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

[Reedbush編]

第95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：科学技術計算の効率化入門」

スパコンへのログイン・
テストプログラム起動

東京大学情報基盤センター　特任准教授　伊田 明弘

スパコンへのログイン・
ファイル転送・基本コマンド

Reedbushへログイン

- ターミナルから、以下を入力する
  $ ssh reedbush-u.cc.u-tokyo.ac.jp –l tYYxxx
  　「-l」はハイフンと小文字のL、
  　「tYYxxx」は利用者番号（数字）
  “tYYxxx”は、利用者番号を入れる
- 接続するかと聞かれるので、 yes を入れる
- 鍵の設定時に入れた
  自分が決めたパスワード（パスフレーズ）
  を入れる
- 成功すると、ログインができる

Reedbushにおける注意

- ログインするとホームディレクトリ(/home/gt00/t001XX)にいます。
- /home ファイルシステムは容量が小さく、ログインに必要なファイルだけを置くための場所です。
  - /home に置いたファイルは計算ノードから参照できません。ジョブの実行もできません。
- 計算に必要なファイルは、/lustre ファイルシステムに移動(mv)させてください。

- ホームディレクトリ: /home/gt00/t001XX
  - cd コマンドで移動できます。
- Lustreディレクトリ: /lustre/gt00/t001XX
  - cdw コマンドで移動できます。

PCのファイルをReedbushに置く

- ターミナルから、以下を入力する

$ scp ./a.f90 tYYxxx@reedbush-u.cc.u-tokyo.ac.jp:

「tYYxxx」は利用者番号（数字）

“tYYxxx”は、利用者番号を入れる

- PCのカレントディレクトリにある”a.f90”を、Reedbush上のホームディレクトリに置く
- ディレクトリごと置くには、”-r”　を指定

$ scp -r ./SAMP tYYxxx@reedbush-u.cc.u-tokyo.ac.jp:

- PCのカレントディレクトリにあるSAMPフォルダを、その中身ごと、Reedbush上のホームディレクトリに置く

ReedbushのデータをPCに取り込む

- ターミナルから、以下を入力する

$ scp tYYxxx@reedbush-u.cc.u-tokyo.ac.jp:~/a.f90 ./

「tYYxxx」は利用者番号（数字）
“tYYxxx”は、利用者番号を入れる

- Reedbush上のホームディレクトリにある”a.f90”を、PCのカレントディレクトリに取ってくる
- ディレクトリごと取ってくるには、”-r” を指定

$ scp -r tYYxxx@reedbush-u.cc.u-tokyo.ac.jp:~/SAMP ./

- Reedbush上のホームディレクトリにあるSAMPフォルダを、
その中身ごと、PCのカレントディレクトリに取ってくる

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

- 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）
- cp ファイル名 フォルダ名: ファイルをコピーする
- mv ファイル名 フォルダ名:ファイルを移動させる

テストプログラムのコンパイルと実行 [ReedBush-U編]

サンプルプログラムのコンパイル

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/z30107 （/homeでないので注意）

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

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(Lustre作業用ディレクトリに移動する)を実行して Lustreファイルシステムに移動する
2. /lustre/gt00/z30107 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/z30107/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

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

サンプルプログラムの実行

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ノード利用（36コア，30分まで）
    - 4ノード利用（144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
  - Reedbush-Uでは、最大128ノード利用可能（4,608コア, 24時間まで）

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

- FX-10の場合とは異なり、Reedbush-Uでは、バッチ実行用とインタラクティブ実行用で、異なるコンパイラを使用する必要はありません。

- 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)

インタラクティブ実行の仕方（参考）

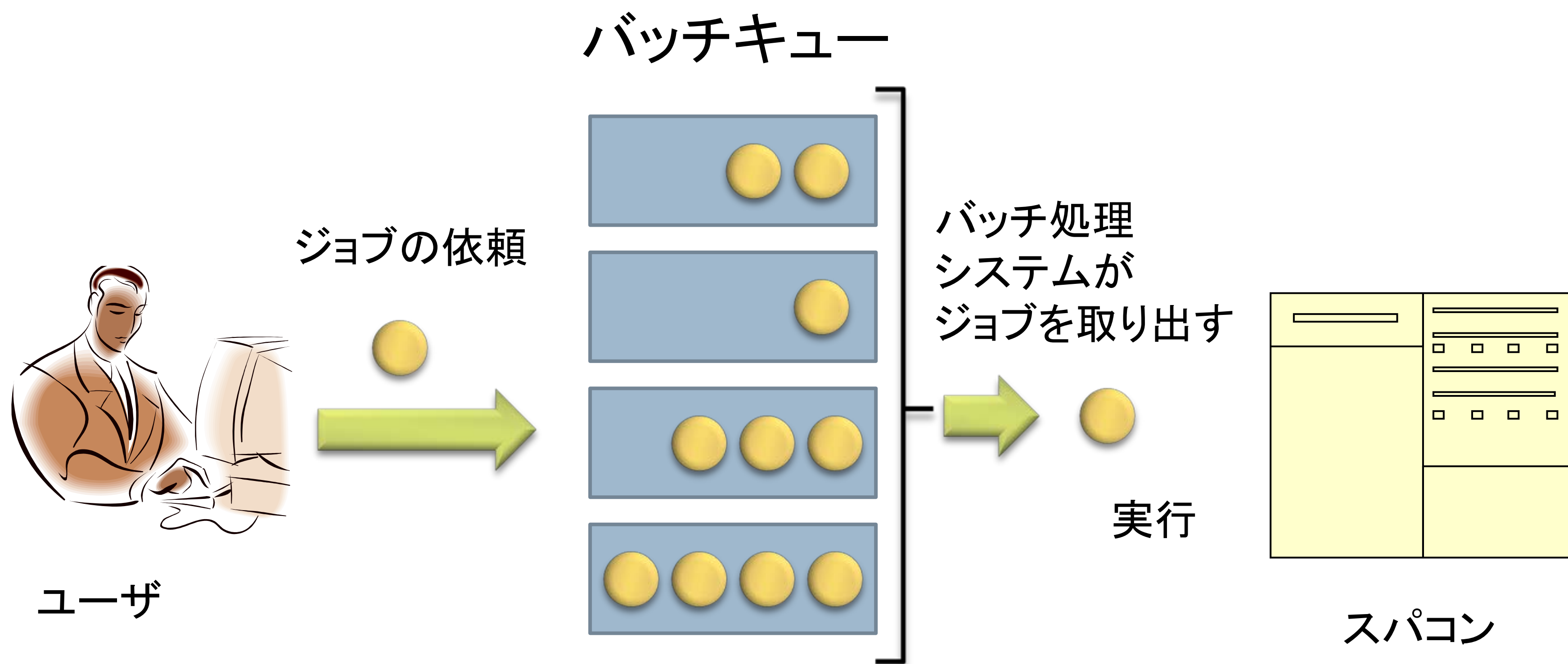
この講習会では使用できません

- 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。

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。



バッチ処理を用いたジョブの実行方法

- Reedbushシステムにおいてバッチ処理は、Altair社のバッチシステム PBS Professionalで管理されています。
- ジョブの投入:
  qsub <ジョブスクリプトファイル名>

```
#!/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
```

キュー名
:u-lecture

利用グループ名
:gt00

ジョブスクリプトファイルの例

本講習会でのグループ名とキュー名

- グループ: gt00

  課金情報(財布)を管理するのに使用される

- キュー 名　: u-tutorial

  キューとは、スパコンにバッチジョブを
  投入する時の待ち行列の名前
  (詳細は後述)

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

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

Reedbush-Uのバッチジョブキュー

■通常キューの一覧

代表キュー名	キュー名	最大ノード数	実行制限時間（経過時間）
u-debug	u-debug	1-24	30 min
u-short	u-short	1-8	4 h
u-regular	u-small u-medium u-large u-x-large	4-16 17-32 33-64 65-128	48 h 48 h 48 h 24 h

■講習会用の特別キュー

u-tutorial	**u-tutorial**	**1-8**	**10 min**
u-lecture	u-lecture	1-8	10 min

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

バッチ処理システムの使い方

- 主要コマンド（Reedbushの場合）
  - 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
  - 同時に投入できる数／実行できる数を見る： rbstat --limit

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
利用可能か

利用可能ノード数

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
利用可能か

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
（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
```

キュー名
：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）

INFORMATION TECHNOLOGY CENTER, THE UNIVERSITY OF TOKYO

並列版Helloプログラムを実行しよう

- 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

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プログラムの説明（C言語）

このプログラムは、全コアで起動される

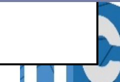
```
#include <stdio.h>
#include <mpi.h>

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
：各コアで値は異なる

全体のプロセッサ台数を取得
：各コアで値は同じ
（演習環境では288）

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、全コアで起動される

```
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
：各コアで値は異なる

全体のプロセッサ台数を取得
：各コアで値は同じ
（演習環境では288）

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

おわり

お疲れさまでした