第61回　お試しアカウント付き 並列プログラミング講習会 「MPI基礎：並列プログラミング入門」 [Reedbush編]

東京大学情報基盤センター

内容に関するご質問は
hanawa @ cc.u-tokyo.ac.jp
まで、お願いします。

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 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。使う時は慎重に。*~ の間に空白が入ってしまうと、全てが消えます。
- 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）

UNIX備忘録その2

- less ファイル名： ファイル名の中身を見る(catでは画面がいっぱいになってしまうとき）
  - スペースキー：1画面スクロール
  - / ：文字列の箇所まで移動する。
  - q ： 終了（訳がわからなくなったとき）

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-rb.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

  /lustre/gt00/z30105 (/homeでないので注意)

並列版Helloプログラムをコンパイルしよう (1/2)

1. cdw コマンドを実行して Lustreファイルシステムに移動する
2. /lustre/gt00/z30105 にある Samples-rb.tar を自分のディレクトリにコピーする
   $ cp /lustre/gt00/z30105/Samples-rb.tar ./
3. Samples-rb.tar を展開する
   $ tar xvf Samples-rb.tar
4. Samples フォルダに入る
   $ cd Samples
5. C言語 : $ cd C
   Fortran90言語 : $ cd F
6. 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

Reedbush-U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
  - Reedbush-Uでは、以下に限定
    - 1ノード（36コア）（15分まで, 9/1以降は30分まで）
    - 4ノード（144コア）（5分まで、9/1以降は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
  - Reedbush-Uでは、最大128ノード（4,608コア）(6時間、9/1以降は24時間)

インタラクティブ実行のやり方

- コマンドラインで以下を入力
  - 1ノード実行用
    $ qsub -I -q u-interactive -l select=1 -l walltime=01:00 -W group_list=gt00
  - 4ノード実行用
    $ qsub -I -q u-interactive -l select=4 -l walltime=01:00 -W group_list=gt00

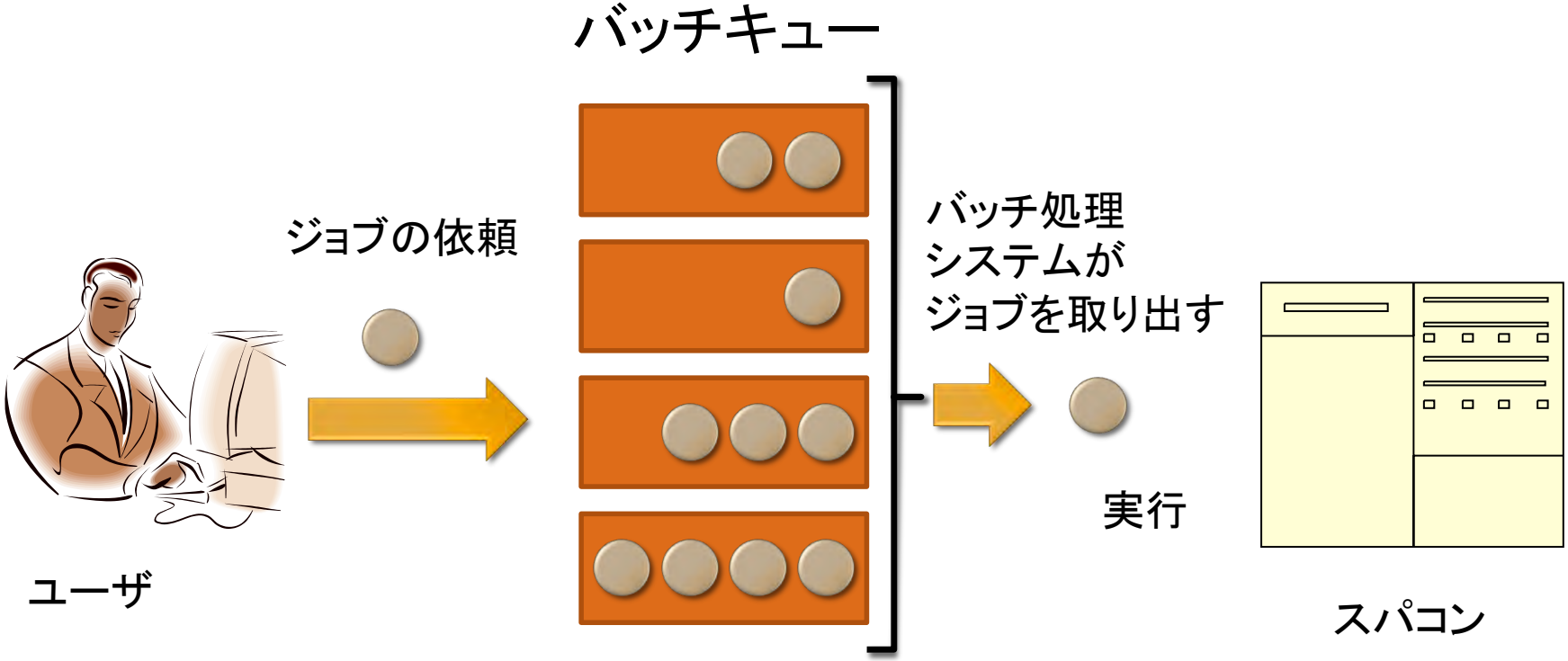
※インタラクティブ用のノードがすべて使われている場合、資源が空くまで、ログインできません。

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

- Reedbush-Uでは、コンパイラはバッチ実行、インタラクティブ実行で共通に使えます。
- 例）Intelコンパイラ
  - Cコンパイラ: icc, mpiicc (Intel MPIを使う場合）
  - Fortran90コンパイラ: ifort, mpiifort (Intel MPIを使う場合）

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、Altair社のバッチシステム PBS Professional 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:
    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: rbstat
  - 投入ジョブの削除: qdel <ジョブID>
  - バッチキューの状態を見る: rbstat --rsc
  - バッチキューの詳細構成を見る: rbstat –rsc -x
  - 投げられているジョブ数を見る: rbstat -b
  - 過去の投入履歴を見る: rbstat -H

本お試し講習会でのキュー名

- 本演習中のキュー名：
  - u-tutorial
  - 最大10分まで
  - 最大ノード数は8ノード(288コア) まで
- 本演習時間以外（24時間）のキュー名：
  - u-lecture
- 利用条件は演習中のキュー名と同様

rbstat --rsc の実行画面例

```
$ rbstat --rsc
QUEUE                    STATUS             NODE
u-debug                  [ENABLE ,START]      54
u-short                  [ENABLE ,START]      16
u-regular                [ENABLE ,START]
  |---- u-small          [ENABLE ,START]     288
  |---- u-medium         [ENABLE ,START]     288
  |---- u-large          [ENABLE ,START]     288
  |---- u-x-large        [ENABLE ,START]     288
u-interactive            [ENABLE ,START]
  |---- u-interactive_1  [ENABLE ,START]      54
  |---- u-interactive_4  [ENABLE ,START]      54
u-lecture                [ENABLE ,START]      54
u-lecture8               [DISABLE,START]      54
u-tutorial               [ENABLE ,START]      54
```

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
利用可能数

rbstat --rsc -x の実行画面例

```
$ rbstat --rsc -x
QUEUE                  STATUS          MIN_NODE MAX_NODE MAX_ELAPSE REMAIN_ELAPSE MEM(GB)/NODE  PROJECT
u-debug                [ENABLE ,START]        1       24   00:30:00      00:30:00         244GB  pz0105,gcXX
u-short                [ENABLE ,START]        1        8   02:00:00      02:00:00         244GB  pz0105,gcXX
u-regular              [ENABLE ,START]
  |---- u-small        [ENABLE ,START]        4       16   12:00:00      12:00:00         244GB  gcXX,pz0105
  |---- u-medium       [ENABLE ,START]       17       32   12:00:00      12:00:00         244GB  gcXX
  |---- u-large        [ENABLE ,START]       33       64   12:00:00      12:00:00         244GB  gcXX
  |---- u-x-large      [ENABLE ,START]       65      128   06:00:00      06:00:00         244GB  gcXX
u-interactive          [ENABLE ,START]
  |---- u-interactive_1 [ENABLE ,START]       1        1   00:15:00      00:15:00         244GB  pz0105,gcXX
  |---- u-interactive_4 [ENABLE ,START]       2        4   00:05:00      00:05:00         244GB  pz0105,gcXX
u-lecture              [ENABLE ,START]        1        8   00:10:00      00:10:00         244GB  gt00,gtYY
u-lecture8             [DISABLE,START]        1        8   00:10:00      00:10:00         244GB  gtYY
u-tutorial             [ENABLE ,START]        1        8   00:10:00      00:10:00         244GB  gt00
```

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

rbstat --rsc -b の実行画面例

```
$ rbstat --rsc -b
QUEUE                 STATUS           TOTAL RUNNING QUEUED HOLD BEGUN WAIT EXIT TRANSIT NODE
u-debug               [ENABLE ,START]      1       1      0    0     0    0    0       0   54
u-short               [ENABLE ,START]      9       3      5    1     0    0    0       0   16
u-regular             [ENABLE ,START]
  |---- u-small        [ENABLE ,START]     38      10      6   22     0    0    0       0  288
  |---- u-medium       [ENABLE ,START]      2       2      0    0     0    0    0       0  288
  |---- u-large        [ENABLE ,START]      4       2      0    2     0    0    0       0  288
  |---- u-x-large      [ENABLE ,START]      1       0      1    0     0    0    0       0  288
u-interactive         [ENABLE ,START]
  |---- u-interactive_1 [ENABLE ,START]     0       0      0    0     0    0    0       0   54
  |---- u-interactive_4 [ENABLE ,START]     0       0      0    0     0    0    0       0   54
u-lecture             [ENABLE ,START]      0       0      0    0     0    0    0       0   54
u-lecture8            [DISABLE,START]      0       0      0    0     0    0    0       0   54
u-tutorial            [ENABLE ,START]      0       0      0    0     0    0    0       0   54
```

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

ノードの利用可能数

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
#PBS -q u-lecture
#PBS -Wgroup_list=gt00
#PBS -l select=8:mpiprocs=36
#PBS -l walltime=00:01:00
cd $PBS_O_WORKDIR
. /etc/profile.d/modules.sh

mpirun ./hello
```

リソースグループ名
:u-lecture

利用グループ名
:gt00

利用ノード数

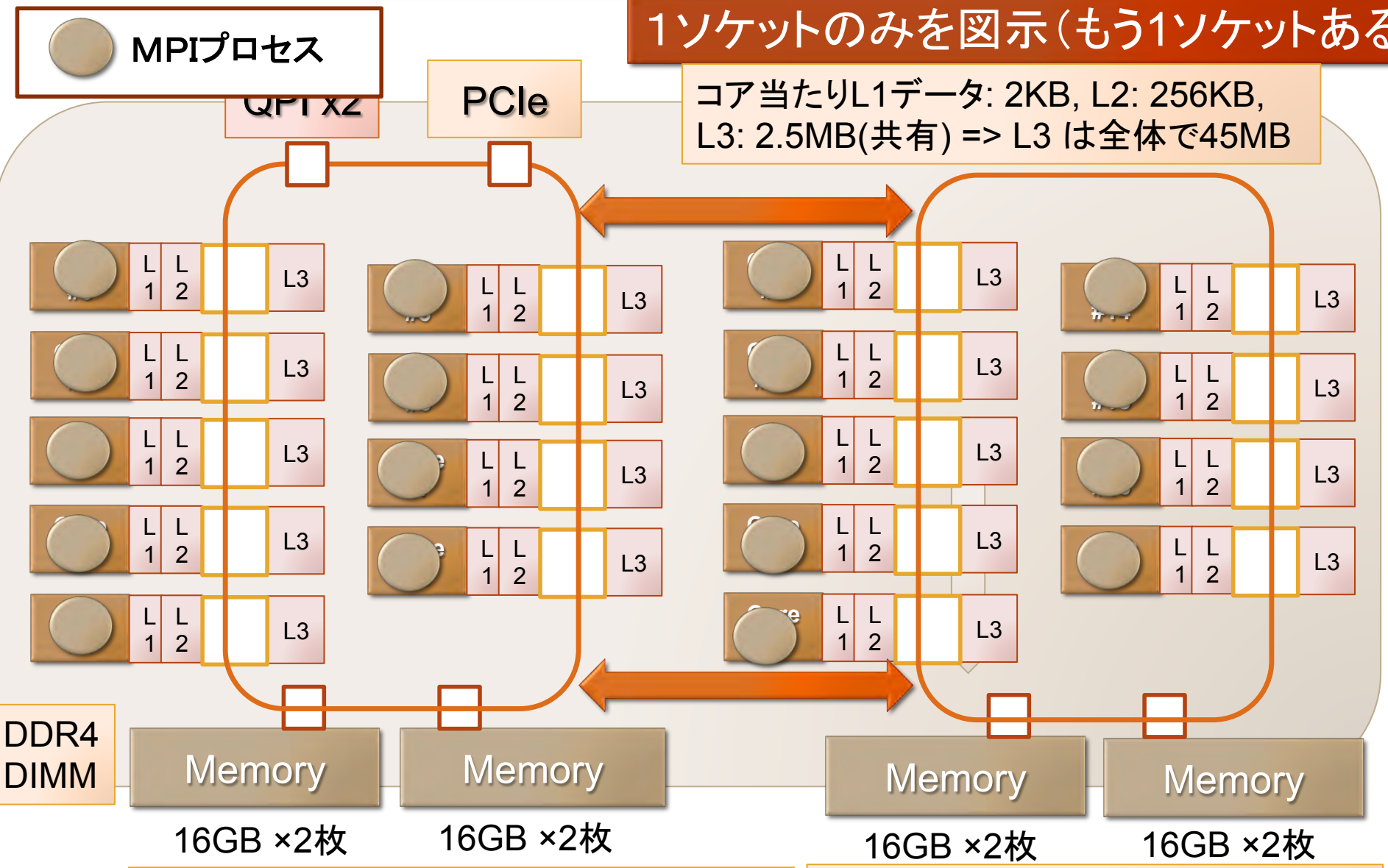
ノード内利用コア数
(MPIプロセス数)

実行時間制限
:1分

MPIジョブを8＊36 = 288 プロセスで実行する。

カレントディレクトリ設定、環境変数設定(必ず入れておく)

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”u-lecture”になっています
- $ emacs hello-pure.bash
  で、“u-lecture” → “u-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
   $ rbstat
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てみる
   $ cat hello-pure.bash.oXXXXXX
5. “Hello parallel world!”が、
   36プロセス*8ノード=288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.oXXXXX --- 標準出力ファイル
ジョブ名.eXXXXX --- 標準エラー出力ファイル
(XXXXX はジョブ投入時に表示されるジョブのジョブID)

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_hy36 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-hy36.bash）のキュー名を変更する。“u-lecture” → “u-tutorial”に変更する。
   $ emacs hello-hy36.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-hy36.bash
2. 自分の導入されたジョブを確認する
   $ rbstat
3. 実行が終了すると、以下のファイルが生成される
   hello-hy36.bash.eXXXXXX
   hello-hy36.bash.oXXXXXX （XXXXXXは数字）
4. 上記標準出力ファイルの中身を見てみる
   $ cat hello-hy36.bash.oXXXXXX
5. “Hello parallel world!”が、
   1プロセス*8ノード=8 個表示されていたら成功。

JOBスクリプトサンプルの説明(ピュアMPI)

(hello-hy36.bash, C言語、Fortran言語共通)

```
#!/bin/bash
#PBS -q u-lecture
#PBS -Wgroup_list=gt00
#PBS –l select=8:mpiprocs=1:
ompthreads=36
#PBS -l walltime=00:01:00
cd $PBS_O_WORKDIR
. /etc/profile.d/modules.sh
mpirun ./hello
```

リソースグループ名
:u-lecture

利用グループ名
:gt00

利用ノード数、
MPIプロセス数

ノード内利用スレッド数

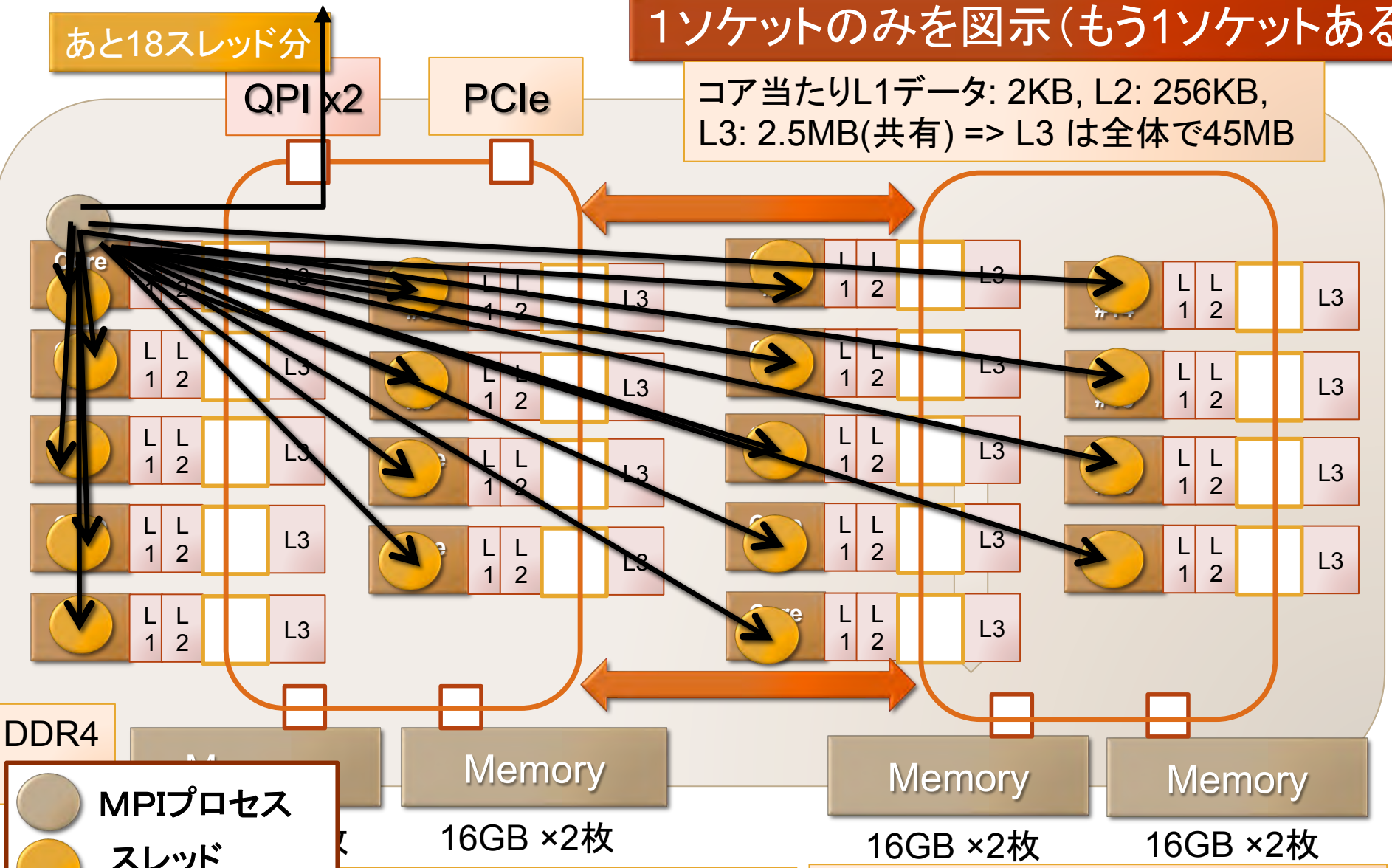
実行時間制限
:1分

MPIジョブを1*8 = 8 プロセスで実行する。

カレントディレクトリ設定、環境変数設定(必ず入れておく)

ハイブリッドMPIの実行状況（ノード内）

1ソケットのみを図示（もう1ソケットある）



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288、もしくは8)

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
288、もしくは8)

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

MPI実行時のリダイレクトについて

- Reedbush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す。
  - 例）mpirun ./a.out < in.txt > out.txt

依存関係のあるジョブの投げ方（ステップジョブ、チェーンジョブ）

- あるジョブスクリプト go1.sh　の後に、go2.sh を投げたい
- さらに、go2.shの後に、go3.shを投げたい、ということがある
- 以上を、ステップジョブまたはチェーンジョブという。
- Reedbushにおけるステップジョブの投げ方
  1. $qsub go1.sh
     12345.reedbush-pbsadmin0
  2. 上記のジョブ番号12345を覚えておき、以下の入力をする
     $qsub -W depend=afterok:12345 go2.sh
     12346.reedbush-pbsadmin0
  3. 以下同様
     $qsub -W depend=afterok:12346 go3.sh
     12347.reedbush-pbsadmin0

  afterok: 前のジョブが正常に終了したら実行する
  afternotok: 前のジョブが正常終了しなかった場合に実行する
  afterany: どのような状態でも実行する

性能プロファイラ

性能プロファイラ

- Webポータルから「利用の手引き」⇨
  「チューニングガイド　性能評価ツール編」
  を参照してください。

MPIプログラミング実習 Ⅱ（演習）

東京大学情報基盤センター　准教授　塙　敏博

講義の流れ

1. 行列-行列とは（30分）
2. 行列-行列積のサンプルプログラムの実行
3. サンプルプログラムの説明
4. 演習課題（1）：簡単なもの

サンプルプログラムの実行（行列-行列積）

行列-行列積のサンプルプログラムの注意点

- C言語版/Fortran言語版の共通ファイル名
  Mat-Mat-rb.tar
- ジョブスクリプトファイルmat-mat.bash　中のキュー名を
  u-lecture から u-tutorial に変更してから
  qsub　してください。
  - u-lecture : 実習時間外のキュー
  - u-tutorial :実習時間内のキュー

行列-行列積のサンプルプログラムの実行

- 以下のコマンドを実行する
  $ cp /lustre/gt00/z30105/Mat-Mat-fx.tar ./
  $ tar xvf Mat-Mat-fx.tar
  $ cd Mat-Mat
- 以下のどちらかを実行
  $ cd C ：C言語を使う人
  $ cd F ：Fortran言語を使う人
- 以下共通
  $ make
  $ qsub mat-mat.bash
- 実行が終了したら、以下を実行する
  $ cat mat-mat.bash.oXXXXXX

行列-行列積のサンプルプログラムの実行（C言語）

- 以下のような結果が見えれば成功

N = 1000

Mat-Mat time = 0.100450 [sec.]

19910.395473 [MFLOPS]

OK!

1コアのみで、20GFLOPSの性能

行列-行列積のサンプルプログラムの実行（Fortran言語）

- 以下のような結果が見えれば成功

```
 NN  =         1000
Mat-Mat time[sec.] =   0.101383924484253
MFLOPS =    19726.9932597759
 OK!
```

1コアのみで、20GFLOPSの性能

サンプルプログラムの説明

- #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 0
  の「0」を「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Nの宣言は、以下のファイルにあります。
  mat-mat.inc
- 行列サイズ変数が、NNとなっています。
  integer NN
  parameter (NN=1000)

演習課題(1)

- MyMatMat関数を並列化してください。
  - #define N 288
  - #define DEBUG 1

  として、デバッグをしてください。
- 行列A、B、Cは、各PEで重複して、かつ全部(N×N)所有してよいです。

MPIプログラミング実習(Ⅲ)(演習)

実習(Ⅱ)が早く終わってしまった方のための演習です

講義の流れ

1. 行列-行列積(2)のサンプルプログラムの実行
2. サンプルプログラムの説明
3. 演習課題(2):ちょっと難しい完全分散版
4. 並列化のヒント

サンプルプログラムの実行（行列-行列積（2））

行列-行列積の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
  Mat-Mat-d-rb.tar
- ジョブスクリプトファイルmat-mat-d.bash 中のキュー名を
  u-lecture から u-tutorial に変更してから
  qsub してください。
  - u-lecture : 実習時間外のキュー
  - u-tutorial: 実習時間内のキュー

行列-行列積(2)のサンプルプログラムの実行

- 以下のコマンドを実行する
  $ cp /lustre/gt00/z30105/Mat-Mat-d-fx.tar ./
  $ tar xvf Mat-Mat-d-fx.tar
  $ cd Mat-Mat-d
- 以下のどちらかを実行
  $ cd C ：C言語を使う人
  $ cd F ：Fortran言語を使う人
- 以下共通
  $ make
  $ pjsub mat-mat-d.bash
- 実行が終了したら、以下を実行する
  $ cat mat-mat-d.bash.oXXXXXX

行列-行列積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  = 576
Mat-Mat time  = 0.000074 [sec.]
 5154623.644043 [MFLOPS]
 Error! in ( 0 , 2 )-th argument in PE 0
 Error! in ( 0 , 2 )-th argument in PE 61
 Error! in ( 0 , 2 )-th argument in PE 51
 Error! in ( 0 , 2 )-th argument in PE 59
 Error! in ( 0 , 2 )-th argument in PE 50
 Error! in ( 0 , 2 )-th argument in PE 58
....
```

並列化が完成していないのでエラーが出ます。ですが、これは正しい動作です

行列-行列積のサンプルプログラムの実行（Fortran言語）

- 以下のような結果が見えれば成功

```
NN  =         576
Mat-Mat time  =   6.604909896850586E-005
MFLOPS =    57866.9439862109
Error! in ( 1 , 3 )-th argument in PE  0
Error! in ( 1 , 3 )-th argument in PE  61
Error! in ( 1 , 3 )-th argument in PE  51
Error! in ( 1 , 3 )-th argument in PE  58
Error! in ( 1 , 3 )-th argument in PE  55
Error! in ( 1 , 3 )-th argument in PE  63
Error! in ( 1 , 3 )-th argument in PE  60
```

…

並列化が
完成して
いないので
エラーが出ます。
ですが、
これは正しい
動作です。

サンプルプログラムの説明

- #define N 576
  - 数字を変更すると、行列サイズが変更できます
- #define DEBUG 1
  - 「0」を「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型の行列A((N/NPROCS)×N行列)と B((N×(N/NPROCS)行列))の行列積をおこない、Double型の(N/NPROCS)×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Nの宣言は、以下のファイルにあります。
  mat-mat-d.inc
- 行列サイズ変数が、NNとなっています。
  integer NN
  parameter (NN=576)

演習課題(1)

- MyMatMat関数(手続き)を並列化してください。
  - デバック時は
    #define N 576
    としてください。

- 行列A、B、Cの初期配置(データ分散)を、十分に考慮してください。

おわり

お疲れさまでした