Vol.2012-HPC-133 No.35
2012/3/27

並列 FMO プログラム OpenFMO の性能最適化

稲富雄一[†]，眞木淳[††]，高見利也[†]，本田宏明[†]，小林泰三[†]，南里豪志[†]，青柳睦[†]，南一生[†††]

数万～数 10 万並列での効率的な実行を目指して，並列フラグメント分子軌道プログラム OpenFMO の性能最適化を行った．その際に，FMO 計算のカーネルコードである分子積分計算の負荷均等化と，中間データであるモノマー密度行列データの保存，アクセス方法に注目して最適化を行った．その結果，分子積分計算の並列化効率向上と，モノマー密度行列へのアクセス性能向上を達成することができ，数万並列実行時での効率的な FMO 計算が可能になった．

Performance Tuning of Parallel Fragment Molecular Orbital Calculation Program, OpenFMO

Yuichi Inadomi[†] Jun Maki[††], Toshiya Takami[†], Hiroaki Honda[†], Taizo Kobayashi[†], Takeshi Nanri[†], Mutsumi Aoyagi[†] and Kazuo Minami[†††]

We optimized the parallel fragment molecular orbital (FMO) calculation program, OpenFMO, for the effective massively parallel execution. In the optimization, we focused on the load balancing of molecular integral calculation which is the kernel of the FMO calculation, and on the way to access the density matrix data needed in the FMO calculation. As the result of our optimization, we achieved the high parallelization efficiency of molecular integral calculation and the effective access to density matrix data. These improvements lead us to carry out the effective FMO calculation over 10000 parallel execution.

1. はじめに

フラグメント分子軌道（Fragment Molecular Orbital, FMO）法はたんぱく質や DNA, 糖鎖などの生体分子に対する第一原理電子状態計算を高速に行うために開発された計算手法である[1-3]. FMO 法では計算対象となる巨大な分子を 20～40 原子程度の小さなフラグメントに分割して，各フラグメント（モノマー）やフラグメントペア（ダイマー）に対する小規模な電子状態計算を行うことで，分子全体の電子状態を近似する．複数のモノマー，あるいはダイマーの電子状態計算を並列に実行（粗粒度並列化）できること，ならびに，各モノマー，ダイマーの電子状態計算自身も更に並列処理（細粒度並列化）が可能であることから，FMO 法は超並列処理向きの計算手法である．GAMESS[4]や ABINIT-MP[3]といった並列 FMO 計算プログラムの既存実装があり，1,000 並列程度であれば効率よく並列処理できることが確認されている[5]．しかし，1 万並列を超えるような超並列実行時に高い並列化効率を保つためには，負荷分散を正確に行う，通信負荷を削減する，あるいは，使用する計算機のアーキテクチャに適したプログラミングを行う，などの最適化を行って，並列化効率を落とす要因を可能な限り取り除く作業が必須である．そのような精緻な最適化作業を行う場合には，対象とするプログラムが単純な構造を持っている方が好ましい．

OpenFMO[6,7]は超並列 FMO 計算を行うことを目的として九州大学と九州先端科学技術研究所でスクラッチから開発された並列 FMO プログラムである．非経験的第一原理電子状態計算手法の中で最も単純な Hartree-Fock（HF）法を基にした FMO 計算に特化したプログラムであるためコードが比較的短く（全体で約 54,000 行），標準的な並列処理ライブラリである MPI[8]を用いた並列化が行われているため，実行プロファイル取得，ならびに，最適化作業が行いやすい．そこで，我々のグループでは，比較的単純な構造を持つ OpenFMO に対して超並列化に向けて最適化を行ってきた．本報告では，その内容と結果について報告する．

2. FMO 法について

ここでは，HF 法を基にした FMO 法の概要について述べる．

2.1 FMO 法概要

FMO 法は，計算対象となる巨大な分子を 20～40 原子程度の小さなフラグメントに分

† 九州大学
Kyushu University
†† 九州先端科学技術研究所
Institute of Systems, Information Technologies and Nanotechnologies
††† 理化学研究所
RIKEN

割して，各フラグメント（モノマー）とフラグメントペア（ダイマー）に対する小規模な電子状態計算を行うことで，分子全体の電子状態を近似する計算手法である．例えば，電子状態計算で得られる重要な値であるエネルギー（E_{FMO}）や密度行列（$\mathbf{D}_{\mathrm{FMO}}$）を，FMO 法では以下の式で求める．

$$E_{\mathrm{FMO}} = \sum_{I}^{N_{\mathrm{frag}}} E_I + \sum_{I}^{N_{\mathrm{frag}}} \sum_{J<I}^{N_{\mathrm{frag}}} \left(E_{IJ} - E_I - E_J \right) \tag{1}$$

$$\mathbf{D}_{\mathrm{FMO}} = \sum_{I}^{N_{\mathrm{frag}}} \mathbf{D}_I + \sum_{I}^{N_{\mathrm{frag}}} \sum_{J<I}^{N_{\mathrm{frag}}} \left(\mathbf{D}_{IJ} - \mathbf{D}_I - \mathbf{D}_J \right) \tag{2}$$

ここで N_{frag} はモノマー数を，$\{E_I\}$，$\{E_{IJ}\}$ は，モノマー，ダイマーのエネルギーを，また $\{\mathbf{D}_I\}$，$\{\mathbf{D}_{IJ}\}$ はモノマー，ダイマーの密度行列を，それぞれ，表している．

モノマーに対する小規模電子状態計算を行う際には，計算対象となっているフラグメントだけでなく，近傍のモノマーの密度行列が必要となる．

$$E_I^{n+1}\left(\mathbf{D}_I^{n+1}\right) \leftarrow f\left(\mathbf{D}_I^n, \left\{\mathbf{D}_K^n\right\}_{K \in \text{neighborhood of monomer } I}\right) \tag{3}$$

(3)式はモノマーのエネルギー，密度行列が，該当モノマーとその近傍にあるモノマー密度行列の関数であることを表している．一般に，(3)式の入力として与えている密度行列 $\mathbf{D}_I^n$ と，出力として得られる密度行列 $\mathbf{D}_I^{n+1}$ は異なる．FMO 法では，この入力 $\mathbf{D}_I^n$ と出力 $\mathbf{D}_I^{n+1}$ の差が十分に小さくなる（モノマー密度行列が収束する）まで，モノマー電子状態計算を繰り返す．この処理を Self-Consistent Charge（SCC）処理，と呼ぶ．さらに，FMO 法では，SCC 処理で収束したモノマー密度行列を用いてダイマー電子状態計算を行う．

$$E_{IJ}\left(\mathbf{D}_{IJ}\right) \leftarrow f\left(\mathbf{D}_I, \mathbf{D}_J, \left\{\mathbf{D}_K\right\}_{K \in \text{neighborhood of monomer } I \text{ and } J}\right) \tag{4}$$

FMO 法では(3)式や(4)式で得られたモノマーやダイマーの電子状態を用いて，(1)式や(2)式で分子全体の電子状態を計算する．

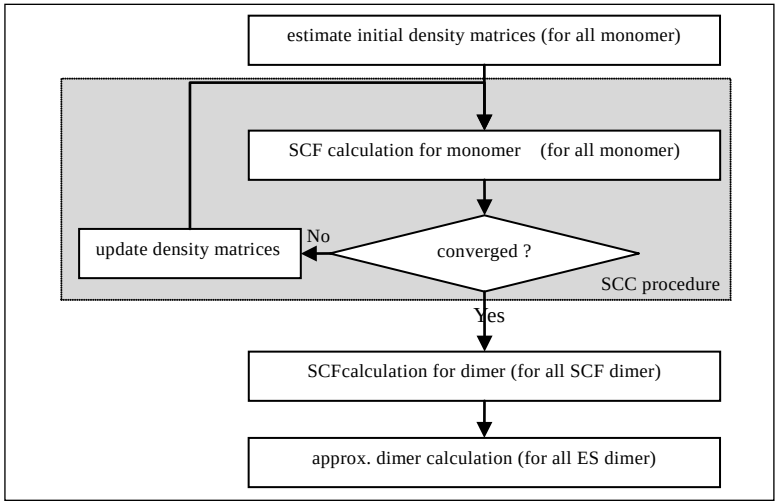

図 1：FMO 計算の流れ
Figure 1: Flowchart of FMO calculation

FMO 法では計算精度を保ったまま計算量を削減するために，ダイマー電子状態計算に対して近似計算を行う．ダイマーを構成する２つのモノマー間の距離が十分に大きい場合には，繰り返し計算や多くの積分計算を省いて，２つのモノマー間の静電相互作用で，ダイマー電子状態を近似する．

$$\begin{aligned} E_{IJ} &\leftarrow f\left(E_I, E_J, \mathbf{D}_I, \mathbf{D}_J\right) \\ \mathbf{D}_{IJ} &\simeq \mathbf{D}_I + \mathbf{D}_J \end{aligned} \tag{5}$$

この近似をダイマー静電（ダイマーES）近似と呼び，この近似を適用するダイマーを ES ダイマーと呼ぶ．一方で，近似をしないダイマーのことを SCF ダイマーと呼ぶ．

図 1 に FMO 計算のフローチャートを示す．はじめに，すべてのモノマーの初期密度行列を計算する．次に与えられたモノマー密度行列を基に，すべてのモノマーに対する小規模電子状態計算を行う．与えたモノマー密度行列と計算して得られたモノマー密度行列が収束していなければ，密度行列を更新して，再び，すべてのモノマーに対する小規模電子状態計算を行う．モノマー密度行列が収束したら，収束したモノマー密度行列を用いて，SCF ダイマーに対する小規模電子状態計算，および，ES ダイマーに対する近似計算を行う．最後に，(1)式や(2)式を用いて分子全体の電子状態を計算する．

2.2 並列 FMO プログラムの構造

FMO 法は複数のモノマー（ダイマー）の電子状態計算を並列に処理する粗粒度並列化と，各モノマー（ダイマー）の電子状態計算自身を並列処理する細粒度並列化を行うことが容易であるため，2 段階並列処理向きの計算手法である．並列 FMO プログラムの基本的な構造を，MPI で並列化した場合を例に**図 1** に示す．MPI プロセスは，1 つが FMO 計算の制御を行うマスタプロセスとなり，残りがモノマー，ダイマー電子状態計算を行うワーカプロセスになる．ワーカプロセスは複数（図では N_g 個）のグループ（worker group）に分けられる．この worker group 単位で，モノマー，ダイマーの電子状態計算を行う（粗粒度並列処理）．各 worker group には複数（図では P 個）のプロセスが含まれており，マスタプロセスに割り当てられたモノマー（ダイマー）の電子状態計算を P プロセスで並列処理する（細粒度並列化）．このように，並列 FMO プログラムは 2 段階並列化されており，かつ，マスタワーカ型の実行スタイルになる．ただ，注意しなければならないのは，各 worker group 間でモノマー密度行列データの授受が必要となるため，純粋なマスタワーカ型プログラムではない点である．負荷バランスの観点から worker group 当たりに割り振るモノマー電子状態計算の個数を多くする必要があるため，入力データのサイズ（モノマー数）に対して worker group 数を十分に小さくする必要がある．したがって，FMO 計算を数万～数 10 万並列で効率的に行うためには，各 worker group が数 100～1000 並列で効率的にモノマーやダイマーに対する小規模電子状態計算を行う必要がある．また，worker group 間でのモノマー密度行列データのやり取りを行う際に通信が必要となるため，計算の邪魔をせずに上手くデータ授受を行うための方法を検討する必要がある．

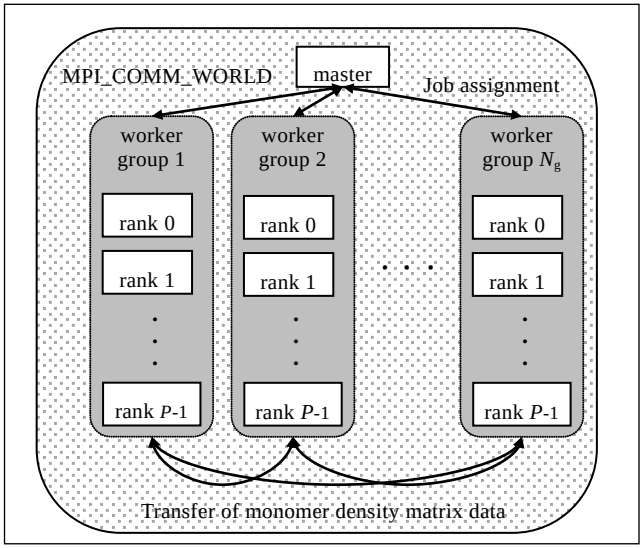

図 2：並列 FMO プログラムの基本構造
Figure 2: Structure of parallel FMO program

3. 超並列化に向けた OpenFMO の最適化

3.1 MPI/OpenMP[9]ハイブリッド並列化

中規模，大規模の計算機として最近主に用いられているクラスタ型並列計算機は，複数のプロセッサ（コア）を搭載した小型計算機（計算ノード）を Gigabit Ether や Infiniband などのネットワークで相互結合したアーキテクチャを持つ. 近年のプロセッサのマルチコア化に伴って計算ノードあたりのコア数は増加しているが，一方で，計算ノードに搭載される通信インターフェイス（NIC）は，消費電力の問題やコスト面などから，あまり増えていない．OpenFMO の超並列実行のターゲットである京コンピュータも 8 コアあたり 4 つの NIC（通信エンジンと呼ばれている）しか存在しない．このように，1 つの NIC を複数のコアで共有する形になっているため Flat MPI プログラムをこのような並列計算機で実行すると，複数のプロセスによる通信資源の競合が発生して，並列性能が低下する恐れがある．そのため，クラスタ型並列計算機を効率的に利用するためには，計算ノード間は MPI を用いたプロセス並列を，また，計算ノード内は OpenMP を用いたスレッド並列を適用する MPI/OpenMP ハイブリッド並列化が望まれており，京コンピュータにおけるプログラミングモデルとしても推奨されている．そこで，OpenFMO では，最も計算時間のかかる 2 電子積分や多中心クーロン積分などの分子積分部分にハイブリッド並列を適用した．

3.2 小規模電子状態計算の高並列化

FMO 計算の効率的な超並列実行のためには, モノマーやダイマーに対する小規模電子状態計算を数 100～1000 並列で効率的に行う必要があるため，まず，小規模電子状態計算の並列性能向上に注目した. 小規模電子状態計算部分の模擬コードを**図 3** に示す. カットオフテーブル計算は 2 電子積分や 4 中心クーロン積分の計算量削減のために用いるテーブルを作成するために行うもので，計算時間は非常に短い（1～数秒）が，すべてのプロセスで計算する必要がある．また，最後に行う SCF 計算は，非線形方程式（Hartree-Fock-Roothaan 方程式）を解くための繰り返し計算である．その内部では，予め計算して主記憶に保存してある 2 電子積分の値を用いた Fock 行列生成と Fock 行列（密対称行列）に対する対角化を複数（～10）回行っている．また，1 電子積分と射影演算子の計算は，非常に計算量が少ない（計算時間として 0.5 秒以下）．このように，小規模電子状態計算内部ではいろいろな計算を行っているが，計算時間の多く（99.9%以上）は 2 電子積分や環境ポテンシャル（4 中心，3 中心，2 中心のクーロン

```
calculation_SCF_energy(int ifrag) {
        for (id=0; id<(Nifc4c+1); id++ ) make_cutoff_table(id);        // making cutoff table
        calc_twoint ( myrank, nprocs, ERI_buffer);                     // two-electron integral
        V=0.0;
        for ( i=0; i<Nifc4c; i++ ) V+=calc_ifc4c( myrank, nprocs, i);  // 4-centered Coulomb integral
        for ( i=0; i<Nifc3c; i++ ) V+=calc_ifc3c( myrank, nprocs, i);  // 3-centered Coulomb integral
        for ( i=0; i<Nfrag; i++ ) V+=calc_ifc2c( myrank, nprocs, i);   // 2-centered Coulomb integral
        if ( myrank == 0 ) {
            calc_oneint( S, H );                                       // one-electron integral
            calc_projection( P );                                      // projection operator
        }
        MPI_Reduce( V, comm );
        SCF_procedure( S, H, P, V, ERI_bufferi, Difrag ,comm );        // SCF calculation
}
```

図 3：FMO 法における小規模電子状態計算部分の模擬コード
Figure 3: Pseudo code of tiny electronic structure calculation in FMO method

N_{frag}= # of monomers, N_{ifc4c}= # of counter monomers to calculate 4-centered Coulomb interaction(=～20), N_{ifc3c}= # of counter monomers to calculate 3-centered Coulomb interaction (=～20)

積分）などの分子積分計算で占められる．
小規模電子状態計算部分の高並列化に向けた最適化を行う前に，既存コードの小規模電子状態計算部分の性能を調べた．その結果得られた MPI プロファイルを図 4 に示す．

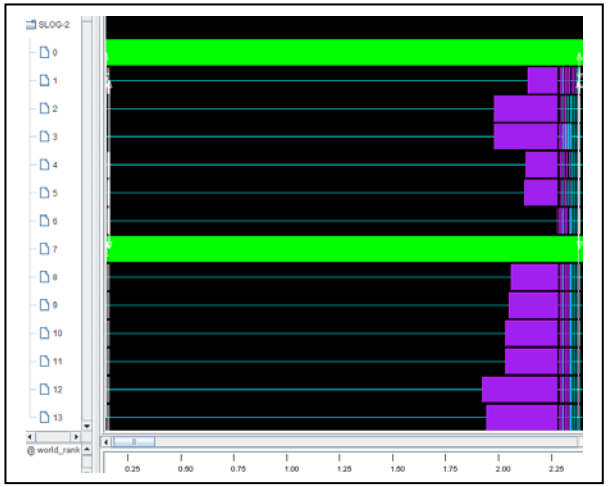

図 4：動的負荷分散導入前のモノマー電子状態計算部分の MPI プロファイル
Figure 4: MPI profile of a monomer SCF calculation without dynamic load-balancing

このプロファイル取得に使用した計算機は，九州大学情報基盤研究開発センターにある小型クラスタ並列計算機（quad core Xeon×2/node, 7 nodes, インターコネクト＝10GbE）で, MPI として MPICH2[10]（version 1.3.1）, コンパイラとして Intel C++ compiler（version 12.0.0）を用いた．入力データは，アデノウィルスの KNOB ドメイン（PDB ID=1NOB，576 フラグメント）である．また，MPI プロファイルは MPE[11]を用いて取得し，MPE に付属している可視化ツールの JUMPSHOT で可視化を行った．既存コードでは，分子積分計算部分にサイクリック分割による静的負荷分散を導入しているが，図 4 を見ると，計算が早く終わったプロセスが，計算時間のかかるプロセス（この図では rank=6 のプロセス）の計算が終了するまで待っており，分子積分計算部分において負荷不均衡が生じていることが分かる．このような負荷不均衡は並列性能を低下させる要因の一つであるため，負荷均等化を行うことで，小規模電子状態計算部分の並列性能向上を行うことにした．
分子積分計算，その中でも計算量が $O(N^4)$と計算量の多い，2 電子積分と 4 中心クーロン積分計算では，計算量を削減するために $O(N^2)$の計算量を持つカットオフテーブル計算を予め行い，作成したカットオフテーブルの結果を用いて，十分に値が大きいと思われる積分だけを計算する．分子の大きさだけでなく，分子軌道の展開に用いる基底関数の種類，分子構造，組成などの様々な要因がカットオフ処理で生き残る（計算を行う）積分数に関係しているため，計算量の正確な予測が困難である．このため，分子積分計算における負荷分散には，すべてのプロセスからアトミックな参照，更新

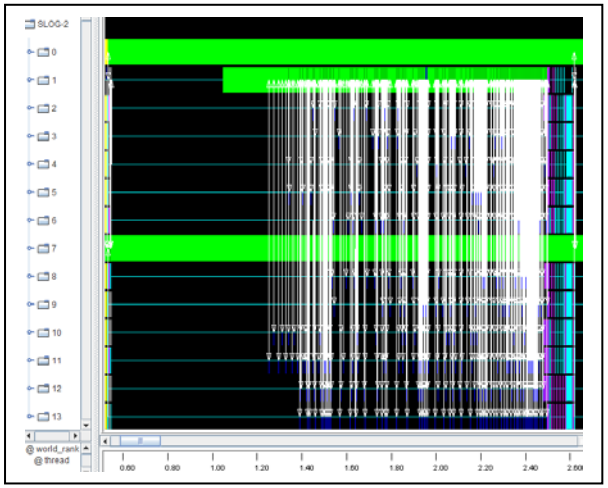

図 5：動的最適化導入後のモノマー電子状態計算部分の MPI プロファイル
Figure 5: MPI profile of a monomer SCF calculation with dynamic load-balancing

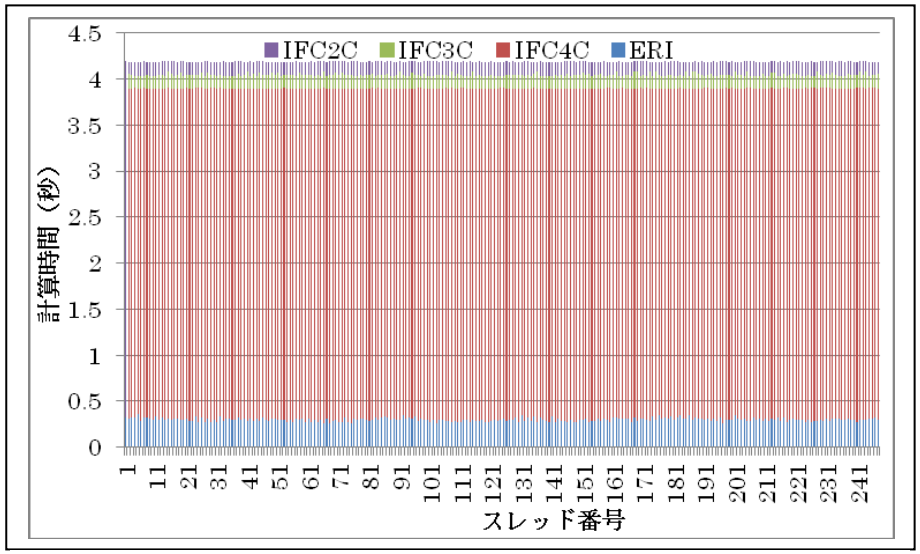

図 6：モノマー電子状態計算の分子積分計算部分の実行プロファイル
Figure 6: Execution profile of molecular integral calculation of a monomer

ができるカウンタ（global counter，以下 GC）を用いた動的負荷分散が用いられる．そこで，OpenFMO の小規模電子状態計算部分の負荷分散において GC を用いた動的最適化を行うことにした．京向けの最適化が行われている GAMESS では，軽量片側通信ライブラリ ARMCI[12]の atomic fetch & add 機能を用いて GC の実装を行っているが，我々はより幅広く用いられている MPI と OpenMP による GC 実装を行うことにした．その際に，MPI_THREAD_SERIALIZED レベルのスレッドサポートを用いて実装した．これは，京コンピュータで動作する MPI ライブラリがこのレベルのスレッドサポートを行うことがアナウンスされているからである．実装の詳細は割愛するが，worker group に含まれるスレッドのうち 1 つがカウンタ値の管理や他のプロセスからのアクセス要求に応答することだけを行うカウンタ専用スレッドとして動作するようにしている．

GC を用いた動的負荷分散を適用したコードの MPI プロファイルを図 5 に示す．このプロファイル取得環境は，動的負荷分散導入前のコードの場合と同じである．この結果をみると，GC 利用に伴う 1 対 1 通信（図の縦方向の白線）が見られるものの，積分計算時間がすべてのプロセスで揃っており，同期待ち時間がほとんどなく，負荷均等化が図られていることが分かる．

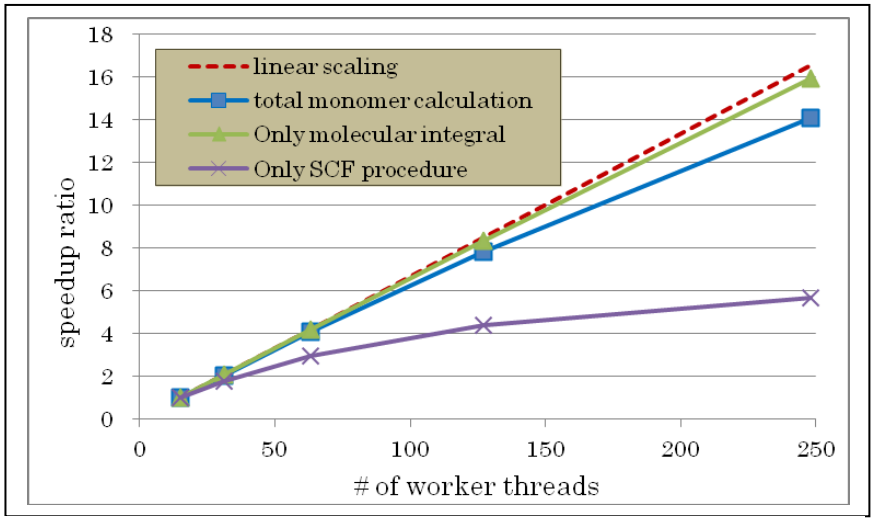

図 7: モノマー電子状態計算部分の速度向上比
Figure 7: Speedup ratios of monomer SCF calculations

次に，worker group に含まれるスレッド数を大きくした場合の動的負荷分散導入後のコードの性能評価を行った．性能評価は理化学研究所の Riken Integrated Cluster of Clusters（RICC）の超並列 PC クラスタ部を用いて行った．MPI ライブラリとして OpenMPI[13]（version 1.4.3），コンパイラとして Intel C++ comiler （version 11.1）をそれぞれ用いた．各種積分計算時間を MPI_Wtime 関数や omp_get_wtime 関数を用いて取得した．ワーカスレッド数 248 の場合における各スレッドの積分計算時間を図 6 に示す．この図の横軸はワーカスレッドの ID で，縦軸は各種分子積分の積算計算時間（秒）を表している．図中の ERI，IFC4C，IFC3C，および，IFC2C は，それぞれ，2 電子積分，4 中心クーロン積分，3 中心クーロン積分，および，2 中心クーロン積分を表している．これを見ると，各スレッドの積分計算がほぼ均等であり，うまく負荷均等化できていることが分かる．また，**図 7** に速度向上比を示す．この図は，GC マスタスレッドを含むワーカースレッド数が 16（4 プロセス 4 スレッド）の場合を基準（速度向上比＝1）として，ワーカスレッド数を 32，64，128，248 と変化させた場合の速度向上比を小規模電子状態計算全体，分子積分計算部分（カットオフ計算含む），および，SCF 計算部分それぞれで示したものである．これを見ると，計算負荷が大きな分子積分部分は 200 並列を超えてもほぼ linear scaling を保っているが，SCF 計算部分の速度向上がワーカースレッド数の増加とともに急激に鈍っており，このことが影響して小

規模電子状態計算全体の速度向上比の伸びが鈍化している．これは SCF 計算にノード間並列化が困難な小規模対称行列（数 100 次元）の対角化や行列積，ベクトル積などの演算部分が含まれており，並列化効率のよい積分計算（Fock 行列計算）時間がワーカスレッド数増加とともに減少するのに対して，対角化などのシリアル部分の影響が顕著になるからである．

3.3 モノマー密度行列データの保存・アクセス方法

FMO 計算では，あるモノマー（ダイマー）の小規模電子状態計算を行う際に，周辺モノマーの密度行列データが必要となる．京コンピュータなどの大型計算機を用いる際にはこれまでよりも大規模な入力データを用いた計算を行うことになると考えられる．小規模な入力を用いる場合には，計算を行うすべてのプロセスが，それぞれ，すべてのモノマー密度行列データを保持することも可能であるが，モノマー密度行列のデータサイズは入力データの規模に比例して増加するため，超並列計算機利用時に想定される大規模入力への対応を行うためには，各計算ノードがすべてのモノマー密度行列データを持つのではなく，複数のプロセスで分散保存して，その分散保存されたデータに効率的にアクセスできるようにする必要がある．これまでの性能予測ツールなどを用いた実験により，密度行列データを分散保存して，必要なデータを片側通信などでアクセスする手法を採った場合，入力規模に関わらず通信コストが一定割合になる，という結果が得られている[14]．そのために，今回はモノマー密度行列データの分散保存の方法とそのアクセス手法について検討した．まずはじめに，計算を行うプロセス（ワーカプロセス）がモノマー密度行列の保存を分担して行う方法（以降，方法１）を実装した．この方法では，各プロセスが密度行列データの保存と計算の両方を行うため，１つのコードで計算とデータ保存の２つの処理を記述することができること，ならびに，すべてのプロセスで分担してデータ保存を行うため入力に対する scalability も高い，という特徴がある．**図 8** の左図にワーカ 8 プロセスを 2 つのワーカグループに分割して，9 個のモノマー密度行列を分散して保存する場合の模式図を示す．ここで密度行列データは数字付きの○で示されており，数字がモノマー番号を表している．ある worker group でモノマー密度行列データが必要となった場合には，worker group のマスタプロセスが必要なデータを持っているプロセス（ターゲットプロセス）からデータを取得し，得られたデータを worker group 内で Bcast する．一方，計算したモノマー密度行列で更新を行う場合には，マスタプロセスがターゲットプロセスにデータを送信する．一般に，各 worker group は独立に計算を行っており，かつ，必要なモノマー密度行列データを他の worker group のプロセスが保持していることがあるため，モノマー密度行列データの取得，更新処理を，ターゲットプロセスが属する worker group の邪魔をせず（余計な同期なし）に行うためには，同期が必要な 1 対 1 通信ではなく，非同期の（受動的）片側通信機構が必要となる．

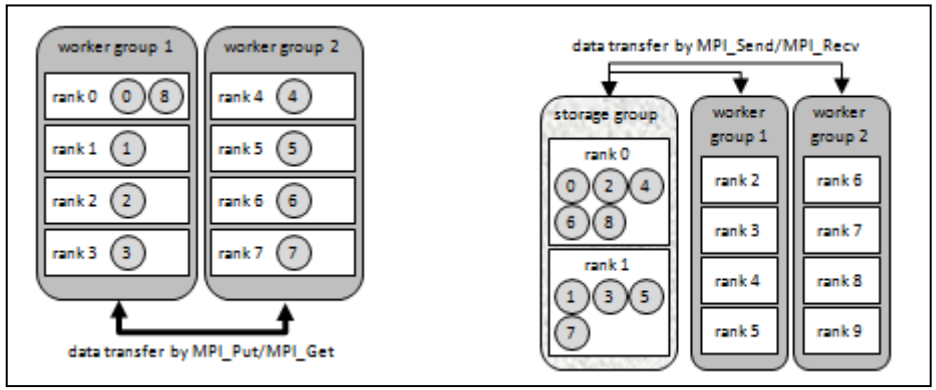

図 8：今回検討したモノマー密度行列保存，アクセス方法
Figure 8: Two methods of storing and accessing density matrix data
Left : Method 1 using MPI-2 one-sided communication (# of worker processes = 8, # of monomers = 9)
Right : Method 2 using MPI-1 P2P communication (# of worker processes = 8, # of monomers = 9,# of storage processes = 2)

これとは別に，片側通信を用いないで，かつ，worker group のジョブ実行を阻害しない方法として，データ保存のための専用プロセスを用いること（以降，方法２）も検討した．この方法では，モノマー密度行列データを保持してワーカプロセスからのアクセス要求に応答することを専門とするストレージプロセス（storage group に属する）と，計算を専門に行うワーカプロセス（いずれかの worker group に属する）とを分離する．図 8 の右図にワーカプロセス数 8，ワーカグループ数 2，ストレージプロセス数 2，および，モノマー数 9 の場合の模式図を示している．各ストレージプロセスはワーカプロセスからのデータアクセス要求を MPI_Recv 関数で待っていて，参照要求の場合には要求元へデータを転送し，更新要求の場合には要求元からデータを取得して該当モノマー密度行列データを更新する．一方，worker group 内でモノマー密度行列データが必要になると，マスタプロセスがターゲットプロセス（ストレージプロセスの１つ）にデータ要求を行い，得られたデータを worker group 内に Bcast する．計算したモノマー密度行列データで更新処理を行う場合には，worker group のマスタプロセスがターゲットプロセスに対して更新要求を行った後，データを転送して更新する．この方法では，ストレージプロセスとワーカプロセスのコードを別々に準備する必要がある，あるいは，ストレージプロセスとワーカプロセス間で負荷不均衡が生じやすい，といった欠点があるが，ターゲットプロセスとワーカプロセス間の通信に，（ブロッキング）1 対 1 通信しか用いていないので，MPI-1 規格でも対応可能であり，入力データの規模に合わせてストレージプロセス数を変化させることで，入力に対す

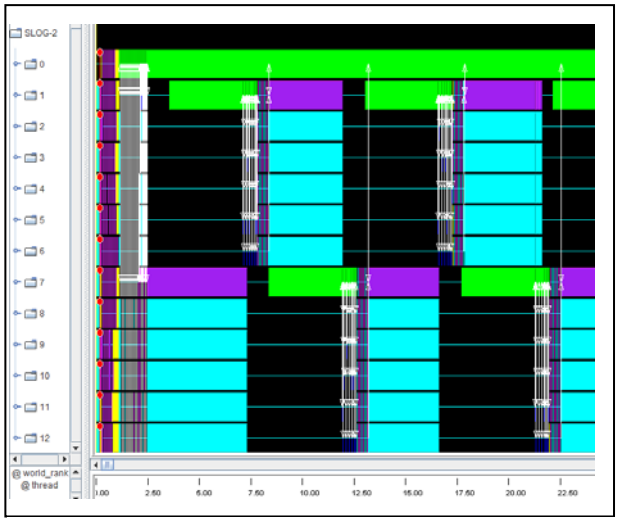

図 9：方法 1 を用いた場合の MPI プロファイル
Figure 9: MPI profile using method 1

る scalability も保つことができるという特徴がある．
この 2 つの密度行列データ保存，アクセス方法を実装して，その性能を評価した．まず，片側通信を利用した方法 1 で実装したコードを用いた場合の MPI プロファイルを**図 9** に示す．この結果の特徴的な点は，モノマー密度行列データ取得のために多くの待ち時間が生じていることである．これは，片側通信の際にデータの実体を持つプロセス（ターゲットプロセス）が何らかの MPI 関数呼び出しを行うまで，片側通信に対する要求に応答しない，という片側通信の実装が MPICH2 で行われているためだと考えられる．例えば，図 9 の経過時間約 2.5 秒～7.5 秒あたりで worker group 2（rank 7～12）のプロセスが片側通信の同期関数である MPI_Win _unlock 関数と後続する MPI_Bcast 関数による待ち時間を生じているが，これは worker group 2 に割り当てられたモノマー電子状態計算で必要なモノマー密度行列データのターゲットプロセスが worker group 1（rank=1～6）に含まれていたため，worker group 1 のプロセスが MPI 関数を呼び出していると見られる経過時間約 7.5 秒付近まで，worker group 2 のプロセスの片側通信に応答していないことが原因であると推測される．このように，少なくとも MPICH2 に実装されている片側通信を用いると，分散保存した密度行列データへのアクセス性能が非常に悪くなることが分かった．MPICH2 は多くのプラットフォームで利用可能であり，OpenFMO のユーザが MPICH2 を利用することが十分に考えられる．その際に方法 1 で実装していると前述と同様の性能低下を招く恐れがあり，好ましくない．

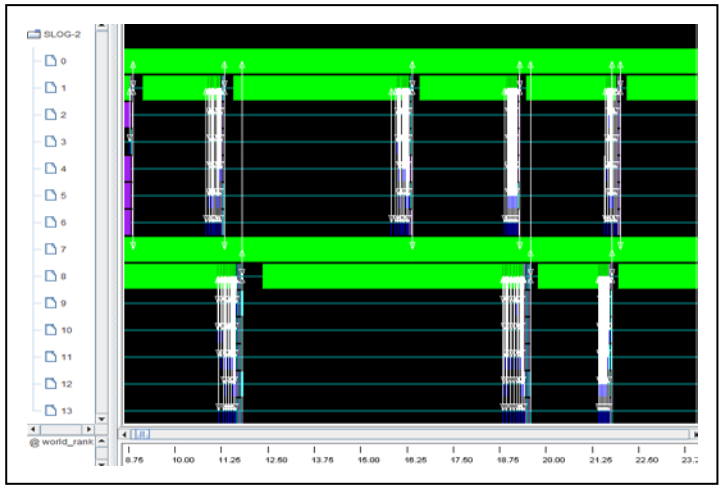

図 10：方法 2 を用いた場合の MPI プロファイル
Figure 10: MPI profile using method 2

次に，モノマー密度行列データを保存して，ワーカプロセスからのアクセス要求に対する応答のみを行うストレージプロセスを用いた方法 2 を用いた場合の性能評価結果を**図 10** に示す．この例は，14 プロセスでの並列実行をした結果である．ランク 0 のプロセスがマスタプロセスで，ランク 7 のプロセスがストレージプロセスであり，残りの 12 プロセスを worker group 1（rank=1～6）と worker group 2（rank=8～14）の 2 つの worker group に分割している．この図を見ると，ランク 0 のマスタプロセスは，方法 1 の場合と同様に，各 worker group に対するジョブ割り当てを行うために MPI_Recv による worker group からの要求待ちをしていることが分かる．各 worker group 内で GC のマスタプロセスとなっている rank= 1 と rank=8 のプロセスは，worker group 内の他のワーカプロセスからのジョブ要求待ちで，こちらも方法 1 と同様に MPI_Recv による待ちとなっていることが分かる．ストレージプロセスである rank=7 のプロセスは，ワーカプロセスからのモノマー密度行列に対するアクセス（参照，更新）要求待ちのため，ほとんどが MPI_Recv での待ち時間となっている．一方で，モノマー密度行列データの参照，更新に伴うワーカプロセス側の待ち時間は，方法 1 の場合と違って，ほとんどないことが分かる．
方法 2 の実装では，ワーカプロセスとストレージプロセスの 2 種類のプログラムを記

述する必要があること，ならびに，ストレージプロセスのほとんどの仕事が保持しているモノマー密度行列データへのアクセス要求待ちであり無駄が多い，という短所がある．しかしながら，ワーカプロセスのモノマー密度行列データへのアクセスに伴う待ち時間が非常に小さくなり，ワーカプロセスによるモノマー（ダイマー）電子状態計算の実行効率が高くなる．したがって，超並列 OpenFMO プログラムにおけるモノマー密度行列データの保存，および，アクセス方法は，現状では，ストレージプロセスを導入した方法 2 が適当であると考えている．

4. まとめ

効率的な超並列 FMO 計算を目指して並列 FMO プログラム OpenFMO の性能最適化を行った. FMO 計算で多数回行う小規模電子状態計算部分におけるカーネルコードである分子積分計算部分に global counter を用いた動的負荷分散を用いることで，負荷が均等化され，248 並列時の分子積分部分の並列化効率が 0.96 と非常によい並列性能を達成できた．今回行った小規模電子状態計算部分への最適化によって，粗粒度並列処理との組み合わせによって，数万並列での効率的な FMO 計算が行えるようになったと考えている．一方で，小規模電子状態計算で行う SCF 計算に現れる対角化などのノード間並列処理が困難な部分の存在が，大規模並列時の並列性能低下を引き起こすことが分かった．更なる並列性能向上のためには，非並列処理の部分の削減や，非並列処理部分と積分計算とのオーバーラップによる非並列処理時間の隠蔽などを行う必要がある．
また，モノマー密度行列の保存，および，アクセス方法について MPI-2 の片側通信を用いた方法と，MPI-1 の 1 対 1 通信を用いた場合の２つの方法について検討した．その結果，1 対 1 通信を用いた場合には効率よく密度行列データへのアクセスを行うことができたが，片側通信機能を使った場合には，ライブラリにおける片側通信の実装方法によっては大きく性能低下することが分かった．1 対 1 通信を用いる場合にはデータ保存を行うプロセスと計算を行うプロセスに異なるプログラムを記述する必要があるのに対して，片側通信を用いた実装では 1 つのプログラムでの記述が可能であるため，プログラム記述の面からは，片側通信を利用できる方が好ましい．今後，高性能な片側通信機能を提供する汎用的なライブラリの実装が望まれる．

謝辞 本研究の一部は理化学研究所と九州大学との共同研究で行っている．また，科学研究費補助金基盤研究（C)「超並列フラグメント分子軌道法プログラムライブラリの開発」(課題番号 22550015)，および，JST,CREST の研究領域「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出」の研究課題「省メモリ技術と動的最適化技術によるスケーラブル通信ライブラリの開発」の支援を受けている．
さらに，学際大規模情報基盤共同利用・共同研究拠点の公募型共同研究平成２３年度採択課題「超並列フラグメント分子軌道法プログラム OpenFMO の 性能評価と高性能化」による計算機利用を行っている．

参考文献

1) K. Kitaura et al., “Pair interaction molecular orbital method: an approximate computational method for molecular interactions”, Chem. Phys. Lett., Vol.312, pp.319-324 (1999)
2) K. Kitaura et al., “Fragment molecular orbital method: an approximate computational method for large molecules”, Chem. Phys. Lett., Vol.313, pp.701-706 (1999)
3) T. Nakano et al., “Fragment molecular orbital method: application to polypeptides”, Chem. Phys. Lett., Vol.318, pp.614-618 (2000)
4) M.W.Schmidt et al.,“General Atomic and Molecular Electronic Structure System”, J. Comput. Chem., Vol.14, pp.1347-1363 (1993)
5) T. Ikegami et al., “Full Electron Calculation Beyond 20,000 Atoms: Ground Electronic State of Photosynthetic Proteins”, SC|05 Seattle, WA, technical paper, 2005
6) OpenFMO homepage : http://www.openfmo.org/OpenFMO/index.html
7) J. Maki et al., “One-sided Communication Implementation in FMO Method”, Proc. HPCAsia07, 137–142(2007)
8) Message Passing Interface Forum homepage : http://www.mpi-forum.org/
9) OpenMP.org homepage : http://openmp.org/wp/
10) MPICH2 homepage : http://www.mcs.anl.gov/research/projects/mpich2/
11) MPI Parallel Environment (MPE) homepage : http://www.mcs.anl.gov/research/projects/perfvis/software/MPE/index.htm
12)Aggregate Remote Memory Copy Interface (ARMCI) homepage : http://www.emsl.pnl.gov/docs/parsoft/armci/
13) OpenMPI home page : http://www.open-mpi.org/
14) 稲富雄一など,「片側通信を用いた並列フラグメント分子軌道計算プログラムの実装」, 2007 年並列／分散／協調処理に関する『旭川』サマー・ワークショップ(SWoPP 旭川 2007)HPC-111-15