第12回　お試しアカウント付き
並列プログラミング講習会（試行）
「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部門

講習会概略

- 開催日： 2010年9月27日（月）10：30 - 17：00
  2010年9月28日（火）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：
- 9月27日（月）
  - 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）
- 9月28日（火）
  - 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　線形代数の基礎（座学）（講師：伊藤）

テストプログラム起動

ＵＮＩＸ備忘録

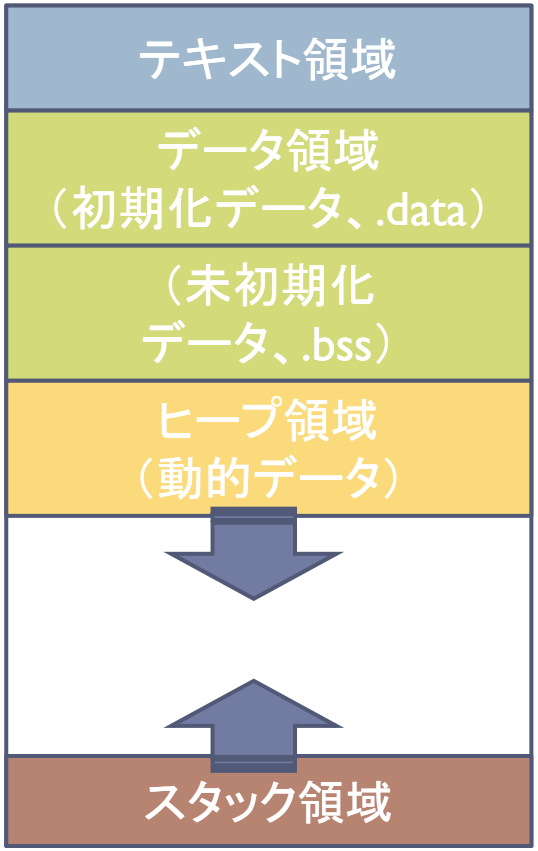
- emacsの起動： emacs　編集ファイル名
  - ^x　^s　（^はcontrol）：テキストの保存
  - ^x　^c　：　終了
    （ ^z で終了すると、スパコンの負荷が上がる。絶対にしないこと。）
  - ^g　：　訳がわからなくなったとき。
  - ^k　：カーソルより行末まで消す。
    消した行は、一時的に記憶される。
  - ^y　：^kで消した行を、現在のカーソルの場所にコピーする。
  - ^s　文字列：　文字列の箇所まで移動する。
  - ^x　goto-line：指定した行まで移動する。

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）

役に立つコマンド

1. 実行ファイルが使用するメモリ量を見る
   $ size -f <実行ファイル名>
   - .text + .data + .bss などの情報が見れます
   - HA8000の場合、1ノードでの物理メモリ上限は32GB（実際は28GB程度）です
   - malloc()関数で動的メモリ確保される場合はsizeコマンドではわかりません。
     ただし、動的確保されるメモリ量が1ノードで処理できる物理サイズを超えると、システムがハングアップしますので、ご注意ください。



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について

- C言語版

  lec2010HA＜サンプルプログラム名＞C.tar
- Fortran版

  lec2010HA＜サンプルプログラム名＞F.tar
- 上記のファイルが置いてある場所

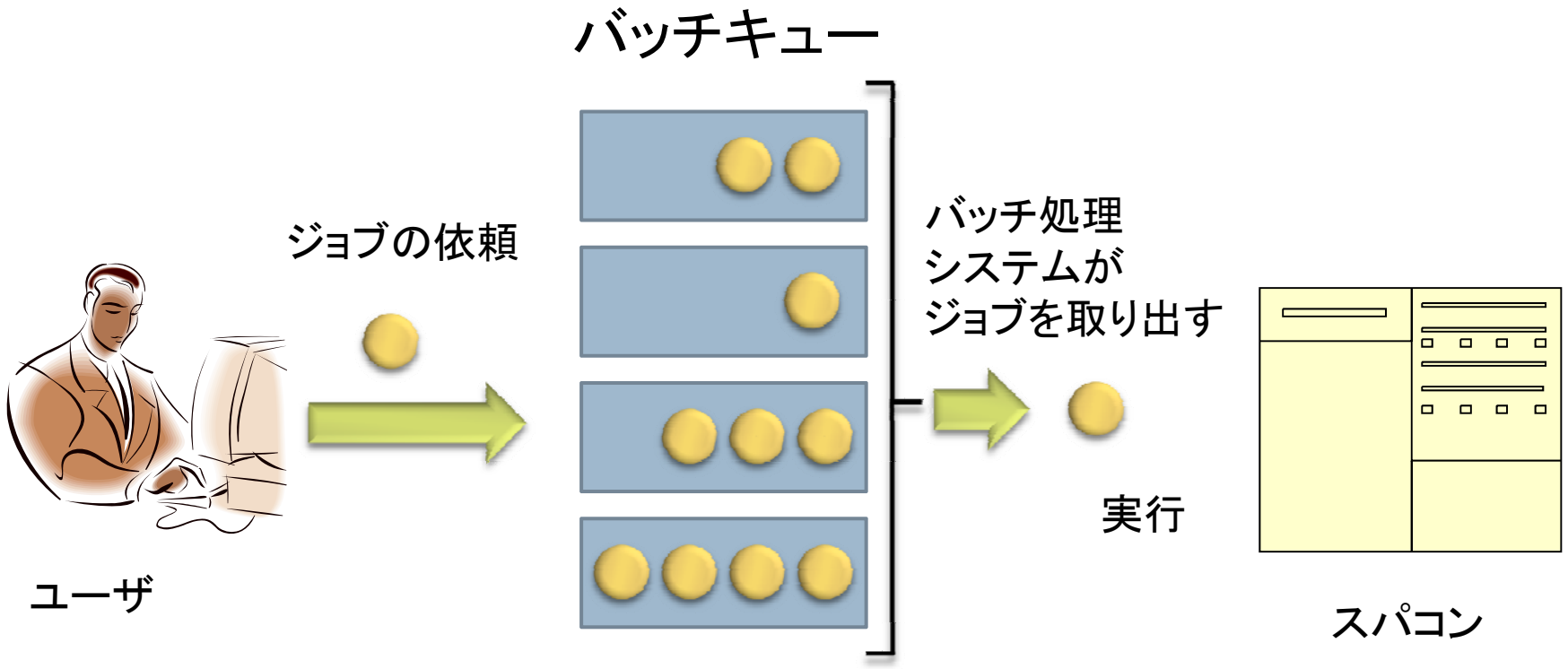
  /home/z30082

並列版Helloプログラムをコンパイルしよう

1. /home/z30082 にある lec2010HAsamplesC.tar を
   自分のディレクトリにコピーする
   $ cp /home/z30082/lec2010HAsamplesC.tar ./
2. lec2010HAsamplesC.tar を展開する
   $ tar xvf lec2010HAsamplesC.tar
3. Hello フォルダに入る
   $ cd Hello
4. ピュアMPI用のMakefileをコピーする
   $ cp Makefile_pure Makefile
5. make する
   $ make
6. 実行ファイル(hello)ができていることを確認する
   $ ls

バッチ処理とは

- スパコン環境では、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。



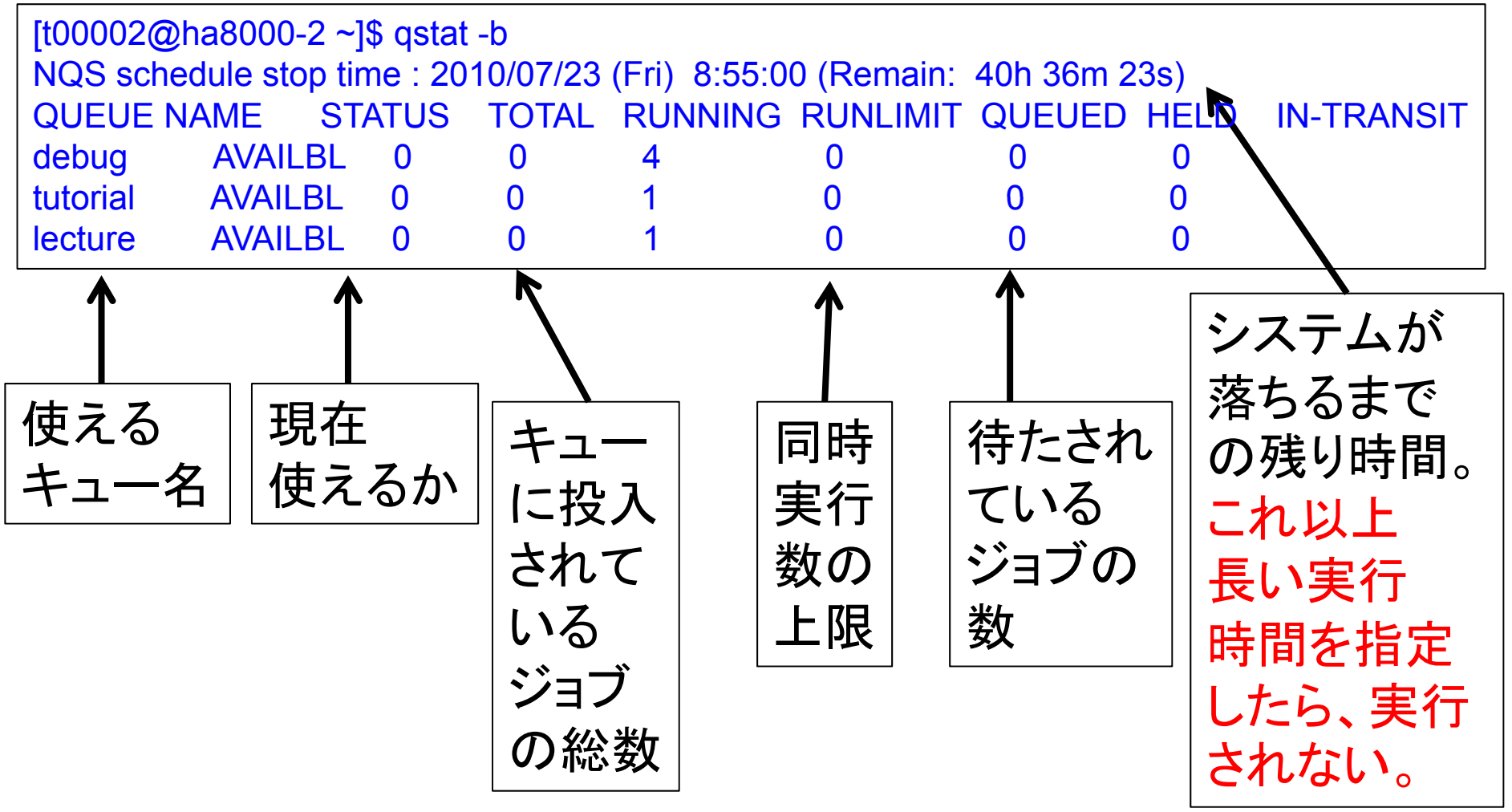
NQSキューの設定のしかた

- バッチ処理は、Network Queuing System(NQS)で管理されています。
- 以下、NQSの主要コマンドを説明します。
  - ジョブの投入： qsub <ジョブスクリプトファイル名>
  - 自分が投入したジョブの状況確認： qstat
  - 混雑度を見る： qstat -b
  - 投入ジョブの削除： qdel <ジョブID>
  - 実行中のジョブの削除： qdel -k <ジョブID>
- 本演習のNQSキュー名： tutorial
  - 最大15分までの実行を許可
  - 最大ノード数は4

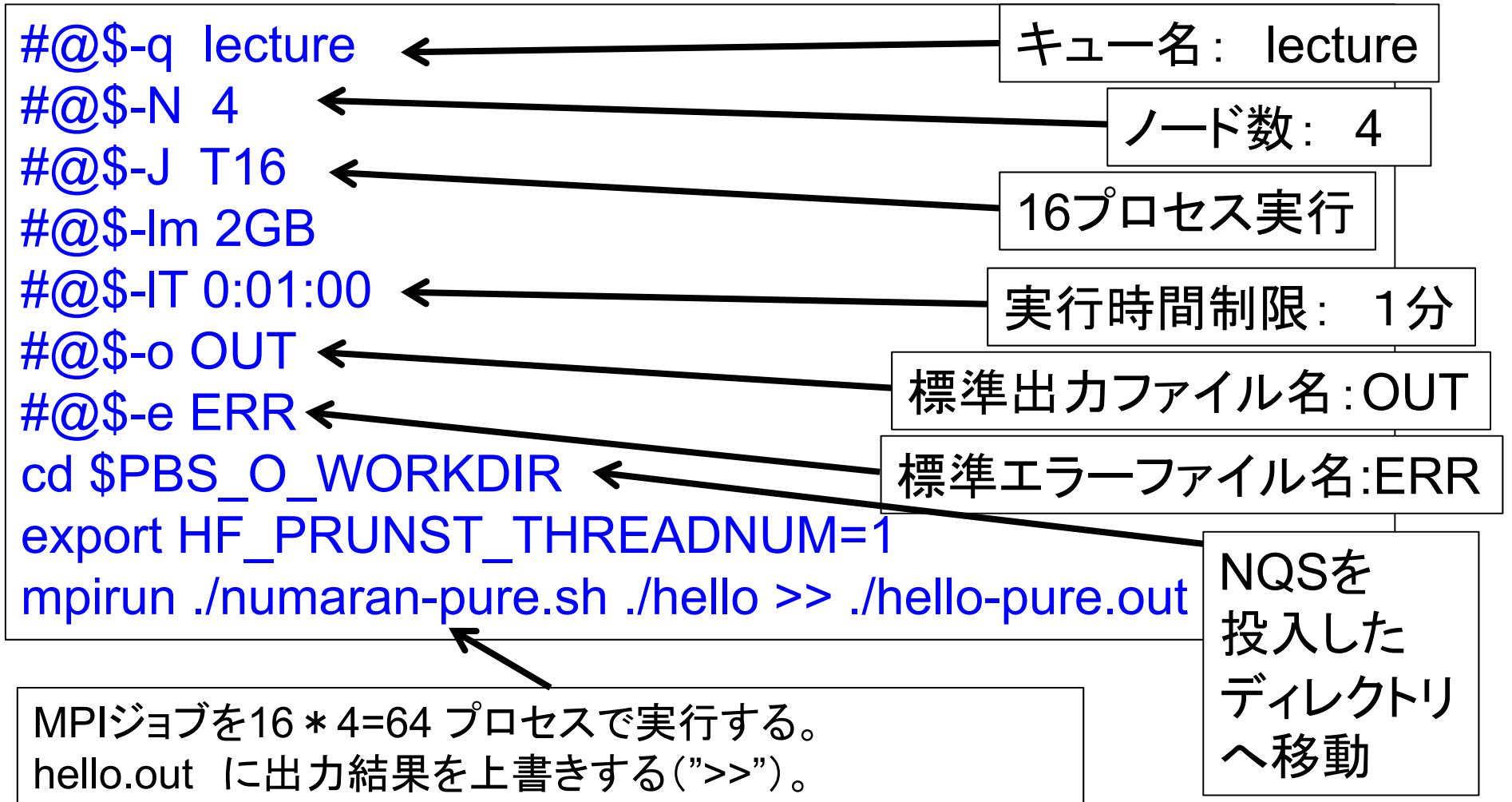
NQSキューの種類

- “qstat -b” と入力すると
  キューの種類を見ることができます
- tutorial：
  本講習会時間内で使えるキュー
- lecture ：
  本講習会時間外で使えるキュー

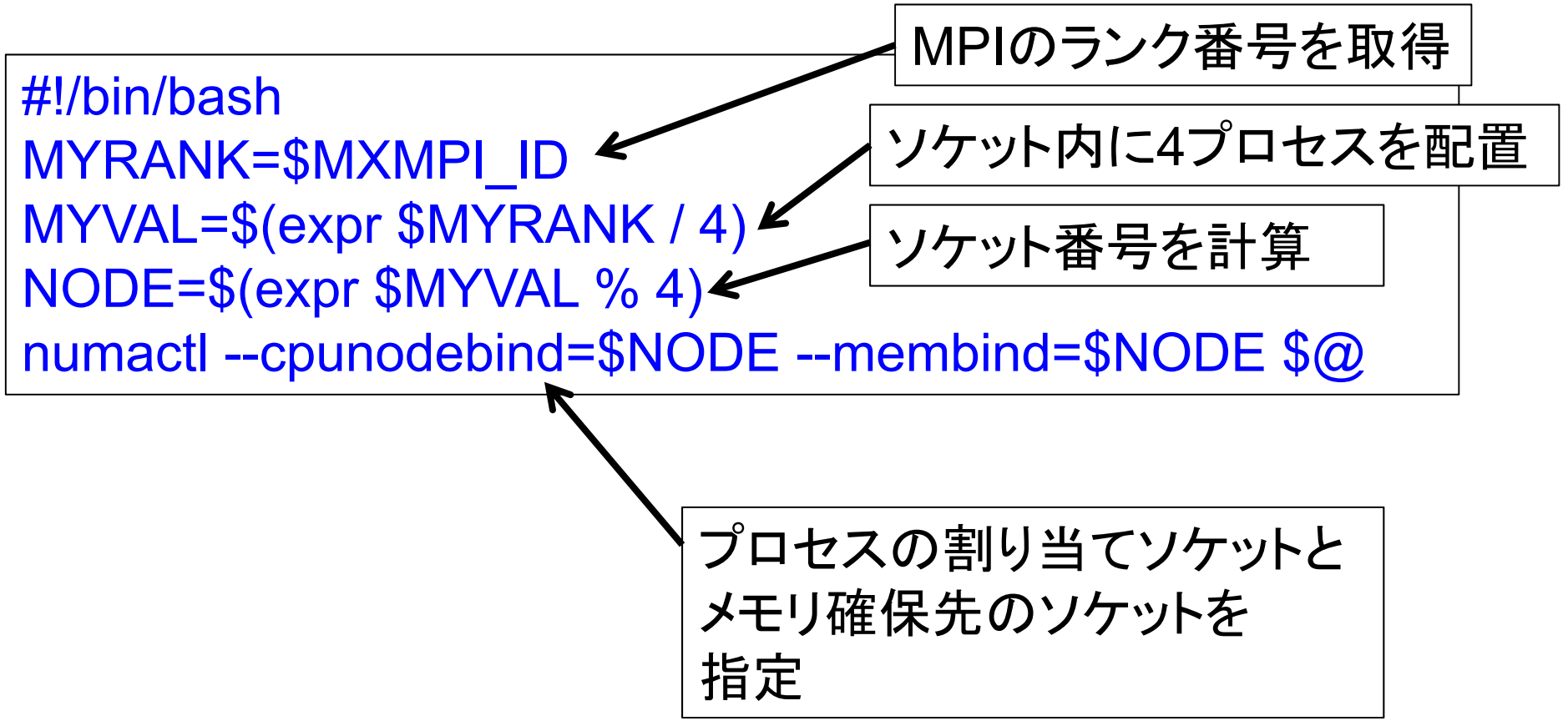
qstat –b の実行画面例



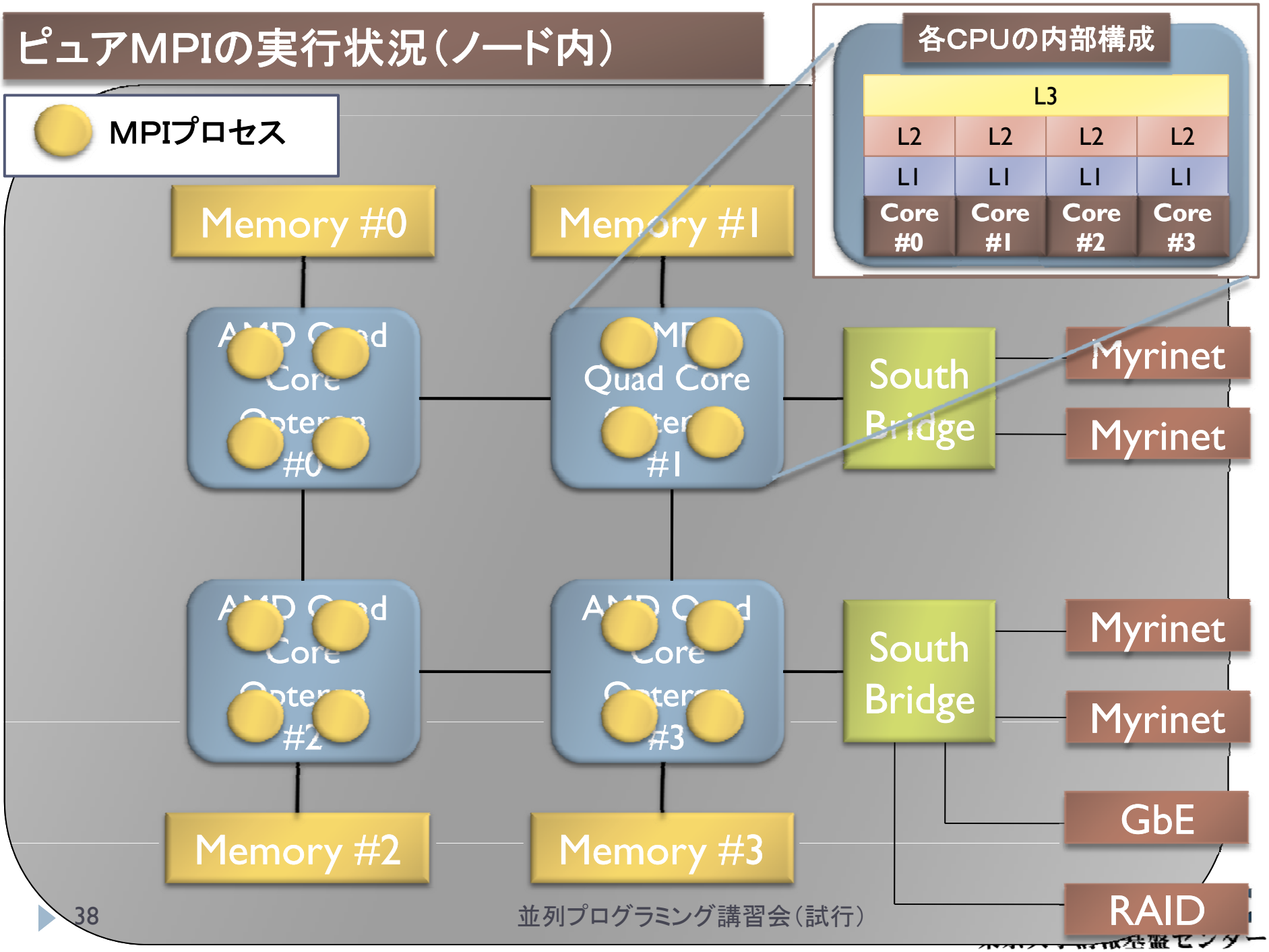
JOBスクリプトサンプルの説明（ピュアMPI）（C言語、Fortran言語共通）



numaran-pure.shの中身（ピュアＭＰＩ）（C言語、Fortran言語共通）



ピュアMPIの実行状況（ノード内）



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
   $ qsub hello-pure.bash
2. 自分の導入されたジョブを確認する
   $ qstat
3. 実行が終了すると、以下のファイルが生成される
   hello-pure.out
4. 上記ファイルの中身を見てみる
   $ cat hello-pure.out
5. “Hello parallel world!”が、16プロセス*4ノード=64表示されていたら成功。

NQSに関連するTIPS

- NQS 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てくる。
- 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

サンプルプログラムについて

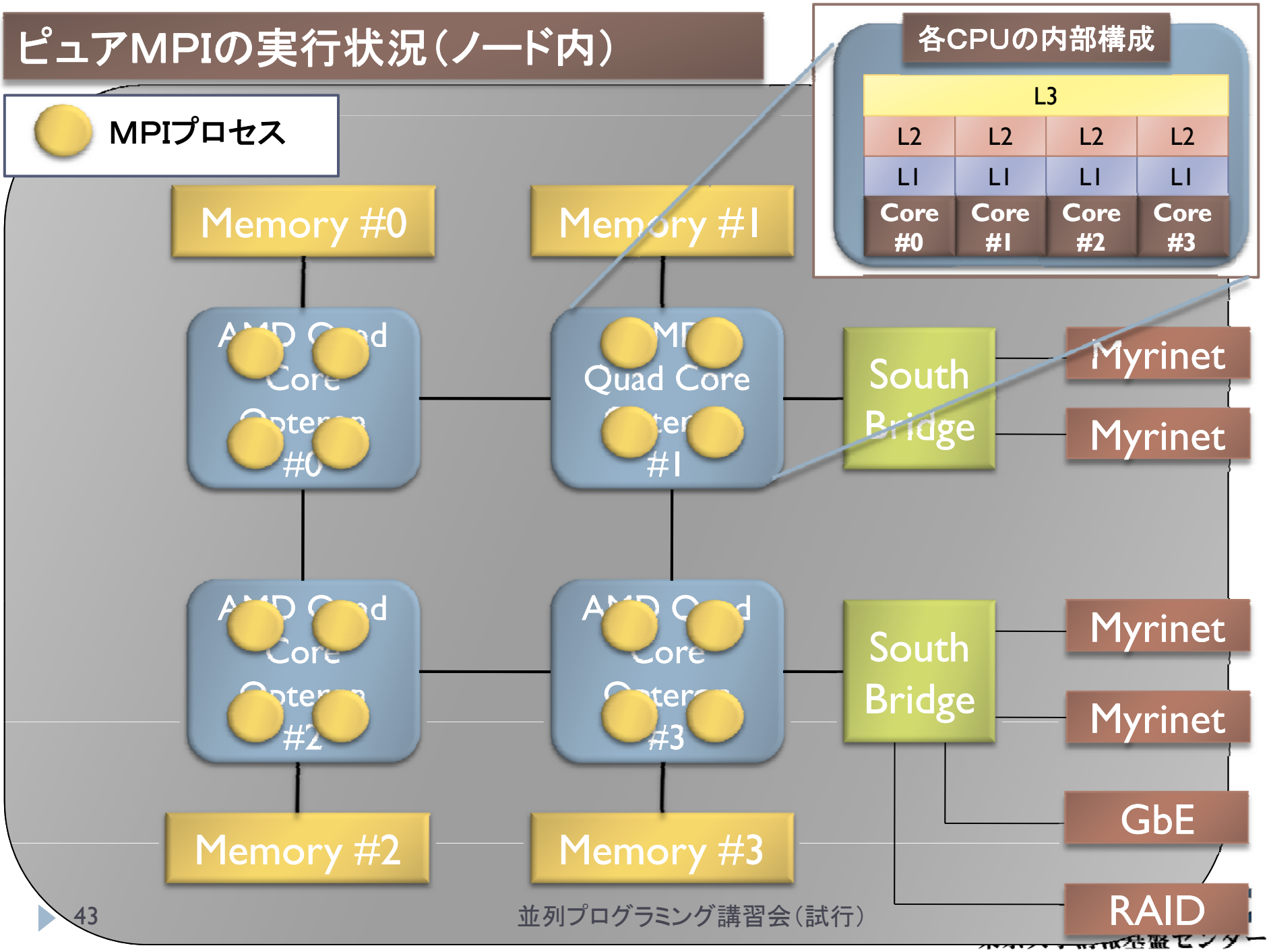
- C言語版

  lec2010HA＜サンプルプログラム名＞C.tar
- Fortran版

  lec2010HA＜サンプルプログラム名＞F.tar
- 上記のファイルが置いてある場所

  /home/z30082

ピュアMPIの実行状況（ノード内）



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
   $ qsub hello-pure.bash
2. 自分の導入されたジョブを確認する
   $ qstat
3. 実行が終了すると、以下のファイルが生成される
   hello-pure.out
4. 上記ファイルの中身を見てみる
   $ cat hello-pure.out
5. “Hello parallel world!”が、16プロセス*4ノード=64表示されていたら成功。

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_hy4 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-hy4.bash）のキュー名を変更する。“lecture”　→“tutorial”に変更する。
   $ emacs hello-hy4.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
   $ qsub hello-hy4.bash
2. 自分の導入されたジョブを確認する
   $ qstat
3. 実行が終了すると、以下のファイルが生成される
   hello-hy4.out
4. 上記ファイルの中身を見てみる
   $ cat hello-hy4.out
5. “Hello parallel world!”が、4プロセス*4ノード=16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MPI）（C言語、Fortran言語共通）

```
#@$-q  lecture
#@$-N  4
#@$-J  T4
#@$-lm 2GB
#@$-lT 0:01:00
#@$-o OUT
#@$-e ERR
cd $PBS_O_WORKDIR
export HF_PRUNST_THREADNUM=4
mpirun ./numaran-4.sh ./hello >> ./hello-hy4.out
```

キュー名： lecture

ノード数： 4

4プロセス実行

実行時間制限： 1分

標準出力ファイル名：OUT

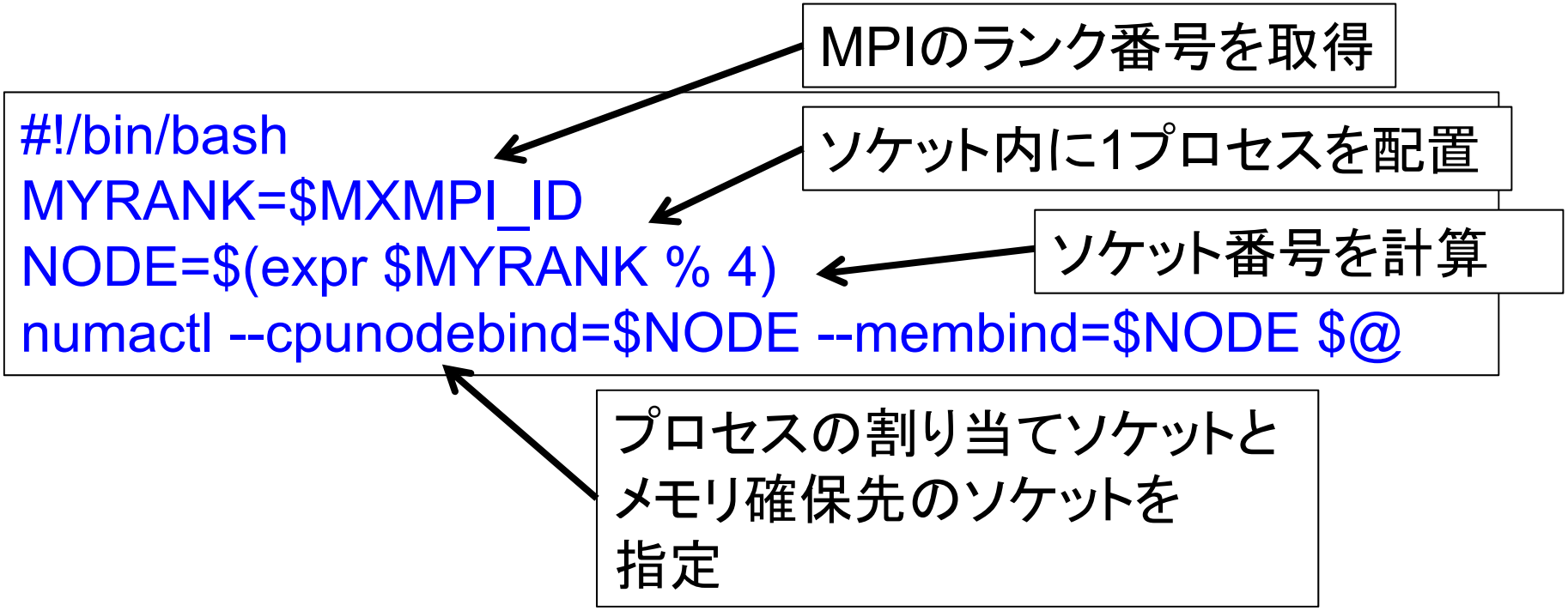
標準エラーファイル名:ERR

NQSを投入したディレクトリへ移動

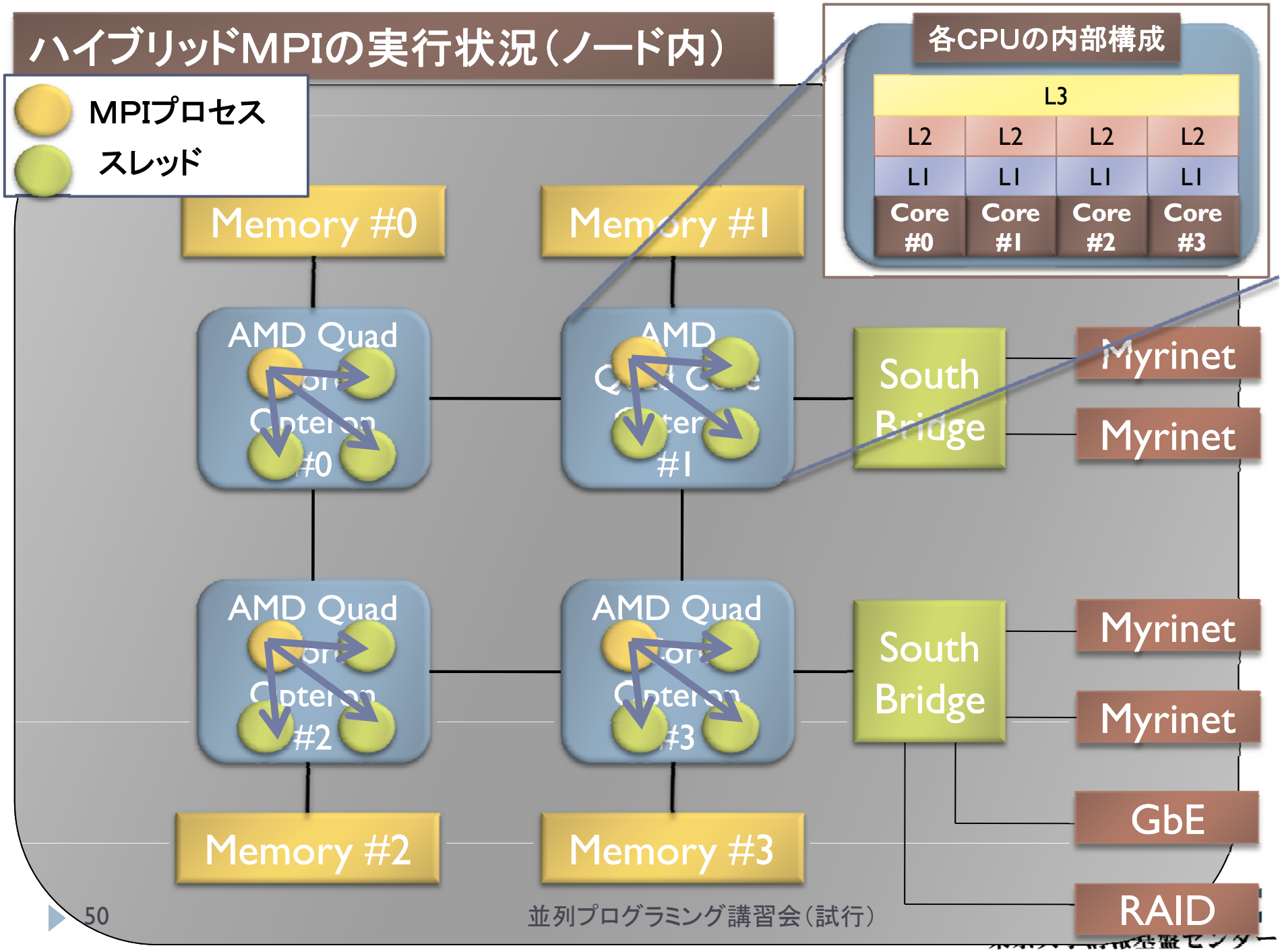
スレッド数（起動プロセス数を含む）の指定

MPIジョブを16プロセスで実行する。
hello.out に出力結果を上書きする（”>>”）。
※上書きしないで新規作成（“＞”）：ファイルが無いことを確認

numaran-4.shの中身（ピュアＭＰＩ）（C言語、Fortran言語共通）



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>

void 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);
}
```

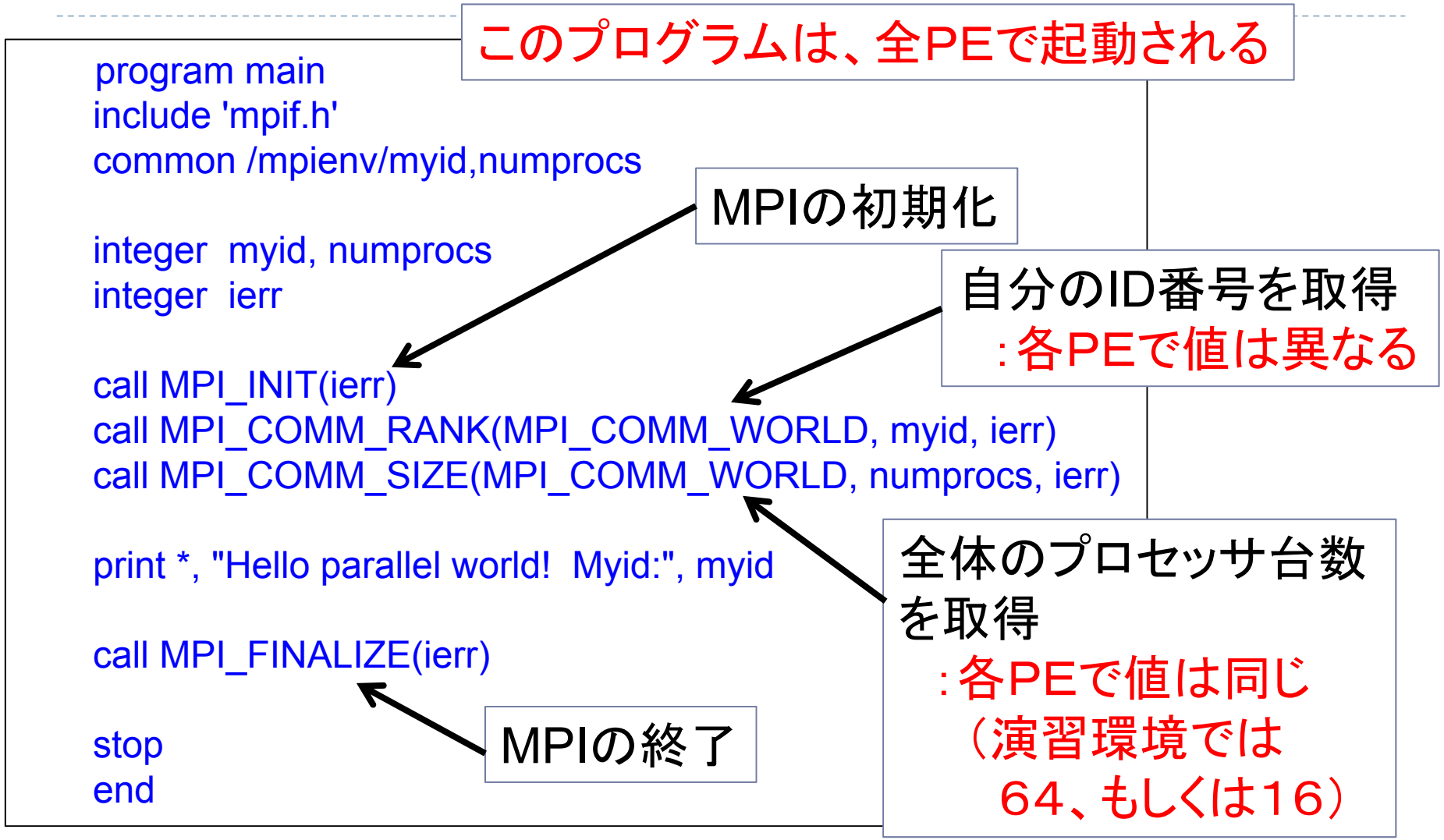
MPIの初期化

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64、もしくは16）

MPIの終了

並列版Helloプログラムの説明（Fortran言語）



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

各プロセッサーで、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

各プロセッサーで、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

任意のプロセス数での実行方法（東大センタ限定）

- 以下のように、*.bashファイルを修正してください
  - 2プロセス／ノードでの実行
    #@$-J T2
  - 4プロセス／ノードでの実行
    #@$-J T4
  - pプロセス／ノードでの実行
    #@$-J Tp
- 実習環境では、1～16プロセス／ノードまで
  しかないことに注意

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-hy4.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

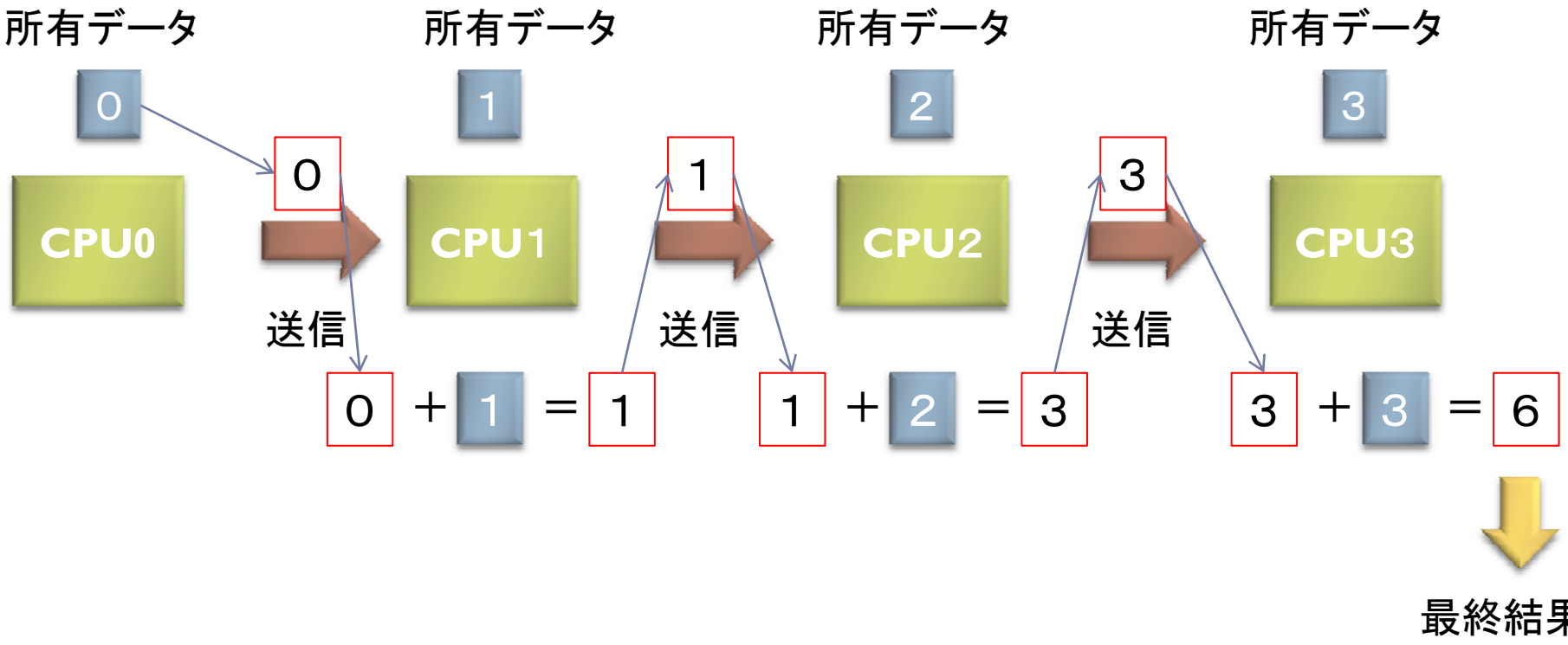
演習課題

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、PE0からChar型のデータ“Hello World!!”を、その他のPEに送信する
   - その他のPEでは、MPI_Recvで受信して表示する

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. （63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

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);
....
}
```

受信用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
```

受信用システム配列の確保

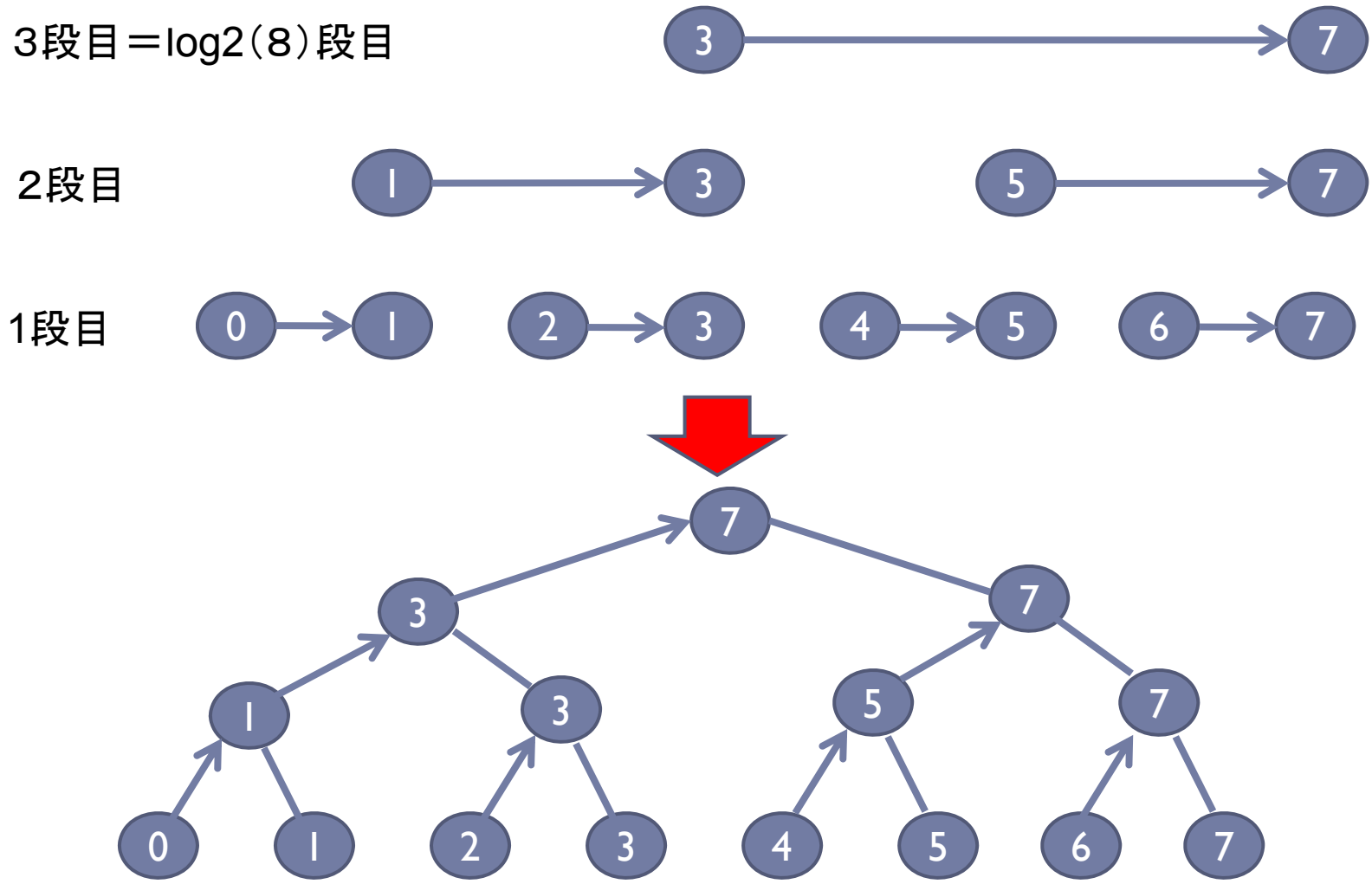
自分より一つ少ないID番号（myid-1）から、double型データ1つを受信しdrecvbuf変数に代入

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 < log2(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）



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：

    myid & k が k 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：

    myid + k
    - つまり 、通信が成立するPE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、log2(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）

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　特任准教授　片桐孝洋

2010年9月27日（月）14:00－15:00

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

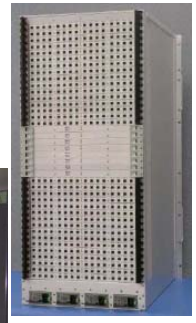
HITACHI SR11000 model J2

Total Peak performance	: 18.8 TFLOPS
Total number of nodes	: 128
Total memory	: 16384 GB
Peak performance per node	: 147.2 GFLOPS
Main memory per node	: 128 GB
Disk capacity	: 94.2 TB
IBM POWER5+ 2.3GHz	

T2K東大（HA8000クラスタシステム）

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	

ノード製品名：HITACHI HA8000-tc/RS425



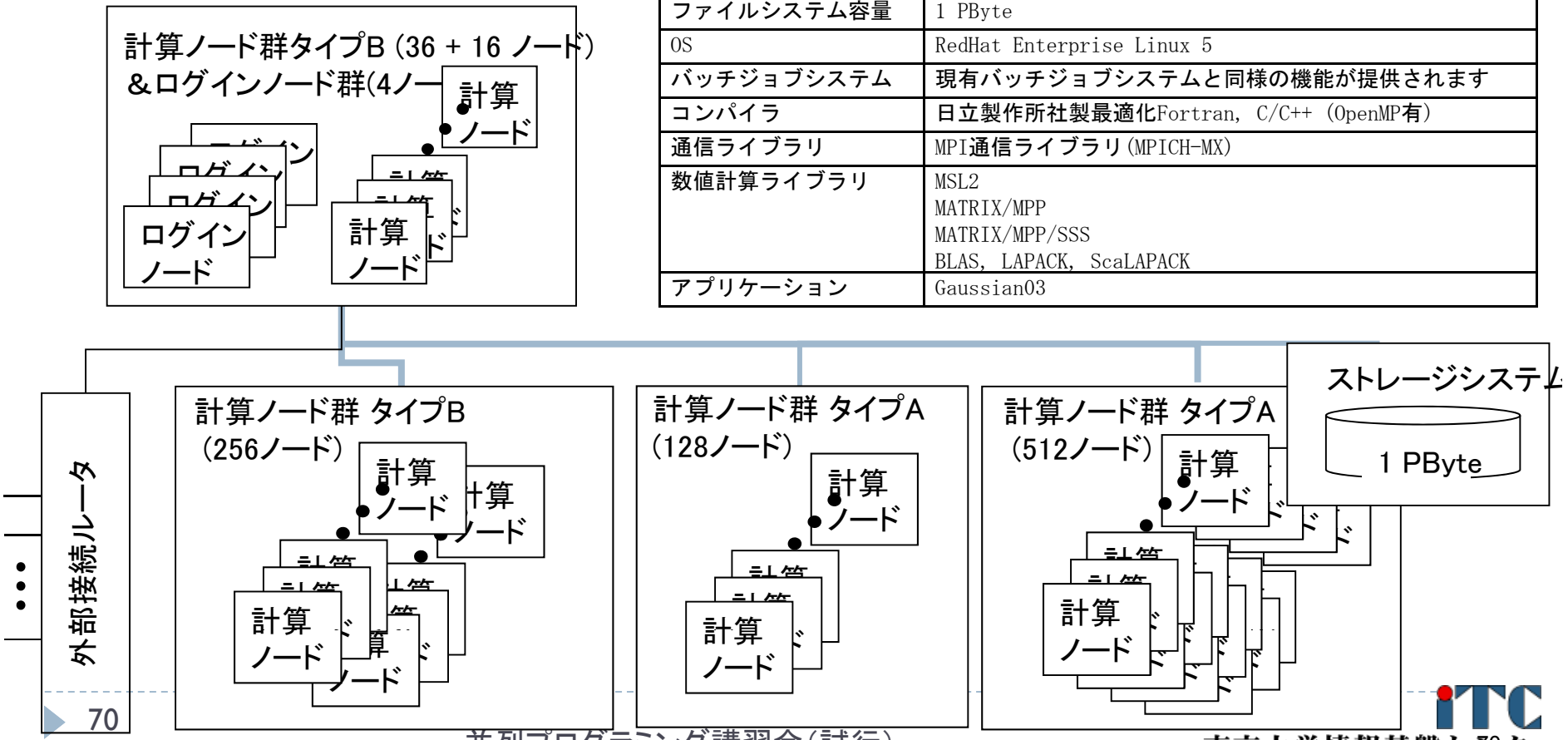
T2Kオープンスパコン東大版
HA8000-tc/RS425クラスタ群

ノード単体仕様

CPU	AMD社製Quad Core Opteron 2.3 GHz
ソケット数(CPU数)	4 (16)
理論演算性能	147.2 GFlops
主記憶容量	32 GByte (936ノード)、128GByte(16ノード)
ディスク	250 GByte

全体仕様

総理論演算性能	140.1344 TFlops
総主記憶容量	31.25 TByte
計算ノード数	948
ログインノード数	4
ネットワーク性能（両方向物理データ転送性能）	タイプA(512,128ノード)：5 GByte/sec (Myrinet-10G x 4) タイプB(256,36,16)： 2.5 GByte/sec (Myrinet-10G x 2)
ファイルシステム容量	1 PByte
OS	RedHat Enterprise Linux 5
バッチジョブシステム	現有バッチジョブシステムと同様の機能が提供されます
コンパイラ	日立製作所社製最適化Fortran, C/C++ (OpenMP有)
通信ライブラリ	MPI通信ライブラリ(MPICH-MX)
数値計算ライブラリ	MSL2 MATRIX/MPP MATRIX/MPP/SSS BLAS, LAPACK, ScaLAPACK
アプリケーション	Gaussian03



東京大学情報基盤センター
Information Technology Center,The University of Tokyo

タイプA

ノード数	512, 128 （2つの群から構成）
ネットワーク性能	5 GB/sec （片方向）
	Full bisection bandwidth
CPU	AMD 社 製 Quad Core Opteron 2.3 GHz
ソケット数（CPU数）	4 (16)
理論演算性能	147.2 GFlops
主記憶容量	32 GByte
ディスク	250 GByte

128ノード

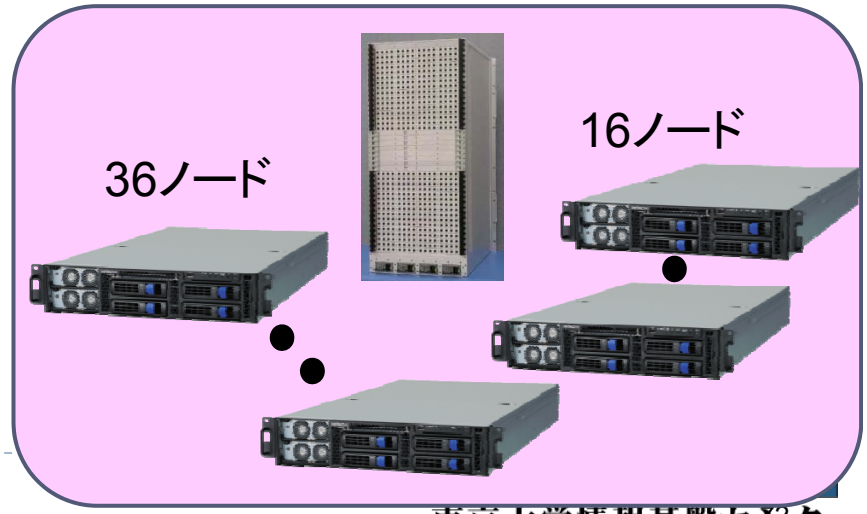
512ノード

タイプB

ノード数	256, 36, 16 （3つの群から構成）
ネットワーク性能	2.5 GB/sec （片方向）
	Full bisection bandwidth
CPU	AMD社製Quad Core Opteron 2.3 GHz
ソケット数(CPU数)	4 (16)
理論演算性能	147.2 GFlops
主記憶容量	32 GByte (16ノードは128GByte)
ディスク	250 GByte

なお、ログインノード4台の構成は36ノードと同様です

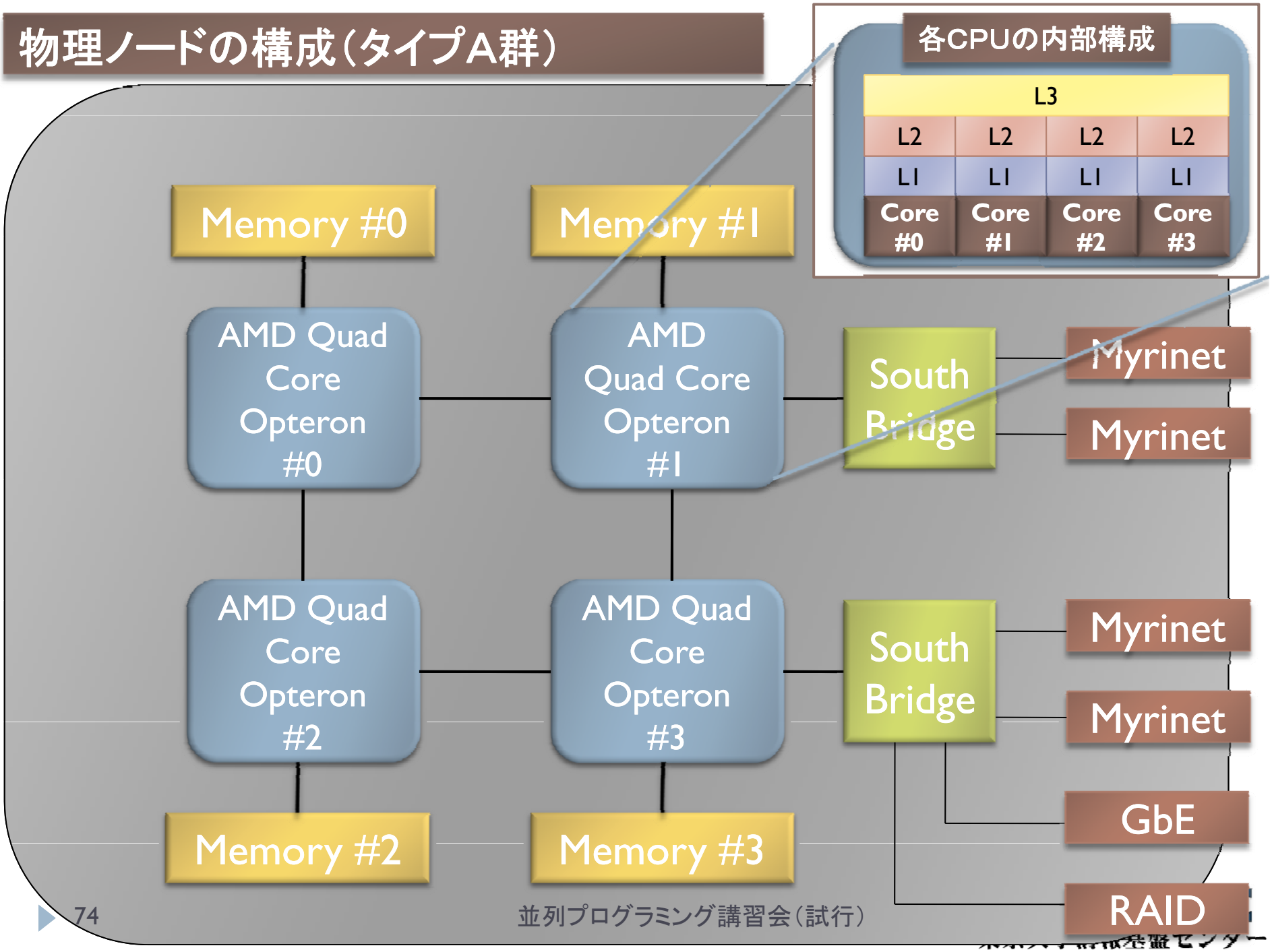
256ノード



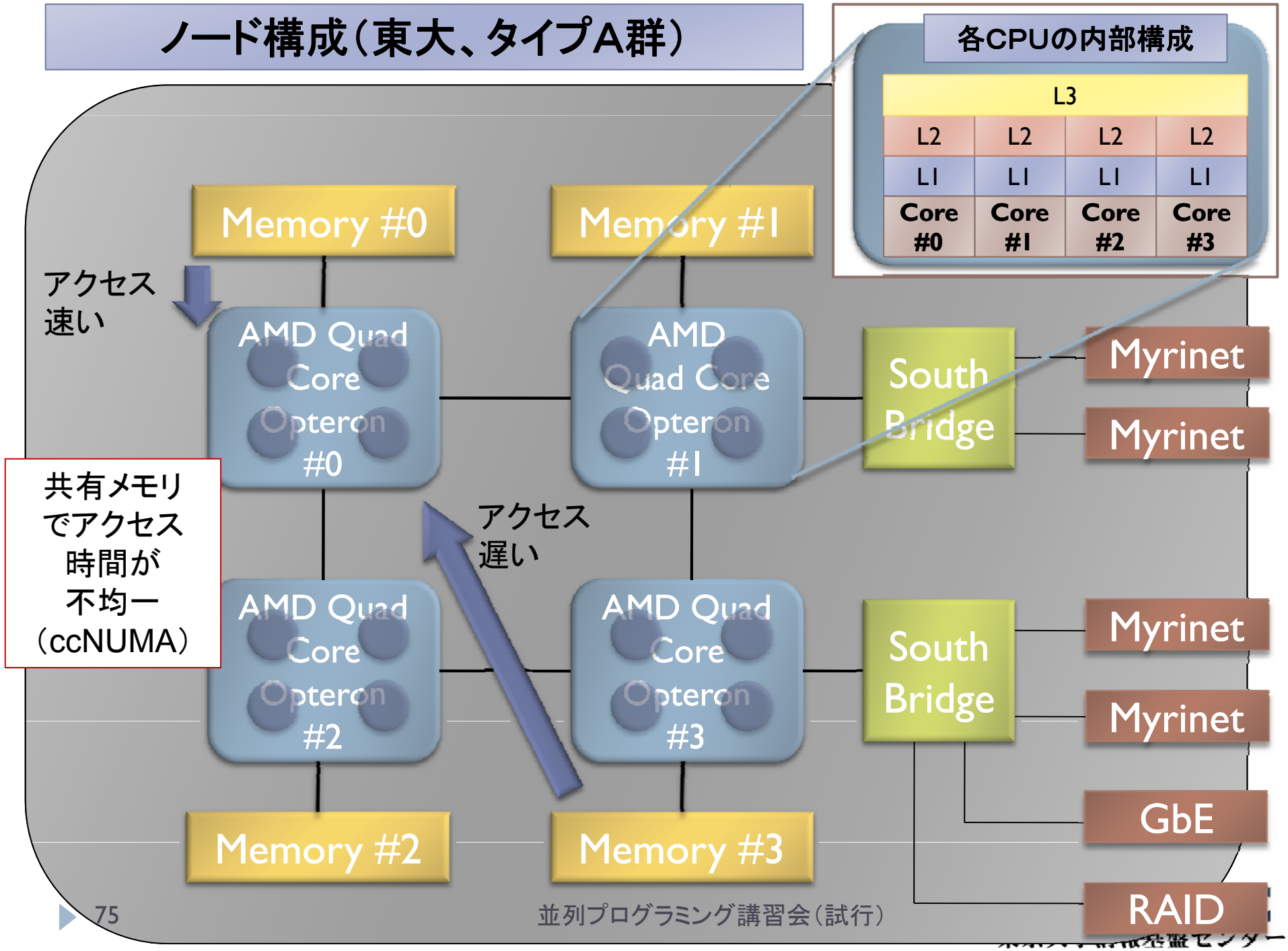
T2Kオープンスパコン CPU緒元
(AMD Quad Core Opteron 2.3GHz)
：東大、京大、筑波大に設置のスパコン

項目	値
L1キャッシュ	64 Kbytes (命令、データ、双方は分離)
	2 Way Associativity(ライトバック、3サイクル) キャッシュライン:64 bytes、LRU置換
L1データTLB	Full Associativity 2Mbyte Page:8エントリ, 4Kbyte Page:32エントリ
L2キャッシュ	512 Kbytes (2300 MHz)
L3キャッシュ	2048 Kbytes
命令デコード	3 Way
演算実行	3 Way (整数、アドレス生成、浮動小数点)
SIMD命令実行	MMX, SSE, SSE2
命令実行	乱発行乱終了(Out-of-order) 整数、浮動小数点
レジスタ	● レガシーモード：汎用32bit: 8個、128bit-XMM:8個、64bit MMX:8個(x87互換用:8個と同一) ● 64bitモード：汎用64bit:16個、128bit-XMM:16個、64bit MMX:8個(x87互換用:8個と同一)
システムバス	1000 MHz

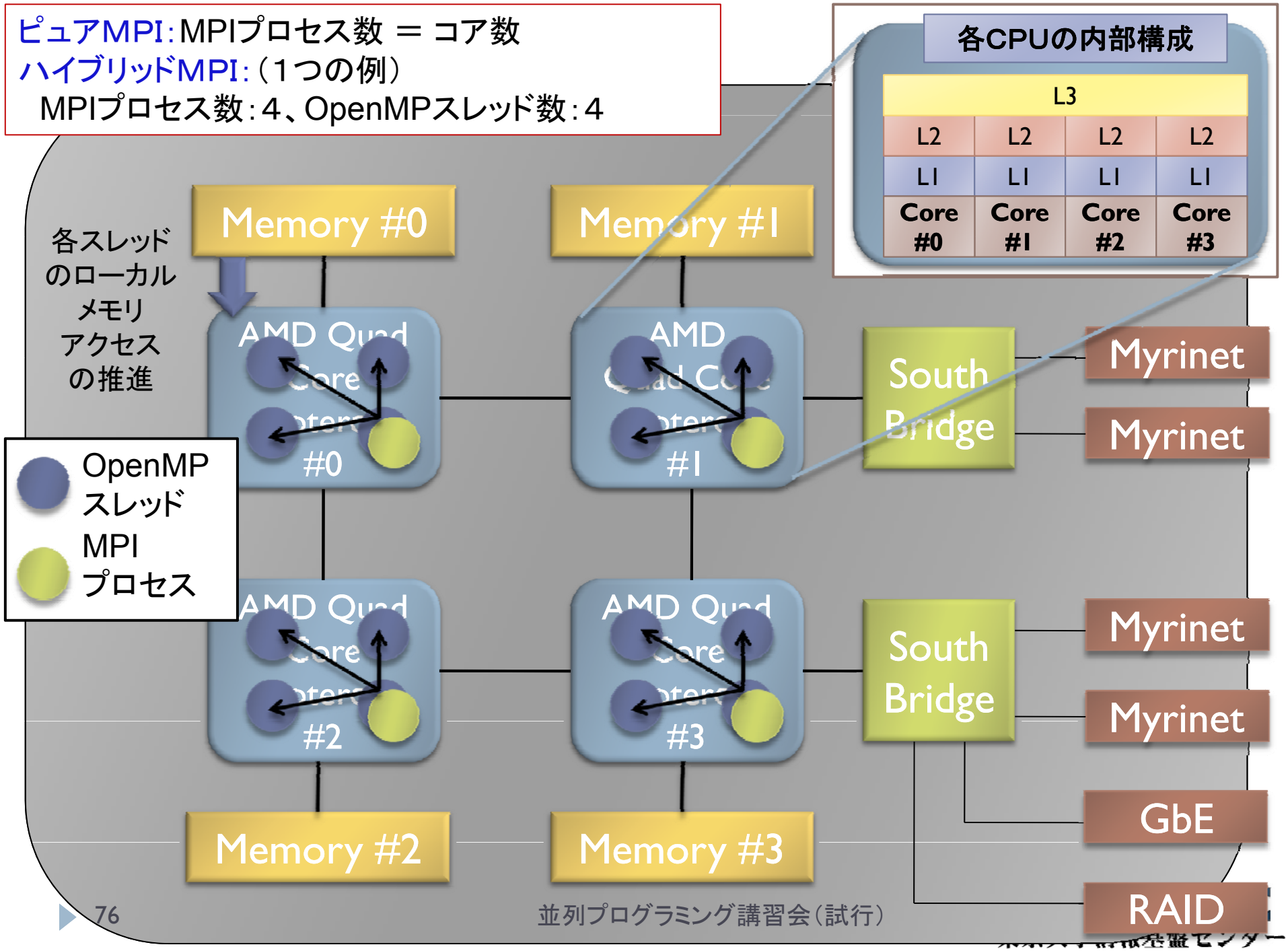
物理ノードの構成（タイプA群）



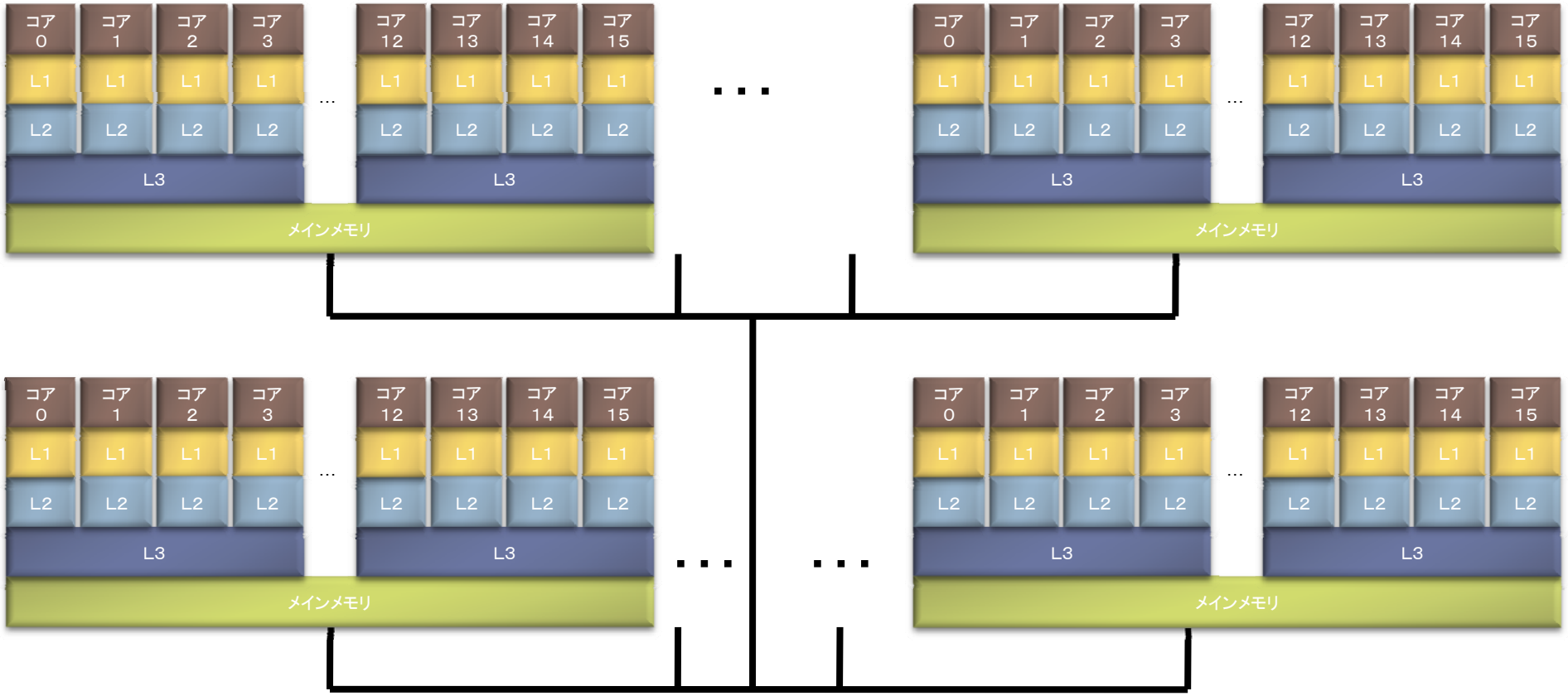
ノード構成（東大、タイプA群）



ピュアMPI：MPIプロセス数 ＝ コア数
ハイブリッドMPI：（1つの例）
MPIプロセス数：4、OpenMPスレッド数：4



スパコンでの全体メモリ構成図



メモリが多段に階層化
（L1、L2、L3、分散メモリ）

高速ネットワーク
（5Gバイト／秒
×双方向）
（タイプA群）

Ｔ２Ｋオープンスパコンの詳細情報

- 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/service/ha8000/

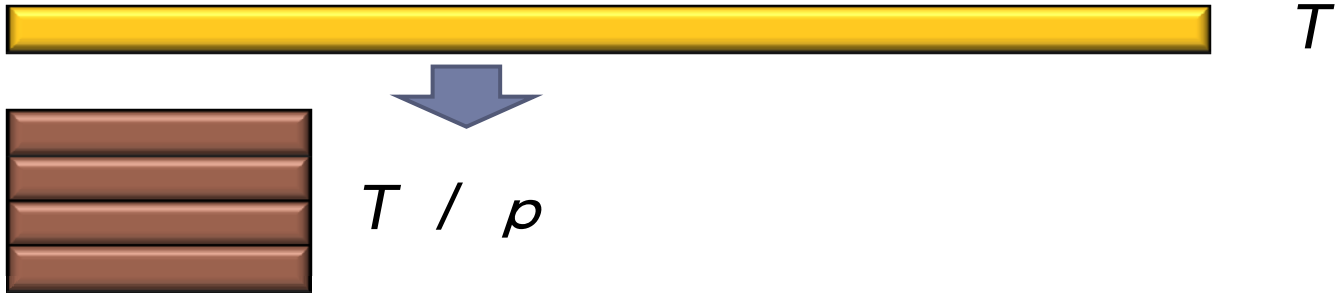
東大情報基盤センタHA8000クラスタシステムの料金表（2008年6月1日制定）

- パーソナルコース（年間）
  - コース1： 110，000円 ： 4ノード
  - コース2： 150，000円 ： 8ノード
  - コース3： 220，000円 ：16ノード
  - コース4： 300，000円 ：32ノード
  - コース5： 440，000円 ：64ノード
  - 月に1度、ひとつ上のコースで実行可能。コース5は、256ノード。
- 専用キュー
  - 1，000，000円 ：8ノード
- ノード固定
  - 1，500，000円 ：8ノード
- 以上は、「大学等」料金

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）

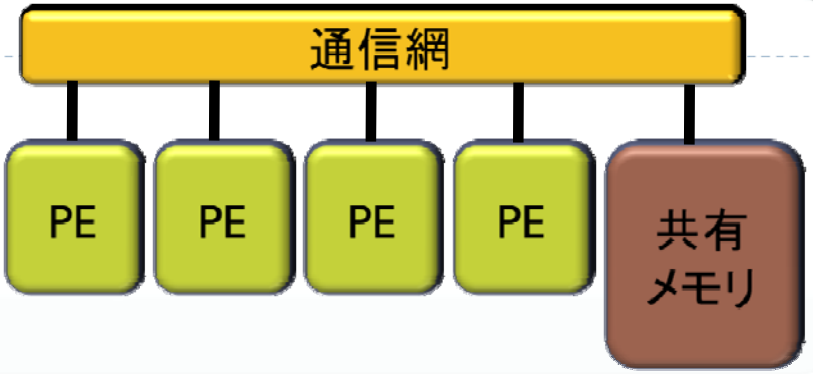
  - 時分割多重、疑似並列
  - OSによるプロセス実行スケジューリング（ラウンドロビン方式）

並列計算機の分類

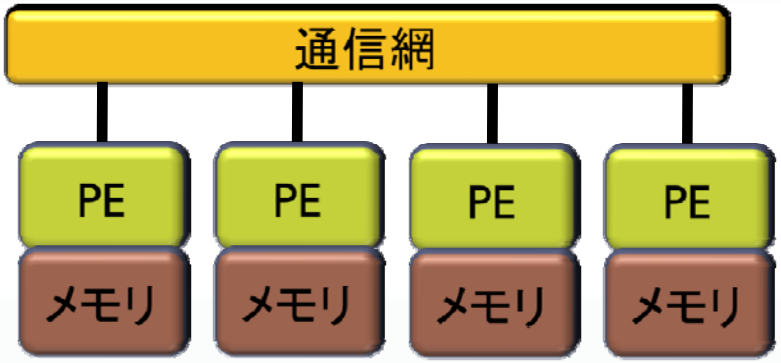
- Michael J. Flynn教授（スタンフォード大）の分類（1966）
- 単一命令・単一データ流
  （SISD, Single Instruction Single Data Stream）
- 単一命令・複数データ流
  （SIMD, Single Instruction Multiple Data Stream）
- 複数命令・単一データ流
  （MISD, Multiple Instruction Single Data Stream）
- 複数命令・複数データ流
  （MIMD, Multiple Instruction Multiple Data Stream）

並列計算機のメモリ型による分類

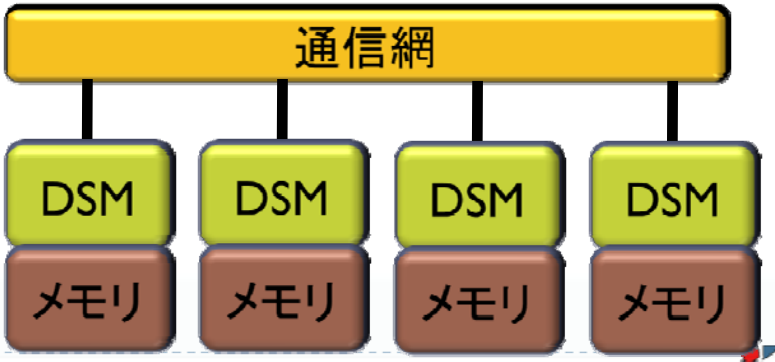
1. 共有メモリ型
（SMP、
Symmetric Multiprocessor）



2. 分散メモリ型
（メッセージパッシング）

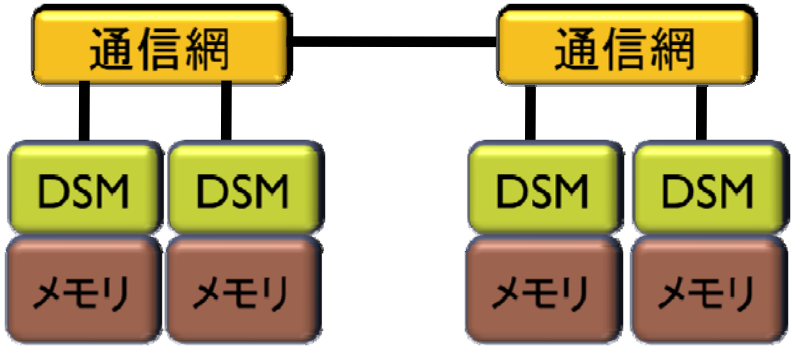


3. 分散共有メモリ型
（DSM、
Distributed Shared Memory）



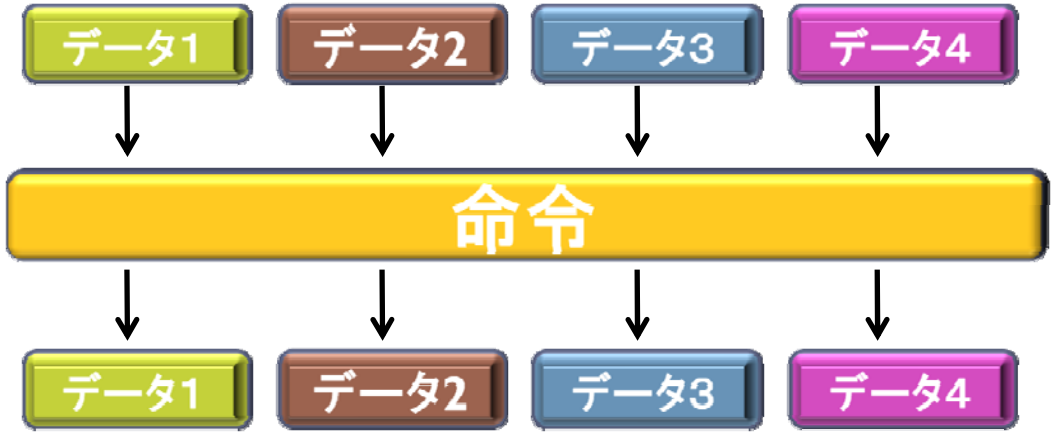
並列計算機のメモリ型による分類

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
- 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

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」

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：行列ー行列積

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

●並列化

全CPUで共有

CPU0 $\begin{pmatrix} 1 & 2 & 3 \end{pmatrix}$

CPU1 $\begin{pmatrix} 4 & 5 & 6 \end{pmatrix}$

CPU2 $\begin{pmatrix} 7 & 8 & 9 \end{pmatrix}$

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

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：カレールゥを入れる

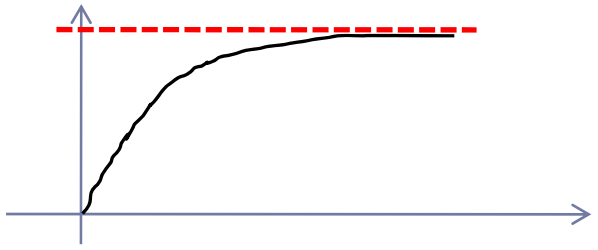
●並列化

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: $S_P = T_S / T_P \quad (0 \leq S_p)$
  - 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。

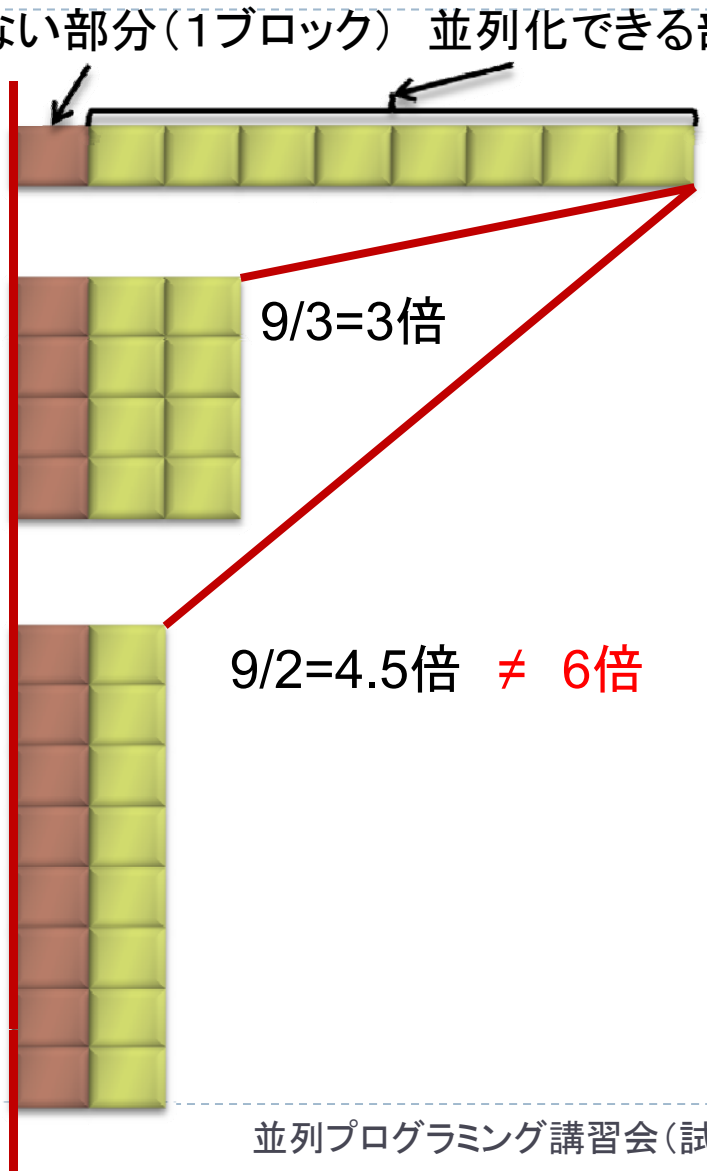
$$S_P = K/(K\alpha/P + K(1-\alpha))$$

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

- 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→∞)、台数効果は、高々 1／(1－α) 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 にしかならない！

  →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



●逐次実行

＝88.8%が並列化可能

●並列実行（4並列）

9/3=3倍

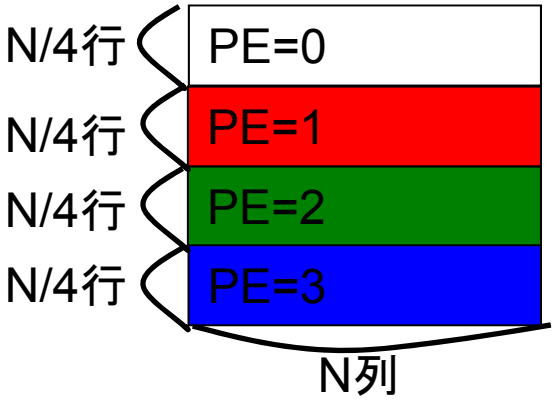
●並列実行（8並列）

9/2=4.5倍 ≠ 6倍

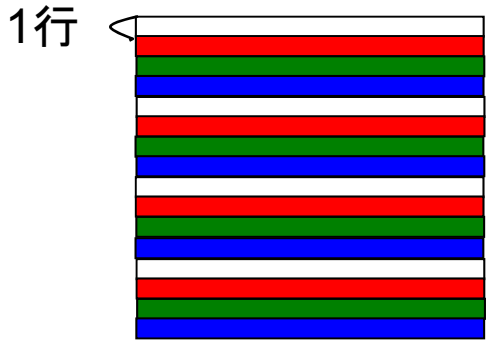
１．基本演算

- 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

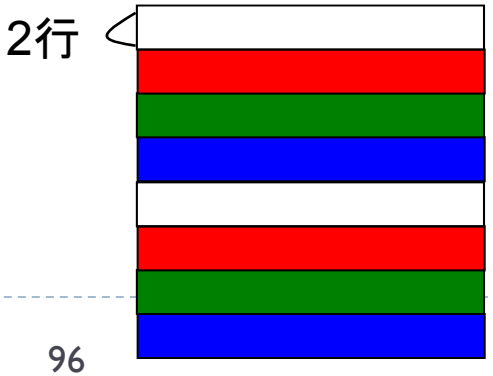
1.1.1　1次元分散



- (行方向) ブロック分割方式
- (Block, *) 分散方式



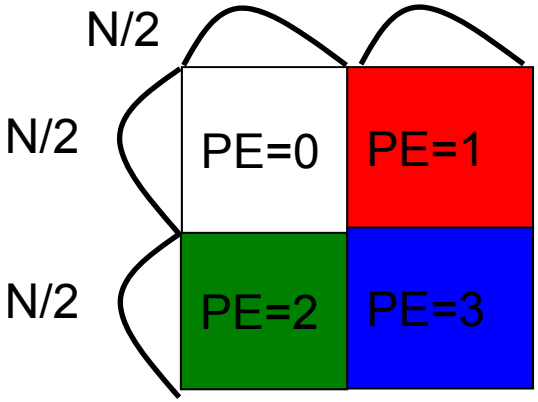
- (行方向) サイクリック分割方式
- (Cyclic, *) 分散方式



- (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

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

- 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

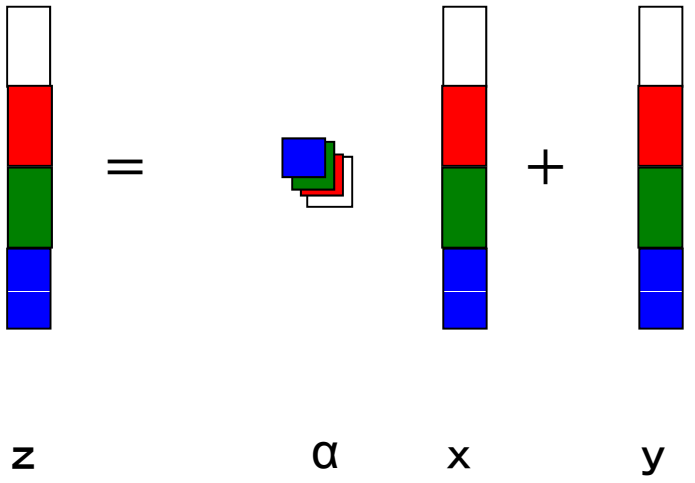
- 二次元ブロック・サイクリック分割方式
- (Cyclic(2), Cyclic(2))分散方式

1.2 ベクトルどうしの演算

- 以下の演算

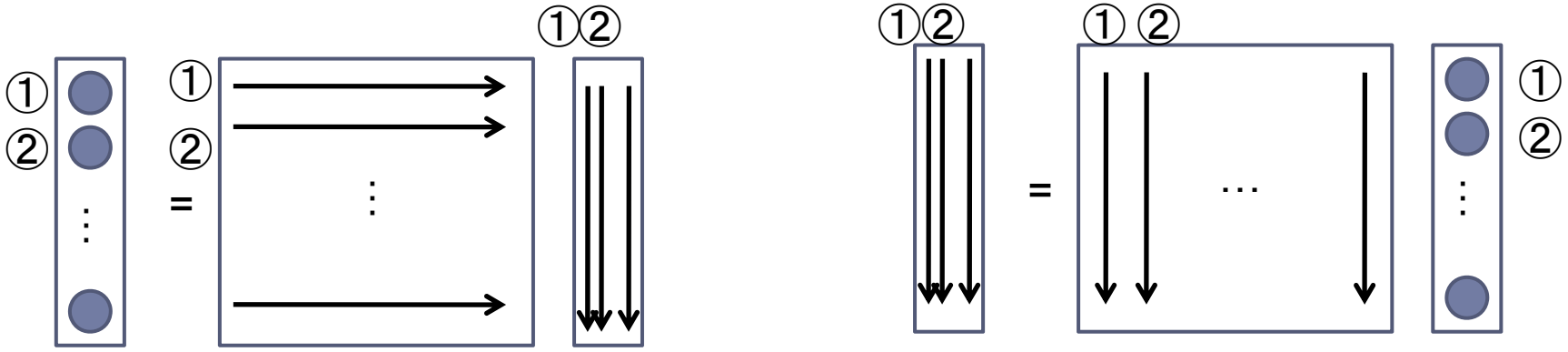
$$z = ax + y$$

  - 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



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

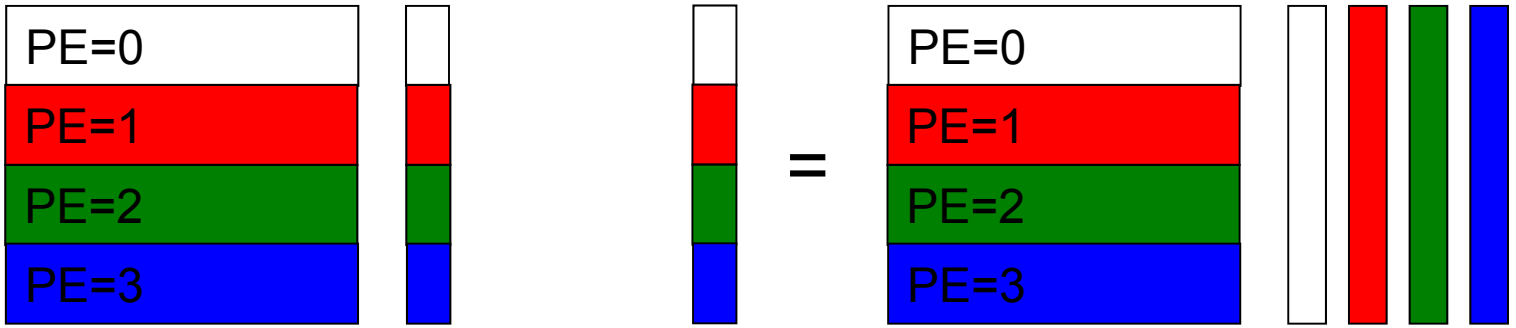
```
for(j=0; j<n; j++) y[j]=0.0;
for(j=0; j<n; j++) {
   for (i=0; i<n; i++) {
      y[i] += a[i][j]*x[j];
   }
}
```

＜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＞

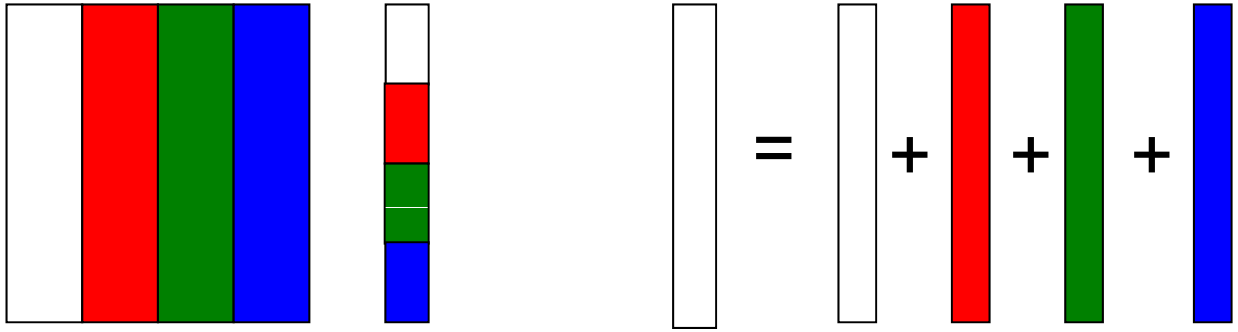
＜行方向分散方式＞　：行方式に向く分散方式



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



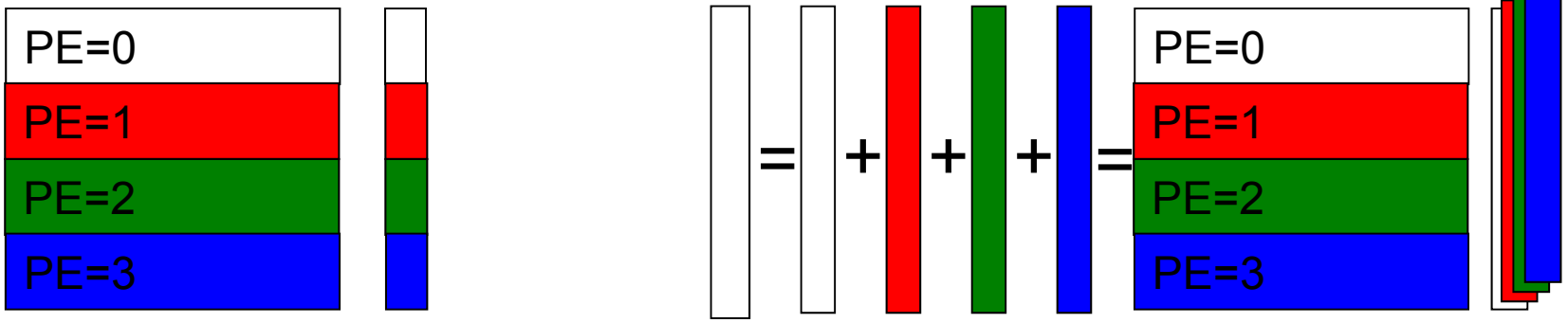
各PE内で行列-ベクトル積を行う

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）

1.3 行列とベクトルの積

＜列方式の場合＞

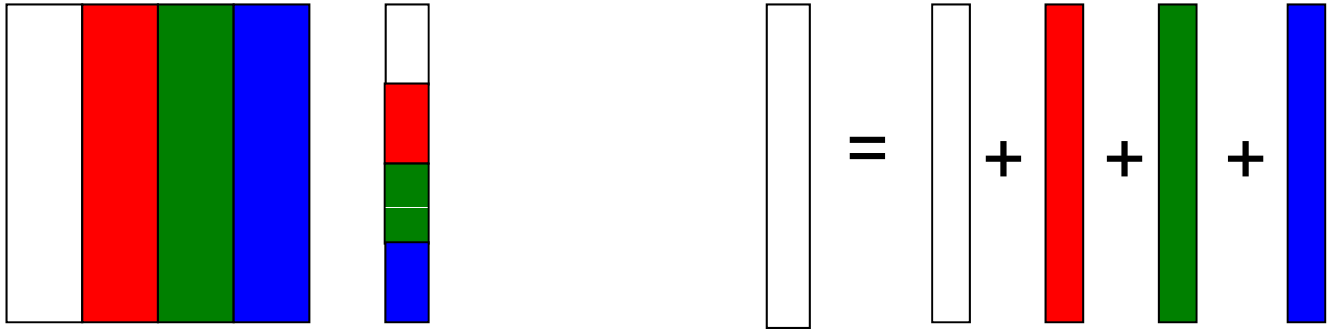
＜行方向分散方式＞　：無駄が多く使われない



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）

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）

プログラム実習Ⅰ(BLAS)（演習）

東京大学情報基盤センター　特任准教授　片桐孝洋

2010年9月27日（月）15:15－17:00

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＞というもののあるが、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、ソフトウエア開発効率が悪い
    - ＜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、数学ソフトウエアにおける＜ソフトウエア工学＞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、計算機ハードウエアの演算性能
  - 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

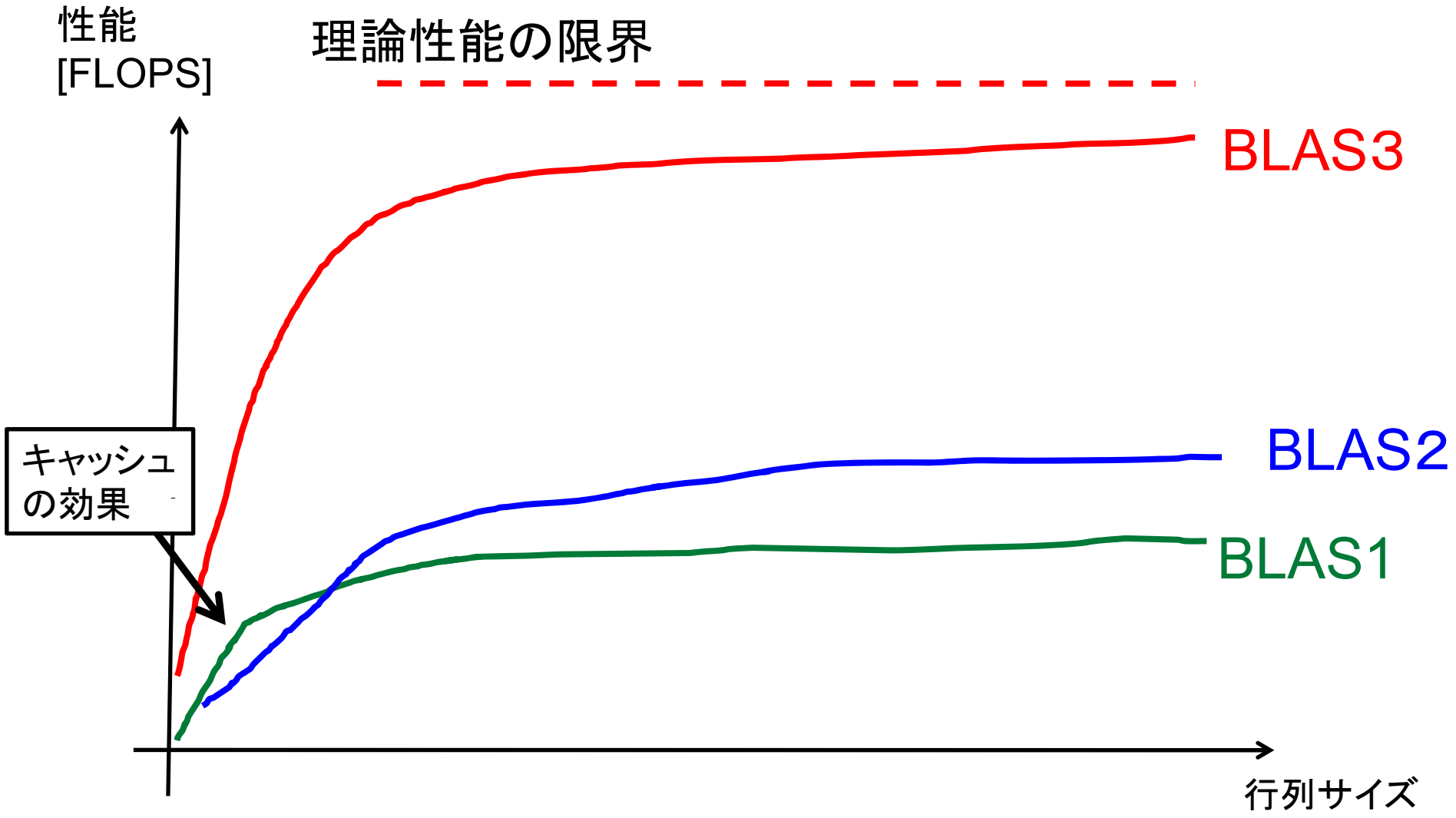
1.8　BLASとPBLAS

- 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、$T x = y$（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、再利用の効果がない。
  - 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



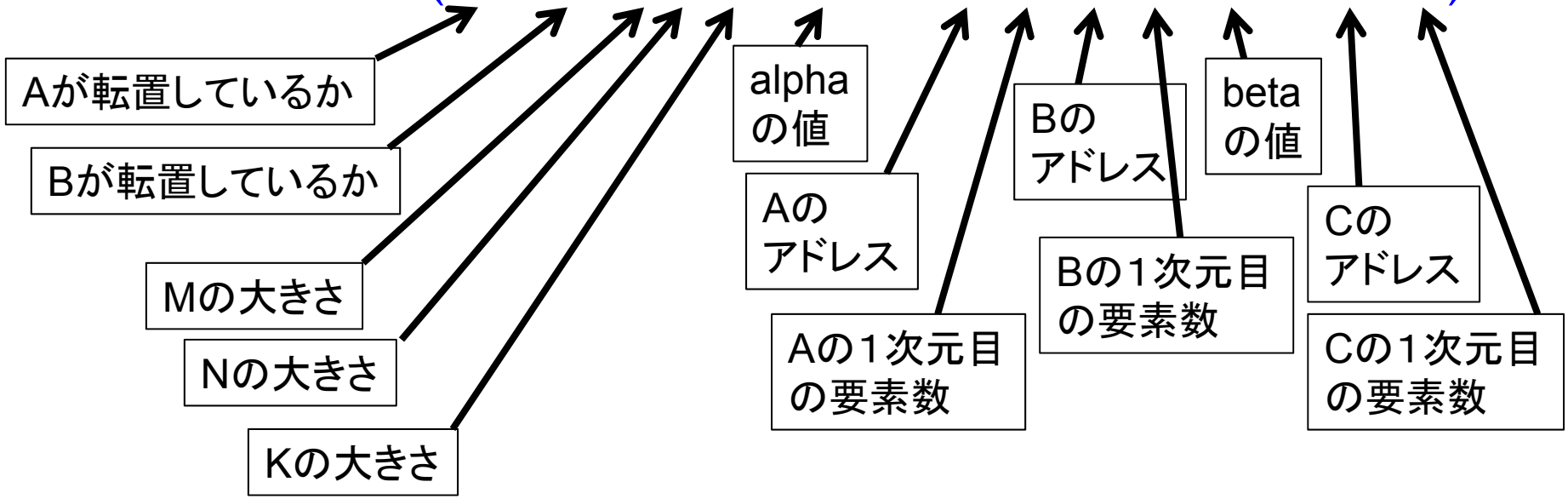
BLAS利用例

- 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:XYYYY
  - 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*op(A)*op(B) + beta*C　の計算をする
  - op(X) = X　もしくは　op(X) = X'（Xの転置行列）
- 引数
  - TRANSA（入力）- CHARACTER*1
    - TRANSA は op(A) の操作を指定する。以下の文字列を指定。
      - TRANSA = 'N' もしくは 'n', op(A) = A
      - TRANSA = 'T' もしくは 't', op(A) = A'
      - TRANSA = 'C' or 'c', op(A) = A'
  - TRANSB（入力）- CHARACTER*1
    - TRANSB は 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×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×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×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×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 × 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 × n行列の演算結果
    (alpha*op(A)*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1BLAS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

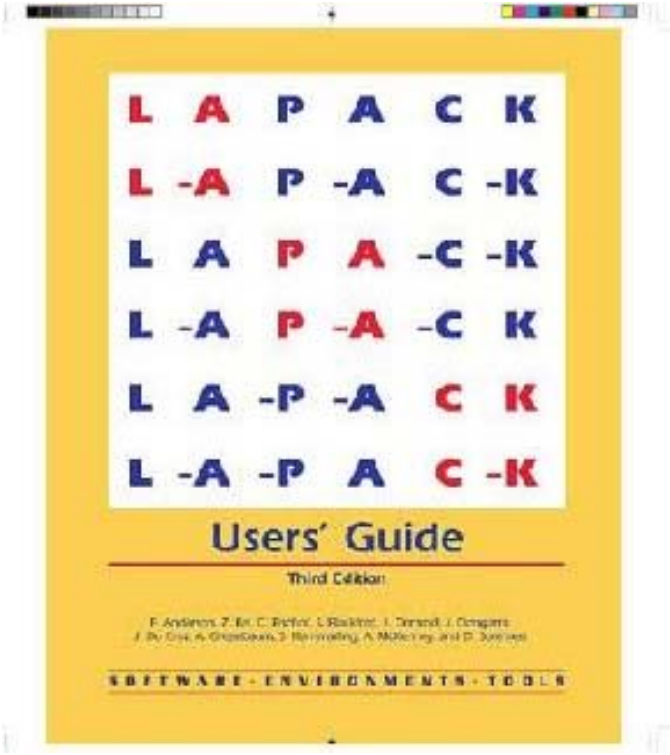
GOTO BLASとは

- 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エア上の実装に特化
    - 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* ×*N*行列で、*X* と *B* は *N* ×*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 >= 0。

インタフェース例：DGESV　（２／３）

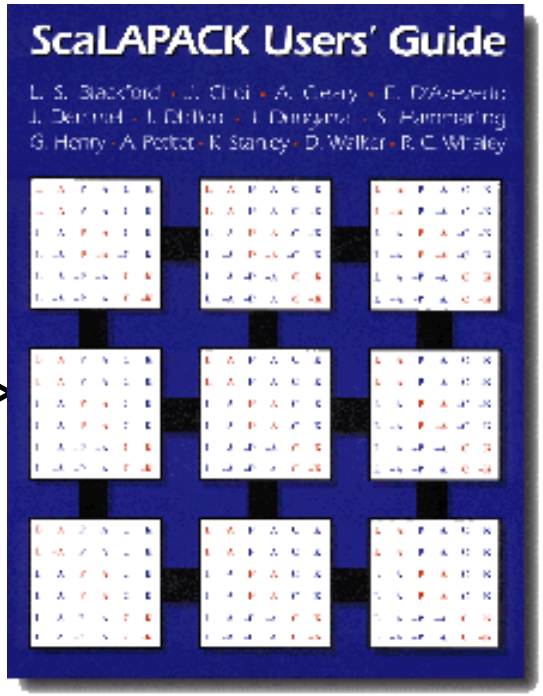
- NRHS (入力)　– INTEGER
  - 右辺ベクトルの数。行列Bの次元数。NRHS >= 0。
- A (入力／出力) – DOUBLE PRECISION, DIMENSION(:,:)
  - 入力時は、N×Nの行列Aの係数を入れる。
  - 出力時は、Aから分解された行列LとU　= P*L*Uを圧縮して出力する。Lの対角要素は1であるので、収納されていない。
- LDA (入力) – INTEGER
  - 配列Aの最初の次元の大きさ。LDA　>= max(1,N)。
- IPIVOT (出力)　– DOUBLE PRECISION, DIMENSION(:)
  - 交換行列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×NRHS　行列Bを入れる。
  - 出力時は、もし、INFO = 0　なら、N×NRHS行列である解行列Xが戻る。
- LDB (入力)　ーINTEGER
  - 配列Bの最初の次元の大きさ。LDB >= max(1,N)。
- INFO (出力) ー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

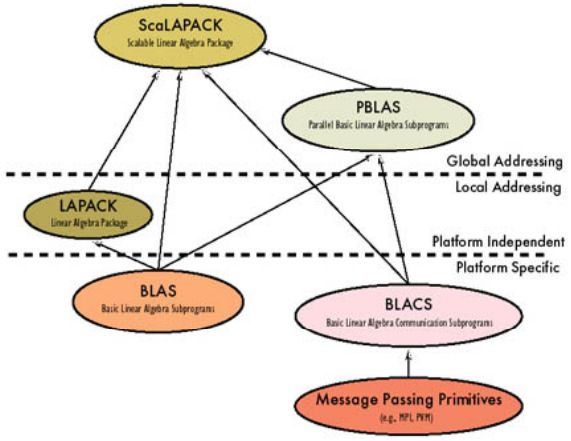
- 密行列に対する、連立一次方程式の解法、および固有値の解法の“標準”アルゴリズムルーチンの並列化版を無償で提供
- ユーザインタフェースはLAPACKに＜類似＞
- ソフトウエアの＜階層化＞がされている
  - 内部ルーチンはLAPACKを利用
  - 並列インタフェースはBLACS
- データ分散方式に、2次元ブロック・サイクリック分散方式を採用
- HP:　http://www.netlib.org/scalapack/



ScaLAPACKソフトウエア構成図

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/

The University of Tennessee　ornl Oak Ridge National Laboratory

（参照）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)
  - *sub(A) X = sub(B)*　の解の行列Xを計算をする
  - ここで sub(*A*)　は*N* ×*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* ×*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 >= 0。
- NRHS (大域入力)　– INTEGER
  - 右辺ベクトルの数。行列Bの次元数。NRHS >= 0。
- A (局所入力／出力) – DOUBLE PRECISION, DIMENSION(:,:)
  - 入力時は、N×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×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×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　（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、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、INFO = -iとなる。
  - > 0: もし INFO = Kのとき U(IA+K-1, 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。
  - ^x　goto-line：　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.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。

  - lecuter : 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）

- 以下のコマンドを実行する

```
$ cp /home/z30082/lecBLAS.tar ./
$ tar xvf lecBLAS.tar
$ cd Mat-Mat-BLAS
$ cd C
$ make
$
```

- 実行が終了したら、以下を実行する

```
$ cat mat-mat-blas.out
```

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  = 0.885865 [sec.]
 2257.680346 [MFLOPS]
OK!
```

BLAS DGEMMのサンプルプログラムの実行（Fortran言語版）

- 以下のコマンドを実行する

```
$ cp /home/z30082/lecBLAS.tar ./
$ tar xvf lecBLAS.tar
$ cd Mat-Mat-BLAS
$ cd F
$ make
$ qsub mat-mat-blas.bash
```

- 実行が終了したら、以下を実行する

```
$ cat mat-mat-blas.out
```

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.] =   0.8536694489999969
MFLOPS =    2342.8271883142970 2
 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×N行列AとBの行列積をおこない、Double型N×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)
```

GOTO BLAS呼び出しオプション

- 日立コンパイラから、GOTO BLASを呼び出す場合、以下のオプションを付けます。

●C言語

cc <プログラム名> -L/opt/itc/lib -lgoto

●Fortran言語

f90 <プログラム名> -L/opt/itc/lib -lgoto

※任意数のスレッド並列版（ノード内）を利用する場合は -lgoto_smp
16スレッド限定版（ノード内）を利用する場合は -lgoto_smp16
を、-lgotoの代わりに付けてください

サンプルプログラムの説（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。
  - GOTO BLAS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

演習課題１（BLAS）

1. MyMatMat関数（手続き）の行列積コードを、BLAS DEGMM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（Fotran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　―高速化と性能向上のために―、
   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　特任准教授　片桐孝洋

2010年9月28日（火）10:30－12:30

演習内容

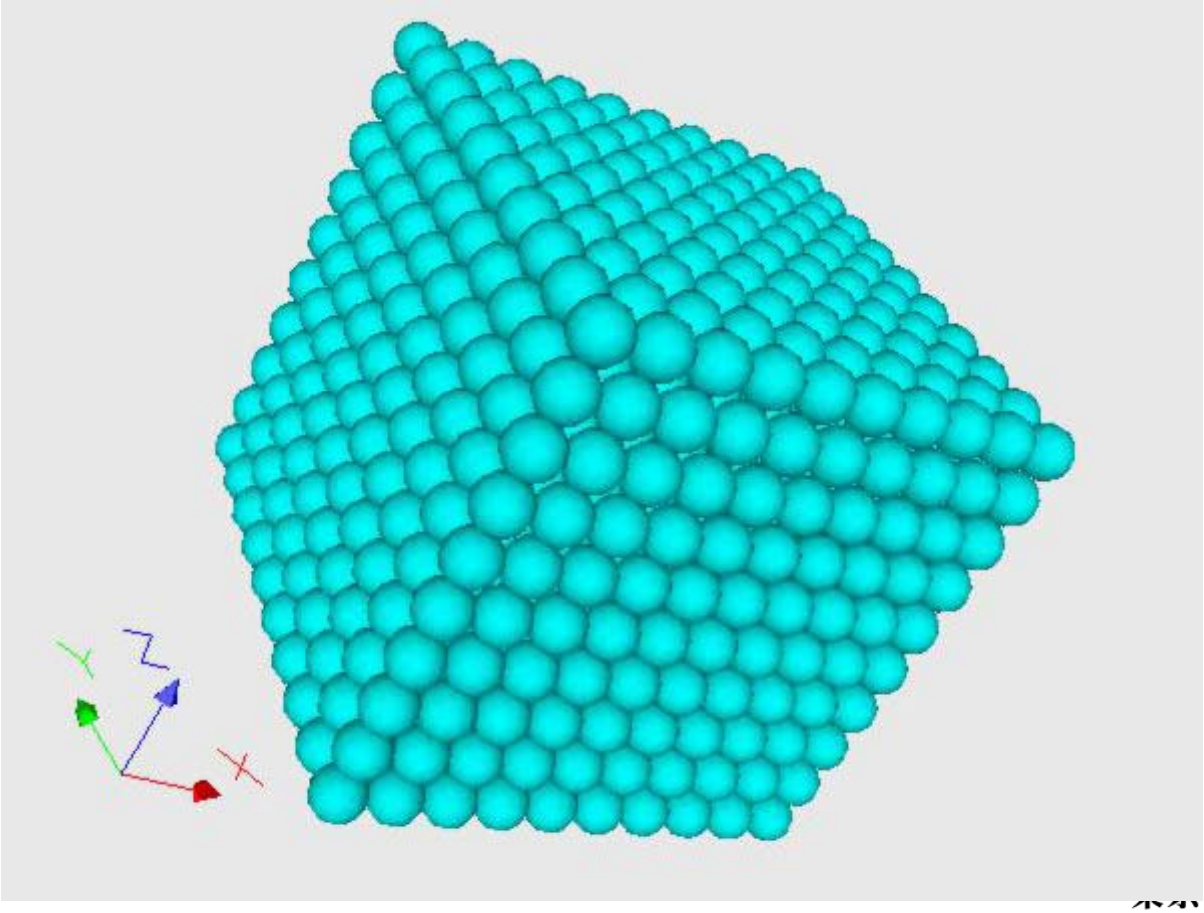
- 粒子間熱伝導問題の説明（座学）
- LAPACKの利用法と演習
  （C言語、Fortran言語）
- ScaLAPACKの利用法と演習
  （Fortran言語のみ）

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

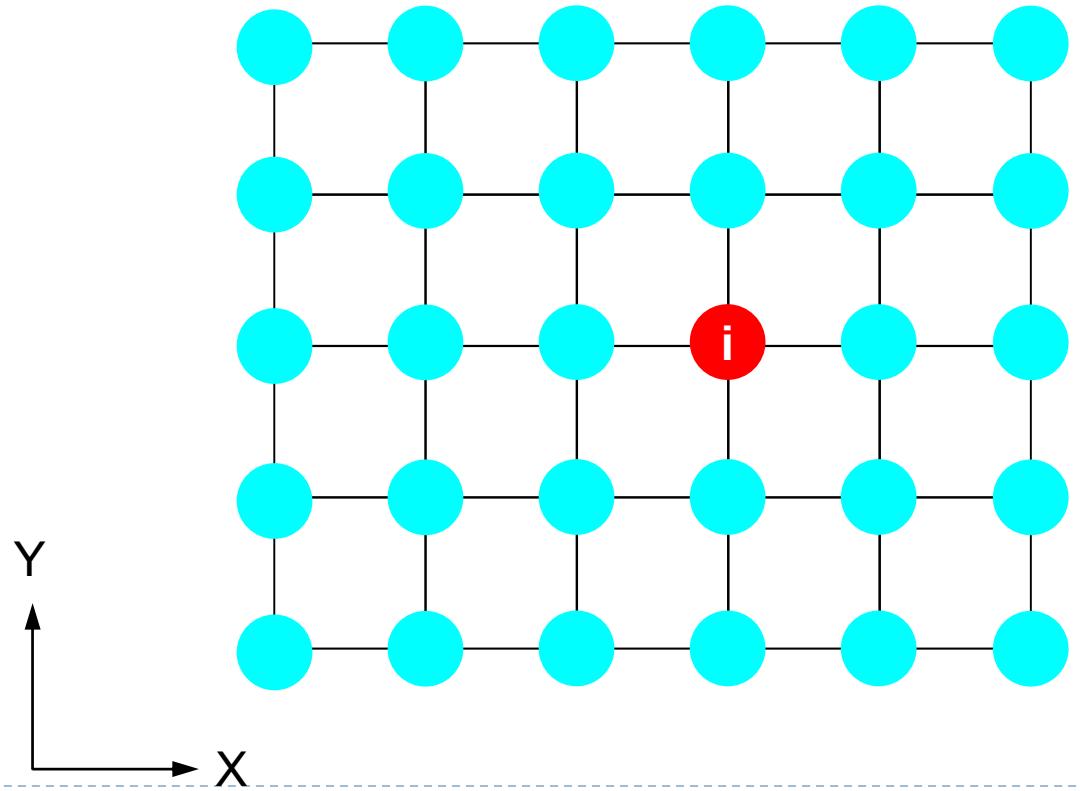
問題設定（1/5）

- 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。



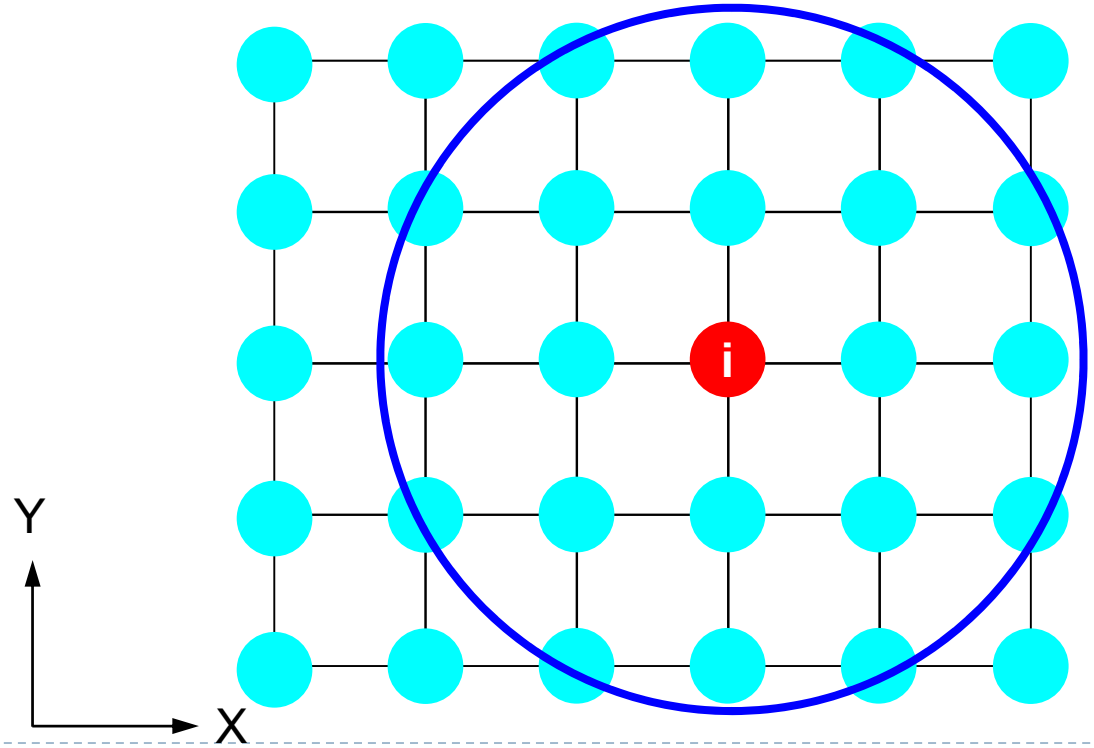
問題設定（2/5）
粒子iに関する熱流束の釣り合い

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



問題設定（3/5）
粒子間熱伝導

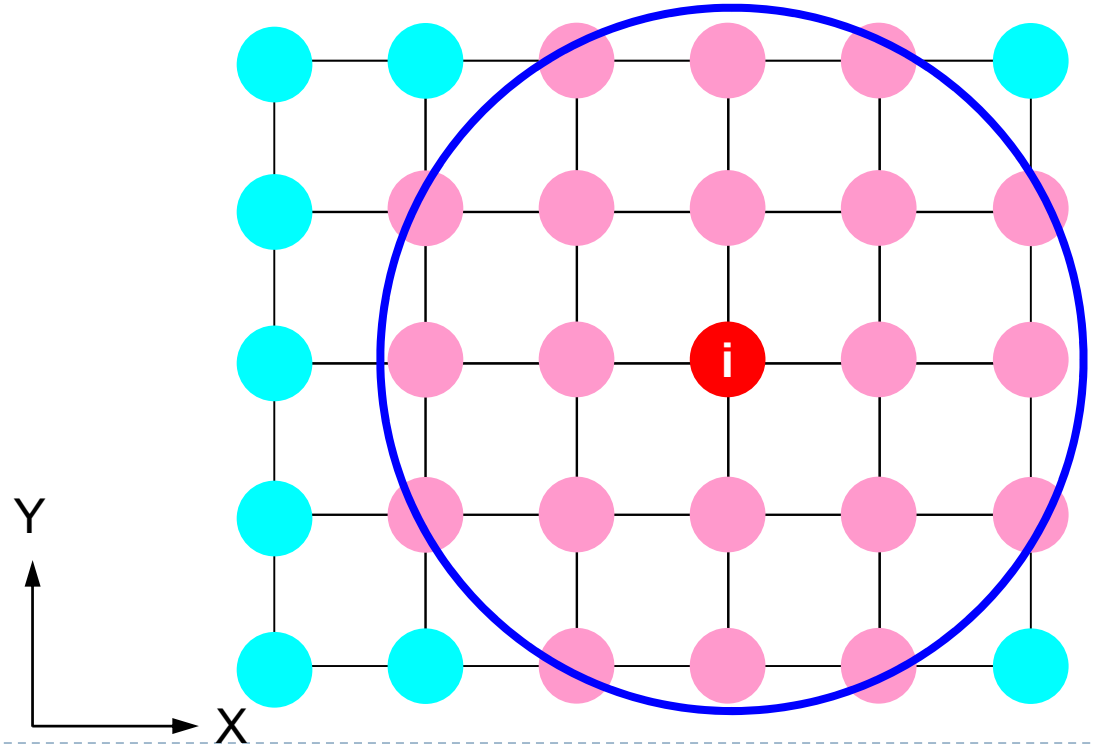
$$\sum_{i}^{DEL<RADI_{\max}} \frac{AREA}{DEL_{ij}/COND_{ij}}\left(T_j - T_i\right) + HCONV_i \cdot SURF \cdot \left(T_0 - T_i\right) + QVOL \cdot \overline{V} = 0$$



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

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

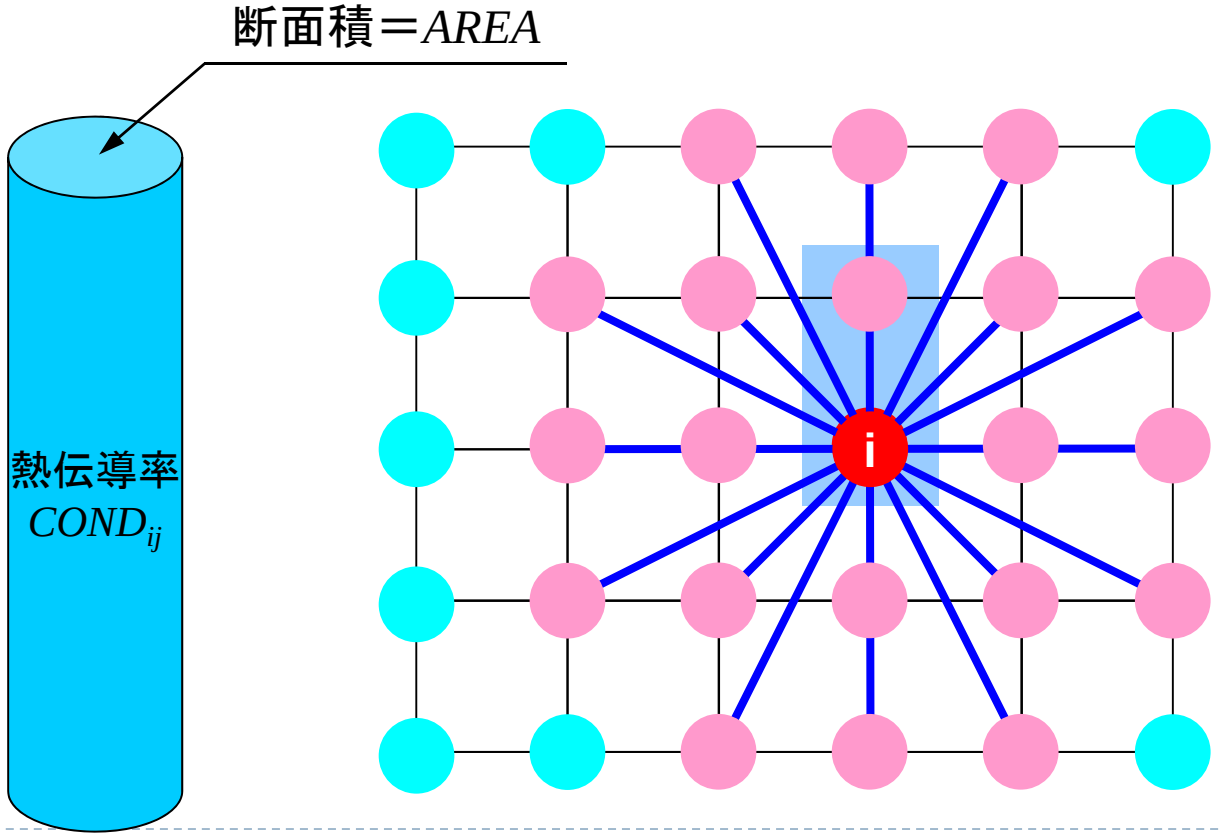


粒子iを中心とする，
半径$RADI_{max}$の球（円）

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

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



粒子iを中心とする，
半径$RADI_{max}$の球（円）

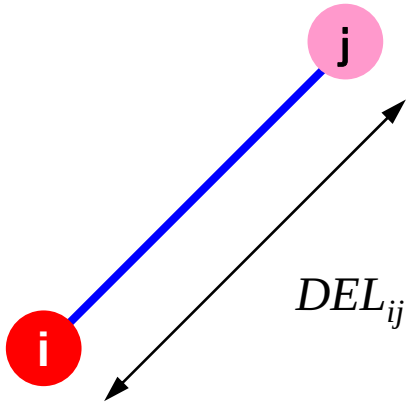
粒子間の距離DEL_{ij}が
$RADI_{max}$のよりも小さい
粒子群とのみ熱伝導が
ある。

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

熱伝導率：$COND_{ij}$

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

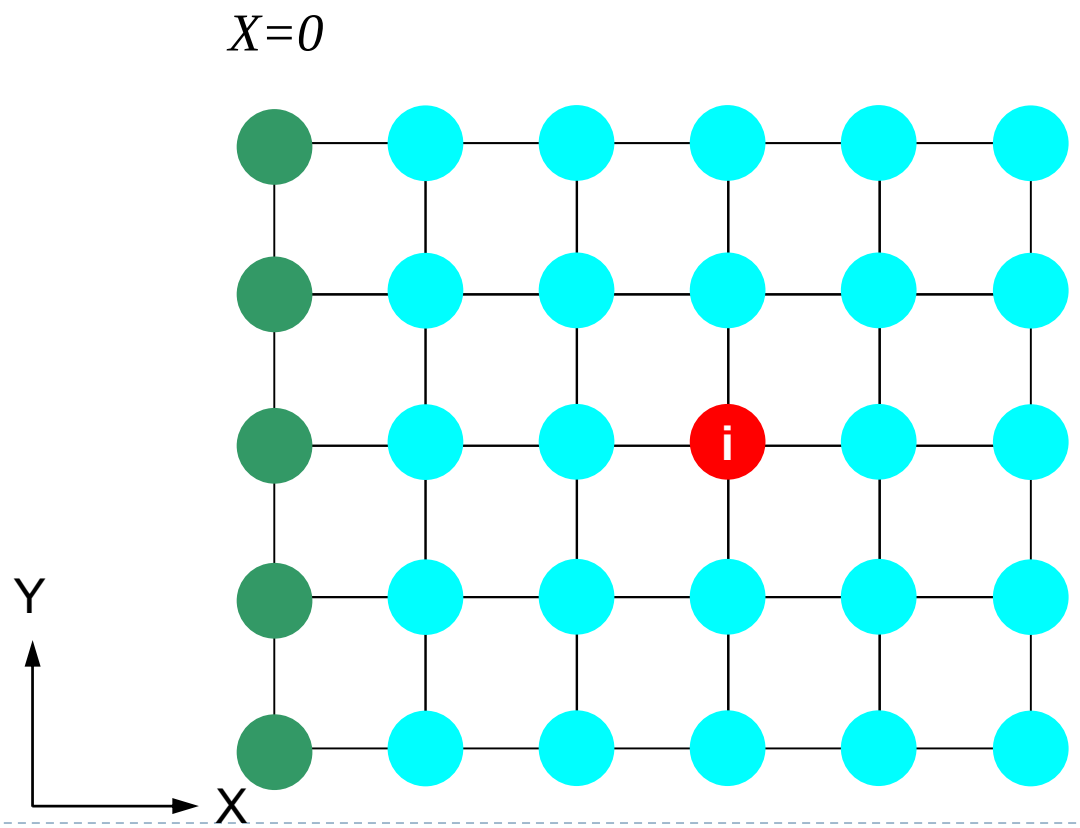
$$COND_{ij} = \frac{COND0}{\min\left(10^{DEL}, 10^{20}\right)}$$



COND0 「基準」熱伝導率

問題設定（4/5）
対流熱伝達

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



対流熱伝達率
$HCONV_i$
$= HCONV$ if $X=0$
$= 0$ otherwise

熱交換面積
$SURF$

雰囲気温度
T_0

問題設定（5/5）
体積発熱

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

体積発熱量　$QVOL$

発熱体積　$\overline{V}$

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

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

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

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

$$\sum_{j}^{DEL<RADI_{\max}} \left[\frac{AREA}{DEL_{ij}/COND_{ij}} T_j\right] - \left[\sum_{j}^{DEL<RADI_{\max}} \frac{AREA}{DEL_{ij}/COND_{ij}}\right] T_i$$

$$+ HCONV_i \cdot SURF \cdot T_0 - HCONV_i \cdot SURF \cdot T_i + QVOL \cdot \overline{V} = 0$$

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

AMAT(i,i)（対角成分）

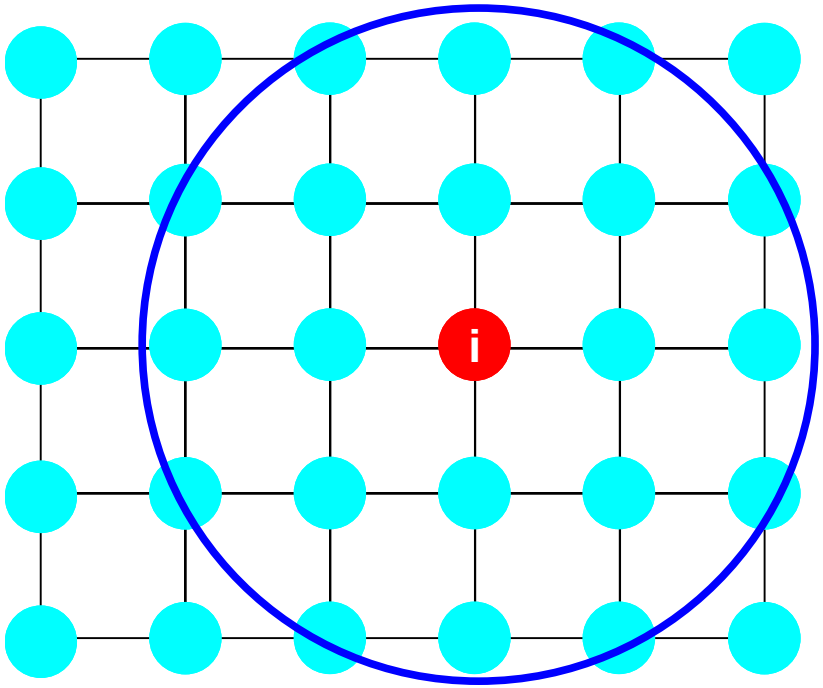
AMAT(i,j)（非対角成分）

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

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
```

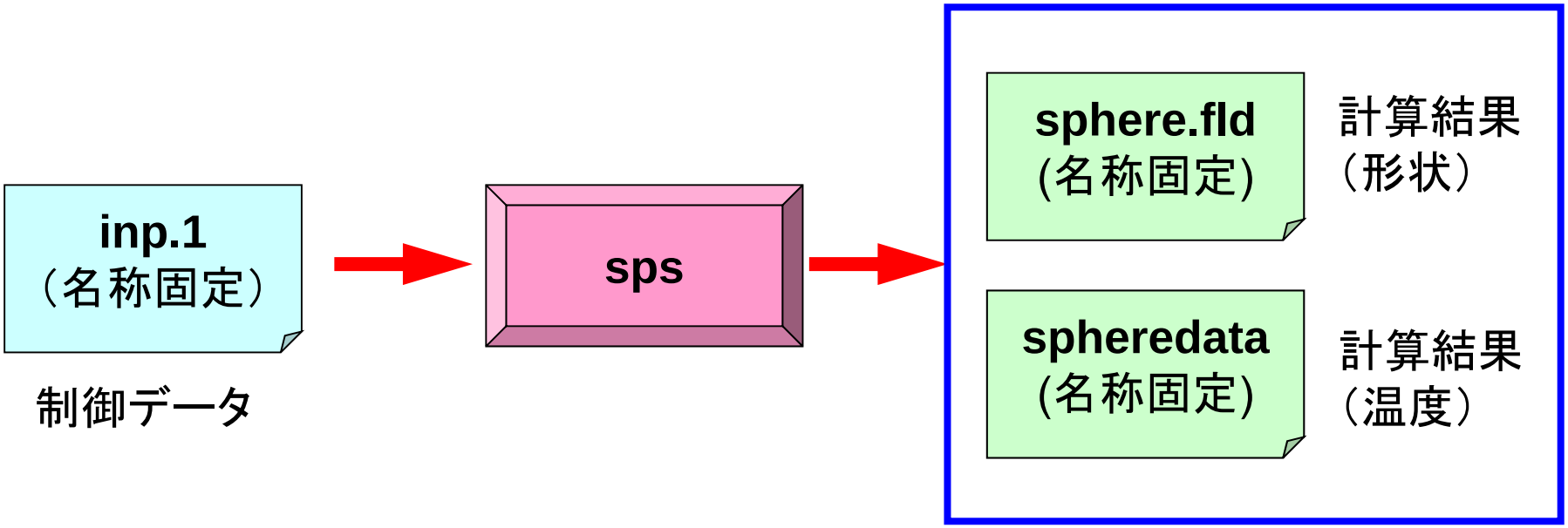
係数行列 AMAT

- 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


inp.1
（名称固定）
制御データ
sps
sphere.fld
(名称固定)
計算結果
（形状）
spheredata
(名称固定)
計算結果
（温度）

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, COND0
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                    必須
ndim=    3                         三次元モデルであることを示す
dim1=   10                         NX
dim2=   10                         NY
dim3=   10                         NZ
nspace=    3
veclen=    1
data= float
field= uniform
label= temperature
variable 1 file=./spheredata filetype=ascii   温度ファイル名
```

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
```

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

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

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
```

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

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===
```

$$\left[-\sum_{j}^{DEL<RADI_{\max}} \frac{AREA}{DEL_{ij}/COND_{ij}} - HCONV_i \cdot SURF \right] T_i +$$

$$\sum_{j}^{DEL<RADI_{\max}} \left[\frac{AREA}{DEL_{ij}/COND_{ij}} T_j \right]$$

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

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

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

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===
```

$$\left[-\sum_{j}^{DEL<RADI_{\max}} \frac{AREA}{DEL_{ij}/COND_{ij}} - HCONV_i \cdot SURF \right] T_i -$$

$$\sum_{j}^{DEL<RADI_{\max}} \left[\frac{AREA}{DEL_{ij}/COND_{ij}} T_j \right]$$

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

前処理付き共役勾配法
Preconditioned Conjugate Gradient Method （CG）

```
Compute r(0)= b-[A]x(0)
for i= 1, 2, …
    solve [M]z(i-1)= r(i-1)
    ρi-1= r(i-1) z(i-1)
    if i=1
      p(1)= z(0)
     else
      βi-1= ρi-1/ρi-2
      p(i)= z(i-1) + βi-1 p(i-1)
    endif
    q(i)= [A]p(i)
    αi  = ρi-1/p(i)q(i)
    x(i)= x(i-1) + αip(i)
    r(i)= r(i-1) - αiq(i)
    check convergence |r|
end
```

前処理：対角スケーリング

$x^{(i)}$ ：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) ⇒ {r}
W(i,2)= W(i,Z) ⇒ {z}
W(i,2)= W(i,Q) ⇒ {q}
W(i,3)= W(i,P) ⇒ {p}
W(i,4)= W(i,DD) ⇒ 1/DIAG

Compute $r^{(0)}= b-[A]x^{(0)}$
for i= 1, 2, …
  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, …
  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
```

Compute $r^{(0)}= b-[A]x^{(0)}$
for i= 1, 2, …
  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, …
  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, 1pe16.6)

        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, …
  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.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。

  - lecuter : 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+）

PGFコンパイラ (ACMLライブラリ利用のため)のコマンド

- C言語、Fortran言語ともに、PGFコンパイラを利用するため、以下のコマンドを実行してください。

```
source /opt/itc/mpi/mpiswitch.sh mpich-mx-pgi
```

●なお、サンプルプログラムのファイルに上記コマンドがあります。以下を入力してください。

$ cat pgf.sh

LAPACK dgesvのサンプルプログラムの実行（C言語版）

- 以下のコマンドを実行する

```
$ cp /home/z30082/lecLAPACK.tar ./
$ tar xvf lecLAPACK.tar
$ cd sphere-LAPACK
$ cd C
$ make
$ qsub go.sh
```

- 実行が終了したら、以下を実行する

```
$ cat test.out
```

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  = 0.000000 [sec.]
    inf [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0秒、
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）

- 以下のコマンドを実行する

```
$ cp /home/z30082/lecLAPACK.tar ./
$ tar xvf lecLAPACK.tar
$ cd sphere-LAPACK
$ cd F
$ make
$ qsub go.sh
```

- 実行が終了したら、以下を実行する

```
$ cat test.out
```

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] =   9.999999999999995E-007
 MFLOPS =    670168000.0000000
    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 =    627.42351139142 24
```

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（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

ACML LAPACK呼び出しオプション

- 以下のオプションを付けます。

●C言語

pgcc <プログラム名>

-lacml -lpgftnrtl -pgf90libs

※C言語からFortranのACMLライブラリが提供するLAPACK手続きを呼び出すため、f90ランタイムライブラリの指定が必要です

●Fortran言語

pgf95 <プログラム名> -lacml

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。

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（Fotran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

  lecLAPACK.tar

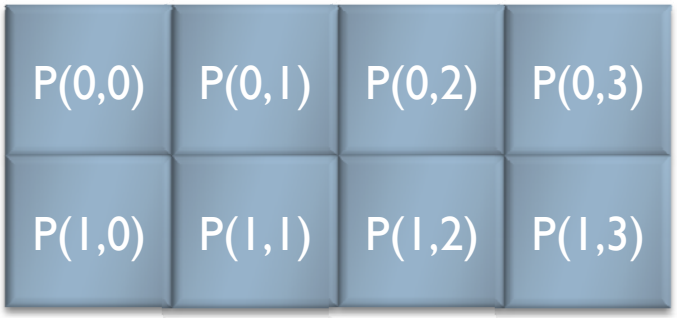
  ※ScaLAPACKはC言語版のサンプルはありません

- 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。
  - lecuter : 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+）

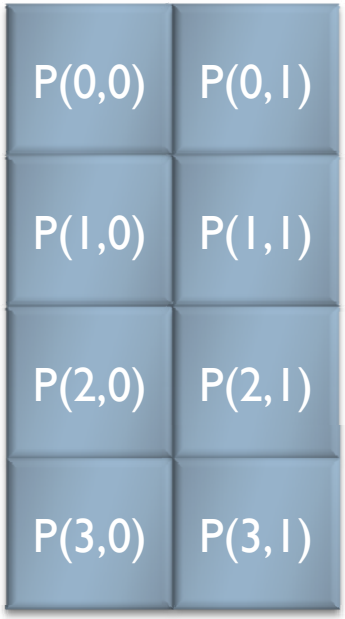
ScaLAPACKにおけるプロセッサー・グリッド

- MPIで起動する総プロセス数は、プロセッサー・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

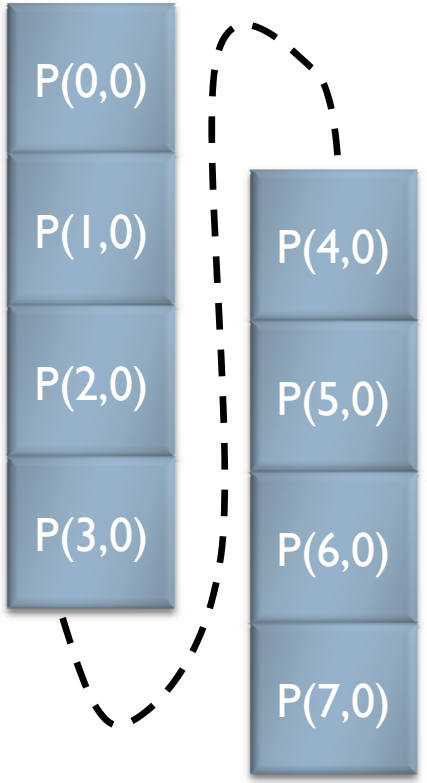
●２×４構成



●４×２構成

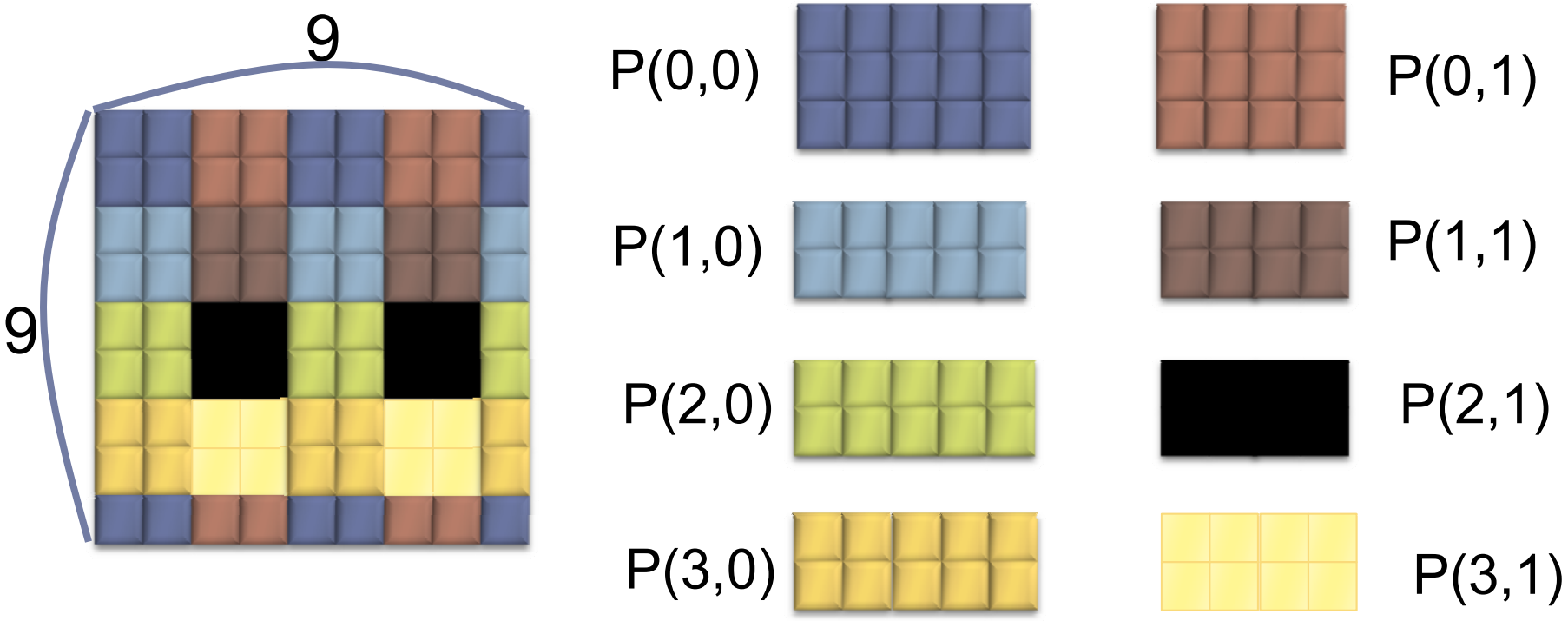


●８×１構成



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プロセッサー・グリッド）、MB=2



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
  - PDELSET関数を用いず
  - 局所ビューの観点で、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、行列の添え字を計算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）

- 以下のコマンドを実行する

```
$ cp /home/z30082/lecScaLAPACK.tar ./
$ tar xvf lecScaLAPACK.tar
$ cd sphere-ScaLAPACK
$ cd F
$ make
$ qsub go.sh
```

- 実行が終了したら、以下を実行する

```
$ cat test.out
```

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] = 0.30994415283203125E-005

MFLOPS = 351361040.384000003

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 = 627.42351139142249 4

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

ScaLAPACK呼び出しオプション

- 以下のオプションを付けます。

●Fortran言語

mpif90 <プログラム名> -noparallel -lscalapack_sc -llapack_sc -lblacsF77_sc -lblacsBASE_sc -lblacsC_sc -L/opt/itc/lib -lgoto

※要素並列化されていない（スレッド並列化されていない）LAPACKとBLASを用いたScaLAPACKの呼び出し手順です。
※BLASは、GOTO BLASを用います。
※MPI通信を用いる版です。

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

ScaLAPACK pdgesv回答

LAPCK 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