ProteinDFの実習

東京大学 生産技術研究所
平野 敏行

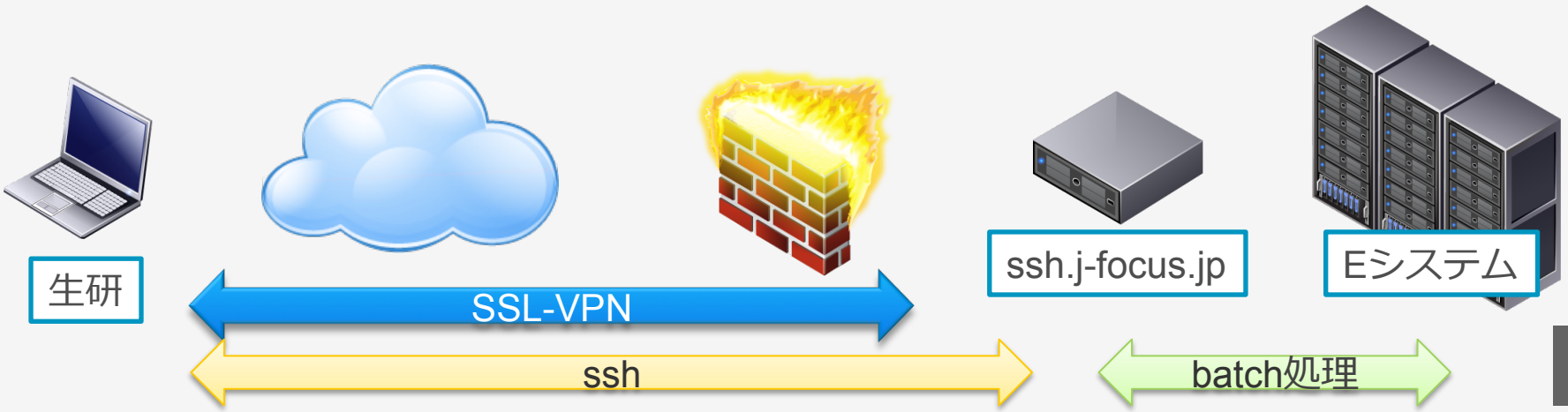
Agenda

- FOCUSスパコンの利用準備
- ProteinDFの演習
  - ProteinDFのインストール
  - ProteinDFの実行
  - 計算結果の可視化
- QCLO法の演習
  - QCLObotの概略
  - QCLObotの実行

FOCUSスパコンへのログイン

FOCUSスパコン接続の概要

- SSL-VPN接続
  - 要: account / password
- sshでssh.j-focus.jpに接続
  - 要: account / password



SSL-VPN接続

- マニュアル: http://www.j-focus.jp/sslvpn/
  - Cisco AnyConnect Secure Mobility Client により接続
- 概要
  1. https://vpn.j-focus.jp/ を開く
  2. GROUPは”Focus”, 配布の *USERNAME/PASSWORD*を入力してLogin
  3. Cisco AnyConnect Secure Mobility Clientをダウンロード・インストール
  4. Cisco AnyConnect Secure Mobility Clientを起動
  5. “vpn.j-focus.jp”に配布の *USERNAME/PASSWORD*を入力して接続
     - 緑色のチェックマークが点けば接続完了

ssh 接続

- Tera Term (Windows), terminal (MacOSX) などのsshクライアントソフトで
  `ff01.j-focus.jp`
  または
  `ff02.j-focus.jp`
  に接続
- 配布の *USERNAME/PASSWORD* で接続
- 正しく接続されていることを確認

ファイルの転送方法

- GUI
  - Windows
    - WinSCP
    - FileZilla Client
  - MacOSX
    - Cyberduck
    - ForkLift
- CUI
  scp または sftp コマンドを利用

プログラミング環境の選択(1)

- module コマンド
  - 現在のmodule環境の表示

    ```
    module list
    ```

  - module の読み込み

    ```
    module load モジュール名
    ```

  - 利用可能なmoduleの表示

    ```
    module avail
    ```

  - moduleの解除

    ```
    module unloadモジュール名
    ```

プログラミング環境の選択(2)

- Intel コンパイラ・Intel MPIの利用

```
module load PrgEnv-intel
module load impi411
```

- GNU gcc, OpenMPIの利用

```
module load PrgEnv-gnu482
module load gnu/openmpi165
```

Eシステム概要

ハードウェア	Cray GreenBlade GB824X (HPC 専用 ブレード型サーバ)
CPU	Intel Xeon E5-2670 v2 (2.5GHz) ×2CPU(計20コア)/ノード
コプロセッサ	Intel Xeon Phi 5110P ×4基 (計240コア)/ノード
メモリ	128 GB/ノード
インターフェース	Infiniband-FDR(56Gbps)×1/ノード

バッチシステム

- ff01, ff02からバッチシステムにより Eシステムを利用
- 演習で利用できるキュー
  - pdfseminar

本演習で使用できるのは全体で16ノードです。
利用はお一人様1ノード(20コア)でお願いします。

バッチシステムのコマンド

- キュー情報の確認

  `sinfo -s`
- 利用可能なノード数の確認

  `freenodes`
- ジョブの投入

  `sbatch ジョブスクリプト`
- ジョブ状態の表示

  `squeue`
- ジョブのキャンセル

  `scancel ジョブID`

ジョブスクリプトの例

```
#!/bin/bash
#SBATCH -p pdfseminar
#SBATCH -n 20
#SBATCH -N 1
#SBATCH -J ALA1
#SBATCH -o stdout.%J
#SBATCH -e stderr.%J

module load PrgEnv-intel
module load impi411

export OMP_NUM_THREADS=20
export PDF_HOME=/home1/glej/share/ProteinDF.intel

${PDF_HOME}/bin/PDF.x 2>&1 > PDF.log
```

キューの指定

ジョブ全体でのプロセス数

ノード数

ジョブの名前(任意)

標準出力の保存先

標準エラー出力の保存先

インストール

ProteinDFビルドの流れ

1. ソースの入手
2. ProteinDF_bridgeのインストール
3. ProteinDF_pytoolsのインストール
4. ProteinDFのインストール
    1. configureスクリプトの実行
    2. ビルド

ProteinDFソースコードの入手

- githubから入手
  - https://proteindf.github.io/

ProteinDF_bridge

- ProteinDFやその他アプリケーション間のデータ連携を図るデータコンテナとアルゴリズム集
- 機能
  - 各種ファイル形式(pdb, xyz, ...)の読み書き
  - プログラミング言語間のデータ授受
  - 2次構造・イオン対・SS結合等の探索
  - モデリング
  - 可視化基本インターフェース

ProteinDF_bridgeのインストール

- python版bridgeのインストール

```
$ python setup.py build
```

```
$ python setup.py install --prefix=${PDF_HOME}
```

- 環境変数PYTHONPATHの設定

– PYTHONPATH: pythonモジュールの探索パス

```
export PYTHONPATH=${PDF_HOME}/lib[64]/python2.7/site-packages
```

ProteinDF bridge クラスの紹介

- Atomクラス
  - 原子情報
- AtomGroupクラス
  - 原子団を表現
  - 複数のAtomを包含
  - 複数のAtomGroupを包含(入れ子)
- Selecterクラス
  - AtomGroupに対する原子選択・集合演算
- Modelingクラス
  - AtomGroupに対して分子モデリング操作
- Superimposeクラス
  - 重ねあわせ処理 (RMSD)

ProteinDF_pytools

- ProteinDFのデータ解析等を扱うpythonモジュール＋スクリプト
  - 膨大なデータを処理するプログラムを簡単に作成するために用意されたモジュール
- 機能
  - 行列・ベクトルファイルの読み書き
  - 軌道データの処理
  - 計算データの可搬化・アーカイブ化
  - 計算結果の可視化

ProteinDF_pytoolsのインストール

- bridgeと同じ方法

```
$ python setup.py build
```

```
$ python setup.py install --prefix=${PDF_HOME}
```

- 環境変数PYTHONPATHの設定

– PYTHONPATH: pythonモジュールの探索パス

```
export PYTHONPATH=${PDF_HOME}/lib[64]/python2.7/site-packages
```

ProteinDFインストールの準備

- 環境変数PDF_HOMEの設定
  - ProteinDFはPDF_HOME以下にインストール

ProteinDFのビルドに必要なもの (1)

- シリアル(逐次)実行に必要なもの
  - C++コンパイラ
    - GNU G++, Intel C++, PGI C++, *etc*.
  - LAPACKライブラリ
    - Netlib LAPACK, Intel MKL, AMD ACML, *etc*.
- パラレル(並列)実行に必要なもの
  - MPIライブラリ
    - OpenMPI, MPICH2, Intel MPI, *etc*.
  - ScaLAPACKライブラリ

ProteinDFのビルドに必要なもの (2)

- `configure`スクリプトの作成に必要なもの
  - autoconf
  - automake
  - libtool

configureスクリプトの作成

- configureスクリプトが
  用意されていない場合、
  自分で作成する必要あり
- ソース内にあるbootstrap.shを実行

configureスクリプトの実行

- configureオプションの表示

  ```
  $ ./configure --help
  ```

- 重要な変数

CXX	C++コンパイラの指定
MPICXX	MPIC++コンパイラの指定
CXXFLAGS	C++コンパイラへのオプション (OpenMP等の指示)

- 重要なオプション

--prefix=[dir]	インストール先(*cf.* ${PDF_HOME})
--with-blas=[lib]	BLASライブラリの場所
--with-lapack=[lib]	LAPACKライブラリの場所
--with-scalapack=[lib]	ScaLAPACKライブラリの場所

configureでのトラブル(1)

- configureの出力を保存する

  ```
  $ configure 2>&1 | tee out.configure
  ```

- configureがエラーで止まった場合
  - config.logにエラーの原因が記載
- ライブラリの認識を確認

```
checking for sgemm_ in -mkl=cluster... yes
checking for cheev_ in -mkl=cluster... yes
checking for mpic++... mpiicpc
checking for MPI_Init... yes
checking for mpi.h... yes
checking for sl_init_ in -mkl=cluster... yes
```

BLAS

LAPACK

MPI

ScaLAPACK

configureでのトラブル(2)

- Intel MKLライブラリのリンク方法
  - https://software.intel.com/en-us/articles/intel-mkl-link-line-advisor

ProteinDFのビルド

- makeの実行
  - トラブル対策のために出力保存を推奨

```
$ make 2>&1 | tee out.make
```

  - FOCUSでは約8分程度
    - -j オプションで並行コンパイルも可

```
$ make -j 4 2>&1 | tee out.make
```

- インストール

```
$ make install 2>&1 | tee
out.make_install
```

[補足] tarballの作成

- `bootstrap.sh`が実行できない(configureスクリプトが作成できない)環境には、予め`configure`スクリプトを用意したtarball(tar形式の圧縮ファイル)を作成

```
$ make dist
```

【演習】ProteinDF環境の構築

- 演習に必要な環境変数を設定してください。

```
source /home1/glej/share/setup_PDF.sh
```

– 備考

- 環境変数PDF_HOMEに /home1/glej/share/ProteinDF.intel を指定しています。
- Intelコンパイラ+Intel MPIでビルドした環境です。
- /home1/glej/share/Python27 に Python 2.7.8を構築済みです。

ProteinDFの実行

ProteinDF実行への準備

- 計算ディレクトリの作成
- 入力ファイル(fl_Userinput)の作成
- 作業用ディレクトリ(fl_Work)の作成
- 環境変数の確認
  - OMP_NUM_THREADS:
    1プロセスあたりのOpenMPスレッド数
  - OMP_SCHEDULE:
    OpenMPスレッドの並列処理方法

ProteinDFの実行方法

- 逐次(シリアル)版の実行
  - 実行ファイル: `${PDF_HOME}/bin/PDF.x`
  - OpenMPを有効にしている場合、OpenMPによるスレッド並列が有効
- MPI並列(パラレル)版の実行
  - 実行ファイル: `${PDF_HOME}/bin/PPDF.x`
  - OpenMPを有効にしている場合、hybrid並列が有効
  - 実行方法はMPI環境に依存

入力ファイルの概要(1)

- `step_control`

  計算内容を指示

  – `create`: 入力ファイル解析(obsolete)

  – `integral`: pre-SCF処理

  – `guess`: 初期電子密度の作成

  – `scf`: SCF計算

  – `force`: 力計算

入力ファイルの概要(2)

- method
  - rks:　閉殻計算
  - uks:　開殻計算
  - roks:　制限付き開殻計算
- method/rks/electron-number: 電子数
- method/rks/occlevel: 占有軌道
  - [1-5, 8-10] のような記述も可能

入力ファイルの概要(3)

- scf_start_guess:　初期値の指定
  - harris:　Harris汎関数法
  - density_matrix: user指定密度行列
    (guess.density.rks.mat)
  - lcao:　user指定LCAO行列
    (guess.lcao.rks.mat, guess.occ.rks.vtr)

入力ファイルの概要(4)

- xc_functional:　　交換相関汎関数
  - HF, SVWN, BLYP, B3LYP
- J_engine:　　クーロン項計算法
- K_engine:　　Fock交換項計算法
  - conventional
  - RI
  - CD
- XC_engine:　　pureDFT XC項計算法
  - grid, gridfree, gridfree_CD

入力ファイルの概要(5)

- geometry/cartesian/input
  - 座標をカーテシアンで入力
  - ダミー原子(電荷)の入力方法:
    X　　[x座標] [y座標] [z座標] [電荷]
  - ラベル機能
    [原子] @[ラベル] [x座標] [y座標] [z座標]

入力ファイルの概要(6)

- 基底関数の入力
  - basis-set/orbital:
    基底関数
  - basis-set/density-auxiliary:
    RI補助基底関数(クーロン項)
  - basis-set/exchange-auxiliary:
    RI補助基底関数(交換相関項)
  - basis-set/gridfree:
    gridfree法補助基底関数(交換相関項)

- 基底関数セットは${PDF_HOME}/data/basis2に記載

[演習] ProteinDFの計算

- shareディレクトリ(/home1/glej/share)に演習で使用するファイルが置いてあります。
- ProteinDF_samplesディレクトリ以下にサンプルがありますので、自分のディレクトリにコピーして使用してください。
- サンプル内のバッチスクリプト(run_serial.sh)をバッチシステムに投入(sbatch)して動作を確認してください。

```
sbatch run_serial.sh
```

計算結果の可視化

収束結果の表示

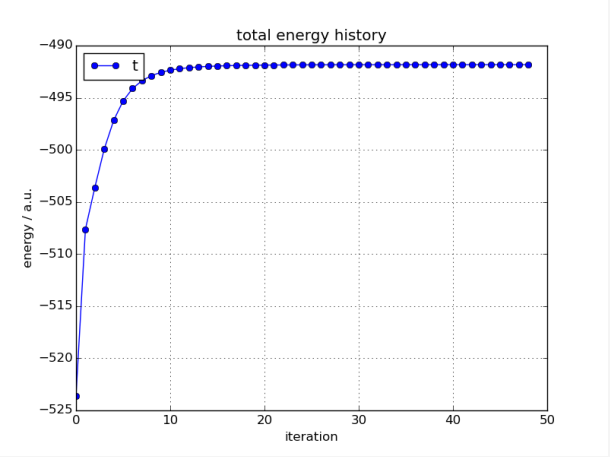
- 計算結果のアーカイブ

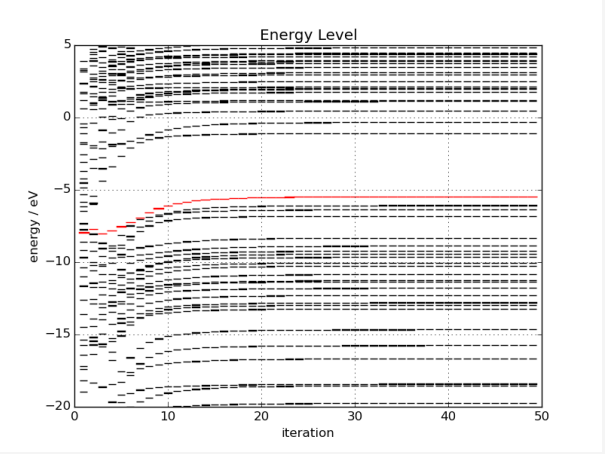
  `$PDF_HOME/bin/pdf  archive`

  – pdfresults.db が出力
- 計算レポートの作成

  `$PDF_HOME/bin/pdf  report`

  – reportディレクトリ中にグラフが出力





分子軌道(MO)の描画

- MOボリュームデータの作成
  – 計算ディレクトリにて実行

```
$PDF_HOME/bin/pdf-mkfld-mo -c cubeファイル　MO番号 …
```

- AVSフィールドデータ等が作成可能
- cubeファイルはChimera 等で表示可能

電子密度・静電ポテンシャルの描画

- 電子密度ボリュームデータの作成

  `$PDF_HOME/bin/pdf-mkfld-dens -c cube`ファイル

- ESPボリュームデータの作成

  `$PDF_HOME/bin/pdf-mkfld-esp -c cube`ファイル

- AVSフィールドデータ等が作成可能
- cubeファイルはChimera 等で表示可能

タンパク質可視化ソフトウェア

- UCSF Chimera
  - https://www.cgl.ucsf.edu/chimera/
  - Windows, MacOSX, Linuxで動作
  - Volume Dataの表示方法
    1. [Tools] -> [Volume Data] -> [Volume Viewer]
       - Volume Viewerダイアログが表示
    2. [File] -> [Open Map…]
       - cubeファイルを開く

[演習] ProteinDF計算結果の可視化

- ProteinDFを計算したディレクトリ上で作業します。

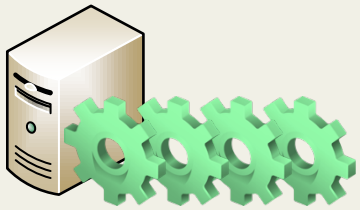
1. `${PDF_HOME}/bin/pdf-archive.py` で計算結果DBを作成します。
2. `${PDF_HOME}/bin/pdf-report.py` で収束グラフを作成します。
   - 作成されたグラフをローカルにコピーして表示します。
3. 分子軌道、電子密度、静電ポテンシャルの volume data (cubeファイル)を作成します。
   - Chimera がインストールされていればPDBファイルとcubeファイルをローカルにコピーして可視化できます。

注意事項

ProteinDFの動作様式

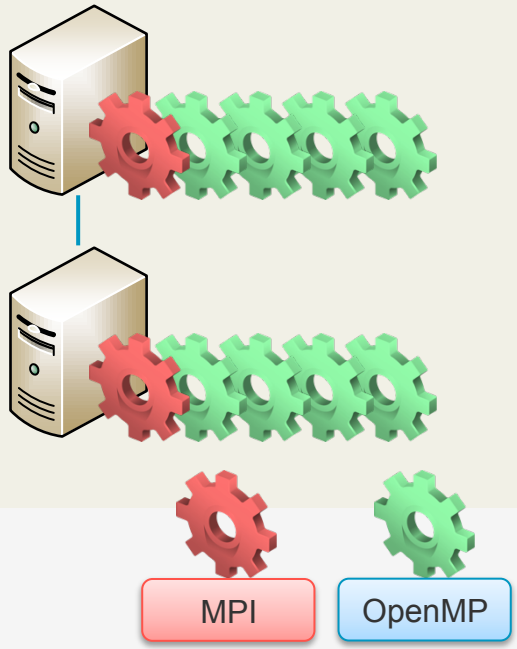
シリアル(逐次)版

- 単一プロセス実行
  - OpenMPスレッド並列
  - ノード内並列のみ



パラレル(並列)版

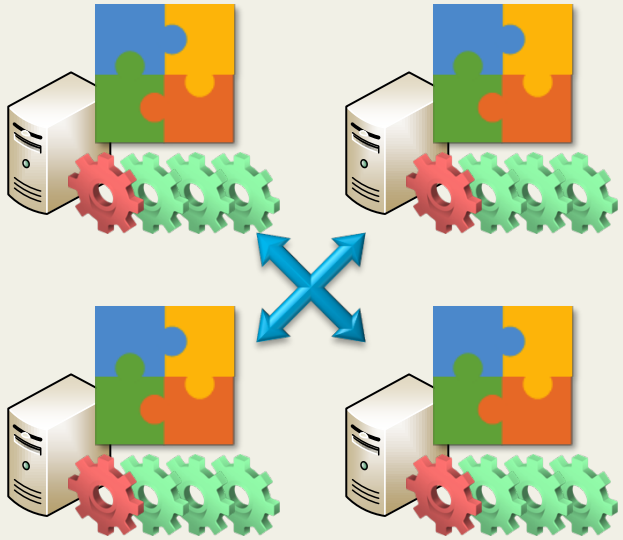
- 複数プロセス実行
  - MPI+OpenMP hybrid 並列
  - ノード間通信可能



ProteinDFの行列保持方法

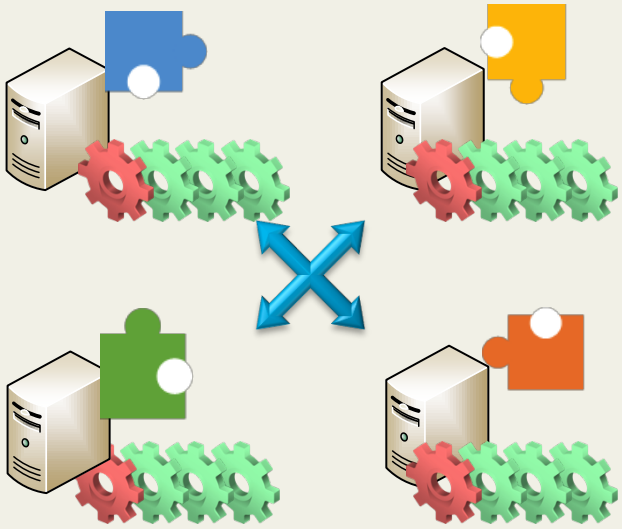
replica

- 各ノードに行列を保持
  - LAPACKで演算



distributed

- 全ノードに行列を分散
  - ScaLAPACKで演算



ProteinDFの並列処理方法

divide-and-conquer法

- タスク全体を
  処理単位が分担して行う
  並列処理方法
- すべての処理単位が
  処理に参加
- タスクの均等化が難しい
- ProteinDFキーワード

```
parallel_processing_type = DC
```

master-slave法

- 1つのmaster(プロセス)が
  多数のslave(プロセス)を
  制御する並列処理方法
- タスクが均等化しやすい
- masterプロセスが必要
- ProteinDF キーワード

```
parallel_processing_type = MS
```

線形従属性の排除

- 巨大な分子を扱う場合、
  また広がりの大きな基底関数を使う場合、
  線形従属の問題が発生する
- → 重なり積分**S** の固有値が
  小さいものを排除
- ProteinDFでは
  `orbital_independence_threshold`
  がその閾値

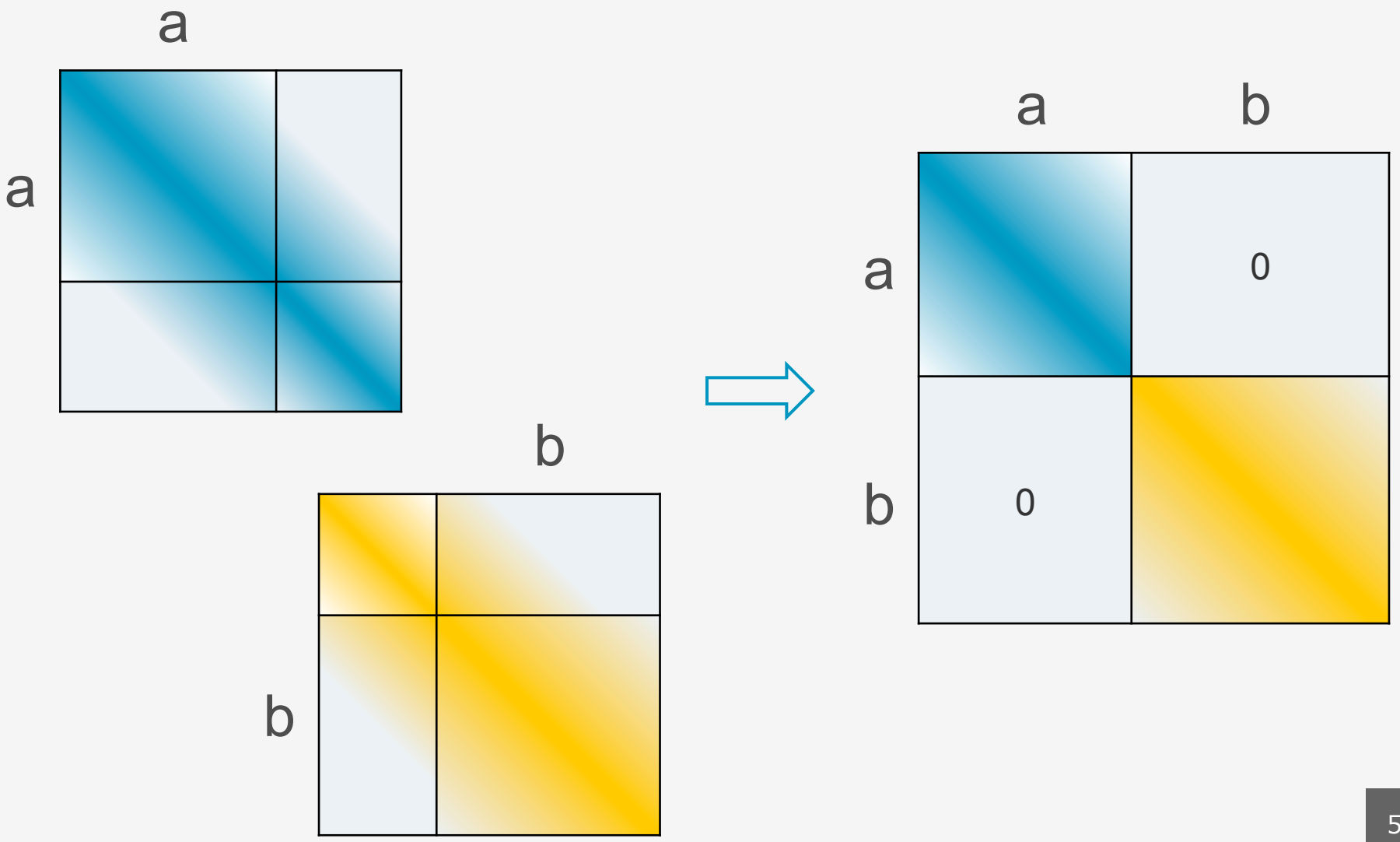
QCLO法による初期値の作成

QCLObotの概要

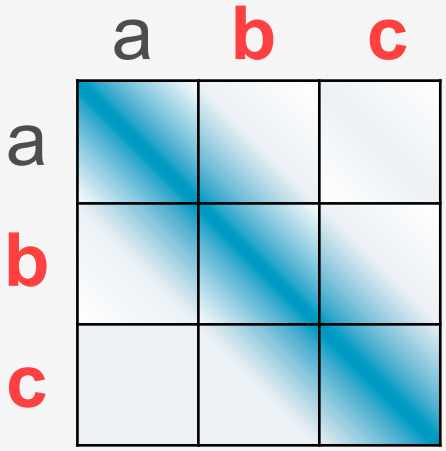
QCLO法

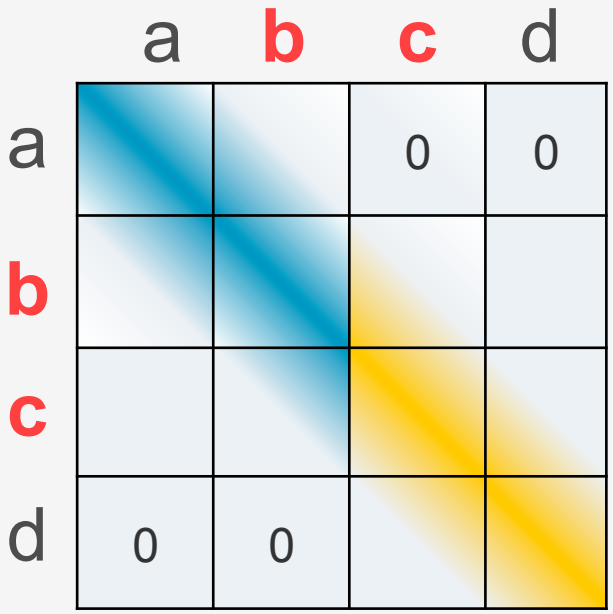
- QCLOを徐々に伸展させながら
大規模分子の高精度な初期値を求める方法
- 用語
  - Quasi-Canonical Localize Orbital (QCLO):
    - 擬カノニカル局在化軌道
    - カノニカル軌道と局在化軌道の両方の性質をあわせ持つ
  - フレーム分子:
    - 量子化学計算の計算単位
  - フラグメント:
    - フレーム分子は1つ以上のフラグメントから構成
    - QCLOを作成する単位
  - 計算シナリオ:
    - QCLO法における計算手順 (伸展方法)
    - 重なり部分を持たせながら伸長する独特な方法

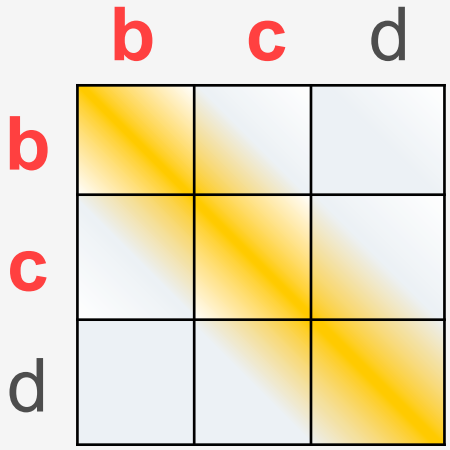
電子密度行列による初期値作成

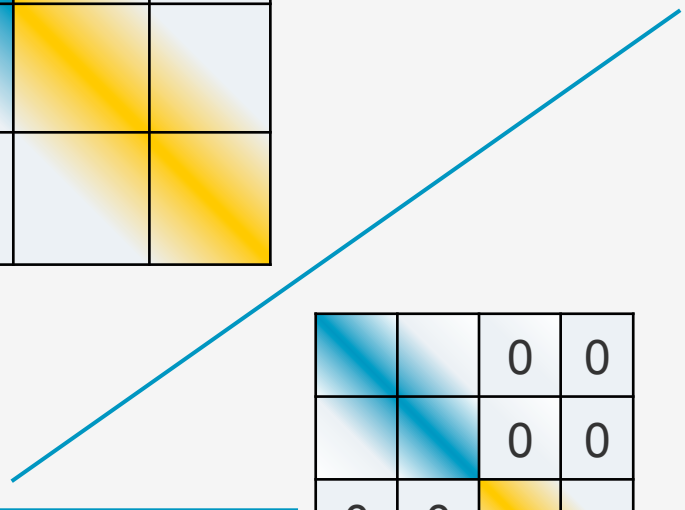


QCLO法による初期値作成





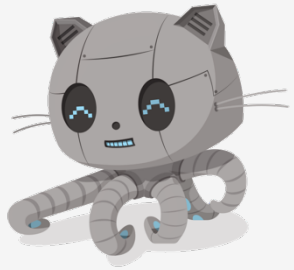




密度行列だと

QCLObotの概要

- QCLO法に基づく
(全)自動量子化学計算プログラム
  - QCLO + bot, QC + robot
- 特徴
  - 煩雑なQCLO計算を隠蔽
  - 任意のフラグメントに分割可能
    - タンパク質以外の巨大分子に適用可
  - Pythonで記述
    - Python 2.x と 3.x サポート
  - オープンソース: GPL v3



QCLObot のインストール

- ソースコード
  - https://github.com/ProteinDF/QCLObot
- 一般的なPythonモジュールと同じ方法でインストール

```
$ python setup.py build
```

```
$ python setup.py install --prefix=${PDF_HOME}
```

  - 環境変数PYTHONPATHの設定をお忘れなく

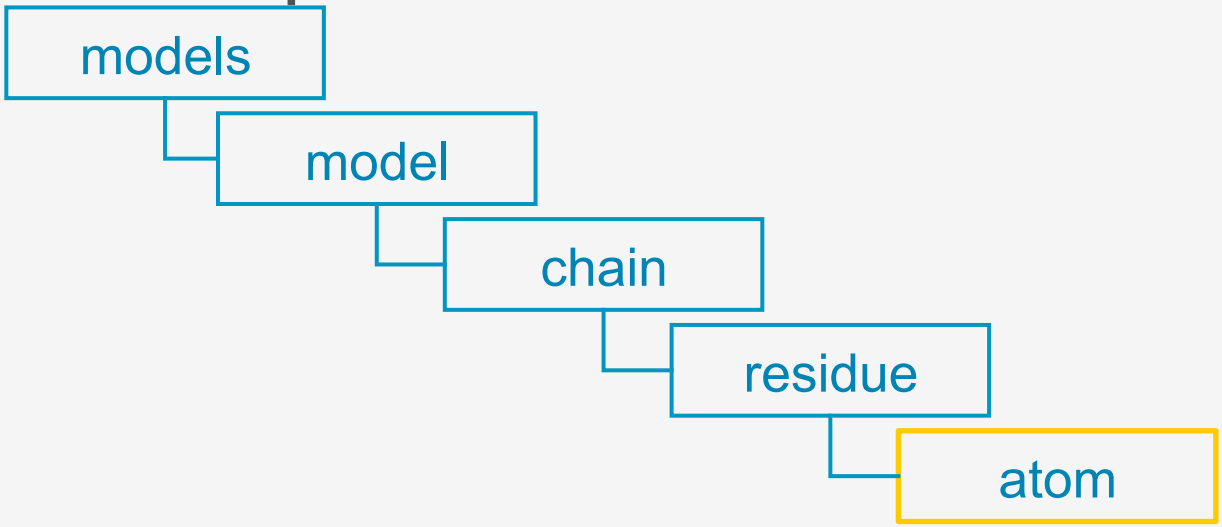
QCLObot モジュール

- QcFrameクラス
  - 複数のQcFragmentを包含
- QcFragmentクラス
  - 複数のQcAtomを包含
  - 複数のQcFragmentを包含(入れ子)
  - 分子構造はProteinDF_bridgeの AtomGroupから構築可能
- QcAtomクラス
  - 原子1つの位置・基底関数等の情報を保持

PDBファイルからのAtomGroup作成

- QCLObotのために、
  PDBからAtomGroupオブジェクトを作成
- pdb2brd.py でPDBファイルを変換

  pdb2brd.py 入力PDBファイル 出力brdファイル
- AtomGroupツリー構造



QCLObot の計算 [演習]

- モデリング済み1UAO_1H.mod.pdb中3残基を計算

GLY	TYR	ASP

- N末, C末にはACE基, NME基を付加
  - ペプチド結合を計算に取り込むため
- step 1では1残基ごとに計算
- step 2では2通りの初期値をテスト
  - 電子密度行列による初期値
  - QCLO法による初期値

QCLO法計算スクリプトの例 (1)

```
import pdfbridge as bridge
import pdfpytools as pdf
import qclobot as qclo

def main():
    brdfile = open("1UAO_1H.mod.brd", 'rb')
    brddata = msgpack.unpackb(brdfile.read())
    brdfile.close()

    models = bridge.AtomGroup(brddata)
    model = models["model_1"]
    chain = model["A"]

    frames = {}

    step1(frames, chain)
    step2_density(frames, chain)
    step2_QCLO(frames, chain)
```

brdファイルの
読み込み

AtomGroupツリー構造から
chainの読み込み

計算済みフレーム分子記憶用

計算シナリオ

QCLO法計算スクリプトの例 (2)

```
def step1(frames, chain):
    modeling = bridge.Modeling()
    max_resid = chain.get_number_of_groups()

    for resid in range(1, max_resid +1):
        frg_res = qclo.QcFragment(chain[resid])

        ACE = qclo.QcFragment()
        if resid > 1:
            prev = chain[resid -1]
            ACE_atmgrp = modeling.get_ACE_simple(prev)
            ACE = qclo.QcFragment(ACE_atmgrp)

        NME = qclo.QcFragment()
        if resid < max_resid:
            ahead = chain[resid +1]
            NME_atmgrp = modeling.get_NME_simple(ahead)
            NME = qclo.QcFragment(NME_atmgrp)
(続く)
```

1残基ごとのループ

空のQcFragment

N末側にACE基を付加

bridge.AtomGroupを返す

QcFragmentを作成

C末側にNME基を付加

QCLO法計算スクリプトの例 (3)

(続き)

```
        name = 'res_{}-{}'.format(resid, resid)
        frame = qclo.QcFrame(name = name)
        frame[name] = frg_res
        frame['ACE'] = ACE
        frame['NME'] = NME

        frames[name] = frame
        setup_calc_conf(frame.pdfparam)

        frame.calc_sp(dry_run = False)

        # for guess_density
        frame.pickup_density_matrix()

        # for guess_QCLO
        frame.calc_lo()
        frame.pickup_lo()
```

3つのフラグメントからなるフレーム分子を作成

← フレーム分子トポロジーの記憶

← 計算条件の設定

フレーム分子のsingle point計算

次step用の準備(密度行列用)

次step用の準備(QCLO用)

QCLO法計算スクリプトの例 (4)

```
# 密度行列を初期値とした3残基の計算
def step2_density(frames, chain):
    frame = qclo.QcFrame(name = 'res_1-3.density')
    frame['res_1-1'] = frames['res_1-1']['res_1-1']
    frame['res_2-2'] = frames['res_2-2']['res_2-2']
    frame['res_3-3'] = frames['res_3-3']['res_3-3']
    frame['ACE'] = frames['res_1-1']['ACE']
    frame['NME'] = frames['res_3-3']['NME']

    setup_calc_conf(frame.pdfparam)
    frame.guess_density()

    frame.calc_sp(dry_run = False)
```

フレーム分子

フラグメント

3残基フレーム分子の構築

密度行列による初期値を作成

QCLO法計算スクリプトの例 (5)

```
# QCLOを初期値とした3残基の計算
def step2_QCLO(frames, chain):
    frame = qclo.QcFrame(name = 'res_1-3.QCLO')
    frame['res_1-1'] = frames['res_1-1']['res_1-1']
    frame['res_2-2'] = frames['res_2-2']['res_2-2']
    frame['res_3-3'] = frames['res_3-3']['res_3-3']
    frame['ACE'] = frames['res_1-1']['ACE']
    frame['NME'] = frames['res_3-3']['NME']

    setup_calc_conf(frame.pdfparam)

    frame.calc_preSCF(dry_run = False)

    frame.guess_QCLO()

    frame.calc_sp(dry_run = False)
```

この名前のディレクトリが作成される

QCLO作成には
重なり積分が必要

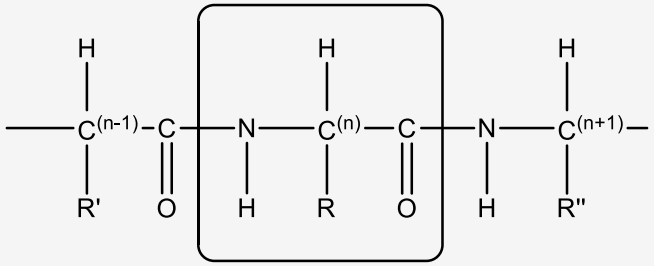
QCLOによる初期値を作成

なぜStep2でQCLO法が使えるのか

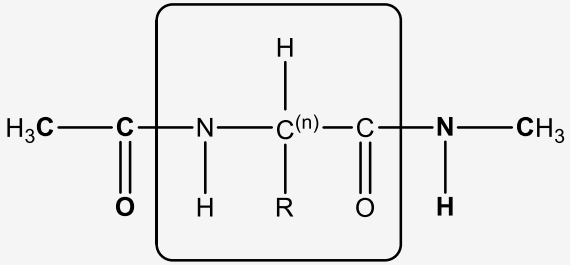
- ACEとNMEを
  隣接残基を基に付加しているため

  -C-C=O, N(H)-C- が重なり部分として認識

元のペプチド鎖



ACE/NME末端処理



QCLO法計算のコツ

- 相互作用の強いものほど先に計算
  - SS結合・イオン対
  - タンパク質二次構造(水素結合ネットワーク)
  - 金属錯体
- 対称性の良いフラグメントを作成しない
  - LO計算・軌道のpickupに失敗
    - 振り分けに困る分子軌道が存在

QCLObot 今後の開発計画

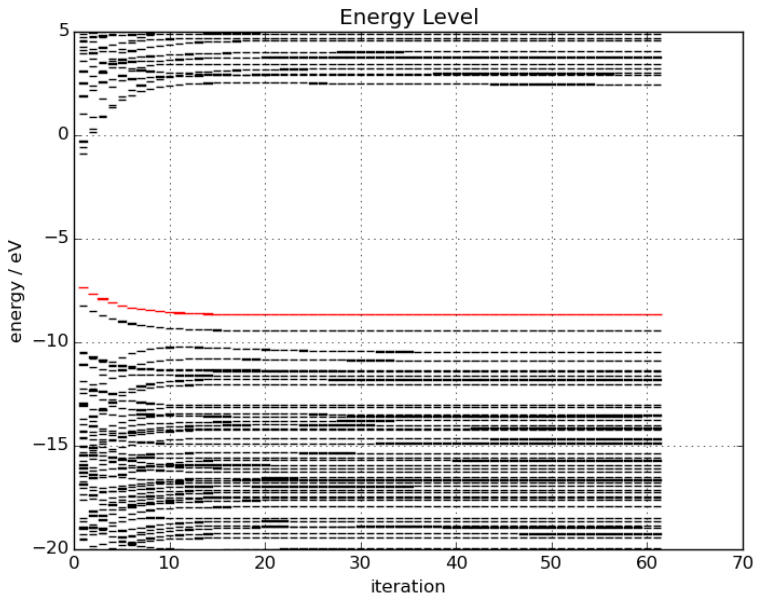
- 使いやすさの向上
  - スクリプトベースからテキストベースに
  - 計算シナリオの自動化
    - 分子構造・相互作用ダイアグラムに基づく計算シナリオの提案
- 高速化・並列化
  - LO・QCLO作成処理の高速化・並列化
  - フレーム分子計算の並行処理・計算機マネジメント
- 構造最適化・遷移状態計算・DFT-MDの実装

【演習】3残基QCLOの計算

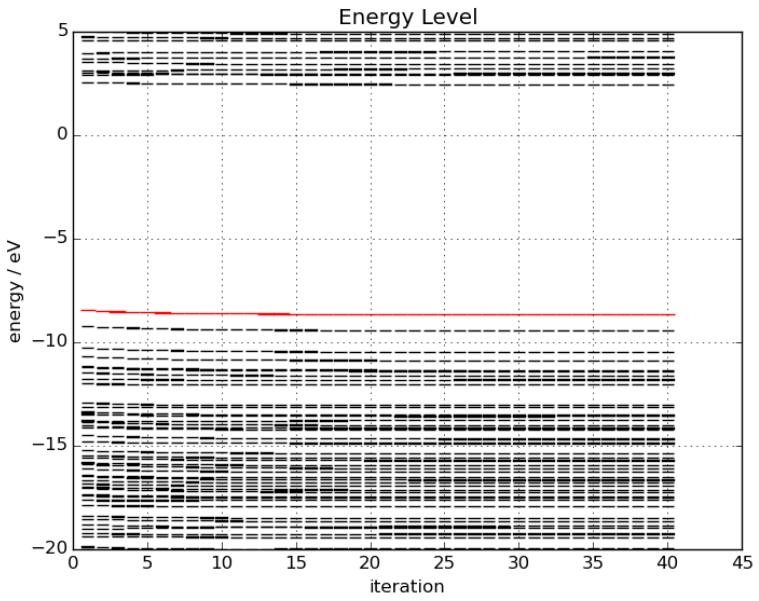
- `/home1/glej/share/QCLO_samples/1UAO` を
  自分の任意のディレクトリにコピーします。
- `run_serial.sh` を
  バッチシステムに投入します。
- 密度行列による初期値と
  QCLO法による初期値の違いを
  収束カーブ・軌道準位などから
  確認してください。

初期値の精度比較

密度行列による初期値



QCLO法による初期値



さいごに

ご支援・ご協力のお願い

- 質問・バグ・要望などがありましたら、是非お寄せください。
  GitHubのIssuesが便利です。
  – 日本語使用可です。