

並列数値処理の基本演算

東京大学情報基盤センター准教授 片桐孝洋

2015年10月13日(火) 10:25－12:10

スパコンプログラミング(1)、(I)



東京大学情報基盤センター
INFORMATION TECHNOLOGY CENTER, THE UNIVERSITY OF TOKYO

講義日程（工学部共通科目）

レポートおよびコンテスト課題
(締切:
2016年2月11日(木)24時 厳守

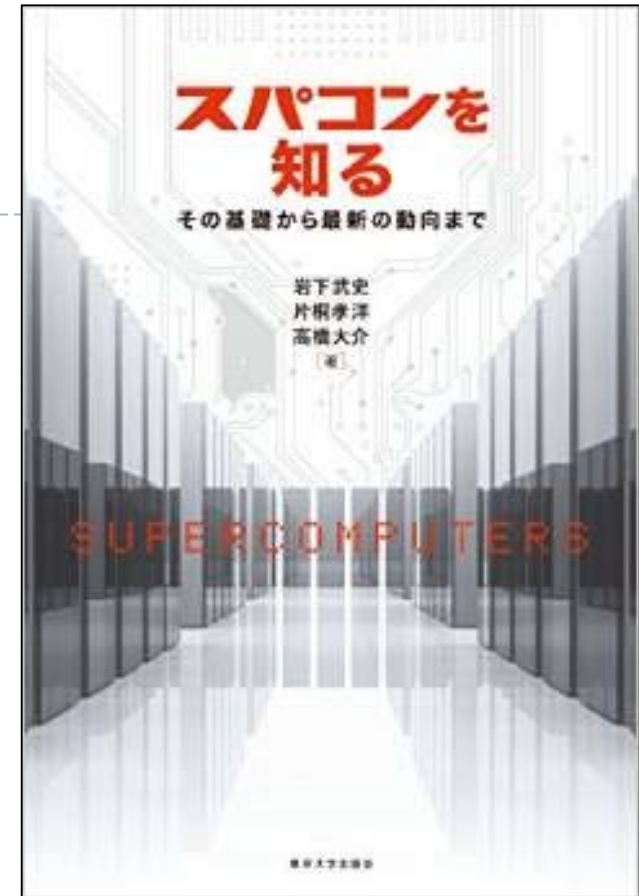
▶ ~~10月6日: ガイダンス~~

1. 10月13日
 - 並列数値処理の基本演算(座学)
2. 10月20日: スパコン利用開始
 - ログイン作業、テストプログラム実行
3. 10月27日
 - 高性能演算技法1
(ループアンローリング)
4. 11月10日
 - 高性能演算技法2
(キャッシュブロック化)
5. 11月24日
 - 行列-ベクトル積の並列化
6. 12月1日(8:30-10:15)
 - ベキ乗法の並列化

6. 12月1日(10:25-12:10)
 - 行列-行列積の並列化(1)
7. 12月8日
 - 行列-行列積の並列化(2)
8. 12月15日
 - LU分解法(1)
 - コンテスト課題発表
9. 12月22日
 - LU分解法(2)
10. 2016年1月5日
 - LU分解法(3)
12. 1月12日
 - ソフトウェア自動チューニング
 - 非同期通信

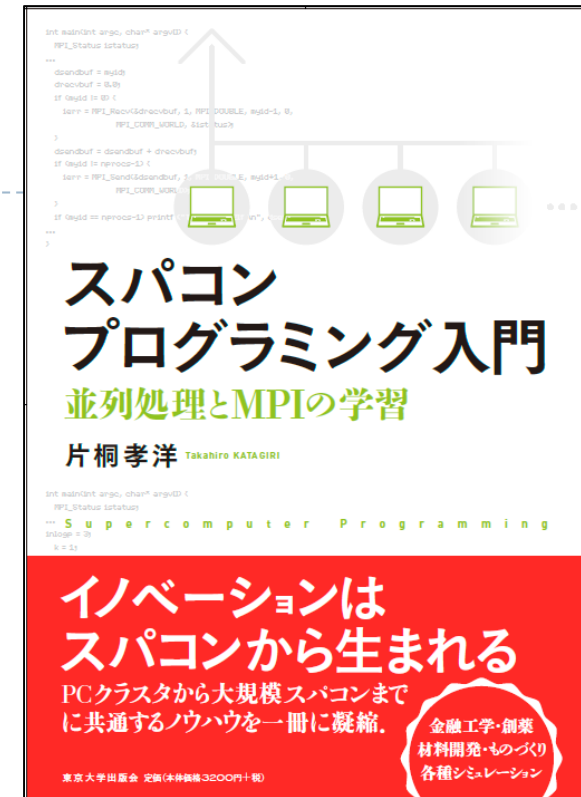
参考書

- ▶ 「スパコンを知る:
その基礎から最新の動向まで」
 - ▶ 岩下武史、片桐孝洋、高橋大介 著
 - ▶ 東大出版会、ISBN-10: 4130634550、
ISBN-13: 978-4130634557、
発売日: 2015年2月18日、176頁
 - ▶ 【本書の特徴】
 - ▶ スパコンの解説書です。以下を
分かりやすく解説しています。
 - スパコンは何に使えるか
 - スパコンはどんな仕組みで、なぜ速く計算できるのか
 - 最新技術、今後の課題と将来展望、など



教科書（演習書）

- ▶ 「スパコンプログラミング入門
ー並列処理とMPIの学習ー」
 - ▶ 片桐 孝洋 著、
 - ▶ 東大出版会、ISBN978-4-13-062453-4、
発売日:2013年3月12日、判型:A5, 200頁
 - ▶ 【本書の特徴】
 - ▶ C言語で解説
 - ▶ C言語、Fortran90言語のサンプルプログラムが付属
 - ▶ 数値アルゴリズムは、図でわかりやすく説明
 - ▶ 本講義の内容を全てカバー
 - ▶ 内容は初級。初めて並列数値計算を学ぶ人向けの入門書



本講義の流れ

1. 並列プログラミングの基礎
2. 性能評価指標
3. 基礎的なMPI関数
4. データ分散方式
5. ベクトルどうしの演算
6. ベクトル-行列積
7. リダクション演算
8. 数値計算ライブラリについて

並列プログラミングの基礎

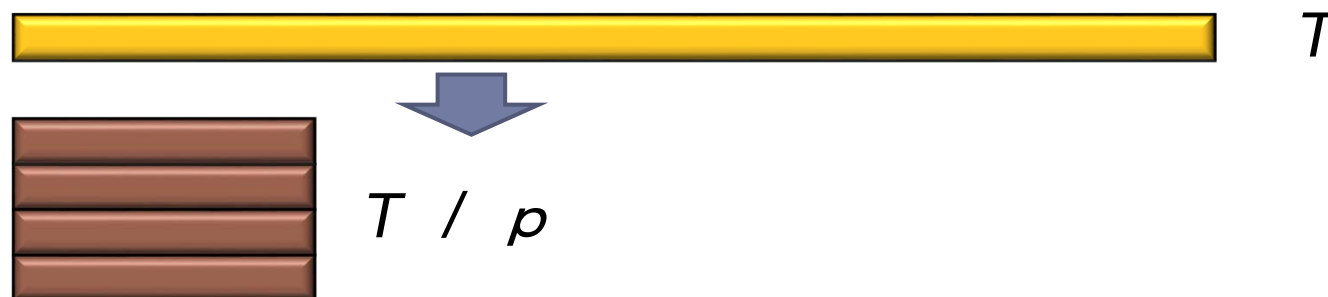
予備知識も重要

お願い

- ▶ 講義のとき、以下の反応をしてください
 - ▶ わからないとき
 - ▶ 質問する (←基本)
 - ▶ わからない顔をする
 - ▶ わかったとき
 - ▶ うなづく
- ▶ 反応がないと、「ぱわぽ」なので、
どんどん進んで行ってしまいます

並列プログラミングとは何か？

- ▶ 逐次実行のプログラム(実行時間 T)を、 p 台の計算機を使って、 T/p にすること。

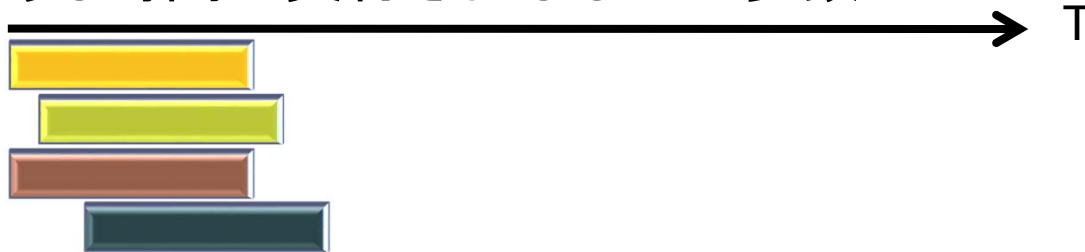


- ▶ 素人考えでは自明。
- ▶ 実際は、できるかどうかは、対象処理の内容(アルゴリズム)で **大きく** 難しさが違う
 - ▶ アルゴリズム上、絶対に並列化できない部分の存在
 - ▶ 通信のためのオーバーヘッドの存在
 - ▶ 通信立ち上がり時間
 - ▶ データ転送時間

並列と並行

▶ 並列 (Parallel)

- ▶ 物理的に並列 (時間的に独立)
- ▶ ある時間に実行されるものは多数



▶ 並行 (Concurrent)

- ▶ 論理的に並列 (時間的に依存)
- ▶ ある時間に実行されるものは1つ (= 1プロセッサで実行)



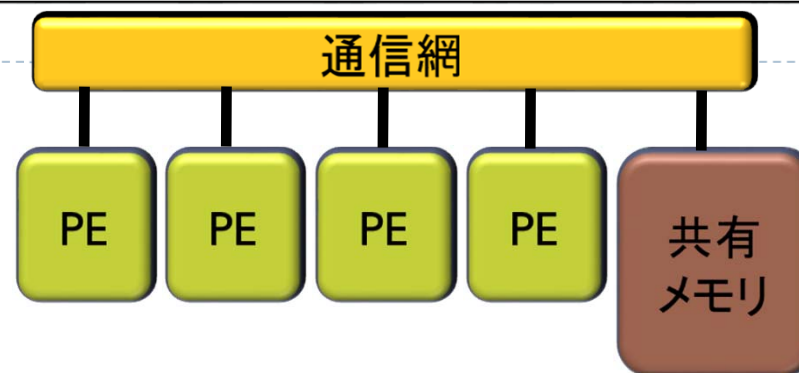
- ▶ 時分割多重、疑似並列
- ▶ OSによるプロセス実行スケジューリング (ラウンドロビン方式)

並列計算機の種類

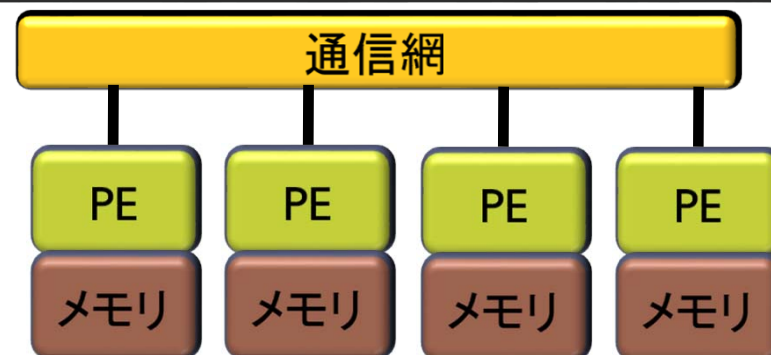
- ▶ Michael J. Flynn教授(スタンフォード大)の分類(1966)
- ▶ 単一命令・単一データ流
(SISD, Single Instruction Single Data Stream)
- ▶ 単一命令・複数データ流
(SIMD, Single Instruction Multiple Data Stream)
- ▶ 複数命令・単一データ流
(MISD, Multiple Instruction Single Data Stream)
- ▶ 複数命令・複数データ流
(MIMD, Multiple Instruction Multiple Data Stream)

並列計算機のメモリ型による分類

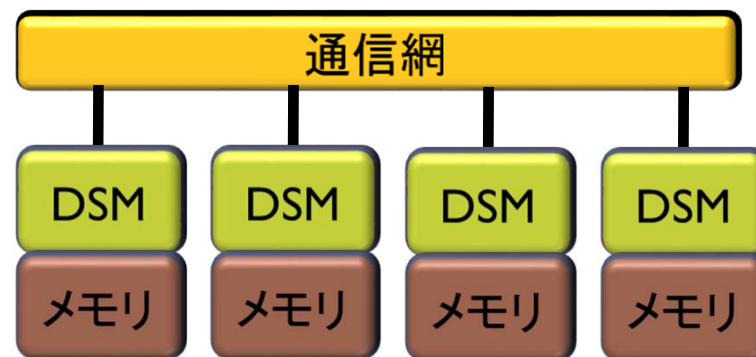
1. 共有メモリ型
(SMP、
Symmetric Multiprocessor)



2. 分散メモリ型
(メッセージパッシング)

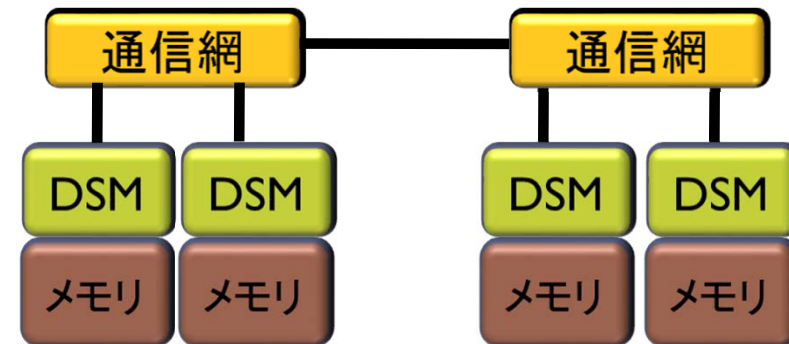


3. 分散共有メモリ型
(DSM、
Distributed Shared Memory)



並列計算機のメモリ型による分類

4. 共有・非対称メモリ型
(ccNUMA、
Cache Coherent Non-
Uniform Memory Access)

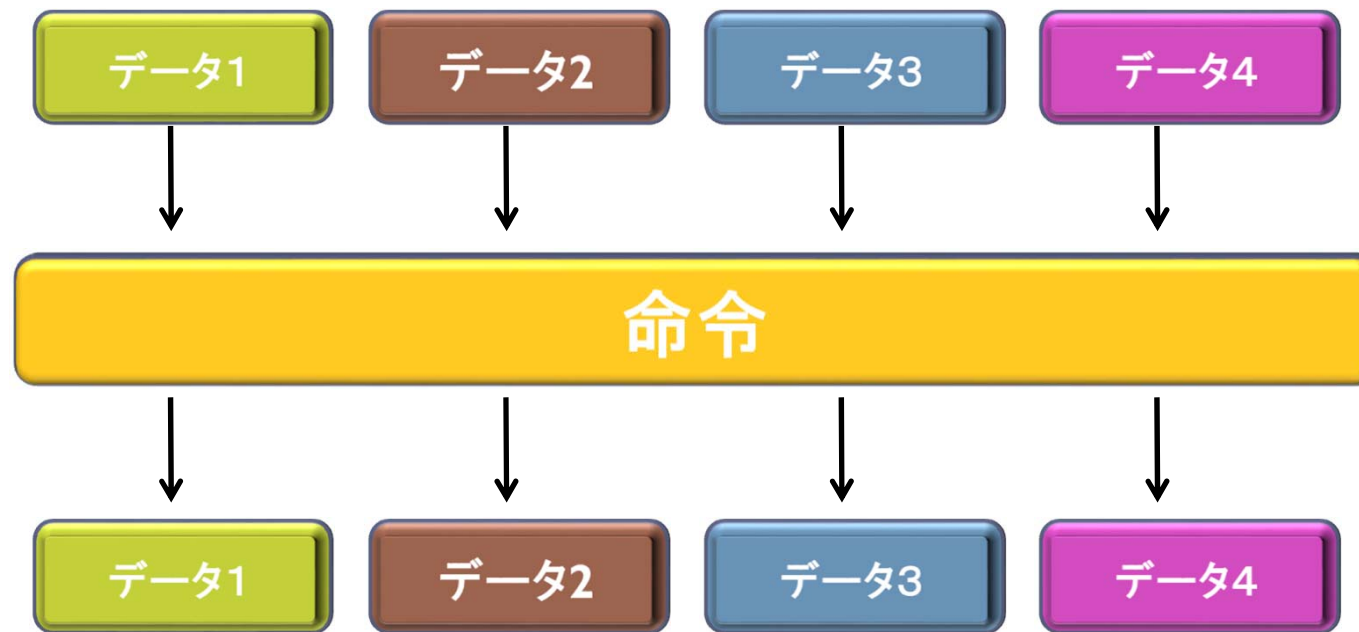


並列計算機の分類とMPIとの関係

- ▶ MPIは分散メモリ型計算機を想定
 - ▶ MPIは、分散メモリ間の通信を定めているため
- ▶ MPIは共有メモリ型計算機でも動く
 - ▶ MPIは、共有メモリ内でもプロセス間通信ができるため
- ▶ MPIを用いたプログラミングモデルは、
(基本的に)SIMD
 - ▶ MPIは、(基本的には)プログラムが1つ(=命令と等価)しかないが、データ(配列など)は複数あるため

並列プログラミングのモデル

- ▶ 実際の並列プログラムの挙動はMIMD
- ▶ アルゴリズムを考えるときは＜SIMDが基本＞
 - ▶ 複雑な挙動は理解できないので



並列プログラミングのモデル

▶ MIMD上での並列プログラミングのモデル

1. SPMD (Single Program Multiple Data)

- ▶ 1つの共通のプログラムが、並列処理開始時に、全プロセッサ上で起動する
- ▶ **MPI (バージョン1) のモデル**



2. Master / Worker (Master / Slave)

- ▶ 1つのプロセス (Master) が、複数のプロセス (Worker) を管理 (生成、消去) する。

並列プログラムの種類

▶ マルチプロセス

- ▶ **MPI (Message Passing Interface)**
- ▶ **HPF (High Performance Fortran)**
 - ▶ 自動並列化Fortranコンパイラ
 - ▶ ユーザがデータ分割方法を明示的に記述

プロセスとスレッドの違い

- メモリを意識するかどうかの違い
- 別メモリは「プロセス」
- 同一メモリは「スレッド」

▶ マルチスレッド

- ▶ Pthread (POSIX スレッド)
- ▶ Solaris Thread (Sun Solaris OS用)
- ▶ NT thread (Windows NT系、Windows95以降)
 - ▶ スレッドの Fork(分離) と Join(融合) を明示的に記述
- ▶ Java
 - ▶ 言語仕様としてスレッドを規定
- ▶ **OpenMP**
 - ▶ ユーザが並列化指示行を記述

マルチプロセスとマルチスレッドは
共存可能
→ハイブリッドMPI/OpenMP実行

並列処理の実行形態（１）

▶ データ並列

- ▶ データを分割することで並列化する。
- ▶ データの操作（＝演算）は同一となる。
- ▶ データ並列の例：行列－行列積

SIMDの
考え方と同じ

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \begin{pmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{pmatrix} = \begin{pmatrix} 1*9+2*6+3*3 & 1*8+2*5+3*2 & 1*7+2*4+3*1 \\ 4*9+5*6+6*3 & 4*8+5*5+6*2 & 4*7+5*4+6*1 \\ 7*9+8*6+9*3 & 7*8+8*5+9*2 & 7*7+8*4+9*1 \end{pmatrix}$$

● 並列化

全CPUで共有

CPU0	1 2 3	9 8 7 6 5 4 3 2 1	=	1*9+2*6+3*3 1*8+2*5+3*2 1*7+2*4+3*1
CPU1	4 5 6			4*9+5*6+6*3 4*8+5*5+6*2 4*7+5*4+6*1
CPU2	7 8 9			7*9+8*6+9*3 7*8+8*5+9*2 7*7+8*4+9*1

並列に計算：初期データは異なるが演算は同一

並列処理の実行形態（２）

▶ タスク並列

- ▶ タスク(ジョブ)を分割することで並列化する。
- ▶ データの操作(=演算)は異なるかもしれない。
- ▶ タスク並列の例: **カレーを作る**
 - ▶ 仕事1: 野菜を切る
 - ▶ 仕事2: 肉を切る
 - ▶ 仕事3: 水を沸騰させる
 - ▶ 仕事4: 野菜・肉を入れて煮込む
 - ▶ 仕事5: カレールーを入れる

● 並列化



MPIの特徴

- ▶ **メッセージパッシング用のライブラリ規格の1つ**
 - ▶ メッセージパッシングのモデルである
 - ▶ コンパイラの規格、特定のソフトウェアやライブラリを指すものではない！
- ▶ 分散メモリ型並列計算機で並列実行に向く
- ▶ 大規模計算が可能
 - ▶ 1プロセッサにおけるメモリサイズやファイルサイズの制約を打破可能
 - ▶ プロセッサ台数の多い並列システム(MPPシステム、Massively Parallel Processingシステム)を用いる実行に向く
 - ▶ 1プロセッサ換算で膨大な実行時間の計算を、短時間で処理可能
 - ▶ 移植が容易
 - ▶ **API(Application Programming Interface)の標準化**
- ▶ スケーラビリティ、性能が高い
 - ▶ 通信処理をユーザが記述することによるアルゴリズムの最適化が可能
 - ▶ プログラミングが難しい(敷居が高い)

MPIの経緯 (1/2)

- ▶ MPIフォーラム (<http://www.mpi-forum.org/>) が仕様策定
 - ▶ 1994年5月1.0版(MPI-1)
 - ▶ 1995年6月1.1版
 - ▶ 1997年7月1.2版、および 2.0版(MPI-2)
- ▶ 米国アルゴンヌ国立研究所、およびミシシッピ州立大学で開発
- ▶ MPI-2 では、以下を強化：
 - ▶ 並列I/O
 - ▶ C++、Fortran 90用インターフェース
 - ▶ 動的プロセス生成/消滅
 - ▶ 主に、並列探索処理などの用途

MPIの経緯 MPI3.0策定

- ▶ 以下のページで経緯・ドキュメントを公開中
 - ▶ http://meetings.mpi-forum.org/MPI_3.0_main_page.php
 - ▶ <http://meetings.mpi-forum.org/mpi3-impl-status-Nov14.pdf>
(Implementation Status, as of November 2014)
- ▶ 注目すべき機能
 - ▶ ノン・ブロッキングの集団通信機能
(MPI_IALLREDUCE、など)
 - ▶ 片方向通信 (RMA、Remote Memory Access)
 - ▶ Fortran2008 対応、など

MPIの経緯 MPI4.0策定

- ▶ 以下のページで経緯・ドキュメントを公開中
 - ▶ http://meetings.mpi-forum.org/MPI_4.0_main_page.php
- ▶ 検討されている機能
 - ▶ ハイブリッドプログラミングへの対応
 - ▶ MPIアプリケーションの耐故障性 (Fault Tolerance, FT)
 - ▶ いくつかのアイデアを検討中
 - ▶ Active Messages (メッセージ通信のプロトコル)
 - 計算と通信のオーバーラップ
 - 最低限の同期を用いた非同期通信
 - 低いオーバーヘッド、パイプライン転送
 - バッファリングなしで、インタラプトハンドラで動く
 - ▶ Stream Messaging
 - ▶ 新プロファイル・インターフェース

MPIの実装

- ▶ **MPICH(エム・ピッチ)**
 - ▶ 米国アルゴンヌ国立研究所が開発
- ▶ LAM(Local Area Multicomputer)
 - ▶ ノートルダム大学が開発
- ▶ その他
 - ▶ OpenMPI (FT-MPI、LA-MPI、LAM/MPI、PACX-MPIの統合プロジェクト)
 - ▶ YAMPII(東大・石川研究室)
(SCore通信機構をサポート)
- ▶ 注意点:メーカー独自機能拡張がなされていることがある

MPIによる通信

- ▶ 郵便物の郵送に同じ
- ▶ 郵送に必要な情報:
 1. 自分の住所、送り先の住所
 2. 中に入っているものはどこにあるか
 3. 中に入っているものの分類
 4. 中に入っているものの量
 5. (荷物を複数同時に送る場合の)認識方法(タグ)
- ▶ MPIでは:
 1. 自分の認識ID、および、送り先の認識ID
 2. データ格納先のアドレス
 3. データ型
 4. データ量
 5. タグ番号

MPI関数

▶ システム関数

- ▶ MPI_Init; MPI_Comm_rank; MPI_Comm_size; MPI_Finalize;

▶ 1対1通信関数

▶ ブロッキング型

- ▶ MPI_Send; MPI_Recv;

▶ ノンブロッキング型

- ▶ MPI_Isend; MPI_Irecv;

▶ 1対全通信関数

- ▶ MPI_Bcast

▶ 集団通信関数

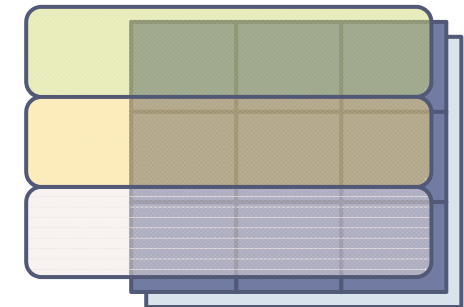
- ▶ MPI_Reduce; MPI_Allreduce; MPI_Barrier;

▶ 時間計測関数

- ▶ MPI_Wtime

コミュニケータ

- ▶ MPI_COMM_WORLDは、**コミュニケータ**とよばれる概念を保存する変数
- ▶ コミュニケータは、操作を行う対象のプロセッサ群を定める
- ▶ 初期状態では、**0番～numprocs - 1番**までのプロセッサが、1つのコミュニケータに割り当てられる
 - ▶ この名前が、“**MPI_COMM_WORLD**”
- ▶ プロセッサ群を分割したい場合、**MPI_Comm_split** 関数を利用
 - ▶ メッセージを、一部のプロセッサ群に放送するときに利用
 - ▶ “マルチキャスト”で利用



性能評価指標

並列化の尺度

性能評価指標－台数効果

▶ 台数効果

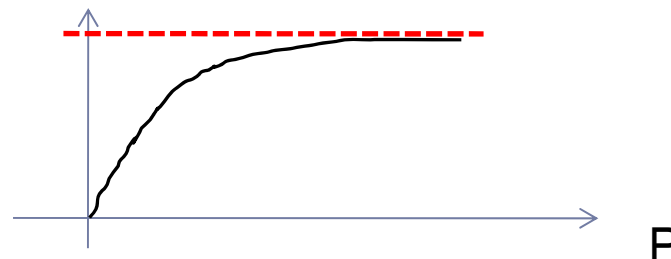
- ▶ 式: $S_p = T_s / T_p$ ($0 \leq S_p$)
- ▶ T_s : 逐次の実行時間、 T_p : P台での実行時間
- ▶ P台用いて $S_p = P$ のとき、理想的な(ideal)速度向上
- ▶ P台用いて $S_p > P$ のとき、スーパリニア・スピードアップ
 - ▶ 主な原因は、並列化により、データアクセスが局所化されて、キャッシュヒット率が向上することによる高速化

▶ 並列化効率

- ▶ 式: $E_p = S_p / P \times 100$ ($0 \leq E_p$) [%]

▶ 飽和性能

- ▶ 速度向上の限界
- ▶ Saturation、「さちる」



アムダールの法則

- ▶ 逐次実行時間を K とする。
そのうち、並列化ができる割合を α とする。
- ▶ このとき、台数効果は以下ようになる。

$$\begin{aligned} S_p &= K / (K\alpha / P + K(1-\alpha)) \\ &= 1 / (\alpha / P + (1-\alpha)) = 1 / (\alpha(1/P - 1) + 1) \end{aligned}$$

- ▶ 上記の式から、たとえ無限大の数のプロセッサを使っても ($P \rightarrow \infty$)、台数効果は、高々 $1 / (1 - \alpha)$ である。

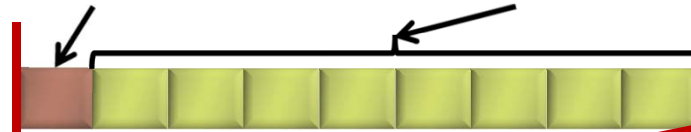
(アムダールの法則)

- ▶ 全体の90%が並列化できたとしても、無限大の数のプロセッサをつかっても、 $1 / (1 - 0.9) = 10$ 倍 にしかない！
→ 高性能を達成するためには、少しでも並列化効率を上げる
実装をすることがとても重要である

アムダールの法則の直観例

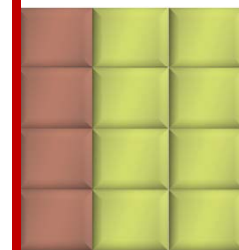
並列化できない部分(1ブロック) 並列化できる部分(8ブロック)

● 逐次実行



=88.8%が並列化可能

● 並列実行(4並列)



$9/3=3$ 倍

● 並列実行(8並列)

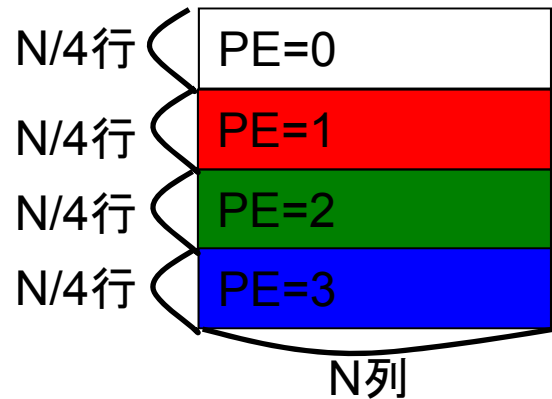


$9/2=4.5$ 倍 $\neq$ 6倍

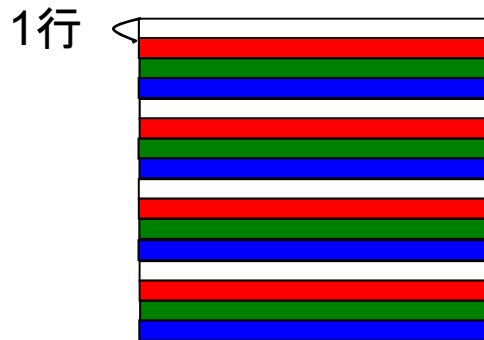
基本演算

- ▶ 逐次処理では、「データ構造」が重要
- ▶ 並列処理においては、「データ分散方法」が重要になる！
 1. 各PEの「演算負荷」を均等にする
 - ▶ ロード・バランシング： 並列処理の基本操作の一つ
 - ▶ 粒度調整
 2. 各PEの「利用メモリ量」を均等にする
 3. 演算に伴う通信時間を短縮する
 4. 各PEの「データ・アクセスパターン」を高速な方式にする
(＝逐次処理におけるデータ構造と同じ)
- ▶ 行列データの分散方法
 - ▶ <次元レベル>： 1次元分散方式、2次元分散方式
 - ▶ <分割レベル>： ブロック分割方式、サイクリック(循環)分割方式

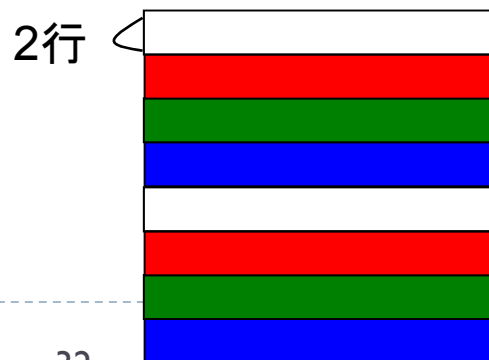
1次元分散



- (行方向) ブロック分割方式
- (Block, *) 分散方式



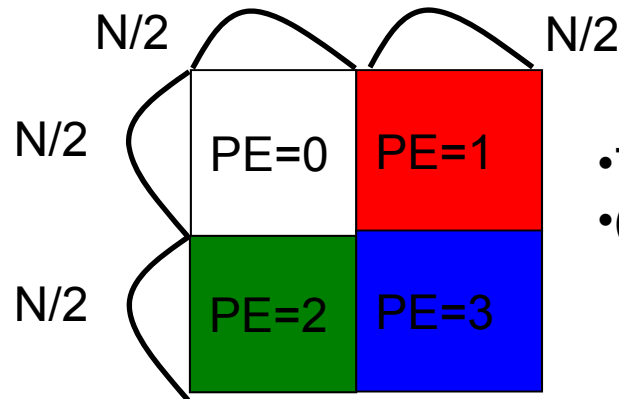
- (行方向) サイクリック分割方式
- (Cyclic, *) 分散方式



- (行方向) ブロック・サイクリック分割方式
- (Cyclic(2), *) 分散方式

この例の「2」: <ブロック幅>とよぶ

2次元分散



- ブロック・ブロック分割方式
- (Block, Block)分散方式

- サイクリック・サイクリック分割方式
- (Cyclic, Cyclic)分散方式

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

- 二次元ブロック・サイクリック分割方式
- (Cyclic(2), Cyclic(2))分散方式

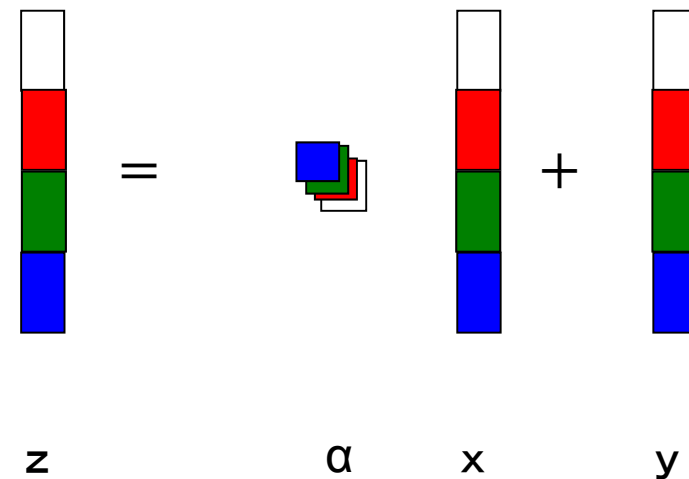
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3

ベクトルどうしの演算

- ▶ 以下の演算

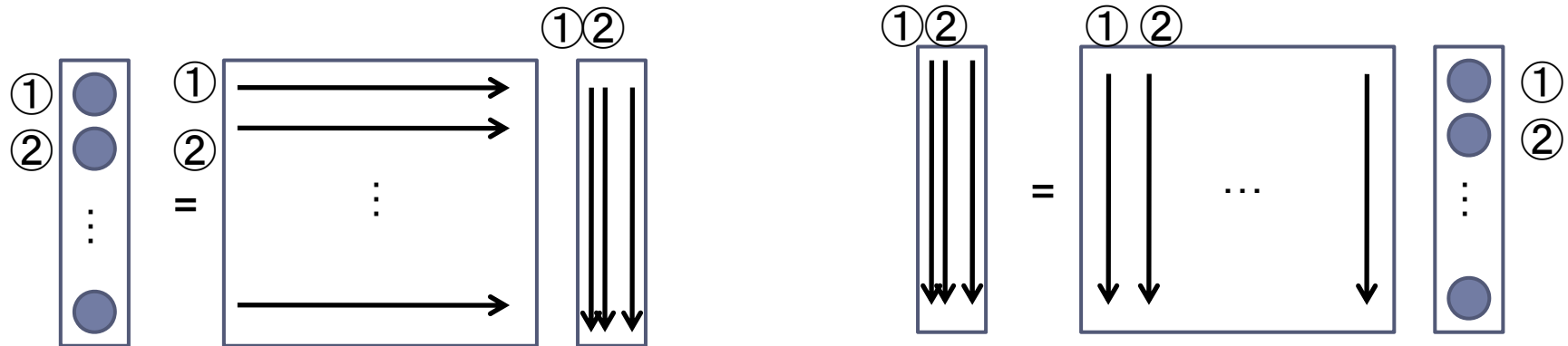
$$Z = \alpha X + y$$

- ▶ ここで、 α はスカラ、 z 、 x 、 y はベクトル
- ▶ どのようなデータ分散方式でも並列処理が可能
 - ▶ ただし、スカラ α は全PEで所有する。
 - ▶ ベクトルは $O(n)$ のメモリ領域が必要なのに対し、スカラは $O(1)$ のメモリ領域で大丈夫。
→スカラメモリ領域は無視可能
- ▶ 計算量： $O(N/P)$
- ▶ あまり面白くない



行列とベクトルの積

- ▶ **＜行方式＞**と**＜列方式＞**がある。
- ▶ **＜データ分散方式＞**と**＜方式＞**組のみ合わせがあり、少し面白い



```
for (i=0; i<n; i++) {  
    y[i]=0.0;  
    for (j=0; j<n; j++) {  
        y[i] += a[i][j]*x[j];  
    }  
}
```

＜行方式＞： 自然な実装
C言語向き

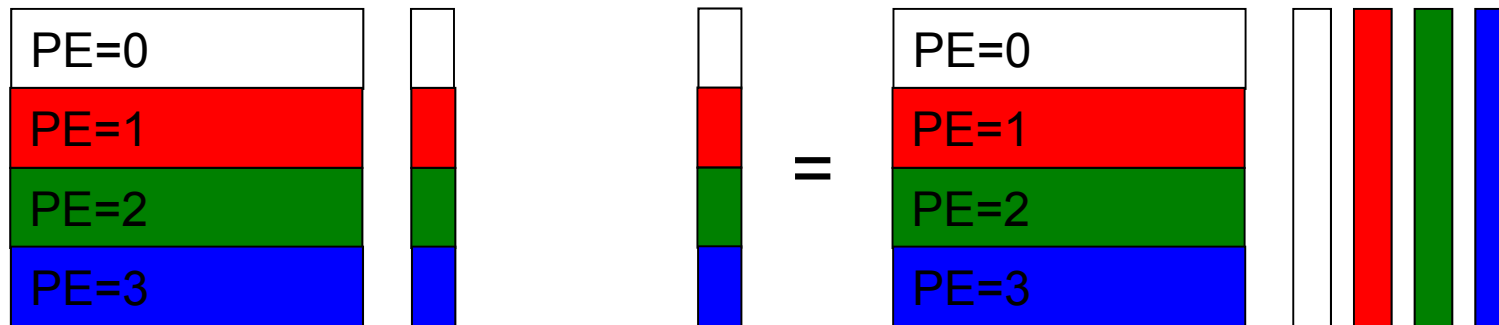
```
for (j=0; j<n; j++) y[j]=0.0;  
for (j=0; j<n; j++) {  
    for (i=0; i<n; i++) {  
        y[i] += a[i][j]*x[j];  
    }  
}
```

＜列方式＞： Fortran言語向き

行列とベクトルの積

＜行方式の場合＞

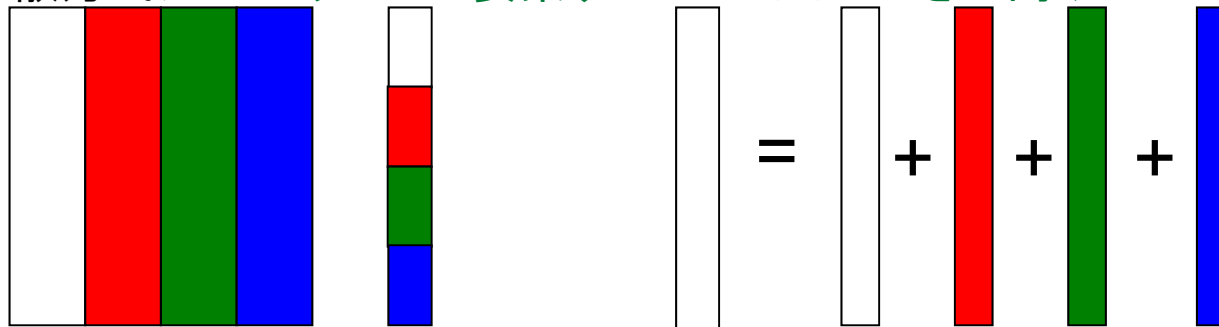
＜行方向分散方式＞ : 行方式に向く分散方式



右辺ベクトルを `MPI_Allgather` 関数
を利用し、全PEで所有する

各PE内で行列ベクトル積を行う

＜列方向分散方式＞ : ベクトルの要素すべてがほしいときに向く



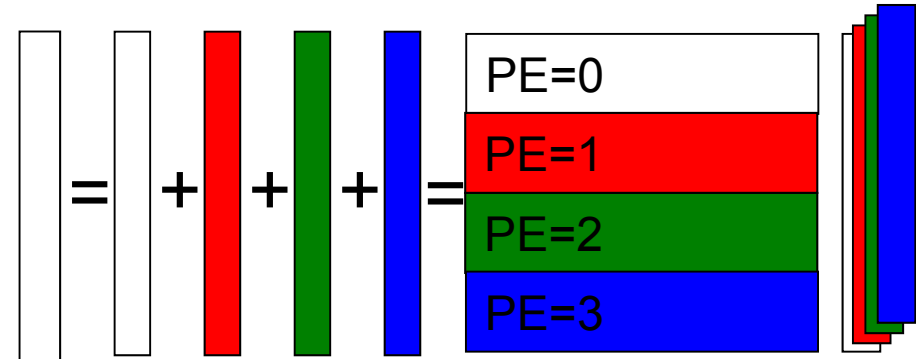
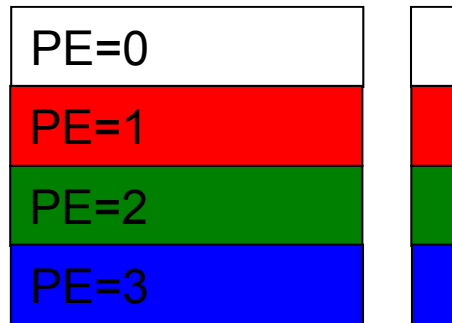
各PE内で行列-ベクトル積
を行う

`MPI_Reduce` 関数で総和を求める
(※ある1PEにベクトルすべてが集まる)

行列とベクトルの積

＜列方式の場合＞

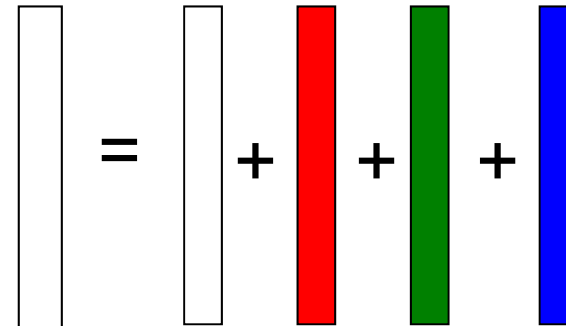
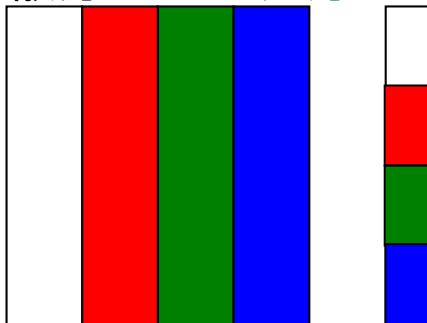
＜行方向分散方式＞ : 無駄が多く使われない



右辺ベクトルを `MPI_Allgather` 関数
を利用して、全PEで所有する

結果を `MPI_Reduce` 関数により
総和を求める

＜列方向分散方式＞ : 列方式に向く分散方式



各PE内で行列-ベクトル積
を行う

`MPI_Reduce` 関数で総和を求める
(※ある1PEにベクトルすべてが集まる)

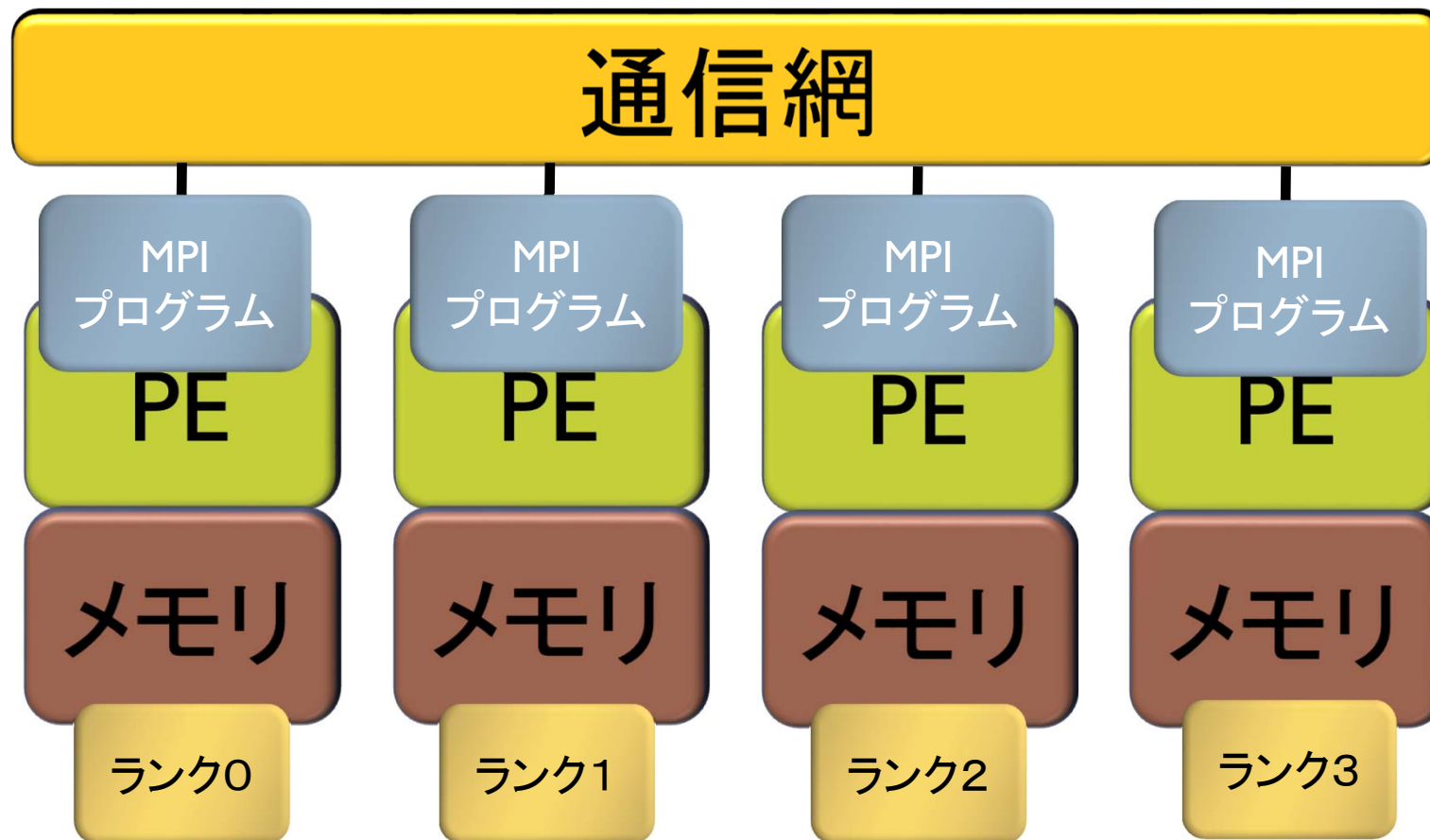
基本的なMPI関数

送信、受信のためのインタフェース

略語とMPI用語

- ▶ MPIは「プロセス」間の通信を行います。
- ▶ プロセスは、HT(ハイパースレッド)などを使わなければ、「プロセッサ」(もしくは、コア)に1対1で割り当てられます。
- ▶ 今後、「MPIプロセス」と書くのは長いので、ここではPE(Processor Elementsの略)と書きます。
 - ▶ ただし用語として「PE」は、現在あまり使われていません。
- ▶ ランク(Rank)
 - ▶ 各「MPIプロセス」の「識別番号」のこと。
 - ▶ 通常MPIでは、MPI_Comm_rank関数で設定される変数(サンプルプログラムではmyid)に、0～全PE数－1 の数値が入る
 - ▶ 世の中の全MPIプロセス数を知るには、MPI_Comm_size関数を使う。(サンプルプログラムでは、numprocs に、この数値が入る)

リンクの説明図



C言語インターフェースと Fortranインターフェースの違い

- ▶ C版は、 整数変数*ierr* が戻り値

`ierr = MPI_Xxxx(...);`

- ▶ Fortran版は、最後に整数変数*ierr*が引数

`call MPI_XXXX(..., ierr)`

- ▶ システム用配列の確保の仕方

- ▶ C言語

`MPI_Status istatus;`

- ▶ Fortran言語

`integer istatus(MPI_STATUS_SIZE)`

C言語インターフェースと Fortranインターフェースの違い

▶ MPIにおける、データ型の指定

□ C言語

MPI_CHAR (文字型)、**MPI_INT** (整数型)、
MPI_FLOAT (実数型)、**MPI_DOUBLE**(倍精度実数型)

□ Fortran言語

MPI_CHARACTER (文字型)、**MPI_INTEGER** (整数型)、
MPI_REAL (実数型)、**MPI_DOUBLE_PRECISION**(倍精度実数型)、**MPI_COMPLEX**(複素数型)

▶ 以降は、C言語インタフェースで説明する

基礎的なMPI関数—MPI_Recv (1 / 2)

```
▶ ierr = MPI_Recv(recvbuf, icount, idatatype, isource,  
                 itag,  icomm, istatus);
```

- ▶ **recvbuf** : 受信領域の先頭番地を指定する。
- ▶ **icount** : 整数型。受信領域のデータ要素数を指定する。
- ▶ **idatatype** : 整数型。受信領域のデータの型を指定する。
 - ▶ **MPI_CHAR** (文字型)、**MPI_INT** (整数型)、**MPI_FLOAT** (実数型)、**MPI_DOUBLE** (倍精度実数型)
- ▶ **isource** : 整数型。受信したいメッセージを送信するPEのランクを指定する。
 - ▶ 任意のPEから受信したいときは、**MPI_ANY_SOURCE** を指定する。

基礎的なMPI関数—MPI_Recv (2 / 2)

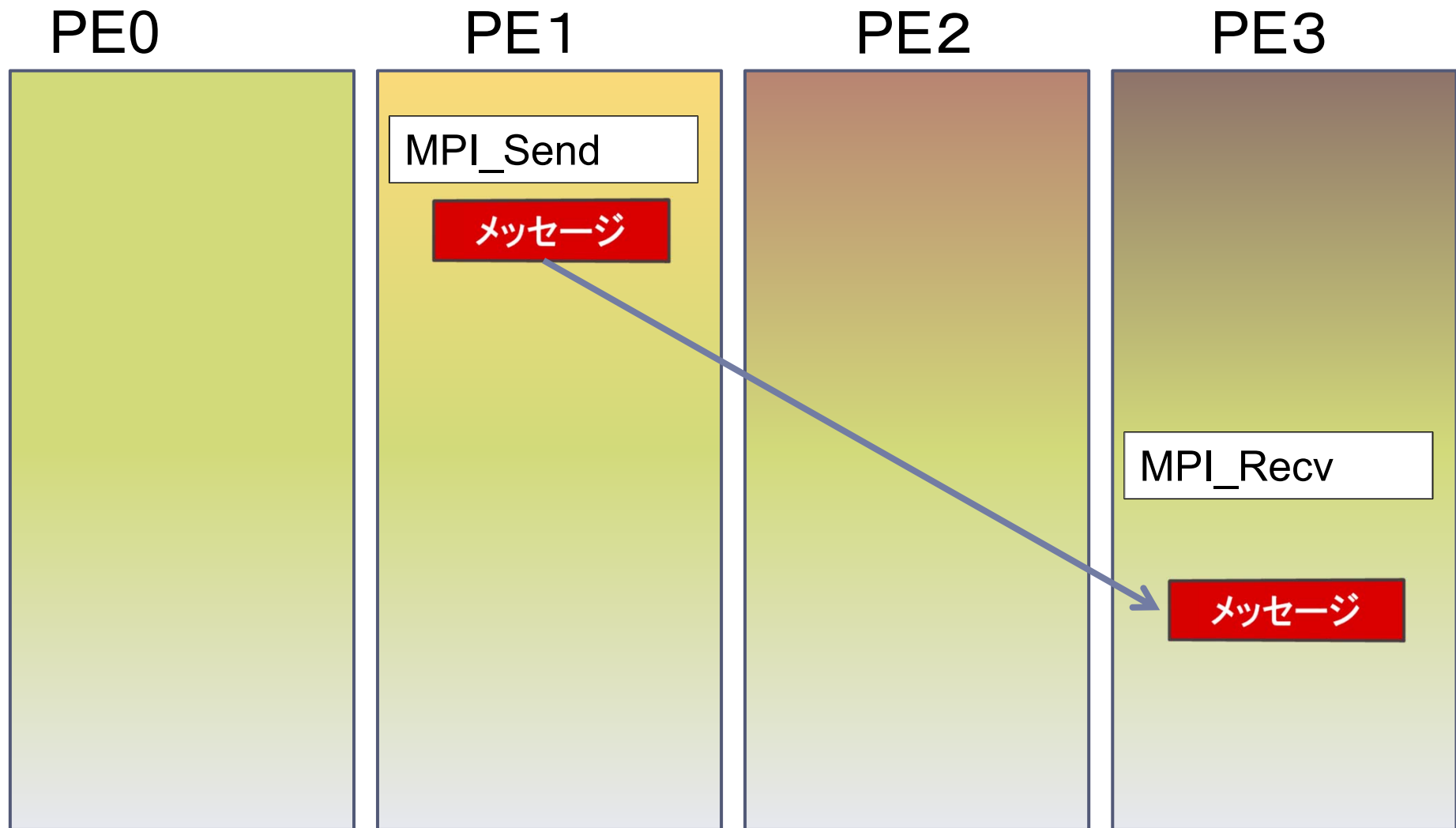
- ▶ **itag** : 整数型。受信したいメッセージに付いているタグの値を指定。
 - ▶ 任意のタグ値のメッセージを受信したいときは、**MPI_ANY_TAG** を指定。
- ▶ **icomm** : 整数型。PE集団を認識する番号であるコミュニケータを指定。
 - ▶ 通常では**MPI_COMM_WORLD** を指定すればよい。
- ▶ **istatus** : MPI_Status型(整数型の配列)。受信状況に関する情報が入る。**かならず専用の型宣言をした配列を確保すること。**
 - ▶ 要素数が**MPI_STATUS_SIZE**の整数配列が宣言される。
 - ▶ 受信したメッセージの送信元のランクが **istatus[MPI_SOURCE]**、タグが **istatus[MPI_TAG]** に代入される。
 - ▶ **C言語**: **MPI_Status istatus;**
 - ▶ **Fortran言語**: **integer istatus(MPI_STATUS_SIZE)**
- ▶ **ierr(戻り値)** : 整数型。エラーコードが入る。

基礎的なMPI関数—MPI_Send

```
▶ ierr = MPI_Send(sendbuf, icount, idatatype, idest,  
    itag,  icomm);
```

- ▶ **sendbuf** : 送信領域の先頭番地を指定
- ▶ **icount** : 整数型。送信領域のデータ要素数を指定
- ▶ **idatatype** : 整数型。送信領域のデータの型を指定
- ▶ **idest** : 整数型。送信したいPEのicomm内でのランクを指定
- ▶ **itag** : 整数型。受信したいメッセージに付けられたタグの値を指定
- ▶ **icomm** : 整数型。プロセッサ集団を認識する番号である
 コミュニケータを指定
- ▶ **ierr (戻り値)** : 整数型。エラーコードが入る。

Send-Recvの概念 (1対1通信)

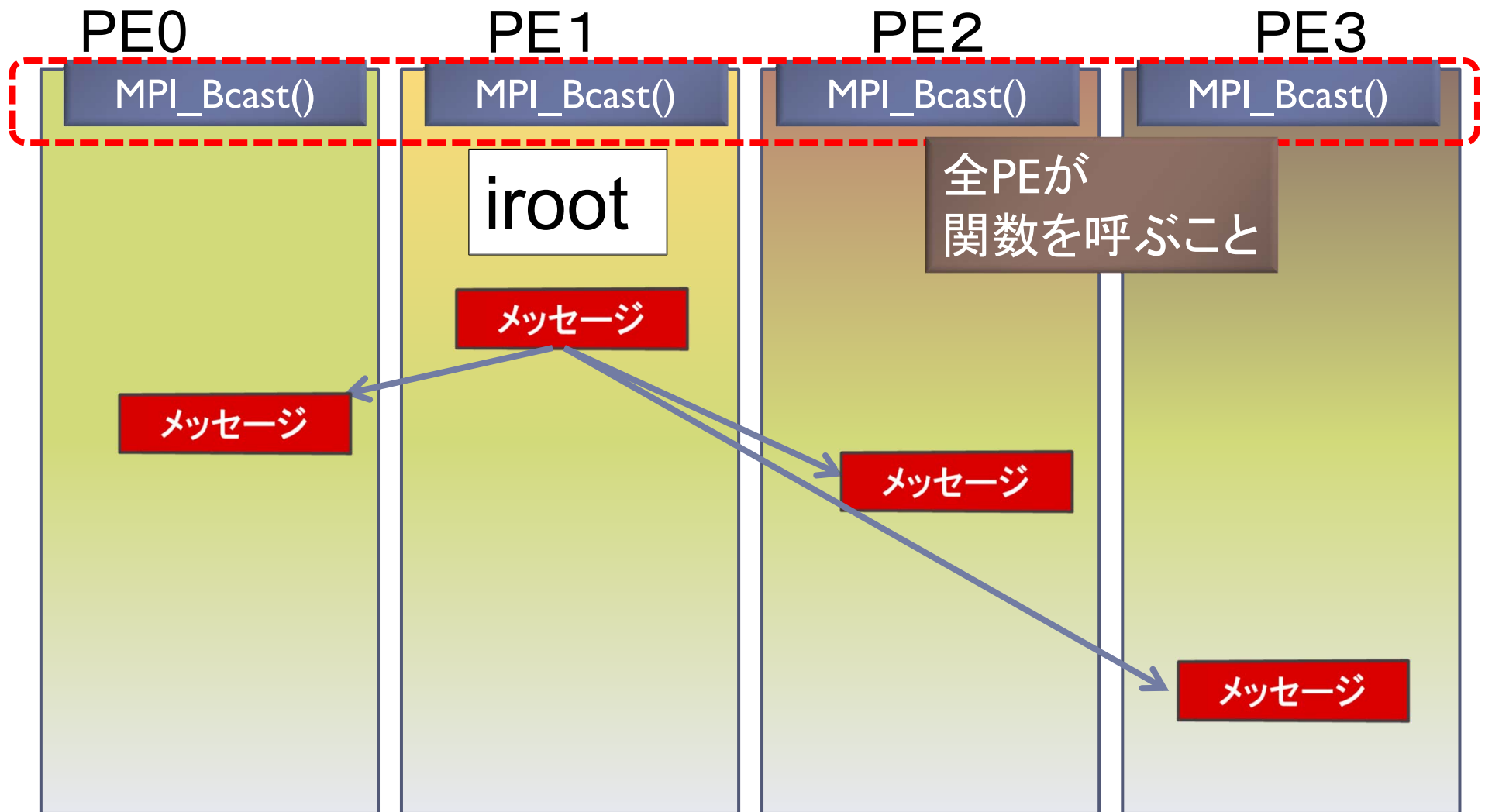


基礎的なMPI関数—MPI_Bcast

```
▶ ierr = MPI_Bcast(sendbuf, icount, idatatype,  
                    iroot, ictop);
```

- ▶ **sendbuf** : 送信および受信領域の先頭番地を指定する。
- ▶ **icount** : 整数型。送信領域のデータ要素数を指定する。
- ▶ **idatatype** : 整数型。送信領域のデータの型を指定する。
- ▶ **iroot** : 整数型。送信したいメッセージがあるPEの番号を指定する。全PEで同じ値を指定する必要がある。
- ▶ **ictop** : 整数型。PE集団を認識する番号である
コミュニケータを指定する。
- ▶ **ierr (戻り値)** : 整数型。エラーコードが入る。

MPI_Bcastの概念 (集団通信)



リダクション演算

- ▶ <操作>によって<次元>を減少
(リダクション)させる処理
 - ▶ 例: 内積演算
ベクトル(n 次元空間) $\rightarrow$ スカラ(1次元空間)
- ▶ リダクション演算は、通信と計算を必要とする
 - ▶ 集団通信演算 (collective communication operation)
と呼ばれる
- ▶ 演算結果の持ち方の違いで、2種の
インターフェースが存在する

リダクション演算

- ▶ 演算結果に対する所有PEの違い

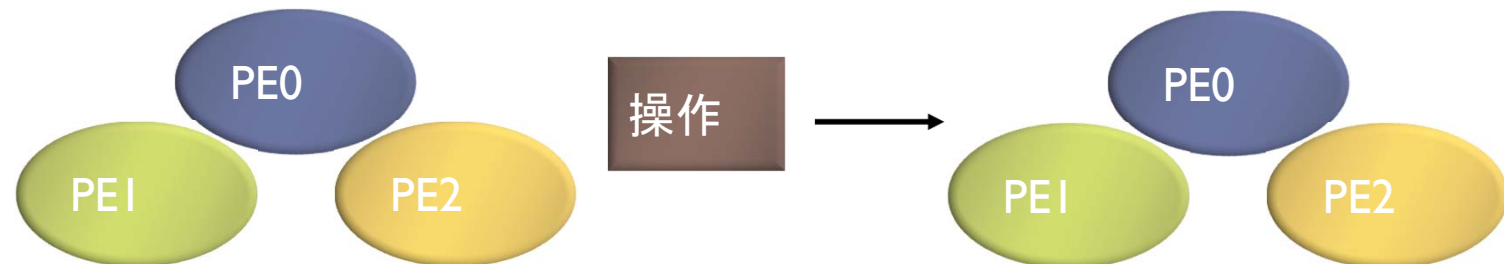
- ▶ **MPI_Reduce**関数

- ▶ リダクション演算の結果を、ある一つのPEに所有させる



- ▶ **MPI_Allreduce**関数

- ▶ リダクション演算の結果を、全てのPEに所有させる



基礎的なMPI関数—MPI_Reduce

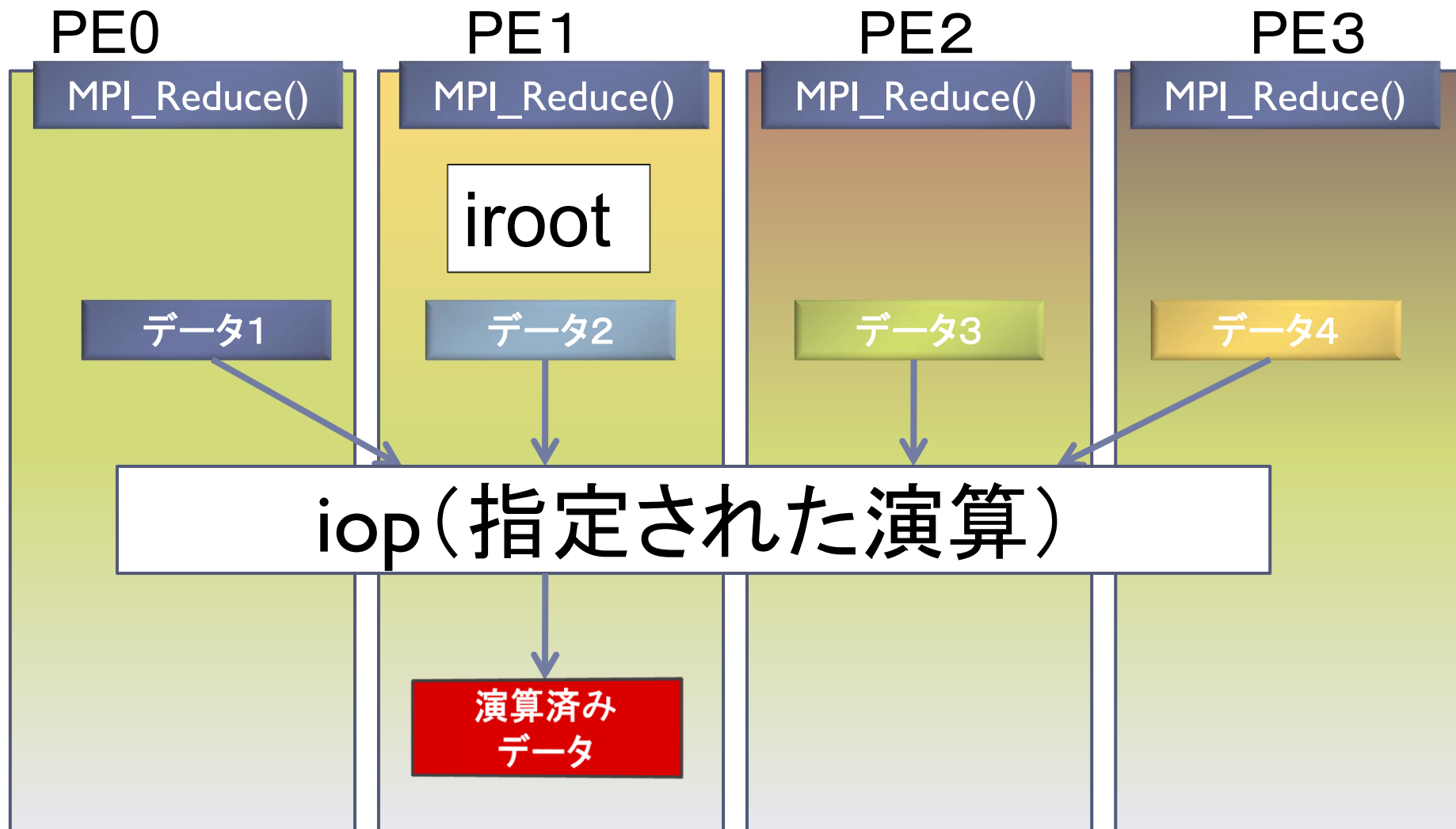
```
▶ ierr = MPI_Reduce(sendbuf, recvbuf, icount,  
    idatatype, iop, iroot, ictop);
```

- ▶ **sendbuf** : 送信領域の先頭番地を指定する。
- ▶ **recvbuf** : 受信領域の先頭番地を指定する。iroot で指定したPEのみで書き込みがなされる。
送信領域と受信領域は、同一であってはならない。
すなわち、異なる配列を確保しなくてはならない。
- ▶ **icount** : 整数型。送信領域のデータ要素数を指定する。
- ▶ **idatatype** : 整数型。送信領域のデータの型を指定する。
 - ▶ (Fortran) <最小／最大値と位置>を返す演算を指定する場合は、**MPI_2INTEGER**(整数型)、**MPI_2REAL**(単精度型)、**MPI_2DOUBLE_PRECISION**(倍精度型)、を指定する。

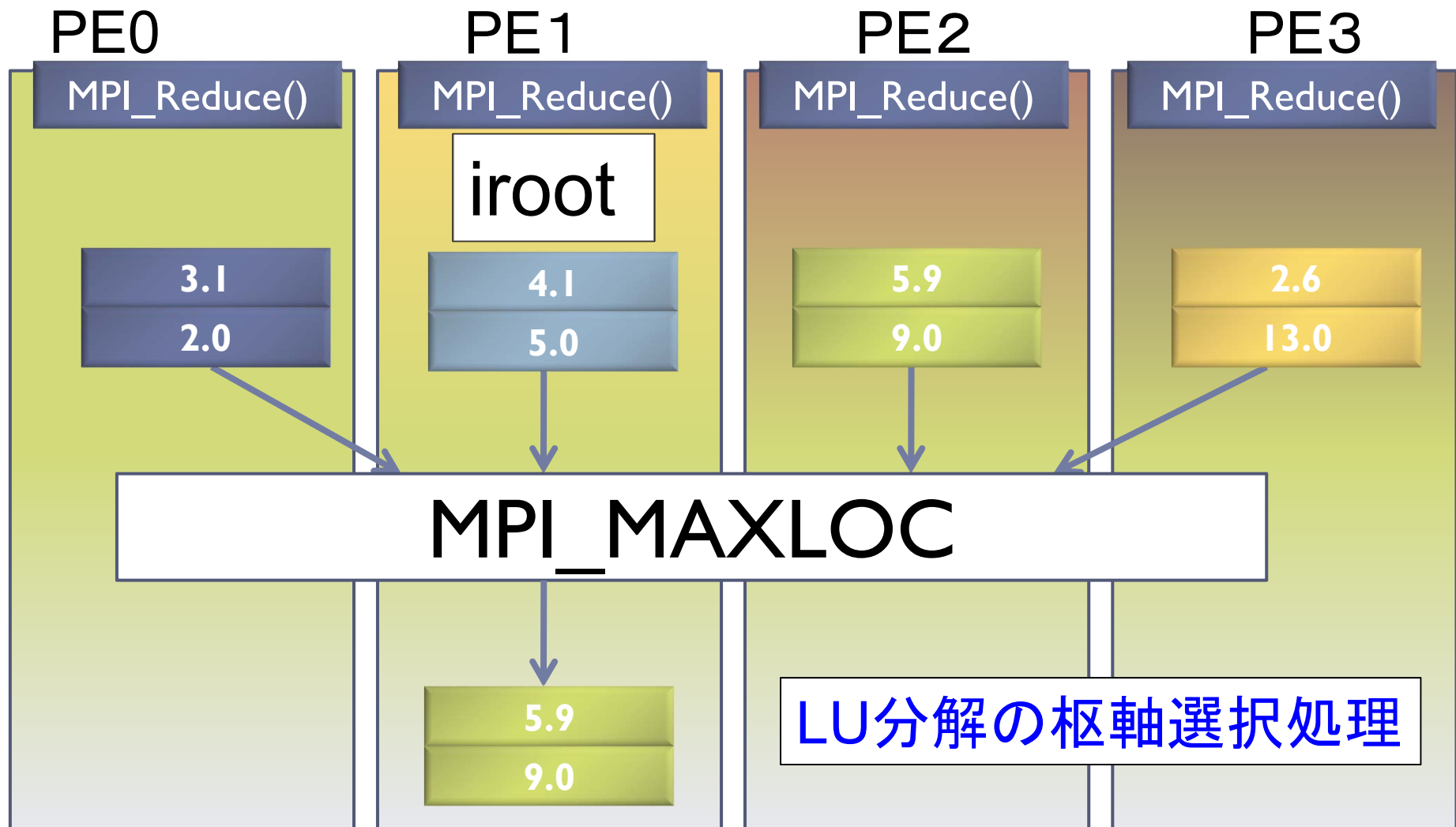
基礎的なMPI関数—MPI_Reduce

- ▶ **iop** : 整数型。演算の種類を指定する。
 - ▶ **MPI_SUM** (総和)、**MPI_PROD** (積)、**MPI_MAX** (最大)、**MPI_MIN** (最小)、**MPI_MAXLOC** (最大と位置)、**MPI_MINLOC** (最小と位置) など。
- ▶ **iroot** : 整数型。結果を受け取るPEのicomm 内でのランクを指定する。全てのicomm 内のPEで同じ値を指定する必要がある。
- ▶ **icomm** : 整数型。PE集団を認識する番号であるコミュニケータを指定する。
- ▶ **ierr** : 整数型。エラーコードが入る。

MPI_Reduceの概念 (集団通信)



MPI_Reduceによる2リスト処理例 (MPI_2DOUBLE_PRECISION と MPI_MAXLOC)



基礎的なMPI関数—MPI_Allreduce

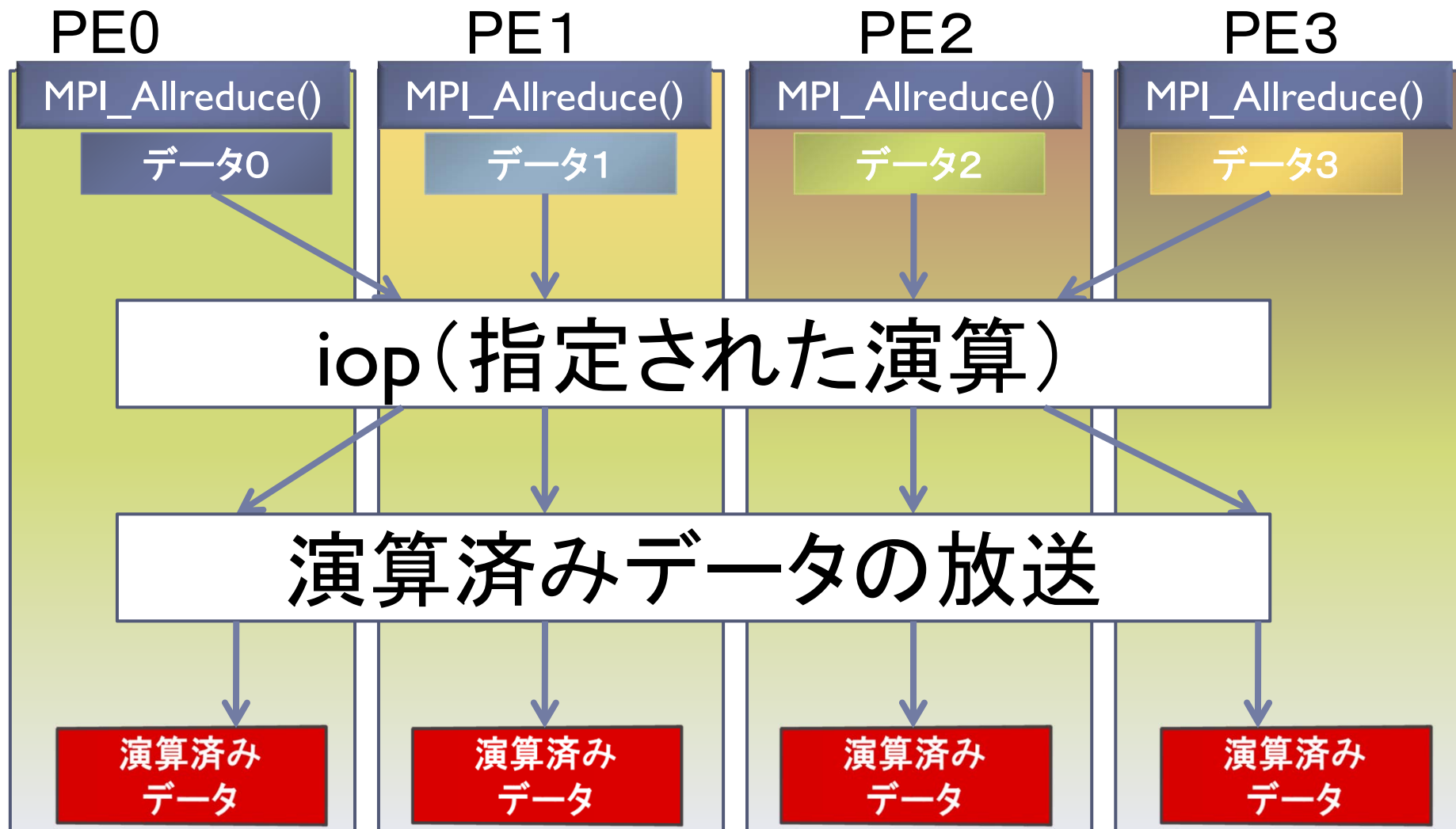
```
▶ ierr = MPI_Allreduce(sendbuf, recvbuf, icount,  
    idatatype, iop, icommm);
```

- ▶ **sendbuf** : 送信領域の先頭番地を指定する。
- ▶ **recvbuf** : 受信領域の先頭番地を指定する。iroot で指定したPEのみで書き込みがなされる。
送信領域と受信領域は、同一であってはならない。
すなわち、異なる配列を確保しなくてはならない。
- ▶ **icount** : 整数型。送信領域のデータ要素数を指定する。
- ▶ **idatatype** : 整数型。送信領域のデータの型を指定する。
 - ▶ 最小値や最大値と位置を返す演算を指定する場合は、**MPI_2INT**(整数型)、**MPI_2FLOAT**(単精度型)、**MPI_2DOUBLE**(倍精度型) を指定する。

基礎的なMPI関数—MPI_Allreduce

- ▶ **iop** : 整数型。演算の種類を指定する。
 - ▶ **MPI_SUM** (総和)、**MPI_PROD** (積)、**MPI_MAX** (最大)、**MPI_MIN** (最小)、**MPI_MAXLOC** (最大と位置)、**MPI_MINLOC** (最小と位置) など。
- ▶ **icomm** : 整数型。PE集団を認識する番号であるコミュニケータを指定する。
- ▶ **ierr** : 整数型。エラーコードが入る。

MPI_Allreduceの概念（集団通信）



リダクション演算

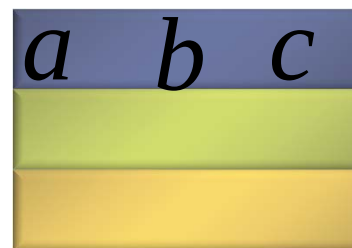
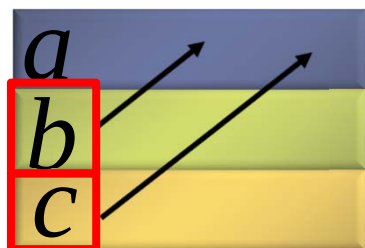
▶ 性能について

- ▶ リダクション演算は、1対1通信に比べ遅い
 - ▶ プログラム中で多用すべきでない！
- ▶ `MPI_Allreduce` は `MPI_Reduce` に比べ遅い
 - ▶ `MPI_Allreduce` は、放送処理が入る。
 - ▶ なるべく、`MPI_Reduce` を使う。

行列の転置

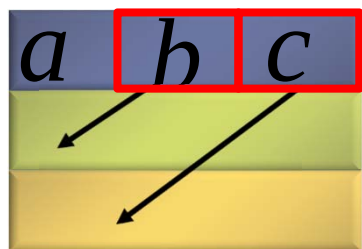
- ▶ 行列 A が (Block, *) 分散されているとする。
- ▶ 行列 A の転置行列 A^T を作るには、MPIでは次の2通りの関数を用いる

- ▶ MPI_Gather関数



集めるメッセージ
サイズが各PEで
均一のとき使う

- ▶ MPI_Scatter関数



集めるサイズが各PEで
均一でないときは:
MPI_GatherV関数
MPI_ScatterV関数

基礎的なMPI関数—MPI_Gather

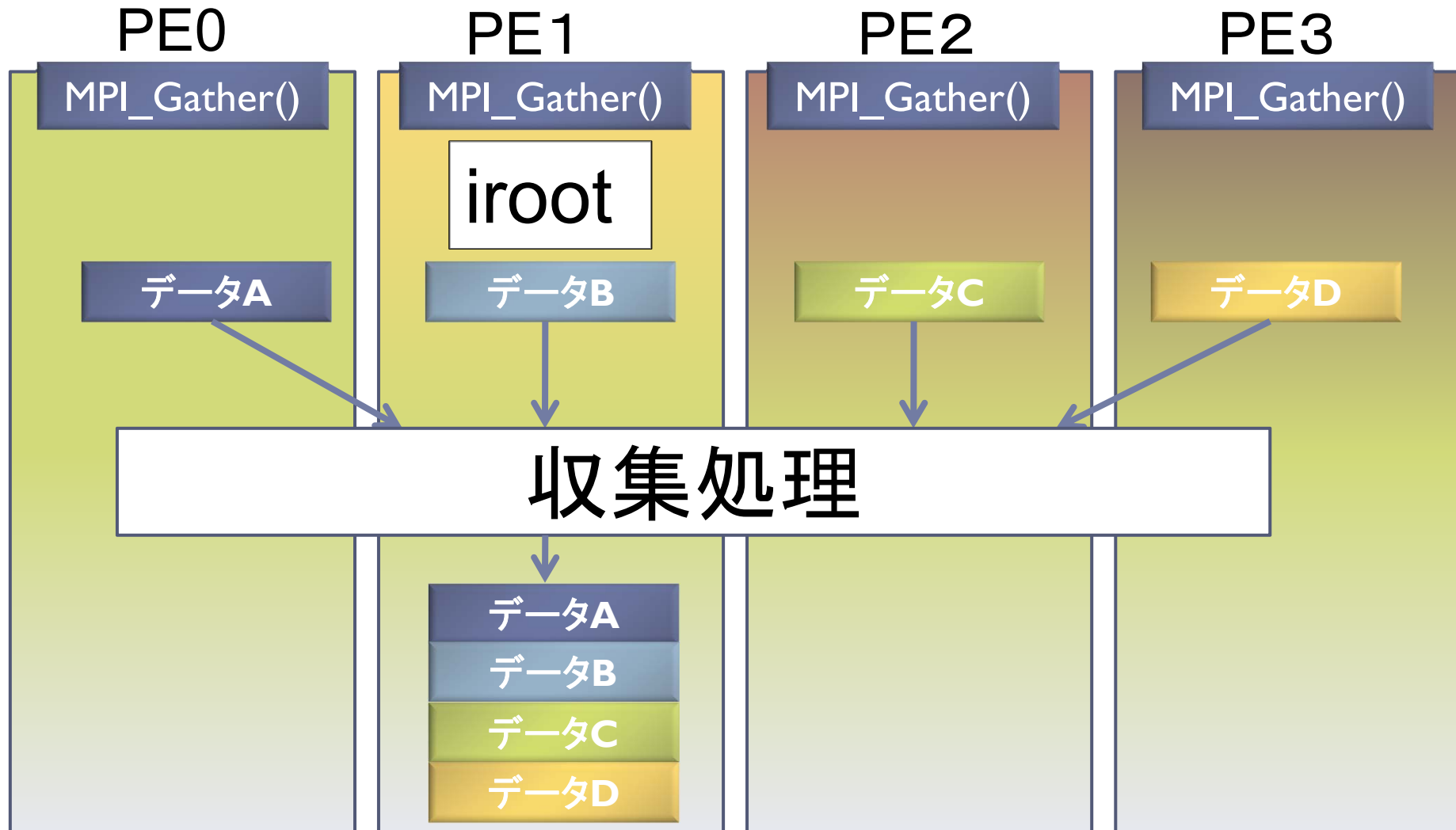
```
▶ ierr = MPI_Gather (sendbuf, isendcount, isendtype,  
                    recvbuf, irecvcount, irecvtype, iroot, icomm);
```

- ▶ **sendbuf**: 送信領域の先頭番地を指定する。
- ▶ **isendcount**: 整数型。送信領域のデータ要素数を指定する。
- ▶ **isendtype**: 整数型。送信領域のデータの型を指定する。
- ▶ **recvbuf**: 受信領域の先頭番地を指定する。iroot で指定したPEのみで書き込みがなされる。
 - ▶ なお原則として、送信領域と受信領域は、同一であってはならない。すなわち、異なる配列を確保しなくてはならない。
- ▶ **irecvcount**: 整数型。受信領域のデータ要素数を指定する。
 - ▶ この要素数は、1PE当たりの送信データ数を指定すること。
 - ▶ MPI_Gather 関数では各PEで異なる数のデータを収集することはできないので、同じ値を指定すること。

基礎的なMPI関数—MPI_Gather

- ▶ **irecvtype** : 整数型。受信領域のデータ型を指定する。
- ▶ **iroot** : 整数型。収集データを受け取るPEの **icomm** 内でのランクを指定する。
 - ▶ 全ての **icomm** 内のPEで同じ値を指定する必要がある。
- ▶ **icomm** : 整数型。PE集団を認識する番号であるコミュニケータを指定する。
- ▶ **ierr** : 整数型。エラーコードが入る。

MPI_Gatherの概念 (集団通信)



基礎的なMPI関数—MPI_Scatter

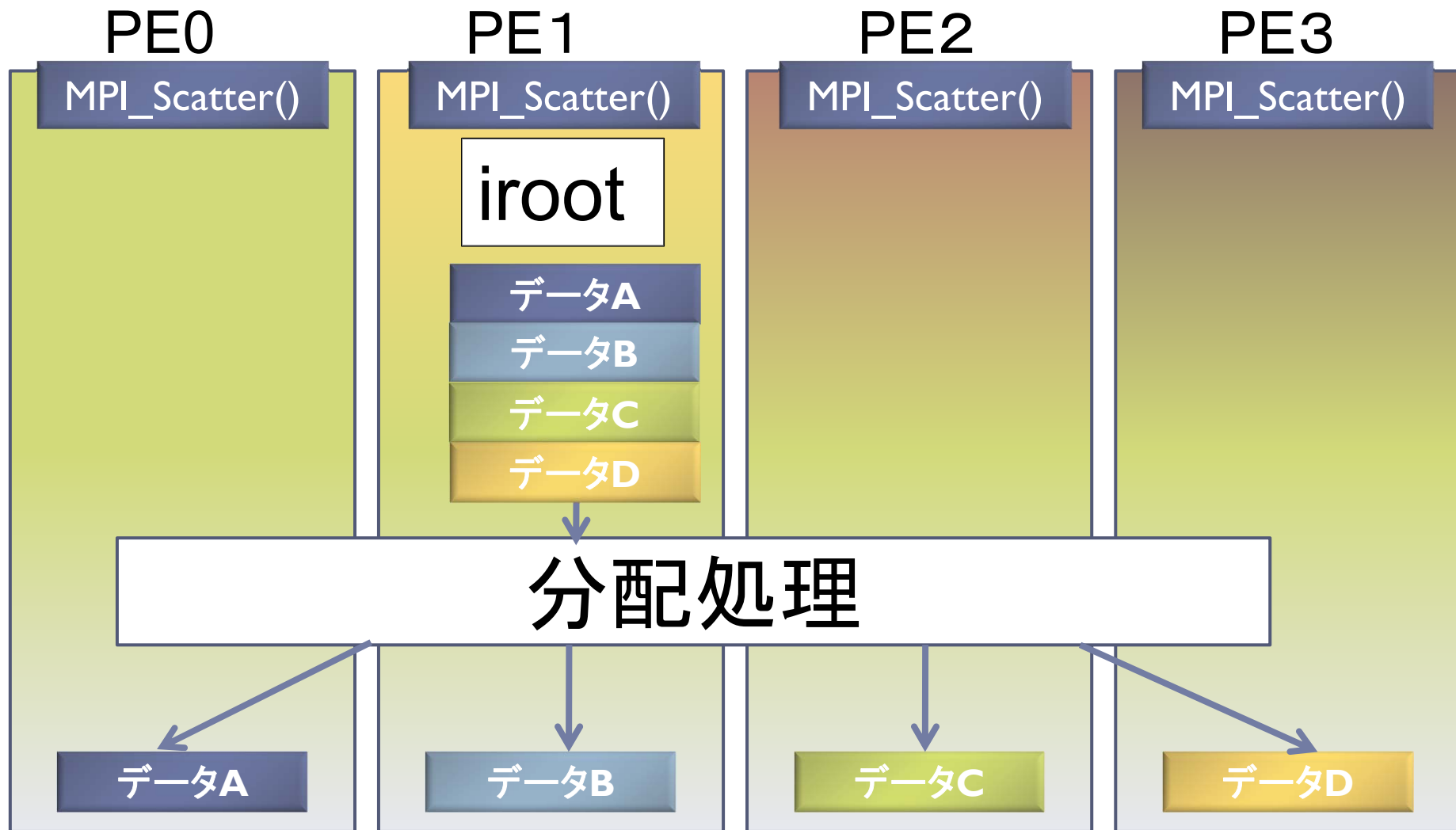
```
▶ ierr = MPI_Scatter ( sendbuf, isendcount, isendtype,  
                      recvbuf, irecvcount, irecvtype, iroot, ictmm);
```

- ▶ **sendbuf**: 送信領域の先頭番地を指定する。
- ▶ **isendcount**: 整数型。送信領域のデータ要素数を指定する。
 - ▶ この要素数は、1PEあたりに送られる送信データ数を指定すること。
 - ▶ MPI_Scatter 関数では各PEで異なる数のデータを分散することはできないので、同じ値を指定すること。
- ▶ **isendtype**: 整数型。送信領域のデータの型を指定する。
iroot で指定したPEのみ有効となる。
- ▶ **recvbuf**: 受信領域の先頭番地を指定する。
 - ▶ なお原則として、送信領域と受信領域は、同一であってはならない。
すなわち、異なる配列を確保しなくてはならない。
- ▶ **irecvcount**: 整数型。受信領域のデータ要素数を指定する。

基礎的なMPI関数—MPI_Scatter

- ▶ **irecvtype** : 整数型。受信領域のデータ型を指定する。
- ▶ **iroot** : 整数型。収集データを受け取るPEの
icomm 内でのランクを指定する。
 - ▶ 全てのicomm 内のPEで同じ値を指定する必要がある。
- ▶ **icomm** : 整数型。PE集団を認識する番号である
コミュニケータを指定する。
- ▶ **ierr** : 整数型。エラーコードが入る。

MPI_Scatterの概念（集団通信）



MPIプログラム実例

MPIの起動

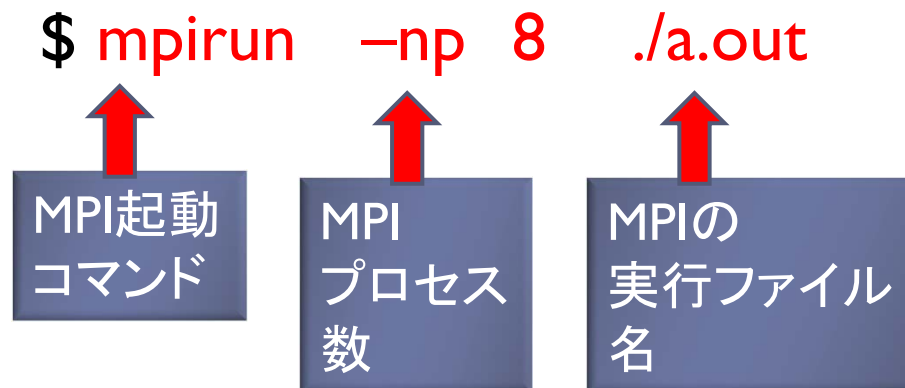
▶ MPIを起動するには

1. MPIをコンパイルできるコンパイラでコンパイル

- ▶ 実行ファイルは a.out とする(任意の名前を付けられます)

2. 以下のコマンドを実行

- ▶ インタラクティブ実行では、以下のコマンドを直接入力
- ▶ バッチジョブ実行では、ジョブスクリプトファイル中に記載

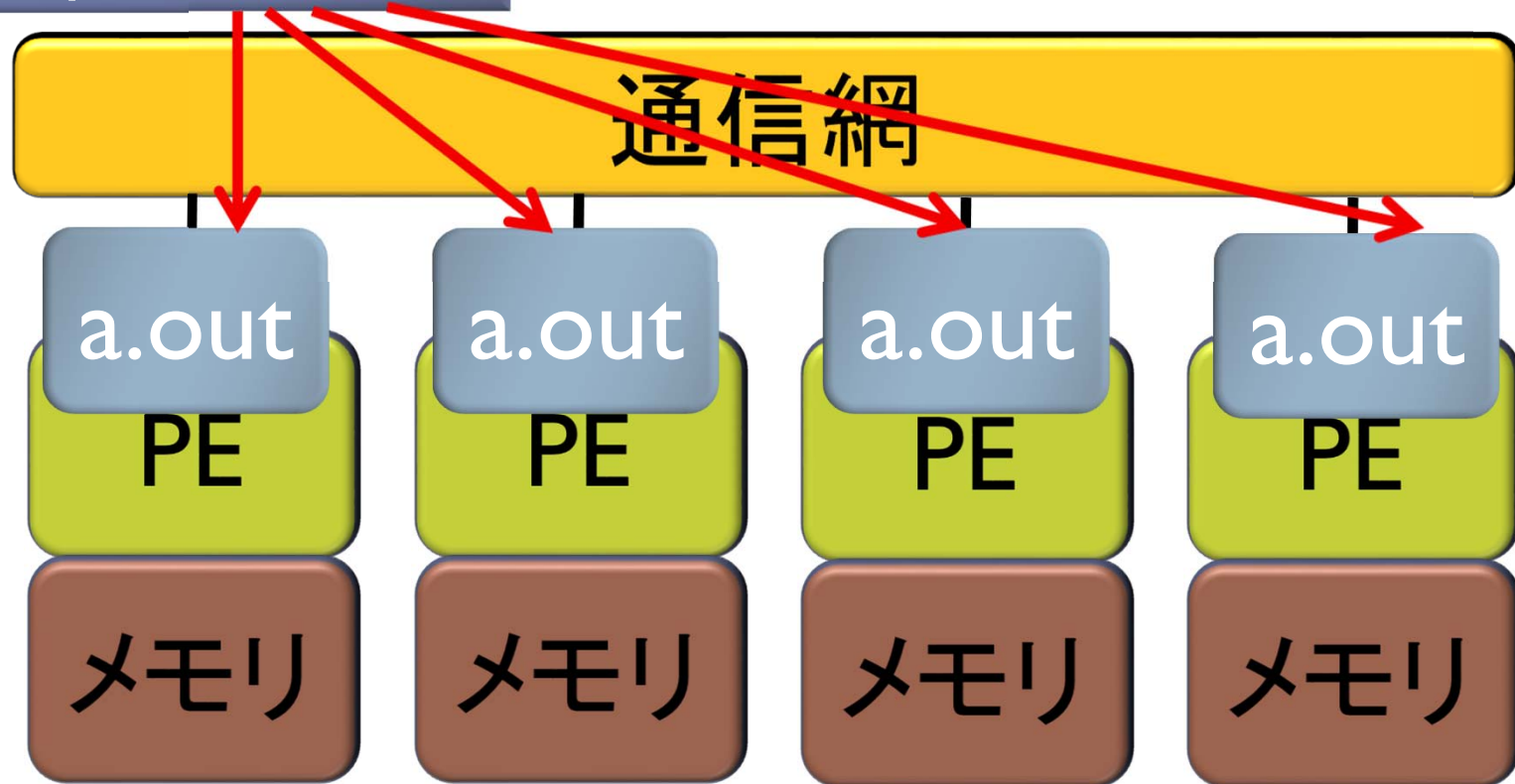


※スパコンのバッチジョブ実行では、MPIプロセス数は専用の指示文で指定する場合があります。その場合は以下になることがあります。

`$mpirun ./a.out`

MPIの起動

```
mpirun -np 4 ./a.out
```



その他の話題（MPIプロセスの割り当て）

- ▶ MPIプロセスと物理ノードとの割り当て
 - ▶ Machine fileでユーザが直接行う
 - ▶ スパコン環境では、バッチジョブシステムが行う
- ▶ バッチジョブシステムが行う場合、通信網の形状を考慮し、通信パターンを考慮し、最適にMPIプロセスが物理ノードに割り当てられるかはわからない
 - ▶ 最悪、通信衝突が多発する
 - ▶ ユーザが、MPIプロセスを割り当てるネットワーク形状を指定できる、バッチジョブシステムもある（例：富士通FX10）
 - ▶ MPIプロセス割り当てを最適化するツールの研究もある
- ▶ スパコンセンタの運用の都合で、ユーザが望むネットワーク形状が常に確保できるとは限らない
 - ▶ →通信を減らす努力、実行時通信最適化の研究進展、が望まれる

数値計算ライブラリの利用

数値計算ライブラリ

▶ 密行列用ライブラリ

- ▶ 行列の要素に0がない(というデータ構造を扱う)
- ▶ 連立一次方程式の解法、固有値問題、FFT、その他
- ▶ 直接解法(反復解法もある)
- ▶ BLAS、LAPACK、ScaLAPACK、SuperLU、MUMPS、FFTW、など

▶ 疎行列用ライブラリ

- ▶ 行列の要素に0が多い
- ▶ 連立一次方程式の解法、固有値問題、その他
- ▶ 反復解法
- ▶ PETSc、Xabclib、Lis、ARPACK、など

疎行列用ライブラリの特徴

- ▶ 疎行列を扱うアプリケーションはライブラリ化が難しい
 - ▶ 疎行列データ形式の標準化が困難
 - ▶ COO、CRS(CCS)、ELL、JDS、BCSR、...
 - ▶ カーネルの演算が微妙に違う、かつ、カーネルは広い範囲に分散
 - ▶ 陽解法(差分法)を基にしたソフトウェア
- ▶ 数値ミドルウェアおよび領域特化型言語 (Domain Specific Language, DSL)
 - ▶ 解くべき方程式や離散化方法に特化させることで、処理(対象となるプログラムの性質)を限定
 - ▶ 以上の限定から、高度な最適化ができる言語(処理系)の作成(DSL)や、ライブラリ化(数値ミドルウェア)ができる
 - ▶ 数値ミドルウェアの例
 - ▶ ppOpen-HPC(東大)、PETSc(Argonne National Laboratory, USA.)、Trilinos (Sandia National Laboratory, USA)、など

BLAS

- ▶ **BLAS (Basic Linear Algebra Subprograms、基本線形代数副プログラム集)**
 - ▶ 線形代数計算で用いられる、基本演算を標準化 (API化) したもの。
 - ▶ 普通は、密行列用の線形代数計算用の基本演算の副プログラムを指す。
 - ▶ 疎行列の基本演算用の **<スパースBLAS>** というものがあるが、まだ定着していない。
 - ▶ スパースBLASはIntel MKL(Math Kernel Library)に入っているが、広く使われているとは言えない。

BLAS

- ▶ BLASでは、以下のように分類わけをして、サブルーチンの命名規則を統一
 1. 演算対象のベクトルや行列の型(整数型、実数型、複素型)
 2. 行列形状(対称行列、三重対角行列)
 3. データ格納形式(帯行列を二次元に圧縮)
 4. 演算結果が何か(行列、ベクトル)
- ▶ 演算性能から、以下の3つに演算を分類
 - ▶ レベル1 BLAS: ベクトルとベクトルの演算
 - ▶ レベル2 BLAS: 行列とベクトルの演算
 - ▶ レベル3 BLAS: 行列と行列の演算

レベル 1 BLAS

▶ レベル1 BLAS

- ▶ ベクトル内積、ベクトル定数倍の加算、など
 - ▶ 例: $y \leftarrow \alpha x + y$
- ▶ データの読み出し回数、演算回数がほぼ同じ
- ▶ データの再利用(キャッシュに乗ったデータの再利用によるデータアクセス時間の短縮)がほとんどできない
 - ▶ 実装による性能向上が、あまり期待できない
 - ▶ ほとんど、計算機ハードウェアの演算性能
- ▶ レベル1BLASのみで演算を実装すると、演算が本来持っているデータ再利用性がなくなる
 - ▶ 例: 行列-ベクトル積を、レベル1BLASで実装

レベル2 BLAS

▶ レベル2 BLAS

▶ 行列-ベクトル積などの演算

▶ 例: $y \leftarrow \alpha A x + \beta y$

▶ 前進/後退代入演算、 $Tx = y$ (T は三角行列)を x について解く演算、を含む

▶ レベル1BLASのみの実装による、データ再利用性の喪失を回避する目的で提案

▶ 行列とベクトルデータに対して、データの再利用性あり

▶ データアクセス時間を、実装法により短縮可能

▶ (実装法により)性能向上がレベル1BLASに比べしやすい(が十分でない)

レベル3 BLAS

▶ レベル3 BLAS

▶ 行列-行列積などの演算

▶ 例: $C \leftarrow \alpha A B + \beta C$

▶ 共有記憶型の並列ベクトル計算機では、レベル2 BLASでも性能向上が達成できない。

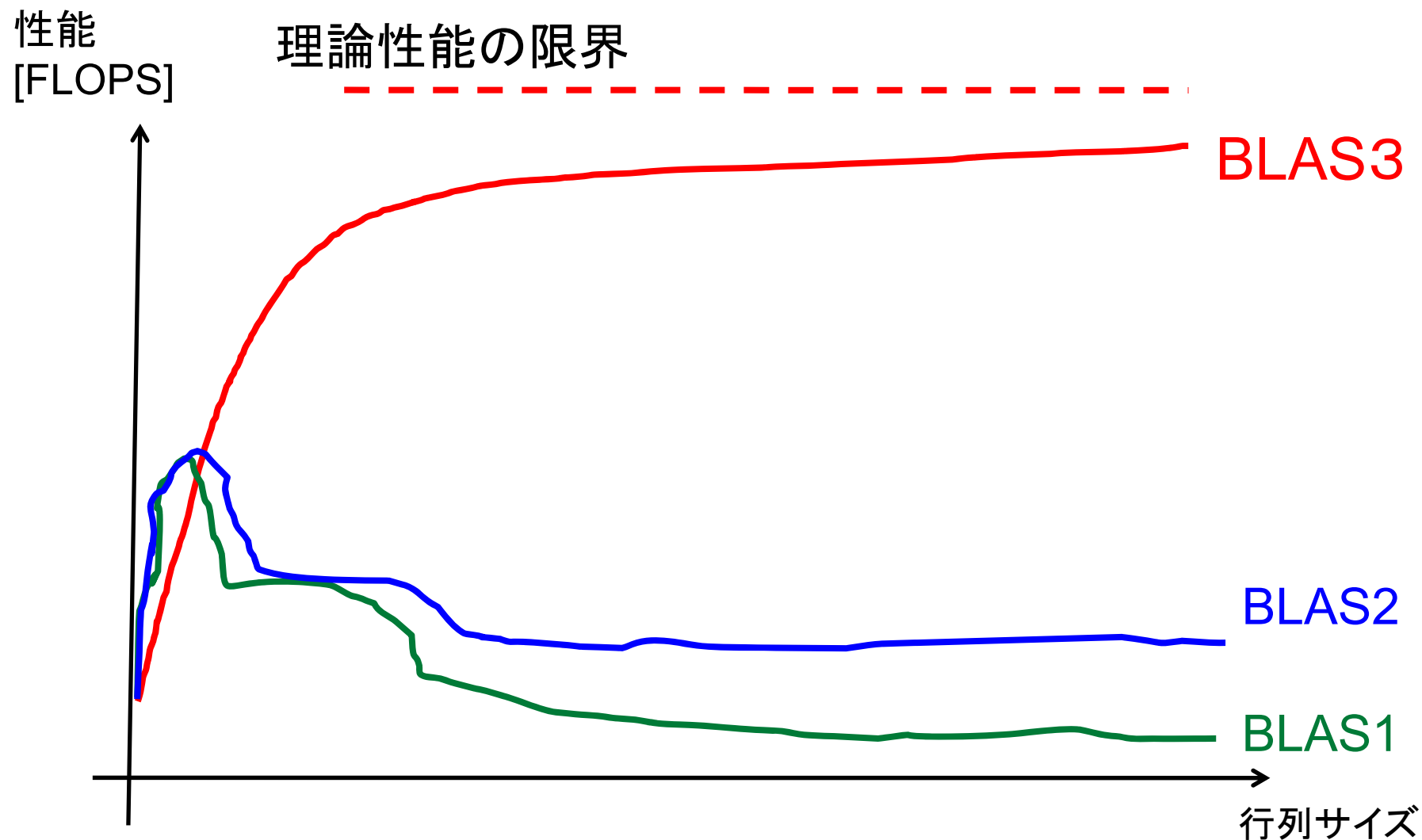
▶ 並列化により1PE当たりのデータ量が減少する。

▶ より大規模な演算をとり扱わないと、再利用の効果が無い。

▶ 行列-行列積では、行列データ $O(n^2)$ に対して演算は $O(n^3)$ なので、データ再利用性が原理的に高い。

▶ 行列積は、アルゴリズムレベルでもブロック化できる。さらにデータの局所性を高めることができる。

典型的なBLASの性能



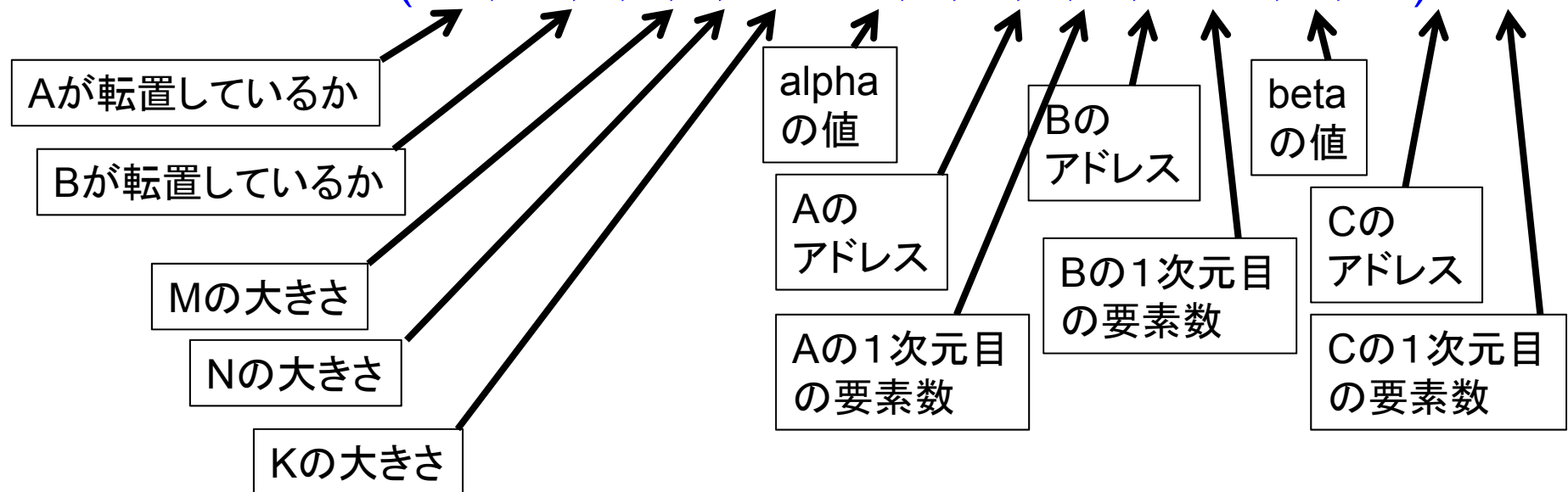
BLAS利用例

▶ 倍精度演算BLAS3

$C := \alpha * op(A) * op(B) + \beta * C$

$A: M * K; B: K * N; C: M * N;$

`CALL DGEMM( 'N', 'N', n, n, n, ALPHA, A, N, B, N, BETA, C, N )`



BLASの機能詳細

- ▶ 詳細はHP: <http://www.netlib.org/blas/>
- ▶ 命名規則: 関数名: **XYYYYY**
 - ▶ **X**: データ型
S:単精度、D:倍精度、C:複素、Z:倍精度複素
 - ▶ **YYYYY**: 計算の種類
 - ▶ レベル1:
例: AXPY: ベクトルをスカラー倍して加算
 - ▶ レベル2:
例: GEMV: 一般行列とベクトルの積
 - ▶ レベル3:
例: GEMM: 一般行列どうしの積

GOTO BLASとは

- ▶ 後藤和茂 氏により開発された、ソースコードが無償入手可能な、高性能BLASの実装(ライブラリ)
 - ▶ 特徴
 - ▶ マルチコア対応がなされている
 - ▶ 多くのコモディティハードウェア上の実装に特化
 - ▶ Intel Nehalem and Atom systems
 - ▶ VIA Nanoprocessor
 - ▶ AMD Shanghai and Istanbul
- 等
- ▶ テキサス大学先進計算センター(TACC)で、GOTO BLAS2として、ソースコードを配布している
 - ▶ HP : <http://www.tacc.utexas.edu/tacc-projects/gotoblas2/>

LAPACK

- ▶ 密行列に対する、連立一次方程式の解法、および固有値の解法の“標準”アルゴリズムルーチンを無償で提供
- ▶ その道の大学の専門家が集結
 - ▶ カリフォルニア大バークレー校：
James Demmel教授
 - ▶ テネシー大ノックスビル校：
Jack Dongarra教授
- ▶ HP
<http://www.netlib.org/lapack/>

LAPACKの命名規則

▶ 命名規則： 関数名：**XYYZZZ**

▶ **X**: データ型

S:単精度、D:倍精度、C:複素、Z:倍精度複素

▶ **YY**: 行列の型

BD: 二重対角、DI: 対角、GB: 一般帯行列、GE: 一般行列、
HE: 複素エルミート、HP: 複素エルミート圧縮形式、SY: 対称
行列、....

▶ **ZZZ**: 計算の種類

TRF: 行列の分解、TRS: 行列の分解を使う、CON: 条件数
の計算、RFS: 計算解の誤差範囲を計算、TRI: 三重対角行
列の分解、EQU: スケーリングの計算、...

インタフェース例：DGESV (1 / 3)

▶ **DGESV**

(N, NRHS, A, LDA, IPIVOT, B, LDB, INFO)

- ▶ $A X = B$ の解の行列 X を計算をする
- ▶ $A * X = B$ 、ここで A は $N \times N$ 行列で、 X と B は $N \times NRHS$ 行列とする。
- ▶ 行交換の部分枢軸選択付きのLU分解 で A を $A = P * L * U$ と分解する。ここで、 P は交換行列、 L は下三角行列、 U は上三角行列である。
- ▶ 分解された A は、連立一次方程式 $A * X = B$ を解くのに使われる。

▶ 引数

▶ **N (入力) - INTEGER**

- ▶ 線形方程式の数。行列 A の次元数。 $N \geq 0$ 。

インタフェース例：DGESV (2 / 3)

▶ NRHS (入力) – INTEGER

- ▶ 右辺ベクトルの数。行列Bの次元数。NRHS ≥ 0 。

▶ A (入力／出力) – DOUBLE PRECISION, DIMENSION(:, :)

- ▶ 入力時は、 $N \times N$ の行列Aの係数を入れる。
- ▶ 出力時は、Aから分解された行列Lと $U = P * L * U$ を圧縮して出力する。
Lの対角要素は1であるので、収納されていない。

▶ LDA (入力) – INTEGER

- ▶ 配列Aの最初の次元の大きさ。LDA $\geq \max(1, N)$ 。

▶ IPIVOT (出力) – DOUBLE PRECISION, DIMENSION(:)

- ▶ 交換行列Aを構成する枢軸のインデックス。行列のi行がIPIVOT(i)行と交換されている。

インタフェース例：DGESV (3 / 3)

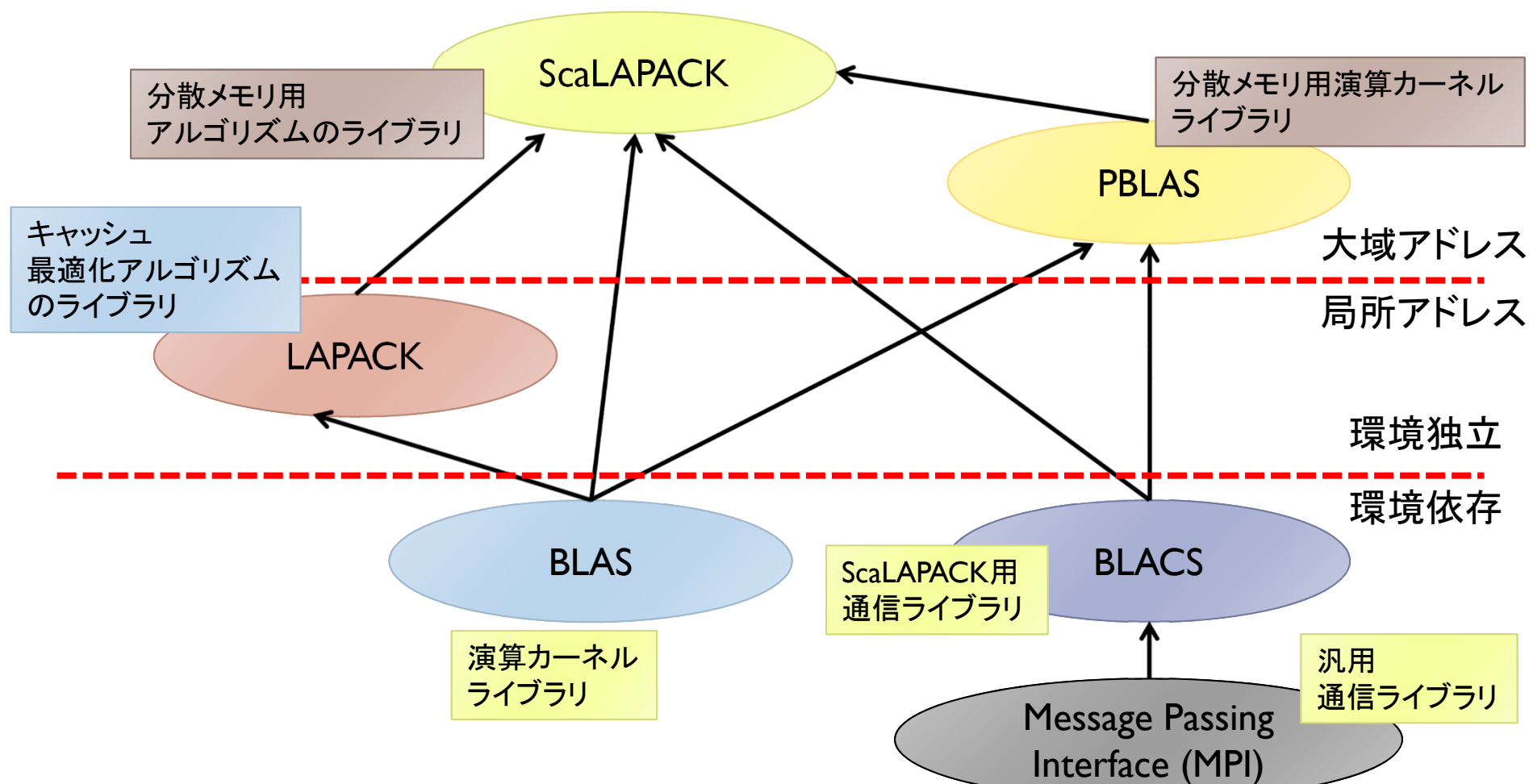
- ▶ **B (入力／出力) – DOUBLE PRECISION, DIMENSION(:, :)**
 - ▶ 入力時は、右辺ベクトルの $N \times NRHS$ 行列Bを入れる。
 - ▶ 出力時は、もし、 $INFO = 0$ なら、 $N \times NRHS$ 行列である解行列Xが戻る。
- ▶ **LDB (入力) – INTEGER**
 - ▶ 配列Bの最初の次元の大きさ。 $LDB \geq \max(1, N)$ 。
- ▶ **INFO (出力) – INTEGER**
 - ▶ $= 0$: 正常終了
 - ▶ < 0 : もし $INFO = -i$ なら i -th 行の引数の値がおかしい。
 - ▶ > 0 : もし $INFO = i$ なら $U(i, i)$ が厳密に0である。分解は終わるが、Uの分解は特異なため、解は計算されない。

ScaLAPACK

- ▶ 密行列に対する、連立一次方程式の解法、および固有値の解法の“標準”アルゴリズムルーチンの並列化版を無償で提供
- ▶ ユーザインタフェースはLAPACKに＜類似＞
- ▶ ソフトウェアの＜階層化＞がされている
 - ▶ 内部ルーチンはLAPACKを利用
 - ▶ 並列インタフェースはBLACS
- ▶ データ分散方式に、2次元ブロック・サイクリック分散方式を採用（**詳細は、「MPI」の講義で説明**）
- ▶ HP: <http://www.netlib.org/scalapack/>

ScaLAPACKのソフトウェア構成図

出典: <http://www.netlib.org/scalapack/poster.html>



BLACSとPBLAS

▶ BLACS

- ▶ ScaLAPACK中で使われる通信機能を関数化したもの。
- ▶ 通信ライブラリは、MPI、PVM、各社が提供する通信ライブラリを想定し、ScaLAPACK内でコード修正せずに使うことを目的とする
 - ▶ いわゆる、通信ライブラリのラッパー的役割でScaLAPACK内で利用
- ▶ 現在、MPIがデファクトになったため、MPIで構築されたBLACSのみ、現実的に利用されている。
 - ▶ なので、ScaLAPACKはMPIでコンパイルし、起動して利用する

▶ PBLAS

- ▶ BLACSを用いてBLASと同等な機能を提供する関数群
- ▶ 並列版BLASといってよい。

ScaLAPACKの命名規則

- ▶ 原則:
LAPACKの関数名の頭に“P”を付けたもの
- ▶ そのほか、BLACS、PBLAS、データ分散を制御するためのScaLAPACK用関数がある。

インタフェース例：PDGESV (1 / 4)

▶ PDGESV

(N, NRHS, A, IA, JA, DESCA, IPIV, B, IB, JB, DESCB, INFO)

- ▶ $\text{sub}(A) X = \text{sub}(B)$ の解の行列 X を計算をする
- ▶ ここで $\text{sub}(A)$ は $N \times N$ 行列を分散した $A(\text{IA}:\text{IA}+N-1, \text{JA}:\text{JA}+N-1)$ の行列
- ▶ X と B は $N \times \text{NRHS}$ 行列を分散した $B(\text{IB}:\text{IB}+N-1, \text{JB}:\text{JB}+\text{NRHS}-1)$ の行列
- ▶ 行交換の部分枢軸選択付きのLU分解 で $\text{sub}(A)$ を $\text{sub}(A) = P * L * U$ と分解する。ここで、 P は交換行列、 L は下三角行列、 U は上三角行列である。
- ▶ 分解された $\text{sub}(A)$ は、連立一次方程式 $\text{sub}(A) * X = \text{sub}(B)$ を解くのに使われる。

インタフェース例：PDGESV (2 / 4)

- ▶ **N (大域入力) – INTEGER**
 - ▶ 線形方程式の数。行列Aの次元数。 $N \geq 0$ 。
- ▶ **NRHS (大域入力) – INTEGER**
 - ▶ 右辺ベクトルの数。行列Bの次元数。 $NRHS \geq 0$ 。
- ▶ **A (局所入力／出力) – DOUBLE PRECISION, DIMENSION(:, :)**
 - ▶ 入力時は、 $N \times N$ の行列Aの局所化された係数を配列A(LLD_A, LOCc(JA+N-1))を入れる。
 - ▶ 出力時は、Aから分解された行列Lと $U = P * L * U$ を圧縮して出力する。Lの対角要素は1であるので、収納されていない。
- ▶ **IA(大域入力) – INTEGER** : sub(A)の最初の行のインデックス
- ▶ **JA(大域入力) – INTEGER** : sub(A)の最初の列のインデックス
- ▶ **DESCA (大域かつ局所入力) – INTEGER**
 - ▶ 分散された配列Aの記述子。

インタフェース例：PDGESV (3 / 4)

- ▶ **IPIVOT (局所出力) – DOUBLE PRECISION, DIMENSION(:)**
 - ▶ 交換行列Aを構成する枢軸のインデックス。行列のi行がIPIVOT(i)行と交換されている。分散された配列($LOCr(M_A)+MB_A$)として戻る。
- ▶ **B (局所入力／出力) – DOUBLE PRECISION, DIMENSION(:, :)**
 - ▶ 入力時は、右辺ベクトルの $N \times NRHS$ の行列Bの分散されたものを($LLD_B, LOCc(JB+NRHS-1)$)に入れる。
 - ▶ 出力時は、もし、 $INFO = 0$ なら、 $N \times NRHS$ 行列である解行列Xが、行列Bと同様の分散された状態で戻る。
- ▶ **IB(大域入力) – INTEGER**
 - ▶ $sub(B)$ の最初の行のインデックス
- ▶ **JB(大域入力) – INTEGER**
 - ▶ $sub(B)$ の最初の列のインデックス
- ▶ **DESCB (大域かつ局所入力) – INTEGER**
 - ▶ 分散された配列Bの記述子。

インタフェース例：PDGESV (4 / 4)

▶ INFO (大域出力) —INTEGER

- ▶ = 0: 正常終了
- ▶ < 0:
 - もし i 番目の要素が配列で、その j 要素の値がおかしいなら、 $\text{INFO} = -(i*100+j)$ となる。
 - もし i 番目の要素がスカラーで、かつ、その値がおかしいなら、 $\text{INFO} = -i$ となる。
- ▶ > 0: もし $\text{INFO} = K$ のとき $U(\text{IA}+K-1, \text{JA}+K-1)$ が厳密に0である。分解は完了するが、分解された U は厳密に特異なので、解は計算できない。

BLAS利用の注意

▶ C言語からの利用

- ▶ BLASライブラリは(たいてい)Fortranで書かれている
- ▶ 行列を1次元で確保する
 - ▶ Fortranに対して転置行列になるので、BLASの引数で転置を指定
- ▶ 引数は全てポインタで引き渡す
- ▶ 関数名の後に“_”をつける(BLASをコンパイルするコンパイラ依存)
 - ▶ 例: dgemm_(...)

▶ 小さい行列は性能的に注意

- ▶ キャッシュに載るようなサイズ(例えば、100次元以下)の行列については、BLASが高速であるとは限らない
 - ▶ BLASは、大規模行列で高性能になるように設計されている
- ▶ 全体の行列サイズは大きくても、利用スレッド数が多くなると、スレッド当たりの行列サイズが小さくなるので注意！
 - ▶ 例) N=8000でも、200スレッド並列だと、スレッドあたりN=570まで小さくなる

その他のライブラリ（主に行列演算）

種類	問題	ライブラリ名	概要
密行列	BLAS	MAGMA	GPU、マルチコア、ヘテロジニアス環境対応
疎行列	連立一次方程式	MUMPS	直接解法
		SuperLU	直接解法
		PETSc	反復解法、各種機能
		Hypre	反復解法
	連立一次方程式、固有値ソルバ	Lis	反復解法 (国産ライブラリ)
		Xabclib	反復解法、自動チューニング (AT) 機能 (国産ライブラリ)

その他のライブラリ（信号処理等）

種類	問題	ライブラリ名	概要
信号処理	FFT	FFTW	離散フーリエ変換、AT機能
		FFTE	離散フーリエ変換（国産ライブラリ）
		Spiral	離散フーリエ変換、AT機能
グラフ処理	グラフ分割	METIS、ParMETIS	グラフ分割
		SCOTCH、PT-SCOTCH	グラフ分割

その他のライブラリ（フレームワーク）

種類	問題	ライブラリ名	概要
プログラミング 環境	マルチ フィジックス、 など	Trilinos	プログラミング フレームワークと 数値計算ライブラリ
	ステンシル 演算	Phisis	ステンシル演算用 プログラミング フレームワーク (国産ライブラリ)
数値 ミドルウェア	FDM、FEM、DEM、 BEM、FVM	ppOpen-HPC	5種の離散化手法に 基づくシミュレーション ソフトウェア、数値 ライブラリ、AT機能 (国産ライブラリ)

レポート課題

1. [L01] 10000台のPEを用いるとき、並列化効率を90%以上に保つためには、全体の何%以上が並列化されていないといけなだろうか？
2. [L10] MPI_Reduce関数 と MPI_Allreduce関数の性能を比較せよ。
3. [L10] MPI_Scatter関数、およびMPI_Gather関数を用いて、行列の転置処理を実装せよ。

問題のレベルに関する記述：

- L00: きわめて簡単な問題。
 - L10: ちょっと考えればわかる問題。
 - L20: 標準的な問題。
 - L30: 数時間程度必要とする問題。
 - L40: 数週間程度必要とする問題。複雑な実装を必要とする。
 - L50: 数か月程度必要とする問題。未解決問題を含む。
- ※L40以上は、論文を出版するに値する問題。

レポート課題（１）（つづき１）

4. [L20] 時間測定関数MPI_Wtimeの測定精度について、プログラムを作成し、計測したデータを基に考察せよ。
5. [L10] 現在公開されているMPIの実装について調べよ。それらについて、短所と長所をまとめよ。
6. [L5～L15] Flynnの分類（４種）を基にして、現在の並列コンピュータの開発動向をまとめよ。なお、以下のキーワードを考慮すること。（キーワード：パイプライン方式）
7. [L5～L20]（記述内容で点数を可変にします）
二分木通信方式は、本当に効率の良い方法であろうか。
計算量の観点から考察してみよ。
（キーワード：コスト最適）

レポート課題（つづき2）

8. [L20] 最寄りの計算機にBLASがインストールされているか調べよ。無い場合は、インストールせよ。
そして、BLASの演算性能を調査せよ。
9. [L30] BLASの性能を自動チューニングするATLAS (Automatically Tuned Linear Algebra Subprograms)について、どのような仕組みで自動チューニングしているか調査せよ。また、ATLASをインストールし、その性能を評価せよ。
10. [L30] 高速なBLASでフリーソフトウェアである、GOTO BLAS(後藤BLAS)について調査せよ。
特に、どのような仕組みでBLAS演算を実装しているのか、調査せよ。

参考文献（１）

1. BLAS
<http://www.netlib.org/blas/>
2. LAPACK
<http://www.netlib.org/lapack/>
3. ScaLAPACK
<http://www.netlib.org/scalapack/>
4. スペースBLAS
<http://math.nist.gov/spblas/>

参考文献（２）

1. MPI並列プログラミング、P.パチェコ 著 ／ 秋葉 博 訳
2. 並列プログラミング虎の巻MPI版、青山幸也 著、
高度情報科学技術研究機構(RIST) 神戸センター
(http://www.hpci-office.jp/pages/seminar_text)
3. Message Passing Interface Forum
(<http://www.mpi-forum.org/>)
4. 並列コンピュータ工学、富田眞治著、昭晃堂(1996)
5. 並列数値処理 ー高速化と性能向上のためにー、
金田康正 編著、コロナ社(2010)

来週へつづく