
第25回 お試しアカウント付き
並列プログラミング講習会
「ライブラリ利用：高性能プログラミング初級入門」

東京大学情報基盤センター
スーパーコンピューティングチーム

講習会概略

- ▶ **開催日:** 2012年12月19日(水) 10:30 – 17:00
2012年12月20日(木) 10:30 – 17:00
- ▶ **場所:** 東京大学情報基盤センター 4階 413遠隔講義室
- ▶ **講習会プログラム:**
- ▶ 12月19日(水)
 - ▶ 10:00 – 10:30 受付
 - ▶ 10:30 – 12:30 ノートパソコンの設定、テストプログラム(MPI)の実行など(演習)
(講師: 片桐)
 - ▶ 14:00 – 15:00 並列プログラミングの基本(座学)(講師: 片桐)
 - ▶ 15:15 – 17:00 プログラム実習 I (BLAS)(演習)(講師: 片桐)
- ▶ 12月20日(木)
 - ▶ 10:30 – 12:30 プログラミング実習 II (LAPACK, ScaLAPACK)(演習)(講師: 片桐)
 - ▶ 14:00 – 15:30 プログラミング実習 III (Lis)(演習)(講師: 伊藤)
 - ▶ 15:45 – 17:00 プログラミング実習 IV (PETSc)(演習)(講師: 伊藤)

有償トライアルユース制度の開始について

- ▶ 2012年4月より、安価に当センターのFX10スーパーコンピュータシステムが使える「**無償トライアルユース**、および、**有償トライアルユース**」制度が開始予定です。
 - ▶ **アカデミック利用**
 - ▶ パーソナルコース、グループコースの双方(1ヶ月～3ヶ月)
 - ▶ **企業利用**
 - ▶ パーソナルコース(1ヶ月～3ヶ月)(最大24ノード、最大96ノード)
本講習会の受講が必須、審査無
 - ▶ グループコース
 - 無償トライアルユース:(1ヶ月～3ヶ月):無料(12ノード、1口)
 - 有償トライアルユース:(1ヶ月～最大通算9ヶ月)、有償(12ノード、1口)
 - **スーパーコンピュータ利用資格者審査委員会の審査が必要(年4回実施)**
 - ▶ **双方のコースともに、簡易な利用報告書の提出が必要**
- ▶ 料金体系や利用条件の詳細は、以下のHPをご覧ください
<http://www.cc.u-tokyo.ac.jp/service/trial/>

テストプログラム起動

UNIX 備忘録

- ▶ Emacsの起動: `emacs` <編集ファイル名>
 - ▶ `^x ^s` (^はcontrol) : テキストの保存
 - ▶ `^x ^c` : 終了
(`^z` で終了すると、スパコンの負荷が上がる。絶対にしないこと。)
 - ▶ `^g` : 訳がわからなくなったとき。
 - ▶ `^k` : カーソルより行末まで消す。
消した行は、一時的に記憶される。
 - ▶ `^y` : `^k`で消した行を、現在のカーソルの場所にコピーする。
 - ▶ `^s 文字列` : 文字列の箇所まで移動する。
 - ▶ `^M goto-line` : 指定した行まで移動する。
(`^M`はESCキーを押す)

UNIX 備忘録

- ▶ **rm ファイル名** : ファイル名のファイルを消す。
 - ▶ **rm ***~ : test.c~ などの、~がついたバックアップファイルを消す。
- ▶ **ls** : 現在いるフォルダの中身を見る。
- ▶ **cd フォルダ名** : フォルダに移動する。
 - ▶ **cd ..** : 一つ上のフォルダに移動。
 - ▶ **cd ~** : ホームディレクトリに行く。訳がわからなくなったとき。
- ▶ **cat ファイル名** : ファイル名の中身を見る
- ▶ **make** : 実行ファイルを作る
(Makefile があるところでしか実行できない)
 - ▶ **make clean** : 実行ファイルを消す。
(clean がMakefileで定義されていないと実行できない)

サンプルプログラムの実行

初めての並列プログラムの実行

サンプルプログラム名

- ▶ C言語版・Fortran90版共通ファイル:
[Samples-fx.tar](#)
- ▶ tarで展開後、C言語とFortran90言語のディレクトリが作られる
 - ▶ [C/](#) : C言語用
 - ▶ [F/](#) : Fortran90言語用
- ▶ 上記のファイルが置いてある場所
[/home/z30082](#)

並列版Helloプログラムをコンパイルしよう (1/2)

1. `/home/z30082` にある `Samples-fx.tar` を
自分のディレクトリにコピーする
`$ cp /home/z30082/Samples-fx.tar ./`
2. `Samples-fx.tar` を展開する
`$ tar xvf Samples-fx.tar`
3. `Samples` フォルダに入る
`$ cd Samples`
4. C言語 : `$ cd C`
Fortran90言語 : `$ cd F`
5. `Hello` フォルダに入る
`$ cd Hello`

並列版Helloプログラムをコンパイルしよう (2/2)

6. ピュアMPI用のMakefileをコピーする

```
$ cp Makefile_pure Makefile
```

7. make する

```
$ make
```

8. 実行ファイル(hello)ができていることを確認する

```
$ ls
```

FX10スーパーコンピュータシステムでの ジョブ実行形態

- ▶ 以下の2通りがあります
- ▶ インタラクティブジョブ実行
 - ▶ PCでの実行のように、コマンドを入力して実行する方法
 - ▶ スパコン環境では、あまり一般的でない
 - ▶ デバック用、大規模実行はできない
 - ▶ FX10では、以下に限定
 - ▶ 1ノード(16コア)(2時間まで)
 - ▶ 8ノード(128コア)(10分まで)
- ▶ バッチジョブ実行
 - ▶ バッチジョブシステムに処理を依頼して実行する方法
 - ▶ スパコン環境で一般的
 - ▶ 大規模実行用
 - ▶ FX10では、最大1440ノード(23,040コア)(6時間)

インタラクティブ実行のやり方

(本講習会では使えません。一般利用者のみ)

- ▶ コマンドラインで以下を入力

- ▶ 1ノード実行用

- ```
$ pjsub --interact
```

- ▶ 8ノード実行用

- ```
$ pjsub --interact -L "node=8"
```

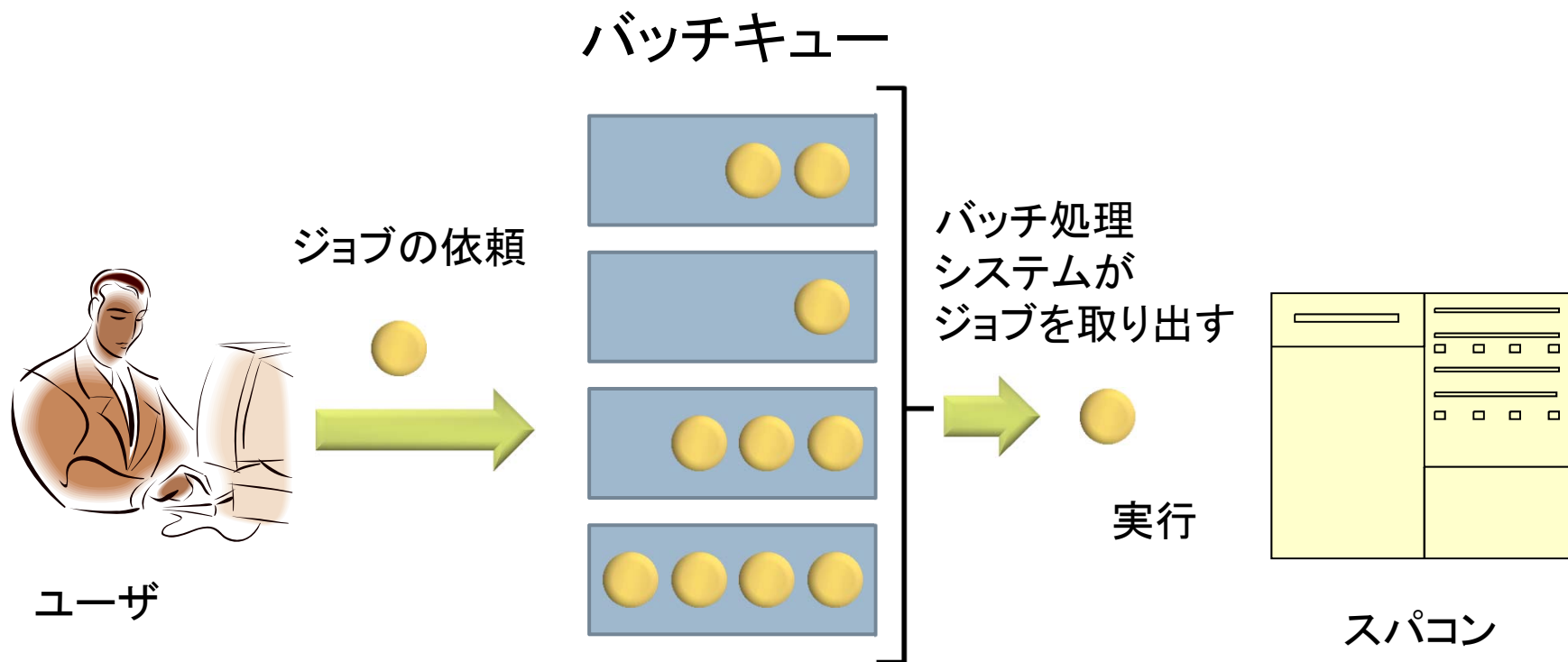
※インタラクティブ用のノード総数は50ノードです。
もしユーザにより50ノードすべて使われている場合、
資源が空くまで、ログインできません。

コンパイラの種類とインタラクティブ実行 およびバッチ実行

- ▶ インタラクティブ実行、およびバッチ実行で、利用するコンパイラ（C言語、C++言語、Fortran90言語）の種類が違います
- ▶ インタラクティブ実行では
 - ▶ オウンコンパイラ（そのノードで実行する実行ファイルを生成するコンパイラ）を使います
- ▶ バッチ実行では
 - ▶ クロスコンパイラ（そのノードでは実行できないが、バッチ実行する時のノードで実行できる実行ファイルを生成するコンパイラ）を使います
- ▶ それぞれの形式
 - ▶ オウンコンパイラ: <コンパイラの種類名>
 - ▶ クロスコンパイラ: <コンパイラの種類名>px
 - ▶ 例) 富士通Fortran90コンパイラ
 - ▶ オウンコンパイラ: frt
 - ▶ クロスコンパイラ: frtpx

バッチ処理とは

- ▶ スパコン環境では、通常は、インタラクティブ実行(コマンドラインで実行すること)はできません。
- ▶ ジョブはバッチ処理で実行します。



バッチキューの設定のしかた

- ▶ バッチ処理は、富士通社のバッチシステムで管理されています。
- ▶ 以下、主要コマンドを説明します。
 - ▶ ジョブの投入：
`pjsub <ジョブスクリプトファイル名> -g <プロジェクトコード>`
 - ▶ 自分が投入したジョブの状況確認: `pjstat`
 - ▶ 投入ジョブの削除: `pjdel <ジョブID>`
 - ▶ バッチキューの状態を見る: `pjstat --rsc`
 - ▶ バッチキューの詳細構成を見る: `pjstat -rsc -x`
 - ▶ 投げられているジョブ数を見る: `pjstat -b`
 - ▶ 過去の投入履歴を見る: `pjstat --history`
 - ▶ 同時に投入できる数／実行できる数を見る: `pjstat --limit`

本お試し講習会でのキュー名

▶ 本演習中のキュー名：

▶ tutorial

▶ 最大15分まで

▶ 最大ノード数は12ノード(192コア)まで

▶ 本演習時間以外(24時間)のキュー名：

▶ lecture

▶ 利用条件は演習中のキュー名と同様

pjstat --rsc の実行画面例

```
$ pjstat --rsc
RSCGRP          STATUS          NODE:COORD
debug           [ENABLE,START]  480:10x3x16
short           [ENABLE,START]  480:10x3x16
regular
|---- small     [ENABLE,START]  3840:20x12x16
|---- medium    [ENABLE,START]  3840:20x12x16
|---- large     [ENABLE,START]  3840:20x12x16
`---- x-large   [ENABLE,START]  3840:20x12x16
interactive
|---- interactive_n1 [ENABLE,START]  50
`---- interactive_n8 [ENABLE,START]  50
```

使える
キュー名
(リソース
グループ)

現在
使えるか

ノードの
物理構成情報

pjstat --rsc -x の実行画面例

```
$ pjstat --rsc -x
```

RSCGRP	STATUS	MIN_NODE	MAX_NODE	ELAPSE	MEM(GB)	PROJECT
debug	[ENABLE,START]	1	240	00:30:00	28	gcXX, gcYY
short	[ENABLE,START]	1	72	06:00:00	28	gcXX, gcYY
regular						
---- small	[ENABLE,START]	12	216	48:00:00	28	gcXX, gcYY
---- medium	[ENABLE,START]	217	372	48:00:00	28	gcXX, gcYY
---- large	[ENABLE,START]	373	480	48:00:00	28	gcXX, gcYY
`---- x-large	[ENABLE,START]	481	1440	24:00:00	28	gcXX, gcYY
interactive						
---- interactive_n1	[ENABLE,START]	1	1	02:00:00	28	gcXX, gcYY
`---- interactive_n8	[ENABLE,START]	2	8	00:10:00	28	gcXX, gcYY

使える
キュー名
(リソース
グループ)

現在
使えるか

ノードの
実行情報

課金情報(財布)
実習では1つのみ

pjstat -b の実行画面例

```
$ pjstat -b
RSCGRP      STATUS      TOTAL  RUNNING  QUEUED  HOLD  OTHER  NODE:COORD
debug  [ENABLE,START]      3        2        0        0        1    480:10x3x16
short  [ENABLE,START]      1        1        0        0        0    480:10x3x16
regular
|---- small [ENABLE,START] 165        81        84        0        0    3840:20x12x16
|---- medium [ENABLE,START] 25         4        20        0        1    3840:20x12x16
|---- large  [ENABLE,START]  0         0         0        0        0    3840:20x12x16
`---- x-large [ENABLE,START]  4         0         4        0        0    3840:20x12x16
interactive
|---- interactive_n1 [ENABLE,START] 2    2        0        0        0    50
`---- interactive_n8 [ENABLE,START] 1    1        0        0        0    50
```

使える
キュー名
(リソース
グループ)

現在
使えるか

ジョブ
の総数

実行して
いるジョブ
の数

待たされて
いるジョブ
の数

ノードの
物理構成
情報

JOBスクリプトサンプルの説明（ピュアMPI）

（hello-pure.bash, C言語、Fortran言語共通）

```
#!/bin/bash
```

```
#PJM -L "rscgrp=lecture"
```

```
#PJM -L "node=12"
```

```
#PJM --mpi "proc=192"
```

```
#PJM -L "elapse=1:00"
```

```
mpirun ./hello
```

リソースグループ名
:lecture

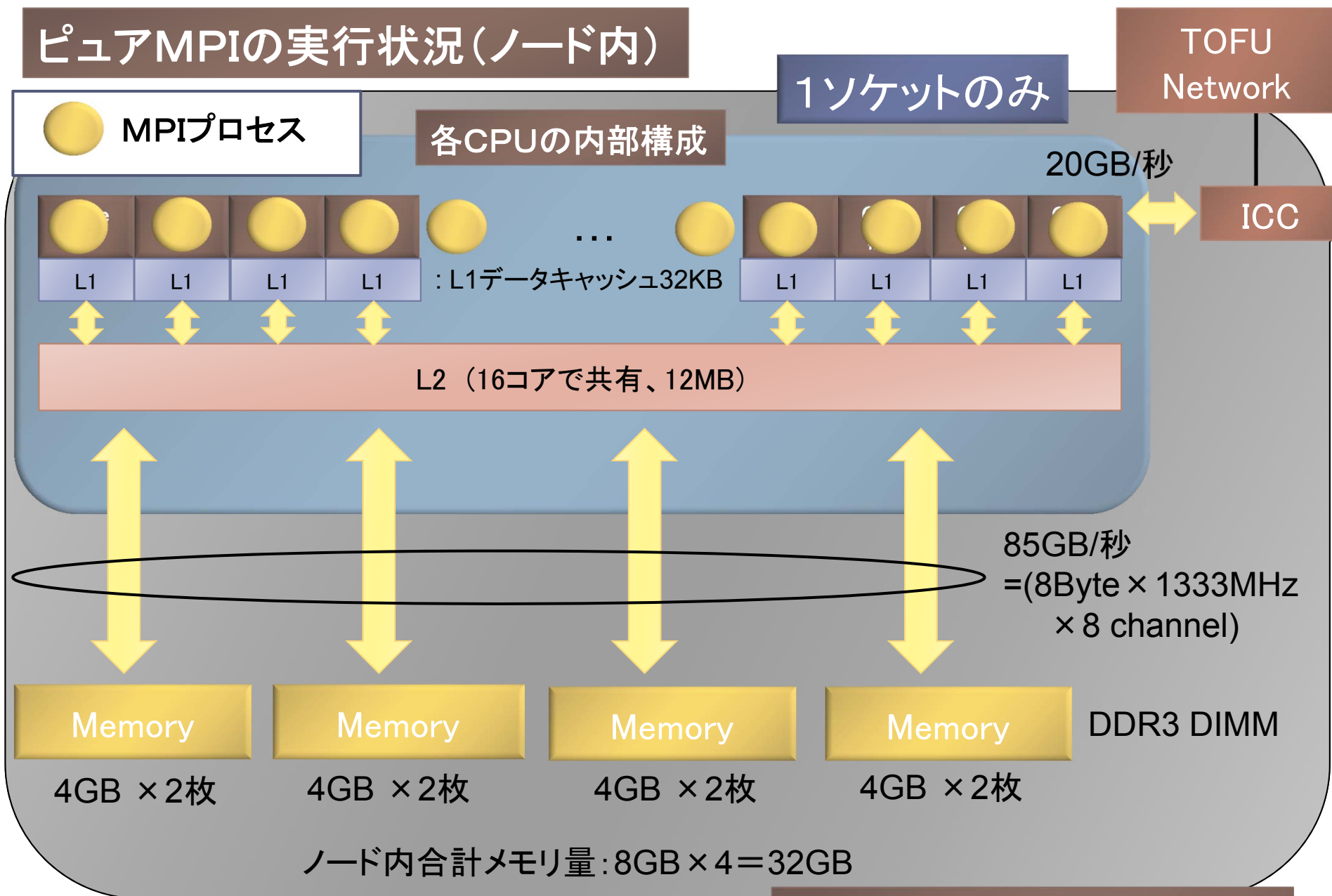
利用ノード数

利用コア数
(MPIプロセス数)

実行時間制限
:1分

MPIジョブを $16 * 12 = 192$ プロセスで実行する。

ピュアMPIの実行状況(ノード内)



FX10計算ノードの構成

並列版Helloプログラムを実行しよう (ピュアMPI)

- ▶ このサンプルのJOBスクリプトは
`hello-pure.bash`
です。
- ▶ 配布のサンプルでは、キュー名
が“`lecture`”になっています
- ▶ `$ emacs hello-pure.bash`
で、“`lecture`” → “`tutorial`”
に変更してください

並列版Helloプログラムを実行しよう (ピュアMPI)

1. Helloフォルダ中で以下を実行する
`$ pjsub hello-pure.bash`
2. 自分の導入されたジョブを確認する
`$ pjstat`
3. 実行が終了すると、以下のファイルが生成される
`hello-pure.bash.eXXXXXX`
`hello-pure.bash.oXXXXXX` (XXXXXXは数字)
4. 上記の標準出力ファイルの中身を見してみる
`$ cat hello-pure.bash.oXXXXXX`
5. “Hello parallel world!”が、
16プロセス*12ノード=192表示されていたら成功。

バッチジョブ実行による標準出力、標準エラー出力

- ▶ バッチジョブの実行が終了すると、標準出力ファイルと標準エラー出力ファイルが、ジョブ投入時のディレクトリに作成されます。
- ▶ 標準出力ファイルにはジョブ実行中の標準出力、標準エラー出力ファイルにはジョブ実行中のエラーメッセージが出力されます。

ジョブ名.oXXXXXX --- 標準出力ファイル

ジョブ名.eXXXXXX --- 標準エラー出力ファイル

(XXXXXX はジョブ投入時に表示されるジョブのジョブID)

Pjsubに関連するTIPS

- ▶ ジョブが実行されるとき、およびNQSジョブが終了したとき、には電子メールが送られる。
- ▶ このメールを読むときは、以下のようにする。

> mail

- ▶ なお、コマンドは以下のとおりです。

- ▶ 番号: メールの表示 Ex. >2

- ▶ h: メール一覧表示

- ▶ s: ファイルへ保存 Ex. > s file

- ▶ d: メールの削除 Ex. > d 1

- ▶ x: コマンドの終了(何もしない)

- ▶ q: コマンドの終了(mboxへ保存)

- ▶ ほかのメールアドレスに飛ばしたい場合は、
ha8000-1.cc.u-tokyo.ac.jp でforward ファイル中にアドレスを記述

並列版Helloプログラムをコンパイルしよう

1. ハイブリッドMPI用の Makefile をコピーする。
`$ cp Makefile_hy16 Makefile`
2. make する。
`$ make clean`
`$ make`
3. 実行ファイル(hello)ができていることを確認する。
`$ ls`
4. JOBスクリプト中(hello-hy16.bash)のキュー名を変更する。“lecture” → “tutorial”に変更する。
`$ emacs hello-hy16.bash`

並列版Helloプログラムを実行しよう (ハイブリッドMPI)

1. Helloフォルダ中で以下を実行する
`$ pjsub hello-hy16.bash`
2. 自分の導入されたジョブを確認する
`$ pjstat`
3. 実行が終了すると、以下のファイルが生成される
`hello-hy16.bash.eXXXXXX`
`hello-hy16.bash.oXXXXXX` (XXXXXXは数字)
4. 上記標準出力ファイルの中身を見してみる
`$ cat hello-hy16.bash.oXXXXXX`
5. “Hello parallel world!”が、
1プロセス*12ノード=12 個表示されていたら成功。

JOBスクリプトサンプルの説明 (ハイブリッドMP I)

(hello-hy16.bash, C言語、Fortran言語共通)

```
#!/bin/bash
#PJM -L "rscgrp=lecture"
#PJM -L "node=12"
#PJM --mpi "proc=12"
#PJM -L "elapse=1:00"
export OMP_NUM_THREADS=16
mpirun ./hello
```

リソースグループ名
:lecture

利用ノード数

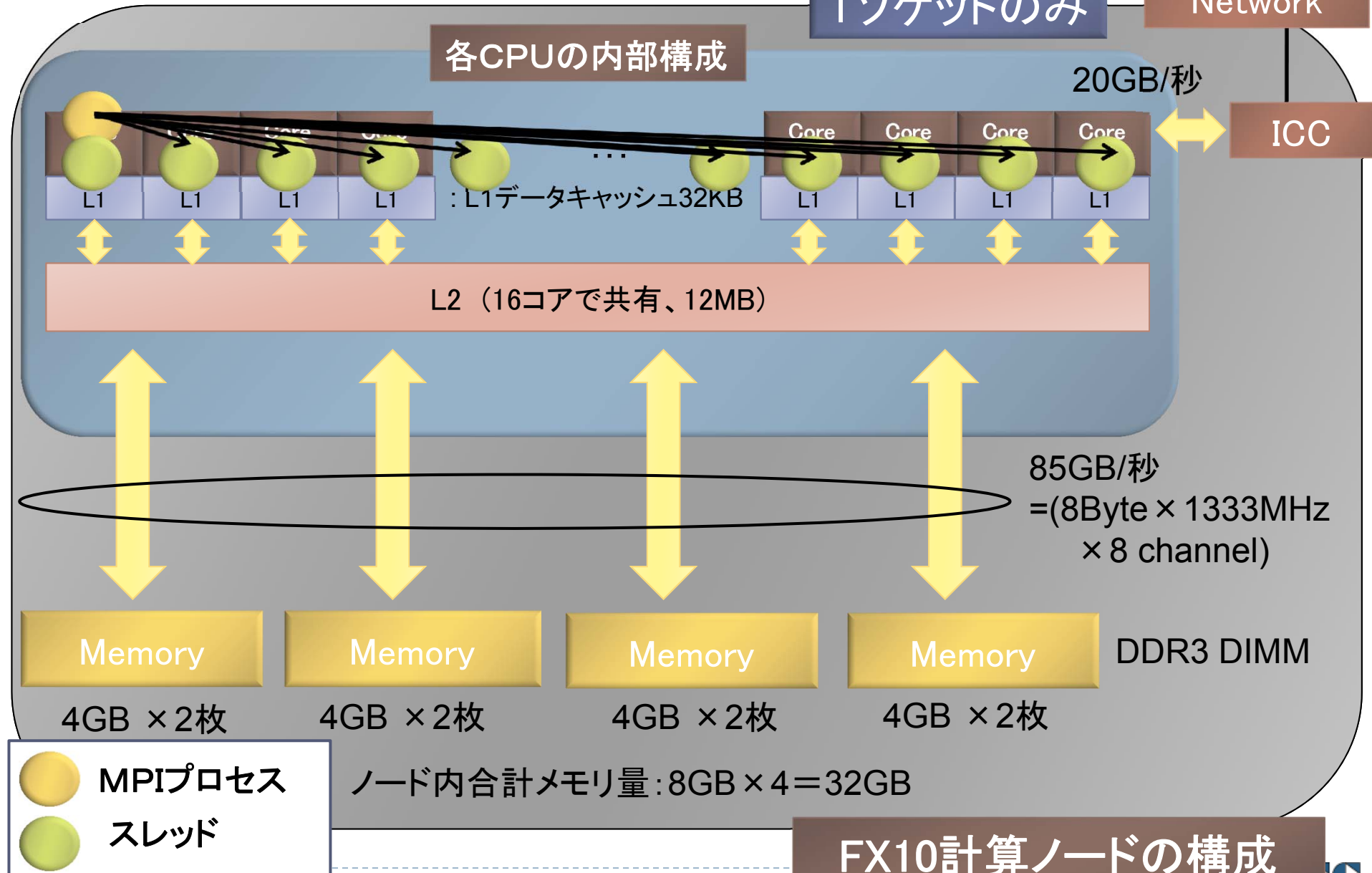
利用コア数
(MPIプロセス数)

実行時間制限: 1分

1 MPIプロセス当たり
16スレッド生成

MPIジョブを $1 * 12 = 12$ プロセスで実行する。

ハイブリッドMPIの実行状況(ノード内)



並列版Helloプログラムの説明（C言語）

このプログラムは、全PEで起動される

```
#include <stdio.h>
#include <mpi.h>
```

```
int main(int argc, char* argv[]) {
```

```
    int  myid, numprocs;
    int  ierr, rc;
```

```
    ierr = MPI_Init(&argc, &argv);
    ierr = MPI_Comm_rank(MPI_COMM_WORLD, &myid);
    ierr = MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
```

```
    printf("Hello parallel world! Myid:%d ¥n", myid);
```

```
    rc = MPI_Finalize();
```

```
    exit(0);
```

```
}
```

MPIの初期化

自分のID番号を取得
:各PEで値は異なる

全体のプロセッサ台数
を取得

:各PEで値は同じ
(演習環境では
192、もしくは12)

MPIの終了

並列版Helloプログラムの説明 (Fortran言語)

このプログラムは、全PEで起動される

program main

common /mpienv/myid,numprocs

integer myid, numprocs
integer ierr

call MPI_INIT(ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD, myid, ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD, numprocs, ierr)

print *, "Hello parallel world! Myid:", myid

call MPI_FINALIZE(ierr)

stop
end

MPIの初期化

自分のID番号を取得
: 各PEで値は異なる

全体のプロセッサ台数
を取得
: 各PEで値は同じ
(演習環境では
192、もしくは12)

MPIの終了

時間計測方法 (C言語)

```
double t0, t1, t2, t_w;  
..  
ierr = MPI_Barrier(MPI_COMM_WORLD);  
t1 = MPI_Wtime();
```

<ここに測定したいプログラムを書く>

```
ierr = MPI_Barrier(MPI_COMM_WORLD);  
t2 = MPI_Wtime();
```

```
t0 = t2 - t1;  
ierr = MPI_Reduce(&t0, &t_w, 1,  
    MPI_DOUBLE, MPI_MAX, 0,  
    MPI_COMM_WORLD);
```

バリア同期後
時間を習得し保存

各プロセッサで、t0の値は
異なる。
この場合は、最も遅いものの
値をプロセッサ0番が受け取る

時間計測方法 (Fortran言語)

```
double precision t0, t1, t2, t_w  
double precision MPI_WTIME
```

```
..  
call MPI_BARRIER(MPI_COMM_WORLD, ierr)  
t1 = MPI_WTIME(ierr)
```

<ここに測定したいプログラムを書く>

```
call MPI_BARRIER(MPI_COMM_WORLD, ierr)  
t2 = MPI_WTIME(ierr)
```

```
t0 = t2 - t1  
call MPI_REDUCE(t0, t_w, 1,  
& MPI_DOUBLE_PRECISION,  
& MPI_MAX, 0, MPI_COMM_WORLD, ierr)
```

バリア同期後
時間を習得し保存

各プロセッサで、t0の値
は異なる。

この場合は、最も遅いも
のの値をプロセッサ0番
が受け取る

MPI実行時のリダイレクトについて

- ▶ FX10スーパーコンピュータシステムでは、
MPI実行時の入出力のリダイレクトができません
 - ▶ ×例) `mpirun ./a.out < ./in.txt > ./out.txt`
- ▶ リダイレクトを行う場合、以下のオプションを指定してください
 - ▶ ○例) `mpirun --stdin ./in.txt --stdout ./out.txt ./a.out`

並列プログラミングの基礎（座学）

東京大学情報基盤センター 准教授 片桐孝洋

本講義の流れ

1. 東大スーパーコンピュータの概略
2. 並列プログラミングの基礎
3. 性能評価指標
4. 基礎的なMPI関数
5. データ分散方式
6. ベクトルどうしの演算
7. ベクトル-行列積
8. リダクション演算

東大スーパーコンピュータの概略

東京大学情報基盤センター スパコン (1 / 2)

HITACHI SR16000

Total Peak performance	: 54.9 TFLOPS
Total number of nodes	: 56
Total memory	: 11200 GB
Peak performance per node	: 980.4 GFLOPS
Main memory per node	: 200GB
Disk capacity	: 556 TB
IBM POWER7 3.83GHz (30.64GFLOPS)	

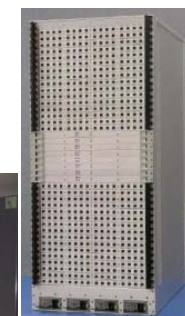
2011年10月～試験運用開始



T2K東大(HA8000クラスシステム)

Total Peak performance	: 140 TFLOPS
Total number of nodes	: 952
Total memory	: 32000 GB
Peak performance per node	: 147.2 GFLOPS
Main memory per node	: 32 GB, 128 GB
Disk capacity	: 1 PB
AMD Quad Core Opteron 2.3GHz (9.2GFLOPS)	

ノード製品名: HITACHI HA8000-tc/RS425



東京大学情報基盤センター スパコン（2 / 2）

Fujitsu PRIMEHPC FX10（FX10スーパーコンピュータシステム）

Total Peak performance	: 1.13 PFLOPS
Total number of nodes	: 4,800
Total memory	: 150TB
Peak performance per node	: 236.5 GFLOPS
Main memory per node	: 32 GB
Disk capacity	: 2.1 PB
SPARC64 IXfx 1.848GHz	

2012年4月2日試験運転開始

2012年7月2日正式運用開始



FX10計算ノードの構成

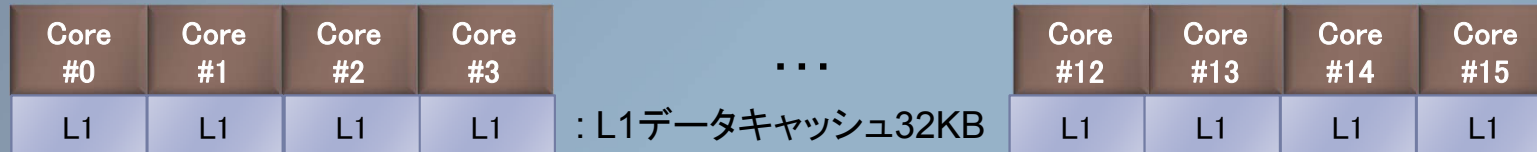
1ソケットのみ

TOFU
Network

各CPUの内部構成

20GB/秒

ICC



L2 (16コアで共有、12MB)

85GB/秒
=(8Byte × 1333MHz
× 8 channel)

Memory

Memory

Memory

Memory

DDR3 DIMM

4GB × 2枚

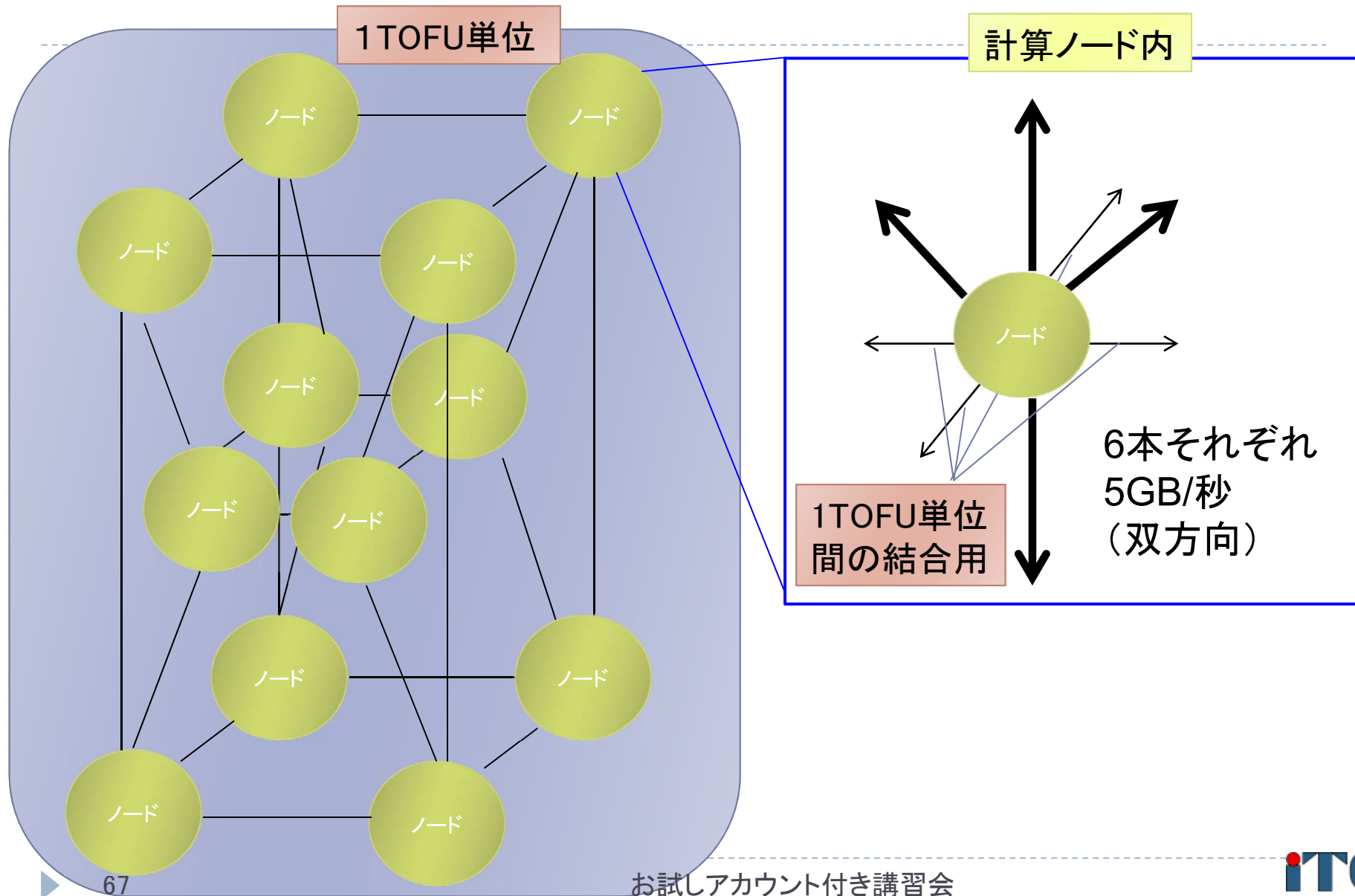
4GB × 2枚

4GB × 2枚

4GB × 2枚

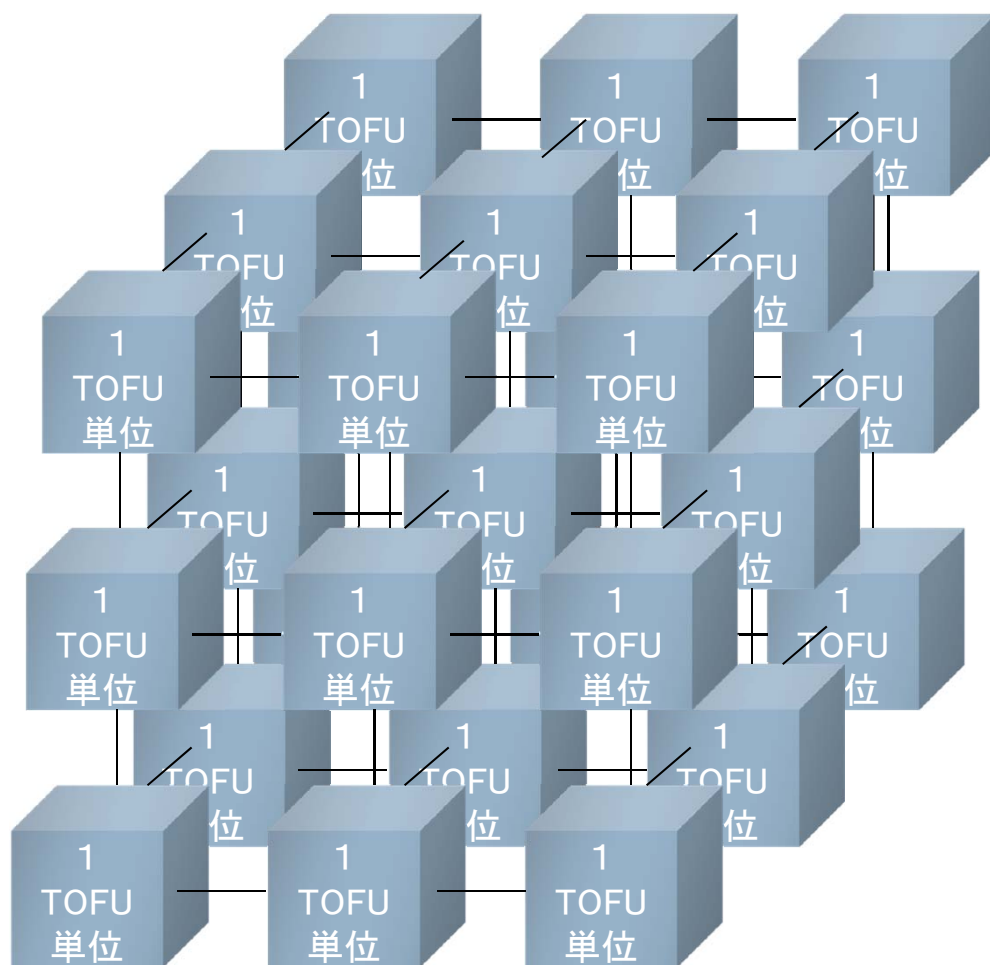
ノード内合計メモリ量: 8GB × 4 = 32GB

FX10の通信網（1 TOFU単位）



FX10の通信網（1 TOFU単位間の結合）

3次元接続



- ユーザから見ると、
X軸、Y軸、Z軸について、
奥の1TOFUと、手前の
1TOFUは、繋がって見えます
(3次元トラス接続)
 - ただし物理結線では
 - X軸はトラス
 - Y軸はメッシュ
 - Z軸はメッシュまたは、
トラス
- になっています

東大情報基盤センターFX10スーパーコンピュータシステムの料金表（2011年4月1日）

▶ パーソナルコース（年間）

▶ コース1： 120, 000円 : 12ノード（優先）、最大24ノードまで

▶ コース2： 250, 000円 : 24ノード（優先）、最大96ノードまで

▶ グループコース

▶ 500, 000円 : 1口、12ノード（優先）、最大1440ノードまで

▶ 以上は、「トークン制」で運営

▶ 申し込みノード（優先ノード）× 360日 × 24時間の「トークン」が与えられる

▶ 優先ノードまでは、トークン消費係数が1.0

▶ 優先ノードを超えると、超えた分は、消費係数が2.0になる

FX10スーパーコンピュータシステムの詳細

▶ 以下のページをご参照ください

- ▶ 利用申請方法
- ▶ 運営体系
- ▶ 料金体系
- ▶ 利用の手引

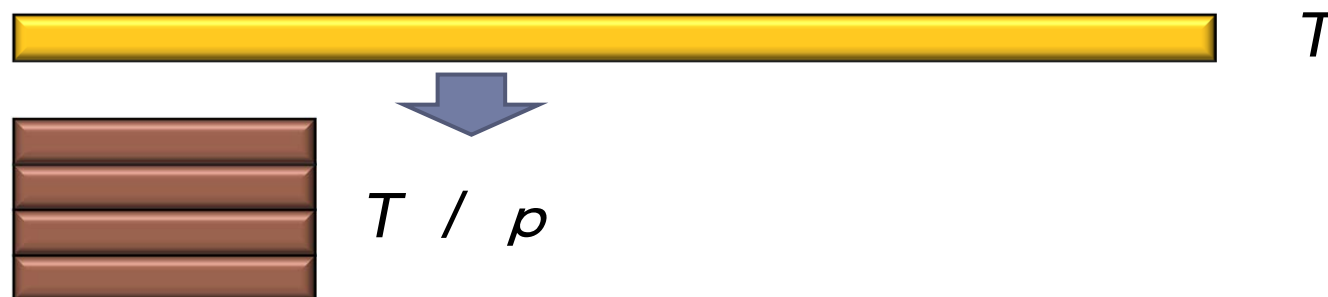
などがご覧になれます。

<http://www.cc.u-tokyo.ac.jp/system/fx10/>

並列プログラミングの基礎

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.

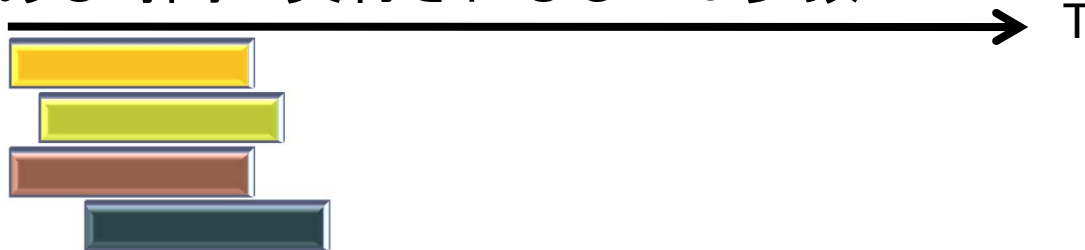


- ▶ 素人考えでは自明。
- ▶ 実際は、できるかどうかは、対象処理の内容(アルゴリズム)で **大きく** 難しさが違う
 - ▶ アルゴリズム上、絶対に並列化できない部分の存在
 - ▶ 通信のためのオーバーヘッドの存在
 - ▶ 通信立ち上がり時間
 - ▶ データ転送時間

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)



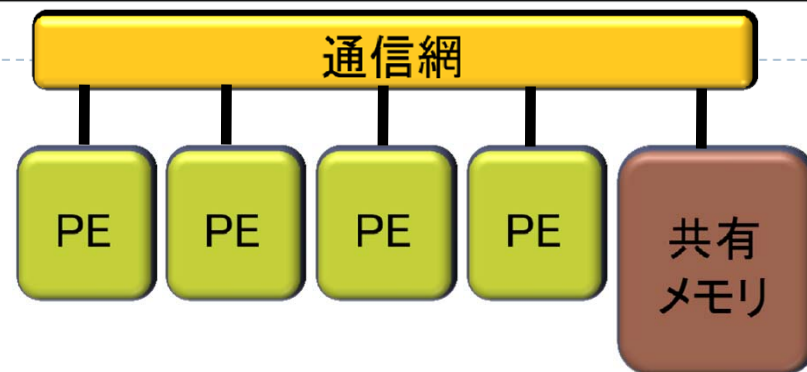
- ▶ 時分割多重、疑似並列
- ▶ OSによるプロセス実行スケジューリング (ラウンドロビン方式)

並列計算機の種類

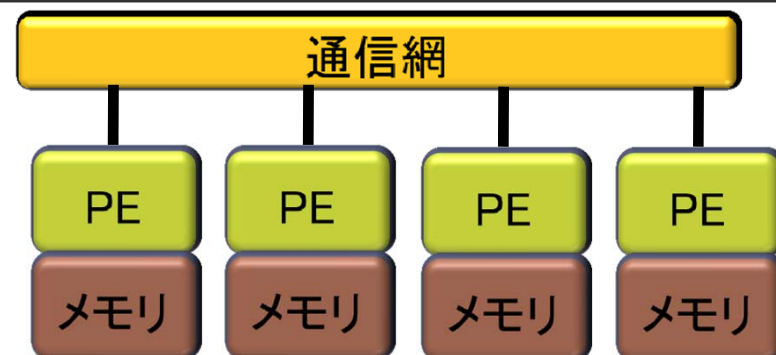
- ▶ Michael J. Flynn教授(スタンフォード大)の種類(1966)
- ▶ 単一命令・単一データ流
(SISD, Single Instruction Single Data Stream)
- ▶ 単一命令・複数データ流
(SIMD, Single Instruction Multiple Data Stream)
- ▶ 複数命令・単一データ流
(MISD, Multiple Instruction Single Data Stream)
- ▶ 複数命令・複数データ流
(MIMD, Multiple Instruction Multiple Data Stream)

並列計算機のメモリ型による分類

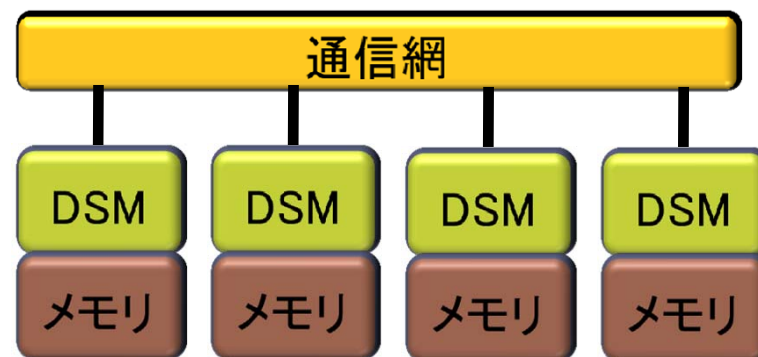
1. 共有メモリ型
(SMP、Symmetric Multiprocessor)



2. 分散メモリ型
(メッセージパッシング)

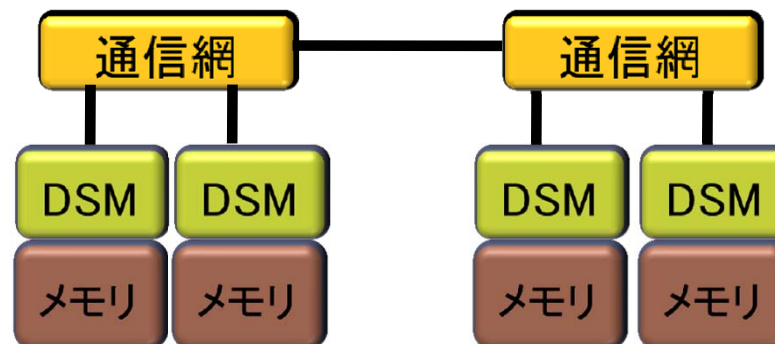


3. 分散共有メモリ型
(DSM、Distributed Shared Memory)



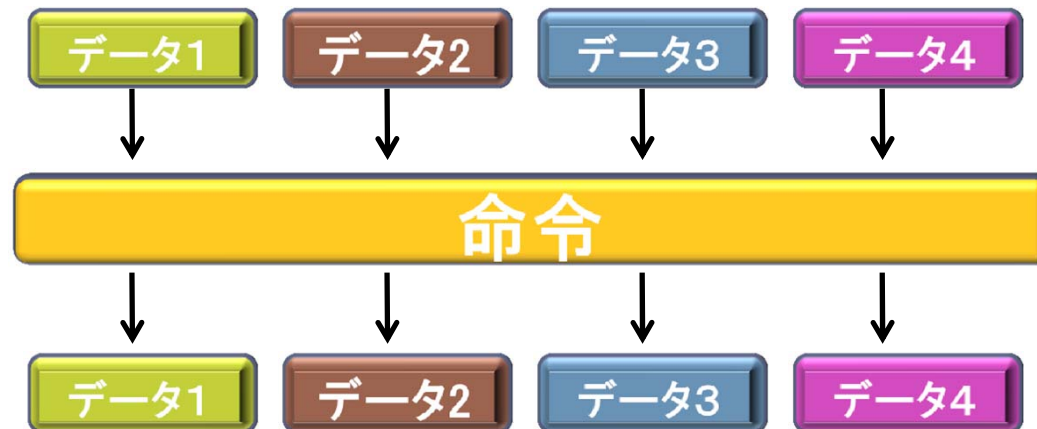
並列計算機のメモリ型による分類

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
- ▶ Open MP
 - ▶ ユーザが並列化指示行を記述

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

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

● 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

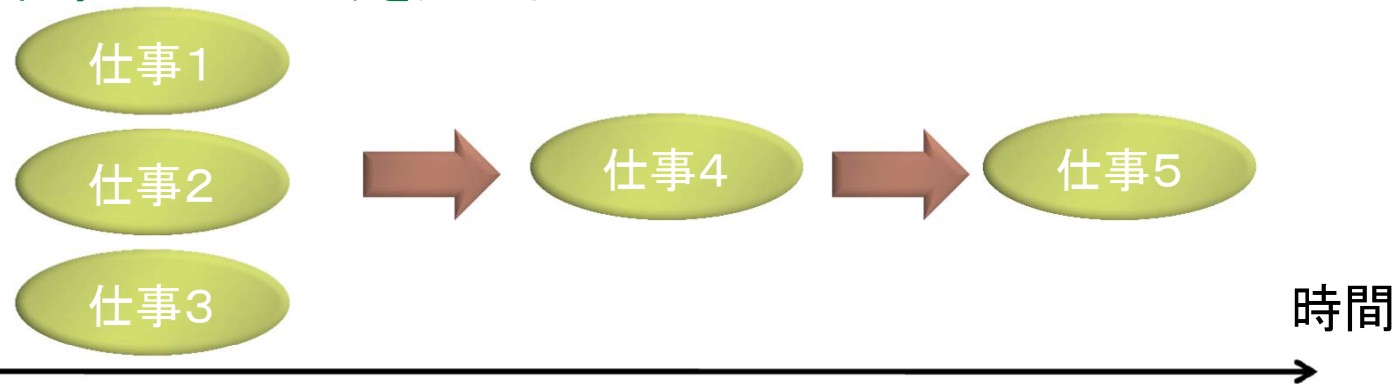
並列に計算：初期データは異なるが演算は同一

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
 - ▶ 片方向通信、RMA (Remote Memory Access)

MPIの経緯 (2/2) MPI3.0策定

- ▶ 以下のページで経緯が公開中
 - ▶ http://meetings.mpi-forum.org/MPI_3.0_main_page.php
- ▶ 注目すべき機能(検討中)
 - ▶ RMA (Remote Memory Access) サポート
 - ▶ FT (Fault Tolerant) stabilization、FT を意識したMPIプログラム作成のAPI群とセマンティクス
 - ▶ MPIT (Performance Tool)、デバッガやパフォーマンスモニタ、開発環境といったツール群とのインターフェース仕様
 - ▶ Neighborhood collectives(物理的に近いノードを対象にした専用コレクティブ通信), 粗通信モデルの為の集団通信API
 - ▶ Fortran bindings – improved Fortran bindings, taking into account Fortran 2003 and 2008 features.
 - ▶ Non-blocking collective I/O – extend non-blocking collective support to include MPI-I/O
 - ▶ Hybrid: Shared memory communicator, Threads, and Endpoint proposals

MPIの実装

▶ 主要なもの

▶ 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

- ▶ ヘッダファイル定義の違いにより動作が異なることがある
- ▶ メーカー独自機能の拡張がなされていることがある

MPIによる通信

- ▶ 郵便物の郵送と同じ
- ▶ 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

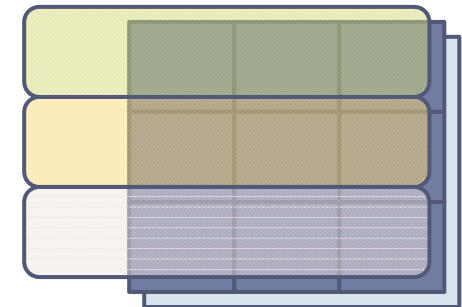
- ▶ MPI_Reduce; MPI_Allreduce; MPI_Barrier;

▶ 時間計測関数

- ▶ MPI_Wtime

コミュニケーター

- ▶ MPI_COMM_WORLDは、コミュニケーターとよばれる概念を保存する変数
- ▶ コミュニケーターは、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～ $\text{numprocs} - 1$ 番までのプロセッサが、1つのコミュニケーターに割り当てられる
 - ▶ 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

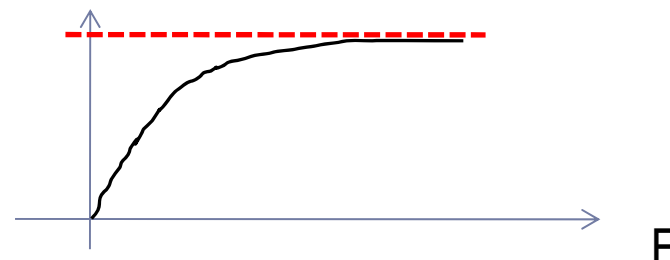
- ▶ 式: $S_p = T_s / T_p$ ($0 \leq S_p$)
- ▶ T_s : 逐次の実行時間、 T_p : P台での実行時間
- ▶ P台用いて $S_p = P$ のとき、理想的な(ideal)速度向上
- ▶ P台用いて $S_p > P$ のとき、スーパリニア・スピードアップ
 - ▶ 主な原因は、並列化により、データアクセスが局所化されて、キャッシュヒット率が向上することによる高速化

▶ 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、台数効果は以下のようになる。

$$S_p = K / (K\alpha / P + K(1-\alpha))$$
$$= 1 / (\alpha / P + (1-\alpha)) = 1 / (\alpha(1/P - 1) + 1)$$

- ▶ 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。

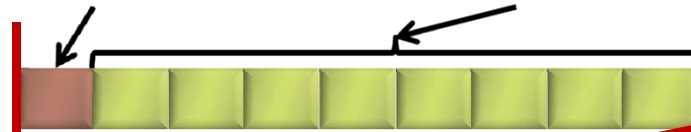
(アムダールの法則)

- ▶ 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

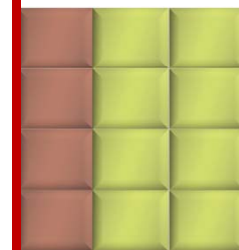
並列化できない部分(1ブロック) 並列化できる部分(8ブロック)

● 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

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(普通は)「プロセッサ」(もしくは、コア)に一対一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 (Processer 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)

基本的なMPI関数

送信、受信のためのインタフェース

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)

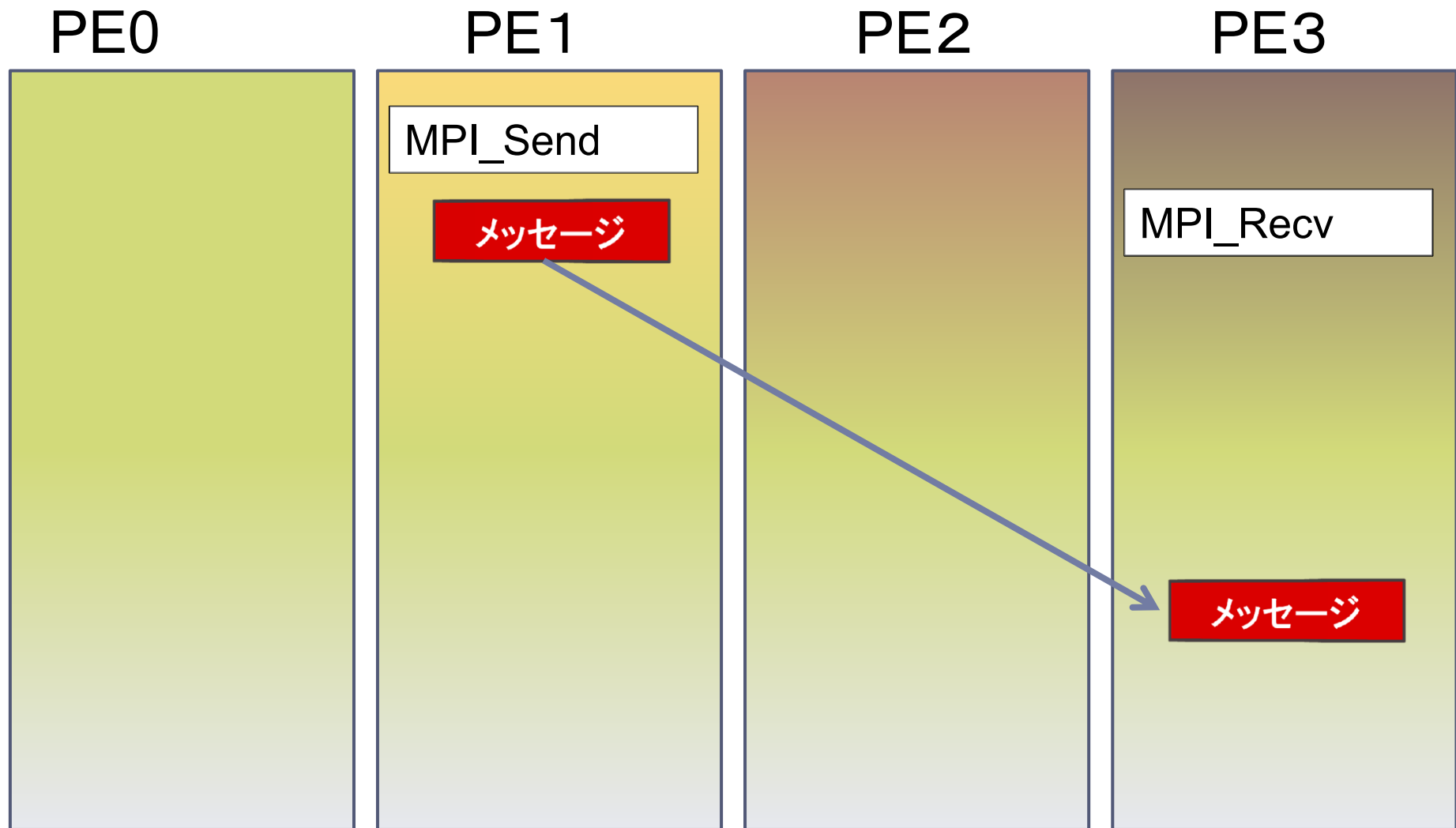
- ▶ **itag** : 整数型。受信したいメッセージに付いているタグの値を指定する。
 - ▶ 任意のタグ値のメッセージを受信したいときは、**MPI_ANY_TAG** を指定する。
- ▶ **icomm** : 整数型。PE集団を認識する番号であるコミュニケータを指定する。
 - ▶ 通常では**MPI_COMM_WORLD** を指定すればよい。
- ▶ **istatus** : MPI_Status型(整数型の配列)。受信状況に関する情報が入る。**かならず専用の型宣言をした配列を確保すること。**
 - ▶ 要素数が**MPI_STATUS_SIZE**の整数配列が宣言される。
 - ▶ 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。
- ▶ **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
- ▶ **idest** : 整数型。送信したいPEのicomm内でのランクを指定する。
- ▶ **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する。
- ▶ **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する。
- ▶ **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）

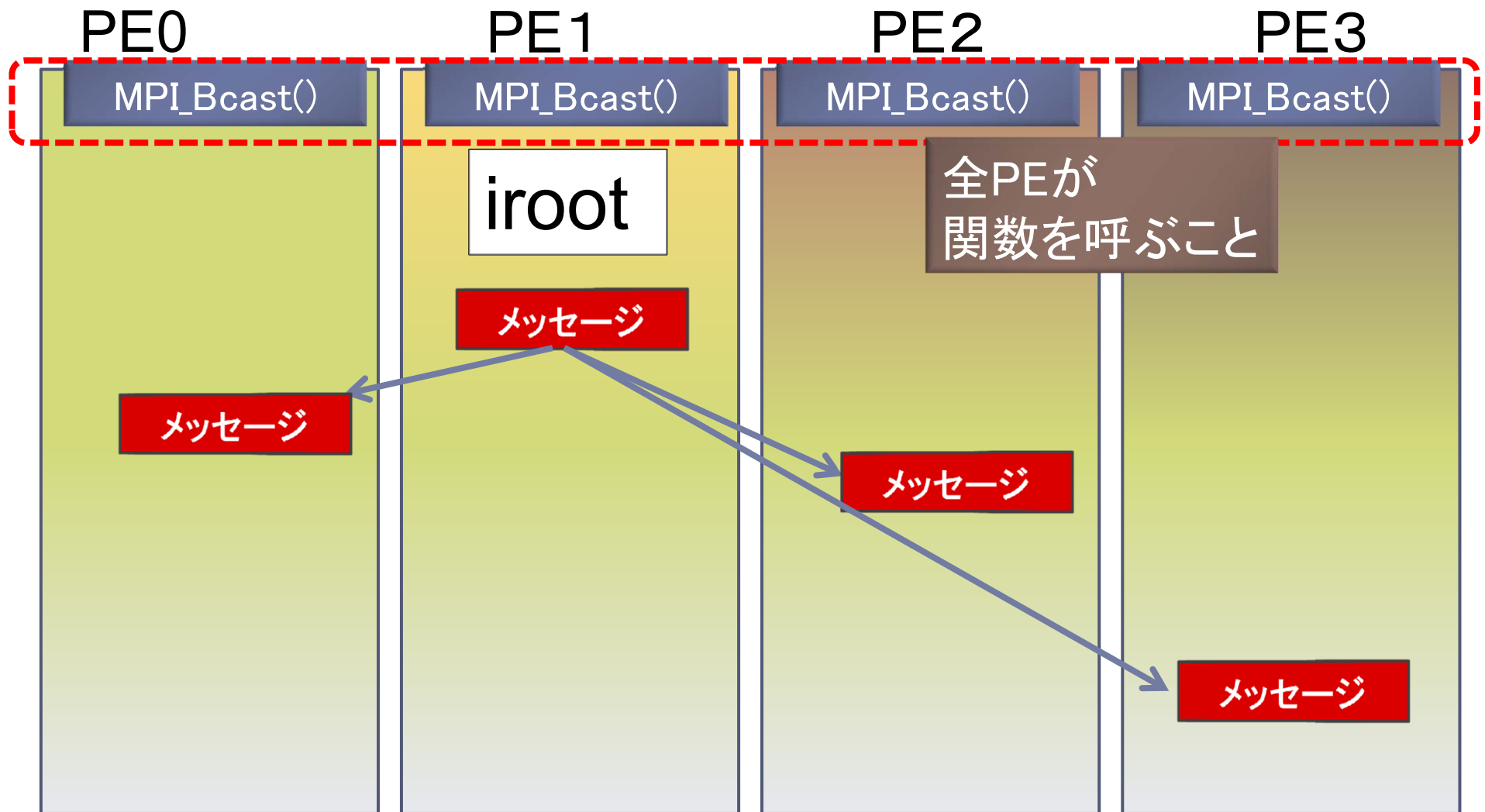


基礎的なMPI関数—MPI_Bcast

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

- ▶ **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。
- ▶ **icommm** : 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）

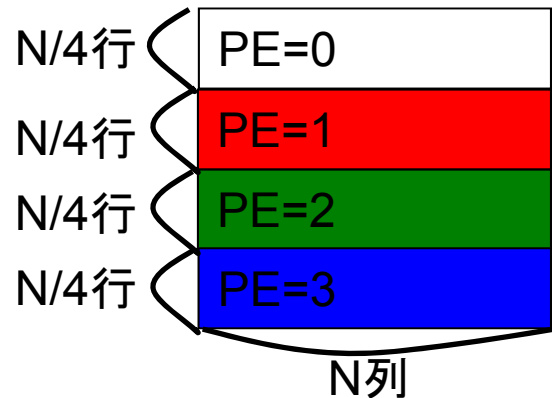


1. 基本演算

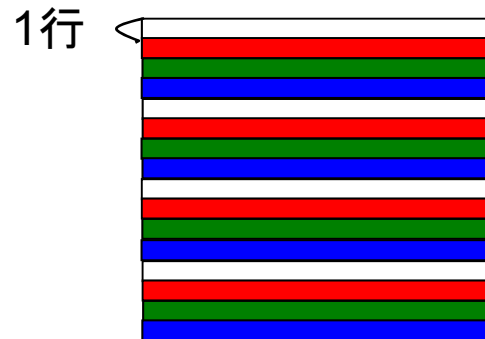
- ▶ 逐次処理では、「データ構造」が重要
- ▶ 並列処理においては、「データ分散方法」が重要になる！
 1. 各PEの「演算負荷」を均等にする
 - ▶ ロード・バランシング： 並列処理の基本操作の一つ
 - ▶ 粒度調整
 2. 各PEの「利用メモリ量」を均等にする
 3. 演算に伴う通信時間を短縮する
 4. 各PEの「データ・アクセスパターン」を高速な方式にする
(= 逐次処理におけるデータ構造と同じ)
- ▶ 行列データの分散方法
 - ▶ <次元レベル>： 1次元分散方式、2次元分散方式
 - ▶ <分割レベル>： ブロック分割方式、サイクリック(循環)分割方式

1.1.1

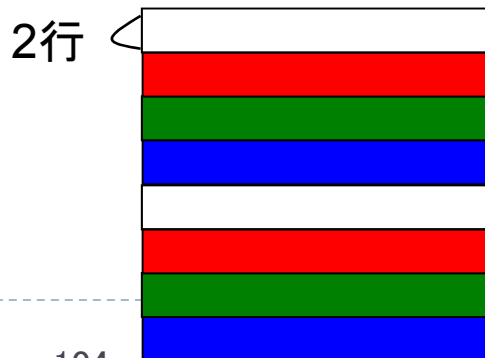
1次元分散



- (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

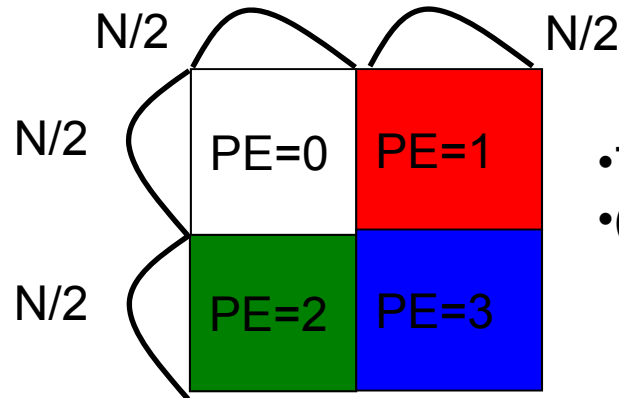


- (行方向)ブロック・サイクリック分割方式
- (Cyclic(2), *) 分散方式

この例の「2」: <ブロック幅>とよぶ

1.1.2

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

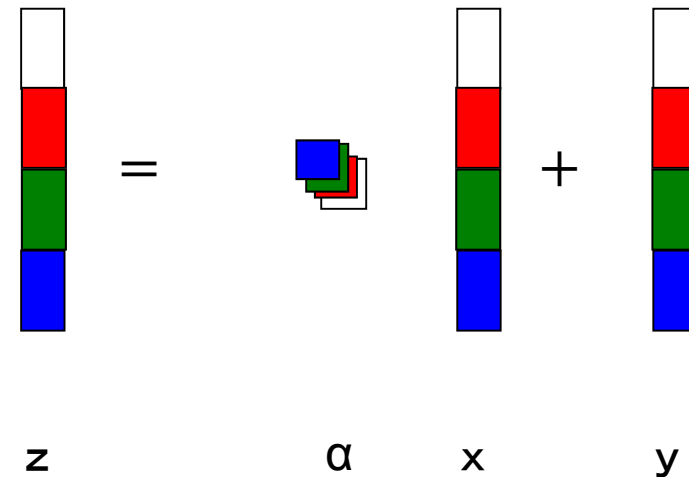
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3
0	1	0	1	0	1	0	1
2	3	2	3	2	3	2	3

1.2 ベクトルどうしの演算

- ▶ 以下の演算

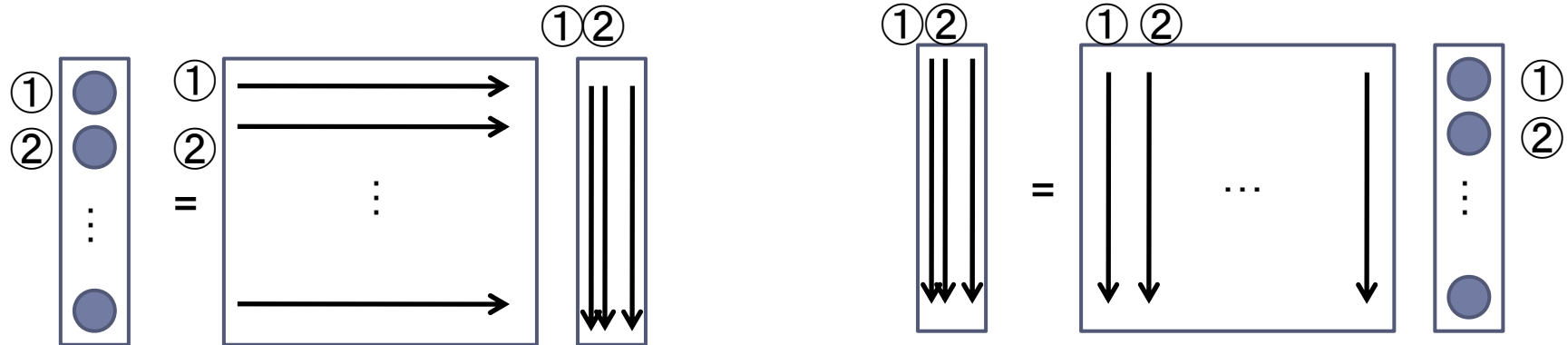
$$z = \alpha x + y$$

- ▶ ここで、 α はスカラー、 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、スカラー 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、スカラーは $O(1)$ のメモリ領域で大丈夫。
→スカラーメモリ領域は無視可能
- ▶ 計算量： $O(N/P)$
- ▶ あまり面白くない



1.3 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

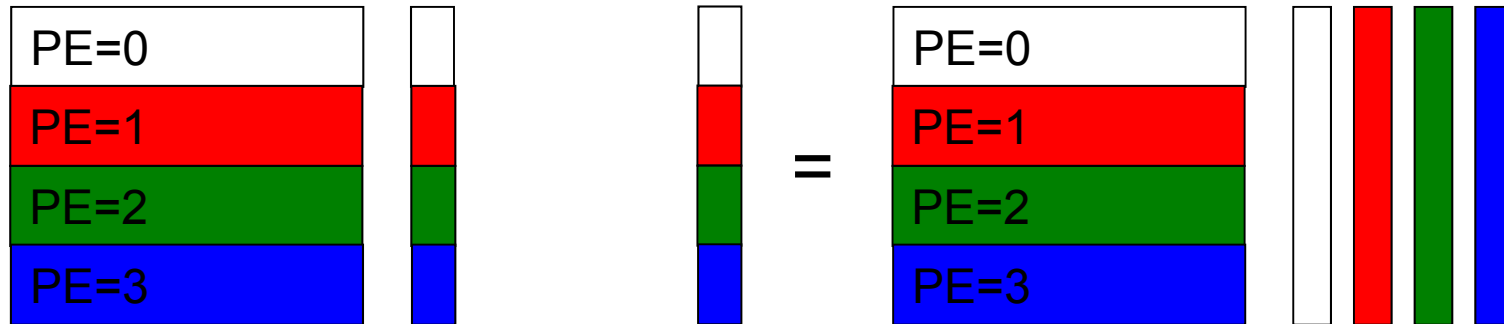
```
for (j=0; j<n; j++) y[j]=0.0;  
for (j=0; j<n; j++) {  
    for (i=0; i<n; i++) {  
        y[i] += a[i][j]*x[j];  
    }  
}
```

＜列方式＞： Fortran言語向き

1.3 行列とベクトルの積

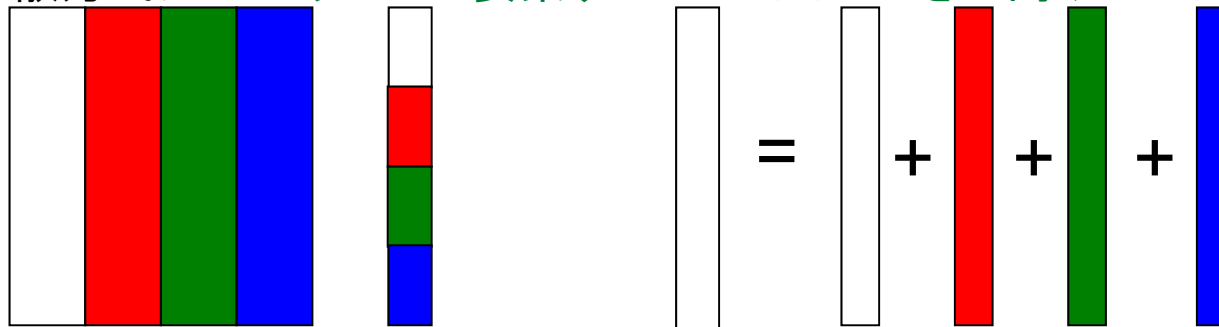
＜行方式の場合＞

＜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 各PE内で行列ベクトル積を行う
を利用し、全PEで所有する

＜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



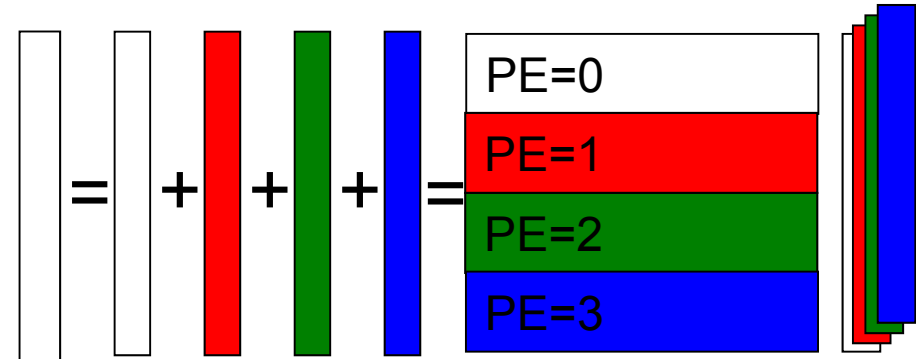
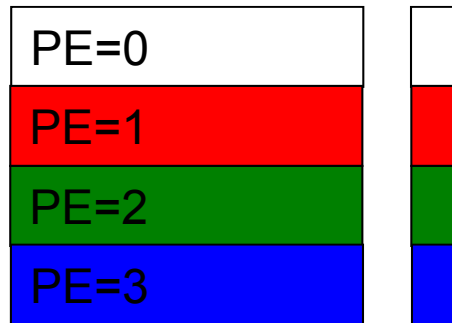
各PE内で行列-ベクトル積
を行う

`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)

1.3 行列とベクトルの積

＜列方式の場合＞

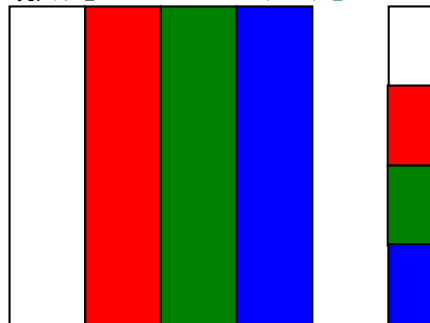
＜行方向分散方式＞ : 無駄が多く使われない



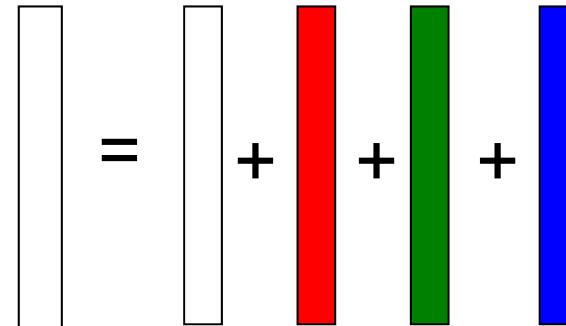
右辺ベクトルを `MPI_Allgather` 関数
を利用して、全PEで所有する

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)

1.7 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種のインタフェースが存在する

1.7 リダクション演算

- ▶ 演算結果に対する所有PEの違い

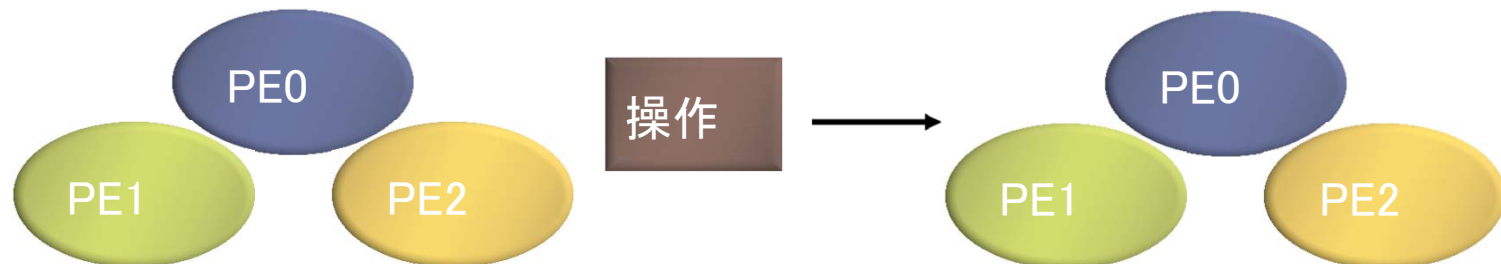
- ▶ **MPI_Reduce**関数

- ▶ 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

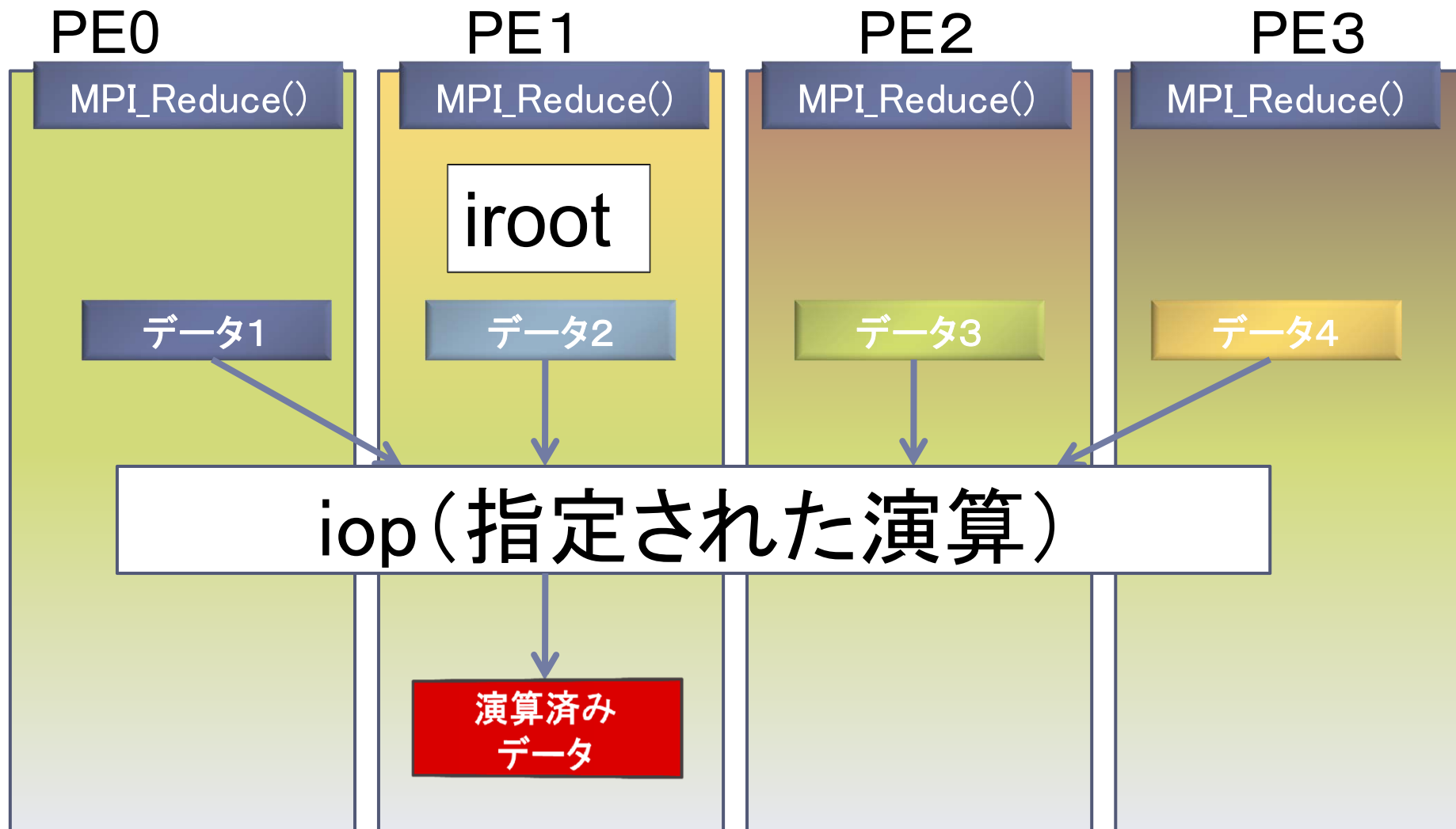
```
▶ ierr = MPI_Reduce(sendbuf, recvbuf, icount,  
    idatatype, iop, iroot, ictop);
```

- ▶ **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。
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／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。

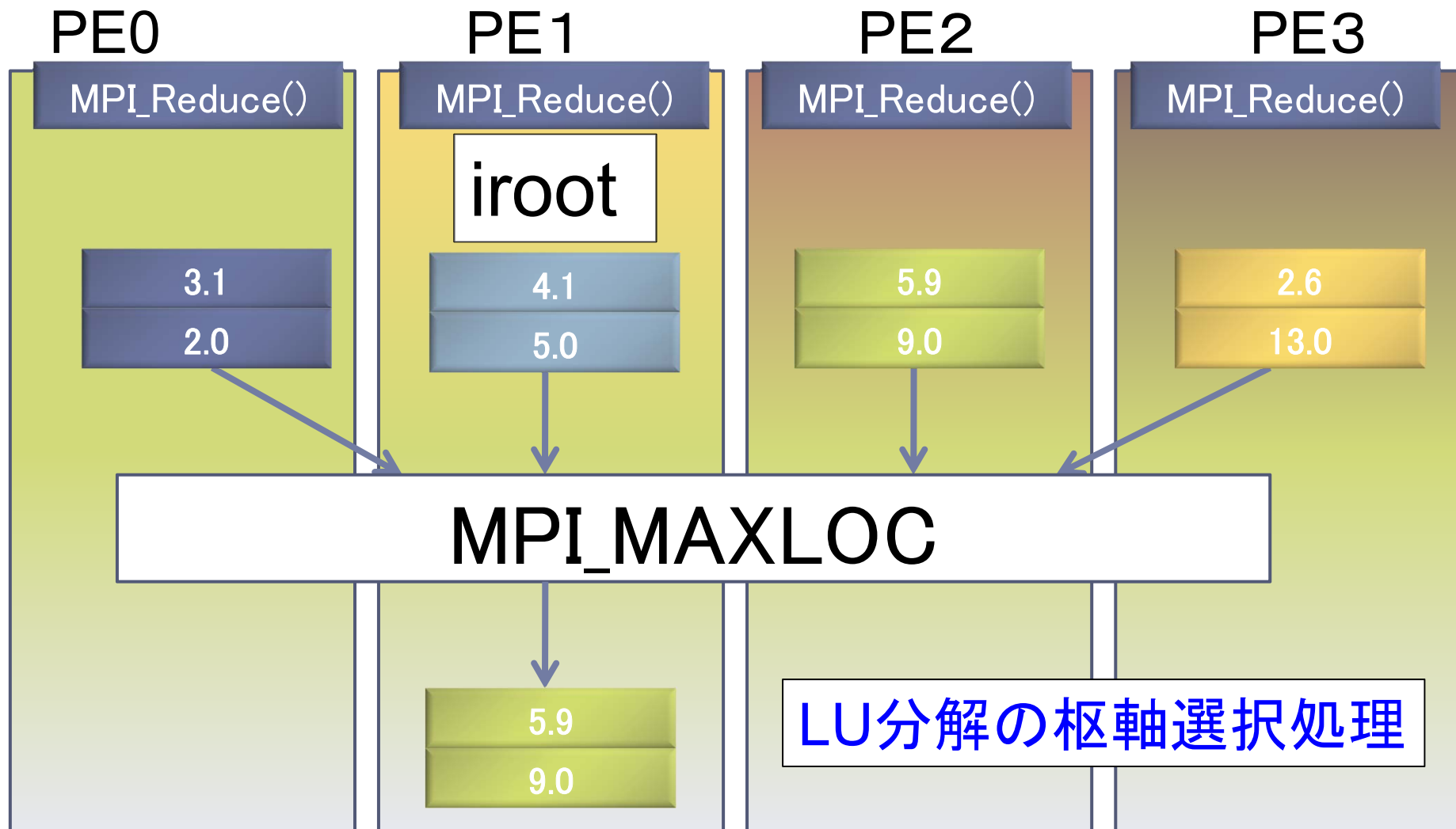
基礎的なMPI関数—MPI_Reduce

- ▶ **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

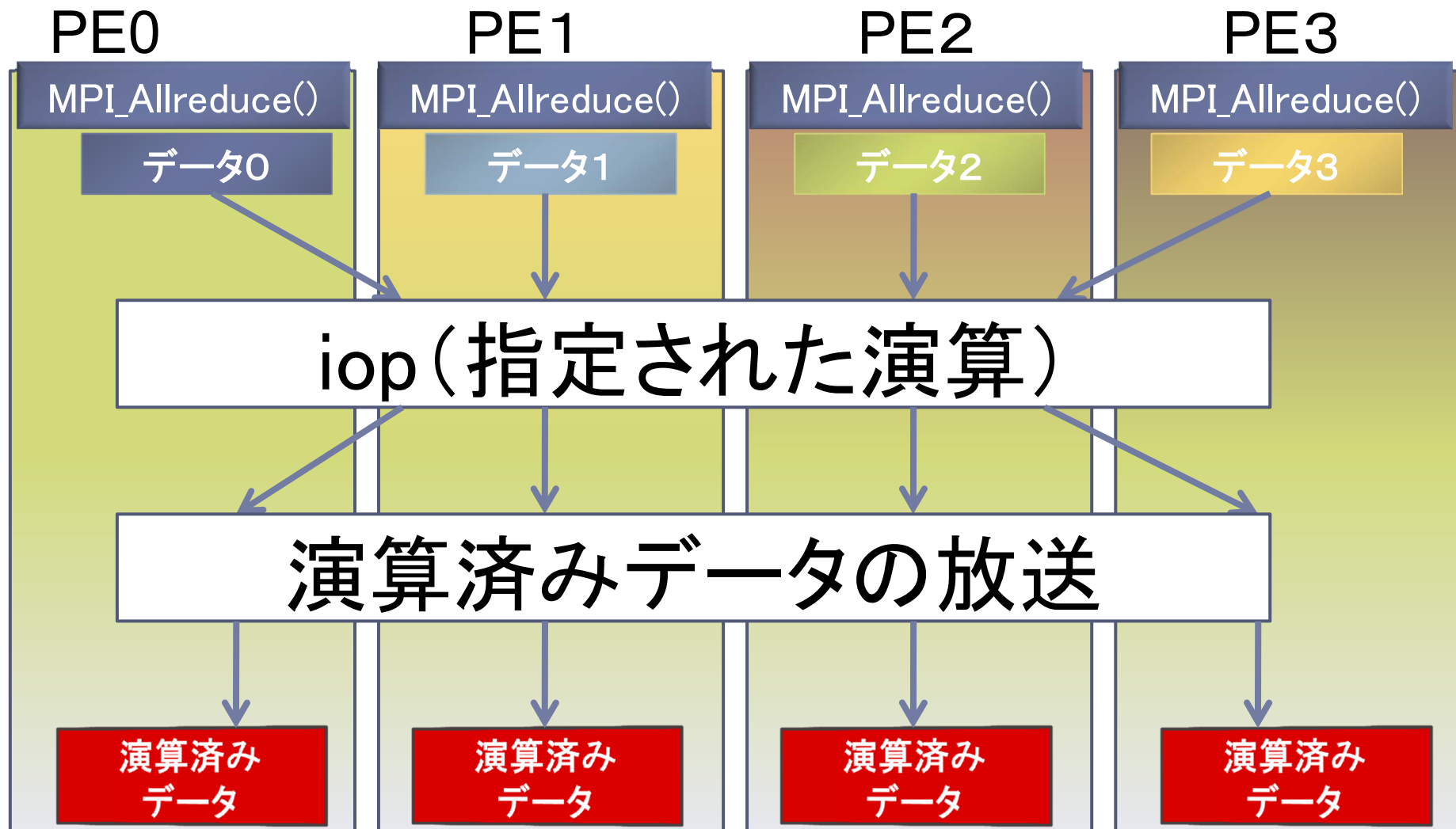
```
▶ ierr = MPI_Allreduce(sendbuf, recvbuf, icount,  
    idatatype, iop, icommm);
```

- ▶ **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。**irroot** で指定したPEのみで書き込みがなされる。
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）



1.7 リダクション演算

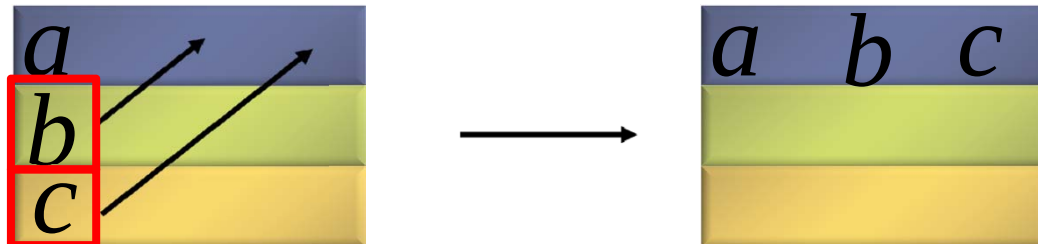
▶ 性能について

- ▶ 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。

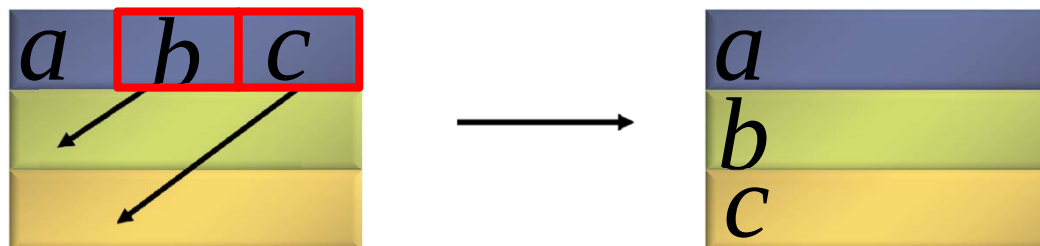
1.4 行列の転置

- ▶ 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



- ▶ MPI_Scatter関数



基礎的なMPI関数—MPI_Gather

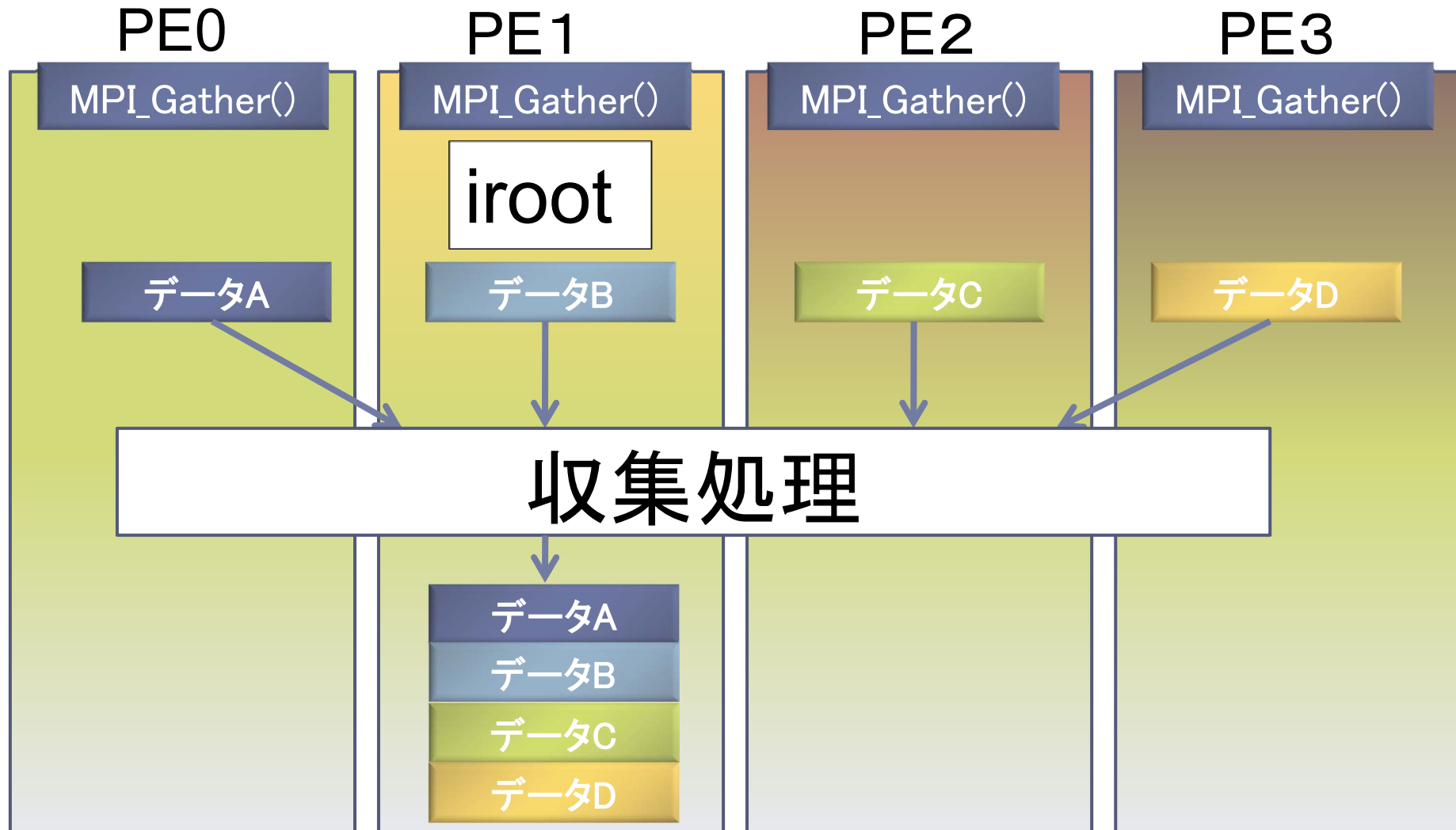
```
▶ ierr = MPI_Gather(sendbuf, isendcount, isendtype,  
                    recvbuf, irecvcount, irecvtype, iroot, ictm);
```

- ▶ **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

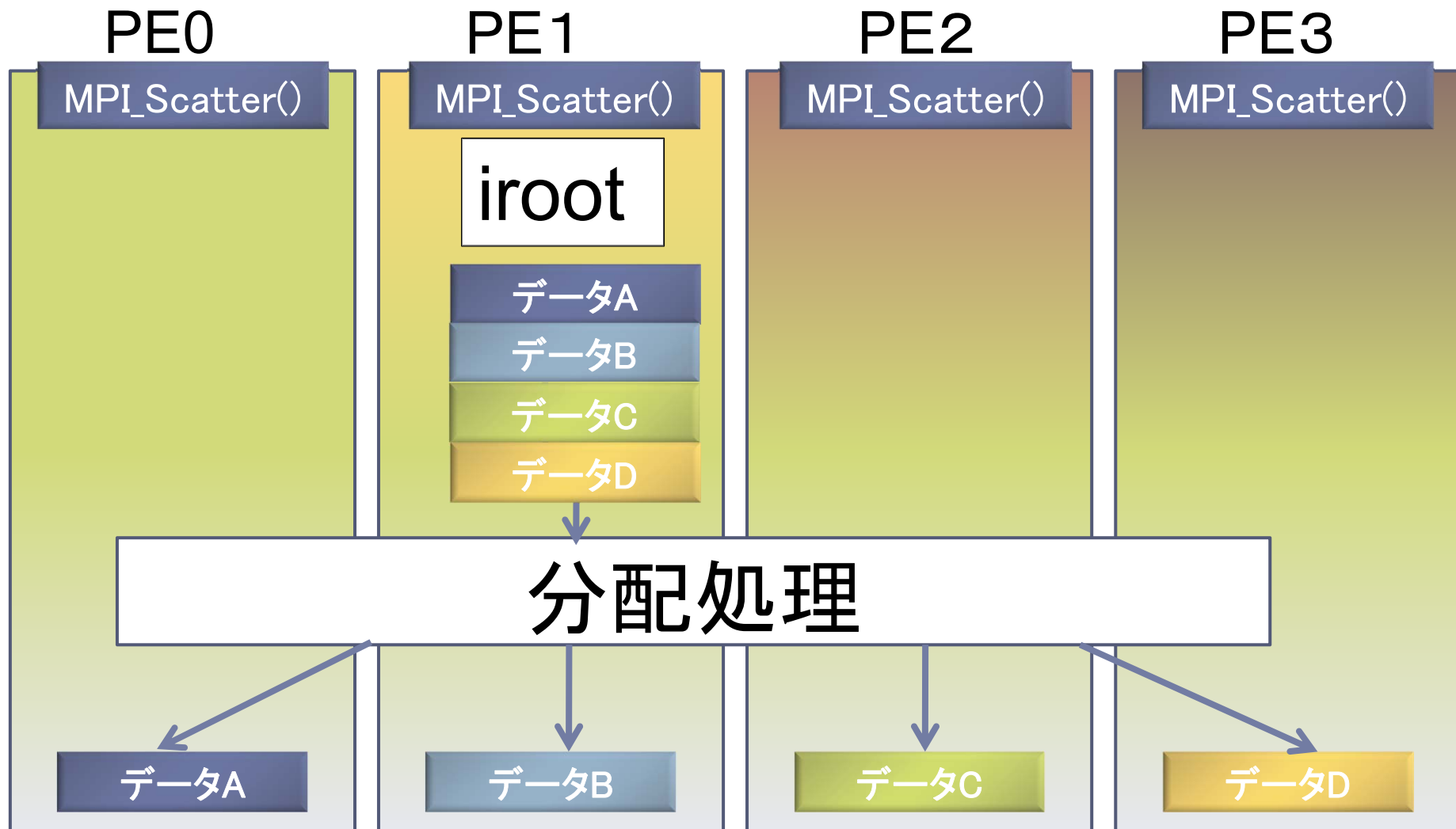
```
▶ ierr = MPI_Scatter ( sendbuf, isendcount, isendtype,  
                      recvbuf, irecvcount, irecvtype, iroot, ictmm);
```

- ▶ **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）



参考文献

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、
理化学研究所情報基盤センタ
(<http://accc.riken.jp/HPC/training/text.html>)
3. Message Passing Interface Forum
(<http://www.mpi-forum.org/>)
4. MPI-Jメーリングリスト
(<http://phase.hpcc.jp/phase/mpi-j/ml/>)
5. 並列コンピュータ工学、富田眞治著、昭晃堂(1996)

MPIプログラム実習 I (演習)

東京大学情報基盤センター 准教授 片桐孝洋

実習課題

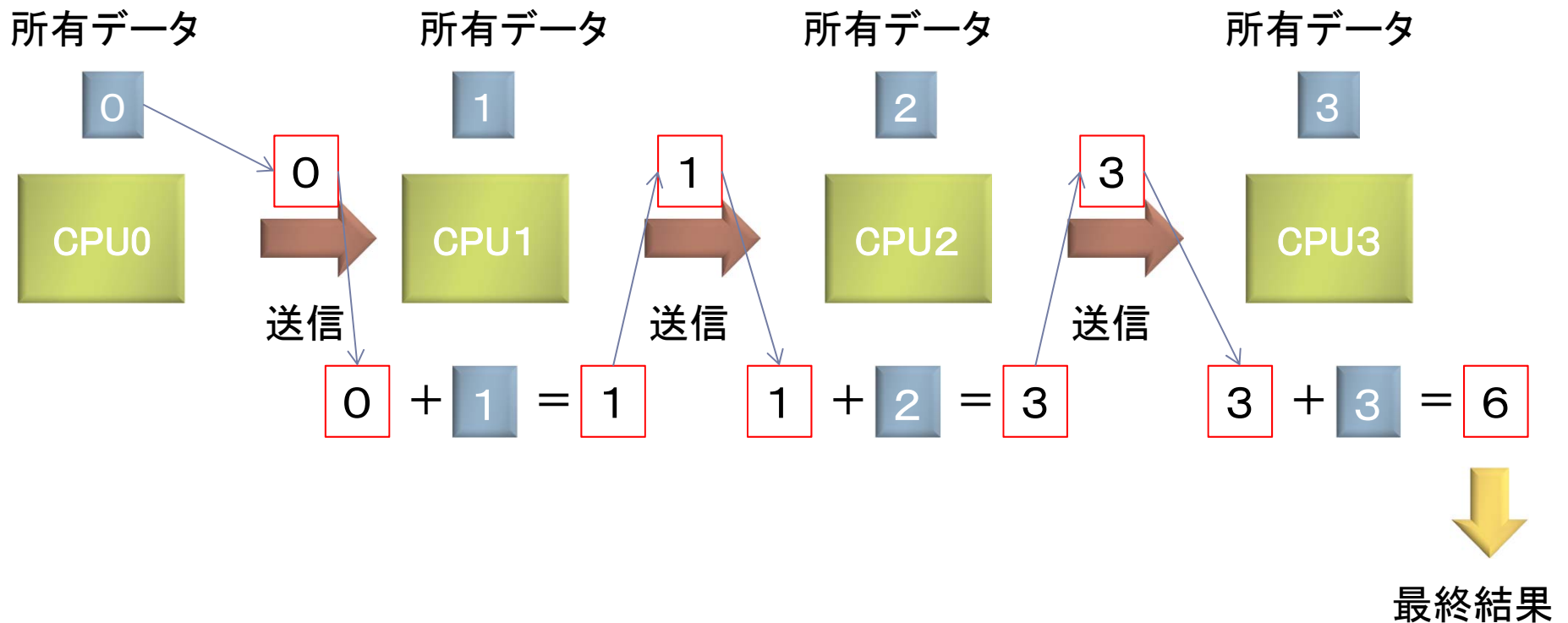
サンプルプログラムの説明

- ▶ **Hello/**
 - ▶ 並列版Helloプログラム
 - ▶ `hello-pure.bash`, `hello-hy16.bash` : NQSジョブスクリプトファイル
- ▶ **Cpi/**
 - ▶ 円周率計算プログラム
 - ▶ `cpi-pure.bash` NQSジョブスクリプトファイル
- ▶ **Wa1/**
 - ▶ 逐次転送方式による総和演算
 - ▶ `wa1-pure.bash` NQSジョブスクリプトファイル
- ▶ **Wa2/**
 - ▶ 二分木通信方式による総和演算
 - ▶ `wa2-pure.bash` NQSジョブスクリプトファイル
- ▶ **Cpi_m/**
 - ▶ 円周率計算プログラムに時間計測ルーチンを追加したもの
 - ▶ `cpi_m-pure.bash` NQSジョブスクリプトファイル

総和演算プログラム（逐次転送方式）

- ▶ 各プロセスが所有するデータを、全プロセスで加算し、あるプロセス1つが結果を所有する演算を考える。
- ▶ 素朴な方法（逐次転送方式）
 1. (0番でなければ)左隣のプロセスからデータを受信する;
 2. 左隣のプロセスからデータが来ていたら;
 1. 受信する;
 2. **<自分のデータ>**と**<受信データ>**を加算する;
 3. (191番でなければ)右隣のプロセスに**<2の加算した結果を>**送信する;
 4. 処理を終了する;
- ▶ 実装上の注意
 - ▶ 左隣りとは、 $(myid-1)$ のIDをもつプロセス
 - ▶ 右隣りとは、 $(myid+1)$ のIDをもつプロセス
 - ▶ $myid=0$ のプロセスは、左隣りはないので、受信しない
 - ▶ $myid=p-1$ のプロセスは、右隣りはないので、送信しない

逐次転送方式（バケツリレー方式） による加算



1 対 1 通信利用例 (逐次転送方式、C言語)

```
void main(int argc, char* argv[]) {
    MPI_Status istatus;
    ....
    dsendbuf = myid;
    drecvbuf = 0.0;
    if (myid != 0) {
        ierr = MPI_Recv(&drecvbuf, 1, MPI_DOUBLE, myid-1, 0,
            MPI_COMM_WORLD, &istatus);
    }
    dsendbuf = dsendbuf + drecvbuf;
    if (myid != nprocs-1) {
        ierr = MPI_Send(&dsendbuf, 1, MPI_DOUBLE, myid+1, 0,
            MPI_COMM_WORLD);
    }
    if (myid == nprocs-1) printf ("Total = %4.2lf ¥n", dsendbuf);
    ....
}
```

受信システム配列の確保

自分より一つ少ない
ID番号(myid-1)から、
double型データ1つを
受信しdrecvbuf変数に
代入

自分より一つ多い
ID番号(myid+1)に、
dsendbuf変数に入っ
ているdouble型データ
1つを送信

1 対 1 通信利用例

(逐次転送方式、Fortran言語)

```
program main
integer istatus(MPI_STATUS_SIZE)
....
dsendbuf = myid
drecvbuf = 0.0
if (myid .ne. 0) then
  call MPI_RECV(drecvbuf, 1, MPI_DOUBLE_PRECISION,
&              myid-1, 0, MPI_COMM_WORLD, istatus, ierr)
endif
dsendbuf = dsendbuf + drecvbuf
if (myid .ne. numprocs-1) then
  call MPI_SEND(dsendbuf, 1, MPI_DOUBLE_PRECISION,
&              myid+1, 0, MPI_COMM_WORLD, ierr)
endif
if (myid .eq. numprocs-1) then
  print *, "Total = ", dsendbuf
endif
....
stop
end
```

受信システム配列の確保

自分より一つ少ない
ID番号(myid-1)から、
double型データ1つを
受信しdrecvbuf変数に
代入

自分より一つ多い
ID番号(myid+1)に、
dsendbuf変数に
入っているdouble型
データ1つを送信

総和演算プログラム（二分木通信方式）

▶ 二分木通信方式

1. $k = 1;$
2. for ($i=0; i < \log_2(nprocs); i++$)
3. if ($(myid \& k) == k$)
 - ▶ $(myid - k)$ 番プロセスからデータを受信;
 - ▶ 自分のデータと、受信データを加算する;
 - ▶ $k = k * 2;$
4. else
 - ▶ $(myid + k)$ 番プロセスに、データを転送する;
 - ▶ 処理を終了する;

総和演算プログラム（二分木通信方式）

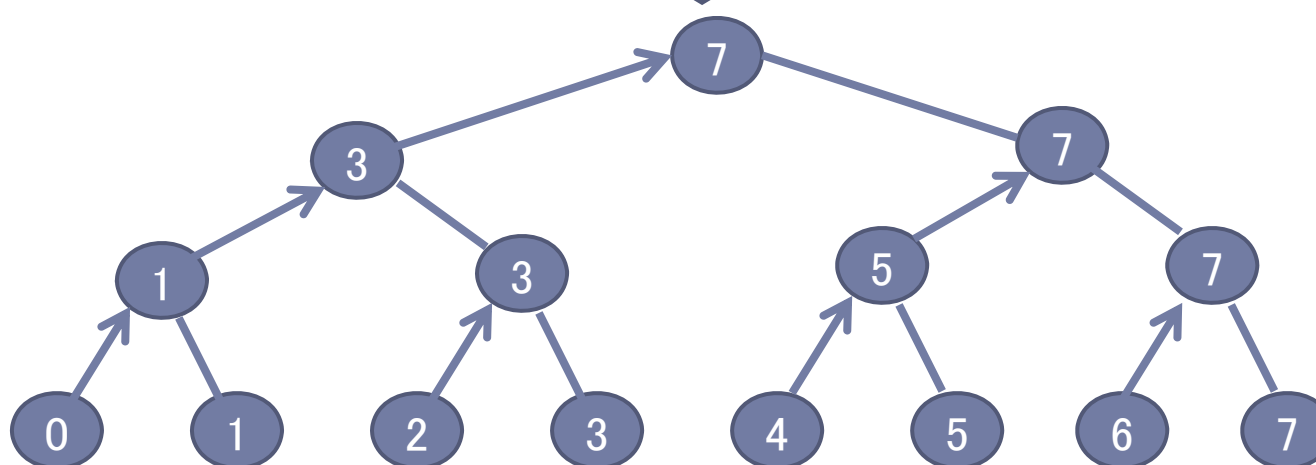
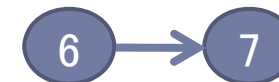
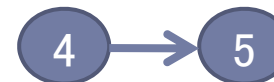
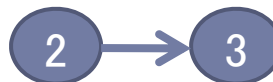
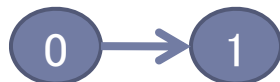
3段目 = $\log_2(8)$ 段目



2段目



1段目



総和演算プログラム（二分木通信方式）

▶ 実装上の工夫

- ▶ **要点:** プロセス番号の2進数表記の情報を利用する
- ▶ 第*i*段において、受信するプロセスの条件は、以下で書ける:
 $\text{myid} \ \& \ k \text{ が } k \text{ と一致}$
 - ▶ ここで、 $k = 2^{(i-1)}$ 。
 - ▶ つまり、プロセス番号の2進数表記で右から*i*番目のビットが立っているプロセスが、送信することにする
- ▶ また、送信元のプロセス番号は、以下で書ける:
 $\text{myid} + k$
 - ▶ つまり、通信が成立するプロセス番号の間隔は $2^{(i-1)}$ ←二分木なので
- ▶ 送信プロセスについては、上記の逆が成り立つ。

総和演算プログラム（二分木通信方式）

- ▶ 逐次転送方式の通信回数
 - ▶ 明らかに、 $nprocs - 1$ 回
- ▶ 二分木通信方式の通信回数
 - ▶ 見積もりの前提
 - ▶ 各段で行われる通信は、完全に並列で行われる（通信の衝突は発生しない）
 - ▶ 段数の分の通信回数となる
 - ▶ つまり、 $\log_2(nprocs)$ 回
- ▶ 両者の通信回数の比較
 - ▶ プロセッサ台数が増すと、通信回数の差（＝実行時間）がとても大きくなる
 - ▶ 1024プロセス構成では、1023回 対 10回！
 - ▶ でも、必ずしも二分木通信方式がよいとは限らない（通信衝突の多発）

性能プロファイラ

- ▶ 富士通コンパイラには、性能プロファイラ機能がある
- ▶ 富士通コンパイラでコンパイル後、実行コマンドで指定し利用する
- ▶ 以下の2種類があります
- ▶ **基本プロファイラ**
 - ▶ **主な用途**: プログラム全体で、最も時間のかかっている関数を同定する
- ▶ **詳細プロファイラ**
 - ▶ **主な用途**: 最も時間のかかっている関数内の特定部分において、メモリアクセス効率、キャッシュヒット率、スレッド実行効率、MPI通信頻度解析、を行う

性能プロファイラの種類の詳細

▶ 基本プロファイラ

- ▶ コマンド例: `fipp -C`
- ▶ 表示コマンド: `fipppx`、GUI(WEB経由)
- ▶ ユーザプログラムに対し一定間隔(デフォルト時100 ミリ秒間隔)毎に割り込みをかけ情報を収集する。
- ▶ 収集した情報を基に、コスト情報等の分析結果を表示。

▶ 詳細プロファイラ

- ▶ コマンド例: `fapp -C`
- ▶ 表示コマンド: GUI(WEB経由)
- ▶ ユーザプログラムの中に測定範囲を設定し、測定範囲のハードウェアカウンタの値を収集。
- ▶ 収集した情報を基に、MFLOPS、MIPS、各種命令比率、キャッシュミス等の詳細な分析結果を表示。

基本プロファイラ利用例

- ▶ プロファイラデータ用の空のディレクトリがないとダメ
- ▶ /Wa2 に Profディレクトリを作成
`$ mkdir Prof`
- ▶ Wa2 の `wa2-pure.bash` 中に以下を記載
`fipp -C -d Prof mpirun ./wa2`
- ▶ 実行する
`$ pjsub wa2-pure.bash`
- ▶ テキストプロファイラを起動
`$ fipp -A -d Prof`

基本プロファイラ出力例 (1/2)

Fujitsu Instant Profiler Version 1.2.0

Measured time : Thu Apr 19 09:32:18 2012
CPU frequency : Process 0 - 127 1848 (MHz)
Type of program : MPI
Average at sampling interval : 100.0 (ms)
Measured range : All ranges
Virtual coordinate : (12, 0, 0)

Time statistics

Elapsed(s)	User(s)	System(s)	
2.1684	53.9800	87.0800	Application
2.1684	0.5100	0.6400	Process 11
2.1588	0.4600	0.6800	Process 88
2.1580	0.5000	0.6400	Process 99
2.1568	0.6600	1.4200	Process 111

...

基本プロファイラ出力例 (2/2)

Procedures profile

Application - procedures

Cost	%	Mpi	%	Start	End	
475	100.0000	312	65.6842	--	--	Application
312	65.6842	312	100.0000	1	45	MAIN_
82	17.2632	0	0.0000	--	--	_GI_sched_yield
80	16.8421	0	0.0000	--	--	_libc_poll
1	0.2105	0	0.0000	--	--	_pthread_mutex_unlock_usercnt

Process 11 - procedures

Cost	%	Mpi	%	Start	End	
5	100.0000	4	80.0000	--	--	Process 11
4	80.0000	4	100.0000	1	45	MAIN_
1	20.0000	0	0.0000	--	--	_GI_sched_yield

...

詳細プロファイラ利用例

- ▶ 測定したい対象に、以下のコマンドを挿入
- ▶ Fortran言語の場合
 - ▶ ヘッダファイル: なし
 - ▶ 測定開始 手続き名: `call fapp_start(name, number, level)`
 - ▶ 測定終了 手続き名: `call fapp_stop(name, number, level)`
 - ▶ 利用例: `call fapp_start("region1",1,1)`
- ▶ C/C++言語の場合
 - ▶ ヘッダファイル: `fj_tool/fapp.h`
 - ▶ 測定開始 関数名: `void fapp_start(const char *name, int number, int level)`
 - ▶ 測定終了 関数名: `void fapp_stop(const char *name, int number, int level)`
 - ▶ 利用例: `fapp_start("region1",1,1);`

詳細プロファイラ利用例

- ▶ 空のディレクトリがないとダメなので、/Wa2 に Profディレクトリを作成

```
$ mkdir Prof
```

- ▶ Wa2のwa2-pure.bash中に以下を記載
(キャッシュ情報取得時)

```
fapp -C -d Prof -L 1 -lhwm -Hevent=Cache mpirun ./wa2
```

- ▶ 実行する

```
$ pjsub wa2-pure.bash
```

詳細プロファイラGPUによる表示例

- ▶ プログラミング支援ツール(FUJITSU Software Development Tools Version 1.2.1 for Windows) をインストール
 - ▶ 以下をアクセス

[https://oakleaf-fx-1.cc.u-tokyo.ac.jp/fsdtfx10tx/
install/index.html](https://oakleaf-fx-1.cc.u-tokyo.ac.jp/fsdtfx10tx/install/index.html)

- ▶ 「ダウンロード」をクリック
- ▶ Serverに、
oakleaf-fx-1.cc.u-tokyo.ac.jp
- ▶ Nameと passwordはセンターから配布したものを入れる
- ▶ うまくいくと、右のボックスがでる



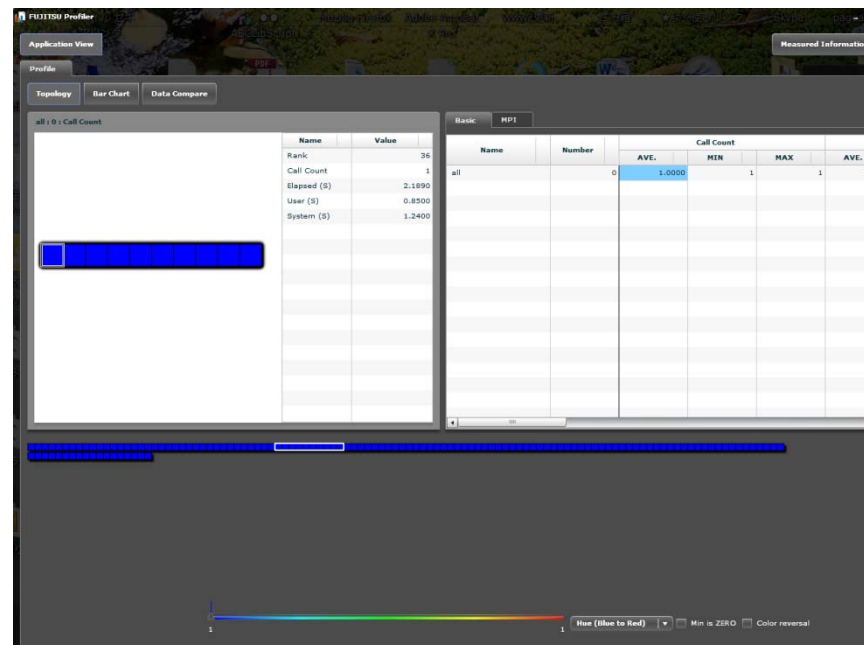
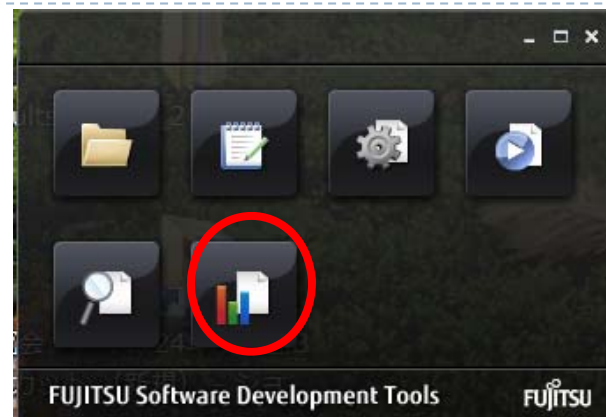
詳細プロファイラGPUによる表示例

- ▶ MACの場合は、以下をアクセス

[https://oakleaf-fx-1.cc.u-tokyo.ac.jp/fsdtfx10tx/
install/mac/index.html](https://oakleaf-fx-1.cc.u-tokyo.ac.jp/fsdtfx10tx/install/mac/index.html)

詳細プロファイラGPUによる表示例

- ▶ 右のボックスで、プロファイラ部分ををクリック
- ▶ プロファイルデータがあるフォルダを指定する
- ▶ うまいくと、右のような解析データが見える



詳細プロファイラで取れるデータ

- ▶ プロセス間の通信頻度情報
(GUI上で色で表示)
- ▶ 各MPIプロセスにおける以下の情報
 - ▶ Cache: キャッシュミス率
 - ▶ Instructions: 実行命令詳細
 - ▶ Mem_access: メモリアクセス状況
 - ▶ Performance: 命令実行効率
 - ▶ Statistics: CPU core 動作状況

演習課題

1. 逐次転送方式のプログラムを実行
 - ▶ Wa1 のプログラム
2. 二分木通信方式のプログラムを実行
 - ▶ Wa2のプログラム
3. 時間計測プログラムを実行
 - ▶ Cpi_mのプログラム
4. プロセス数を変化させて、サンプルプログラムを実行
5. Helloプログラムを、以下のように改良
 - ▶ MPI_Sendを用いて、プロセス0からChar型のデータ“Hello World!!”を、その他のプロセスに送信する
 - ▶ その他のプロセスでは、MPI_Recvで受信して表示する

プログラム実習 I (BLAS) (演習)

東京大学情報基盤センター 准教授 片桐孝洋

演習内容

- ▶ BLASとは
- ▶ GOTO BLASとは
- ▶ LAPACKとは
- ▶ ScaLAPACKとは
- ▶ BLASの利用法と実習 (DGEMM)

1.8 BLASとPBLAS

- ▶ 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。

1.8 BLASとPBLAS

- ▶ 高性能なライブラリの＜作成の手間＞と、＜プログラム再利用性＞を高める目的で提案
- ▶ BLAS演算の性能改善を、個々のユーザが、個々のプログラムで独立して行うのは、
ソフトウェア開発効率が悪い
 - ▶ ＜工学的＞でない
- ▶ 性能を改善するチューニングは、経験のないユーザには無理
 - ▶ ＜職人芸＞ ≠ ＜科学、工学＞
- ▶ BLASは、数学ソフトウェアにおける
＜ソフトウェア工学＞のはしり

1.8 BLASとPBLAS

- ▶ 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

1.8 BLASとPBLAS

▶ 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

1.8 BLASとPBLAS

▶ 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)

1.8 BLASとPBLAS

▶ 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。

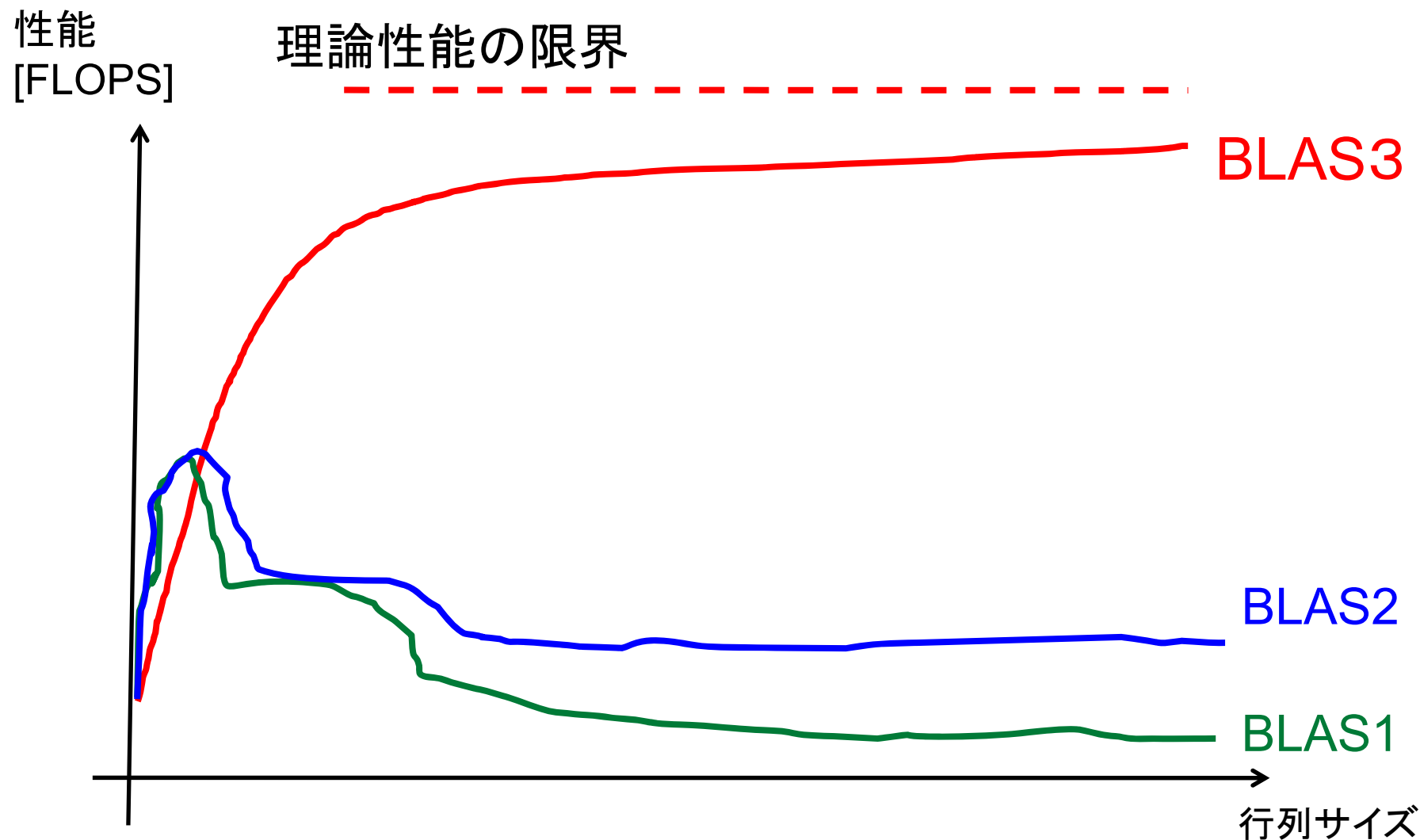
▶ 並列化により1PE当たりのデータ量が減少する。

▶ 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



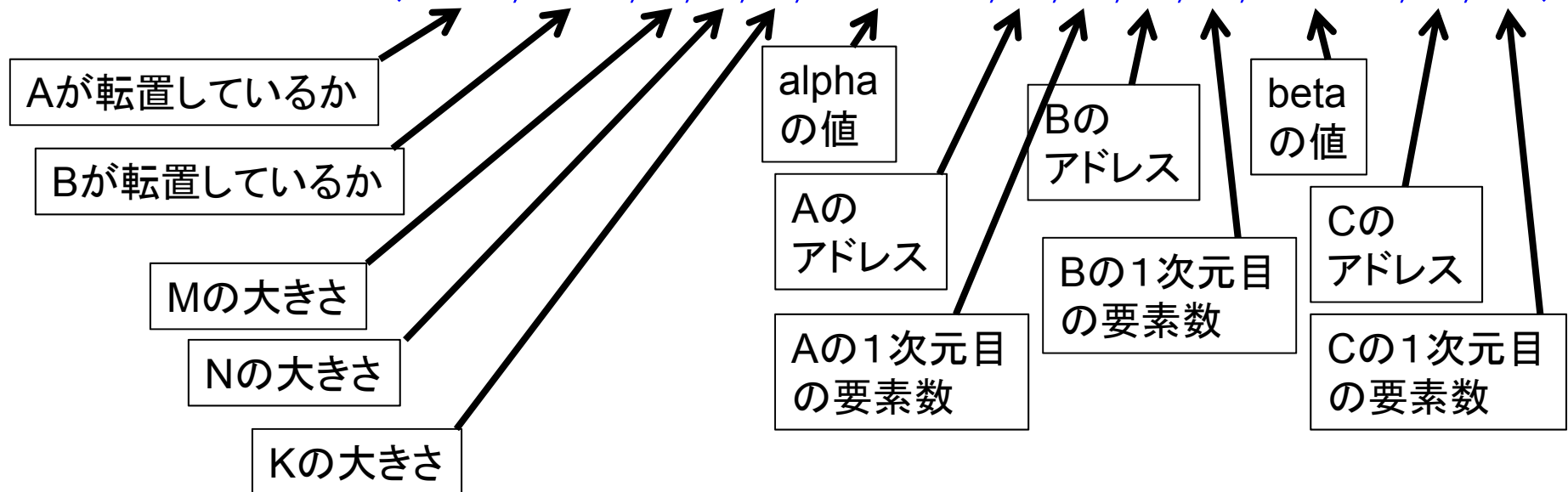
BLAS利用例

▶ 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)



引数が多すぎ！ とても使いにくい！

1.8 BLASとPBLAS

▶ PBLAS

- ▶ 並列版のBLAS
- ▶ BLASとほぼ同じインタフェースをもつ
- ▶ BLAS利用者が、容易に移行できる
- ▶ ライブラリ再利用の目的で開発
 - ▶ 連立一次方程式用ライブラリ: *LINPACK* (リン・パック)
 - ▶ 固有値計算用ライブラリ: *EISPACK* (アイス・パック)
 - ▶ これらを統合したライブラリ: *LAPACK* (エル・エー・パック)
 - アルゴリズムレベルでブロック化して、レベル3 BLASを利用するアルゴリズムを新規開発
 - ▶ LAPACKを分散メモリ並列化: *ScaLAPACK* (スカラ・パック)
 - 並列版BLASとして、PBLASを利用

BLASの機能詳細

- ▶ 詳細はHP: <http://www.netlib.org/blas/>
- ▶ 命名規則: 関数名: **X****YYYY**
 - ▶ **X**: データ型
S:単精度、D:倍精度、C:複素、Z:倍精度複素
 - ▶ **YYYY**: 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

インタフェース例：DGEMM (1 / 4)

▶ DGEMM

(TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDC)

- ▶ $C := \alpha * \text{op}(A) * \text{op}(B) + \beta * C$ の計算をする
- ▶ $\text{op}(X) = X$ もしくは $\text{op}(X) = X'$ (Xの転置行列)

▶ 引数

▶ TRANSA(入力) - CHARACTER*1

- ▶ TRANSA は $\text{op}(A)$ の操作を指定する。以下の文字列を指定。
 - $\text{TRANSA} = 'N'$ もしくは $'n'$, $\text{op}(A) = A$
 - $\text{TRANSA} = 'T'$ もしくは $'t'$, $\text{op}(A) = A'$
 - $\text{TRANSA} = 'C'$ or $'c'$, $\text{op}(A) = A'$

▶ TRANSB(入力) - CHARACTER*1

- ▶ TRANSB は $\text{op}(B)$ の操作を指定する。以下同様。

インタフェース例：DGEMM (2 / 4)

- ▶ **M(入力) – INTEGER**

- ▶ op(A) と 行列 Cの行の大きさを指定する。

- ▶ **N(入力) – INTEGER**

- ▶ op(B) と 行列 Cの列の大きさを指定する。

- ▶ **K(入力) – INTEGER**

- ▶ op(A) の列の大きさ、および op(B) の行の大きさを指定する。

- ▶ **ALPHA(入力) – DOUBLE PRECISION**

- ▶ スカラ値 ALPHAの値を設定する。

- ▶ **A(入力) – DOUBLE PRECISION**

- ▶ 行列Aの配列。大きさは (LDA, ka)で、ka はTRANSA = 'N' or 'n' のときはk。そうでないときは、m。

- ▶ TRANSA = 'N' or 'n' のときは、 $m \times k$ の配列の要素に行列Aを含まないといけない。そうでないときは、 $k \times m$ の配列に行列Aの転置を入れる。

インタフェース例：DGEMM (3 / 4)

▶ LDA(入力) – INTEGER

- ▶ 行列Aの最初の次元数を入れる。TRANSA = 'N' もしくは 'n' なら、LDA は $\max(1, m)$ でなくてはならない。そうでないなら、LDA は $\max(1, k)$ でなくてはならない。

▶ B(入力) – DOUBLE PRECISION

- ▶ 行列Bの配列。大きさは (LDB, kb) で、kb はTRANSA = 'N' or 'n' のときはn。そうでないときは、k。
- ▶ TRANSA = 'N' or 'n' のときは、 $k \times n$ の配列の要素に行列Bを含まないといけない。そうでないときは、 $n \times k$ の配列に行列Bの転置を入れる。

▶ LDB(入力) – INTEGER

- ▶ 行列Bの最初の次元数を入れる。TRANSA = 'N' もしくは 'n' なら、LDA は $\max(1, k)$ でなくてはならない。そうでないなら、LDB は $\max(1, n)$ でなくてはならない。

インタフェース例：DGEMM (4 / 4)

▶ BETA (入力) – DOUBLE PRECISION

- ▶ スカラ値 BETAの値を設定する。

▶ C (入力／出力) – DOUBLE PRECISION

- ▶ 行列Cの配列。
- ▶ 入力時、 $m \times n$ の配列に行列Cを入れる。配列には、BETAが0でない限り、行列Cを入れる。この場合は、Cの入力は必要ない。
- ▶ 出力時、この配列に、 $m \times n$ 行列の演算結果
($\alpha * \text{op}(A) * \text{op}(B) + \beta * C$)が上書きされて戻る。

▶ LDC (入力) – INTEGER

- ▶ 行列Cの最初の次元数を入れる。LDC は $\max(1, m)$ でなくてはならない。

BLASの問題点

▶ BLASの問題点

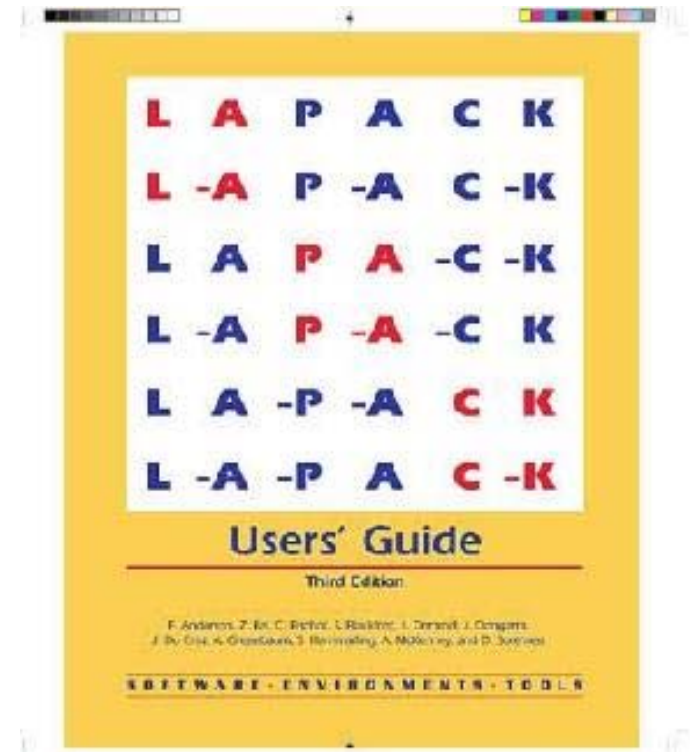
1. BLASやPBLASを用いると、データ再利用性や並列性が低下するかもしれない
 - ▶ 例: レベル1 BLASにおける行列-ベクトル積
2. インタフェースに合わせるため、無駄な処理(配列への代入等)が必要になる場合も
 - ▶ <メモリ浪費>や<演算性能低下>の要因に
3. ソースコードが読みにくくなる
 - ▶ BLASのインタフェースを熟知しないと、かえって処理が理解できない
 - まあ、再利用性・性能とのトレードオフでしょうが

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(:)

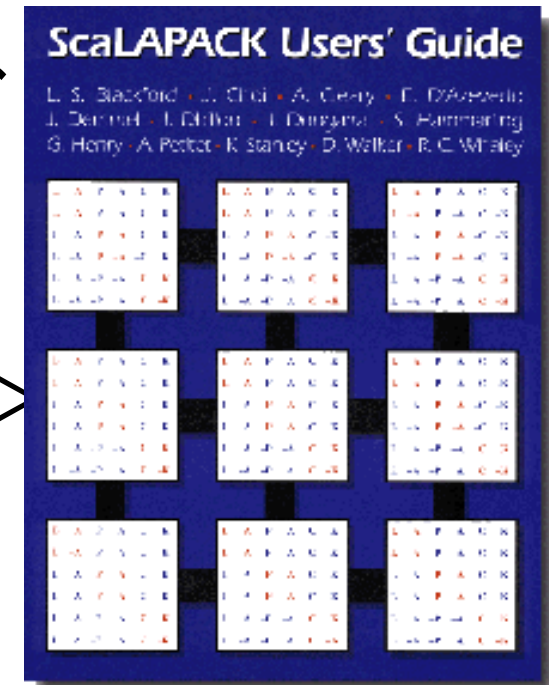
- ▶ 交換行列Aを構成する枢軸のインデックス。行列のi行がIPIVOT(i)行と交換されている。

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 $N \times \text{NRHS}$ 行列Bを入れる。
 - ▶ 出力時は、もし、 $\text{INFO} = 0$ なら、 $N \times \text{NRHS}$ 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。 $\text{LDB} \geq \max(1, N)$ 。
- ▶ **INFO (出力) – INTEGER**
 - ▶ $= 0$: 正常終了
 - ▶ < 0 : もし $\text{INFO} = -i$ なら i -th 行の引数の値がおかしい。
 - ▶ > 0 : もし $\text{INFO} = i$ 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

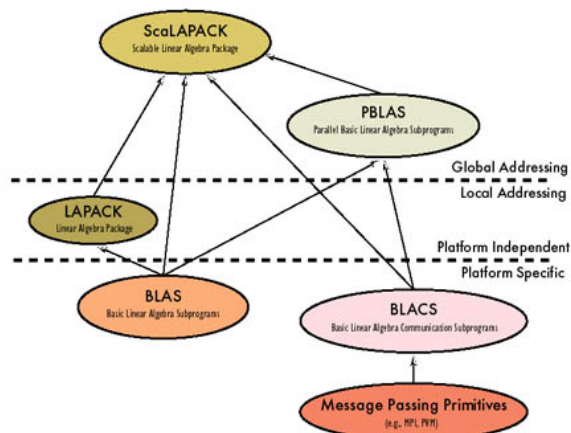
- ▶ 密行列に対する、連立一次方程式の解法、および固有値の解法の“標準”アルゴリズムルーチンの並列化版を無償で提供
- ▶ ユーザインタフェースはLAPACKに<類似>
- ▶ ソフトウェアの<階層化>がされている
 - ▶ 内部ルーチンはLAPACKを利用
 - ▶ 並列インタフェースはBLACS
- ▶ データ分散方式に、2次元ブロック・サイクリック分散方式を採用
- ▶ HP: <http://www.netlib.org/scalapack/>



ScaLAPACKソフトウェア構成図

ScaLAPACK

A Software Library for Linear Algebra Computations on Distributed-Memory Computers



AVAILABLE SOFTWARE:

Dense, Band, and Tridiagonal Linear Systems

- general
- symmetric positive definite

Full-Rank Linear Least Squares

Standard and Generalized

Orthogonal Factorizations

Eigensolvers

- SEP: Symmetric Eigenproblem
- NEP: Nonsymmetric Eigenproblem
- GSEP: Generalized Symmetric Eigenproblem

SVD

Prototype Codes

- HPF interface to ScaLAPACK
- Matrix Sign Function for Eigenproblems
- Out-of-core solvers (LU, Cholesky, QR)
- Super LU
- PBLAS (algorithmic blocking and no alignment restrictions.)

DOCUMENTATION:

ScaLAPACK Users' Guide

http://www.netlib.org/scalapack/slug/scalapack_slug.html

Future Work

- Out-of-core Eigensolvers
- Divide and Conquer routines
- C++ and Java Interfaces

Commercial Use

ScaLAPACK has been incorporated into the following software packages:

- NAG Numerical Library
- IBM Parallel ESSL
- SGI Cray Scientific Software Library

and is being integrated into the VNI IMSL Numerical Library, as well as software libraries for Fujitsu, HP/Convex, Hitachi, and NEC.

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

(参照) <http://www.netlib.org/scalapack/poster.html>

UT The University of Tennessee

ornl Oak Ridge National Laboratory

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)

- ▶ $sub(A) X = sub(B)$ の解の行列 X を計算をする
- ▶ ここで $sub(A)$ は $N \times N$ 行列を分散した $A(IA:IA+N-1, JA:JA+N-1)$ の行列
- ▶ X と B は $N \times NRHS$ 行列を分散した $B(IB:IB+N-1, JB:JB+NRHS-1)$ の行列
- ▶ 行交換の部分枢軸選択付きのLU分解 で $sub(A)$ を $sub(A) = P * L * U$ 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 $sub(A)$ は、連立一次方程式 $sub(A) * X = sub(B)$ 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(LOC_r(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 $N \times \text{NRHS}$ の行列Bの分散されたものを (LLD_B, LOC_c(JB+NRHS-1))に入れる。
 - ▶ 出力時は、もし、INFO = 0 なら、 $N \times \text{NRHS}$ 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
 - ▶ $\text{sub}(B)$ の最初の行のインデックス
- ▶ JB(大域入力) — INTEGER
 - ▶ $\text{sub}(B)$ 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 (3 / 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。

サンプルプログラムの実行 (BLAS DGEMM)

UNIX防忘録

- ▶ emacsの起動: emacs 編集ファイル名
 - ▶ ^x ^s (^はcontrol) : テキストの保存
 - ▶ ^x ^c : 終了
(^z で終了すると、スパコンの負荷が上がる。絶対にしないこと。)
 - ▶ ^g : 訳がわからなくなったとき。
 - ▶ ^k : カーソルより行末まで消す。消した行は一時記憶される。
 - ▶ ^y : ^kで消した行を、現在のカーソルの場所にコピーする。
 - ▶ ^s 文字列 : 文字列の箇所まで移動する。
 - ▶ ^M x goto-line : (^MはESC) 指定した行まで移動する。

UNIX防忘録

- ▶ **rm ファイル名** : ファイル名のファイルを消す。
 - ▶ **rm ***~ : test.c~ などの、~がついたバックアップファイルを消す。
- ▶ **ls** : 現在いるフォルダの中身を見る。
- ▶ **cd フォルダ名** : フォルダに移動する。
 - ▶ **cd ..** : 一つ上のフォルダに移動。
 - ▶ **cd ~** : ルートディレクトリに行く。訳がわからなくなったとき。
- ▶ **cat ファイル名** : ファイル名の中身を見る
- ▶ **make** : 実行ファイルを作る (Makefile があるところ
でしか実行できない)
 - ▶ **make clean** : 実行ファイルを消す。

BLAS DGEMMサンプルプログラムの注意点

- ▶ C言語版、Fortran言語版のファイル名（共通）

`lecBLAS-fx.tar`

- ▶ ジョブスクリプトファイル`mat-mat-blas.bash`
中のキュー名を

`lecture` から `tutorial` に変更してから

`qsub` してください。

- ▶ `lecture` : 実習時間外のキュー（同時実行数1）
- ▶ `tutorial` : 実習時間内のキュー（同時実行数4+）

BLAS DGEMMのサンプルプログラムの実行 (C言語版/Fortran言語版共通)

- ▶ 以下のコマンドを実行する

```
$ cp /home/z30082/lecBLAS-fx.tar ./
```

```
$ tar xvf lecBLAS-fx.tar
```

```
$ cd Mat-Mat-BLAS
```

```
$ cd C //C言語の人
```

- ▶ \$ cd F //Fortran言語の人

```
$ make
```

```
$ pjsub mat-mat-blas.bash
```

- ▶ 実行が終了したら、以下を実行する

```
$ cat mat-mat-blas.bash.XXXXXXX (XXXXXXは数値)
```


BLAS DGEMMのサンプルプログラムの実行 (C言語版)

▶ 以下のような結果が見えれば成功

N = 1000

Mat-Mat time = 6.759452 [sec.]

295.881965 [MFLOPS]

OK!

BLAS DGEMMのサンプルプログラムの実行 (Fortran言語版)

▶ 以下のような結果が見えれば成功

N = 1000

Mat-Mat time[sec.] = 6.788638369180262

MFLOPS = 294.6098887856930

OK!

サンプルプログラムの説明（C言語）

- ▶ `#define N 1000`

の、数字を変更すると、行列サイズが変更できます

- ▶ `#define DEBUG 1`

「1」にすると、行列-行列積の演算結果が検証できます。

- ▶ `MyMatMat`関数の仕様

- ▶ `Double`型 $N \times N$ 行列AとBの行列積をおこない、`Double`型 $N \times N$ 行列Cにその結果が入ります

Fortran言語のサンプルプログラムの注意

- ▶ 行列サイズNNの宣言は、以下のファイルにあります。

mat-mat-blas.inc

- ▶ 行列サイズ変数が、NNとなっています。

integer NN

parameter (NN=1000)

富士通BLAS呼び出しオプション

- ▶ 富士通コンパイラから、BLAS (SSL II (Scientific Subroutine Library II) 数学ライブラリ)を呼び出す場合、以下のオプションを付けます。

●C/Fortran言語共通 (BLASが逐次(1コア実行))

mpifrtpx <プログラム名> **-SSL2** :Fortran言語

mpiccpix <プログラム名> **-SSL2** :C言語

●C/Fortran言語共通 (BLASがスレッド実行)

mpifrtpx <プログラム名> **-SSL2BLAMP** :Fortran言語

mpiccpix <プログラム名> **-SSL2BLAMP** :C言語

サンプルプログラムの説明 (BLAS)

- ▶ **1コア(逐次)実行版です**
- ▶ スレッド並列化版は、対応するスレッド並列化版をリンクしてコンパイルの上、並列実行数の指定をする必要があります。
 - ▶ BLASスレッドでは、**OpenMPのスレッド数指定法と同じ方法**で、実行するスレッド数が指定できます
 - ▶ **OMP_NUM_THREADS**環境変数に実行スレッド数を入れる
 - ▶ ジョブスクリプト内に、以下を記載(16スレッド時):
extern OMP_NUM_THREADS=16

演習課題 1 (BLAS)

1. **MyMatMat**関数(手続き)の行列積コードを、BLAS DCGMMルーチンの呼び出しにより高速化してください
 - ▶ 引数の並びに注意してください。
 - ▶ C言語は、DGEMM のFortran手続きを呼び出します。以下に注意してください。
 - ▶ 関数名: DGEMM_
 - ▶ 全ての引数が **ポインタ**引き渡し

演習課題 2 (BLAS)

2. スレッド並列版のDGEMMを呼び出し、
いろいろなスレッド数で実行してください。
また逐次実行に比べ、どれだけ高速化されたか、
性能評価をしてください。

BLAS DGEMM回答

BLAS DGEMMの回答 (C言語版)

```
double ALPHA, BETA;  
char TEX[1] = {'N'};  
  
ALPHA=1.0;  
BETA=1.0;  
dgemm_(&TEX, &TEX, &n, &n, &n, &ALPHA,  
      A, &n, B, &n, &BETA, C, &n);
```

BLAS DGEMMの回答 (Fortran言語版)

double precision ALPHA,BETA

ALPHA=1.0d0

BETA=1.0d0

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

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. スパース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、
理化学研究所情報基盤センタ
(<http://accc.riken.jp/HPC/training/text.html>)
3. Message Passing Interface Forum
(<http://www.mpi-forum.org/>)
4. MPI-Jメーリングリスト
(<http://phase.hpcc.jp/phase/mpi-j/ml/>)
5. 並列コンピュータ工学、富田眞治著、昭晃堂(1996)
6. 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)

プログラム実習Ⅱ (LAPACK, ScaLAPACK) (演習)

東京大学情報基盤センター 准教授 片桐孝洋

演習内容

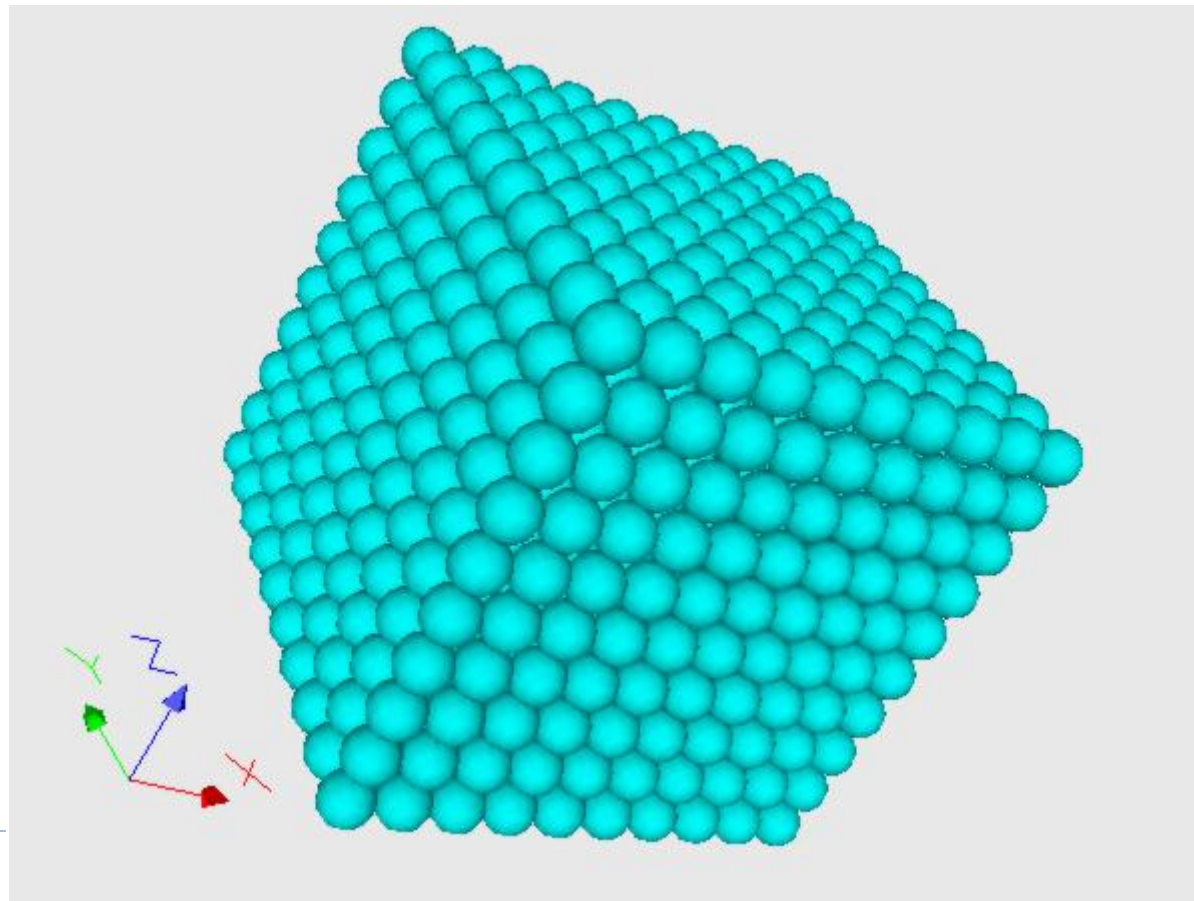
- ▶ 粒子間熱伝導問題の説明(座学)
- ▶ LAPACKの利用法と演習
(C言語、Fortran言語)
- ▶ ScaLAPACKの利用法と演習
(Fortran言語のみ)

粒子間熱伝導問題

- ▶ 東京大学情報基盤センター 中島研吾教授から提供頂いた、PPTとサンプルプログラムを利用
- ▶ 詳細資料
 - ▶ <http://nkl.cc.u-tokyo.ac.jp/09e/CE23/CE23.pdf>

問題設定 (1/5)

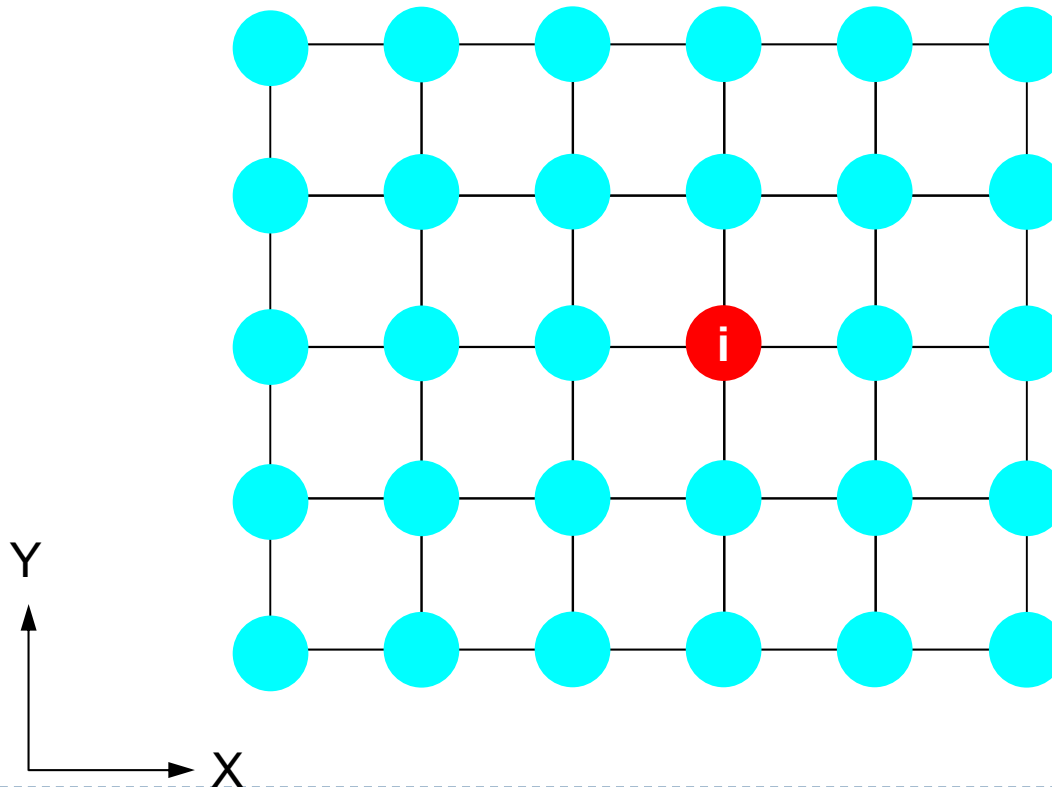
- ▶ 空間上に規則正しく, X, Y, Z 方向に NX, NY, NZ 個ずつ等間隔 (DX, DY, DZ) に配置された粒子の集合体を考える。



問題設定 (2/5)

粒子*i*に関する熱流束の / 釣り合い

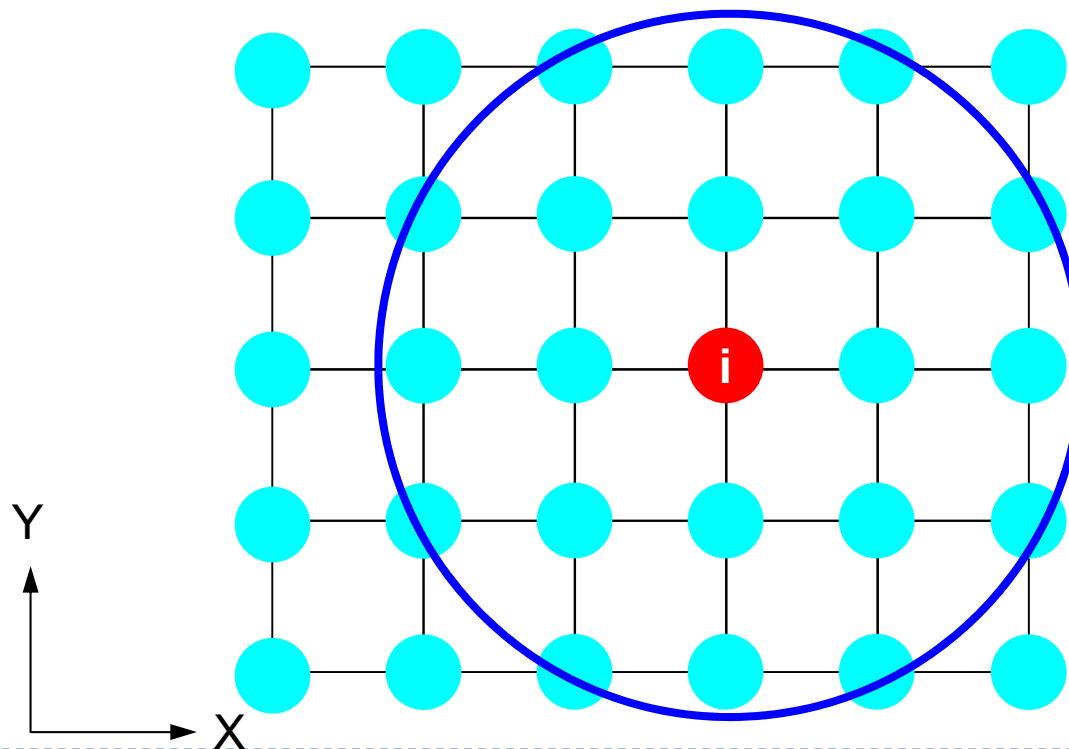
$$\sum_j^{DEL < RADI_{\max}} \frac{AREA}{DEL_{ij} / COND_{ij}} (T_j - T_i) + HCONV_i \cdot SURF \cdot (T_0 - T_i) + QVOL \cdot \bar{V} = 0$$



問題設定 (3/5)

粒子間熱伝導

$$\sum_i^{DEL < RADI_{max}} \frac{AREA}{DEL_{ij} / COND_{ij}} (T_j - T_i) + HCONV_i \cdot SURF \cdot (T_0 - T_i) + QVOL \cdot \bar{V} = 0$$

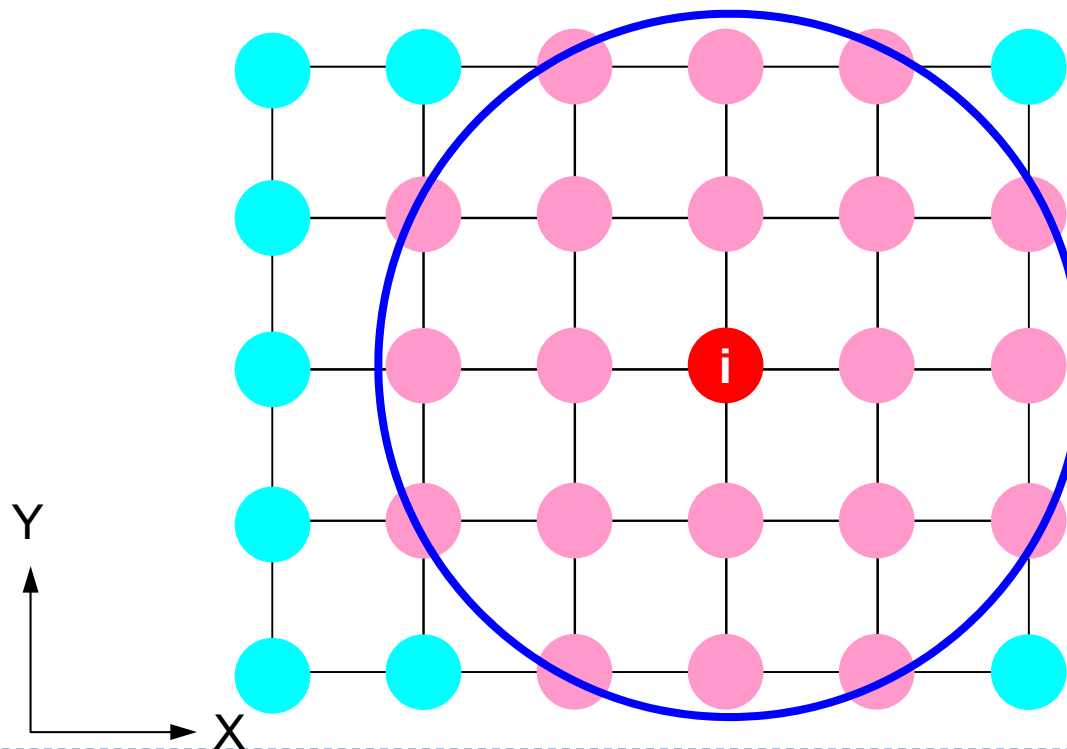


粒子*i*を中心とする,
半径 $RADI_{max}$ の球(円)

問題設定 (3/5)

粒子間熱伝導

$$\sum_i^{DEL < RADI_{max}} \frac{AREA}{DEL_{ij} / COND_{ij}} (T_j - T_i) + HCONV_i \cdot SURF \cdot (T_0 - T_i) + QVOL \cdot \bar{V} = 0$$



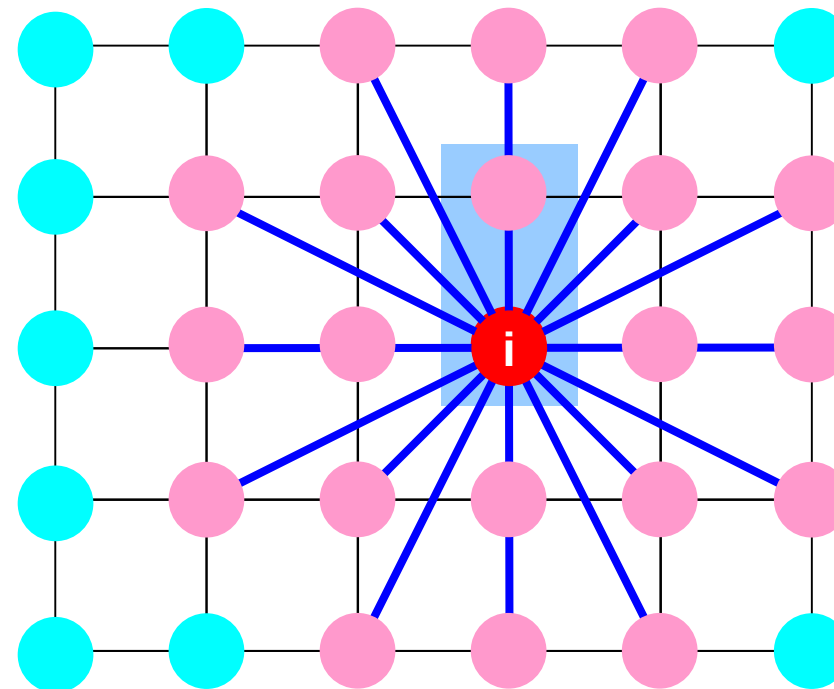
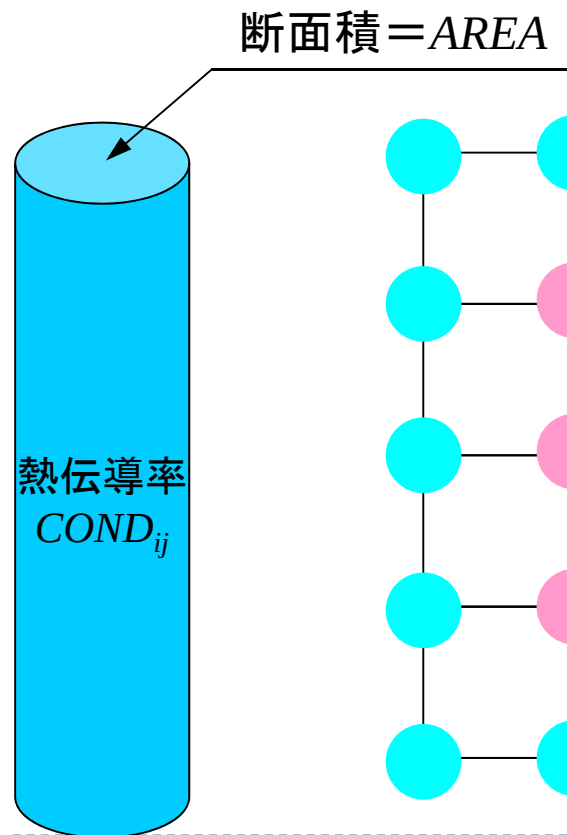
粒子*i*を中心とする、
半径 $RADI_{max}$ の球(円)

粒子間の距離 DEL_{ij} が
 $RADI_{max}$ のよりも小さい
粒子群とのみ熱伝導が
ある。

問題設定 (3/5)

粒子間熱伝導

$$\sum_i^{DEL < RADI_{max}} \frac{AREA}{DEL_{ij} / COND_{ij}} (T_j - T_i) + HCONV_i \cdot SURF \cdot (T_0 - T_i) + QVOL \cdot \bar{V} = 0$$



粒子*i*を中心とする,
半径 $RADI_{max}$ の球(円)

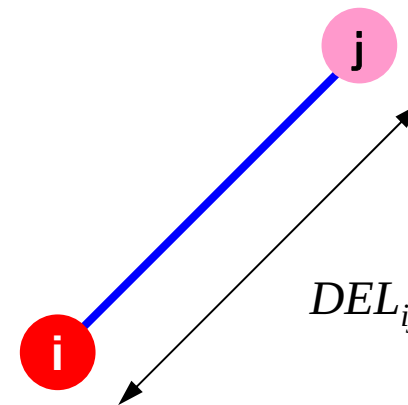
粒子間の距離 DEL_{ij} が
 $RADI_{max}$ のよりも小さい
粒子群とのみ熱伝導が
ある。

長さ DEL , 断面積 $AREA$,
熱伝導率 $COND_{ij}$ の管で
連結

熱伝導率：COND_{ij}

$$\sum_i^{DEL < RAD_{\max}} \frac{AREA}{DEL_{ij} / COND_{ij}} (T_j - T_i) + HCONV_i \cdot SURF \cdot (T_0 - T_i) + QVOL \cdot \bar{V} = 0$$

$$COND_{ij} = \frac{COND0}{\min(10^{DEL}, 10^{20})}$$

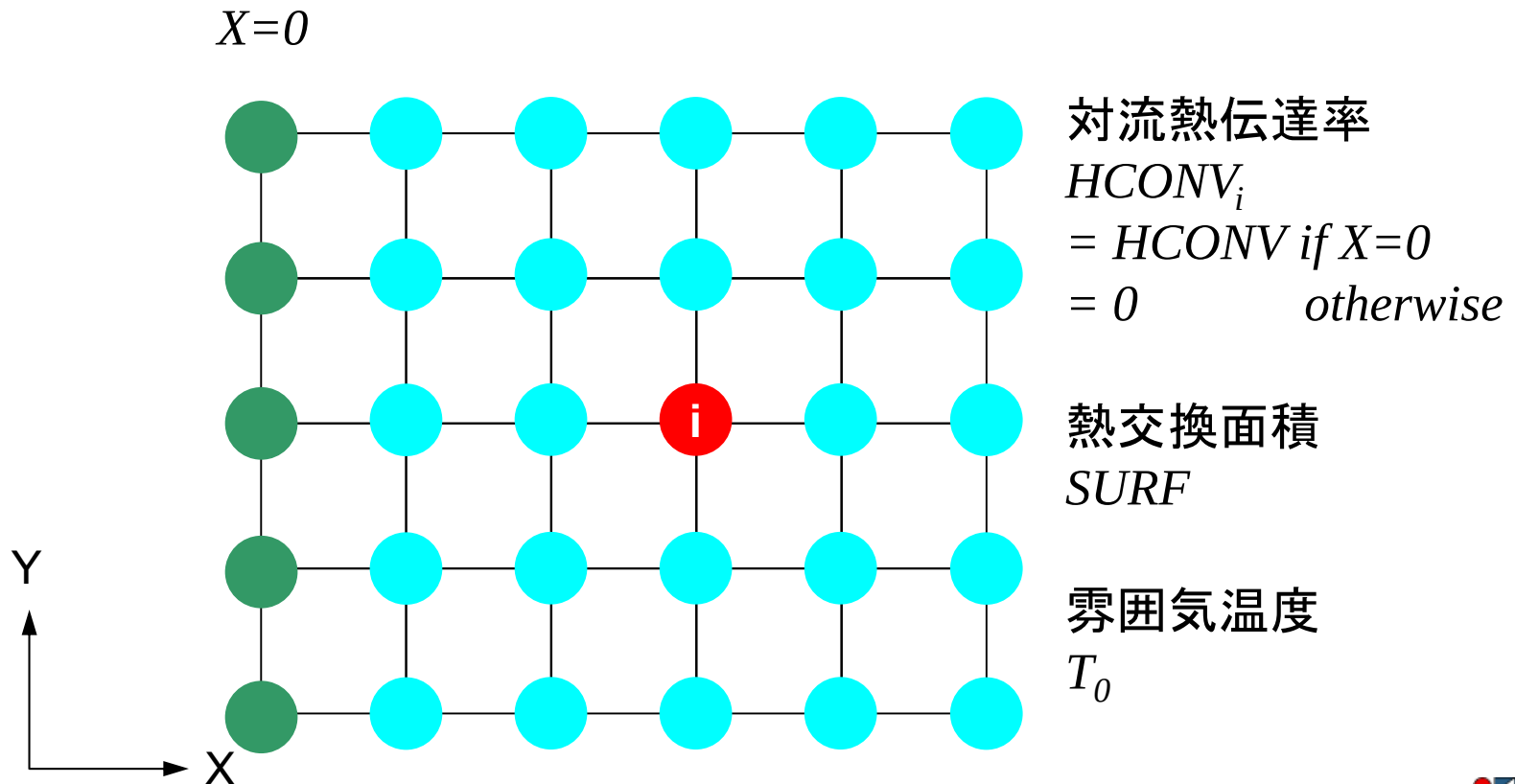


COND0 「基準」熱伝導率

問題設定 (4/5)

対流熱伝達

$$\sum_i^{DEL < RAD I_{\max}} \frac{AREA}{DEL_{ij} / COND_{ij}} (T_j - T_i) + \boxed{HCONV_i \cdot SURF \cdot (T_0 - T_i)} + QVOL \cdot \bar{V} = 0$$



問題設定 (5/5)

体積発熱

$$\sum_i^{DEL < RAD_{\max}} \frac{AREA}{DEL_{ij} / COND_{ij}} (T_j - T_i) + HCONV_i \cdot SURF \cdot (T_0 - T_i) + QVOL \cdot \bar{V} = 0$$

体積発熱量 $QVOL$

発熱体積 $\bar{V}$

$$\bar{V} = C_1 \cdot VOL + C_2 \cdot VOL \cdot DELQ$$

$$DELQ = \sqrt{X_i^2 + Y_i^2 + Z_i^2}$$

C_1, C_2 定数

VOL 各粒子の「基準」体積

$DELQ$ 粒子中心と原点との距離

粒子*i*に関するつりあい

$$\sum_j^{DEL < RAD_{i\max}} \frac{AREA}{DEL_{ij} / COND_{ij}} (T_j - T_i) + HCONV_i \cdot SURF \cdot (T_0 - T_i) + QVOL \cdot \bar{V} = 0$$

$$\sum_j^{DEL < RAD_{i\max}} \left[\frac{AREA}{DEL_{ij} / COND_{ij}} T_j \right] - \left[\sum_j^{DEL < RAD_{i\max}} \frac{AREA}{DEL_{ij} / COND_{ij}} \right] T_i + HCONV_i \cdot SURF \cdot T_0 - HCONV_i \cdot SURF \cdot T_i + QVOL \cdot \bar{V} = 0$$

$$\left[- \sum_j^{DEL < RAD_{i\max}} \frac{AREA}{DEL_{ij} / COND_{ij}} - HCONV_i \cdot SURF \right] T_i + \sum_j^{DEL < RAD_{i\max}} \left[\frac{AREA}{DEL_{ij} / COND_{ij}} T_j \right]$$

AMAT(i,i) (対角成分)

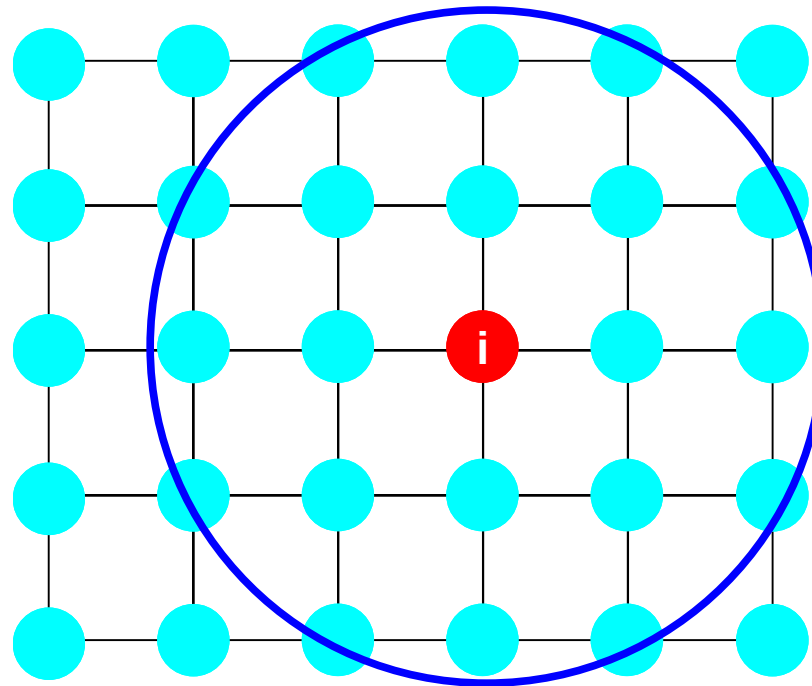
AMAT(i,j) (非対角成分)

$$= -HCONV_i \cdot SURF \cdot T_0 - QVOL \cdot \bar{V}$$

RHS (右辺)

係数行列 AMAT

- ▶ 影響範囲 $RADI_{max}$ が小さければ, 係数行列は疎
 - ▶ 局所的な効果
- ▶ 影響範囲が大きければ, 係数行列は密
 - ▶ 非ゼロ成分の割合が大きくなる



疎行列と密行列： $\{q\}=[A]\{p\}$

疎行列：差分法，有限要素法，有限体積法

```
do i= 1, N
  q(i)= D(i)*p(i)
  do k= index(i-1)+1, index(i)
    q(i)= q(i) + AMATs(k)*p(item(k))
  enddo
enddo
```

密行列：スペクトル法，分子動力学，境界要素法

```
do i= 1, N
  q(i)= 0.d0
  do j= 1, N
    q(i)= q(i) + AMATd(i,j)*p(j)
  enddo
enddo
```

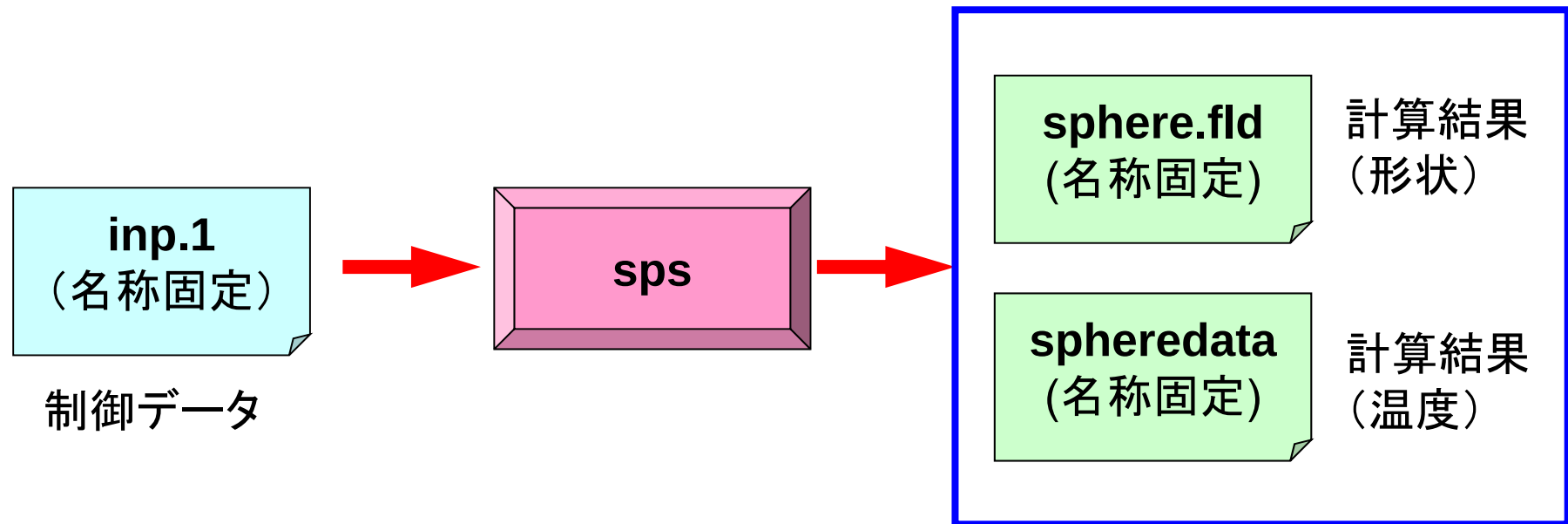
係数行列 AMAT

- ▶ 影響範囲 $RADI_{max}$ が小さければ、係数行列は疎
 - ▶ 局所的な効果
- ▶ 影響範囲が大きければ、係数行列は密
 - ▶ 非ゼロ成分の割合が大きくなる
- ▶ 影響範囲が大きくなる場合のことを考えて、プログラムの中では係数行列を「密 (AMATd(i,j))」として扱う。

▶ 問題設定

▶ 1CPU用プログラムの解説

sps: ファイル, データ



制御データ：inp1

10 10 10	NX, NY, NZ
1.e0 1.e0 1.e0	DX, DY, DZ
1.e0 10.e0 1.e0 1.e0	VOL, AREA, QVOL, CONDO
1.e24 0.e0 10.e0	HCONV, T0, SURF
8.0e0	RADImax
1.0e0 0.1e0	C1, C2

計算結果（形状）：sphere.fld

*.fldの形式であれば名称は任意

```
# AVS field file
ndim=      3
dim1=     10
dim2=     10
dim3=     10
nspace=      3
vec len=      1
data= float
field= uniform
label= temperature
variable 1 file=./spheredata filetype=ascii 温度ファイル名
```

必須
三次元モデルであることを示す
NX
NY
NZ

計算結果（温度）：spheredata

計算結果（形状）からの参照が正しければ名称は任意

```
1.188402E-24  
5.388751E+00  
9.485117E+00  
1.311638E+01  
1.630137E+01
```

...

```
open (22, file='spheredata', status='unknown')  
do i= 1, NX*XY*NZ  
  write (22, '(1pe16.6)') PHI(i)  
enddo
```

シリアル版の計算手順

- ▶ 制御データ入力
- ▶ 粒子生成
- ▶ 係数マトリクス計算
 - ▶ 熱伝導
 - ▶ 対流熱伝達
 - ▶ 発熱
- ▶ CG法による連立一次方程式求解
(オリジナル版 [test_org.f](#), [test_org.c](#))
 - ▶ 密行列
 - ▶ 点ヤコビ前処理
- ▶ 出力
 - ▶ MicroAVS用

test.f：シリアル版 (1/10)

初期設定

```
implicit REAL*8 (A-H,O-Z)

real(kind=8) :: VOL, AREA, QVOL, COND0, COND, RADImax
real(kind=8) :: HCONV, T0, SURF, DEL, DEL0, coef1, coef2
real(kind=8), dimension(:) , allocatable :: XC, YC, ZC
real(kind=8), dimension(:, :) , allocatable :: AMAT
real(kind=8), dimension(:) , allocatable :: RHS
real(kind=8), dimension(:) , allocatable :: PHI
real(kind=8), dimension(:, :) , allocatable :: W

integer :: NX, NY, NZ, N
integer :: R, Z, P, Q, DD

!C
!C +-----+
!C | INIT |
!C +-----+
!C===
  open (11, file= 'inp1', status='unknown')
    read (11,*) NX, NY, NZ
    read (11,*) DX, DY, DZ
    read (11,*) VOL, AREA, QVOL, COND0
    read (11,*) HCONV, T0, SURF
    read (11,*) RADImax
    read (11,*) coef1, coef2
  close (11)

N= NX*NY*NZ
```

test.f：シリアル版 (2/10)

粒子中心の座標 (XC,YC,ZC)

```
allocate (XC(N), YC(N), ZC(N))
```

```
icou= 0  
do k= 1, NZ  
  do j= 1, NY  
    do i= 1, NX  
      icou= icou + 1  
      XC(icou)= dfloat(i-1)*DX  
      YC(icou)= dfloat(j-1)*DY  
      ZC(icou)= dfloat(k-1)*DZ  
    enddo  
  enddo  
enddo
```

```
!C==
```

粒子の通し番号 ii (1~N)

$ii = (k-1) * NX * NY + (j-1) * NX + i$

test.f：シリアル版 (3/10)

マトリクス生成：熱伝導部分

```
!C
!C +-----+
!C | MATRIX |
!C +-----+
!C ===
      allocate (AMAT(N,N), RHS(N))

      AMAT= 0.d0
      RHS = 0.d0
      do i= 1, N
        do j= 1, N
          if (j.ne.i) then
            DEL= dsqrt((XC(i)-XC(j))**2 + (YC(i)-YC(j))**2
&
            + (ZC(i)-ZC(j))**2)
            if (DEL.le.RADImax) then
              COND= COND0/(10.d0**dmin1(DEL,20.d0))
              coef= COND*AREA / DEL
              AMAT(i,j)= coef
              AMAT(i,i)= AMAT(i,i) - coef
            endif
          endif
        enddo
      enddo
```

$$\left[- \sum_j^{DEL < RADI_{max}} \frac{AREA}{DEL_{ij} / COND_{ij}} - HCONV_i \cdot SURF \right] T_i + \sum_j^{DEL < RADI_{max}} \left[\frac{AREA}{DEL_{ij} / COND_{ij}} T_j \right] = -HCONV_i \cdot SURF \cdot T_0 - QVOL \cdot \bar{V}$$

$$COND_{ij} = \frac{COND0}{\min(10^{DEL}, 10^{20})}$$

test.f：シリアル版（3/10）

マトリクス生成：熱伝導部分

```
!C
!C +-----+
!C | MATRIX |
!C +-----+
!C ===
      allocate (AMAT(N,N), RHS(N))

      AMAT= 0.d0
      RHS = 0.d0
      do i= 1, N
        do j= 1, N
          if (j.ne.i) then
            DEL= dsqrt((XC(i)-XC(j))**2 + (YC(i)-YC(j))**2
&
            + (ZC(i)-ZC(j))**2)
            if (DEL.le.RADImax) then
              COND= COND0/(10.d0**dmin1(DEL,20.d0))
              coef= COND*AREA / DEL
              AMAT(i,j)= coef
              AMAT(i,i)= AMAT(i,i) - coef 対角項
            endif
          endif
        enddo
      enddo
```

$$\left[- \sum_j^{DEL < RADI_{max}} \frac{AREA}{DEL_{ij} / COND_{ij}} - HCONV_i \cdot SURF \right] T_i + \sum_j^{DEL < RADI_{max}} \left[\frac{AREA}{DEL_{ij} / COND_{ij}} T_j \right] = -HCONV_i \cdot SURF \cdot T_0 - QVOL \cdot \bar{V}$$

test.f：シリアル版 (4/10)

マトリクス生成：体積発熱

```

do i= 1, N
  DELQ= dsqrt(XC(i)**2 + YC(i)**2 + ZC(i)**2)
  RHS(i)= -QVOL*(coef1*VOL + coef2*DELQ*VOL)
enddo

i= 1
do k= 1, NZ
do j= 1, NY
  ic= (k-1)*NX*NY + (j-1)*NX + i
  AMAT(ic,ic)= -HCONV*SURF + AMAT(ic,ic)
  RHS (ic )= -HCONV*SURF*T0 + RHS (ic)
enddo
enddo
!C==
  
```

$$\begin{aligned}
 & \left[- \sum_j^{DEL < RAD_{\max}} \frac{AREA}{DEL_{ij} / COND_{ij}} - HCONV_i \cdot SURF \right] T_i + \\
 & \sum_j^{DEL < RAD_{\max}} \left[\frac{AREA}{DEL_{ij} / COND_{ij}} T_j \right] \\
 & = -HCONV_i \cdot SURF \cdot T_0 - QVOL \cdot \bar{V}
 \end{aligned}$$

$$\begin{aligned}
 \bar{V} &= C_1 \cdot VOL + C_2 \cdot VOL \cdot DELQ \\
 DELQ &= \sqrt{X_i^2 + Y_i^2 + Z_i^2}
 \end{aligned}$$

test.f：シリアル版（4/10）

マトリクス生成：対流熱伝達

```

do i= 1, N
  DELQ= dsqrt(XC(i)**2 + YC(i)**2 + ZC(i)**2)
  RHS(i)= -QVOL*(coef1*VOL + coef2*DELQ*VOL)
enddo

i= 1
do k= 1, NZ
do j= 1, NY
  ic= (k-1)*NX*NY + (j-1)*NX + i
  AMAT(ic,ic)= -HCONV*SURF + AMAT(ic,ic)
  RHS (ic )= -HCONV*SURF*T0 + RHS (ic)
enddo
enddo

```

!C==

$$\begin{aligned}
 & \left[- \sum_j^{DEL < RAD_{\max}} \frac{AREA}{DEL_{ij} / COND_{ij}} - \underline{HCONV_i \cdot SURF} \right] T_i - \\
 & \sum_j^{DEL < RAD_{\max}} \left[\frac{AREA}{DEL_{ij} / COND_{ij}} T_j \right] \\
 & = \underline{-HCONV_i \cdot SURF \cdot T_0} - QVOL \cdot \bar{V}
 \end{aligned}$$

前処理付き共役勾配法

Preconditioned Conjugate Gradient Method (CG)

```
Compute  $\mathbf{r}^{(0)} = \mathbf{b} - [\mathbf{A}]\mathbf{x}^{(0)}$ 
for i= 1, 2, ...
  solve  $[\mathbf{M}]\mathbf{z}^{(i-1)} = \mathbf{r}^{(i-1)}$ 
   $\rho_{i-1} = \mathbf{r}^{(i-1)} \mathbf{z}^{(i-1)}$ 
  if i=1
     $\mathbf{p}^{(1)} = \mathbf{z}^{(0)}$ 
  else
     $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$ 
     $\mathbf{p}^{(i)} = \mathbf{z}^{(i-1)} + \beta_{i-1} \mathbf{p}^{(i-1)}$ 
  endif
   $\mathbf{q}^{(i)} = [\mathbf{A}]\mathbf{p}^{(i)}$ 
   $\alpha_i = \rho_{i-1} / \mathbf{p}^{(i)} \mathbf{q}^{(i)}$ 
   $\mathbf{x}^{(i)} = \mathbf{x}^{(i-1)} + \alpha_i \mathbf{p}^{(i)}$ 
   $\mathbf{r}^{(i)} = \mathbf{r}^{(i-1)} - \alpha_i \mathbf{q}^{(i)}$ 
  check convergence  $|\mathbf{r}|$ 
end
```

前処理: 対角スケーリング

$\mathbf{x}^{(i)}$: ベクトル

α_i : スカラー

test_org.f : シリアル版 (5/10)

CG法①

```
!C
!C +-----+
!C | CG iterations |
!C +-----+
!C===
      EPS= 1.d-08
      allocate (W(N,4), PHI(N))
      W = 0.d0
      PHI= 0.d0

      R = 1
      Z = 2
      Q = 2
      P = 3
      DD= 4
      do i= 1, N
        W(i,DD)= 1.0D0 / AMAT(i,i)
      enddo
```

$W(i,1) = W(i,R) \Rightarrow \{r\}$
 $W(i,2) = W(i,Z) \Rightarrow \{z\}$
 $W(i,2) = W(i,Q) \Rightarrow \{q\}$
 $W(i,3) = W(i,P) \Rightarrow \{p\}$
 $W(i,4) = W(i,DD) \Rightarrow 1/DIAG$

Compute $r^{(0)} = b - [A]x^{(0)}$
for $i = 1, 2, \dots$
 solve $[M]z^{(i-1)} = r^{(i-1)}$
 $\rho_{i-1} = r^{(i-1)} z^{(i-1)}$
 if $i=1$
 $p^{(1)} = z^{(0)}$
 else
 $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$
 $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$
 endif
 $q^{(i)} = [A]p^{(i)}$
 $\alpha_i = \rho_{i-1} / p^{(i)} q^{(i)}$
 $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$
 $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$
 check convergence $|r|$
end

test_org.f: シリアル版 (6/10)

CG法②

```
!C
!C-- {r0}= {b} - [A]{xini} |

      do i= 1, N
        W(i,R) = RHS(i)
        do j= 1, N
          W(i,R) = W(i,R) - AMAT(i,j)*PHI(j)
        enddo
      enddo

      BNRM2= 0.0D0
      do i= 1, N
        BNRM2 = BNRM2 + RHS(i)**2
      enddo

!C*****
      do iter= 1, N
!C
!C-- {z}= [Minv]{r}

        do i= 1, N
          W(i,Z)= W(i,DD) * W(i,R)
        enddo
```

Compute $r^{(0)} = b - [A]x^{(0)}$

for $i = 1, 2, \dots$

solve $[M]z^{(i-1)} = r^{(i-1)}$

$\rho_{i-1} = r^{(i-1)} z^{(i-1)}$

if $i=1$

$p^{(1)} = z^{(0)}$

else

$\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$

$p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$

endif

$q^{(i)} = [A]p^{(i)}$

$\alpha_i = \rho_{i-1} / p^{(i)} q^{(i)}$

$x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$

$r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$

check convergence $|r|$

end

test_org.f: シリアル版 (7/10)

CG法③

```
!C
!C-- RHO= {r}{z}

      RHO= 0.d0
      do i= 1, N
        RHO= RHO + W(i,R)*W(i,Z)
      enddo

!C
!C-- {p} = {z} if      ITER=1
!C   BETA= RHO / RHO1  otherwise

      if ( iter.eq.1 ) then
        do i= 1, N
          W(i,P)= W(i,Z)
        enddo
      else
        BETA= RHO / RHO1
        do i= 1, N
          W(i,P)= W(i,Z) + BETA*W(i,P)
        enddo
      endif
endif
```

Compute $r^{(0)} = b - [A]x^{(0)}$

for $i = 1, 2, \dots$

 solve $[M]z^{(i-1)} = r^{(i-1)}$

$\rho_{i-1} = r^{(i-1)} z^{(i-1)}$

if $i=1$

$p^{(1)} = z^{(0)}$

else

$\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$

$p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$

endif

$q^{(i)} = [A]p^{(i)}$

$\alpha_i = \rho_{i-1} / p^{(i)} q^{(i)}$

$x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$

$r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$

 check convergence $|r|$

end

test_org.f: シリアル版 (8/10)

CG法④

```
!C
!C-- {q}= [A]{p}

      do i= 1, N
        W(i,Q) = 0.d0
        do j= 1, N
          W(i,Q) = W(i,Q) + AMAT(i,j)*W(j,P)
        enddo
      enddo

!C
!C-- ALPHA= RHO / {p}{q}

      C1= 0.d0
      do i= 1, N
        C1= C1 + W(i,P)*W(i,Q)
      enddo
      ALPHA= RHO / C1

!C
!C-- {x}= {x} + ALPHA*{p}
!C   {r}= {r} - ALPHA*{q}

      do i= 1, N
        PHI(i) = PHI(i) + ALPHA * W(i,P)
        W(i,R)= W(i,R) - ALPHA * W(i,Q)
      enddo
```

Compute $r^{(0)} = b - [A]x^{(0)}$

for $i = 1, 2, \dots$

 solve $[M]z^{(i-1)} = r^{(i-1)}$

$\rho_{i-1} = r^{(i-1)} z^{(i-1)}$

if $i=1$

$p^{(1)} = z^{(0)}$

else

$\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$

$p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$

endif

$q^{(i)} = [A]p^{(i)}$

$\alpha_i = \rho_{i-1} / p^{(i)} q^{(i)}$

$x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$

$r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$

 check convergence $|r|$

end

test_org.f: シリアル版 (9/10)

CG法⑤

```
DNRM2 = 0.0
do i= 1, N
  DNRM2= DNRM2 + W(i,R)**2
enddo

RESID= dsqrt(DNRM2/BNRM2)

write (*, 1000) iter, RESID
1000  format (i5, 1p16.6)

if ( RESID.le.EPS) goto 900
RHO1 = RHO

enddo
!C*****

IER = 1

900 continue
!C===
```

Compute $r^{(0)} = b - [A]x^{(0)}$

for $i = 1, 2, \dots$

 solve $[M]z^{(i-1)} = r^{(i-1)}$

$\rho_{i-1} = r^{(i-1)} z^{(i-1)}$

if $i=1$

$p^{(1)} = z^{(0)}$

else

$\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$

$p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$

endif

$q^{(i)} = [A]p^{(i)}$

$\alpha_i = \rho_{i-1} / p^{(i)} q^{(i)}$

$x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$

$r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$

check convergence |r|

end

test_org.f：シリアル版（10/10）

結果出力

```
!C
!C +-----+
!C | OUTPUT |
!C +-----+
!C===
      N1= 1
      N3= 3
!C
!C-- MESH
      open (22, file='sphere.fld', status='unknown')
      write (22,'(a)')      '# AVS field file'
      write (22,'(a,i5)') 'ndim=', N3
      write (22,'(a,i5)') 'dim1=', NX
      write (22,'(a,i5)') 'dim2=', NY
      write (22,'(a,i5)') 'dim3=', NZ
      write (22,'(a,i5)') 'nspace=', N3
      write (22,'(a,i5)') 'veclen=', N1
      write (22,'(a,i5)') 'data= float'
      write (22,'(a,i5)') 'field= uniform'
      write (22,'(a,i5)') 'label= temperature'
      write (22,'(a,i5)') 'variable 1 file=./spheredata filetype=ascii'
      close (22)
!C
!C-- RESULTS
      open (22, file='spheredata', status='unknown')
      do i= 1, N
        write (22,'(1pe16.6)') PHI(i)
      enddo
!C===
```

サンプルプログラムの実行 (LAPACK dgesv)

LAPACK dgesvサンプルプログラムの注意点

- ▶ C言語版、Fortran言語版のファイル名（共通）

lecLAPACK-fx.tar

- ▶ ジョブスクリプトファイルgo.sh 中のキュー名を

lecture から tutorial に変更してから
qsub してください。

- ▶ lecture : 実習時間外のキュー（同時実行数1）
- ▶ tutorial : 実習時間内のキュー（同時実行数4+）

LAPACK dgesvのサンプルプログラムの実行 (C言語版/ Fortran言語版共通)

- ▶ 以下のコマンドを実行する

```
$ cp /home/z30082/lecLAPACK-fx.tar ./
```

```
$ tar xvf lecLAPACK-fx.tar
```

```
$ cd sphere-LAPACK
```

```
$ cd C //C言語の人
```

- ▶ \$ cd F //Fortran言語の人

```
$ make
```

```
$ pjsub go.bash
```

- ▶ 実行が終了したら、以下を実行する

```
$ cat go.bash.XXXXXXX (XXXXXXは数字)
```

LAPACK dgesvのサンプルプログラムの実行 (C言語版)

▶ 以下のような結果が見えれば成功

time = 0.000013 [sec.]

53612532.46 [MFLOPS]

990 -2.272792e+00

991 -2.276715e+00

992 -2.288410e+00

993 -2.307670e+00

994 -2.334166e+00

995 -2.367479e+00

996 -2.407125e+00

997 -2.452584e+00

998 -2.503330e+00

999 -2.558846e+00

err = 6.274235e+02

連立一次方程式求解部分が
実装されて無いので、
時間が極端に短く、
誤差ERRが大きく
なっていますが、
正常な動作です。

LAPACK dgesvのサンプルプログラムの実行 (Fortran言語版)

▶ 以下のような結果が見えれば成功

TIME[sec] = 1.529976725578308E-05

MFLOPS = 43802496.39070075

991 -2.272792E+00

992 -2.276715E+00

993 -2.288410E+00

994 -2.307670E+00

995 -2.334166E+00

996 -2.367479E+00

997 -2.407125E+00

998 -2.452584E+00

999 -2.503330E+00

1000 -2.558846E+00

EPS = 627.4235113914225

連立一次方程式求解部分が
実装されて無いので、
時間が極端に短く、
誤差ERRが大きく
なっていますが、
正常な動作です。

サンプルプログラムの説明 (LAPACK)

- ▶ 1コア(逐次)実行版です
- ▶ スレッド並列化版は、スレッド並列化版をリンクしてコンパイルの上、並列実行数の指定をする必要があります。
- ▶ LAPACKはスレッド並列化のみ対応しているため、複数ノードを通信しつつ利用することはできません
 - ▶ 通信しつつ利用するには、分散版であるScaLAPACKを利用します
 - ▶ ノード内のみLAPACKを使い、ノード間はMPIで並列化するような使い方は可能です
 - ▶ たとえば、領域分割法を利用し、ノード内のみLAPACKで連立一次方程式を解く場合

サンプルプログラムの説明 (LAPACK)

- ▶ 誤差ERRの計算は、 $10 \times 10 \times 10$ の問題に特化されています。
それ以外の問題サイズでは機能しません。
- ▶ test_org.c、test_org.fは、オリジナルのCG法を用いた反復解法のコードです。

富士通LAPACK呼び出しオプション

- ▶ 富士通コンパイラから、LAPACK(SSL II (Scientific Subroutine Library II) 数学ライブラリ) を呼び出す場合、以下のオプションを付けます。

●C/Fortran言語共通(LAPACKが逐次(1コア実行))

mpifrtpx <プログラム名> **-SSL2** :Fortran言語

mpiccpix <プログラム名> **-SSL2** :C言語

●C/Fortran言語共通(LAPACKがスレッド実行)

mpifrtpx <プログラム名> **-SSL2BLAMP** :Fortran言語

mpiccpix <プログラム名> **-SSL2BLAMP** :C言語

演習課題1 (LAPACK)

1. `test.c` もしくは `test.f` のメイン関数(手続き)の連立一次方程式求解部分を、LAPACK `dgesv` ルーチンの呼び出しにより、高速化してください。
 - ▶ 引数の並びに注意してください。
 - ▶ C言語は、`dgesv` のFortran手続きの呼び出しになります。以下に注意してください。
 - ▶ 関数名: `dgesv_`
 - ▶ 全ての引数がポインタ引き渡し
 - ▶ デバッグとして、誤差(ERR)が十分小さいことを確認してください。

演習課題2 (LAPACK)

2. CG法による解法ルーチン[test_org.c](#)もしくは[test_org.f](#) をコンパイルして、LAPACK版と実行速度を比べてください。
3. 問題サイズを大きくして、LAPACK版、CG法版の実行速度を比べてください。
 - ▶ NX, NY, NZの値、および、計算打ち切り半径RADImaxを適切に設定してください。
 - ▶ RADImaxの値が小さいと、CG法が有利（高速）になる可能性があります。

演習課題 3 (LAPACK)

4. 富士通コンパイラからスレッド並列版のLAPACKを呼び出すように変更してください。
 - ▶ 16並列までのスレッド版LAPACKを利用し、性能評価を行ってください。

LAPACK dgesv回答

LAPCK dgesvの回答 (C言語版)

```
dgesv_(&nn, &inc, amat2, &nn,  
      piv, rhs, &nn, &info);
```

LAPCK dgesvの回答 (Fortran言語版)

```
call DGESV(N, INC, AMAT, N,  
&          PIV, RHS, N, INFO)
```

サンプルプログラムの実行 (ScaLAPACK pdgesv)

ScaLAPACK pdgesvサンプルプログラムの 注意点

- ▶ Fortran言語版のファイル名

lecScaLAPACK-fx.tar

※ScaLAPACKはC言語版のサンプル
はありません

- ▶ ジョブスクリプトファイルgo.sh 中のキュー名を
lecture から tutorial に変更してから
qsub してください。

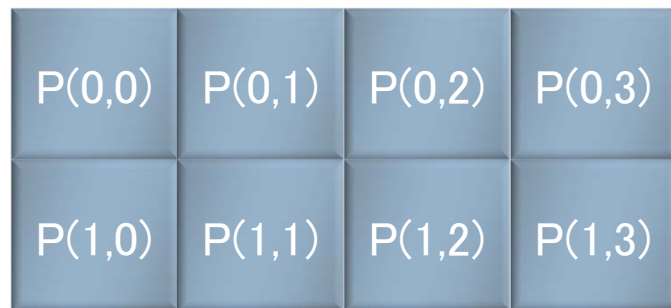
- ▶ lecture : 実習時間外のキュー(同時実行数1)
- ▶ tutorial : 実習時間内のキュー(同時実行数4+)

ScaLAPACKにおけるプロセッサ・グリッド

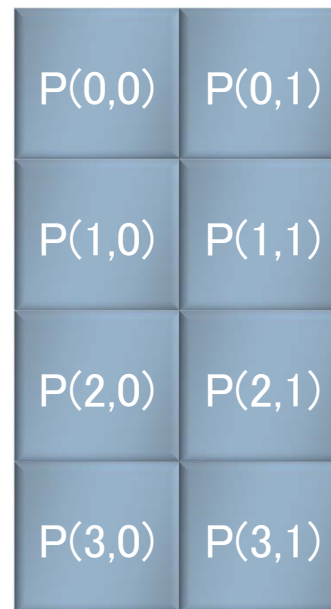
- ▶ MPIで起動する総プロセス数は、プロセッサ・グリッドと呼ばれる2次元プロセッサ構造で管理されます。

- ▶ 例: 8プロセス

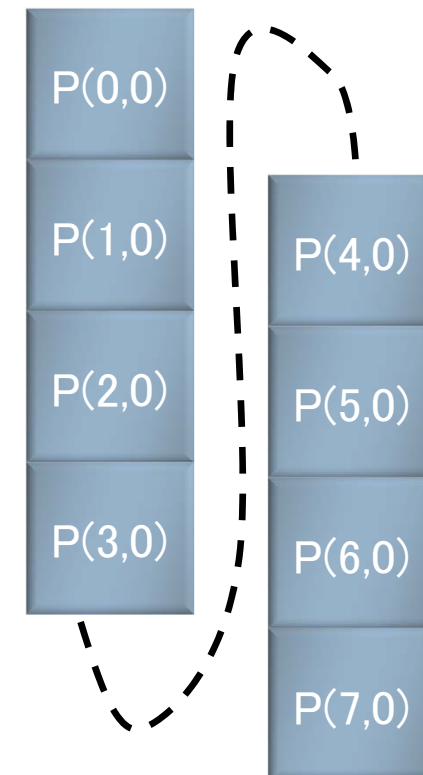
● 2×4 構成



● 4×2 構成



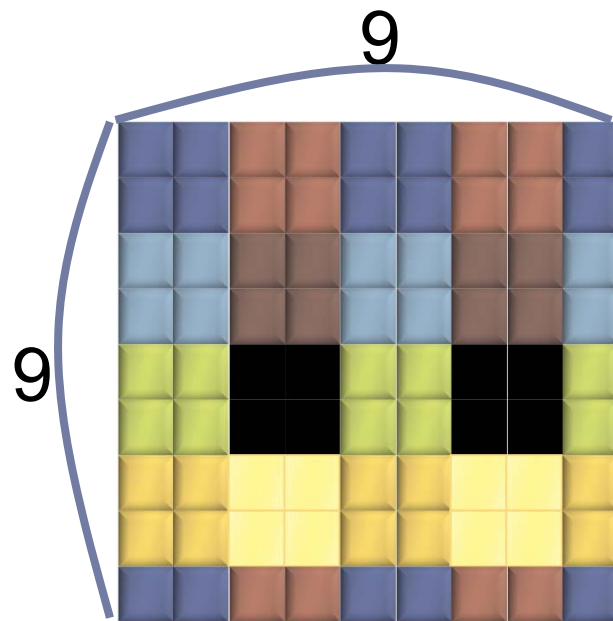
● 8×1 構成



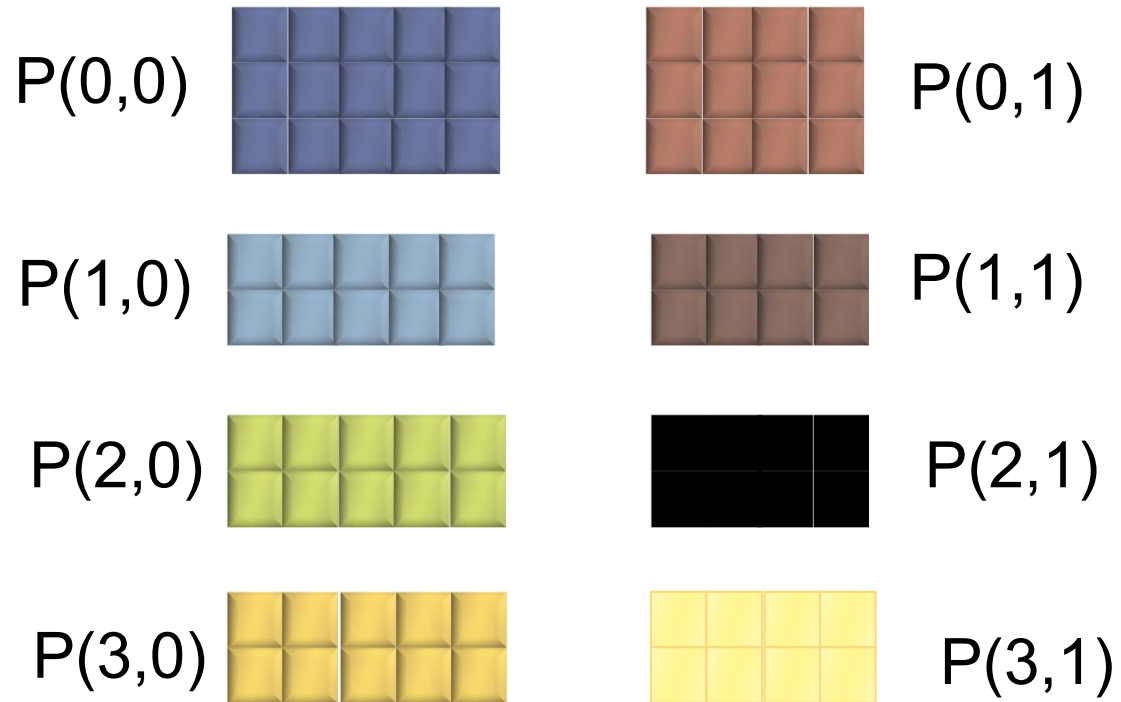
ScaLAPACKにおけるデータ分散

- ▶ ブロック幅MBの2次元ブロックサイクリック分散となります
- ▶ 例: 8プロセス(2×4プロセッサ・グリッド)、MB=2

※行列サイズで割り切れないプロセス数の場合も考慮して、
sub(A)の確保領域を計算する必要があります



大域ビュー(行列A)



局所ビュー(sub(A))

コンテキストの定義

●コンテキスト:プロセッサ・グリッドに関する情報体

BLACS_GET(ICONTXT, WHAT, VAL)

●BLACSの内部情報を取得する

●ICONTXT: 入力、整数型、

WHATで指定するコンテキストに設定を試みる値(引数)。
(引数が)無い時は無視される。

●WHAT: 入力、整数型、コンテキストに設定する内容

WHAT = 0 : デフォルトのシステムコンテキスト

WHAT = 1 : BLACS メッセージの ID 幅

WHAT = 2 : コンパイルされるBLACS デバックレベル

WHAT = 10: ICONTXTにより制御されるBLACSコンテキストを定義するために
用いられるシステムコンテキストの参照

WHAT = 11: 現在使っているマルチリング構造のリング数

WHAT = 12: 現在使っている一般化した木構造の枝数

●VAL: 出力、整数型、コンテキスト情報

BLACSグリッドの定義

BLACS_GRIDINIT(ICONTXT, ORDER, NPROW, NPCOL)

● **ICONTXT** : 入力／出力、整数型
コンテキスト

● **ORDER** : 入力、文字型 * 1

プロセスをどのようにBLACSのグリッドに割り当てるか指定

● 'R' : 行方向の自然なオーダリングを使う。

● 'C' : 列方向の自然なオーダリングを使う。

● そうでないなら: 行方向の自然なオーダリングを使う。

BLACSグリッドの情報入手

BLACS_GRIDINFO

(ICONTXT, NPROW, NPCOL, MYPROW, MYPCOL)

- ICONTXT: 入力、整数型
コンテキスト
- NPROW: 出力、整数型
プロセッサ・グリッドの行数
- NPCOL: 出力、整数型
プロセッサ・グリッドの列数
- MYPROW: 出力、整数型
プロセッサ・グリッドにおける自分の行方向の認識番号
- MYPCOL: 出力、整数型
プロセッサ・グリッドにおける自分の列方向の認識番号

データ分散を容易にする関数

PDELSET(A, IA, JA, DESCA, ALPHA)

●大域配列の添え字から、局所配列の適する場所にデータを収納する

●A: 局所出力、倍精度型

収納すべき局所配列

●IA: 大域入力、整数型

大域的配列における1次元目の添え字

●JA: 大域入力、整数型

大域的配列における2次元目の添え字

●DESCA: 大域かつ局所入力、整数型配列

局所配列Aの記述子

●ALPHA: 局所入力、倍精度型

代入すべき値

データ分散を容易にする関数:例

大域配列AMATを、適切な局所行列Aに代入

```
DO J = 1, N
  DO I = 1, N
    CALL PDELSET( A, I, J, DESCA, AMAT(I,J))
  ENDDO
  CALL PDELSET( B, J, 1, DESCB, RHS(J))
ENDDO
```

大域ベクトルRHSを、適切な局所ベクトルBに代入

PDELSET関数による方法の注意点

- ▶ PDELSET関数を用いて行列を分散する方法は容易ですが、**サンプルプログラムのように、大域行列Aを全プロセスで所有していると、メモリに関するスケーラビリティが無くなります。**
 - ▶ つまり、実行できる行列サイズNが、1ノードのメモリ総量で決まる。実行ノード数を増加しても、Nを大きくできない。
- ▶ 本格的な並列プログラムを書くためには
 - ▶ **局所ビューの観点で、行列の値を直接sub(A)に値を代入する**
プログラムを書く必要があります。
- ▶ そのためには、ブロック・サイクリック分散方式の特徴を理解して、局所配列の確保をする必要があります
 - ▶ **本講習会のMPI基礎編で、関連技術の演習があります。**

ScaLAPACK pdgesvのサンプルプログラムの実行 (Fortran言語版のみ)

- ▶ 以下のコマンドを実行する

```
$ cp /home/z30082/lecScaLAPACK-fx.tar ./
```

```
$ tar xvf lecScaLAPACK-fx.tar
```

```
$ cd sphere-ScaLAPACK
```

```
$ cd F
```

```
$ make
```

```
$ pjsub go.bash
```

- ▶ 実行が終了したら、以下を実行する

```
$ cat go.bash.XXXXXXX (XXXXXXXは数字)
```


ScaLAPACK pdgesvの サンプルプログラムの実行 (Fortran言語版)

▶ 以下のような結果が見えれば成功

TIME[sec] = 3.399793058633804E-06

MFLOPS = 216189698.2744275

ScaLAPACK Example Program -- Sep, 2010

Solving $Ax=b$ where A is a 1000 by 1000 matrix
with a block size of 32

Running on 64 processes, where the process grid is 8 by 8

INFO code returned by PDGESV = 0

EPS = 627.4235113914225

連立一次方程式求解部分が実装されて無いので、時間が
極端に短く、誤差ERRが大きくなっていますが、正常な動作です。

サンプルプログラムの説明(ScaLAPACK)

- ▶ **ピュアMPI版です**
 - ▶ スレッド並列を利用したハイブリッドMPI版を利用するには、スレッド並列版をリンクしてコンパイルし、スレッド並列数を指定する必要があります
- ▶ 64プロセス実行(プロセッサー・グリッド 8×8)、問題サイズ1000、ブロック幅32に**特化されています**
 - ▶ プロセス数、プロセッサー・グリッド、問題サイズ、ブロック幅を変更する場合、PARAMETER文の定数を変更する必要があります
- ▶ 誤差ERRの計算は、 $10 \times 10 \times 10$ の問題に特化されています。それ以外の問題サイズでは機能しません。

富士通ScaLAPACK呼び出しオプション

- ▶ 富士通コンパイラから、ScaLAPACK(SSL II (Scientific Subroutine Library II) 数学ライブラリ) を呼び出す場合、以下のオプションを付けます。

●C/Fortran言語共通 (ScaLAPACKが逐次(1コア実行))

mpifrtpx <プログラム名> **-SCALAPACK -SSL2** :Fortran言語

mpiccpix <プログラム名> **-SCALAPACK -SSL2** :C言語

●C/Fortran言語共通 (ScaLAPACKがスレッド実行)

mpifrtpx <プログラム名> **-SCALAPACK -SSL2BLAMP**
:Fortran言語

mpiccpix <プログラム名> **-SCALAPACK -SSL2BLAMP**
:C言語

演習課題1 (ScaLAPACK)

1. `test.f` のメイン手続きの連立一次方程式求解部分を、ScaLAPACK `pdgesv` 手続きの呼び出しにより、高速化してください
 - ▶ 引数の並びに注意してください。
 - ▶ デバッグとして、誤差 (ERR) が十分小さいことを確認してください。

演習課題2 (ScaLAPACK)

2. 問題サイズを大きくして実行速度を比べてください。
3. 並列実行数を64並列から変更して実行してください。
 - ▶ プロセッサー・グリッドを 8×8 から変更する必要があります。
 - ▶ 上記の2、3ともに、データ分散(行列の確保のサイズ)の変更(コードの修正)が必須ですので、注意してください。

演習課題 3 (ScaLAPACK)

4. 大域配列`mat`に大域的な係数を代入せず、局所配列 `A = sub(mat)` に直接、各プロセスで必要な係数を代入するプログラムを作成し、メモリに関するスケーラビリティがあるように、プログラムを改良してください。
- ▶ 大域配列`mat`の確保は不要となります
 - ▶ 解ベクトル`x`は分散されて戻ります。
解の検証(解の利用方法)の並列化も必要です。

ScaLAPACK pdgesv回答

ScaLAPCK dgesvの回答 (Fortran言語版)

```
CALL PDGESV(  
&      NN, NRHS,  
&      A, IA, JA, DESCA, IPIV,  
&      B, IB, JB, DESCB,  
&      INFO )
```

お疲れさまでした