

# KOBE HPC サマースクール 2018

## 講義資料 2018/8/6

兵庫県立大学シミュレーション学研究科  
安田 修悟

# 受講対象

- 並列計算やアクセラレータの基礎的なプログラミング技術を習得したい。
- プログラミング言語(講義ではC言語)とLinuxの基本的な知識。

# 講義スタッフ

- 計算機・アカウント管理
  - 島、安田(兵科大・シミュ学)
- TA
  - 後藤、松崎 (神戸大)
  - 川手、金子、酒井、仲村、平野(兵科大)
- 講師
  - 横川(神大・ECCSE)、八木(理研・R-CCS)、安田

# 講義内容

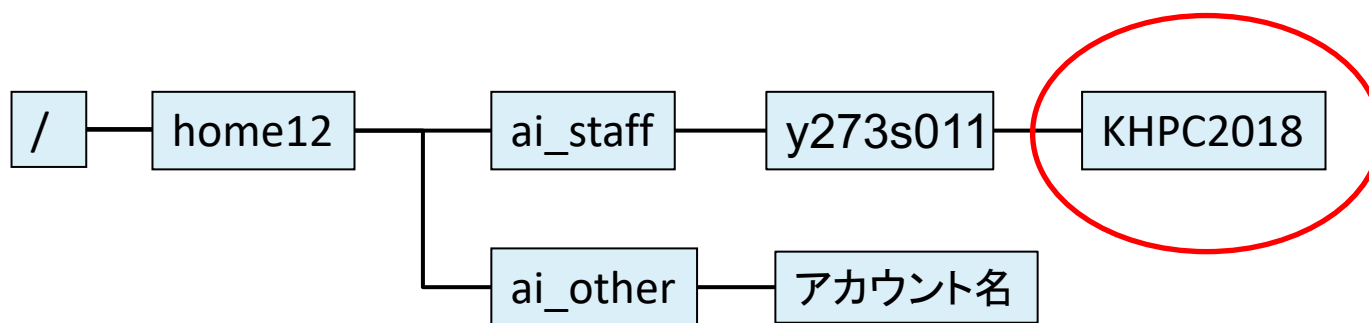
1. 計算機サーバーの環境設定と使い方 (担当 安田)
2. シリアルプログラムの高速化
3. 熱伝達問題の差分計算
4. スレッド並列とは (担当 八木)
5. OpenMPによるループ処理の並列化
6. 差分化された偏微分方程式の並列化
7. アムダール法則と並列化効率の評価
8. 分散メモリ型並列計算機とは何か？ (担当 横川)
9. 1対1通信関数、集団通信関数
10. 並列計算性能の評価方法
11. 熱伝導問題の並列計算
12. ハイブリッド並列(スレッド並列+プロセス並列)
13. アクセラレータとは (担当 安田)
14. OpenACCプログラミング
15. まとめと確認テスト

# 利用環境

- 注意事項
  - 飲食禁止。
  - 指定の端末を利用。
- 起動とログイン
  - 端末PCの電源ON後, **Linux**を選択。
  - 「ユーザーID」と「パスワード」でログイン。
- ブラウザ(Fire Fox)設定
  - 詳細 -> ネットワーク
  - Proxy: 172.25.50.4: 8080

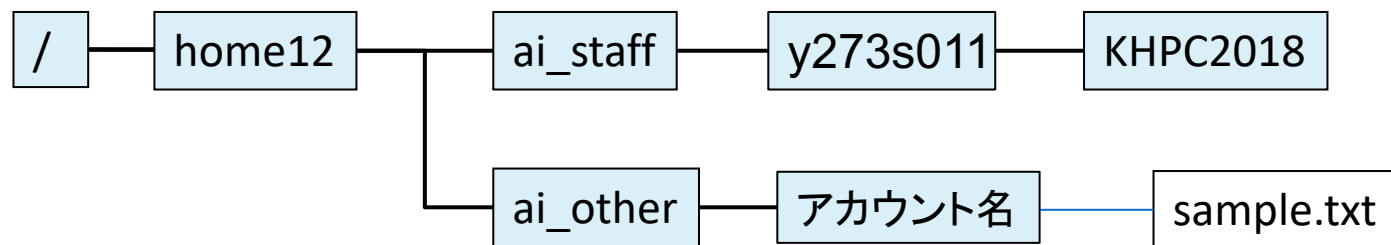
# ディレクトリ -1

- ルートディレクトリから始まり階層構造をなしている。
- ディレクトリの階層は“/”で区切る。
- 一番上のディレクトリをルートディレクトリと呼ぶ。
- 現在いるディレクトリをカレントディレクトリと呼ぶ。
- /home12/ai\_other/に各アカウントのホームディレクトリがある。
- /home12/ai\_staff/y273s011/KHPC2018に講義資料を置きます。



# ディレクトリ -2

- 絶対パスとは、ルートディレクトリ"/"からのパス
- 相対パスとは、カレントディレクトリからのパス
- “~/”はホームディレクトリを表す。



sample.txtは、

絶対パス: /home12/ai\_other/<アカウント>/sample.txt

相対パス: ./sample.txt (カレントディレクトリがホームディレクトリの場合)

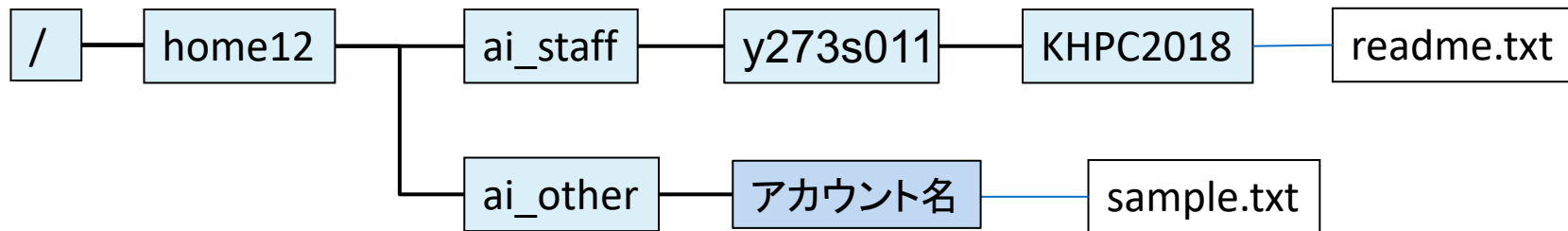
又は: ~/sample.txt

で指定できる。

# ディレクトリ -3

## その他

- .. (ドット2つ)は、一つ上のディレクトリ
- . (ドット1つ)は、カレントディレクトリ



例 readme.txtは、

## カレントディレクトリが<アカウント名>のとき

相対パス(relative path): ../../ai\_staff/y273s011/KHPC2018/readme.txt

cat \_/home12/ai\_staff/y273s011/KHPC2018/readme.txt ↵でも

cat\_ ../..ai\_staff/y273s011/KHPC2018/readme.txt ↵でも同じ結果。



# ディレクトリ・ファイル操作 -1

Linuxのコマンドで、カレントディレクトリの内容を表示する。

`ls`↵

Linuxのコマンドは、オプションや引数をとるものがある。

## lsの主なオプション

|    |               |    |                              |
|----|---------------|----|------------------------------|
| -a | ドットファイルなども表示  | -F | 実行可能ファイル、ディレクトリなどに記号を付けて表示する |
| -l | ファイルの詳しい情報を表示 |    |                              |

試してみよう

`ls -F`↵

`ls -lF`↵ (複数のオプションをまとめて書いた) etc

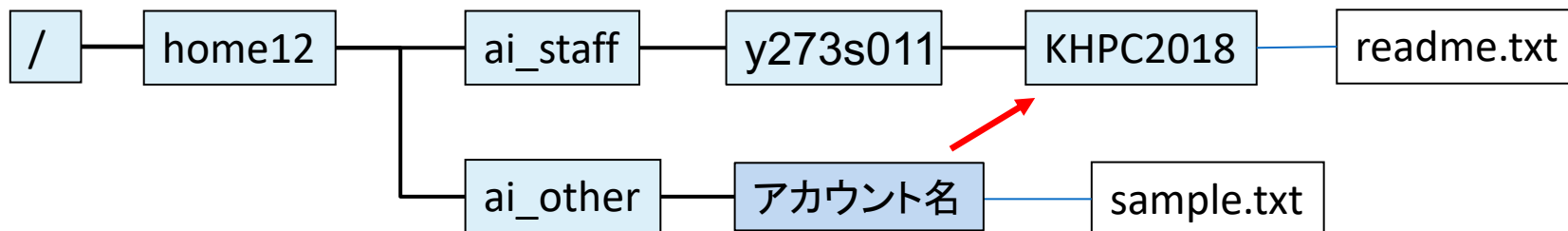
# ディレクトリ・ファイル操作 -2

pwd↵

- カレントディレクトリの表示。

cd \_ 移動先ディレクトリ↵

- **cd \_/home12/ai\_staff/y273s011/KHPC2018** で矢印の様に移動。
- ホームディレクトリへ移動. **cd**↵ でもどる



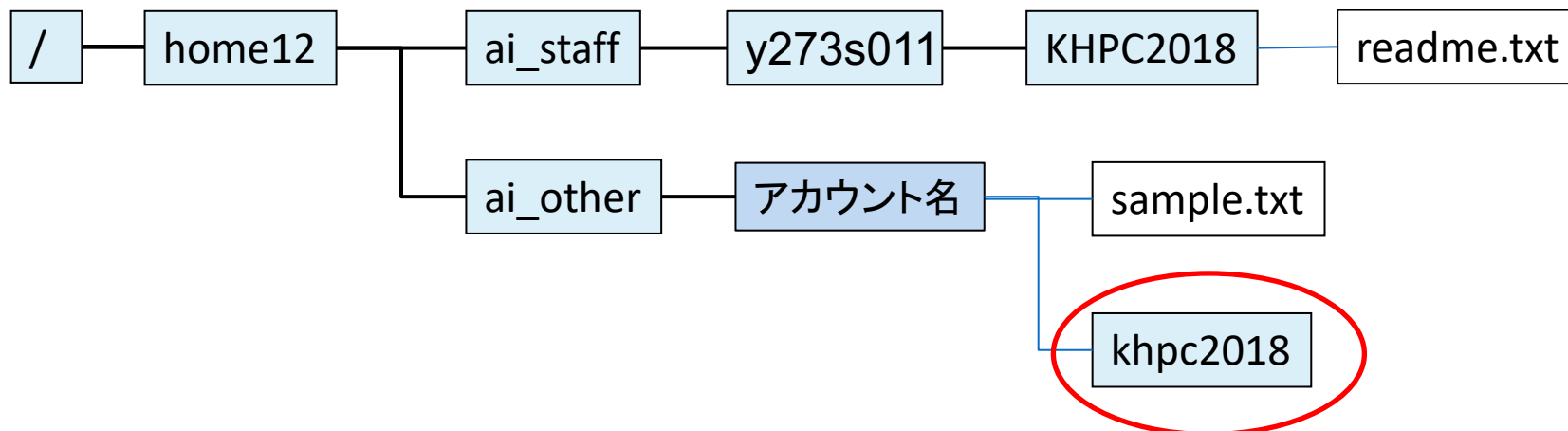
# ディレクトリ・ファイル操作 -3

mkdir\_ディレクトリ名 ↵

- 新しいディレクトリの新規作成
- ホームディレクトリの下にkhpc2018という名前のディレクトリを作成.

mkdir\_khpc2018

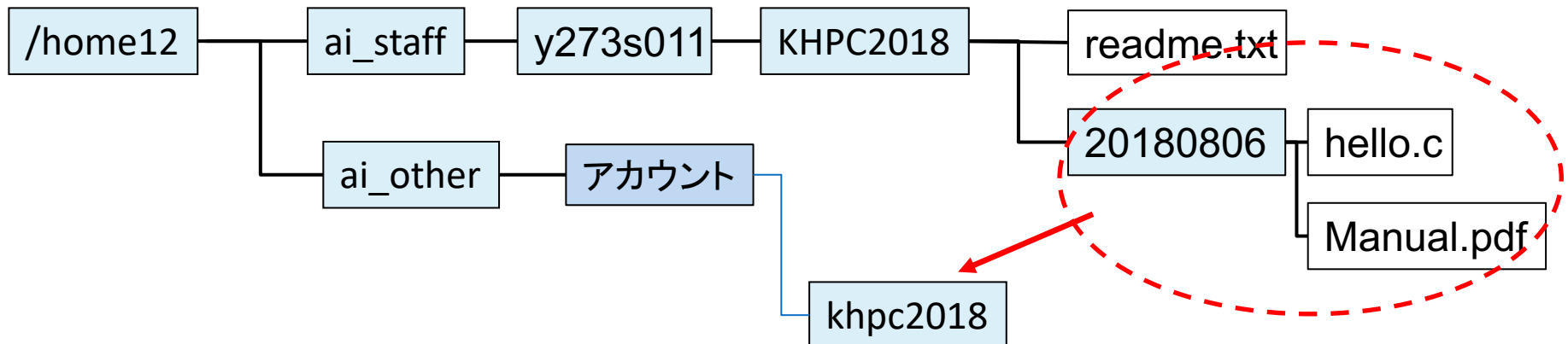
- ls\_-F ↵ で確認



# ディレクトリ・ファイル操作 -4

cp\_オプション\_引数1\_引数2↵

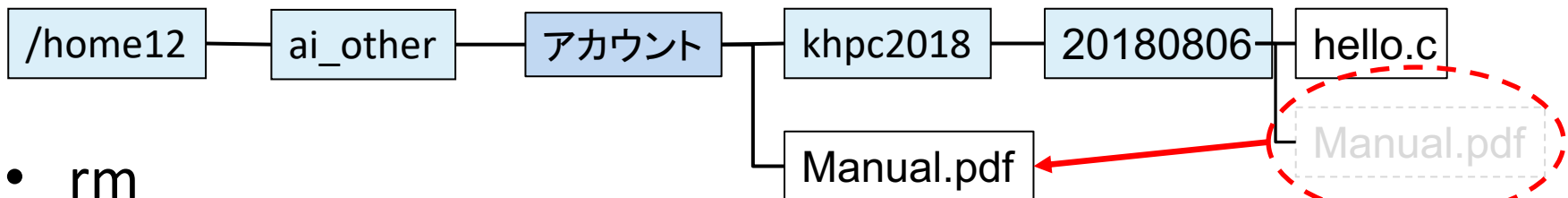
- 引数1(ファイル)を引数2で指定したディレクトリにコピーする.



- 講義資料の入ったディレクトリをホームディレクトリにコピー.
- **cp -r** /home12/ai\_staff/y273s011/KHPC2018/20180806 ./khpc2018/
- “-r”はディレクトリをコピーする時のオプション.

# ディレクトリ・ファイル操作 -5

- mv
  - ファイルの移動.
  - 例: `mv ./khpc2018/20180806/Manual.pdf ./` ⇐
  - カレントディレクトリにManual.pdfが移動.

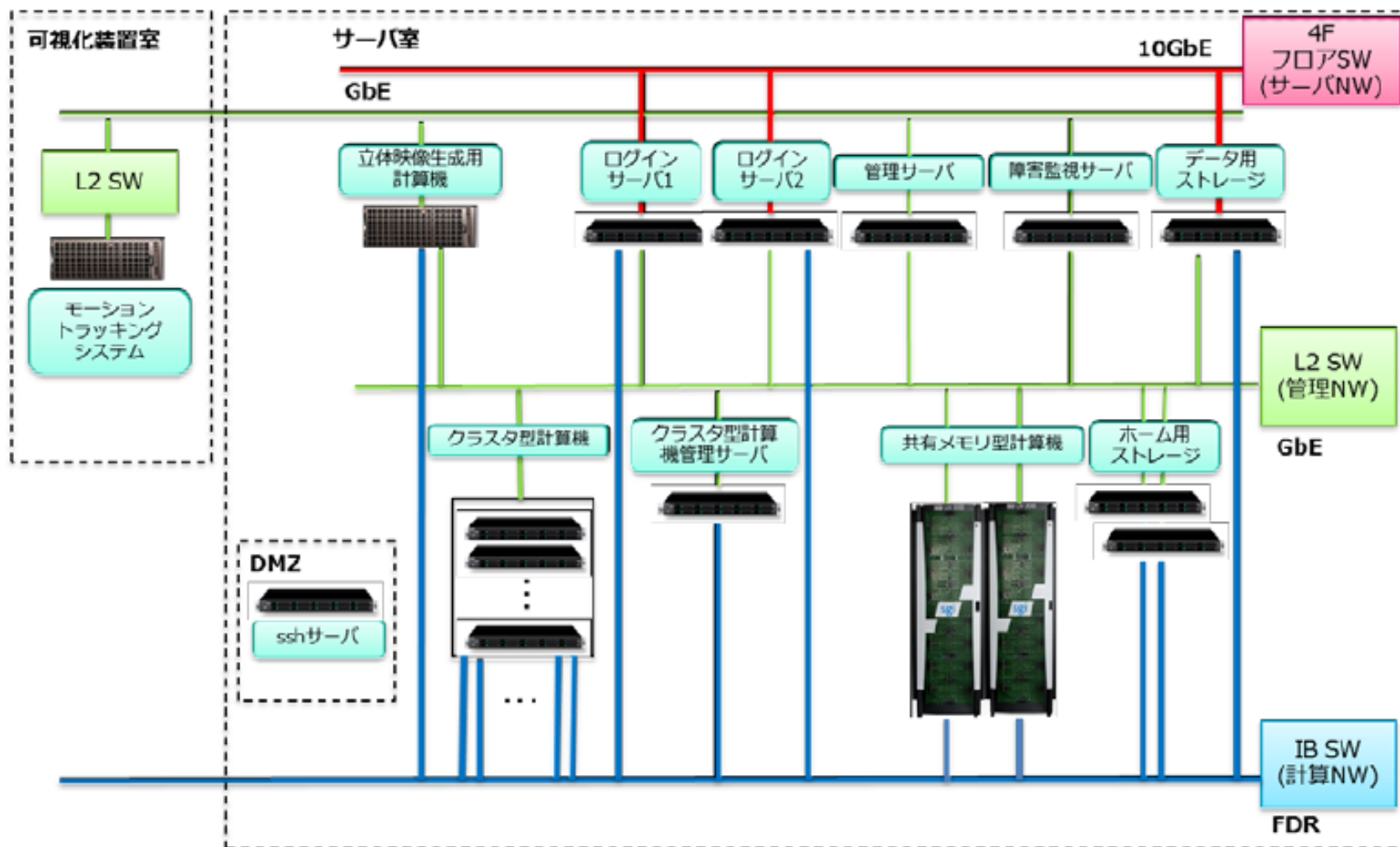


- rm
  - ファイル、ディレクトリの削除。ディレクトリの削除には、-rオプションが必要

# ディレクトリ・ファイル操作 -6

- “\*” ワイルドカード
  - 任意の1文字以上を補完
  - 例: カレントディレクトリに、a.out, bonjour.out, hello.c, bonjour.c, hola.cがあるとする
    - `ls *.c`で、hello.c, bonjour.c, hola.cが表示される
    - `ls *.out`で、a.out, bonjour.outが表示される
    - `ls hello.*`で、hello.c, hello.outが表示される
- man
  - コマンドの説明を表示。例 `man cp` (cpの使い方表示)。終了するときは、q(アルファベットのq)を押す
- その他
  - 矢印キー “↑” “↓” で、入力したコマンドの履歴が見れる
  - “Tab” キーで、コマンドやファイル名を補完してくれる

# 計算機サーバシステム概要



# 計算機サーバシステム概要

|                     | Host                                                 | System                                 | CPU/Accelerator                                     | Memory         |
|---------------------|------------------------------------------------------|----------------------------------------|-----------------------------------------------------|----------------|
| フロントエンドサーバ          | rokko1<br>[172.25.61.11]<br>rokko2<br>[172.25.61.11] | SGI Standard-Depth Server<br>C1110-GP2 | Intel Xeon<br>E5-2667v3<br>3.2GHz<br>(8coresx2)     | 128GB          |
| クラスタ型<br>(通常ノード)    | r03n01~r03n28,<br>r04n01~r04n24                      | SGI Standard-Depth Server              | Intel Xeon E5-2650v3 2.3GHz<br>(10cores x2)         | 128GB          |
| クラスタ型<br>(mic搭載ノード) | r03p01~r03p08                                        |                                        | Xeon Phi 5110P<br>(1.053GHz 60cores<br>/240threads) | 8GB            |
| クラスタ型<br>(gpu搭載ノード) | r04g01~r04g08                                        |                                        | NVIDIA Tesla<br>K40m (754MHz<br>192cores x15)       | 12GB           |
| 共有メモリ型              | kofuji1<br>kofuji2                                   | SGI UV3000                             | Intel Xeon E5-4627v3 2.6GHz<br>(10coresx8)          | 2TB (256GB x8) |



# フロントエンドサーバ(rokko)へのログイン

- アカウント名  
“guest受講者番号(2桁)”となっています.  
例: guest01, guest31, ...
- ログインパスワードはアカウント名と同一です.
- 接続用アドレス  
IP: 172.25.51.93, Port: 50001 => rokko1 (番号奇数)  
IP: 172.25.51.93, Port: 50002 => rokko2 (番号偶数)
- 接続方法(例: 受講者番号2)  
**ssh\_Y\_guest02@172.25.51.93\_-p\_50002**
- ファイル<hello.c>をサーバのホームディレクトリにコピー.  
**scp\_-P\_50002\_./hello.c\_guest02@172.25.51.93:~/**

# プログラミング環境設定

- moduleコマンド
  - Module環境のロード／アンロード  
**module\_load\_intel / \$module\_unload\_intel**
  - 現在の設定  
**module\_list**
  - 利用可能なmoduleの確認  
**module\_avail**
- その他module設定について  
「利用者マニュアル\_rev2.0」(UserManual.pdf)  
の13頁参照

# プログラミング環境設定

## 1. “.module”ファイルの作成

> **gedit \_module**

```
source _/etc/profile.d/modules.sh  
module _load _intel
```

## 2. “~/.bashrc”の最下部に次の1行を追記.

> **gedit \_bashrc**

```
._ ~/.module
```

## 3. 端末に “bash” と入力. エラーが無ければOK.

# プログラミング環境

- コンパイラ(「利用者マニュアル」37頁以降)
  - Intel  
`icc -O3 hello.c -o hello.out`  
オプション                      実行ファイルの指定
- 実行はPBSジョブ管理システムを使います.  
(「利用者マニュアル」15頁)
  - インタラクティブジョブ  
`qsub -I`
  - ジョブスクリプトの利用  
`qsub <ジョブスクリプト>`

# PBSジョブ管理システム

- バッチスクリプトの作成  
**gedit\_single\_run.sh**

```
#!/bin/bash
#PBS_-q_S
#PBS_-l_select=1:ncpus=1:mem=1gb
#PBS_-N_single_job
#PBS_-j_oe
source_/etc/profile.d/modules.sh
module_load_intel
cd_${PBS_O_WORKDIR}
dplace_./hello.out
```

- ・キューの指定
- ・計算リソースの指定
- ・任意のジョブ名の指定
- ・標準出力と標準エラー出力を統合
- ・moduleを利用する為の環境設定
- ・Intelコンパイラの環境設定
- ・投入ディレクトリに移動
- ・dplaceコマンドの指定

- ジョブ投入  
**qsub\_single\_run.sh**

# PBSジョブ管理システム

## ● キュー構成

表 3 キュー編成

| キュー名     |   | ノード/コア<br>数制限 | 最長<br>実行時間 | 使用できるノード      | アクセラ<br>レータ数   | 優先度 | インタラ<br>クティブ |
|----------|---|---------------|------------|---------------|----------------|-----|--------------|
| T        |   | 1 ノード         | 30min      | クラスタ通常ノード     |                | 150 | 可            |
| C        | S | 1-4 ノード       | 24h        | クラスタ通常ノード     |                | 130 | 可            |
|          | M | 5-10 ノード      | 24h        | クラスタ通常ノード     |                | 90  |              |
|          | L | 11-52 ノード     | 24h        | クラスタ通常ノード     |                | 70  |              |
|          | H | 53-68 ノード     | 24h        | クラスタ全ノード      | GPU 8<br>Phi 8 | 50  |              |
| LONG     |   | 1-4 ノード       | 168h       | クラスタ通常ノード     |                | 130 |              |
| G        |   | 1-2 ノード       | 24h        | GPU 搭載ノード     | GPU 2          | 130 | 可            |
| GL       |   | 3-8 ノード       | 24h        | GPU 搭載ノード     | GPU 8          | 70  |              |
| P        |   | 1-2 ノード       | 24h        | MIC 搭載ノード     | Phi 2          | 130 | 可            |
| PL       |   | 3-8 ノード       | 24h        | MIC 搭載ノード     | Phi 8          | 70  |              |
| SMP1     |   | 1-80 コア       | 24h        | 共有メモリ kofuji1 |                | 130 | 可            |
| SMP2     |   | 1-80 コア       | 24h        | 共有メモリ kofuji2 |                | 130 | 可            |
| SMP-LONG |   | 1-80 コア       | 168h       | 共有メモリ kofuji2 |                | 130 |              |

- キュー名 : ジョブ投入の際に指定するキュー名です。
- ノード/コア数制限 : ジョブ投入の際に予約できる最小・最大ノード/コア数です。
- 最長実行時間 : ジョブ投入の際に予約できる最大実行時間です。
- 使用できるノード : 各キューで利用可能なノードです。
- アクセラレータ数 : 各キューで利用可能なアクセラレータ数です。
- 優先度 : キューの優先順位。大きい値の方が優先度が高くなります。

# PBSジョブ管理システム

- 結果表示

**cat\_single\_job.o<ジョブID>**

- 実行ジョブの確認（「利用者マニュアル」31頁）

**qstat**

表 5 qstat コマンドの主なオプション

| オプション          | 説明                                                          |
|----------------|-------------------------------------------------------------|
| -u_<user_name> | 指定したユーザのジョブの状態を表示します.                                       |
| -f_<ジョブ ID>    | 特定のジョブの詳細な状況を表示します. ジョブ ID を省略すると, すべてのジョブについての詳細な状況を表示します. |
| -Q             | キューの状況を表示します.                                               |
| -s             | ジョブコメントおよびその他の情報が表示されます. キューイング中のジョブは実行待ちになっている理由が表示されます.   |

- ジョブのキャンセル（「利用者マニュアル」32頁）

**qdel\_<ジョブID>**

# PBSジョブ管理システム

- インタラクティブキュー
  - **qsub -l -q T** クラスタノードにログイン.  
“-q”で利用するキュー(S or T)の指定.
  - Job IDが割り当てられる. **qstat**で確認.
  - 作業ディレクトリに移動して実行  
**dplace ./a.out**
  - **exit** コマンドで終了. (ノードからログアウト.)

**必ずexitでログアウトして下さい!!**

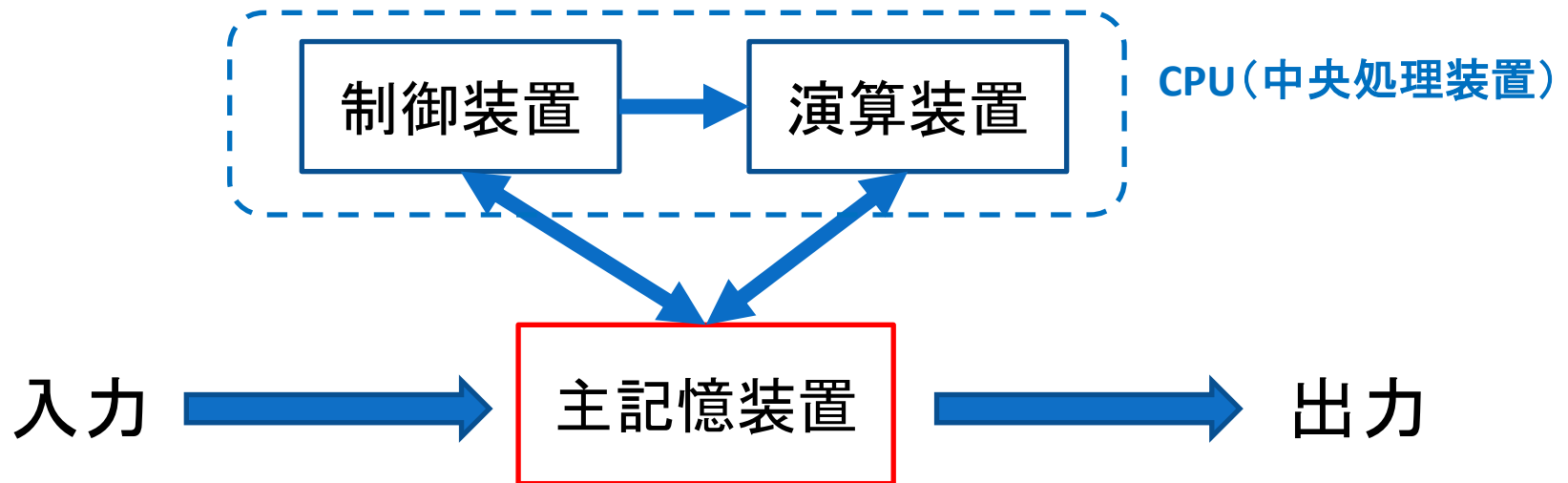


# 演習0

- hello.cをインタラクティブキューを使って実行せよ。
  1. `qsub -l -q T`
  2. `cd 作業ディレクトリ`
  3. `icc hello.c -o hello.out`
  4. `./hello.out` 任意の文字列
- バッチスクリプトを使って実行せよ。

# 計算機の基本構造

- フォン・ノイマン型 (von Neumann architecture)



1. 主記憶装置(メモリ), **1次元のアドレス空間**をもつ,
2. 命令とデータがともに記憶装置に記憶される.
3. 演算は制御装置からの命令信号によって実行される.
4. 命令は, プログラムカウンタにより逐次的に実行される.

# 計算機の動作原理

- チューリングマシン

1. 無限に長いテープ



- テープの情報の読み書き.
- 機械の内部状態の変更.
- ヘッドの移動.



2. ヘッド(情報の読み書き)

3. 内部状態メモリ

- ノイマン型



メモリ



プログラムカウンタ -> 実行する命令の番地を示す

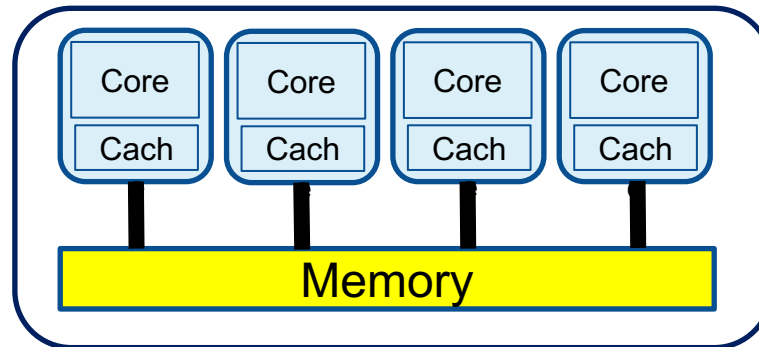
0 1 3 5 6 1

※分岐命令でカウンタを書き換える.

# 並列計算機

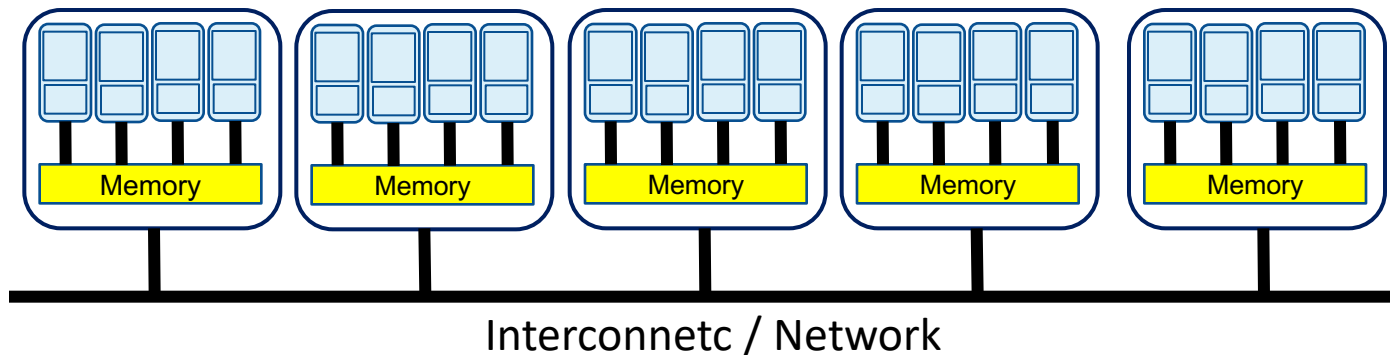
- 並列計算機の種類

SMP (共有メモリ型)



Node

SMP Cluster



# 並列計算

- 並列計算・・・演算を分割して、複数のプロセッサに割り当てて処理する
  - プロセス並列
    - 分割された演算(プロセス)が、それぞれ**独立のメモリ空間**を参照する。
    - **プロセス間にデータ転送が必要。**
  - スレッド並列
    - 分割された演算(スレッド)は、**メモリ空間を共有できる(複数のスレッドで変数を共有できる)。**
    - 各スレッドは他のスレッドからはアクセスできない固有の変数も用意できる。

# 逐次計算の高速化

## ● ループ展開

- 繰り返し処理で毎回発生する終了条件のチェックの低減
- ループ制御のカウンタやポインタの更新回数の低減

行列積の計算

```
for(i=0;i<IMAX;i++)  
  for(j=0;j<IMAX;j++)  
    for(k=0;k<IMAX;k++)  
      c[i][j]+=a[i][k]*b[k][j];
```

2つ目のループについて展開

```
for(i=0;i<IMAX;i++)  
  for(j=0;j<IMAX;j+=8)  
    for(k=0;k<IMAX;k++){  
      c[i][j]+=a[i][k]*b[k][j];  
      c[i][j+1]+=a[i][k]*b[k][j+1];  
      c[i][j+2]+=a[i][k]*b[k][j+2];  
      c[i][j+3]+=a[i][k]*b[k][j+3];  
      c[i][j+4]+=a[i][k]*b[k][j+4];  
      c[i][j+5]+=a[i][k]*b[k][j+5];  
      c[i][j+6]+=a[i][k]*b[k][j+6];  
      c[i][j+7]+=a[i][k]*b[k][j+7];  
    }
```

# 逐次計算の高速化

## ● 一時変数の活用

- ループ内で同じ変数へのアクセスが多い場合、一時変数を用いると冗長な命令を省く事ができる.

```
for(i=0;i<IMAX;i++)  
  for(j=0;j<IMAX;j+=8)  
    for(k=0;k<IMAX;k++){  
      c[i][j]+=a[i][k]*b[k][j];  
      c[i][j+1]+=a[i][k]*b[k][j+1];  
      c[i][j+2]+=a[i][k]*b[k][j+2];  
      c[i][j+3]+=a[i][k]*b[k][j+3];  
      c[i][j+4]+=a[i][k]*b[k][j+4];  
      c[i][j+5]+=a[i][k]*b[k][j+5];  
      c[i][j+6]+=a[i][k]*b[k][j+6];  
      c[i][j+7]+=a[i][k]*b[k][j+7];  
    }
```

配列a[i][k]へ冗長なアクセス

```
for(i=0;i<IMAX;i++)  
  for(j=0;j<IMAX;j+=8)  
    for(k=0;k<IMAX;k++){  
      t=a[i][k];  
      c[i][j]+=t*b[k][j];  
      c[i][j+1]+=t*b[k][j+1];  
      c[i][j+2]+=t*b[k][j+2];  
      c[i][j+3]+=t*b[k][j+3];  
      c[i][j+4]+=t*b[k][j+4];  
      c[i][j+5]+=t*b[k][j+5];  
      c[i][j+6]+=t*b[k][j+6];  
      c[i][j+7]+=t*b[k][j+7];  
    }
```

# 演習1 (ex01.c)

- ループ展開無し, ループ展開で2回処理, 4回処理, 8回処理の場合について実行速度を比較せよ.
- ループ展開で8回処理した後に, さらに一時変数を用いることで実行速度が速くなるか確認せよ.

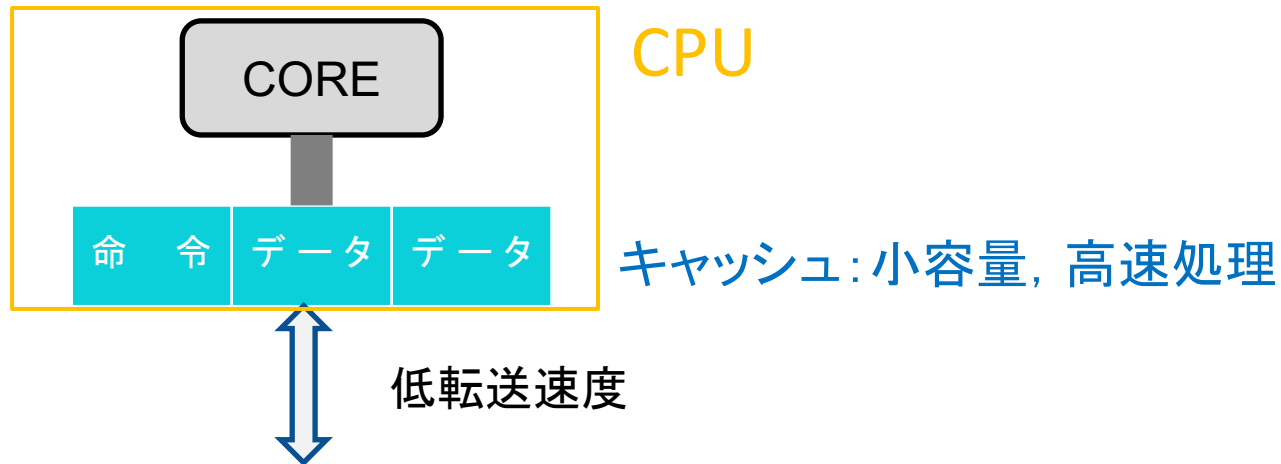
※コンパイルオプションを“-O0”(最適化無し)としてコンパイルする.



# 逐次計算の高速化

- キャッシュメモリ

- データアクセスの時間を短縮する.



主記憶装置(メモリ): 大容量, 1次元アドレス空間

番地で区切られたデータ量をメモリとキャッシュでやりとりする.

# 逐次計算の高速化

## 1. 水平参照と垂直参照

- 水平参照の方がキャッシュミスが少ない.

2次元配列  $A[i][j]$      $A[i][j]$ は1次元メモリ空間で  $i \times JMAX + j$  番目のデータ.

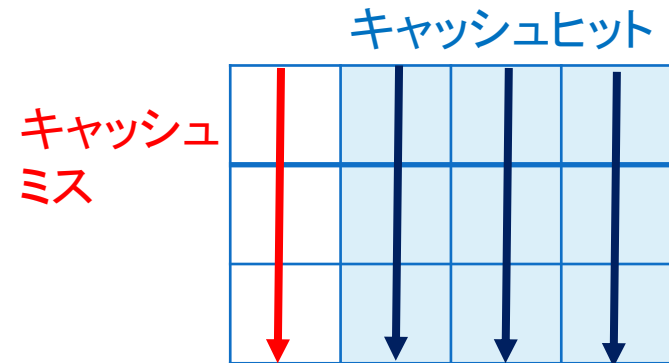
|   |       |       |          |       |       |       |       |       |
|---|-------|-------|----------|-------|-------|-------|-------|-------|
| 0 | 1     | 2     | 3        | ..... | J - 4 | J - 3 | J - 2 | J - 1 |
| J | J + 1 | J + 2 | J + 3    | ..... | 2J-4  | 2J-3  | 2J-2  | J2-1  |
| ⋮ |       |       |          | ⋮     |       |       |       |       |
|   |       |       | $[i][j]$ |       |       |       |       |       |
|   |       |       |          |       |       |       |       |       |
|   |       |       |          |       |       |       |       |       |
|   |       |       |          |       |       |       |       |       |

$A[i][j]$ のデータが必要な場合, その前後のデータをまとめキャッシュにあげる.

Fortranでは $A(i,j)$ は $j \times IMAX + i$ の順番になるので注意!!

# 逐次計算の高速化

## 2. サブブロック分割

[illegible]

# 演習2

- 2つの高速化処理についてベンチマーク実行.  
(コンパイルの最適化オプションは“-O0”を指定)
  - a. 垂直参照と水平参照の比較  
ex02a.cを水平参照のコードに改良せよ.
  - b. 垂直参照とサブブロック分割の比較.  
ex02b.cでoffsetのサイズを変更して比較せよ.

# 演習3

- 事前読込(プリフェッチ)処理
  - ex01.cではkについてのループで配列b[k][j]に毎回キャッシュミスが起こる. ex03.cでは配列b[k][j]について事前読込することで, キャッシュミスを低減させている.
  - ex01.cとex03.cを比較して事前読込処理についてベンチマーク.

※コンパイルオプションを”-O0”(最適化無し)としてコンパイルする.