ScaLAPACKの使い方

森田 悟史 （東大物性研）

ScaLAPACKとは?

Scalable **L**inear **A**lgebra **PACK**age

http://www.netlib.org/scalapack/

- 線形計算ライブラリ LAPACK のメモリ分散並列計算版
  - 線形方程式，線形最小二乗問題，固有値問題，特異値問題, etc.
  - LU分解，コレスキー分解，QR分解, etc.
  - 行列積，ベクトル行列積，etc.（同梱のpBLACSを利用）

- Netlib上で公開，自由に利用できる.
  - ライセンス: 修正BSDライセンス (LAPACKと同じ)
- 開発者:
  - Jim Demmel (Univ. California, Berkeley, USA)
  - Jack Dongarra (Univ. Tennessee and ORNL, USA)
  - Julien Langou (Univ. Colorado Denver, USA)

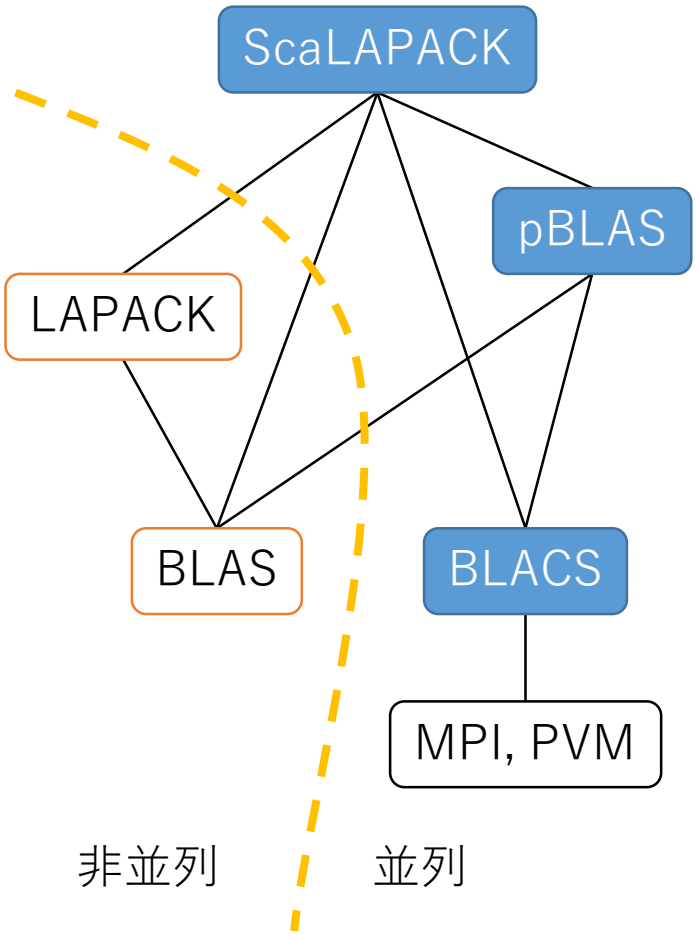
ScaLAPACKに対するユーザーの反応

- 2016 Dense Linear Algebra Software Packages Survey
  - 2016年1月から6月まで，252人の回答
  - 詳細は LAWN 290 (**LA**PACK **W**orking **N**ote)
    http://www.netlib.org/lapack/lawnspdf/lawn290.pdf
- ScaLAPACKについて80人が回答 (LAPACKは186人)
- 使用言語: Fortran 60%, C 50%, C++ 40%
- ScaLAPACKのインターフェイス: **52%**が使いにくい
- ScaLAPACKを使用しない理由:
  - データ分散方法が分かりづらい
  - ドキュメントが難しい
  - 性能が出ない
  - 他のライブラリ (Elemental, Eigen, etc.) を使用している

リリース履歴

	ScaLAPACK	LAPACK
1990		1992/02/29 ver. 1.0
1995	1995/02/28 ver. 1.0 1996/11/17 ver. 1.4 1997/11/15 ver. 1.6	1994/09/30 ver. 2.0
2000	2001/08/31 ver. 1.7	1999/06/30 ver. 3.0
2005	2007/04/05 ver. 1.8.0	2006/11/12 ver. 3.1.0 2008/11/18 ver. 3.2.0
2010	2011/11/11 ver. 2.0.0 2012/05/01 ver. 2.0.2	2010/11/14 ver. 3.3.0 2011/11/11 ver. 3.4.0 2013/11/19 ver. 3.5.0
2015	最新のアルゴリズムには 対応していない	2015/11/15 ver. 3.6.0 2016/06/18 ver. 3.6.1

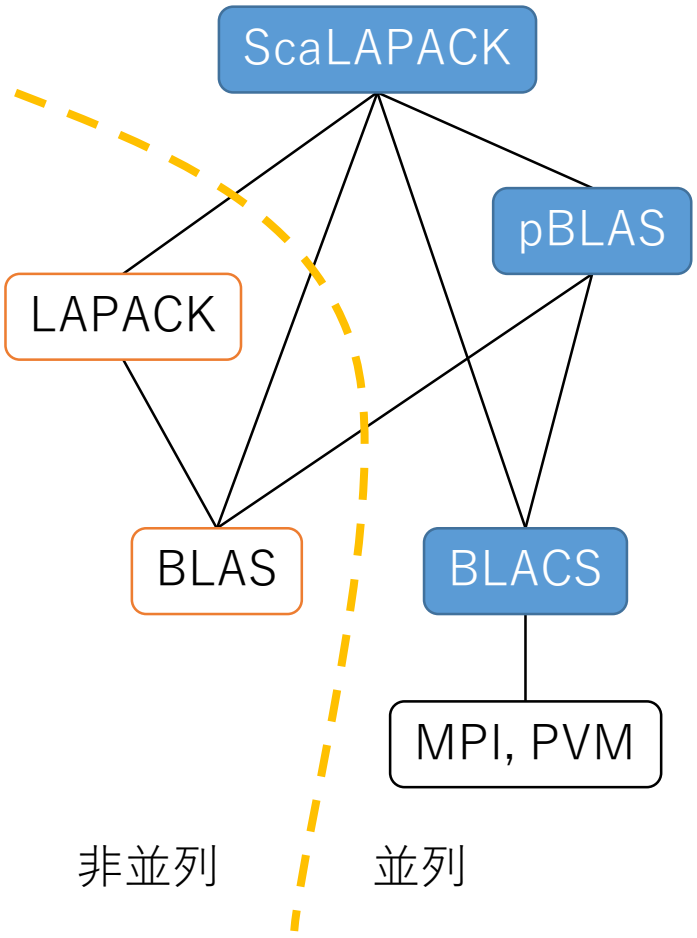
ライブラリの階層構造



- LAPACK (**L**inear **A**lgebra **PACK**age)
  線形演算ライブラリ
  - ドライバールーチン
    線形方程式，線形最小二乗問題，
    固有値問題，特異値問題，etc.
  - 計算ルーチン
    LU分解，コレスキー分解，QR分解, etc.

- BLAS (**B**asic **L**inear **A**lgebra **S**ubprograms)
  ベクトルと行列に関する基本線形代数操作
  - Level-1: ベクトル演算
    $$\boldsymbol{y} \leftarrow \alpha\boldsymbol{x} + \boldsymbol{y}$$
  - Level-2: 行列ベクトル演算
    $$\boldsymbol{y} \leftarrow \alpha A\boldsymbol{x} + \beta\boldsymbol{y}$$
  - Level-3: 行列同士の演算
    $$C \leftarrow \alpha AB + \beta C$$

ライブラリの階層構造



- ScaLAPACK
  (**Sca**lable **L**inear **A**lgebra **PACK**age)
  分散並列線形演算ライブラリ
- pBLAS (**p**arallel **BLAS**)
  BLASの分散並列版
- BLACS (**B**asic **L**inear **A**lgebra **C**ommunication **S**ubprograms)
  線形演算のための通信を担当
  現在は，ほぼMPIのラッパー
  http://www.netlib.org/blacs/
- MPI (**M**essage **P**assing **I**nterface)
  分散メモリ型並列計算の通信プロトコル
- PVM (**P**arallel **V**irtual **M**achine)

関数の命名規則

- BLAS
  - Level 1: **DAXPY**
  - Level 2 & 3: **DGEMM**

Quick Reference: http://www.netlib.org/blas/blasqr.pdf

AXPY: $\alpha \boldsymbol{x} + \boldsymbol{y}$
DOT: $\boldsymbol{x}^T \boldsymbol{y}$ (複素数はDOTU)
DOTC: $\boldsymbol{x}^H \boldsymbol{y}$
NRM2: $|\boldsymbol{x}|_2$ etc.

S: 単精度実数 (**s**ingle)
D: 倍精度実数 (**d**ouble)
C: 単精度複素数 (**c**omplex)
Z: 倍精度複素数

MV: $\alpha A \boldsymbol{x} + \beta \boldsymbol{y}$ (**M**atrix-**V**ector)
R: $\alpha \boldsymbol{x} \boldsymbol{x}^T + A$ (**R**ank-1 operation)
R2: $\alpha \boldsymbol{x} \boldsymbol{y}^T + \alpha \boldsymbol{y} \boldsymbol{x}^T + A$ (**R**ank-**2** operation)

MM: $\alpha AB + \beta C$ (**M**atrix-**M**atrix)
RK: $\alpha AA^T + \beta C$ (**R**ank-**k** operation)
R2K: $\alpha AB^T + \alpha^* BA^T + \beta C$ etc.

GE: 一般行列 (**Ge**neral)
SY: 対称行列 (**Sy**mmetric)
HE: エルミート (**He**rmitian)
TR: 三角行列 (**Tr**iangular)
GB: 一般バンド行列 (**G**eneral **B**and) etc.

可能な組み合わせは制限されている.
例） OK: DSYMM, DSYRK
NG: DHEMM, DGERK

関数の命名規則

- LAPACK
  - 基本的には BLAS Level 2&3 と同じ．
  - 行列の種類が増える．

DGEEV

倍精度実数
一般行列
特異値問題

GG:	**g**eneral matrix, **g**eneral problem
OR:	**or**thogonal
PO:	symmetric or Hermitian **po**sitive definite
UN:	**un**itary etc.

  - ドライバルーチン

SV:	連立一次方程式 (**S**ol**v**e)
LS:	最小二乗法 (**L**east **s**quare)
EV:	固有値問題 (**E**igen**v**alues)
SVD:	特異値分解

接尾辞によって
アルゴリズム等が変わる

EV:	シンプルドライバー
EVX:	エキスパートドライバー
EVD:	分割統治法ドライバー
EVR:	RRRドライバー Relatively robust representations

- pBLAS, ScaLAPACK
  - BLAS, LAPACKの関数名に “P” をつける．
  - LAPACKにあるが，ScaLAPACKにない関数が多数ある．

関数のインターフェイス

- 例: PDGEQRF (一般実行列のQR分解)

[g] : Global
[l] : Local

```
subroutine PDGEQRF(
    INTEGER M,                                  列サイズ [g]
    INTEGER N,                                  行サイズ [g]
    DOUBLE PRECISION, dimension(*) A,           入出力行列 [l]
    INTEGER IA,                                 部分行列の先頭列 [g]
    INTEGER JA,                                 部分行列の先頭行 [g]
    INTEGER, dimension(*) DESCA,                Descriptor [g/l]
    DOUBLE PRECISION, dimension(*) TAU,         スカラー出力 [l]
    DOUBLE PRECISION, dimension(*) WORK,        作業領域 [l]
    INTEGER LWORK,                              作業領域サイズ [l]
    INTEGER INFO                                情報出力 [g]
)
```

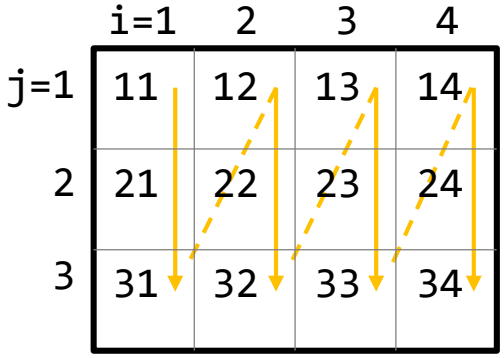
Leading dimension

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

行優先 (row-major)と列優先 (column-major)

- 行列の一次元配列への格納方法

➢ Column-major order : Fortran形式

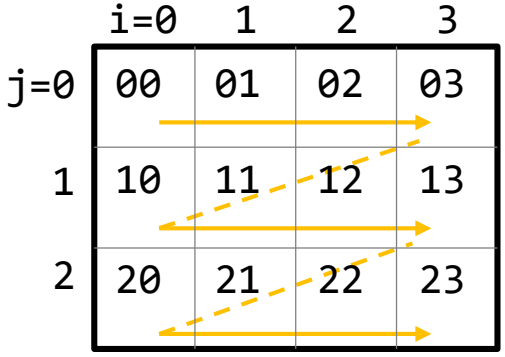


⇨ (11,21,31, 12,22,32, 13,23,33, 14,24,34)

A(i,j) = v(j*m+i)　　i=1,2,…,m, j=1,2,…,n

ScaLAPACK等の行列データ

➢ Row-major order : C言語形式



⇨ (00,01,02,03, 10,11,12,13, 20,21,22,23)

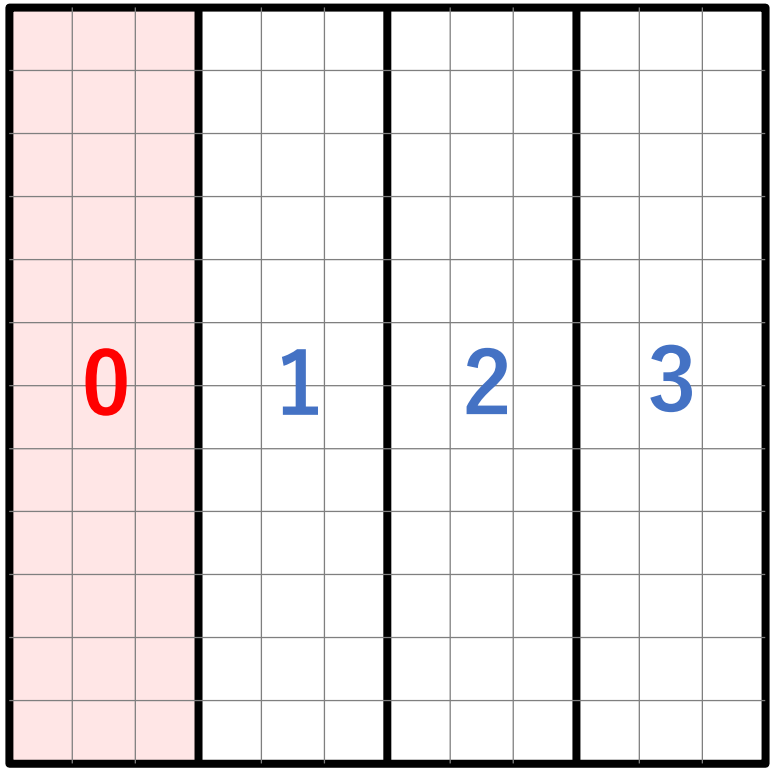
A[i][j] = v[i*n+j]　　i=0,1,…,m-1, j=0,1,…,n-1

BLACSにおけるプロセスの並べ方

画像は http://id.fnshr.info/2015/10/17/matrix/ より引用

行列の分散方式あれこれ

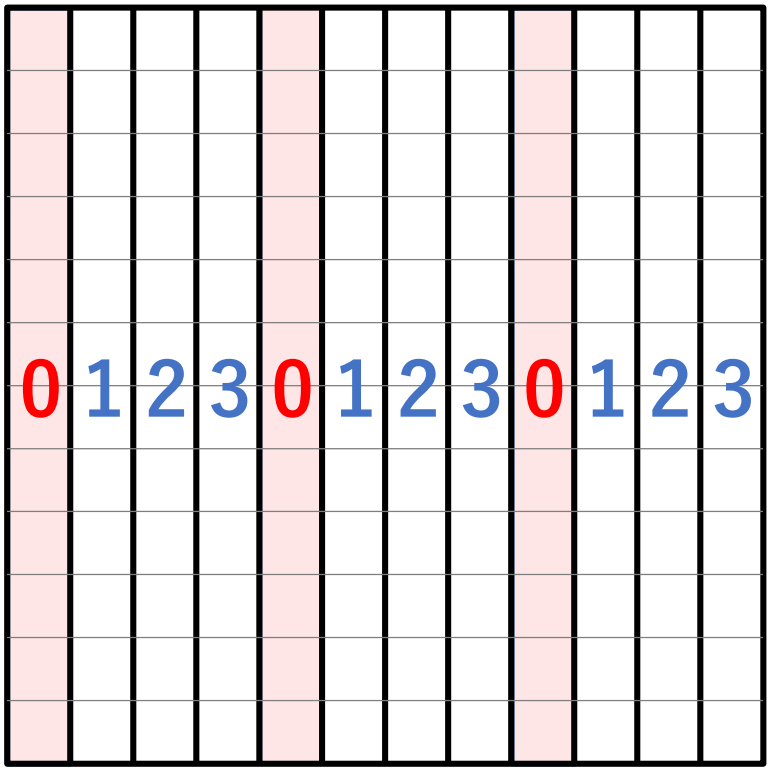
- 1-d block column dist.



列ブロックに分割する

データの連続性を重視

- 1-d cyclic column dist.

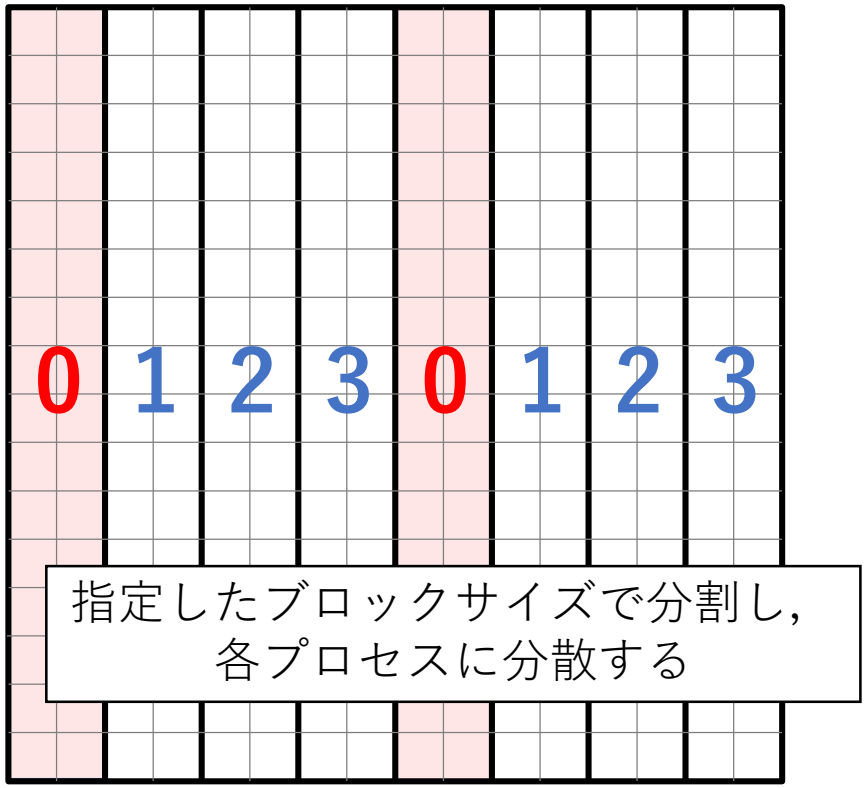


1列ごとに配分する

ロードバランスを重視

行列の分散方式あれこれ

- 1-d block-cyclic column dist.



Block size = 2

- 2-d block-cyclic dist.

ScaLAPACKの行列分散方式

0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3

プロセスも2次元に並べる必要がある
→ BLACS が担当

BLACSのコンテキスト

- プロセスグリッド
  - MPIプロセスを2次元に配置する.
  - マッピング方法は自由
    - デフォルトはrow-major
    - プロセスの一部だけ使用することも可能
- コンテキスト
  - プロセスグリッド情報を保持
  - MPI_Commに相当

$p_c = 0$	1	2
$p_r = 0$	0	1	2
1	3	4	5

(p_r, p_c)

Rank 0 → (0, 0)
Rank 1 → (0, 1)
Rank 2 → (0, 2)
Rank 3 → (1, 0) …

プロセス関連の番号は,
0から始まるので注意

- BLACSの初期化・終了処理の流れ

MPIの初期化	`MPI_Init`
グリッドサイズの計算	
グリッドの初期化	`blacs_gridinit`
（計算）	
BLACSの終了処理	`blacs_exit`
MPIの終了処理	`MPI_Finalize`

- MPIを初期していない場合，blacs_gridinitは裏で勝手にMPI_Init を呼び出すが,
自分で初期化したほうが無難.
- MPI_Dims_createを使うと
グリッドの行数，列数が簡単に計算可能.

BLACSのコンテキストに関する主な関数

blacs_gridinit(icontxt, order, nprow, npcol)
コンテキストの初期化
icontxt: [in/out] コンテキスト
order: [in] “R”ow-major 又は “C”olumn-major
nprow, npcol: [in] プロセスグリッドのサイズ

blacs_gridmap
コンテキストの初期化
プロセスからグリッドへの対応関係を直接指定出来る

blacs_exit(continue)
BLACSの終了処理
continue: [in] 0 の場合，MPIの終了処理 (MPI_Finalize) を内部で呼ぶ．
それ以外の場合，BLACS終了後もMPI通信が可能．

blacs_gridinfo(icontxt, nprow, npcol, myprow, mypcol)
コンテキストの情報取得
icontxt: [in] コンテキスト
nprow, npcol: [out] プロセスグリッドのサイズ
myprow, mypcol: [out] 自分のグリッド上の位置

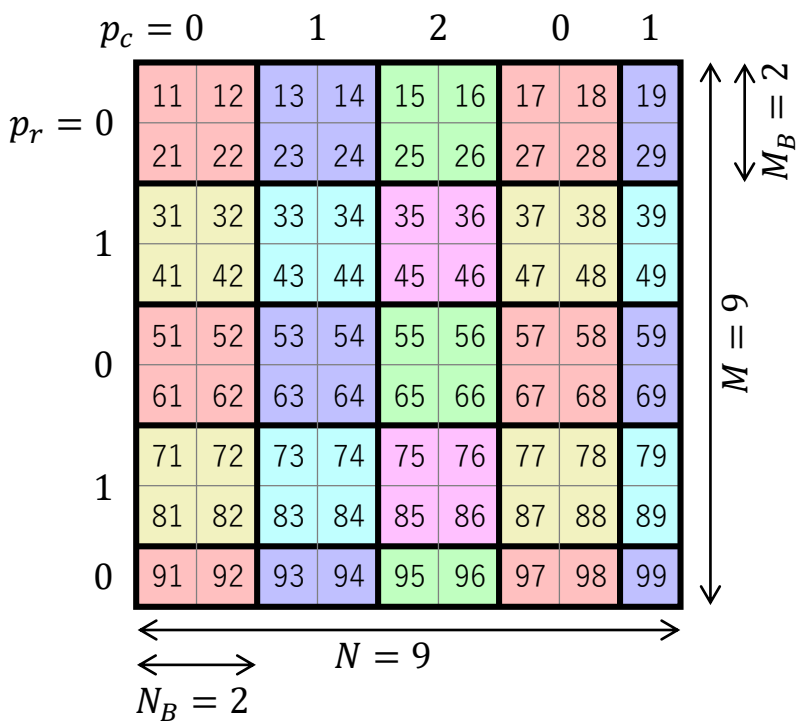
blacs_pnum(icontxt, prow, pcol)
(prow, pcol) のプロセス番号取得

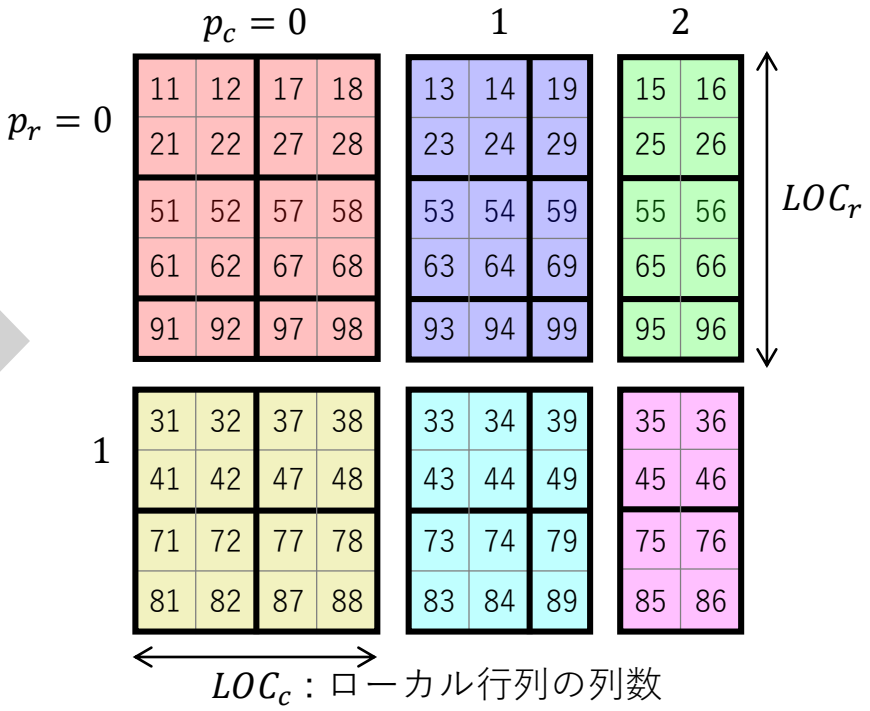
blacs_pinfo(mypnum, nprocs)
自分のプロセス番号とプロセス数の取得

他にもBLACSには行列成分を通信するための関数がある．
詳細は，BLACSのUser’s Guide (LAWN94) や Quick Ref. を参照
http://www.netlib.org/lapack/lawnspdf/lawn94.pdf
http://www.netlib.org/blacs/BLACS/QRef.html

Two-dimensional block-cyclic distribution

- Example: 9x9 matrix, 2x2 block size, 2x3 process grid

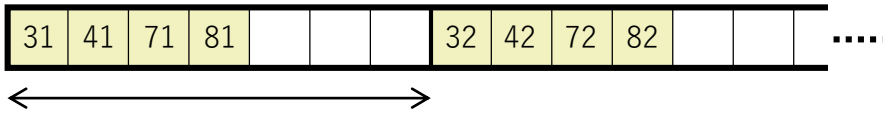




(p_r, p_c)	LOC_r	LOC_c
(0, 0)	5	4
(0, 1)	5	3
(0, 2)	5	2

(p_r, p_c)	LOC_r	LOC_c
(1, 0)	4	4
(1, 1)	4	3
(1, 2)	4	2

プロセス内はcolumn-major



Array Descriptor

- 分散行列の記述子
  - 密行列の場合: 長さ9の整数配列

#	Symbol	Scope	Definition	コメント
0	DTYPE	global	Descriptor type.	密行列の場合，1
1	CTXT	global	BLACSコンテキスト	
2	M	global	行列の行数	0 以上
3	N	global	行列の列数	0 以上
4	MB	global	ブロックの行サイズ	1 以上
5	NB	global	ブロックの列サイズ	1 以上
6	RSRC	global	先頭行があるプロセスの行座標.	通常は 0
7	CSRC	global	先頭列があるプロセスの列座標.	通常は 0
8	LLD	local	ローカル行列のLeading dimension	LOC_r以上，0 は不可

  - 記述子の初期化用関数

`descinit(DESC,CTXT,M,N,MB,NB,RSRC,CSRC,LLD,INFO)`
配列DESCを与えられた引数で初期化する．エラーチェック付.
配列DESCの領域確保は事前に行う必要あり.

Block-cyclic distributionに関する便利な関数

- ローカル行列のサイズ取得

numroc(N,NB,IPROC,ISRCPROC,NPROCS)
NUMber of **R**ows **O**r **C**olumns
ローカル行列の列数，行数を計算する

- ✓ 配列インデックスは 1 スタート
  プロセス番号は 0 スタート
- ✓ 行と列は別々に計算する
- ✓ 何度も呼び出す必要があるときは
  手動インライン展開も検討する

- インデックス変換

indxg2l(INDXGLOB,NB,IPROC,ISRCPROC,NPROCS)
グローバルインデックス → ローカルインデックス

```
INDXG2L = NB*((INDXGLOB-1)/(NB*NPROCS)) + MOD(INDXGLOB-1,NB) + 1
```

indxg2p(INDXGLOB,NB,IPROC,ISRCPROC,NPROCS)
グローバルインデックス → プロセスグリッド座標

```
INDXG2P = MOD(ISRCPROC+(INDXGLOB-1)/NB, NPROCS)
```

indxl2g(INDXLOC,NB,IPROC,ISRCPROC,NPROCS)
ローカルインデックス → グローバルインデックス

```
INDXL2G = NPROCS*NB*((INDXLOC-1)/NB) + MOD(INDXLOC-1,NB) +
          MOD(NPROCS+IPROC-ISRCPROC, NPROCS)*NB + 1
```

インストール方法

- ソースコードから
  - http://www.netlib.org/scalapack/scalapack-2.0.2.tgz
  - インストールガイド LAWN93 (ver 1.7用なので情報が古いかも)

  1. SLmake.inc.example を修正し SLmake.ink を作成
  2. makeする

- インストーラー (Linux用)
  - http://www.netlib.org/scalapack/scalapack_installer.tgz
  - ダウンロード，コンパイル，インストールまで自動化

- ScaLAPACK for Windows
  - http://icl.cs.utk.edu/lapack-for-windows/scalapack/

  多くの場合，ベンダーから提供されている．
  （ただしバージョンが古い場合あり）

Intel MKLを利用する

- Intel MKL (物性研スパコンシステムBなど)
  - リンクオプション例

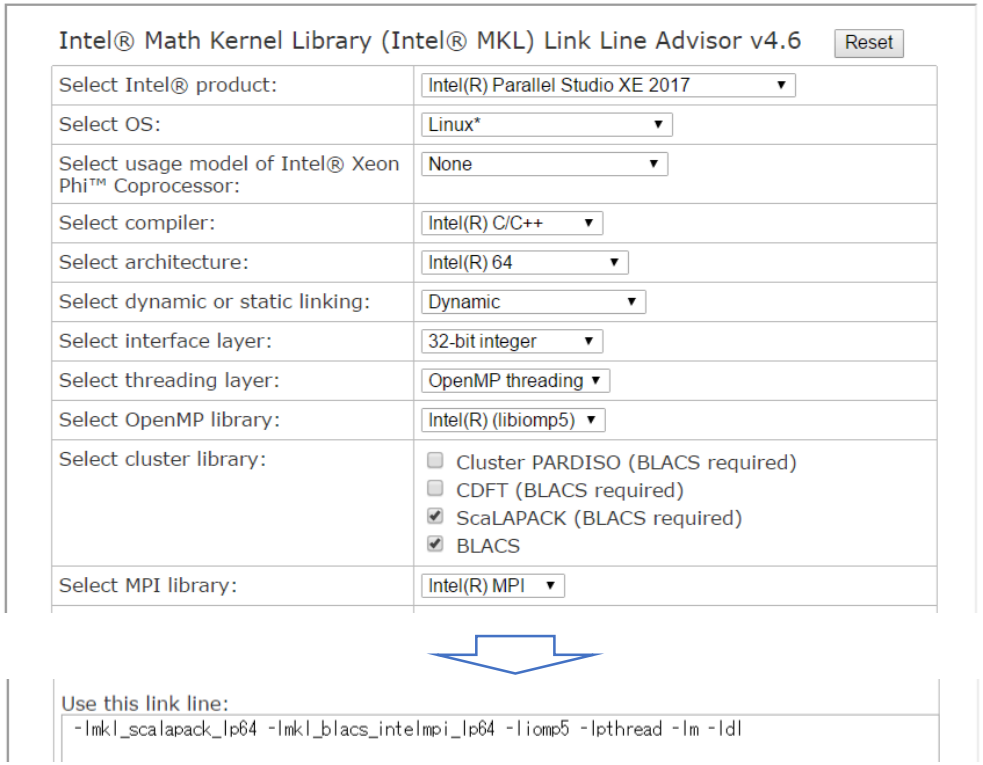
```
-lmkl_scalapack_lp64
-lmkl_blacs_intelmpi_lp64
-liomp5 -lpthread -lm -ldl
```

  - コンパイルオプション例

```
-mkl=parallel
```

- Intel Math Kernel Library Link Line Advisor

  https://software.intel.com/en-us/articles/intel-mkl-link-line-advisor



富士通系コンパイラの場合 (京，システムCなど)
-SCALAPACK -SSL2BLAMP （スレッド並列版SSL2ライブラリ）

ドキュメント

- 公式サイト
  - トップページ http://www.netlib.org/scalapack/
  - User's Guide http://netlib.org/scalapack/slug/index.html
  - コード検索 http://www.netlib.org/scalapack/explore-html/
  - BLACS http://www.netlib.org/blacs/
    - Quick Ref. http://www.netlib.org/blacs/BLACS/QRef.html
  - LAPACK http://www.netlib.org/lapack/
    - コード検索 http://www.netlib.org/lapack/explore-html/
  - BLAS http://www.netlib.org/blas/
    - Quick Ref. http://www.netlib.org/blas/blasqr.pdf
- Intel MKL Developer's Reference
  - C言語 https://software.intel.com/en-us/mkl-reference-manual-for-c
  - Fortran https://software.intel.com/en-us/mkl-reference-manual-for-fortran

C/C++言語からScaLAPACKを利用する

- C言語の場合
  - Fortranは参照渡しなので，引数は全てポインタで宣言.
  - 配列以外の変数は，アドレス演算子 (&) が必要.

宣言
```
extern void pdgeqrf_(int *M, int *N,
     double *A, int *ia, int *ib, int *desca,
     double *tau, double *work, int *lwork, int *info);
```

呼出
```
pdgeqrf_(&M, &N, A, &ia, &ib, desca,
         tau, work, &lwork, &info);
```

- C++の場合
  - 配列以外の変数は参照を利用して宣言.
  - アドレス演算子 (&) は不要になる.
  - constはキャスト (const_cast) で外す.
  - 「extern “C”」で名前修飾を回避.
  - 配列がstd::vectorの場合，先頭のアドレスを渡す &(v[0])

宣言
```
extern “C” void pdgeqrf_(int &M, int &N,
     double *A, int &ia, int &ib, int *desca,
     double *tau, double *work, int &lwork, int &info);
```

呼出
```
pdgeqrf_(M, N, A, ia, ib, desca, tau, work, lwork, info);
```

BLACS context と MPI communicator の相互変換

- Fortranの場合,
  - ➢BLACS context = MPI_COMM (どちらも整数)
- C言語/C++の場合,
  - ➢MPI_Commは整数型とは限らない (実装依存)

```
int Csys2blacs_handle(MPI_Comm comm);
```
MPI_Comm → BLACS context

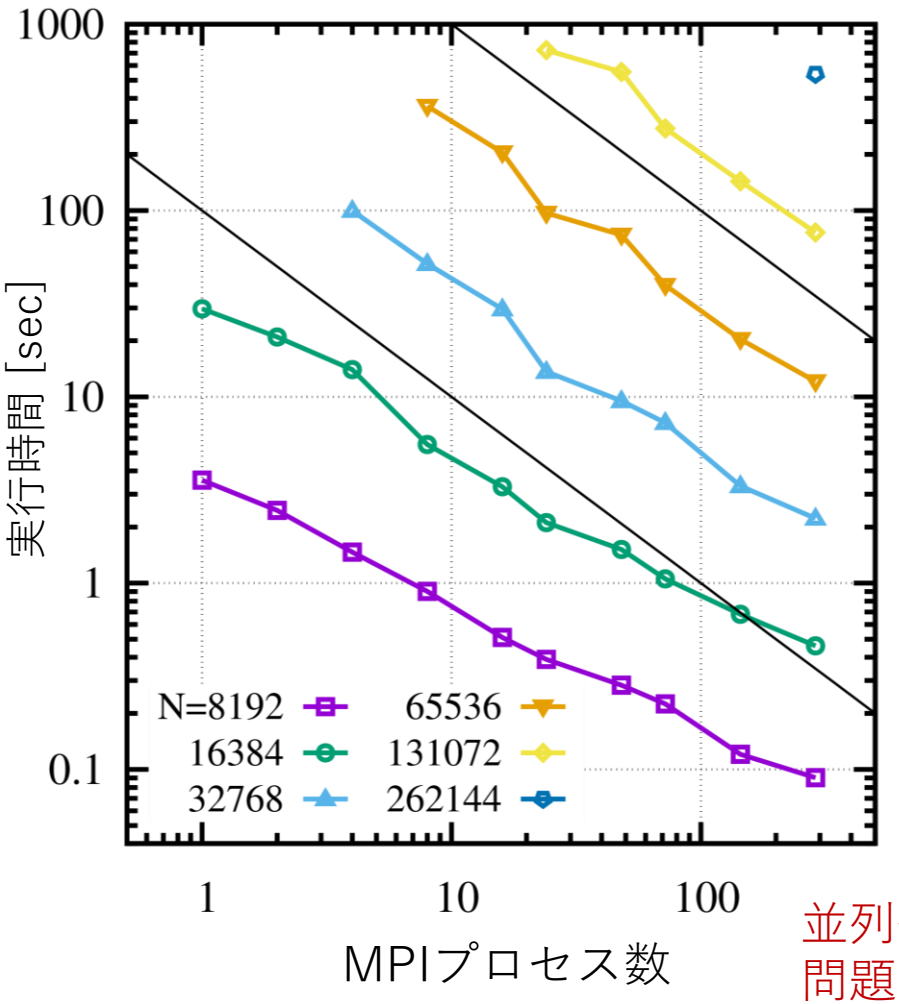
```
MPI_Comm Cblacs2sys_handle(int ictxt);
```
BLACS context → MPI_Comm

“Outstanding Issues in the MPIBLACS”, R. C. Whaley (1997)
http://www.netlib.org/blacs/mpiblacs_issues.ps

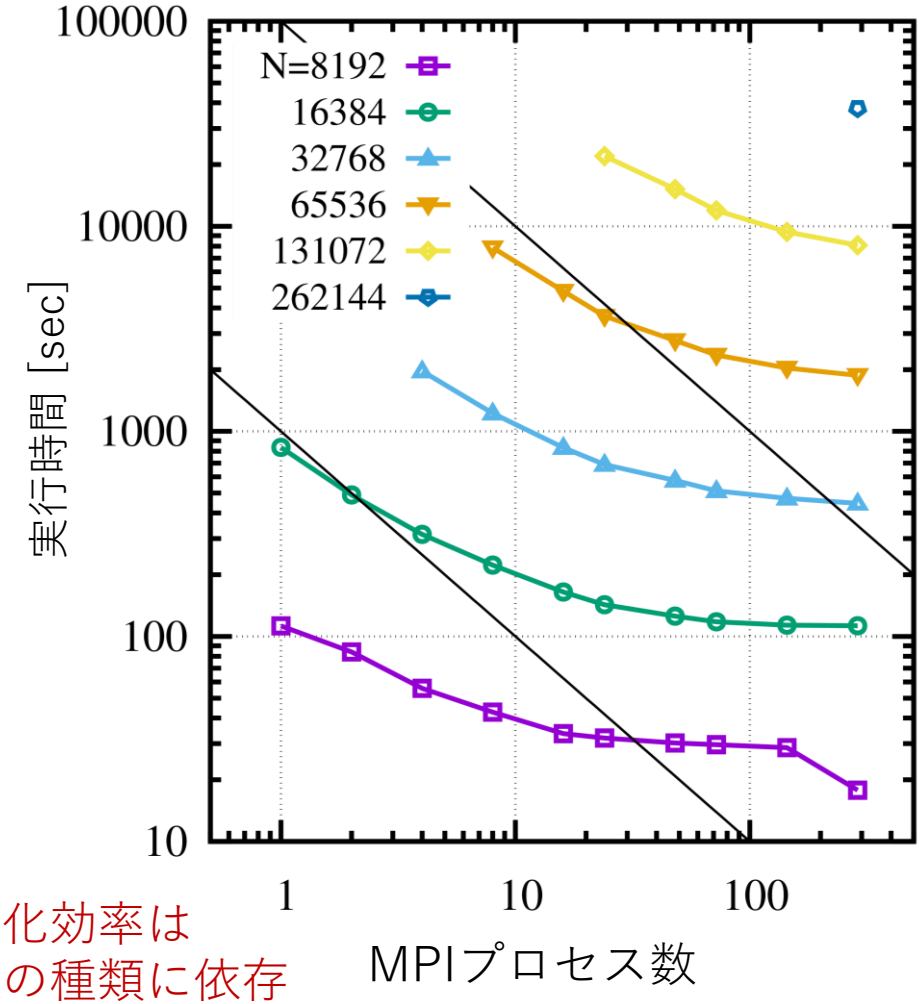
ベンチマーク @ システムB

Block size=64
12 threads

- 行列積 (pdgemm)



- 特異値分解 (pdgesvd)

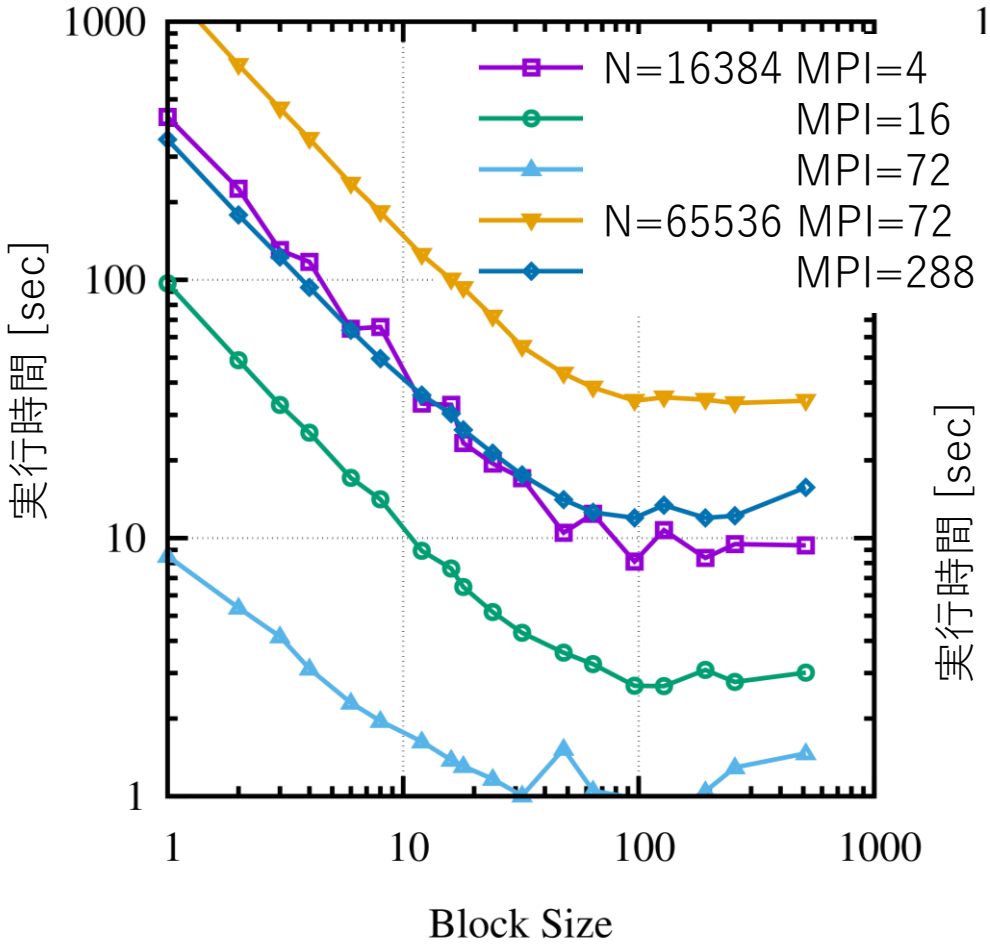


並列化効率は
問題の種類に依存

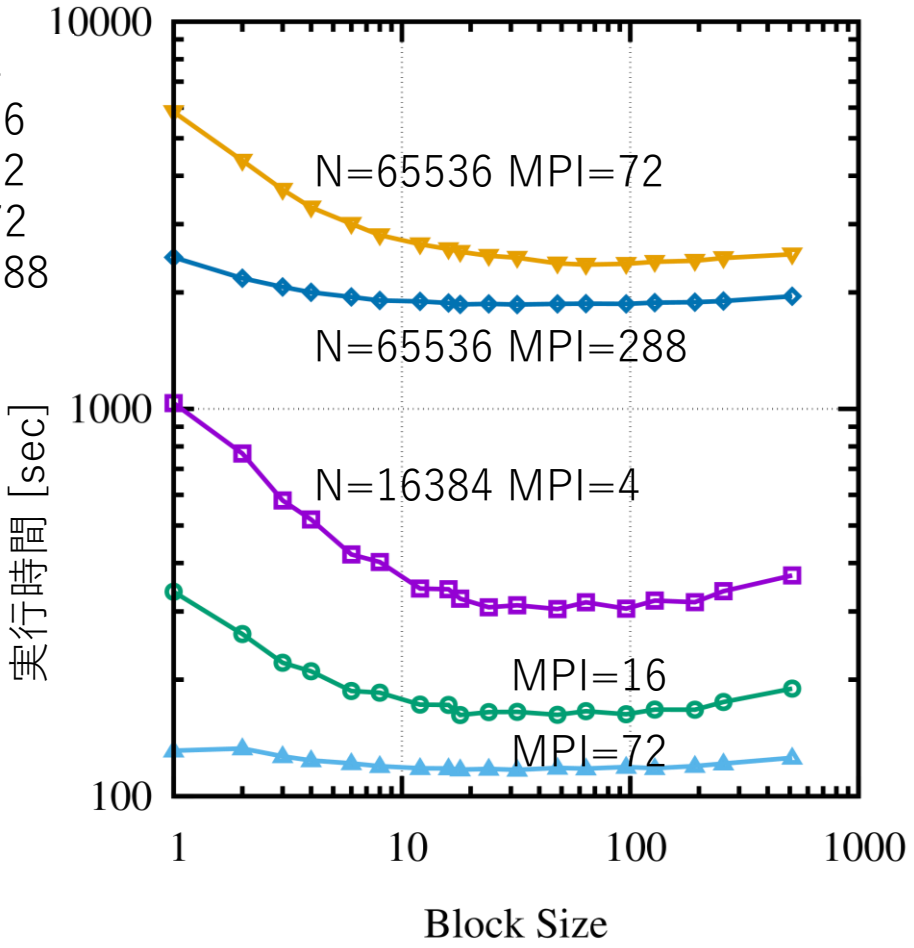
計算時間のブロックサイズ依存性

ISSP System B
12 threads

- 行列積 (pdgemm)



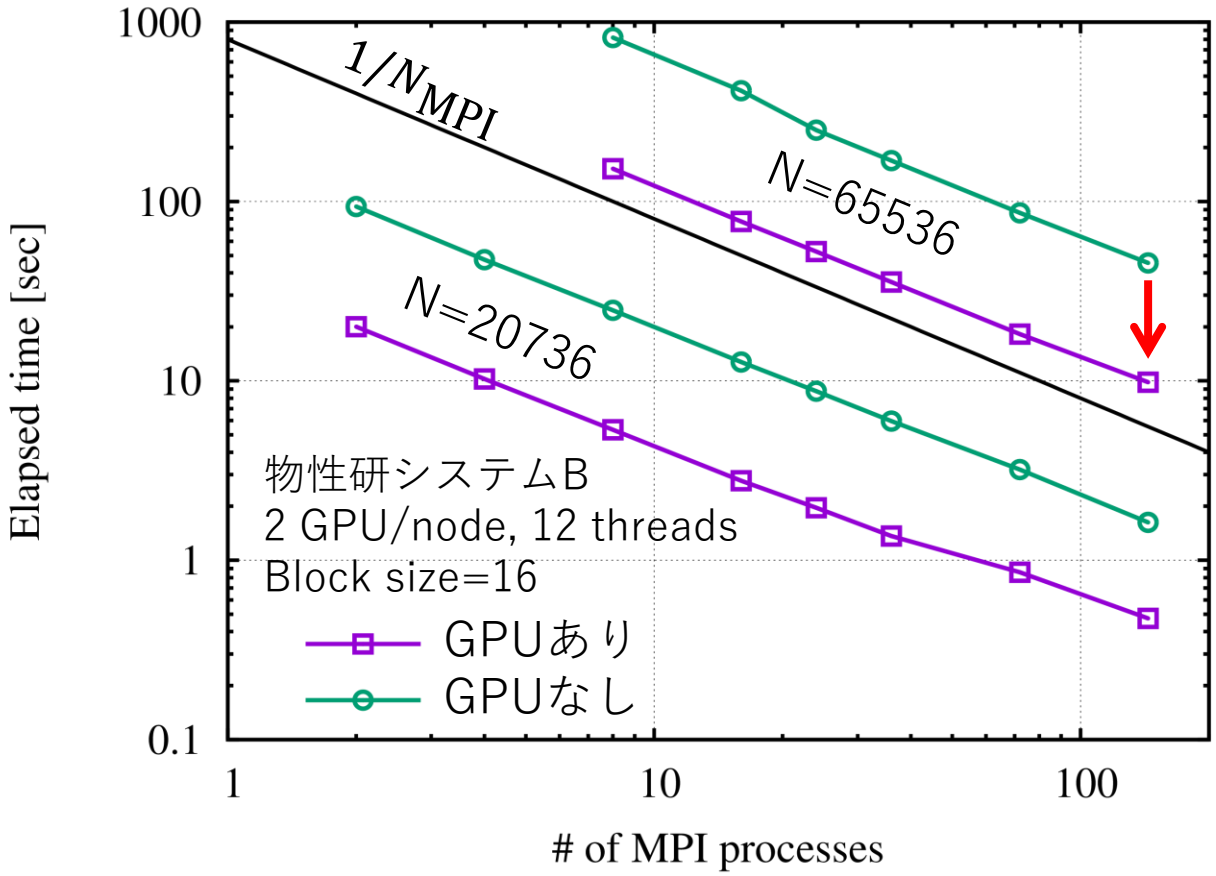
- 特異値分解 (pdgesvd)



システムBの場合，24～128が良い.

cuScaLAPACKの紹介

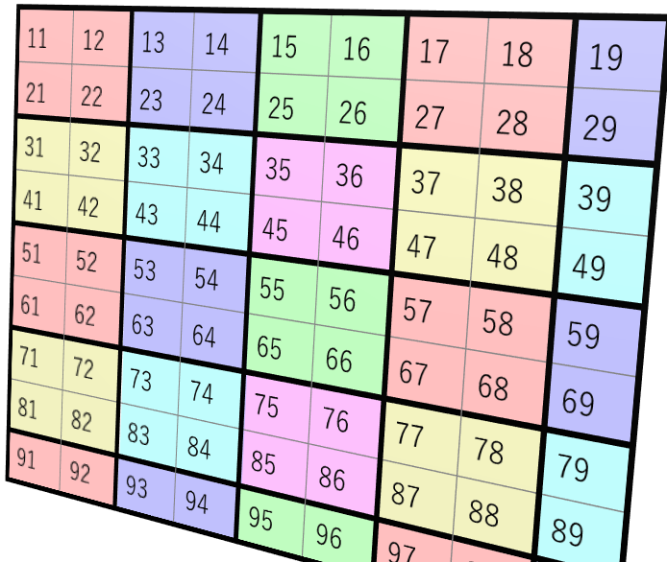
- GPUを利用した並列行列積演算 (pdgemm, pzgemm)
  - 物性研スパコン共同利用GPGPU化支援サービスの成果
  - 詳細はGithubへ https://github.com/smorita/cuscalapack



約 4.6 倍の高速化

まとめ

- ScaLAPACKはやっぱり使いにくい
  - 関数を呼び出すまでの事前準備が大変
    - ✓プロセスグリッドとBLACSのコンテキスト
    - ✓Block-cyclic distributionに従ったデータ分散
  - 準備が終われればLAPACKの関数を置き換えるのは簡単.
    - ただし，対応する関数がScaLAPACKに未実装の場合もある
  - ほぼ全ての場所で，利用環境が整備されているメリットは大きい.
  - 問題の種類によっては並列化効率がでない.
  - ブロックサイズで実行性能は大きく変わる



- 他の選択肢?
  - Elemental http://libelemental.org/
  - DPLASMA http://icl.utk.edu/dplasma/
  - 詳しい人がいたら，教えてください.