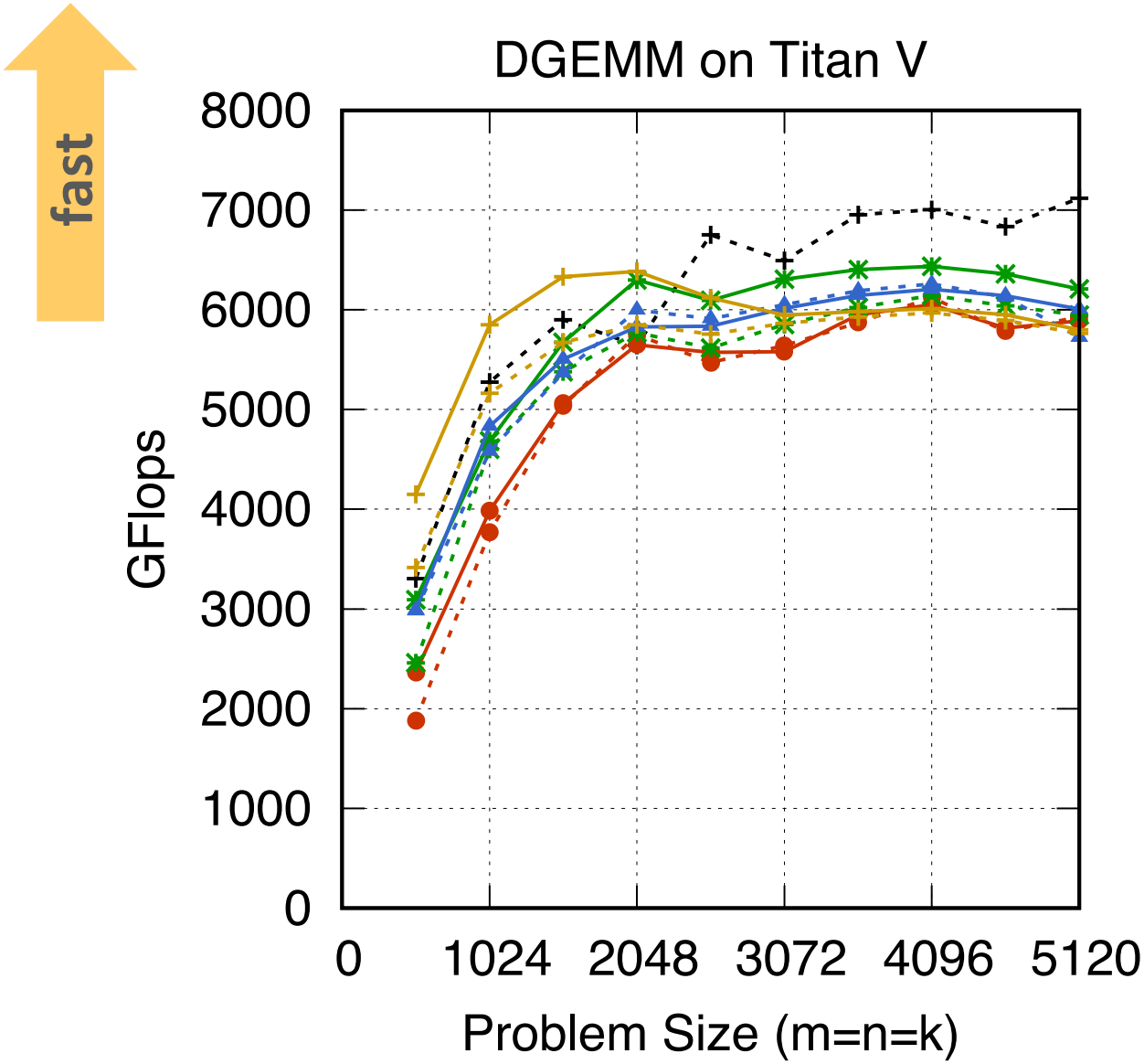


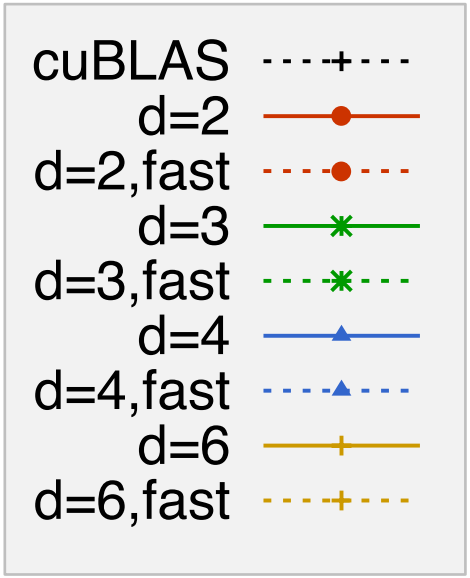
d=2
d=2,fast
d=3
d=3,fast
d=4
d=4,fast
d=6
d=6,fast

Straight lines: theoretical

- Input: ϕ=12
- Block-size: b=2048
- GPU: NVIDIA Titan V (Volta architecture, 7449.6 GFlops on DP, 652.8 GB/s 12GB)
- CUDA 9.1 (cuBLAS 9.1), driver version: 390.67

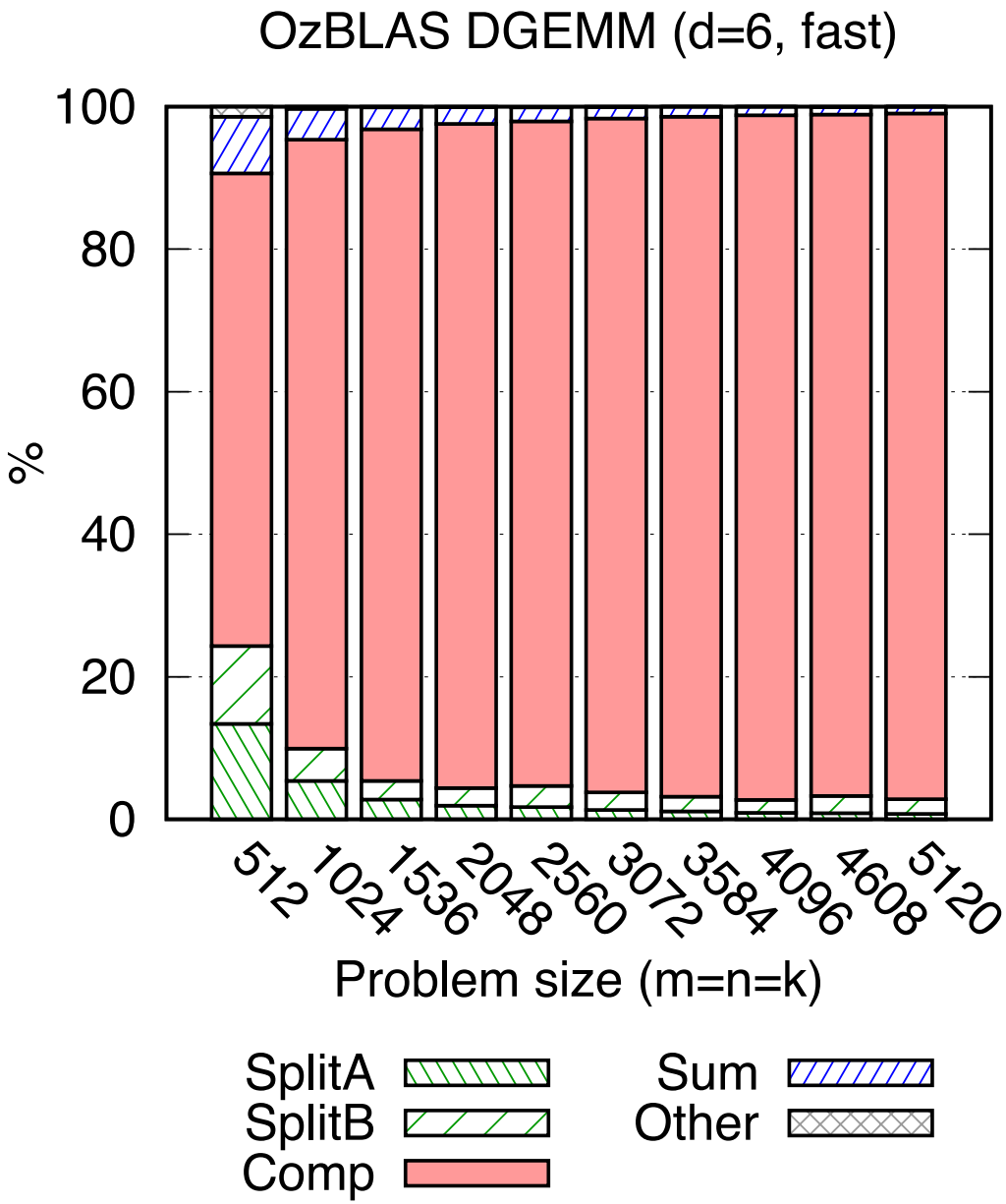
GEMM – スループット（Flops）





- Input: ϕ=12
- Block-size BK=2048
- GPU: NVIDIA Titan V (Volta architecture, 7449.6 GFlops on DP, 652.8 GB/s 12GB)
- CUDA 9.1 (cuBLAS 9.1), driver version: 390.67

GEMM – 性能内訳


OzBLAS DGEMM (d=6, fast)
100
80
60
40
20
0
%
512
1024
1536
2048
2560
3072
3584
4096
4608
5120
Problem size (m=n=k)
SplitA
SplitB
Comp
Sum
Other

実装上の今後の課題

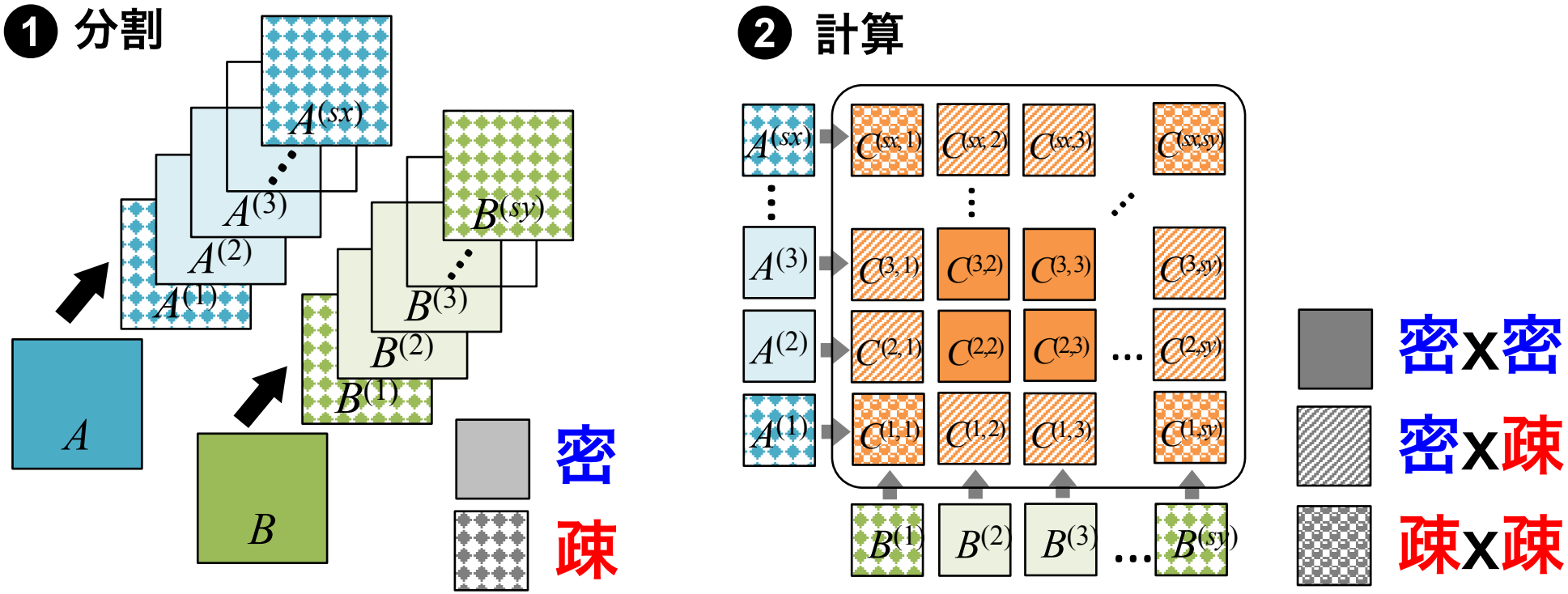
疎行列化と自動チューニングの適用

- **分割行列の疎行列化 [Ozaki et al. 2012] （本研究では未実装）**
  - 入力データの絶対値レンジが大きい場合に分割データはゼロ要素を多く含む可能性があり，ゼロ要素数がある一定以上の場合に疎行列として扱うことで，計算時間を短縮できる可能性がある
  - 自動チューニングの適用可能性 [Ishiguro et al. 2018]
    1. 疎度に応じた自動疎行列化
    2. 疎行列計算のフォーマット自動選択

タスクベースプログラミングの適用

■ 分割行列計算のバッチ・タスク処理

- 分割行列がすべて密ならばbatched GEMMで計算可能であるが，疎行列化を行うと密x密，密x疎，疎x疎が混在する
- この処理をタスクによって効率的に記述・処理できるとよい



今村科研費基盤Bの紹介

- **「エクサ時代の非同期タスクを応用した高性能高次元数値線形代数の研究」（#19H04127，代表者：今村先生，2019年度〜）**
  - 研究途上にある以下のような複雑なタスク処理について線形代数演算における実装を検討する
    1. 非同期タスク方式・バッチ処理とタスク並列モデルの融合
    2. タスクスケジューリングへの静・動的コスト(ヒント)通知機能
    3. 非同期・条件付き・競争的タスクスケジューリング
  - 椋木（分担者）の担当：尾崎スキームの分割行列計算における上記の適用の検討
    - データ（密/疎）に応じた最適な処理（実装）の自動適用
    - 処理内容（負荷）に応じた自動スケジューリング
  - 自動化にあたっては当然自動チューニングの活用を視野に入れる

まとめ

まとめ

■ まとめ

- 尾崎スキームに基づく高精度かつ再現性のあるDOT，GEMV，GEMMのGPU実装
- 尾崎スキームの特徴: (1) 調節可能な計算精度，(2) 計算結果の再現可能性，(3) 計算にlevel-3 BLASを利用可能（実装コストが低い）
- Titan Vにおいて理論性能の約8割以上の性能が得られることを実証

■ 今後の開発計画

- 疎行列化，自動チューニングの適用，他ルーチンの実装，疎行列計算への応用，他手法との比較（科研費若手#19K20286「超並列計算環境のための高精度かつ再現性のある行列計算ライブラリの開発」）
- タスクベースプログラミングの導入（科研費基盤B（代表者：今村先生）#19H04127「エクサ時代の非同期タスクを応用した高性能高次元数値線形代数の研究」）