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」 [FX10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-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/z30105

並列版Helloプログラムをコンパイルしよう (1/2)

1. /home/z30105 にある Samples-fx.tar を自分のディレクトリにコピーする
   $ cp /home/z30105/Samples-fx.tar ./
2. Samples-fx.tar を展開する
   $ tar xvf Samples-fx.tar
3. Samples フォルダに入る
   $ cd Samples
4. C言語 : $ cd C
   Fortran90言語 : $ cd F
5. Hello フォルダに入る
   $ cd Hello

並列版Helloプログラムをコンパイルしよう(2/2)

6. ピュアMPI用のMakefileをコピーする
   $ cp Makefile_pure Makefile
7. make する
   $ make
8. 実行ファイル(hello)ができていることを確認する
   $ ls

FX10スーパ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）(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

    $ pjsub --interact

  - 8ノード実行用

    $ pjsub --interact -L “node=8”

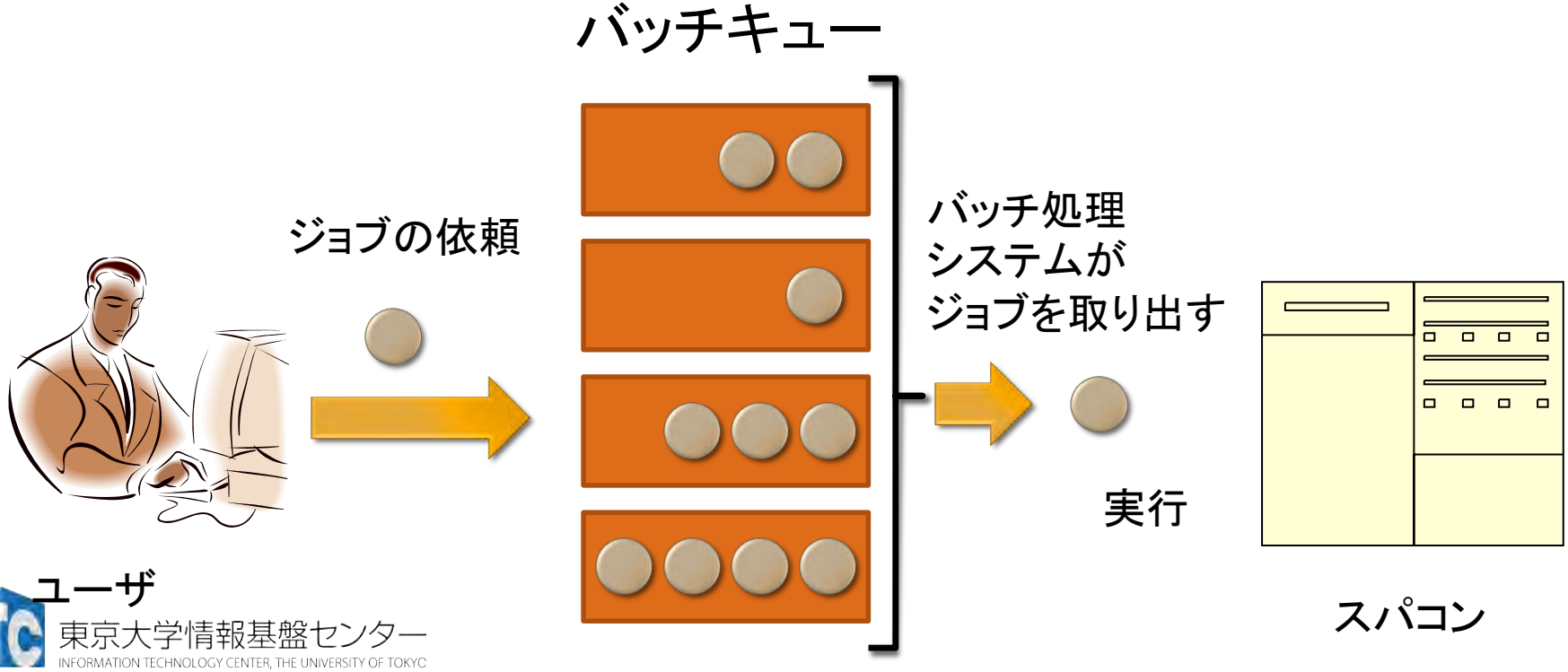
※インタラクティブ用のノード総数は50ノードです。
もしユーザにより50ノードすべて使われている場合、
資源が空くまで、ログインできません。

コンパイラの種類とインタラクティブ実行およびバッチ実行

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

バッチ処理とは

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



バッチキューの設定のしかた

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

本お試し講習会でのキュー名

- 本演習中のキュー名：
  - tutorial
  - 最大15分まで
  - 最大ノード数は12ノード(192コア) まで
- 本演習時間以外（24時間）のキュー名：
  - lecture
- 利用条件は演習中のキュー名と同様

pjstat --rsc の実行画面例

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

使える
キュー名
(リソース
グループ)

現在
使えるか

ノードの
物理構成情報

pjstat --rsc -x の実行画面例

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

使える
キュー名
(リソース
グループ)

現在
使えるか

ノードの
実行情報

課金情報（財布）
実習では1つのみ

pjstat -b の実行画面例

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

使えるキュー名(リソースグループ)

現在使えるか

ジョブの総数

実行しているジョブの数

待たされているジョブの数

ノードの物理構成情報

JOBスクリプトサンプルの説明（ピュアMPI）

（hello-pure.bash, C言語、Fortran言語共通）

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

リソースグループ名：lecture

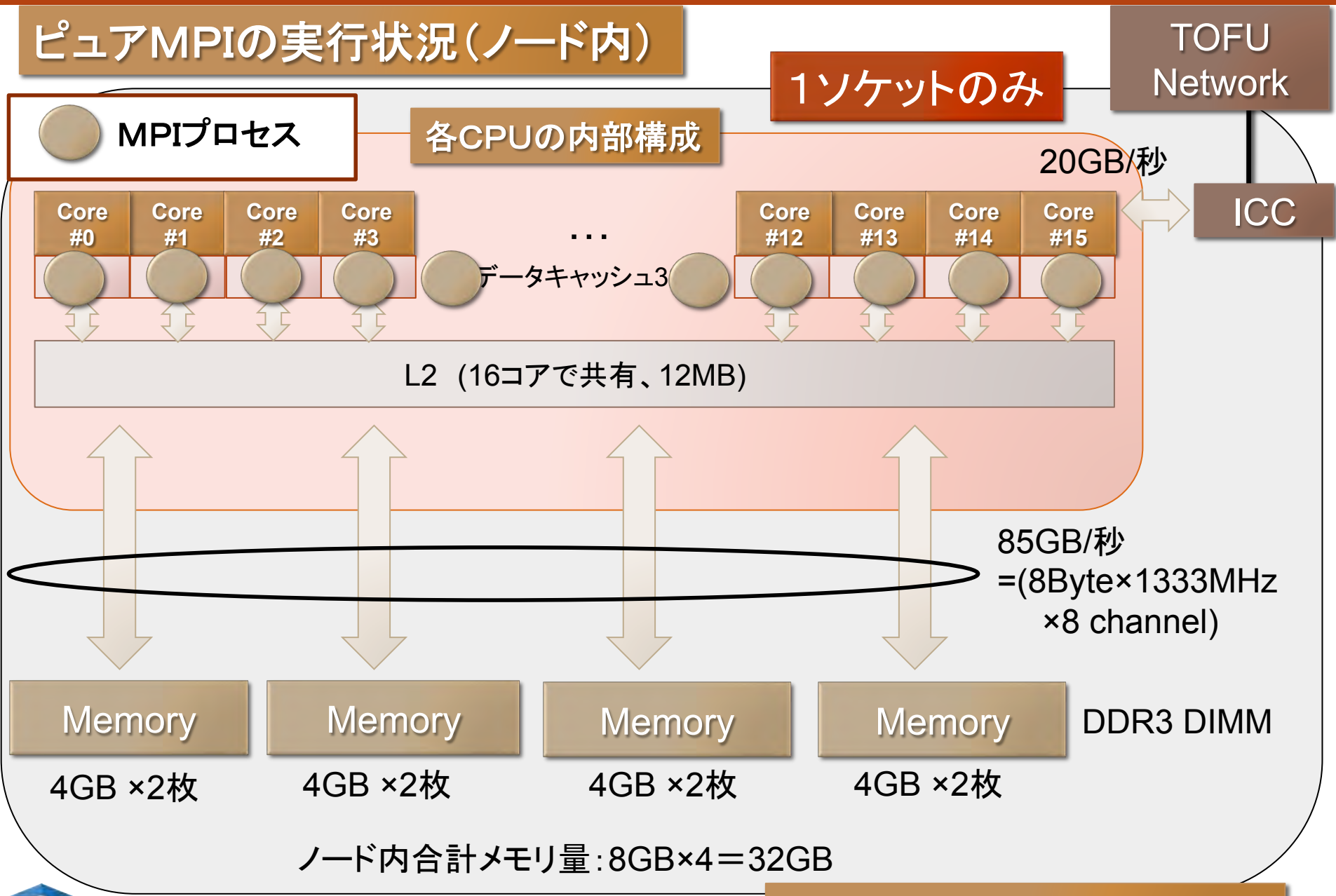
利用ノード数

利用コア数（MPIプロセス数）

実行時間制限：1分

MPIジョブを16＊12 = 192 プロセスで実行する。

ピュアMPIの実行状況（ノード内）



FX10計算ノードの構成

並列版Helloプログラムを実行しよう（ピュアMPI）

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

並列版Helloプログラムを実行しよう（ピュアMPI）

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

バッチジョブ実行による標準出力、標準エラー出力

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

ジョブ名.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_hy16 Makefile
2. make する。
   $ make clean
   $ make
3. 実行ファイル（hello）ができていることを確認する。
   $ ls
4. JOBスクリプト中（hello-hy16.bash）のキュー名を変更する。“lecture”　→　“tutorial”に変更する。
   $ emacs hello-hy16.bash

並列版Helloプログラムを実行しよう（ハイブリッドMPI）

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

JOBスクリプトサンプルの説明（ハイブリッドMPI）

（hello-hy16.bash, C言語、Fortran言語共通）

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

リソースグループ名：lecture

利用ノード数

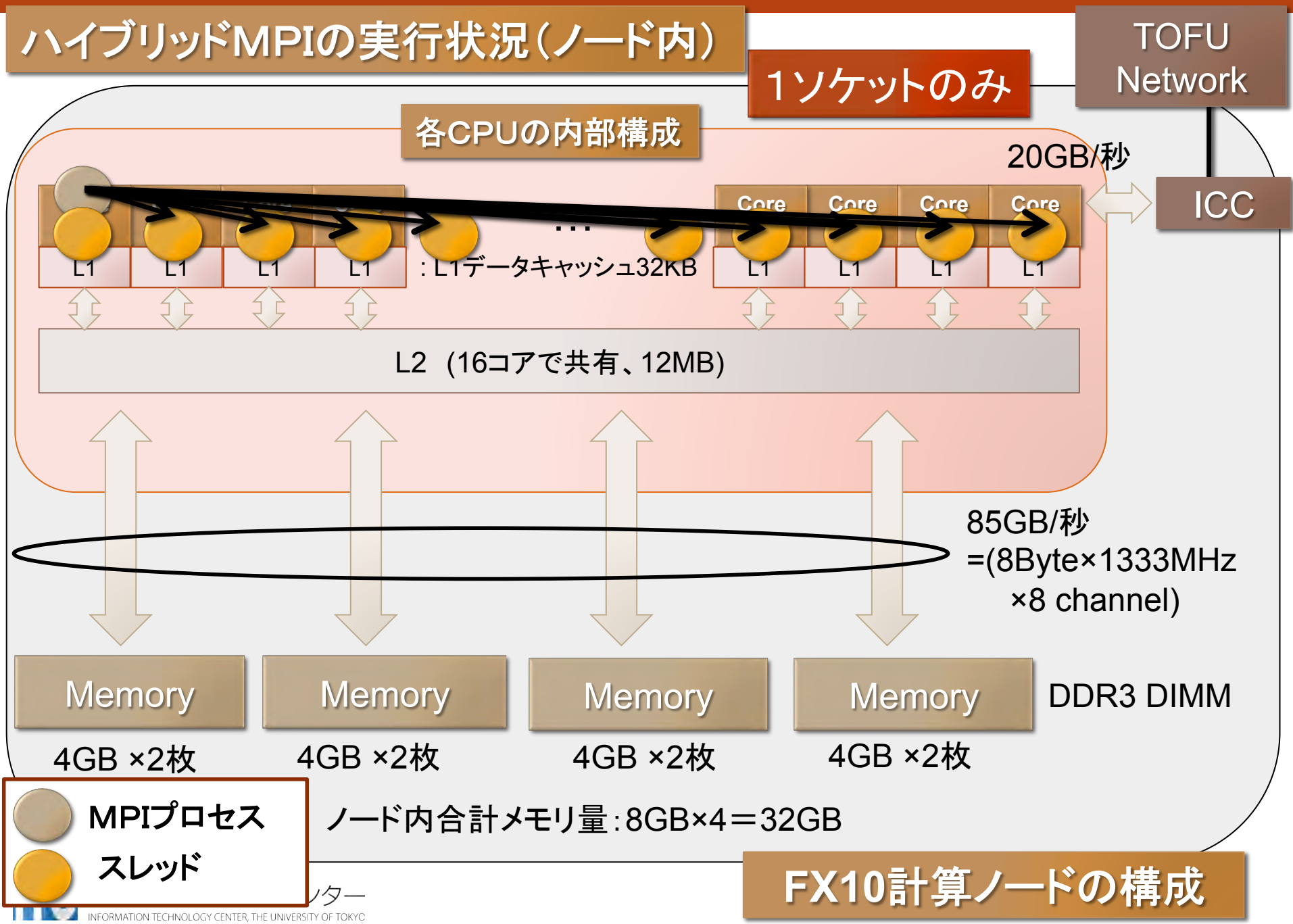
利用コア数（MPIプロセス数）

実行時間制限：1分

1MPIプロセス当たり16スレッド生成

MPIジョブを1＊12 = 12 プロセスで実行する。

ハイブリッドMPIの実行状況（ノード内）



FX10計算ノードの構成

並列版Helloプログラムの説明（C言語）

このプログラムは、全PEで起動される

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

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

    int    myid, numprocs;
    int    ierr, rc;

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

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

    rc = MPI_Finalize();

    exit(0);
}
```

MPIの初期化

自分のID番号を取得
：各PEで値は異なる

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

MPIの終了

並列版Helloプログラムの説明（Fortran言語）

このプログラムは、全PEで起動される

```
program main

common /mpienv/myid,numprocs

integer  myid, numprocs
integer  ierr

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

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

call MPI_FINALIZE(ierr)

stop
end
```

MPIの初期化

自分のID番号を取得
：各PEで値は異なる

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

MPIの終了

時間計測方法（C言語）

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

＜ここに測定したいプログラムを書く＞

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

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

バリア同期後
時間を習得し保存

各プロセッサーで、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

- 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

性能プロファイラ

性能プロファイラ

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

性能プロファイラの種類の詳細

- 基本プロファイラ
  - コマンド例：fipp –C
  - 表示コマンド：fipppx、GUI（WEB経由）
  - ユーザプログラムに対し一定間隔(デフォルト時100 ミリ秒間隔)毎に割り込みをかけ情報を収集する。
  - 収集した情報を基に、コスト情報等の分析結果を表示。
- 詳細プロファイラ
  - コマンド例：fapp –C
  - 表示コマンド：GUI(WEB経由)
  - ユーザプログラムの中に測定範囲を設定し、測定範囲のハードウェアカウンタの値を収集。
  - 収集した情報を基に、MFLOPS、MIPS、各種命令比率、キャッシュミス等の詳細な分析結果を表示。

基本プロファイラ利用例

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

基本プロファイラ出力例（1/2）

```
---------------------------------------------------------------------------------------
Fujitsu Instant Profiler Version 1.2.0
  Measured time              : Thu Apr 19 09:32:18 2012
  CPU frequency              : Process      0 -    127  1848 (MHz)
  Type of program            : MPI
  Average at sampling interval  : 100.0 (ms)
  Measured range             : All ranges
  Virtual coordinate         : (12, 0, 0)
---------------------------------------------------------------------------------------

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

基本プロファイラ出力例(2/2)

```
Procedures profile
  ********************************************************************************
    Application - procedures
  ********************************************************************************
         Cost       %         Mpi       %   Start     End
  --------------------------------------------------------------------
          475  100.0000         312   65.6842      --      --  Application
  --------------------------------------------------------------------
          312   65.6842         312  100.0000       1      45  MAIN__
           82   17.2632           0    0.0000      --      --  __GI___sched_yield
           80   16.8421           0    0.0000      --      --  __libc_poll
            1    0.2105           0    0.0000      --      --  __pthread_mutex_unlock_usercnt
  ********************************************************************************
    Process    11 - procedures
  ********************************************************************************
         Cost       %         Mpi       %   Start     End
  --------------------------------------------------------------------
            5  100.0000           4   80.0000      --      --  Process    11
  --------------------------------------------------------------------
            4   80.0000           4  100.0000       1      45  MAIN__
            1   20.0000           0    0.0000      --      --  __GI___sched_yeld
....
```

詳細プロファイラ利用例

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

詳細プロファイラ利用例

- 空のディレクトリがないとダメなので、/Wa2 に Profディレクトリを作成
  $ mkdir Prof
- Wa2のwa2-pure.bash中に以下を記載
  (キャッシュ情報取得時)
  fapp -C -d Prof –L 1 –Ihwm –Hevent=Cache mpirun ./wa2
- 実行する
  $ pjsub wa2-pure.bash

詳細プロファイラGUIによる表示例

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

https://oakleaf-fx-1.cc.u-tol
fsdtFX10tx/
install/index.html

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

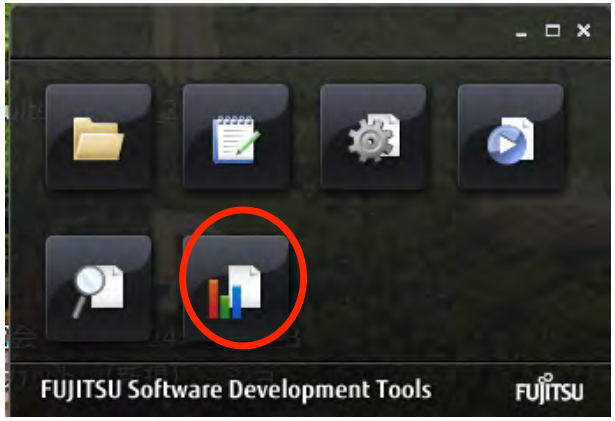
詳細プロファイラGUIによる表示例

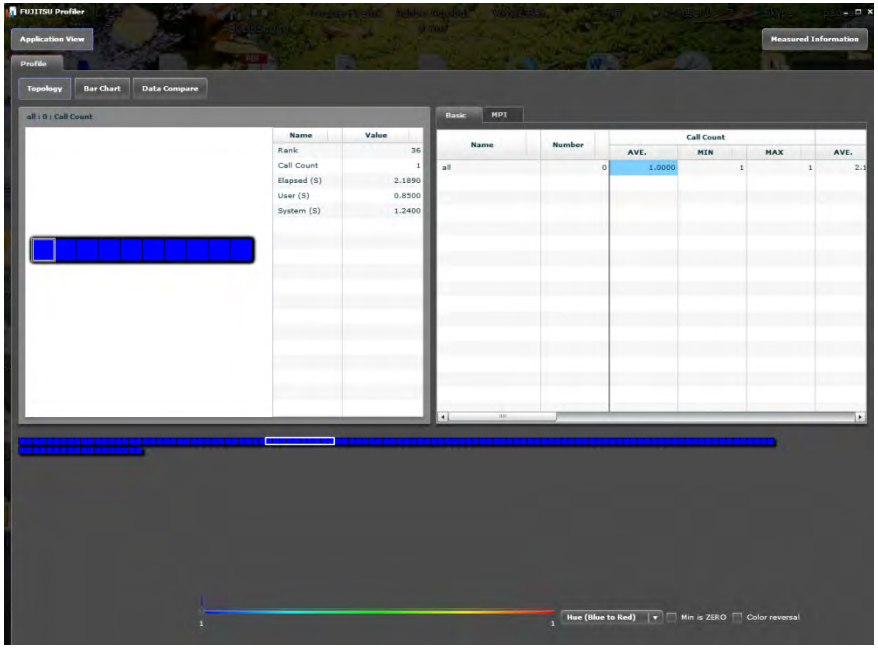
- MACの場合は、以下をアクセス

https://oakleaf-fx-1.cc.u-tokyo.ac.jp/fsdtFX10tx/install/mac/index.html

詳細プロファイラGUIによる表示例

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





詳細プロファイラで取れるデータ

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

精密PA可視化機能（エクセル形式）

- 性能プロファイルは見にくい
- 7回程度実行しないといけないが、性能プロファイルデータ（マシン語命令の種類や、実行時間に占める割合など）を、Excelで可視化してくれるツール（精密PA可視化ツール）が、東京大学情報基盤センターのFX10では、提供されている
- 手順
  1. 対象箇所（ループ）を、専用のAPIで指定する
  2. プロファイルを入れるフォルダ7か所をつくる
  3. プロファイルのためのコマンドで7回実行する
  4. エクセル形式に変換する
  5. 4のエクセル形式を手元のパソコンに持ってくる
  6. 5のファイルを、指定のエクセルと同一のフォルダに入れてから、指定のエクセルを開く

精密PA可視化のための指示API

- 以下のAPIで、対象となるループを挟む（Fortranの場合）

```
call start_collection(“region”)
    <対象となるループ>
call stop_collection(“region”)
```

- “region”は、対象となる場所の名前なので、任意の名前を付けることが可能
（後で、専用エクセルを開くときに使う）

実行のさせ方

- a.out という実行ファイルの場合以下のように7回実行する
- 実行するディレクトリに、pa1、pa2、…、pa7という、ディレクトリを作っておく必要がある
- 以下のように実行する

```
fapp -C –d pa1 -Hpa=1 mpiexec a.out
fapp -C –d pa2 -Hpa=2 mpiexec a.out
fapp -C –d pa3 -Hpa=3 mpiexec a.out
fapp -C –d pa4 -Hpa=4 mpiexec a.out
fapp -C –d pa5 -Hpa=5 mpiexec a.out
fapp -C –d pa6 -Hpa=6 mpiexec a.out
fapp -C –d pa7 -Hpa=7 mpiexec a.out
```

エクセルデータへの変換

- pa1、pa2、…、pa7の中に、プロファイルデータがあることを確認する
- 以下のコマンドを実行する

```
fapppx -A -d pa1 -o output_prof_1.csv -tcsv -Hpa
fapppx -A -d pa2 -o output_prof_2.csv -tcsv -Hpa
fapppx -A -d pa3 -o output_prof_3.csv -tcsv -Hpa
fapppx -A -d pa4 -o output_prof_4.csv -tcsv -Hpa
fapppx -A -d pa5 -o output_prof_5.csv -tcsv -Hpa
fapppx -A -d pa6 -o output_prof_6.csv -tcsv -Hpa
fapppx -A -d pa7 -o output_prof_7.csv -tcsv -Hpa
```

エクセルデータを手元のPCに転送

- 以下のFX10上のエクセルデータを、手元のPCに転送
  - output_prof_1.csv、output_prof_2.csv、…、output_prof_7.csv
- 上記のエクセルデータが入ったフォルダで、ポータル上で公開されている専用エクセルを開く
- 詳細なエクセルデータの利用法、分析されたデータの見方は、マニュアル参照
  - J2UL-1490-05Z0(00)　2014年3月、Technical Computing Suite V1.0、プロファイラ使用手引書(PRIMEHPC FX10用)

性能最適化情報の提示（1/2）

- C、Fortranを問わず、コンパイラが行った最適化情報を知ることは性能最適化で重要である
- 各ファイルのソースコードごとに、どのような最適化が行われたか出力する、コンパイラオプションがある
- C、Fortranで共通の翻訳情報
  - -Nlst=p ：標準の最適化情報を出力(デフォルト)
  - -Nlst=t ：詳細な最適化情報を出力

性能最適化情報の提示(2/2)

- Fortran のみ、p、t以外も指定可能
  - -Nlst=a ：名前の属性情報を出力
  - -Nlst=d ：派生型の構成情報を出力
  - -Nlst=I ：インクルードされたファイルのプログラムリストおよびインクルードファイル名一覧を出力を出力
  - -Nlst=m ：自動並列化の状況をOpenMP指示文によって表現した原始プログラムを出力
  - -Nlst=x ：名前および文番号の相互参照情報を出力
  - 詳細は、オンラインマニュアルの以下を参照のこと
    - C言語使用手引書： P.26
    - C++言語使用手引書： P.28
    - Fortran使用手引書： P.46 P.52 P.53

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-fx.tar
- ジョブスクリプトファイルmat-mat.bash　中のキュー名を
  lecture から tutorial に変更してから、
  pjsub　してください。
  - lecture : 実習時間外のキュー
  - 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 /home/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
  $ pjsub 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 = 0.209609 [sec.]

9541.570931 [MFLOPS]

OK!

1コアのみで、9.5GFLOPS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.] = 0.204734672995982 7

MFLOPS = 9768.741003580422

OK!

1コアのみで、9.7GFLOPS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)

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 192
  - #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-fx.tar
- ジョブスクリプトファイルmat-mat-d.bash　中のキュー名を　lecture から tutorial に変更し、pjsub　してください。
  - lecture : 実習時間外のキュー
  - 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 /home/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  = 384
Mat-Mat time  = 0.000135 [sec.]
 841973.194818 [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  =  384
Mat-Mat time  =  1.295508991461247E-03
MFLOPS =  87414.45135502046
 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 384
  - 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=384)

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 384
    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