ScaLAPACKの性能分析と次世代アルゴリズム研究への指針

2016年 11月 29日

計算物質科学における時空間アップスケーリングと数理手法
@電気通信大学

深谷 猛
（北海道大学 情報基盤センター）

本発表の概要

◆ScaLAPACKの現状について

- ✓ ScaLAPACKの概要の紹介
- ✓ 京コンピュータ上での性能事例の紹介

◆通信回避型アルゴリズムの研究事例について

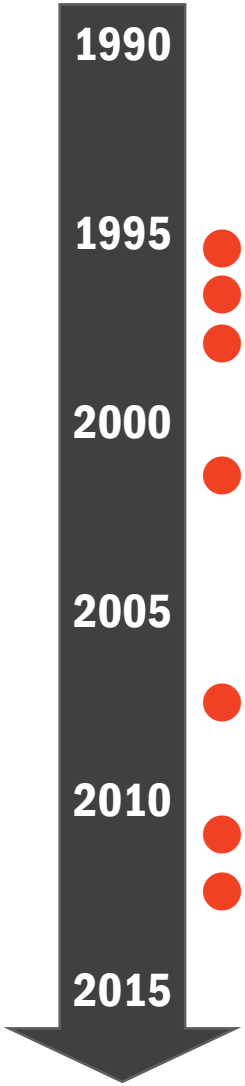
- ✓ 縦長行列のQR分解アルゴリズム
- ✓ TSQRアルゴリズムの長所/短所

ScaLAPACKの概要

ScaLAPACK とは？

- ScaLAPACK：Scalable Linear Algebra PACKage
- 線形計算ライブラリ（LAPACK：Linear Algebra PACKage）の分散並列計算機向けバージョン
- 入手・利用方法：
  - ✓ netlibよりフリーで入手可能
  - ✓ 多くの環境ではベンダーにより提供
- LAPACKのルーチンに準拠（LAPACKのルーチン名の頭にP）
  ※ただし，LAPACKのルーチンの一部のみ提供（後述）

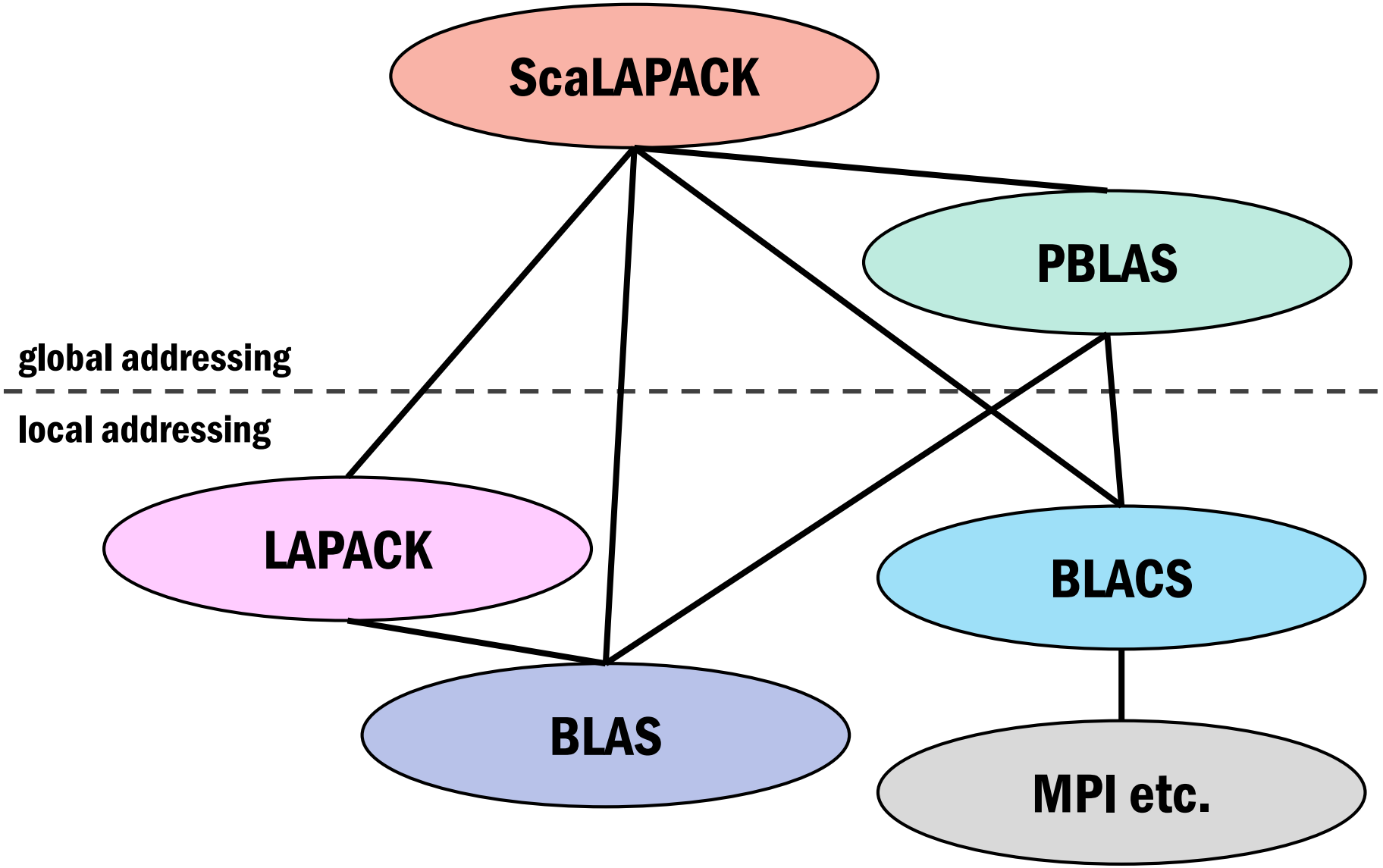
ScaLAPACKのRelease History



- ScaLAPACK ver. 1.0 (1995/2/28)
- ScaLAPACK ver. 1.4 (1996/11/17)
- ScaLAPACK ver. 1.6 (1997/11/15)
- ScaLAPACK ver. 1.7 (2001/8/31)
- ScaLAPACK ver. 1.8.0 (2007/4/5)
- ScaLAPACK ver. 2.0.0 (2011/1/18)
- ScaLAPACK ver. 2.0.2 (2012/5/1)

- LAPACK ver. 1.0 (1992/2/29)
- LAPACK ver. 2.0 (1994/9/30)
- LAPACK ver. 3.0 (1999/6/30)
- LAPACK ver. 3.1.0 (2006/11/12)
- LAPACK ver. 3.2.0 (2008/11/18)
- LAPACK ver. 3.3.0 (2010/11/14)
- LAPACK ver. 3.4.0 (2011/11/11)
- LAPACK ver. 3.5.0 (2013/11/19)
- LAPACK ver. 3.6.0 (2015/11/15)
- LAPACK ver. 3.6.1 (2016/6/18)

ScaLAPACKの階層構造



ScaLAPACKが提供するアルゴリズム

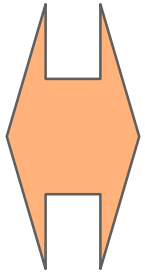
Problem	LAPACK	ScaLAPACK
Linear Equation	GESV (LU)	PxGESV
	POSV (Cholesky)	PxPOSV
	SYSV (LDL^T)	missing
Least Squares	GELS (QR)	PxGELS
	GELSY (QR with pivoting)	missing driver
Symmetric EVD	SYEV (inverse iteration)	PxSYEV
	SYEVD (D&C)	missing
	SYEVR (RRR)	missing
Nonsymmetric EVD	GEES (HQR)	missing driver
	GEEV (HQR + vectors)	missing driver
SVD	GESVD (QR)	PxGESVD
	GESDD (D&C)	missing

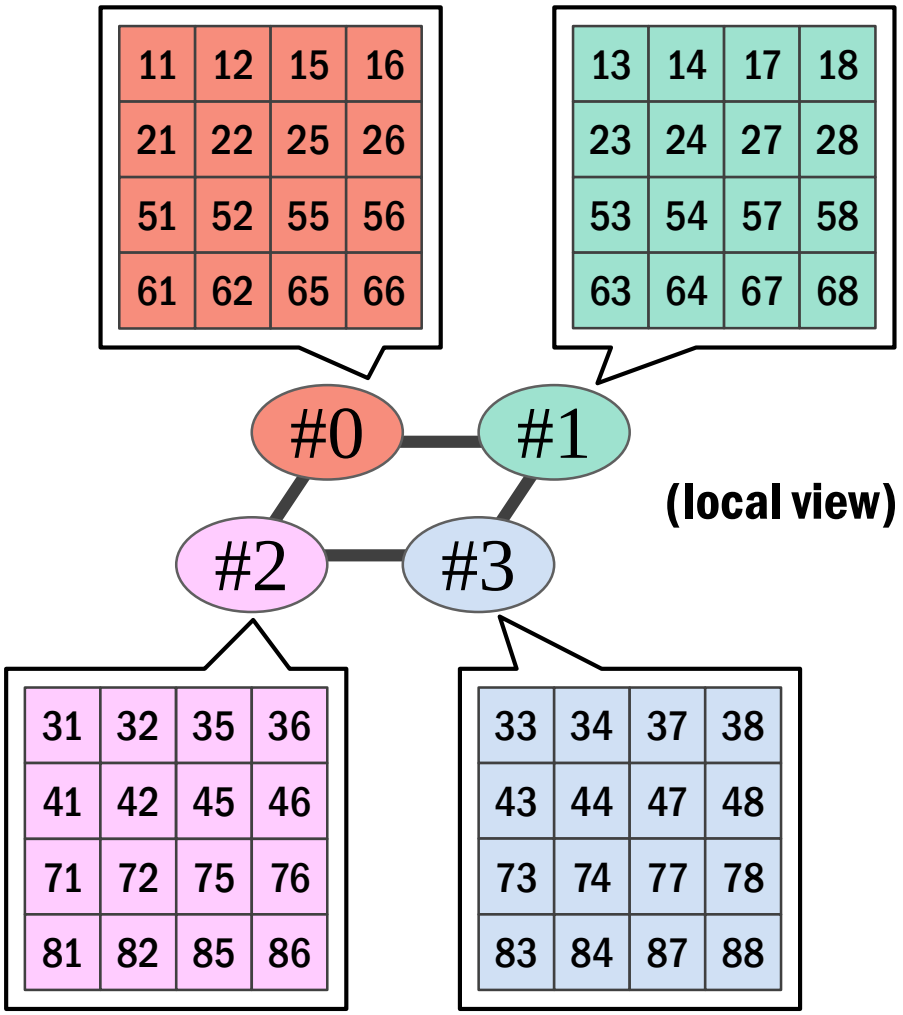
ScaLAPACKにおけるデータ分散方式

2次元ブロックサイクリック分散が基本

11	12	13	14	15	16	17	18
21	22	23	24	25	26	27	28
31	32	33	34	35	36	37	38
41	42	43	44	45	46	47	48
51	52	53	54	55	56	57	58
61	62	63	64	65	66	67	68
71	72	73	74	75	76	77	78
81	82	83	84	85	86	87	88

(global view)





(local view)

ScaLAPACKのインターフェースの例

```
subroutine PDGERQF  (
  INTEGER  M,
  INTEGER  N,
  DOUBLE PRECISION, dimension( * )  A,
  INTEGER  IA,
  INTEGER  JA,
  INTEGER, dimension( * )  DESCA,
  DOUBLE PRECISION, dimension( * )  TAU,
  DOUBLE PRECISION, dimension( * )  WORK,
  INTEGER  LWORK,
  INTEGER  INFO
)
```

参考 : subroutine dgeqrf (M, N, A, LDA, TAU, WORK, LWORK, INFO)

ScaLAPACKに対するユーザの反応

2016 Dense Linear Algebra Software Package Surveyより
（2016年1月～9月，252人が回答，詳細はLAWN 290）

- ScaLAPACK：80人が回答（LAPACK：186人）
- ScaLAPACKのインターフェースは使い難い：52%
- ScaLAPACKを検討したが利用しなかった理由：
  - ✓ データ分散が分かりにくい
  - ✓ 他のライブラリ（Elemental, EigenExa, ELPA）が良い
- 改良を希望する点：
  - ✓ 作業用メモリの自動確保
  - ✓ アルゴリズムの自動選択
  - ✓ 適切なデータ構造への自動変換

ScaLAPACKの今後について

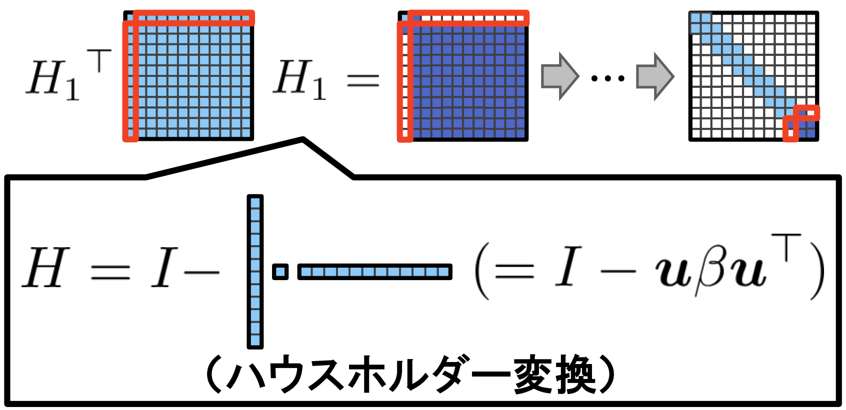
Dr. Osni Marques (LBNL)の話によると・・・

- DemmelとDongarraが中心となってLAPACK/ScaLAPACK等の線形計算ライブラリ開発のための研究費を申請中らしい.
- 研究費が獲得できた場合：
  - ✓ 通信回避型のアルゴリズムの組み込み等の改良
  - ✓ SC17で新しいバージョンをリリース予定
- 研究費が獲得できなかった場合：
  - ✓ 開発の継続は未定？

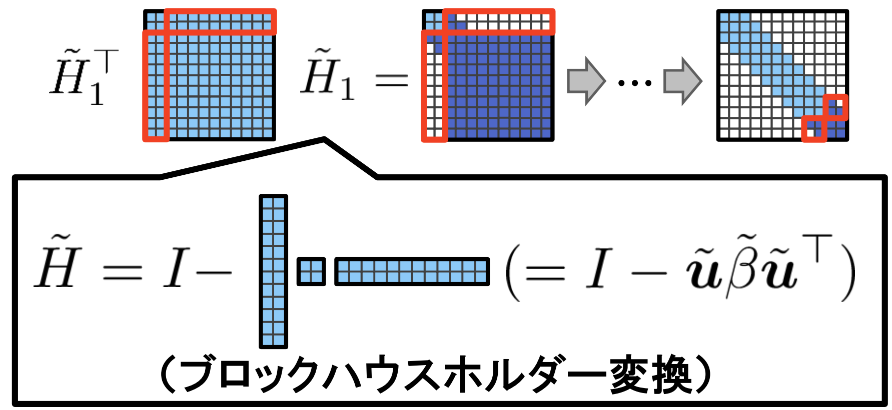
ScaLAPACKの性能事例（環境：京コンピュータ）

事例1：三重対角化/五重対角化

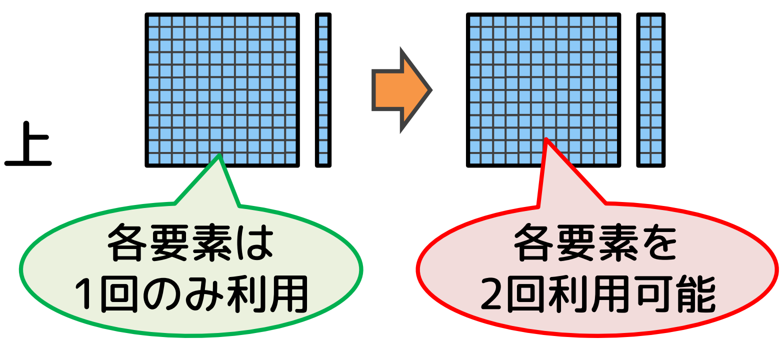
三重対角化



五重対角化

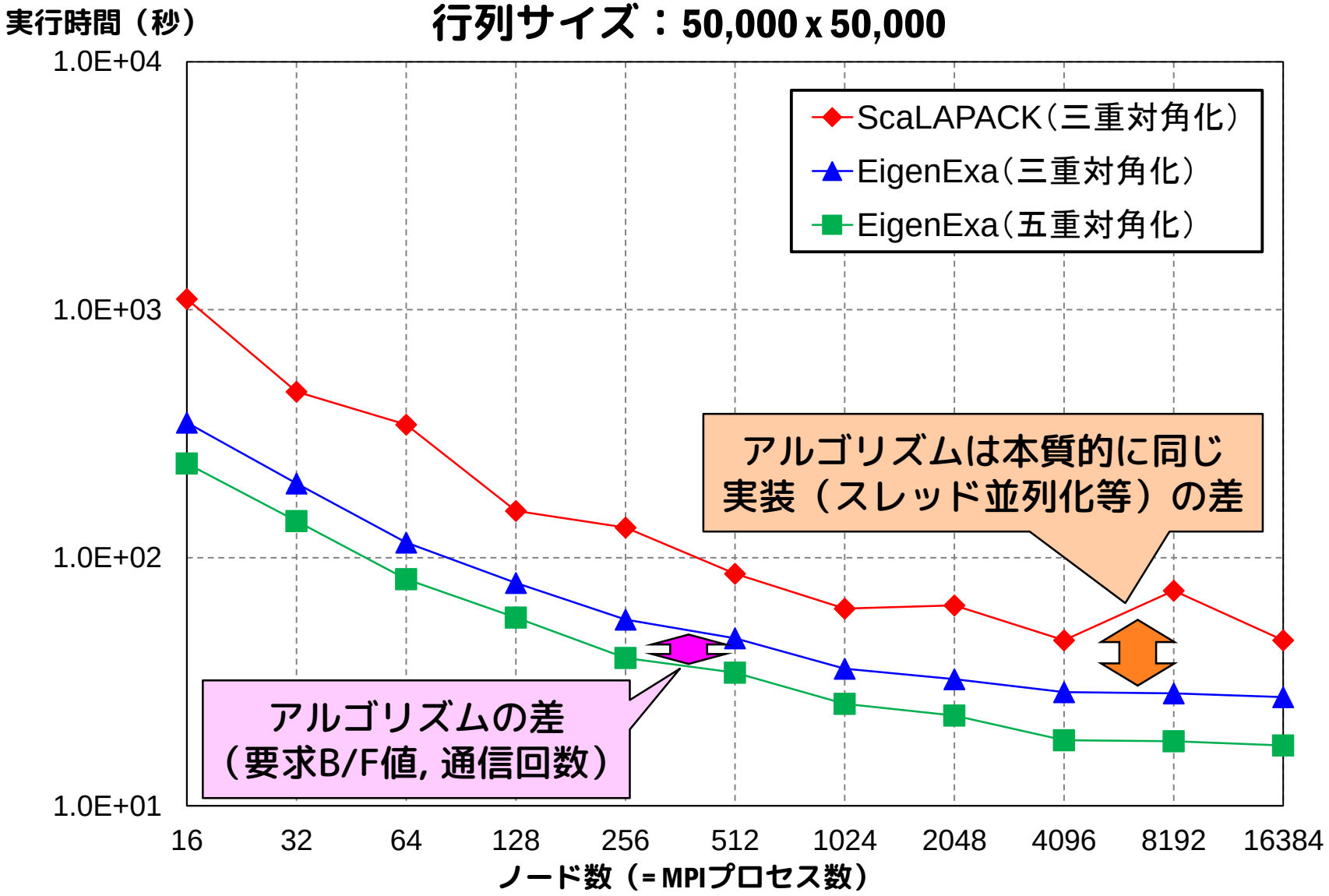


- 要求 byte/flop 値が削減
  - 行列ベクトル積のデータ再利用性の向上
  - 全体の演算量自体は変わらない

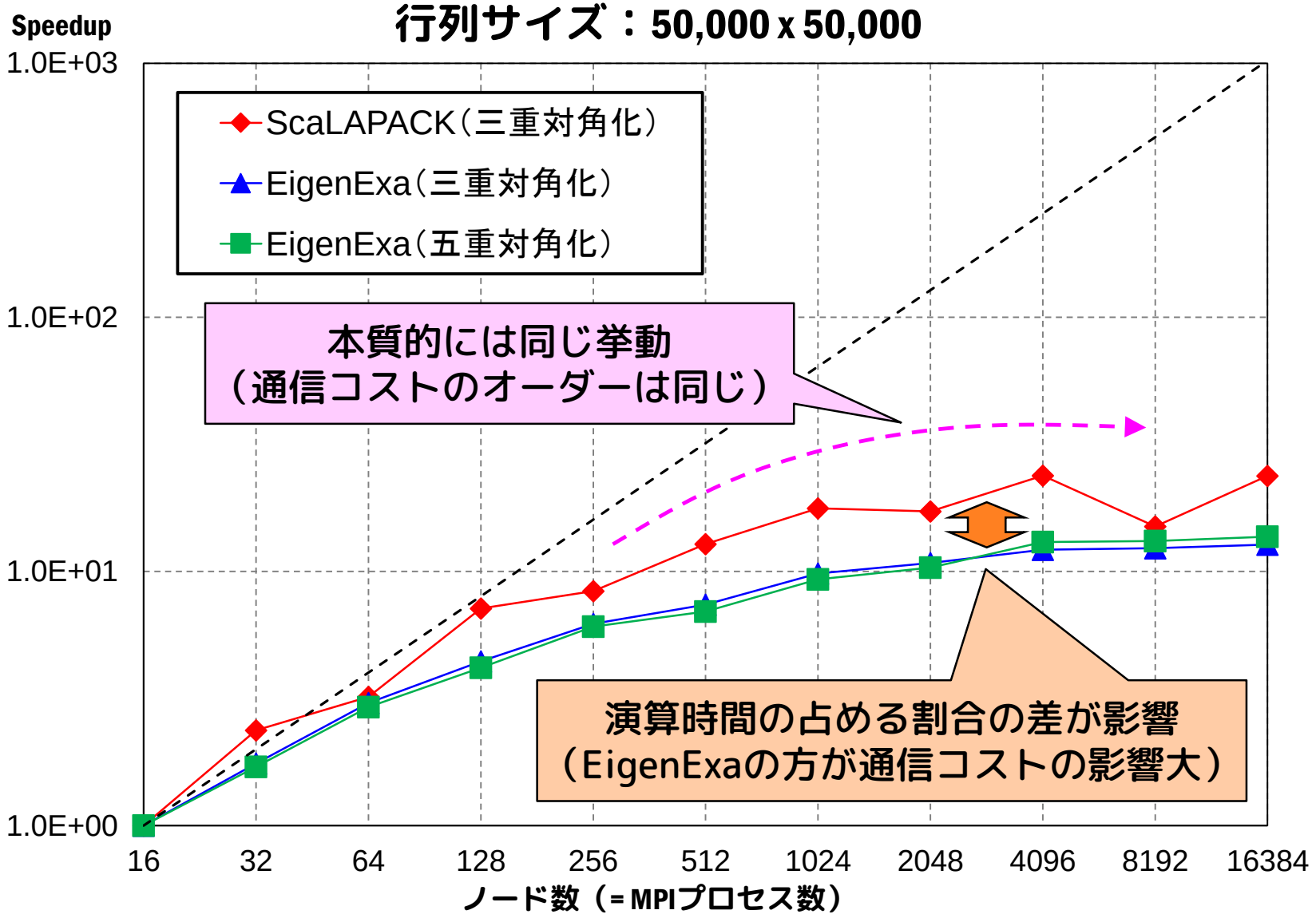


- 通信回数が約半分に削減
  - 三重対角化の通信回数： $O(N)$　※一列単位で処理
  - 五重対角化の通信回数： $O(N/2)$　※二列単位で処理
  - 通信するデータ量の合計は同じ　※1回の通信データ量が増加

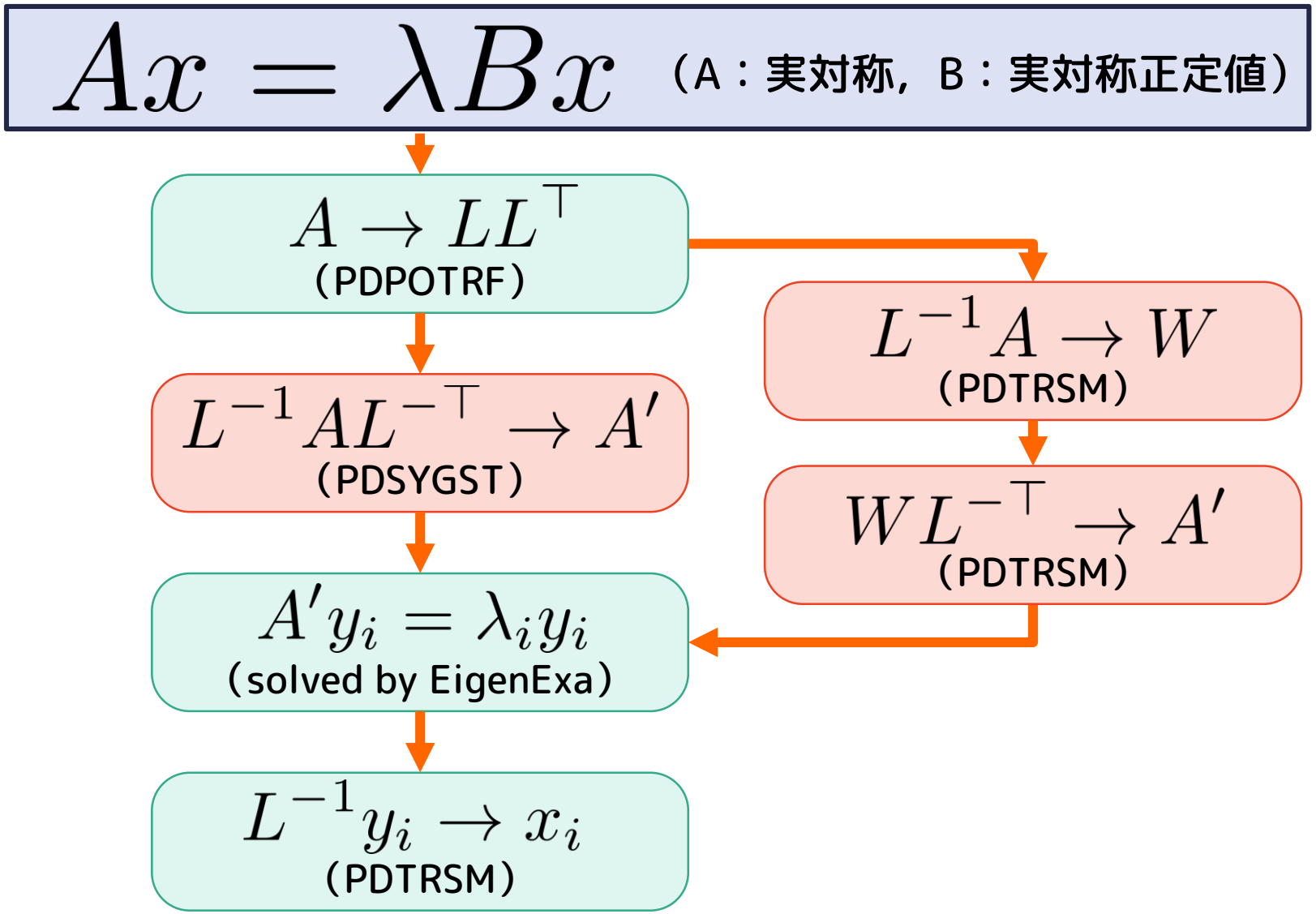
三重（五重）対角化の実行時間



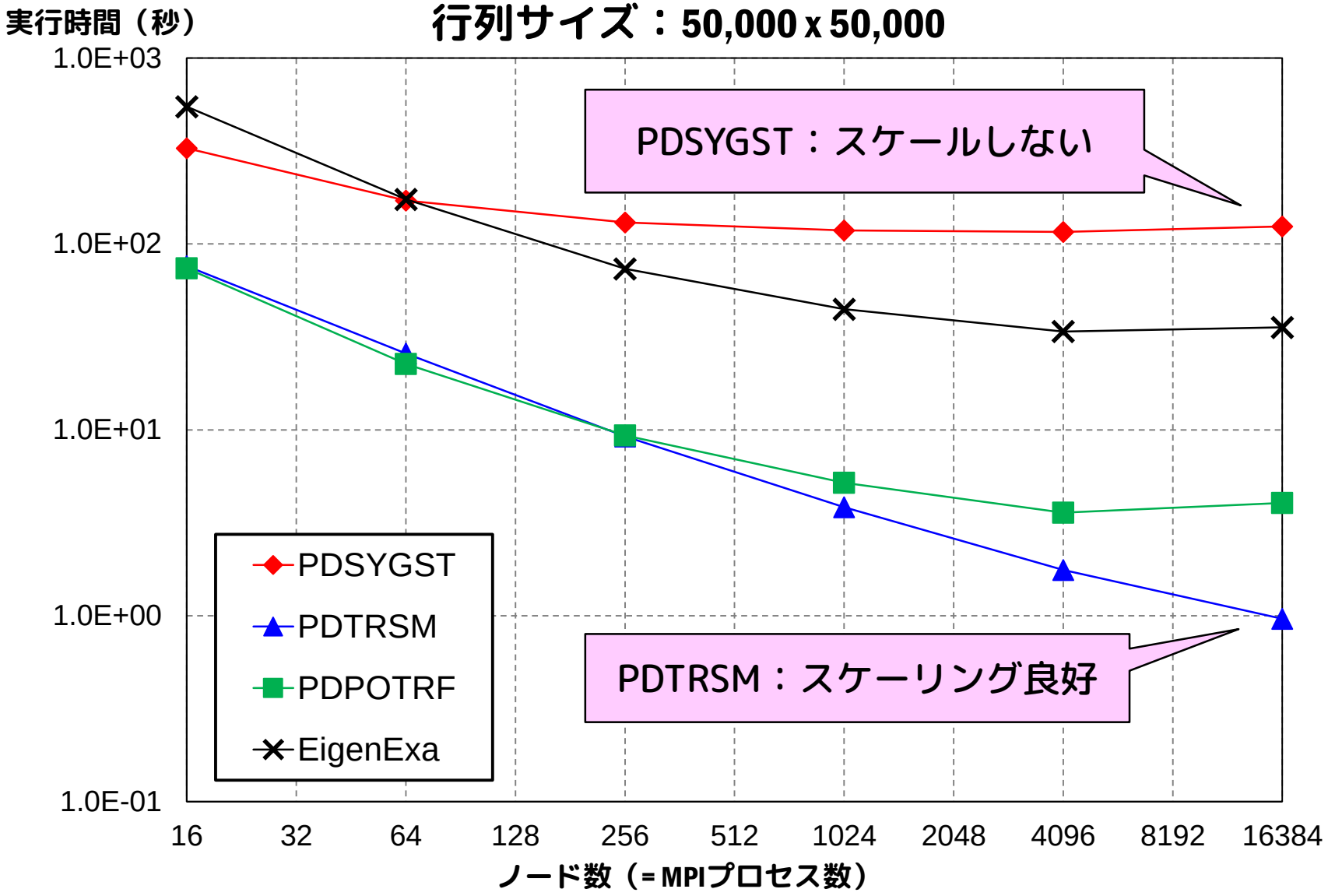
三重（五重）対角化のSpeedup



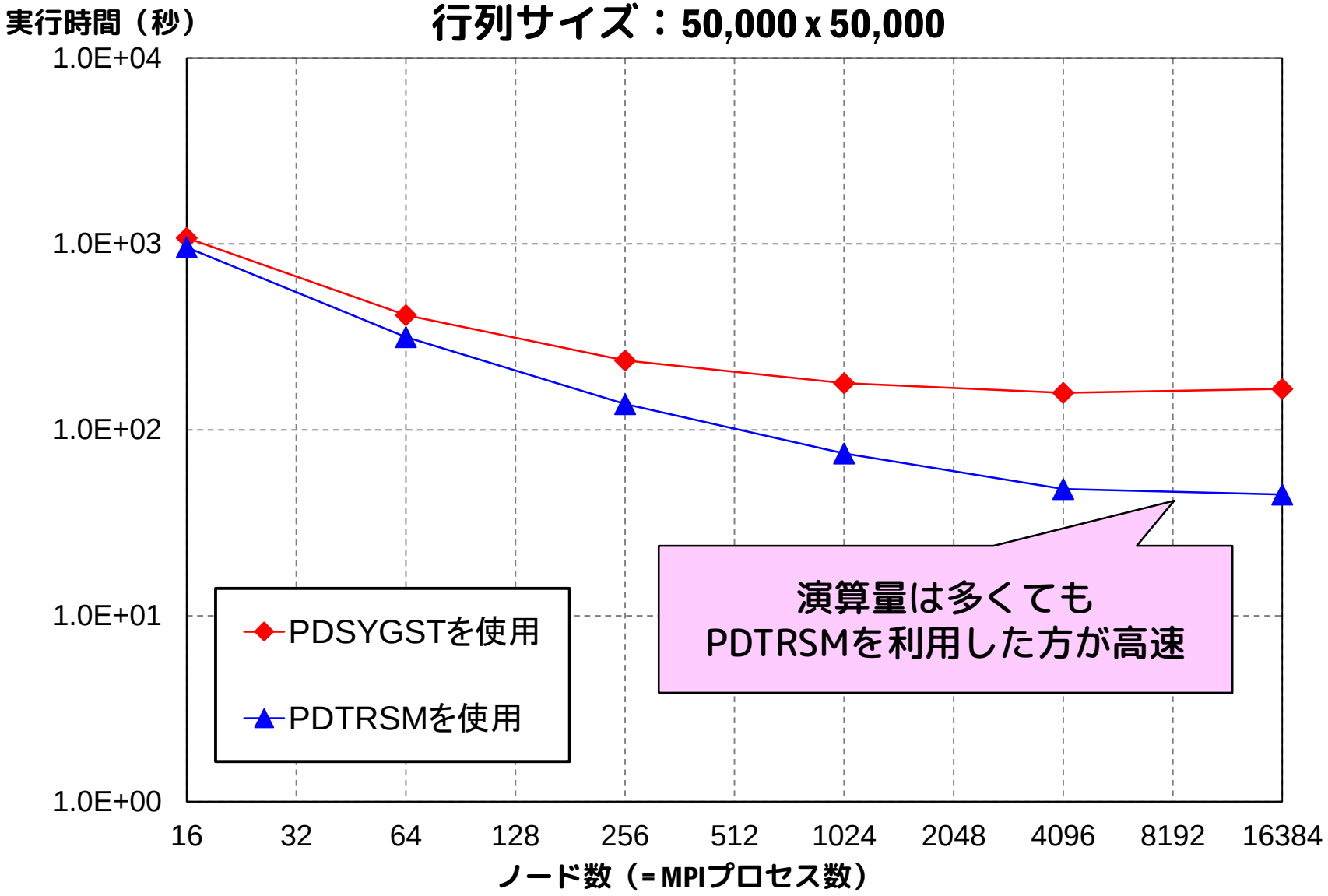
事例2：一般化固有値問題の変換ルーチン



各ルーチンのスケーリングの様子



変換方法の比較


実行時間（秒）
行列サイズ：50,000 x 50,000
1.0E+04
1.0E+03
1.0E+02
1.0E+01
1.0E+00
16
32
64
128
256
512
1024
2048
4096
8192
16384
ノード数（= MPIプロセス数）
PDSYGSTを使用
PDTRSMを使用
演算量は多くても
PDTRSMを利用した方が高速

ScaLAPACKに関するまとめ

- （性能をあまり気にしなければ）自分でゼロから実装する場合よりもコストが非常に小さい.
- 並列数が少ない or 問題サイズが大きい場合は，それなりに良くスケールする？
- ほとんど（全く）スケールしないルーチンもある.
- アプリケーションの内部で使用する場合，データ分散の方式が合致するかどうか.
  （分散方式の変更ルーチンの実装は相応の手間となる）
- 性能を突き詰める（特に並列数が多い）場合には，ScaLAPACKの利用は得策ではない.
- 他の選択肢：Elemental, FLAME, DPLASMA?

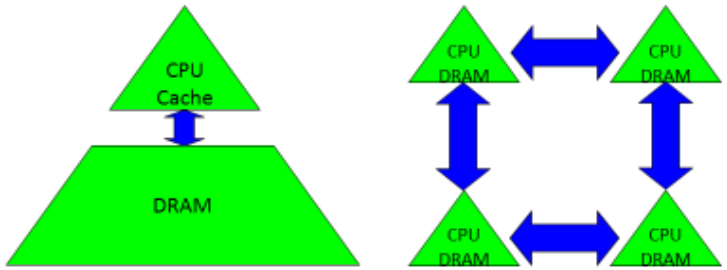
通信回避型アルゴリズムの研究
（QR分解を例にして）

通信とは？

Why avoid communication? (1/2)

Algorithms have two costs (measured in time or energy):

1. Arithmetic (FLOPS)
2. Communication: moving data between
   - levels of a memory hierarchy (sequential case)
   - processors over a network (parallel case).



2

Why avoid communication? (2/3)

- Running time of an algorithm is sum of 3 terms:
  - # flops * time_per_flop
  - # words moved / bandwidth
  - # messages * latency

  } communication
- Time_per_flop << 1/ bandwidth << latency
  - Gaps growing exponentially with time [FOSC]

Annual improvements			
Time_per_flop		Bandwidth	Latency
59%	Network	26%	15%
	DRAM	23%	5%

- Avoid communication to save time

3

[slides of SC14 tutorial by J. Demmel]

実行時間 = 演算時間 + データ移動の時間

通信（Communication）

※本発表では，ノード間の通信（MPIによるデータ転送）に着目

通信コストの具体例（@京コンピュータ）

$$T_{\mathrm{comm.}} = \alpha + \beta \cdot w$$

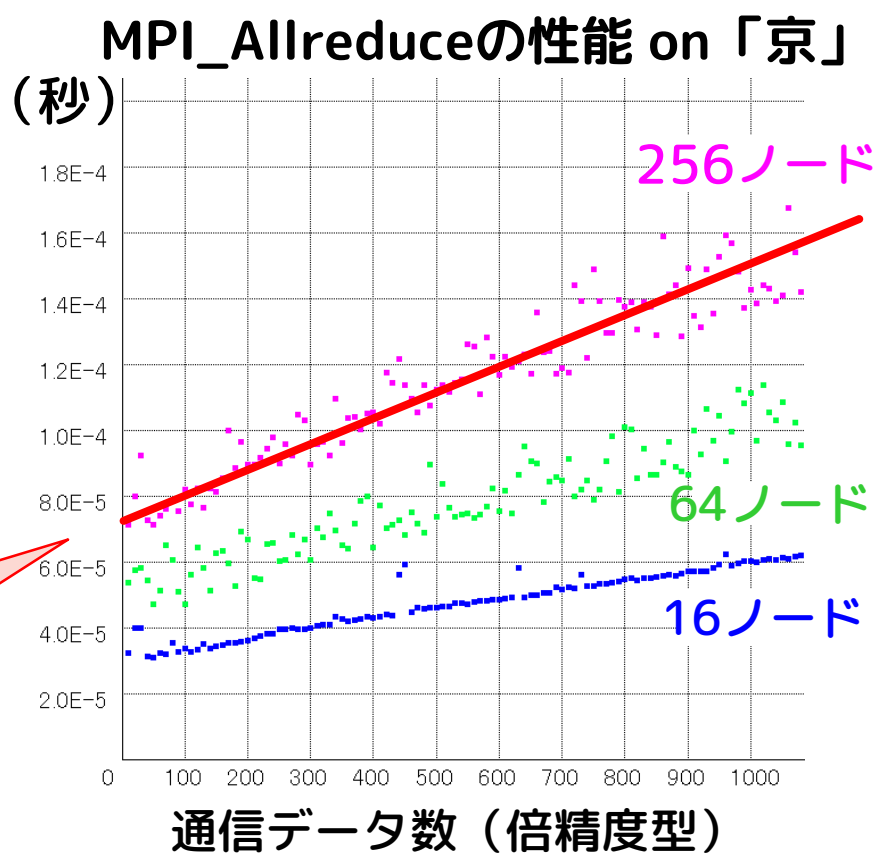
- α ：セットアップコスト
- β ：バンド幅の逆数
- w ：通信データ数

※ α, βはノード数や環境に依存

$\alpha = 7.5 \times 10^{-5}$ (秒)

$\beta = 7.6 \times 10^{-8}$ (秒/個)

※10^{-9}～10^{-11} (秒/演算)



通信のレイテンシ（セットアップコスト）の削減が最も重要

⇒ 通信回数の削減（CA：Communication Avoiding）が重要！

ハードウェアの技術動向

item	FX10	FX100	improvement
peak FLOPS / node (GFLOPS)	236.5	1100	x 4.7
peak memory bandwidth / node (GB/sec)	85	240	x 2.8
network bandwidth (GB/sec)	5.0	12.5	x 2.5
network latency (µsec)	0.92	0.71	x 1.3

References:

- T. Yoshida, “SPARC64 XIfx: Fujitu’s next generation processor for HPC”, HC26, 2014.
- N. Shida, et al., “MPI library and low-Level communication on the k computer”, Open MPI Publications, 2012.
- Y. Ajima, et al., “The tofu interconnect 2”, Hot Interconnect 22, 2014.

ネットワークのレイテンシの向上は容易ではない

研究動向

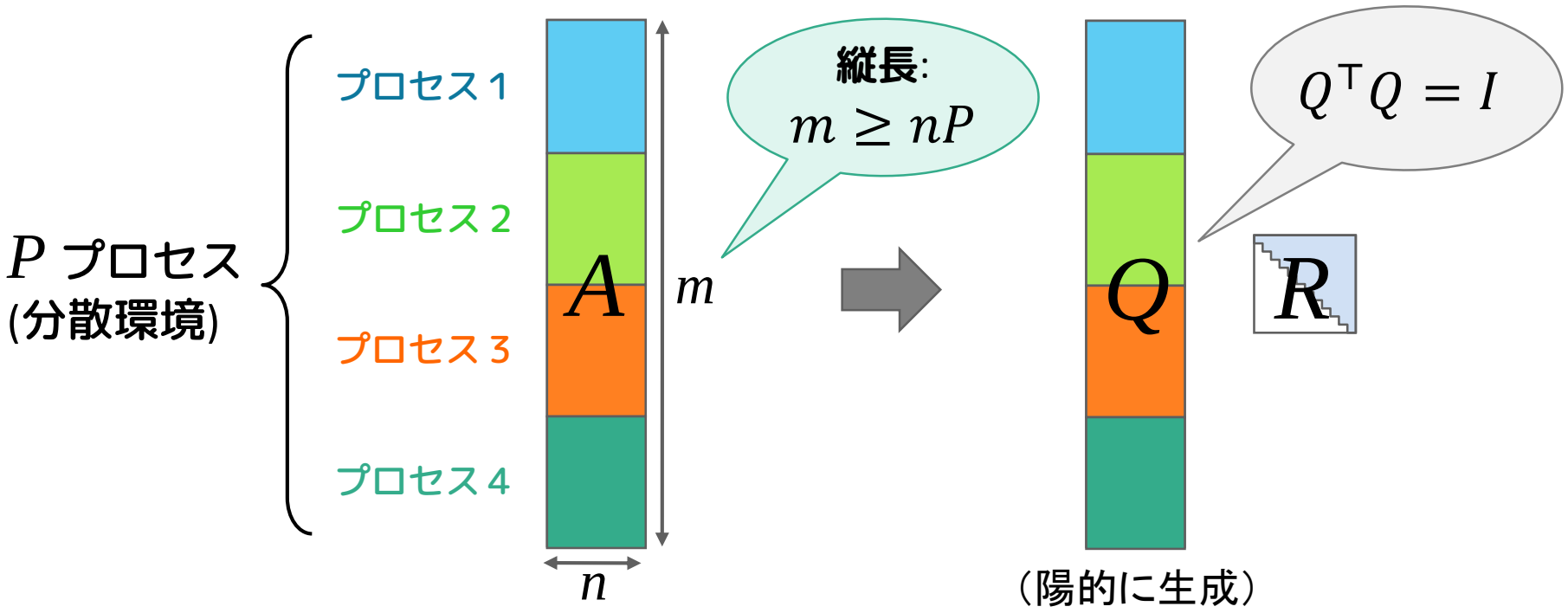
◆ 主な研究事例

- QR分解：J. Langou (2007), J. Demmel, et al. (2012)
- LU分解：L. Grigori, et al. (2011)
- Strassen：B. Lipshitz, et al. (2012)
- Krylov部分空間法：M. Hoemmen (2010), R. Suda, et al. (2013), E. Carson, et al. (2014)
- 固有値計算：G. Ballard, et al. (2015)
- その他（通信回数の下界など）：G. Ballard, et al. (2011)

◆ 補足

- アルゴリズムの根本的なアイディア自体はもっと古い場合が多い
  （最近の成果：系統化，有効性の検証など）
- 必ずしも実機上で性能評価している訳ではない（モデルで代替等）
- 関連：同期削減（Synchronous avoiding），タイルアルゴリズム

問題設定：縦長行列のQR分解



◆ 縦長行列のQR分解の応用例

- 部分空間射影法（のブロック版）におけるベクトルの直交化
- 長方行列の特異値分解の前処理
- 行列の帯行列化のための直交変換の生成

例えば,
m ：百万以上
n ：数十~数百

従来法（非通信回避型）

◆ Gram-Schmidt (CGS)

$$\boldsymbol{q}_1 := \frac{1}{\|\boldsymbol{a}_1\|}\boldsymbol{a}_1$$

comm.

do k=2, n

$$\boldsymbol{a}_k := \left(I - Q_{k-1}Q_{k-1}^{\top}\right)\boldsymbol{a}_k$$

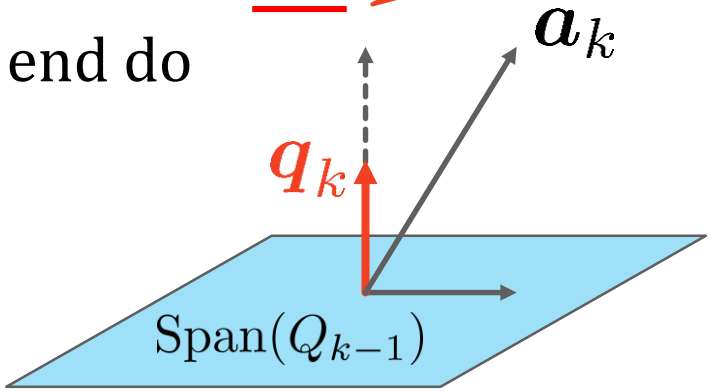
comm.

※$Q_{k-1} := [\boldsymbol{q}_1 \cdots \boldsymbol{q}_{k-1}]$

$$\boldsymbol{q}_k := \frac{1}{\|\boldsymbol{a}_k\|}\boldsymbol{a}_k$$

comm.

end do



◆ Householder QR (ScaLAPACK)

do k=1, n

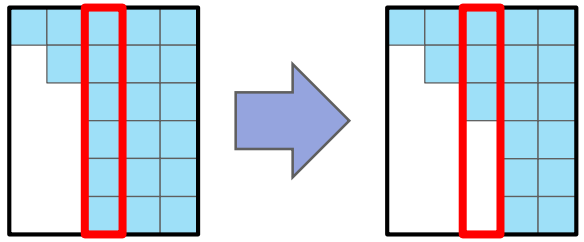
$$[t_k, \boldsymbol{y}_k] := \mathrm{house}(\boldsymbol{a}_k)$$

comm.

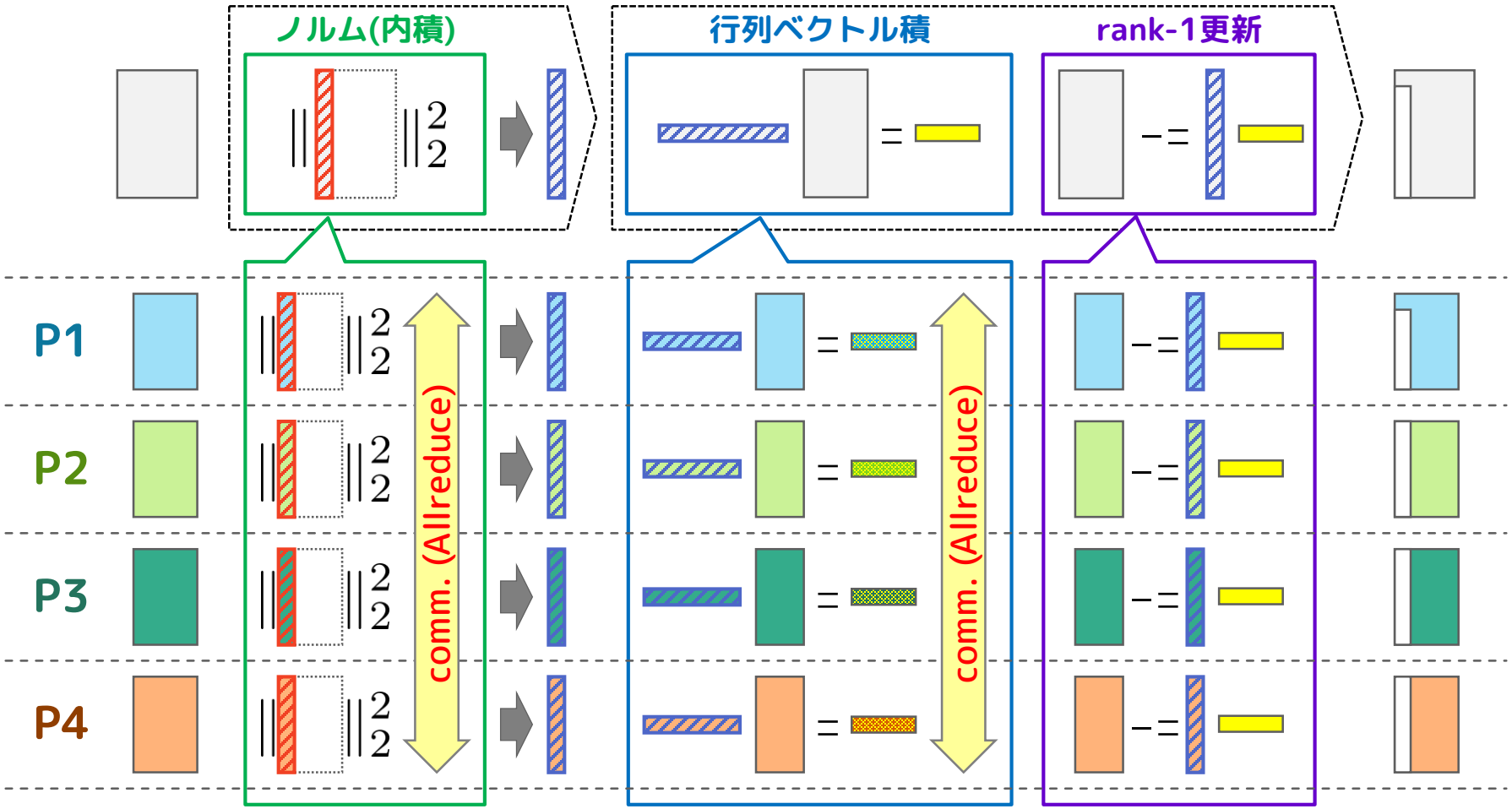
$$A := \left(I - t_k \boldsymbol{y}_k \boldsymbol{y}_k^{\top}\right)A$$

comm.

end do

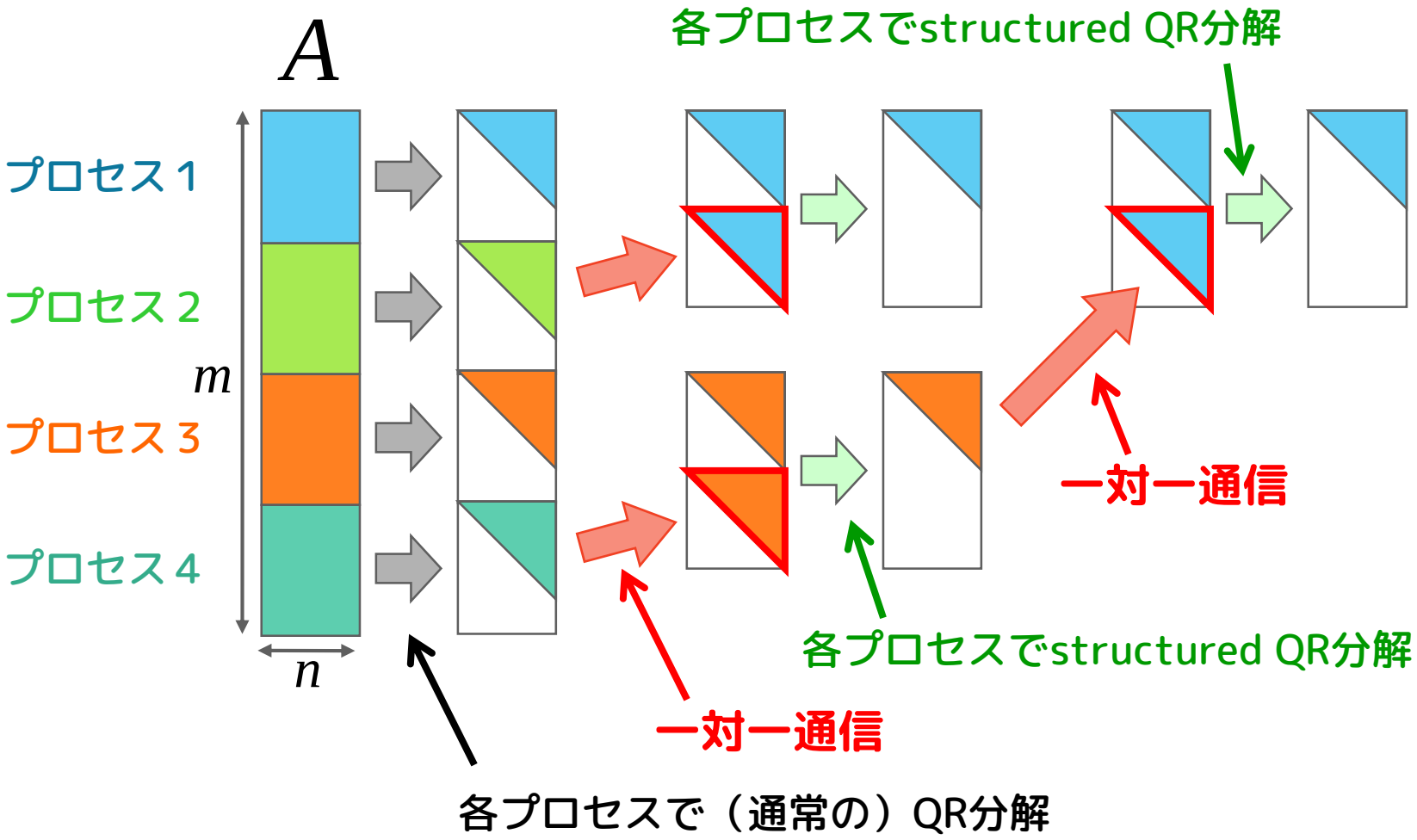


Householder QRの分散並列化



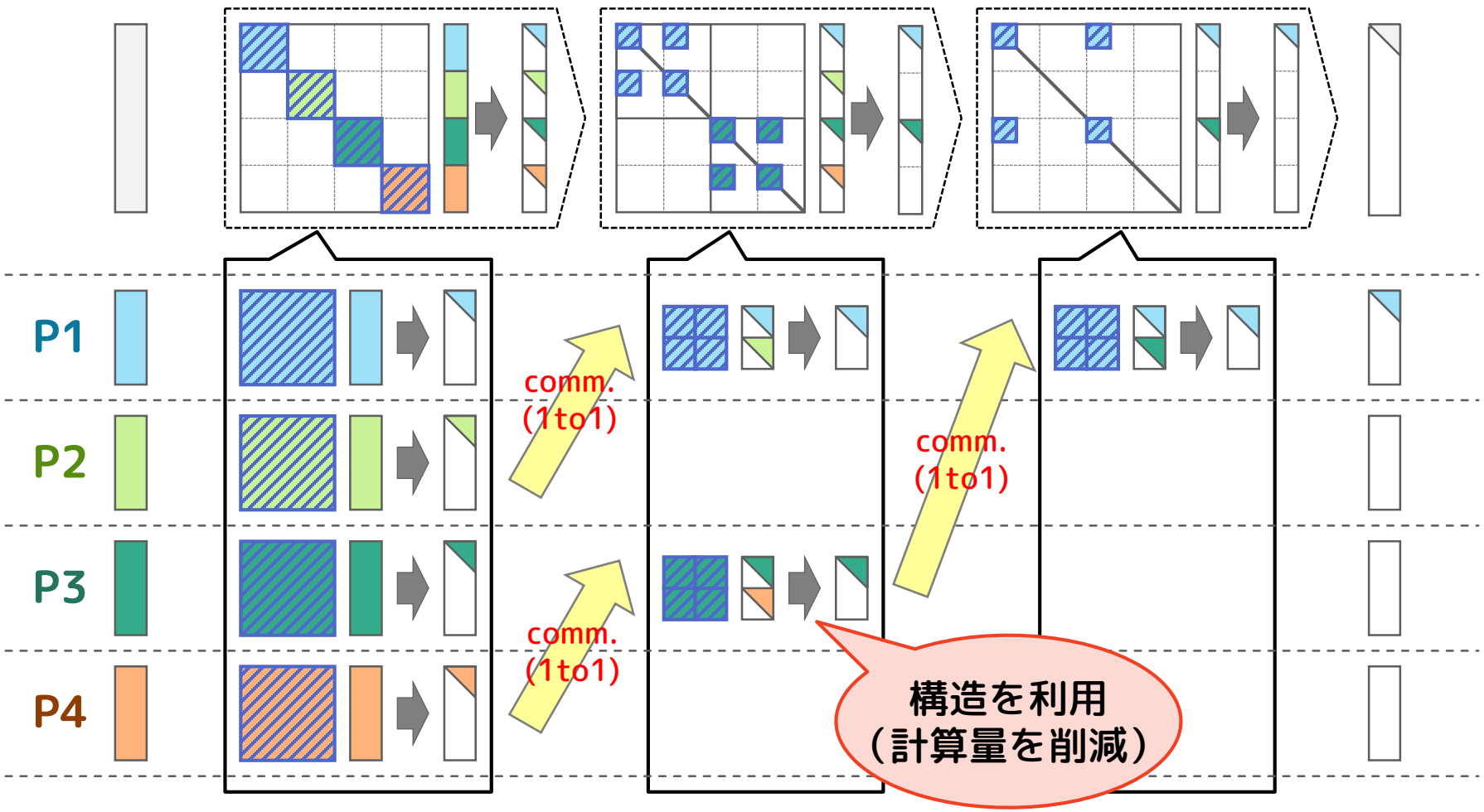
一列の消去ごとに2回(全体で2n回)の集団通信が必要

TSQRアルゴリズム [J. Demmel, et al., 2012]



通信回数 = $\log_2 P$回の一対一通信 ≒ 1回の集団通信

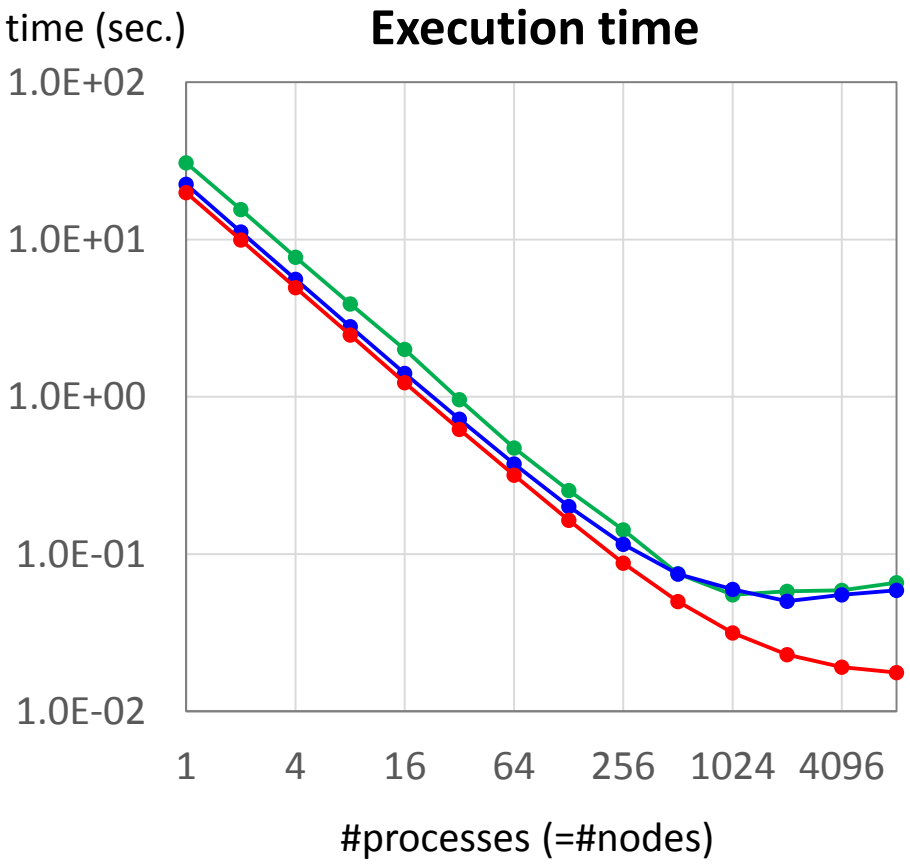
TSQRの分散並列化

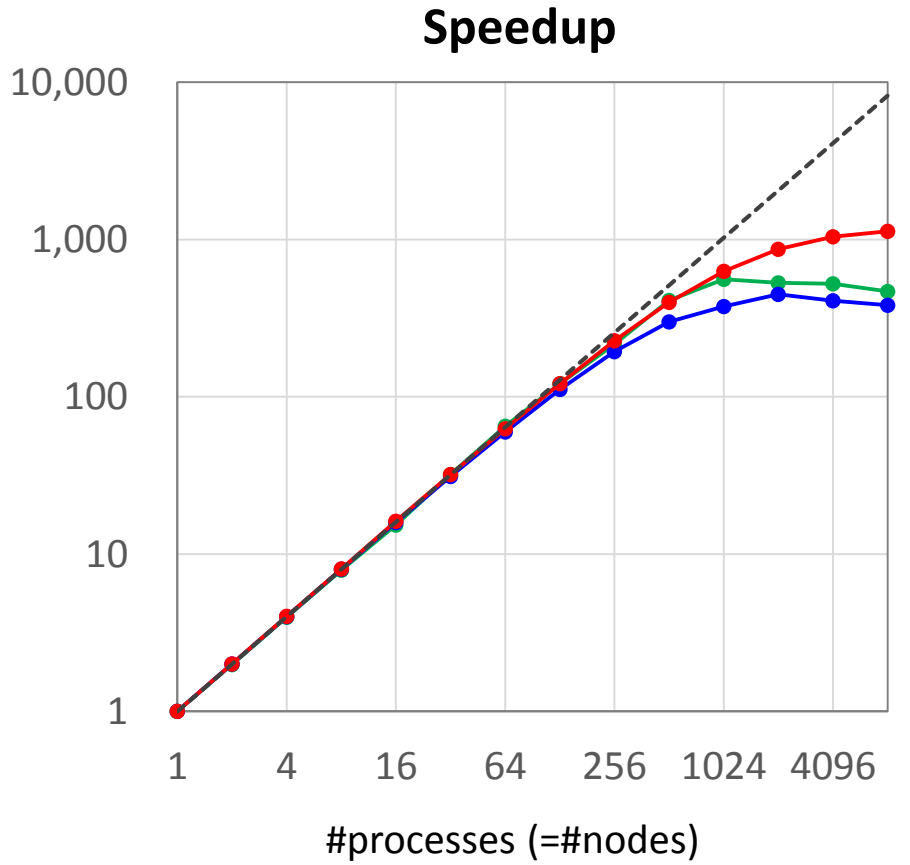


$\log_2$(並列数)回程度の一対一通信で済む

性能例（京，サイズ：5,000,000 x 100）

-●-:Householder QR(non-block) -●-:Householder QR(recursive QR) -●-:TSQR





TSQRの性能モデリング

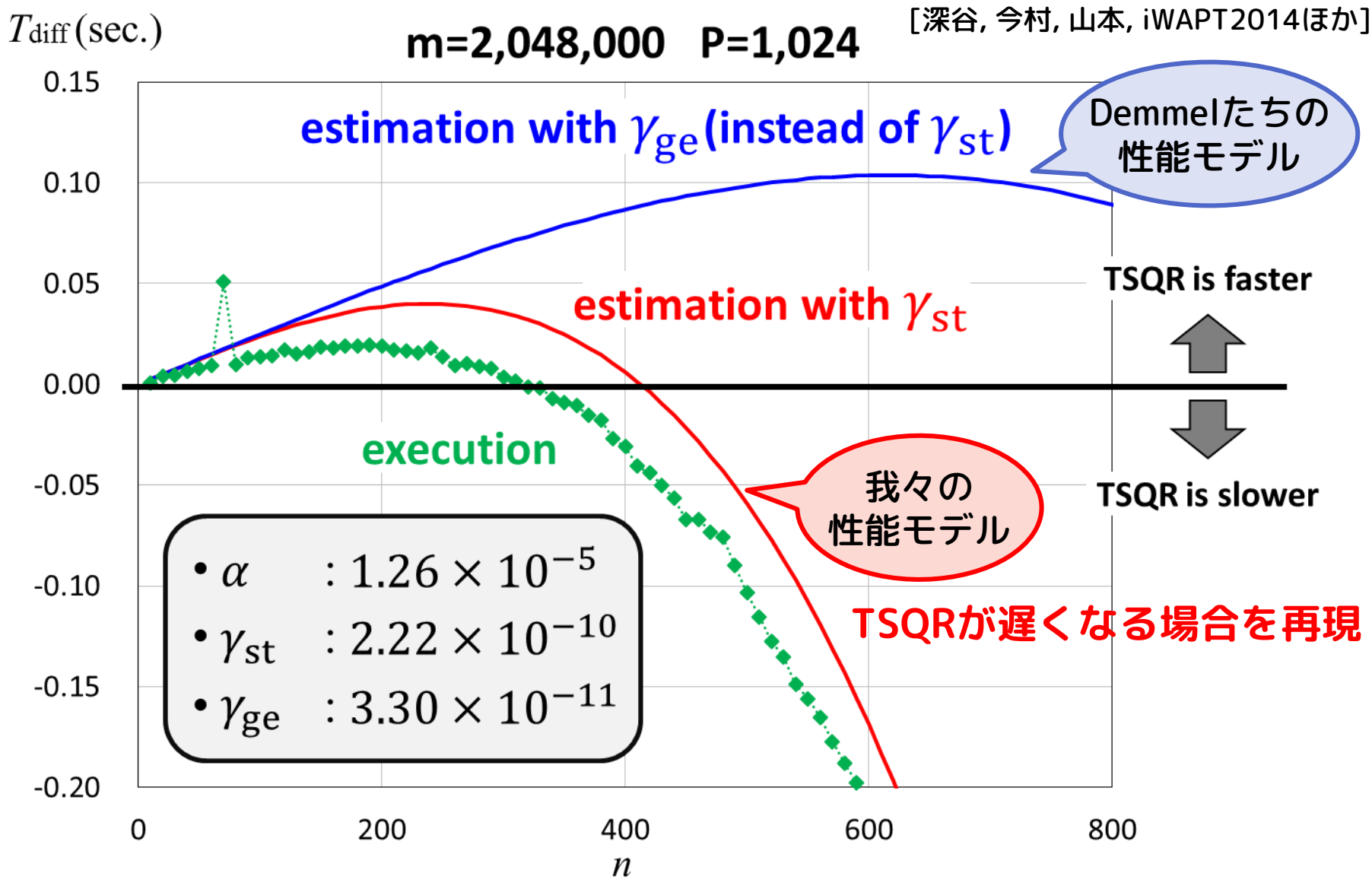
$$T_{\mathrm{ex}} = \gamma_{\mathrm{geqr}} \cdot (\#flops\_geqr) + \gamma_{\mathrm{stqr}} \cdot (\#flops\_stqr) + \alpha \cdot (\#msgs) + \beta \cdot (\#words)$$

演算カーネルごとにモデルパラメータ（浮動小数点演算のコスト）を区別

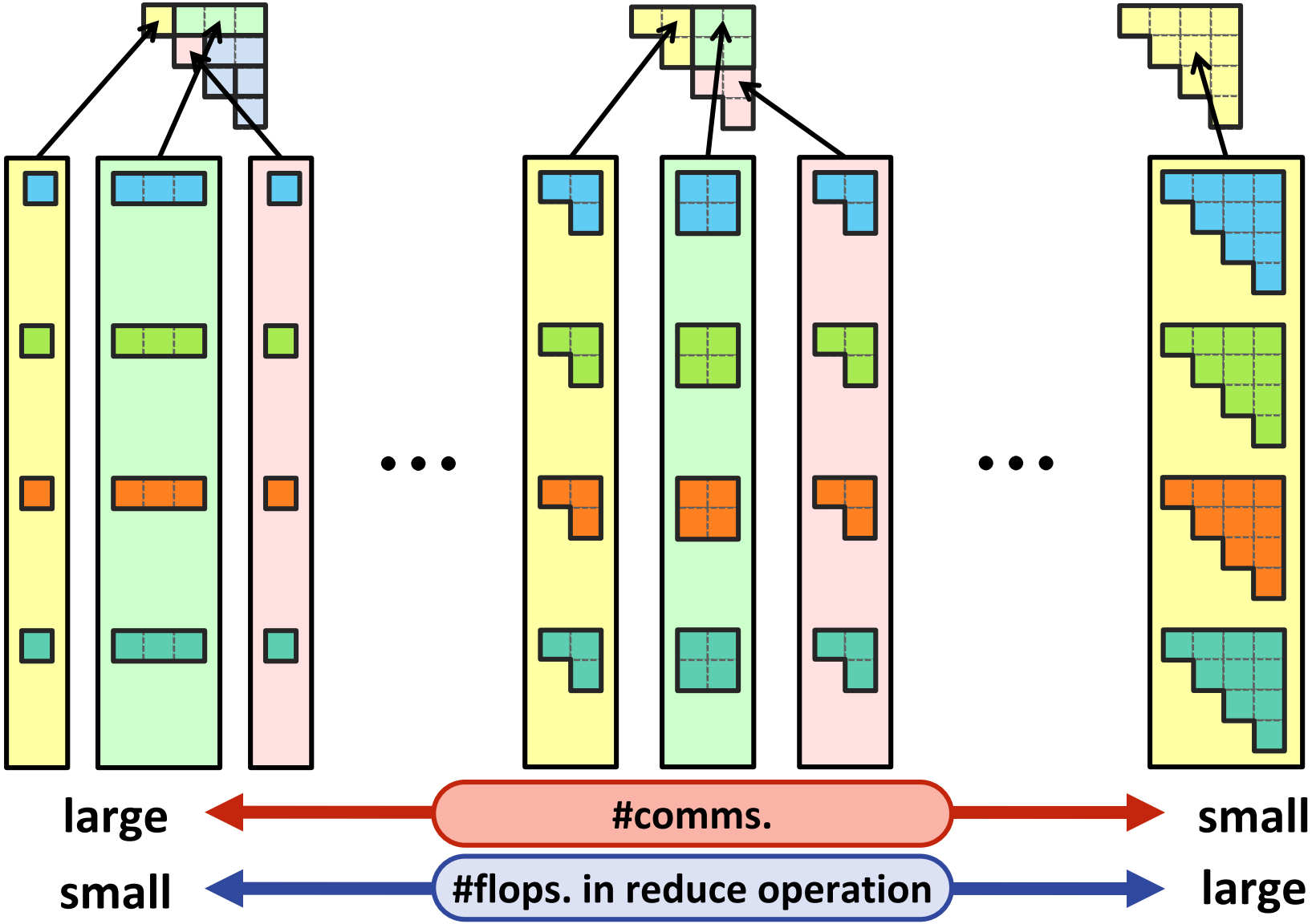
item	Householder	TSQR
#flops_ge	$2\frac{m}{P}n^2$	$2\frac{m}{P}n^2 - \frac{2}{3}n^3$
#flops_st		$\frac{2}{3}n^3\log_2 P$
#msgs	$2n\log_2 P$	$\log_2 P$
#words	$\frac{1}{2}n^2\log_2 P$	$\frac{1}{2}n^2\log_2 P$

行列の列数（n）が増加するとTSQRは不利になる？

TSQRは万能か？


T_{diff}(sec.)
m=2,048,000 P=1,024
[深谷, 今村, 山本, iWAPT2014ほか]
0.15
0.10
0.05
0.00
-0.05
-0.10
-0.15
-0.20
0
200
400
600
800
n
estimation with γ_{ge} (instead of γ_{st})
Demmelたちの
性能モデル
estimation with γ_{st}
execution
TSQR is faster
TSQR is slower
我々の
性能モデル
TSQRが遅くなる場合を再現
• α : 1.26×10^{-5}
• γ_{st} : 2.22×10^{-10}
• γ_{ge} : 3.30×10^{-11}

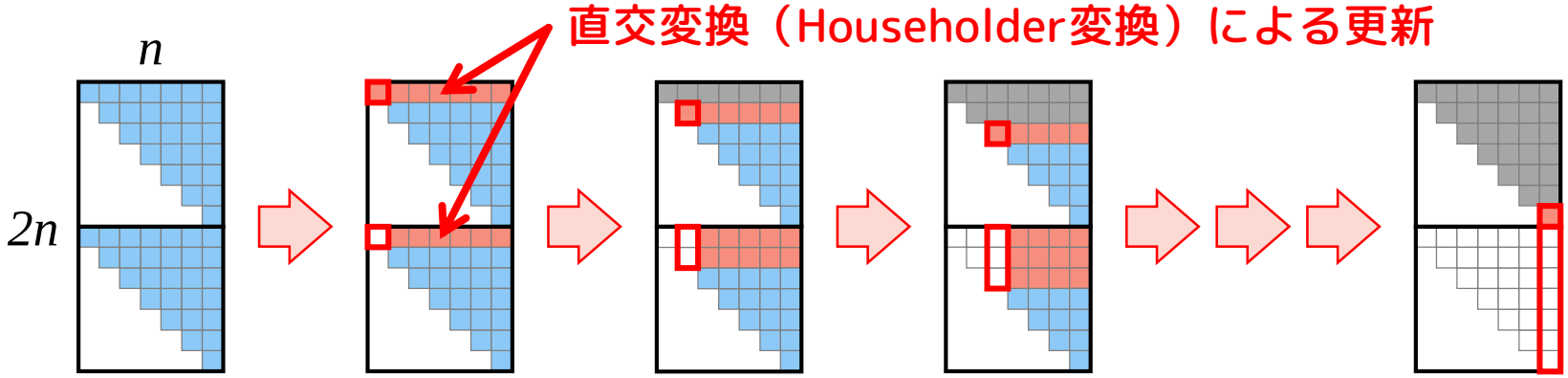
演算量と通信回数のトレードオフ



TSQRで鍵となる演算カーネル

◆ Structured QR分解

[深谷, 今村, 2014年計算工学講演会ほか]

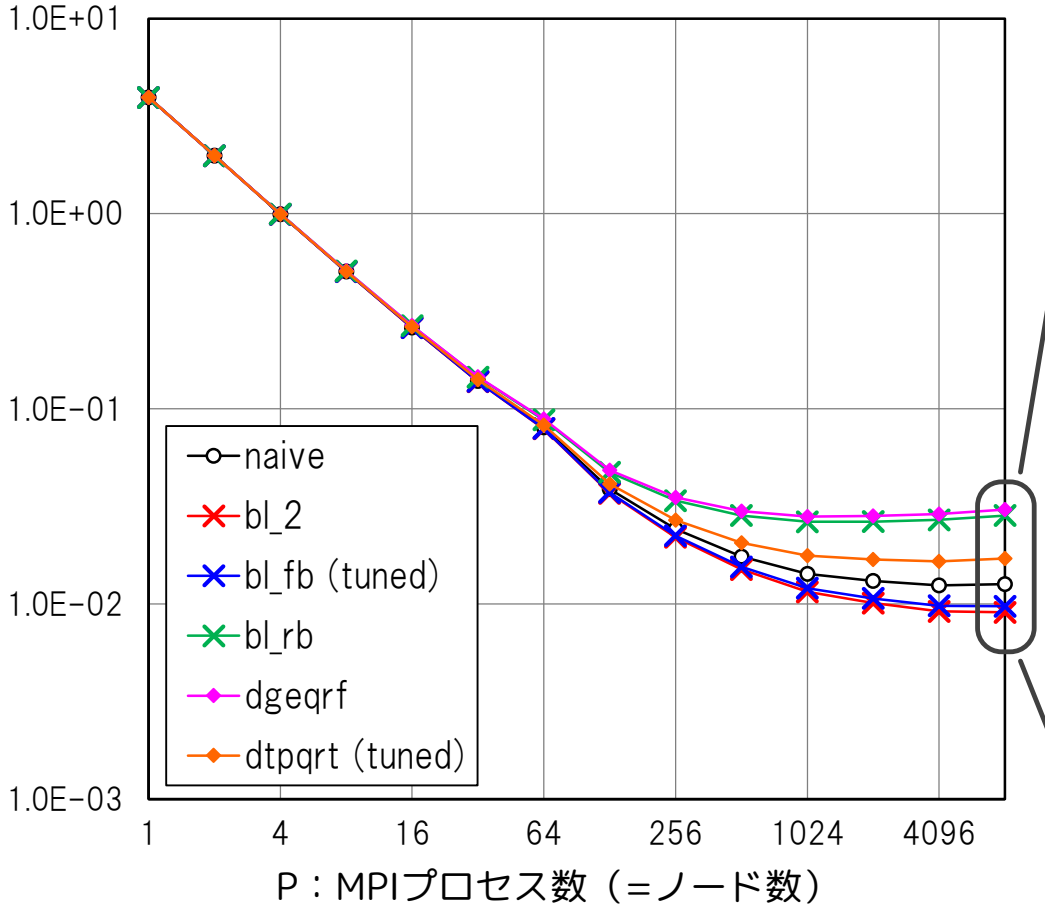


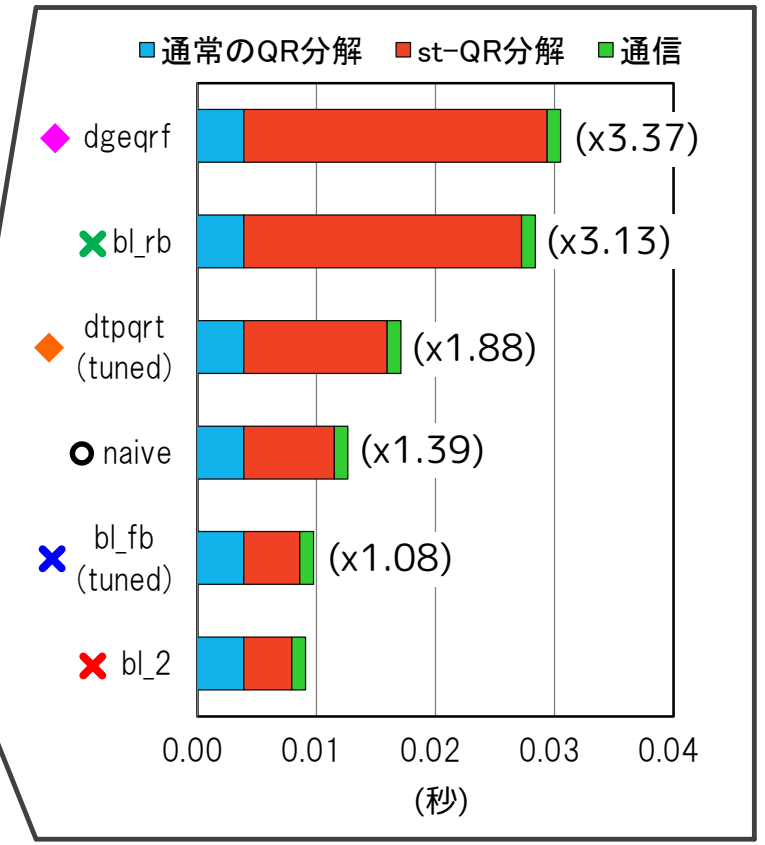
- TSQR中では$\log_2 P$回繰り返すため，ボトルネックになる可能性大
- 行列の構造（上三角）を生かすことで効率的に計算可能
  - ✓ 構造を考慮した場合の演算量：$\frac{2}{3}n^3$
  - ✓ 一般行列とした場合の演算量：$\frac{10}{3}n^3$ $(= 2 \cdot 2n \cdot n^2 - \frac{2}{3}n^3)$
- 行列サイズが比較的小さく，構造を考慮する必要があるため，従来の高性能化技法（ブロック化による行列積の活用など）の効果が不明

実装方法とその違いによる影響を調査

Structured QRの実装の重要性

- 京コンピュータ上での1,000,000 x 100の行列のQR分解を想定
- st-QR分解の実行時間 = (1回のst-QR分解の時間) x $\log_2 P$
- 通常のQR分解と通信の時間は実測値を使用





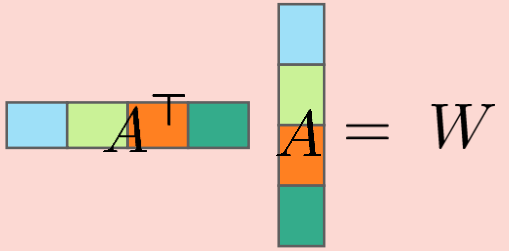
実装次第で大きな差が生じる

別の方法：CholeskyQR

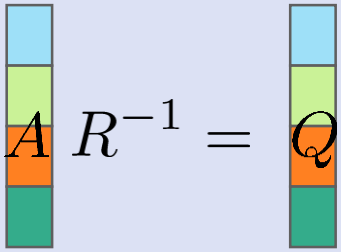
CholQR$(A) =: [Q, R]$

1. $W := A^\top A$
2. $R^\top R := W$
3. $Q := AR^{-1}$

各プロセスで行列積 ⇒ 集団通信



各プロセスで後退代入



CholQR2$(A) =: [Q, R]$

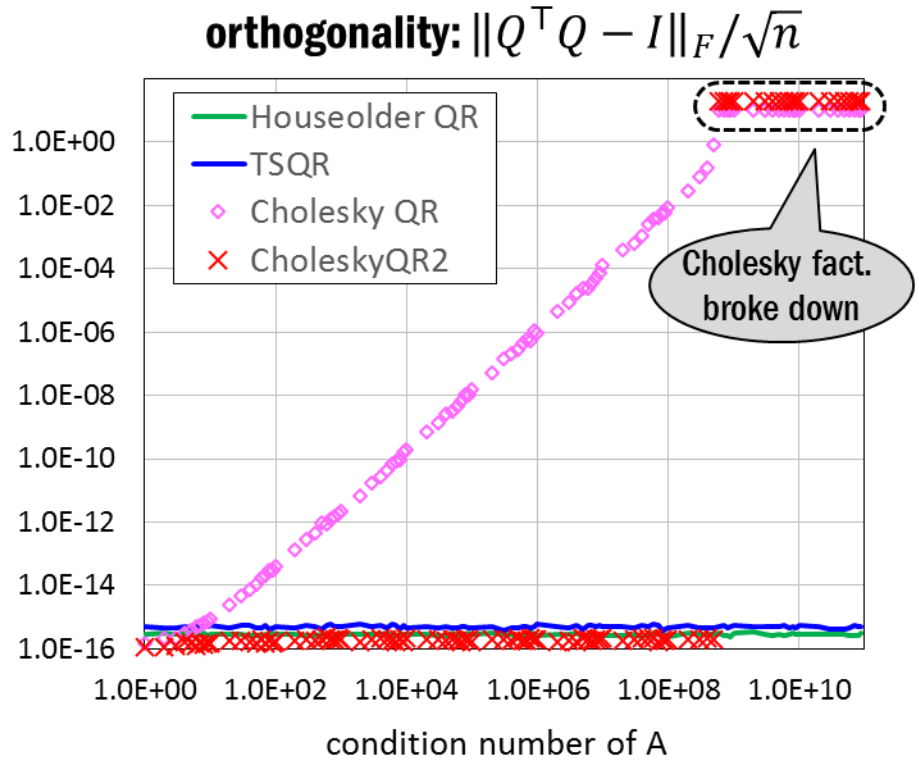
1. $[Q_1, R_1] := \mathrm{CholQR}(A)$
2. $[Q, R_2] := \mathrm{CholQR}(Q_1)$
3. $R := R_2 R_1$

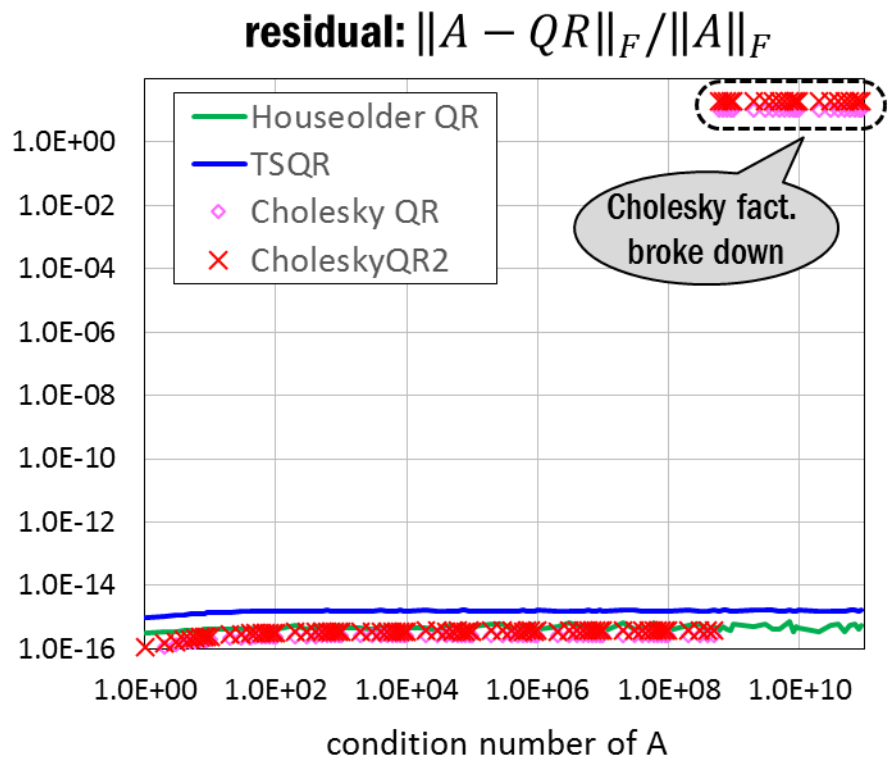
再直交化

CholeskyQR2の計算精度

◆ CholeskyQR2（Cholesky QRを2回繰り返す）の安定性

(m=2,097,152 n=64 P=8,192 @ Fx10 supercomputing system Oakleaf-FX)

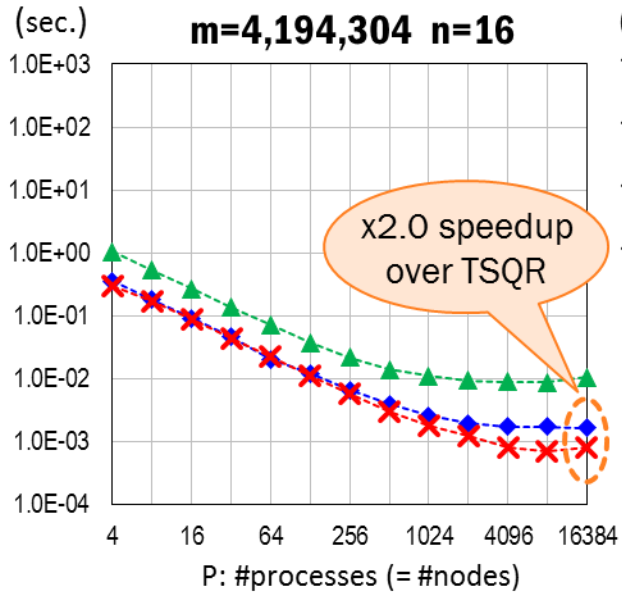


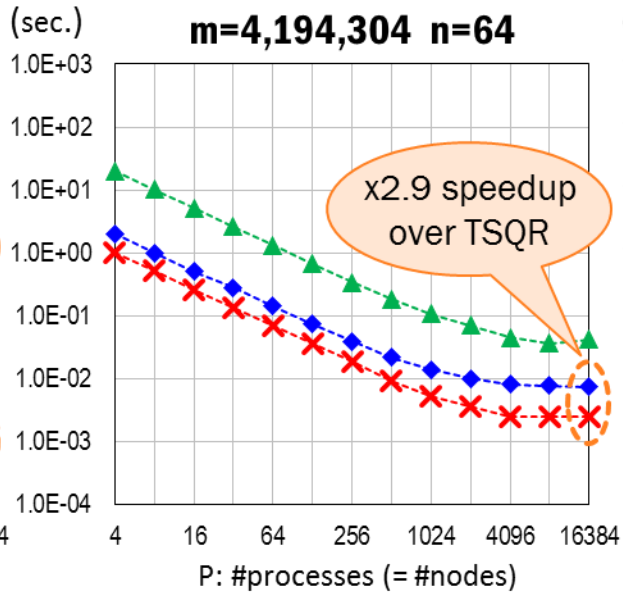


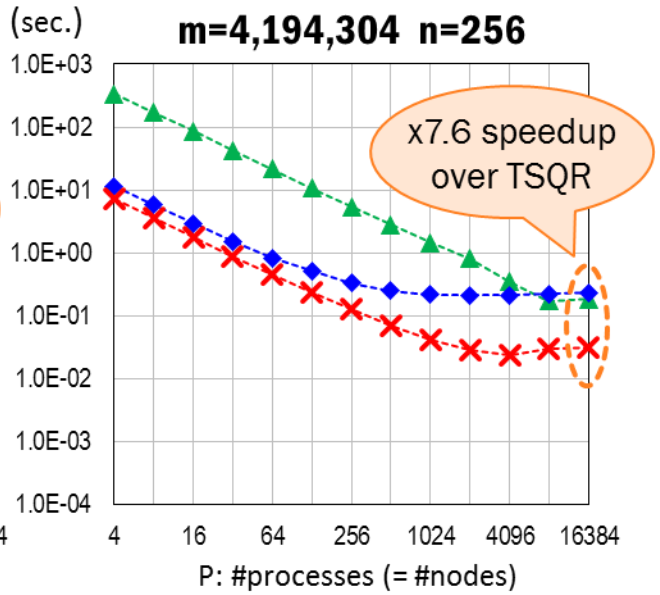
- 理論誤差解析の結果もあり（山本他, 2015）
- 条件数がより大きい場合は，必要に応じてグラム行列をシフトしてCholesky QR分解を繰り返すことで対応可能（柳澤他, 2014）

CholeskyQR2の性能

◆ CholeskyQR2の性能評価結果@京 (深谷他, ScalA'14等)





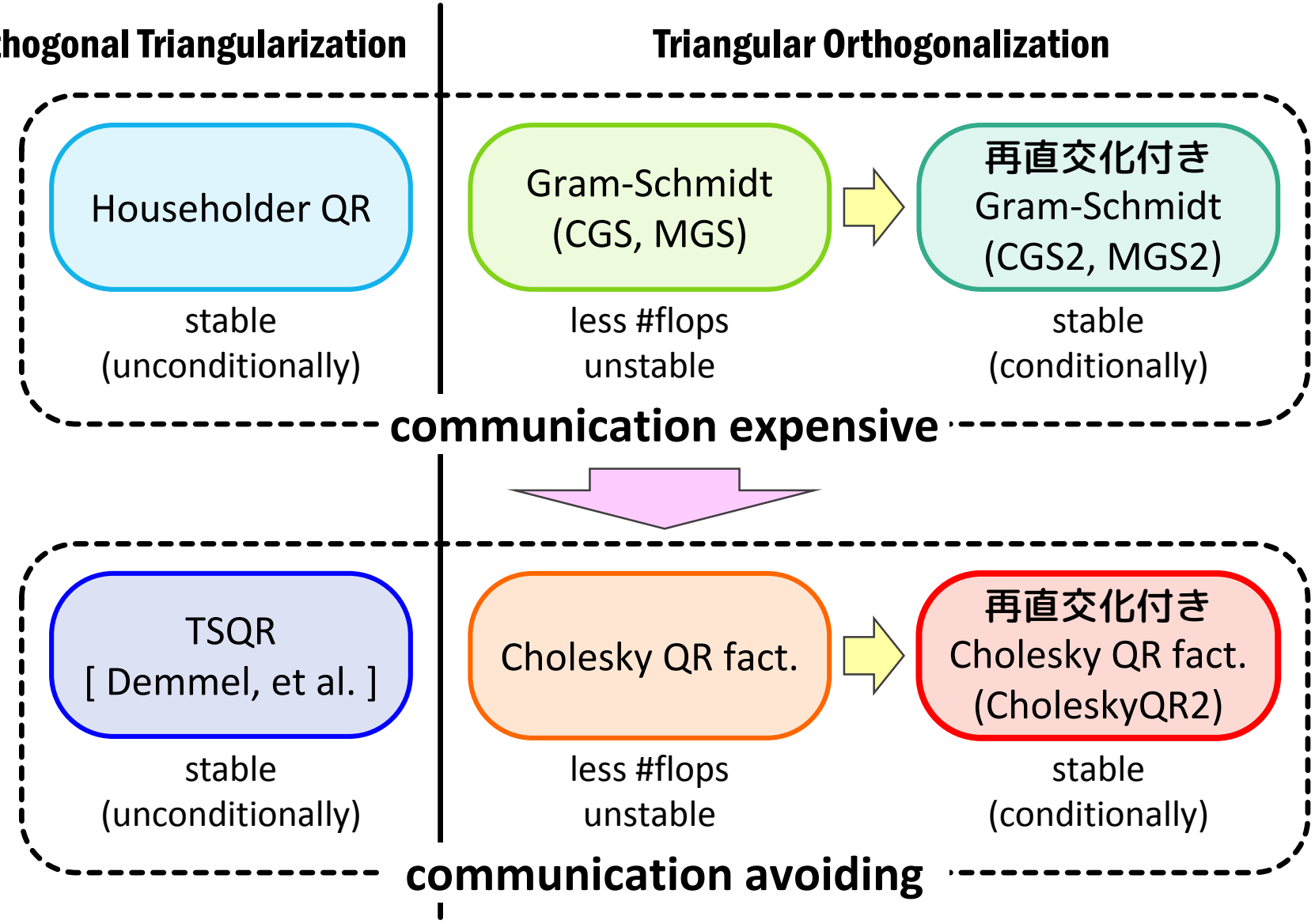


- 通信回数：Householder QR >> CholeskyQR2 > TSQR
- 通信データ量：TSQR > TSQR = Householder QR

しかし，依然としてCholeskyQR2の方がTSQRより高速！

（３回Cholesky QRを繰り返しても高速である可能性が高い）

縦長行列のQR分解方法の一覧



おわりに

今後の展望

- 通信回避型アルゴリズム：
  - ✓ 通信のレイテンシを削減可能（限界はある）
  - ✓ 通信回数の削減に伴うトレードオフが存在
    （演算量の増加，計算の不安定性など）
- 分散並列環境向けの線形計算ライブラリ：
  - ✓ 少ない並列数ならばScaLAPACKでも対応可能？
    （マルチコア・メニーコア向けBLASは必須？）
  - ✓ 異なるデータ分散への対応は容易ではない？
  - ✓ 一定数以上のユーザが期待されないと開発は難しい？
    （共有メモリ向けの方が期待できる）
- アプリケーションの設定ごとに，アルゴリズム・実装方法を検討する価値があるのではないか？