流体学校でのCfCA計算機利用マニュアル

1　システムの概略

水沢キャンパスにある XD2000 と三鷹キャンパスにある解析サーバを使うので，それらとネットワークを含むシステムの概略を以下に載せた。ちなみに三鷹キャンパスには他の機材 (GP-PC, GPU クラスタ，ファイルサーバ) もあるが，今回は使わないので載せていない。

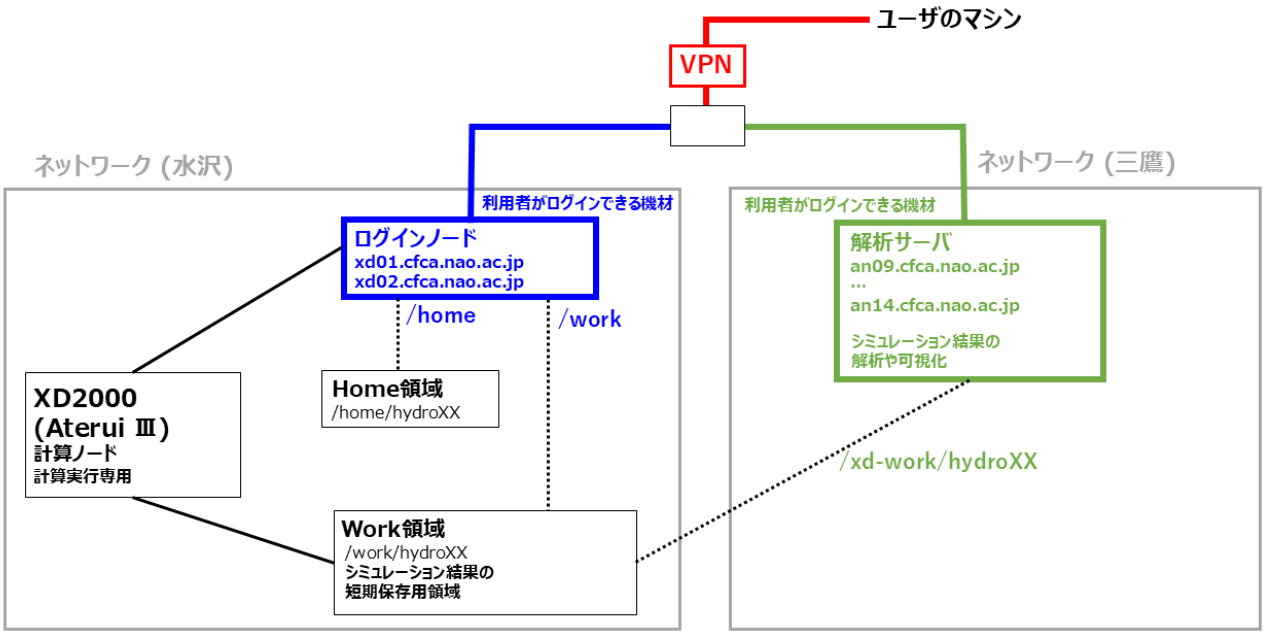

図 1: CfCA 計算機システムの概略 (XD2000 と解析サーバのみ抜粋)

XD2000 システムは複数の機材で構成されている。XD2000 本体の計算ノードには直接入れないようになっている。「XD2000 にログインする」とは，青字で書かれた XD2000 のログインノードに `ssh` 接続することを意味する。計算データの保存のために `work` 領域 (`/work/hydroXX`) があり，流体学校の実習ではこのディレクトリで作業をする。

データ解析は三鷹キャンパスにある解析サーバでおこなう。XD2000 の `work` 領域内のデータは，解析サーバに転送しなくても，解析サーバから直接読む込むことができる (書き換えることはできない)。例えば XD2000 において`/work/hydroXX/test` というファイルがあったとすると，解析サーバからは`/xd-work/hydroXX/test` として参照できる。

2　ジョブ管理システム

XD2000 を含む大規模並列計算機では，多くのユーザの様々な規模の計算 (ジョブと呼ぶ) を効率的に計算ノードに配分するジョブ管理システムが導入されている。XD2000 では `Slurm` と呼ばれるジョブ管理システムが使われている。ジョブの投入の仕方は第 4 章で説明する。

XD2000 のログインノードで直接計算を実行することは，他のユーザの迷惑になるので禁止されている。
解析サーバではジョブ管理システムを使っていないので，直接可視化や解析の計算を実行してよい。

2.1　流体学校用アカウントで使える計算資源

流体学校用アカウントでは，1 ジョブ当たり，計算ノードを 1 台 (112 コア) を 4 時間だけ使うことができる。アカウント名は `hydroXX` である。`XX`」には `09` など整数が入り，各個人で異なる。CfCA ホームページの個人ページに記載している。

- **流体学校の実習時間中** 5 ノードを流体学校用に占有している。参加者ではない他のユーザはその占有しているノードが使えない状態。1 ノード当たり 3 名くらい使う想定 (上限 32 並列くらい)。占有している時間帯は,
  - 3 月 24 日 13:00〜17:30
  - 3 月 25 日 13:00〜17:30
  - 3 月 26 日 13:00〜15:30

  である。
- **その他の時間**：流体学校参加者も使えるが，ノードは占有されない。計算ノードが既に予約されていたら，計算が走り出すまでに時間がかかる。

流体学校用アカウントは，3 月 31 日 9:00 まで有効。3 月 31 日 9:00 頃から XD2000 では定期メンテナンスが行われるのでログインできなくなる。それまでに必要なデータは手元のマシンなどにダウンロードしておくこと。

3　コマンド関連

3.1　基本コマンドの使い方

- `ls` ディレクトリ内のディレクトリやファイルを一覧表示
- `cd dir` ディレクトリ dir に移動
- `cd` ホームディレクトリへ移動 (`/home/hydroXX`)
- `pwd` 現在いるディレクトリを表示
- `mkdir dir` ディレクトリ dir を作成
- `rm file`　ファイル file を削除
- `rm -rf dir` ディレクトリ dir をその中身ごと強制的に削除
- `cp file1 file2` file1 を file2 にコピー
- `cp -r dir1 dir2` dir1 と dir2 にコピー
- `mv file1 file2` file1 を file2 に移動 (名前変更)
- `mv file1 dir1` file1 を dir1 の中に移動
- `head file` file の先頭の 10 行を表示
- `tail file` file の末尾の 10 行を表示
- `tail -f file` file が更新されるたびに表示し続ける。

3.2　ディレクトリを表す代表的なシンボル

- 「チルダ〜」 ホームディレクトリのこと
- 「ピリオド.」 カレントディレクトリのこと
- 「連続した 2 つのピリオド..」 カレントディレクトリの一つ上段のディレクトリ

3.3　知っておくと便利なこと

- **補完機能**
  コマンドの入力中やディレクトリ名・ファイル名の入力中に `Tab` キーを押すと，それまでに入力した文字以降に他と重複がなければ自動で補完され，重複があれば候補リストが表示される。

- **履歴機能**
  コマンド実行の履歴を辿ることができる。再度コマンドを入力しなくてよい。
  - `Control+p` または「↑」キーで前に
  - `Control+n` または「↓」キーで次に

  `history` コマンドも使える。
  - `history` と打つと，過去に実行したコマンドの一覧が表示
  - 履歴には番号が振られている。`!番号` という風に，ビックリマークのあとに実行したいコマンドに対応する番号を付けてエンターキーを押すと，そのコマンドが実行される。

4 ジョブ実行@XD2000

- **XD2000 にログイン**
  `ssh -Y -l hydroXX xd01.cfca.nao.ac.jp`
  または
  `ssh -Y -l hydroXX xd02.cfca.nao.ac.jp`
  ログインした際には home 領域にいる (/home/hydroXX)。
  コマンド pwd で現在するディレクトリを確認可能。

- **環境構築に必要なファイルをコピー**
  ここの作業は今回だけでよい。
  `cp /home/hydro00/{.bashrc,.bash_profile} ./`
  設定を有効にする。`source .bashrc`

- **work 領域に移動**
  作業は work 領域でおこなう。
  `cd /work/hydroXX`

- **ソースコード (圧縮ファイル) を/home/hydro00 からコピーして，展開**
  `cp /home/hydro00/NanoASURAhydro2024.tar.gz ./`
  `tar xfvz NanoASURAhydro2024.tar.gz`
  nanoasura というディレクトリができている。

- **nanoasura ディレクトリに移動**
  `cd nanoasura`
  NanoASURA のディレクトリ構造 (2 日目の SPMHD の実習で使うバージョンには，runs の中に MHD 用のディレクトリがある)

  - src: ソースコード群
    - setup: 初期条件設定ファイル群
  - runs: 作業ディレクトリ
    - shocktube: 1D shock tube
    - hydrostatic: 2D hydrostatic
    - kh: 2D Kelvin-Helmholtz test
    - rt: 2D Rayleigh-Taylor test
    - sedov: 3D Sedov-Taylor test
    - keplar: 2D Kepler disk test
    - galaxy: 2D barred galaxy model
    - evrard: Evrard test
  - plot: 計算結果を可視化するためのツール

- **コンパイル**
  `cd ./src` ソースコード群が入っているディレクトリに移動
  `make` コンパイル
  実行ファイル asura.out ができている。ls コマンドで確認
  ソースコードを変更した場合は，必ず再度コンパイル (make) をする。

- **作業ディレクトリに移動**
  衝撃波管問題を解く場合は,
  `cd ../runs/shocktube`
  に移動する。他の問題を解く場合には対応するディレクトリに移動する。

- **ジョブを投入**

`sbatch ./batch.sh`

batch.sh の中身

```
#!/bin/bash
#SBATCH --job-name=ryuutai
#SBATCH --partition=M-large-t
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=16
#SBATCH --hint=nomultithread
#SBATCH --mem=32G
#SBATCH --time=04:00:00
#SBATCH -o %x_%J.out
#SBATCH -e %x_%J.err

module list
export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK}
export KMP_AFFINITY=compact

cd $SLURM_SUBMIT_DIR
../../src/asura.out > log.out
```

各項目を以下で説明する。赤で示した項目は変更可能。

- `#SBATCH --job-name=`: ジョブの名前を指定する。
- `#SBATCH --partition=`: 使うパーティションを指定。パーティションによって使える計算資源が異なる。流体学校では `M-large-t` を指定。**これは変更しない。**
- `#SBATCH --nodes=`: 使用する計算ノードの数を指定する。今回は 1 台までしか使えないので，1 で固定。
- `#SBATCH --ntasks=`: 使用するプロセス数。今回は MPI 並列しないので，1 で固定。
- `#SBATCH --cpus-per-task=`: プロセス当たりに使用する CPU 数。今回は OpenMP を使って並列化するので，スレッド数に対応。今回は 32 までにしていく。
  `param.toml` の `[Thread]` にある `ThreadNumber` と数値を合わせる必要があるので注意
- `#SBATCH --hint=nomultithread`: ハイパースレッディング機能をオフにする。
- `#SBATCH --mem=`: ノード当たりに使うメモリ量。今回は 32Gbyte で固定。
- `#SBATCH --time=`: 計算時間の指定 (書式 時間: 分: 秒)。上限値はパーティションによって異なる。今回の上限は 4 時間。より大規模なジョブの実行が予約されているために，計算ノードが空いている場合，この値を短く設定すると，空きノードで計算が走る可能性がある。占有時間以外で計算がなかなか走らない場合は，この値を短めに設定するとよい。
- `#SBATCH -o %x_%J.out`: 標準出力内容を出力するファイル名を指定。`%x` には Job 名が，`%J` にはジョブ ID が入る。
- `#SBATCH -e %x_%J.err`: エラー文を出力するファイル名を指定。
- `module list` から `export` から始まる 2 行まではおまじない。
- `cd $SLURM_SUBMIT_DIR`: `SLURM_SUBMIT_DIR` は，ジョブが投入されたディレクトリ名が入っている。
- `../../src/asura.out`: 計算の実行

4.1 ジョブ管理システム Slurm の代表的なコマンドの使い方

- **ジョブを投入**

  ジョブスクリプトが batch.sh とすると，

  `sbatch batch.sh`

  ジョブを投入すると，割り振られた ID が画面上に表示される。以下は，2 回連続して計算を投入した例 (ID は 42549 と 42550)。

```
hydro16@xd01:/work/hydro16/nanoasura/runs/kh$ sbatch batch.sh
sbatch: [NAOJ XD2000] Requested node is 1. Nodes are shared.
Submitted batch job 42549
hydro16@xd01:/work/hydro16/nanoasura/runs/kh$ sbatch batch.sh
sbatch: [NAOJ XD2000] Requested node is 1. Nodes are shared.
Submitted batch job 42550
```

- **ジョブの状態を確認**

  hydroXX が投入したジョブの状態を確認するには，

  `squeue --me`

```
hydro16@xd01:/work/hydro16/nanoasura/runs/kh$ squeue --me
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
             42550 M-large-t  ryuutai  hydro16 PD       0:00      1 (QOSMaxJobsPerUserLimit)
             42549 M-large-t  ryuutai  hydro16  R       0:16      1 cna085
```

  専用アカウントでは同時に複数の計算が走らない設定になっている。そのため上の例では計算 42549 が開始している (状態 ST が R (Run の意)) が，計算 42550 は待ち状態になっている (状態 ST が PD (PenDing の意))。

  計算の進行具合を確認するには，

  `tail -f 出力ファイル`

  を実行する。上の例でのファイル名は ryuutai_42549.out。

  tail はファイルの末尾 10 行を表示するコマンドで，オプション-f を付けると，リアルタイムで更新する。出力を止めたい場合は，Control を押しながら c を押す。

- **ジョブのキャンセル**

  `scancel ジョブ ID`

  ジョブ ID は `squeue --me` の JOBID の項目で確認。

5　可視化@解析サーバ

計算結果の可視化は解析サーバでおこなう。図 1 のように，XD2000 の work ディレクトリは，解析サーバから直接読み込むことができる。

Windows ユーザの方は，MobaXterm でもう一度 Session をクリックして，解析サーバ an??.cfca.nao.ac.jp にログインする (??には 09 から 14 までの整数を入れる)。Mac ユーザの方は，XQuartz のウィンドウをもう一つ開いて，解析サーバに ssh を使ってログインする。

ログインしたら**環境構築に必要なファイルをコピー**する。この作業は今回だけでよい。
`cp /home/hydro00/.bashrc ./`
設定を有効にする。`source .bashrc`

5.1　出力ファイル

ファイルは ascii 形式で出力される。詳細は src/Log.c を参照のこと。出力されたファイルには，最初に header が，その後にデータが記載されている。header 部には以下のデータが載っている。

DISPH のフラグ (0 ならオフで，1 ならオン)
粒子数 (ThisRun.NParticles の値)
時刻 (t の値)
時間の規格化因子 (ThisRun.TNorm の値)
比熱比 (ThisRun.Gamma の値)

データ部は表になっている。各行に各粒子の以下のデータが並んでいる。`head ファイル名` で，ファイルの最初の 10 行を確認できる。

1 日目の SPH の場合

- 1 ∼ 3 列目：粒子の x, y, z 座標
- 4 ∼ 6 列目：粒子の v_x, v_y, v_z
- 7 列目：粒子の質量
- 8 列目：粒子の smoothing 長
- 9 列目：粒子の密度
- 10 列目：粒子の内部エネルギー
- 11 列目：粒子の圧力
- 12 列目：粒子の q
- 13 列目：近傍粒子数
- 14 列目：粒子のタグ番号

2 日目の SPMHD の場合

- 1 ∼ 3 列目：粒子の x, y, z 座標
- 4 ∼ 6 列目：粒子の v_x, v_y, v_z
- 7 ∼ 9 列目：粒子の B_x, B_y, B_z
- 10 列目：粒子の密度
- 11 列目：粒子の圧力
- 12 列目：粒子の全エネルギー
- 13 列目：粒子のスカラー場 ψ
- 14 列目：粒子の質量
- 15 列目：粒子の Smoothing 長
- 16 列目：粒子位置での $\boldsymbol{\nabla} \cdot \boldsymbol{B}$
- 17 列目：近接粒子の個数
- 18 列目：粒子のタグ番号

5.2 NanoASURA の可視化ツール

NanoASURA の可視化ツールを使って，簡単に図を作成することができる。内部では python を呼んでいる。

- **ソースコード (圧縮ファイル) を/home/hydro00 からコピーして，展開**
  解析サーバと XD2000 の home 領域は別の領域なので，解析サーバにも NanoASURA をコピーする必要がある。
  `cp /home/hydro00/NanoASURA_hydro2024.tar.gz ./`
  `tar xfvz NanoASURAhydro2024.tar.gz`
- **可視化用のプログラムが入ったディレクトリへ移動**
  `cd nanoasura/plot`
- **コンパイル**
  `make`
  実行ファイル./plot.out が生成される。
- **可視化プログラムを実行し，画像ファイルを生成**
  パラメータファイル param.toml を編集し，`./plot.out`を実行。
  - param.toml では，各テスト問題において，シミュレーション結果が出力されたディレクトリを指定する必要がある。DataDir に/xd-work/hydroXX/nanoasura/runs/の下にある対応するディレクトリを指定する。ディレクトリ名がわからない場合は，ls コマンドで確認する。
- **アニメーション作成と再生**
  作成
  `./run_ffmpeg.sh "./data_st_disph/shocktube_four.%04d.png" shocktube_disph.mp4`
  再生
  `ffplay {loop 100 ./shocktube_disph.mp4`

6　データ解析

ここは余力のあれば参照して下さい。

6.1　解析用 python スクリプト

ログファイルを読み込んで，解析をするための python スクリプトを nanoasura/runs に置いた。それぞれの作業ディレクトリにコピーして使うこと。

例えば，RunName が data で，RunName が kh の場合，0 番目から 10 番目のログファイルに対して解析したい場合は，

```
python3 analysis.py data kh 0 10
```

を実行する。ログファイルを格納する場所 Outdir とファイル名の接頭語 RunName は param.toml で設定されている。

データは dict 形式で格納している。dict 形式では，各データを”Key”と”Value”のペアで保存する。dict の名前を data とした場合のデータ一覧を以下に載せる。例えば data['time'] は時刻を，data['rho'][10] は，10 番目の粒子の密度を表す。

変数名	物理量	型
data['N']	粒子数	整数
data['time']	時刻	実数
data['dim']	次元	整数
data['gam']	比熱比	実数
data['x']	粒子の x 座標	numpy 1 次元配列
data['y']	粒子の y 座標	numpy 1 次元配列
data['z']	粒子の z 座標	numpy 1 次元配列
data['vx']	粒子の x 方向速度	numpy 1 次元配列
data['vy']	粒子の y 方向速度	numpy 1 次元配列
data['vz']	粒子の z 方向速度	numpy 1 次元配列
data['mass']	粒子の質量	numpy 1 次元配列
data['h']	粒子の Smoothing 長	numpy 1 次元配列
data['rho']	粒子の密度	numpy 1 次元配列
data['u']	粒子の内部エネルギー	numpy 1 次元配列
data['pre']	粒子の圧力	numpy 1 次元配列
data['Nlist']	粒子の近接粒子数	numpy 1 次元配列
data['Tag']	粒子の Tag 番号	numpy 1 次元配列

main 関数は以下になる。

```
1  ##################
2  #  main function
3  ##################
4  def main():
5      if len(sys.argv) != 5:
6          print("Error! Usage: python3 read.py OutDir RunName StartStep
               EndStep")
7          sys.exit(1)
8
9      outdir = sys.argv[1]
10     runname = sys.argv[2]
11     startstep = int(sys.argv[3])
12     endstep = int(sys.argv[4])
13
14     print("\nread from " + outdir + "/" + runname + ".%04d"%(startstep)
           + \
15             " to " + outdir + "/" + runname + ".%04d\n\n"%(endstep))
16
17     #open file to output the analysis results
18     outputfile = "output.dat"
19     fout = open(outputfile,"w")
20     # loop over startstep <= step <= endstep
21     for step in range(startstep,endstep+1):
22         # read data at step
23         data = read_data_file(outdir, runname, step )
24
25         # analyize using data at step
26         new0, new1 = analysis(data)
27
28         fout.write("{0} {1} {2}\n".format(data['time'], new0, new1))
29
30     fout.close()
31
32     return 0
```

21 行目から始まる loop で，指定した step 区間でデータを読み込み (read_data_file)，それを使って解析ができる (analysis)。サンプルプログラムでは，analysis で求めた二つの変数 new0 と new1 を受け取り，ファイル (output.dat) に出力している。

サンプルプログラムでの analysis(data) 関数を示す。

```
1  ##########################
2  #  use data to analyse
3  ##########################
4  def analysis(data):
5
6      newvariable0 = data['mass']*data['rho']
7      newvariable1 = data['pre']*data['vx']
8      new0 = np.sum( newvariable0[1:100])
9      new1 = np.average( newvariable1[1:40])
10
11     return new0, new1
```

関数 analysis(data) の中で解析をする。サンプルプログラムでは以下のような意味のない計算をしている。

- 6 行目と 7 行目：2 つの新しい配列 newvariable0 (質量×密度) と newvariable1 (圧力× v_x) を作成
- 8 行目： 1 番目から 99 番目の粒子の newvariable0 を合計した値を new0 に保存
- 9 行目： 1 番目から 39 番目の粒子の newvariable1 を平均した値を new1 に保存