Moller Documentation

リリース *1.0-dev*

ISSP, University of Tokyo

2023 年 12 月 28 日

目次

第 1 章

概要

1.1 moller とは?

近年、機械学習を活用した物性予測や物質設計 (マテリアルズインフォマティクス) が注目されています。機械学習の精度は、適切な教師データの準備に大きく依存しています。そのため、迅速に教師データを生成するためのツールや環境の整備は、マテリアルズインフォマティクスの研究進展に大きく貢献すると期待されます。

moller は、ハイスループット計算を支援するためのパッケージ HTP-Tools の一つとして提供しています。moller ではスーパーコンピュータやクラスタ向けにバッチジョブスクリプトを生成するツールであり、多重実行の機能を利用し、パラメータ並列など一連の計算条件について並列にプログラムを実行することができます。現状では、東京大学 物性研究所の提供するスーパーコンピュータ ohtaka (slurm ジョブスケジューラ) と kugui (PBS ジョブスケジューラ) がサポートされています。

1.2 ライセンス

1.3 開発貢献者

本ソフトウェアは以下の開発貢献者により開発されています。

- ver.1.0-beta (2023/12/28 リリース)
  - 開発者
    - 吉見 一慶 (東京大学 物性研究所)
    - 青山 龍美 (東京大学 物性研究所)

    - 本山 裕一 (東京大学 物性研究所)
    - 福田 将大 (東京大学 物性研究所)
    - 井戸 康太 (東京大学 物性研究所)
    - 福島 鉄也 (産業技術総合研究所)
    - 笠松 秀輔 (山形大学 学術研究院 (理学部主担当))
    - 是常 隆　(東北大学大学院理学研究科)
  - プロジェクトコーディネーター
    - 尾崎 泰助 (東京大学 物性研究所)

1.4 コピーライト

本ソフトウェアは 2023 年度 東京大学物性研究所 ソフトウェア高度化プロジェクトの支援を受け開発されており、その著作権は東京大学が所持しています。

1.5 動作環境

以下の環境で動作することを確認しています。

- Ubuntu Linux + python3

第 2 章

インストールと基本的な使い方

必要なライブラリ・環境

HTP-tools に含まれる網羅計算ツール moller を利用するには、以下のプログラムとライブラリが必要です。

- Python 3.x
- ruamel.yaml モジュール
- tabulate モジュール
- GNU Parallel (ジョブスクリプトを実行するサーバ・計算ノード上にインストールされていること)

ソースコード配布サイト

- GitHub リポジトリ

ダウンロード方法

git を利用できる場合は、以下のコマンドで moller をダウンロードできます。

```
$ git clone https://github.com/issp-center-dev/Moller.git
```

インストール方法

moller をダウンロード後、以下のコマンドを実行してインストールします。moller が利用するライブラリも必要に応じてインストールされます。

```
$ cd ./Moller
$ python3 -m pip install .
```

実行プログラム `moller` および `moller_status` がインストールされます。

ディレクトリ構成

```
.
|-- LICENSE
|-- README.md
|-- pyproject.toml
|-- docs/
|   |-- ja/
|   |-- en/
|   |-- tutorial/
|-- src/
|   |-- moller/
|       |-- __init__.py
|       |-- main.py
|       |-- platform/
|       |   |-- __init__.py
|       |   |-- base.py
|       |   |-- base_slurm.py
|       |   |-- base_pbs.py
|       |   |-- base_default.py
|       |   |-- ohtaka.py
|       |   |-- kugui.py
|       |   |-- pbs.py
|       |   |-- default.py
|       |   |-- function.py
|       |   |-- utils.py
|       |-- moller_status.py
|-- sample/
```

基本的な使用方法

moller はスーパーコンピュータ向けにバッチジョブスクリプトを生成するツールです。多重実行の機能を利用して、パラメータ並列など一連の計算条件について並列にプログラムを実行します。

1. 構成定義ファイルの作成

   moller を使用するには、まず、計算内容を記述した構成定義ファイルを YAML 形式で作成します。詳細についてはファイルフォーマットの章を参照してください。

2. コマンドの実行

   作成した構成定義ファイルを入力として moller プログラムを実行します。バッチジョブスクリプトが生成されます。

```
$ moller -o job.sh input.yaml
```

3. バッチジョブの実行

   生成されたバッチジョブスクリプトを対象となるスーパーコンピュータシステムに転送します。並列実行する各パラメータごとにディレクトリを用意し、 list.dat にディレクトリ名を列挙します。list.dat には、ジョブを実行するディレクトリからの相対パスまたは絶対パスを記述します。

   リストファイルが用意できたらバッチジョブを投入します。以下では、物性研システム B(ohtaka) およびシステム C(kugui) で実行するケースをそれぞれ示します。

   - 物性研システム B(ohtaka) の場合

     ohtaka では slurm ジョブスケジューラが使用されています。バッチジョブを投入するには、バッチジョブスクリプトを引数として sbatch コマンドを実行します。ジョブスクリプト名に続けてスクリプトのパラメータを渡すことができます。パラメータとしてリストファイルを指定します。

     ```
     $ sbatch job.sh list.dat
     ```

     リストファイルの指定がない場合は list.dat がデフォルトとして使われます。

   - 物性研システム C(kugui) の場合

     kugui では PBS ジョブスケジューラが使用されています。バッチジョブを投入するには、バッチジョブスクリプトを引数として qsub コマンドを実行します。スクリプトのパラメータの指定はできないので、リストファイルは list.dat として用意する必要があります。

     ```
     $ qsub job.sh
     ```

4. 結果の確認

   バッチジョブ終了後に、

   ```
   $ moller_status input.yaml list.dat
   ```

   を実行すると、各パラメータセットについて計算が正常に終了したかどうかを集計したレポートが出力されます。

5. ジョブの再開・再実行

   ジョブが途中で終わった場合、続けて実行するには、同じリストファイルを指定してもう一度バッチジョブを投入します。未実行 (未完了を含む) のタスクから実行が継続されます。

   - 物性研システム B(ohtaka) の場合

     以下のように、リストファイルを指定して sbatch コマンドを実行します。

```
$ sbatch job.sh list.dat
```

エラーで終了したタスクを再実行するには、--retry オプションを付けてバッチジョブを投入します。

```
$ sbatch job.sh --retry list.dat
```

- 物性研システム C(kugui) の場合

  job.sh を編集して retry=0 の行を retry=1 に書き換えた後、

  ```
  $ qsub job.sh
  ```

  を実行します。

参考文献

[1] O. Tange, GNU Parallel - The command-Line Power Tool, ;login: The USENIX Magazine, February 2011:42-47.

第 3 章

チュートリアル

3.1 基本的な使い方

網羅計算のためのバッチジョブスクリプト生成ツール moller を使うには、入力ファイルとして実行内容を記述する構成定義ファイルを用意した後、プログラム moller を実行します。生成されたバッチジョブスクリプトを対象とするスーパーコンピュータシステムに転送し、バッチジョブを投入して計算を行います。以下では、`docs/tutorial/moller` ディレクトリにあるサンプルを例にチュートリアルを実施します。

3.1.1 構成定義ファイルを作成する

構成定義ファイルにはバッチジョブで実行する処理の内容を記述します。ここで、バッチジョブとはスーパーコンピュータシステム等のジョブスケジューラに投入する実行内容を指します。それに対し、moller が対象とするプログラムの多重実行において、多重実行される一つのパラメータセットでの実行内容をジョブと呼ぶことにします。一つのジョブはいくつかの処理単位からなり、その処理単位をタスクと呼びます。moller ではタスクごとに多重実行し、タスクの前後で同期がとられます。

以下に構成定義ファイルのサンプルを記載します。構成定義ファイルは YAML フォーマットのテキストファイルで、実行するプラットフォームやバッチジョブのパラメータと、タスクの処理内容、前処理・後処理を記述します。

```
name: testjob
description: Sample task file

platform:
  system:  ohtaka
  queue:   i8cpu
  node:    1
  elapsed: 00:10:00

```

(次のページに続く)

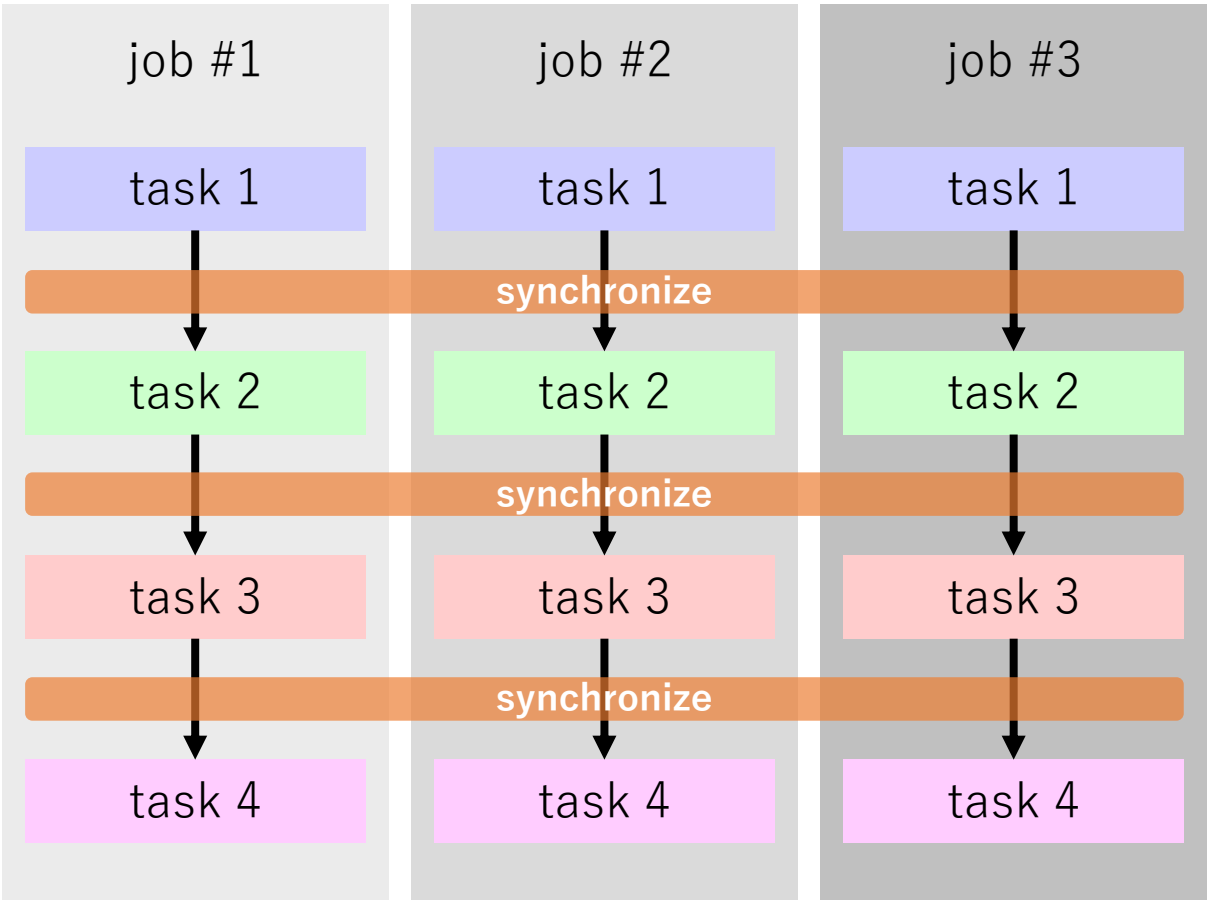


図 3.1　例: 一つのバッチジョブ内で job #1〜#3 の 3 つのジョブを実行する。ジョブはそれぞれ異なるパラメータセットなどに対応する。ジョブの実行内容は task 1〜4 の一連のタスクからなる。タスクごとに job #1〜#3 の処理を並列に行う。

(前のページからの続き)

```
prologue:
  code: |
    module purge
    module load oneapi_compiler/2023.0.0 openmpi/4.1.5-oneapi-2023.0.0-classic

    ulimit -s unlimited

    source /home/issp/materiapps/intel/parallel/parallelvars-20210622-1.sh

jobs:
  start:
    parallel: false
    run: |
```

(次のページに続く)

(前のページからの続き)

```
    echo "start..."

  hello:
    description: hello world
    node: [1,1]
    run: |
      echo "hello world." > result.txt
      sleep 2

  hello_again:
    description: hello world again
    node: [1,1]
    run: |
      echo "hello world again." >> result.txt
      sleep 2

epilogue:
  code: |
    echo "done."
    date
```

platform セクションでは、実行するプラットフォームの種類を指定します。この場合は、物性研システム B(ohtaka) での設定をしています。

prologue セクションでは、バッチジョブの前処理を記述します。タスクを実行する前に実行する共通のコマンドラインを記述します。

jobs セクションでは、タスクの処理内容を記述します。ジョブで実行する一連のタスクを、タスク名をキー、処理内容を値として記述するテーブルの形式で記述します。

この例では、最初に"start..."を出力するタスクを start というタスク名で定義しています。ここでは `parallel = false` に設定しています。この場合、ジョブ単位での並列は行われず、run に記述した内容が逐次的に実行されます。

次に、"hello world."を出力するタスクを hello world というタスク名で定義しています。ここでは `parallel` が設定されていないので、 `paralle = true` として扱われます。この場合、ジョブ単位での並列が行われます。同様に、次に "hello world again." を出力するタスクを hello_again というタスク名で定義しています。

最後に、epilogue セクションでは、バッチジョブの後処理を記述します。タスクを実行した後に実行する共通のコマンドラインを記述します。

仕様の詳細については **ファイルフォーマット** の章を参照してください。

3.1.2 バッチジョブスクリプトを生成する

構成定義ファイル (input.yaml) を入力として moller を実行します。

```
$ moller -o job.sh input.yaml
```

バッチジョブスクリプトが生成され出力されます。出力先は構成定義ファイル内のパラメータ、または、コマンドラインの -o または --output オプションで指定するファイルです。両方指定されている場合はコマンドラインパラメータが優先されます。いずれも指定がない場合は標準出力に書き出されます。

必要に応じて moller で生成したバッチジョブスクリプトを対象のシステムに転送します。なお、スクリプトの種類は bash スクリプトです。ジョブ実行時に使用するシェルを bash に設定しておく必要があります。(ログインシェルを csh 系などにしている場合は注意)

3.1.3 リストファイルを作成する

実行するジョブのリストを作成します。moller では、ジョブごとに個別のディレクトリを用意し、そのディレクトリ内で各ジョブを実行する仕様になっています。対象となるディレクトリのリストを格納したファイルを、たとえば以下のコマンドで、リストファイルとして作成します。

```
$ /usr/bin/ls -1d > list.dat
```

チュートリアルには、データセットとリストファイルを作成するユーティリティープログラムが付属しています。

```
$ bash ./make_inputs.sh
```

を実行すると、 output ディレクトリの下にデータセットに相当する dataset-0001 〜 dataset-0020 のディレクトリと、リストファイル list.dat が作成されます。

3.1.4 網羅計算を実行する

moller で生成したバッチジョブスクリプトをジョブスケジューラに投入します。この例ではジョブスクリプトと入力ファイルを output ディレクトリにコピーし、 output に移動してジョブを投入しています。

```
$ cp job.sh input.yaml output/
$ cd output
$ sbatch job.sh list.dat
```

ジョブが実行されると、リストに記載されたディレクトリにそれぞれ "result.txt" というファイルが生成されます。"result.txt" には、ジョブ実行結果の "hello world.", "hello world again." という文字列が出力されていることが確認できます。

3.1.5 実行状況を確認する

タスクの実行状況はログファイルに出力されます。ログを収集してジョブごとに実行状況を一覧するツール moller_status が用意されています。ジョブを実行するディレクトリで以下を実行します。

```
$ moller_status input.yaml list.dat
```

引数には構成定義ファイル input.yaml とリストファイル list.dat を指定します。リストファイルは省略可能で、その場合はログファイルからジョブの情報を収集します。

出力サンプルを以下に示します。

```
| job          | hello   | hello_again   |
|--------------|---------|---------------|
| dataset-0001 | o       | o             |
| dataset-0002 | o       | o             |
| dataset-0003 | o       | o             |
| dataset-0004 | o       | o             |
| dataset-0005 | o       | o             |
| dataset-0006 | o       | o             |
| dataset-0007 | o       | o             |
| dataset-0008 | o       | o             |
| dataset-0009 | o       | o             |
| dataset-0010 | o       | o             |
| dataset-0011 | o       | o             |
| dataset-0012 | o       | o             |
| dataset-0013 | o       | o             |
| dataset-0014 | o       | o             |
| dataset-0015 | o       | o             |
| dataset-0016 | o       | o             |
| dataset-0017 | o       | o             |
| dataset-0018 | o       | o             |
| dataset-0019 | o       | o             |
| dataset-0020 | o       | o             |
```

「o」は正常終了したタスク、「x」はエラーになったタスク、「-」は前のタスクがエラーになったためスキップされたタスク、「.」は未実行のタスクを示します。今回は全て正常終了していることがわかります。

3.2 HPhi による *moller* 計算の例

3.2.1 このチュートリアルについて

これは、量子多体問題の正確な対角化方法を実行するためのオープンソースソフトウェアパッケージである HPhi を用いた moller の例です。この例では、周期境界条件下の $S = 1/2$ (2S_1 ディレクトリ) と $S = 1$ (2S_2) 反強磁性ハイゼンベルク鎖の励起ギャップ Δ のシステムサイズ依存性を計算します。moller を使用することで、異なるシステムサイズの計算を並列に実行します。これは HPhi 公式チュートリアルの セクション 1.4 に対応しています。

3.2.2 準備

moller （HTP-tools）パッケージと HPhi がインストールされていることを確認してください。このチュートリアルでは、ISSP のスーパーコンピュータシステム ohtaka を使用して計算を実行します。

3.2.3 実行方法

1. データセットを準備する

   2S_1, 2S_2 に含まれるスクリプト make_inputs.sh を実行します。

   ```
   $ bash ./make_inputs.sh
   ```

   L_8, L_10, ..., L_24 (2S_2 の場合は L_18 まで) の作業ディレクトリが生成されます。ディレクトリのリストは list.dat ファイルに書き込まれます。さらに、作業ディレクトリからエネルギーギャップを集めるためのシェルスクリプト、 extract_gap.sh が生成されます。

2. moller を使用してジョブスクリプトを生成する

   input.yaml からジョブスクリプトを生成し、 job.sh というファイル名で保存します。

   ```
   $ moller -o job.sh input.yaml
   ```

3. バッチジョブを実行する

   ジョブリストを引数としてバッチジョブを送信します。

   ```
   $ sbatch job.sh list.dat
   ```

4. 状態を確認する

   タスク実行の状態は moller_status プログラムによって確認できます。

```
$ moller_status input.yaml list.dat
```

5. 結果を集める

   計算が終了した後、ジョブからエネルギーギャップを以下のようにして集めます。

```
$ bash extract_gap.sh
```

   このスクリプトは、長さ L とギャップ Δ のペアをテキストファイル `gap.dat` に書き込みます。

   結果を視覚化するために、Gnuplot ファイル `gap.plt` が利用可能です。このファイルでは、得られたギャップデータが予想される曲線によってフィットされます。

$$\Delta(L; S = 1/2) = \Delta_\infty + A/L \tag{3.1}$$

   および

$$\Delta(L; S = 1) = \Delta_\infty + B\exp(-CL). \tag{3.2}$$

   グラフは次のコマンドで描画できます。

```
$ gnuplot --persist gap.plt
```

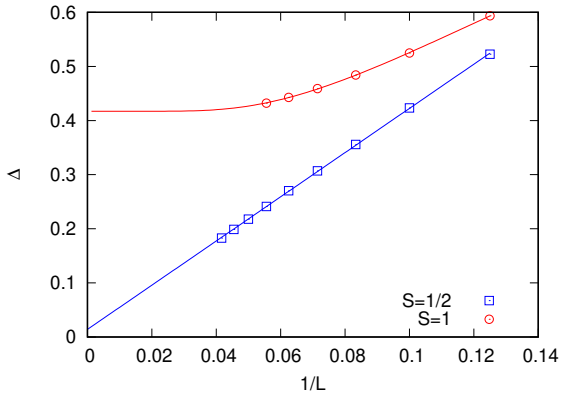


図 3.2　スピンギャップの有限サイズ効果

$S = 1/2$ の場合、対数補正によりスピンギャップは有限のままです。一方で、$S = 1$ の場合、外挿値 $\Delta_\infty = 0.417(1)$ は以前の結果（例えば、QMC による $\Delta_\infty = 0.41048(6)$ （Todo and Kato, PRL **87**, 047203 (2001)））とよくあっています。

3.3 DSQSS による *moller* 計算の例

3.3.1 このチュートリアルについて

これは、量子多体問題の経路積分モンテカルロ法を実行するためのオープンソースソフトウェアパッケージである DSQSS を用いた moller の例です。この例では、周期境界条件下の $S = 1/2$ (DSQSS の用語では $M = 1$) および $S = 1$ ($M = 2$) 反強磁性ハイゼンベルク鎖の磁気感受率 χ の温度依存性を計算します。 moller を使用することで、異なるパラメーター (M, L, T) の計算を並列に実行します。

この例は 公式チュートリアルの一つ に対応しています。

3.3.2 準備

moller （HTP-tools）パッケージと DSQSS がインストールされていることを確認してください。このチュートリアルでは、ISSP のスーパーコンピュータシステム ohtaka を使用して計算を実行します。

3.3.3 実行方法

1. データセットを準備する

   このパッケージに含まれるスクリプト make_inputs.sh を実行します。

   ```
   $ bash ./make_inputs.sh
   ```

   これにより、 output ディレクトリが作成されます（すでに存在する場合は、まず削除し、再度作成します)。 output の下には、各パラメーター用の作業ディレクトリ（例： L_8__M_1__T_1.0）が生成されます。ディレクトリのリストは list.dat ファイルに書き込まれます。

2. moller を使用してジョブスクリプトを生成する

   ジョブ記述ファイルを使用してジョブスクリプトを生成し、 job.sh というファイル名で保存します。

   ```
   $ moller -o job.sh input.yaml
   ```

   次に、job.sh を output ディレクトリにコピーし、 output ディレクトリに移動します。

3. バッチジョブを実行する

   ジョブリストを引数としてバッチジョブを送信します。

   ```
   $ sbatch job.sh list.dat
   ```

4. 状態を確認する

タスク実行の状態は moller_status プログラムによってまとめられます。

```
$ moller_status input.yaml list.dat
```

5. 結果を集める

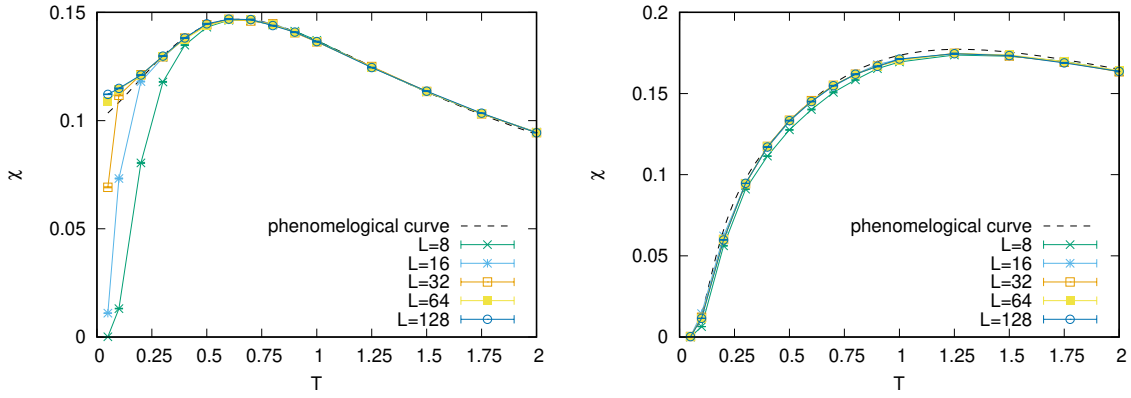
計算が終了した後、結果を以下のようにして集めます。

```
$ python3 ../extract_result.py list.dat
```

このスクリプトは、M, L, T, χ の平均、および χ の標準誤差を含む 5 列のテキストファイル result.dat に結果を書き込みます。

結果を視覚化するために、GNUPLOT ファイル plot_M1.plt および plot_M2.plt が利用可能です。

```
$ gnuplot --persist plot_M1.plt
$ gnuplot --persist plot_M2.plt
```



$S = 1/2$ と $S = 1$ AFH 鎖の主な違いは、励起ギャップが消失するか ($S = 1/2$)、残るか ($S = 1$) のどちらかです。これを反映して、非常に低温領域での磁気感受率は、有限になる ($S = 1/2$) か、消失する ($S = 1$) かのどちらかです。$S = 1/2$ の場合には、有限サイズ効果によりスピンギャップが開き、そのため小さいチェーンの磁気感受率が低下します。

第 4 章

コマンドリファレンス

4.1 moller

網羅計算のためのバッチジョブスクリプトを生成する

書式:

```
moller [-o job_script] input_yaml
```

説明:

input_yaml に指定した構成定義ファイルを読み込み、バッチジョブスクリプトを生成します。以下のオプションを受け付けます。

- -o, --output job_script

  出力先のファイル名を指定します。構成定義ファイル内の output_file パラメータより優先されます。ファイル名の指定がない場合は標準出力に書き出されます。

- -h

  ヘルプを表示します。

4.2 moller_status

網羅計算ジョブの実行状況をレポートする

書式:

```
moller_status [-h] [--text|--csv|--html] [--ok|--failed|--skipped|--collapsed|--
↪yet] [-o output_file] input_yaml [list_file]
```

説明:

moller で生成したジョブスクリプトを実行した際に、ジョブごとの各タスクが完了したかどうかを集計してレポートを作成します。input_yaml に指定した構成定義ファイルからタスクの内容を読み込みます。ジョブのリストは list_file に指定したファイルから取得します。list_file が指定されていないときは、実行時ログファイルから収集します。出力形式をオプションで指定できます。デフォルトはテキスト形式です。出力先を -o または --output オプションで指定します。指定がない場合は標準出力に書き出されます。

- 出力モード

  出力形式を指定します。以下のいずれかを指定できます。複数同時に指定した場合はエラーになります。デフォルトはテキスト形式です。

  - --text テキスト形式で出力します。
  - --csv CSV (カンマ区切りテキスト) 形式で出力します。
  - --html HTML 形式で出力します。

- input_yaml

  moller の構成定義ファイルを指定します。

- list_file

  ジョブのリストを格納したファイルを指定します。指定がない場合は、バッチジョブから出力されるログファイル log_{task}.dat から収集します。

- -o, --output output_file

  出力先のファイル名を指定します。指定がない場合は標準出力に書き出されます。

- フィルタ

  出力内容を指定します。以下のいずれかを指定できます。指定がない場合は全てのジョブの情報が出力されます。

  - --ok 全てのタスクが完了したジョブのみを表示します。
  - --failed エラー、スキップまたは未実行のタスクがあるジョブを表示します。
  - --skipped 実行をスキップしたタスクがあるジョブを表示します。
  - --yet 未実行のタスクがあるジョブを表示します。
  - --collapsed エラー終了したタスクがあるジョブを表示します。
  - --all 全てのジョブを表示します。(デフォルト)

- -h

  ヘルプを表示します。

ファイル:

moller で生成したジョブスクリプトを用いてプログラムを並列実行すると、実行状況がログファイル log_{task}.dat に出力されます。moller_status はこのファイルを集計し、読みやすい形式に整形します。

第 5 章

ファイルフォーマット

5.1 構成定義ファイル

構成定義ファイルでは、moller でバッチジョブスクリプトを生成するための設定情報を YAML 形式で記述します。本ファイルは以下の部分から構成されます。

1. 全般的な記述: ジョブ名や出力ファイル名などを設定します。
2. platform セクション: バッチジョブを実行するシステムやバッチジョブに関する設定を記述します。
3. prologue, epilogue セクション: バッチジョブ内で行う環境設定や終了処理などを記述します。
4. jobs セクション: タスクを記述します。

5.1.1 全体

`name`

: バッチジョブのジョブ名を指定します。指定がない場合は空欄となります。(通常はジョブスクリプトのファイル名がジョブ名になります)

`description`

: バッチジョブの説明を記述します。コメントとして扱われます

`output_file`

: moller の出力先ファイル名を指定します。コマンドライン引数の指定がある場合はコマンドライン引数の指定を優先します。いずれも指定がない場合は標準出力に出力されます。

5.1.2 platform

`system`

対象となるシステムを指定します。現状では ohtaka と kugui が指定できます。

`queue`

使用するバッチキューの名称を指定します。キューの名称はシステムに依存します。

`node`

使用するノード数を指定します。指定方法は ノード数 (整数値) または [ノード数, ノードあたりのコア数] (整数値のリスト) です。数値の範囲はシステムとキューの指定に依存します。(ノードあたりのコア数の指定は kugui,default のみ有効。ohtaka の場合は使われません。)

`core`

1 ノードあたり使用するコア数を指定します。数値の範囲はシステムとキューの指定に依存します。`node` パラメータに同時にノードあたりのコア数が指定されている場合、 `core` の指定が優先します。(kugui,default のみ)

`elapsed`

バッチジョブの実行時間を指定します。書式は HH:MM:SS です。

`options`

その他のバッチジョブオプションを指定します。書式は、ジョブスクリプトのオプション行の内容をリスト形式または複数行からなる文字列で記述したものです。各行の冒頭の指示語 (`#PBS` や `#SBATCH` など) は含めません。以下に例を示します。

- SLURM の場合 (文字列で指定する例)

```
options: |
  --mail-type=BEGIN,END,FAIL
  --mail-user=user@sample.com
  --requeue
```

- PBS の場合 (リストで指定する例)

```
options:
  - -m bea
  - -M user@sample.com
  - -r y
```

5.1.3 prologue, epilogue

prologue セクションはタスク開始前に実行する内容を記述します。ライブラリやパスなど環境変数の設定等を行うのに利用できます。epilogue セクションは全タスク終了後に実行する内容を記述します。

`code`

処理内容をシェルスクリプトの記法で記述します。記述内容はバッチジョブスクリプト中に埋め込まれてバッチジョブ内で実行されます。

5.1.4 jobs

ジョブで実行する一連のタスクを、タスク名をキー、処理内容を値として記述するテーブルの形式で記述します。

キー

タスク名

値

以下の項目からなるテーブル:

`description`

タスクの説明を記述します。コメントとして扱われます。

`node`

並列度を指定します。指定方法は以下のいずれかです。

- [プロセス数, プロセスあたりのスレッド数]
- [ノード数, プロセス数, プロセスあたりのスレッド数]
- ノード数

ノード数を指定した場合、その数のノードが排他的にジョブに割り当てられます。ノード数を指定しない 1 番目の形式の場合、使用コア数が 1 ノードに満たないときは複数のジョブがノードに詰めて割当られます。1 ノード以上を使う場合は必要ノード数を占有して実行されます。

`parallel`

ジョブ間で多重実行する場合は `true`, 逐次実行する場合は `false` を指定します。デフォルトは `true` です。

`run`

タスクの処理内容をシェルスクリプトの記法で記述します。MPI プログラムまたは MPI/OpenMP ハイブリッドプログラムを実行する箇所は

```
srun prog [arg1, ...]
```

と記述します。 srun の他に mpirun, mpiexec のキーワードが有効です。このキーワードは、実際のバッチジョブスクリプト中では、並列実行のためのコマンド (srun や mpirun) と node パラメータで指定した並列度の設定に置き換えて記述されます。

5.2 リストファイル

ジョブのリストを指定します。ファイルはテキスト形式で、一行に一つのジョブ名を記述します (ディレクトリ名がジョブ名となります)。

moller では、ジョブごとにディレクトリを用意し、ジョブ内の各タスクはディレクトリに移動して実行されます。ディレクトリはバッチジョブを実行するディレクトリの直下に配置されているものと仮定します。

第 6 章

拡張ガイド

(註: 以下の内容は moller のバージョンによって変わる可能性があります。)

6.1 moller によるバルク実行

バルク実行とは、大型のバッチキューに投入した一つのバッチジョブの中で、複数の小さいタスクを並行して実行するというものです。動作のイメージとしては、次のように N 個のタスクをバックグラウンドで実行し同時に処理させ、wait 文によりすべてのタスクが終了するまで待ちます。

```
task param_1 &
task param_2 &
     ...
task param_N &
wait
```

このとき、バッチジョブに割り当てられたノード・コアを適宜分配し、param_1〜param_N のタスクがそれぞれ別のノード・コアで実行されるように配置する必要があります。また、多数のタスクがある時に、割当てリソースに応じて最大 N 個のタスクが実行されるよう実行を調整することも必要です。

moller で生成したジョブスクリプトを以下では moller script と呼ぶことにします。moller script では、タスクの並行実行と制御には GNU parallel [1] を利用します。GNU parallel は param_1〜param_N のリストを受取り、これらを引数としてコマンドを並行して実行するツールです。以下は GNU parllel を使った実行イメージで、list.dat の各行に param_1〜param_N を列挙しておきます。

```
cat list.dat | parallel -j N task
```

同時実行数については、バッチジョブに割当てられたノード数・コア数を実行時に環境変数等から取得し、各タスクの並列度 (ノード数・プロセス数・スレッド数) の指定 (node パラメータ) を元に計算します。

ノード・コアへのタスクの配置についてはジョブスケジューラによって方法が異なります。SLURM 系のジョブス

ケジューラでは、リソースの排他利用のオプションを使うことで、バッチジョブ内部で発行された複数の srun をジョブスケジューラが適宜配置して実行します。具体的な指定方法はプラットフォームの設定に依存します。

一方、PBS 系のジョブスケジューラはそのような仕組みがなく、リソースの配分を moller script 内部で処理する必要があります。moller script では、バッチジョブに割り当てられた計算ノードとコアをスロットに分割し、GNU parallel で並行処理されるタスクに分配します。スロットへの分割は、実行時に取得される割当てノード・コアとタスクの並列度指定から計算し、テーブルの形で保持します。タスク内部では、mpirun (mpiexec) の引数や環境変数を通じて計算ノードの指定と割当コアのピン留めを行いプログラムを実行します。この処理は使用する MPI 実装に依存します。

参考文献

[1] O. Tange, GNU Parallel - The command-Line Power Tool, ;login: The USENIX Magazine, February 2011:42-47.

6.2 moller の動作について

6.2.1 moller で生成されるスクリプトの構成

moller は、入力された YAML ファイルの内容をもとに、バルク実行のためのジョブスクリプトを生成します。生成されるジョブスクリプトは先頭から順に次のような構成になっています。

1. ヘッダ

   ジョブスケジューラへの指示が記述されます。platform セクションに指定した内容が、ジョブスケジューラの種類に応じた形式に整形されて出力されます。この処理はプラットフォーム依存です。

2. プロローグ

   prologue セクションに指定した内容です。code ブロックの中身がそのまま転記されます。

3. 関数の定義

   ジョブスクリプト内部で使う関数および変数の定義が出力されます。関数の概要については次節で説明します。この箇所はプラットフォーム依存です。

4. コマンドライン引数の処理

   SLURM 系のジョブスケジューラでは、リストファイルの指定やタスクの再実行などのオプション指定を sbatch コマンドの引数として与えることができます。PBS 系のジョブスケジューラでは引数の指定は無視されるため、オプション指定はジョブスクリプトを編集してパラメータをセットする必要があります。

5. タスクの記述

   jobs セクションに記述されるタスクの内容を出力します。タスクが複数ある場合はタスクごとに以下の処理を実行します。

parallel = false の場合は run ブロックの中身がそのまま転記されます。

parallel = true (デフォルト) の場合、task_タスク名 という関数が生成され、並列実行のための前処理と run ブロックの内容が出力されます。並列計算のためのキーワード (`srun` 、 `mpiexec` または `mpirun`) はプラットフォームに応じたコマンドに置き換えられます。関数定義に続いて並列実行のコマンドが書き出されます。

6. エピローグ

   epilogue セクションに指定した内容です。code ブロックの中身がそのまま転記されます。

6.2.2 moller script の関数の概要

moller script の内部で使用する主な関数の概要を以下に説明します。

- `run_parallel`

  タスクの内容を記述した関数 (タスク関数) を並行実行する関数です。並列度、タスク関数、ステータスファイル名を引数に取ります。内部では `_find_multiplicity` 関数を呼んで多重度を計算し、GNU parallel を起動してタスクを並行実行します。GNU parallel の多段処理に対応するために、タスク関数は `_run_parallel_task` 関数でラップされます。

  プラットフォーム依存性は `_find_multiplicity` および `_setup_run_parallel` 関数としてくくり出しています。

- `_find_multiplicity`

  並列実行の多重度を、割当てリソース (ノード数・コア数) とタスクの並列度指定から計算します。PBS 系のジョブスケジューラでは、さらに計算ノード・コアをスロットに分割し、テーブルで保持します。実行時に環境から取得する情報は次の通りです。

  - SLURM 系

    割当てノード数 _nnodes
    : `SLURM_NNODES`

    割当てコア数 _ncores
    : `SLURM_CPUS_ON_NODE`

  - PBS 系

    割当てノード _nodes[]
    : `PBS_NODEFILE` で指定されるファイルから計算ノードのリストを取得

    ノード数 _nnodes
    : _nodes[] の項目数

割当てコア数 _ncores

以下の順に検索されます。

* NCPUS (PBS Professional)
* OMP_NUM_THREADS
* platform セクションの core 指定 (スクリプト中に moller_core 変数として書き込まれる)
* ヘッダの ncpus または ppn パラメータ

- _setup_run_parallel

  GNU parallel による並行実行を開始する前にいくつか処理を追加するために呼ばれます。PBS 系ではスロットに分割されたノード・コアのテーブルをタスク関数から参照できるよう export します。SLURM 系では実行する内容はありません。

各タスクに対応するタスク関数の構成については次の通りです。

- タスク関数の引数は 1) 並列度指定 (ノード数・プロセス数・スレッド数) 2) 実行ディレクトリ 3) GNU parallel のスロット ID です。
- _setup_taskenv で実行環境の設定を行います。この関数はプラットフォーム依存です。PBS 系ではスロット ID に基づいて計算ノード・コアをテーブルから取得します。SLURM 系では実行する内容はありません。
- 直前に実行するタスクが正常終了したかどうかを _is_ready 関数を呼んでチェックします。正常終了している場合はタスクの処理を継続します。それ以外の場合は -1 のステータスでタスクの処理を中断します。
- code ブロックの内容を転記します。その際に、並列計算のためのキーワード (srun 、 mpiexec または mpirun) はプラットフォームに応じたコマンドに置き換えられます。

6.3 moller を他のシステムで使うには

moller には現在、物性研スーパーコンピュータシステム ohtaka および kugui 向けの設定が用意されています。moller を他のシステムで使うための拡張ガイドを以下で説明します。

6.3.1 クラス構成

moller の構成のうちプラットフォーム依存の部分は `platform/` ディレクトリにまとめています。クラス構成は次のとおりです。

```
Platform (base.py)
 |
 +-- BaseSlurm (base_slurm.py) ------- Ohtaka (ohtaka.py)
 |
 +-- BasePBS (base_pbs.py) --+-------- Kugui (kugui.py)
 |                           |
 |                           `-------- Pbs (pbs.py)
 |
 `-- BaseDefault (base_default.py) --- DefaultPlatform (default.py)
```

プラットフォームの選択についてはファクトリが用意されています。`register_platform(`**登録名，クラス名**`)` でクラスをファクトリに登録し、`platform/__init__.py` にクラスを import しておくと、入力パラメータファイルの platform.system パラメータに指定できるようになります。

6.3.2 SLURM 系ジョブスケジューラ

SLURM 系のジョブスケジューラを利用している場合、BaseSlurm クラスを元にシステム固有の設定を行います。並列計算を実行するキーワードを置き換える文字列は `parallel_command()` メソッドの戻り値で与えます。リソースの排他利用を行うための srun のパラメータをここに指定します。具体例は ohtaka.py を参照してください。

6.3.3 PBS 系ジョブスケジューラ

PBS 系のジョブスケジューラ (PBS Professional, OpenPBS, Torque など) を利用している場合、BasePBS クラスを元にシステム固有の設定を行います。

PBS 系ではバッチジョブのノード数の指定の仕方に 2 通りあり、PBS Professional は select=N:ncpus=n という書式で指定しますが、Torque などは node=N:ppn=n と記述します。後者の指定を用いる場合は `self.pbs_use_old_format = True` をセットします。

計算ノードのコア数は node パラメータで指定できますが、対象システムを限定してコア数のデフォルト値を設定しておくこともできます。kugui.py ではノードあたり 128 コアを設定しています。

6.3.4 細かいカスタマイズが必要な場合

基底クラスを参照して必要なメソッドを再定義します。メソッド構成は次のようになっています。

- `setup`

  platform セクションのパラメータの取り出しなどを行います。

- `parallel_command`

  並列計算のキーワード (`srun`, `mpiexec`, `mpirun`) を置き換える文字列を返します。

- `generate_header`

  ジョブスケジューラオプションの指定を記述したヘッダを生成します。

- `generate_function`

  moller script 内部で使用する関数の定義を生成します。変数および関数の実体はそれぞれ以下のメソッドで作られます。

  - `generate_variable`
  - `generate_function_body`

  それぞれの関数は埋め込み文字列としてクラス内で定義されています。

6.3.5 新しいタイプのジョブスケジューラに対応させるには

moller script の動作のうちプラットフォーム依存な箇所は、並行実行の多重度の計算、リソース配置に関する部分、並列計算のコマンドです。

- 割当てノード・ノード数・ノードあたりのコア数を実行時に環境変数等から取得する方法
- 並列計算を実行するコマンド (mpiexec 等) と、実行ホストやコア割当の指定のしかた

これらをもとに moller script 内で使う関数を作成します。`printenv` コマンドでジョブスクリプト内で有効な環境変数の一覧を取得できます。

6.3.6 トラブルシューティング

moller script 内の _debug 変数を 1 にセットすると、バッチジョブ実行時にデバッグ出力が書き出されます。もしジョブがうまく実行されないときは、デバッグ出力を有効にして、内部パラメータが正しくセットされているかを確認してみてください。