

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

[Oakleaf-FX(FX-10)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

Oakleaf-FXへログイン

- ターミナルから、以下を入力する
`$ ssh oakleaf-fx.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

PCのファイルをOakleaf-FXに置く

- ターミナルから、以下を入力する

```
$ scp ./a.f90 tYYxxx@oakleaf-fx.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、Oakleaf-FX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には、“-r” を指定

```
$ scp -r ./SAMP tYYxxx@oakleaf-fx.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、Oakleaf-FX上のホームディレクトリに置く

Oakleaf-FX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@oakleaf-fx.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

- Oakleaf-FX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@oakleaf-fx.cc.u-tokyo.ac.jp:~/SAMP ./
```

- Oakleaf-FX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
 - ▶ $\hat{x} \hat{s}$ ($\hat{}$ はcontrol) : テキストの保存
 - ▶ $\hat{x} \hat{c}$: 終了
(\hat{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。)
 - ▶ \hat{g} : 訳がわからなくなったとき。
 - ▶ \hat{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。
 - ▶ \hat{y} : \hat{k} で消した行を、現在のカーソルの場所にコピーする。
 - ▶ \hat{s} 文字列 : 文字列の箇所まで移動する。
 - ▶ $\hat{M} x$ 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 [Oakleaf-FX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-fx.tar`
- ▶ tarで展開後、C言語とFortran90言語のディレクトリが作られる
 - ▶ `C/` : C言語用
 - ▶ `F/` : Fortran90言語用
- ▶ 上記のファイルが置いてある場所
`/home/z30107`

並列版Helloプログラムをコンパイルしよう (1/2)

1. `/home/z30107` にある `Samples-fx.tar` を
自分のディレクトリにコピーする
`$ cp /home/z30107/Samples-fx.tar ./`
2. `Samples-fx.tar` を展開する
`$ tar xvf Samples-fx.tar`
3. `Samples` フォルダに入る
`$ cd Samples`
4. C言語 : `$ cd C`
Fortran90言語 : `$ cd F`
5. `Hello` フォルダに入る
`$ cd Hello`

並列版Helloプログラムをコンパイルしよう (2/2)

6. ピュアMPI用のMakefileをコピーする

```
$ cp Makefile_pure Makefile
```

7. make する

```
$ make
```

8. 実行ファイル(hello)ができていることを確認
する

```
$ ls
```

サンプルプログラムの実行

FX10スーパーコンピュータシステムでの ジョブ実行形態

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

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

インタラクティブ実行の仕方（参考）

▶ コマンドラインで以下を入力

▶ 1ノード実行用

```
$ pjsub --interact
```

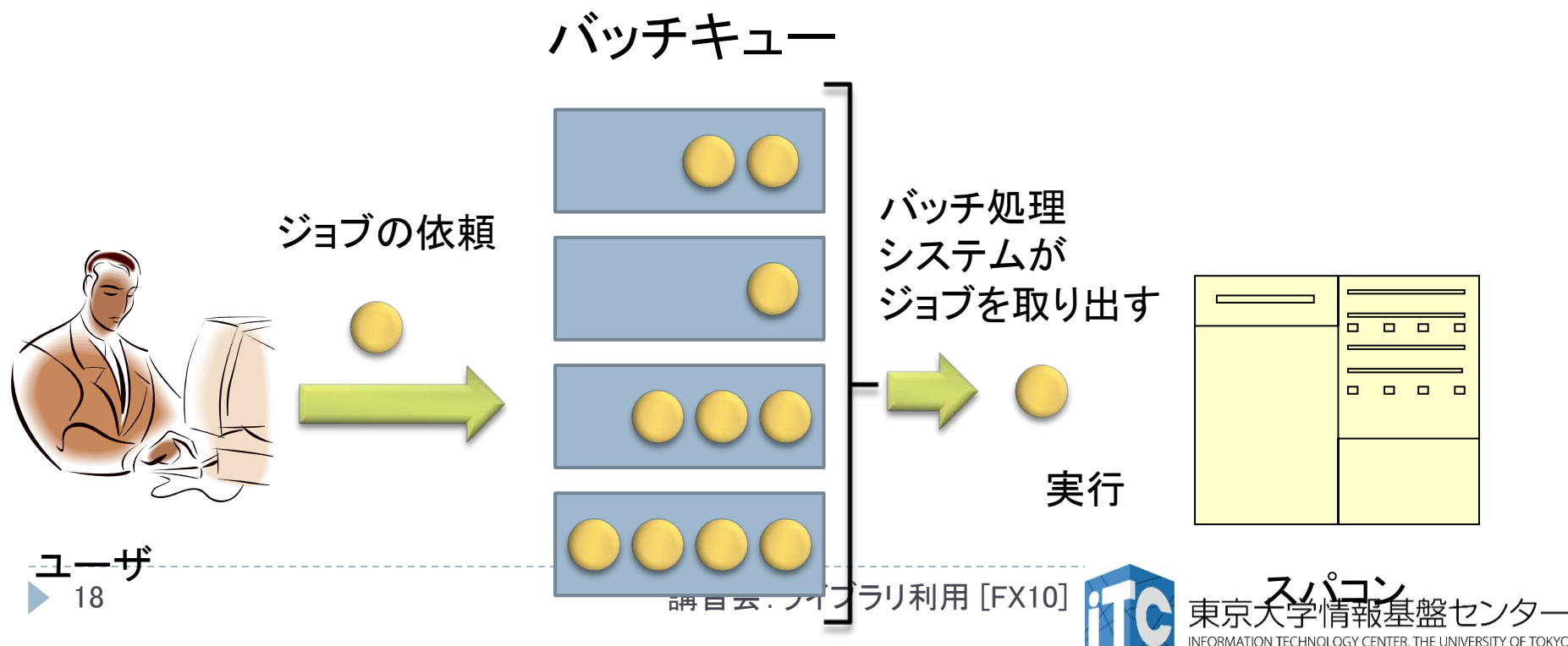
▶ 8ノード実行用

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

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

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。
 - ▶ キュー: 待ち行列



バッチ処理を用いたジョブの実行方法

- ▶ Oakleaf-FXにおいてバッチ処理は、富士通製のバッチシステムで管理されています。
- ▶ ジョブの投入：
`pjsub <ジョブスクリプトファイル名>`

```
#!/bin/bash
#PJM -L "rscgrp=lecture"
#PJM -L "node=12"
#PJM --mpi "proc=192"
#PJM -L "elapse=1:00"
mpirun ./hello
```

キュー名
:lecture

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 : 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)

本お試し講習会でのキュー名

▶ 本演習中のキュー名：

▶ tutorial

▶ 最大15分まで

▶ 最大ノード数は12ノード(192コア)まで

▶ 本演習時間以外(24時間)のキュー名：

▶ lecture

▶ 利用条件は演習中のキュー名と同様

Oakleaf-FX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)	ノード当り メモリー容量 (GB)
debug	debug	1-240	30 min	28
short	short	1-72	6 h	28
regular	small	12-216	48 h	28
	medium	217-372	48 h	28
	large	373-480	48 h	28
	x-large	481-1,440	24 h	28

■講習会用の特別キュー

tutorial	tutorial	1-12	15 min	28
lecture	lecture	1-12	15 min	28

バッチ処理システムの使い方

- ▶ **主要コマンド**(富士通製バッチ処理システムの場合)
 - ▶ ジョブの投入:
`pjsub <ジョブスクリプトファイル名> -g <プロジェクトコード>`
 - ▶ 自分が投入したジョブの状況確認: `pjstat`
 - ▶ 投入ジョブの削除: `pjdel <ジョブID>`
 - ▶ バッチキューの状態を見る: `pjstat --rsc`
 - ▶ バッチキューの詳細構成を見る: `pjstat -rsc -x`
 - ▶ 投げられているジョブ数を見る: `pjstat -b`
 - ▶ 過去の投入履歴を見る: `pjstat --history`
 - ▶ 同時に投入できる数／実行できる数を見る: `pjstat --limit`

pjstat --rsc の実行画面例

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

使える
キュー名
(リソース
グループ)

現在
使えるか

ノードの
物理構成情報

pjstat --rsc -x の実行画面例

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

使える
キュー名
(リソース
グループ)

現在
使えるか

ノードの
実行情報

課金情報(財布)
実習では1つのみ

pjstat -b の実行画面例

```
$ pjstat -b
```

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

使える
キュー名
(リソース
グループ)

現在
使えるか

ジョブ
の総数

実行して
いるジョブ
の数

待たされて
いるジョブ
の数

ノードの
物理構成
情報

JOBスクリプトサンプルの説明

(hello-pure.bash, C言語、Fortran言語共通)

```
#!/bin/bash
#PJM -L "rscgrp=lecture"
#PJM -L "node=12"
#PJM --mpi "proc=192"
#PJM -L "elapse=1:00"
#PJM -g gt00
mpirun ./hello
```

キュー名
:lecture

利用ノード数

利用コア数
(MPIプロセス数)

実行時間制限
:1分

利用グループ名
:gt00

MPIジョブを $16 * 12 = 192$ プロセスで実行する。

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

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

バッチジョブ実行による標準出力、標準エラー出力

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

ジョブ名.oXXXXXX --- 標準出力ファイル

ジョブ名.eXXXXXX --- 標準エラー出力ファイル

(XXXXXX はジョブ投入時に表示されるジョブのジョブID)

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192)

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192)

MPIの終了

時間計測方法 (C言語)

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

<ここに測定したいプログラムを書く>

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

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

バリア同期後
時間を習得し保存

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

時間計測方法 (Fortran言語)

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

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

<ここに測定したいプログラムを書く>

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

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

バリア同期後
時間を習得し保存

各プロセッサで、t0の値
は異なる。

この場合は、最も遅いもの
の値をプロセッサ0番
が受け取る

MPI実行時のリダイレクトについて

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

依存関係のあるジョブの投げ方 (ステップジョブ)

- ▶ 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**という。
- ▶ FX10におけるステップジョブの投げ方

1. `$pjsub --step go1.sh`

```
[INFO] PJM 0000 pjsub Job 800967_0 submitted.
```

2. 上記のジョブ番号800967を覚えておき、以下の入力をする

```
$pjsub --step --sparam jid=800967 go2.sh
```

```
[INFO] PJM 0000 pjsub Job 800967_1 submitted
```

3. 以下同様

```
$pjsub --step --sparam jid=800967 go3.sh
```

```
[INFO] PJM 0000 pjsub Job 800967_2 submitted
```

おわり

お疲れさまでした