11-NA05

超並列分子軌道法プログラム OpenFMO の性能評価と高性能化

南一生（理化学研究所）

概要
次世代スーパーコンピュータでの実行を目指して，並列 FMO プログラム OpenFMO の高性能化を行った．FMO 計算で必要となるモノマー密度行列データの保存方法について検討して，MPI-2 の片側通信を用いた実装と 1 対 1 通信を用いた実装の 2 つの方法について性能評価を行った．その結果，片側通信を用いた場合には，ライブラリ側での片側通信の実装によっては，密度行列データへのアクセス性能が大きく低下することが分かった．一方で，1 対 1 通信を用いた場合には，効率よくアクセスが行えることが分かり，この方法を用いた 8000 並列という大規模並列実行時でも，密度行列アクセスに伴う通信時間の増加が非常に小さいことも分かった．

1. 研究の目的と意義

【研究の目的】

フラグメント分子軌道（Fragment Molecular Orbital Method, 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．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（大粒度並列化）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 や ABINIT-MP 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．しかし，10 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．九州大学で開発されている並列 FMO 計算プログラム OpenFMO は，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．そのため，ソース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 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．昨年度は，OpenFMO プログラムを用いて FMO 計算の効率的な超並列処理において重要となる，各モノマー，ダイマーの電子状態計算の細粒度並列処理部分についての性能評価を行い，効率を低下させている場所を特定して，動的負荷分散を用いることで負荷均等化を図り，並列性能を向上させることができた．今年度は，FMO 計算で扱うモノマー密度行列と呼ばれる中間データに対するアクセスに伴う通信に注目して，アクセス性能を向上させ，FMO プログラム全体のさらなる性能向上を目指した．そのために，モノマー密度行列データの保存，アクセス方法を理論化学の研究者と協力して検討して，最近の大型計算機のアーキテクチャに適合するような最適化を行うことを本研究の目的とした．

【研究の意義】

FMO 法は，これまでにもたんぱく質や DNA などの生体分子と薬剤との相互作用解析に応用されてきており，特に，創薬分野における有用な道具として

期待されているシミュレーション手法の１つである．創薬分野では，候補化合物の絞り込み（スクリーニング）に計算機シミュレーションを用いることが考えられる．FMO 法をスクリーニングに用いるためには，現在よりも短時間に，多数回計算することが必要になる．そのためには，中規模，あるいは，大規模分子に対する FMO 計算を，今よりも格段に高速に行う必要がある．

FMO 法は大粒度，細粒度の 2 段階並列化が可能な計算手法であるため，超並列処理向きの計算手法であるが，既存実装は最近の大型計算機の主流となっている小型 SMP 計算機（計算ノード）を超高速相互結合網で接続したSMPクラスタ型計算機の特徴を考慮したプログラム設計になっていないため，数万～数 10 万並列（超並列）実行時には，効率的な処理が困難だと考えられる．現在稼働中，あるいは，開発中のスーパーコンピュータは数万プロセッサ（コア）を搭載した超並列計算機であり，今後そのような計算機を利用する機会が増加することを考えると，近い将来には，比較的容易に数万並列のプログラムを実行できるようになると思われる．そのような環境下で，大規模 FMO 計算を高速に実行するためには，超並列実行時に効率的に並列処理できるプログラムの開発が必須である．超並列計算機アーキテクチャを考慮して高性能な超並列 FMO プログラムが作成できれば，創薬分野におけるスクリーニングを計算機シミュレーションでサポートすることが可能となるため，薬剤の開発コストの削減や開発期間の短縮に寄与できると考えている．

2. 当拠点公募型共同研究として実施した意義

(1) 共同研究を実施した大学名と研究体制

大学名：九州大学

研究体制：

南一生（理化学研究所）

分散データ保存，アクセス方法に対する計算機科学的な観点からの支援

高見利也（九州大学）

OpenFMO の性能評価

稲富雄一（九州大学）

OpenFMO プログラムの実装

(2) 共同研究分野

超大規数値計算系応用分野

(3) 当公募型共同研究ならではという事項など

超並列化に向けたプログラムの最適化には，実機を用いた性能評価を行うことが不可欠である．今回，スーパーコンピュータを実際に利用して，性能評価を行うことができ，並列性能を向上させるための方針を決めるための情報を得ることができた．

3. 研究成果の詳細と当初計画の達成状況

(1) 研究成果の詳細について

【FMO 法概要】

FMO 法は，計算対象となる大規模分子を，20～40 原子の小さなフラグメントに分割して，分割した各フラグメント（モノマー），および，フラグメントペア（ダイマー）に対する小規模電子状態（分子のエネルギー，電子分布の様子など）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．FMO 法で最終的に求めたい量は，分子全体のエネルギー $E_{\text{total}}^{\text{FMO}}$，および，密度行列 $\mathbf{D}_{\text{total}}^{\text{FMO}}$ であるが，FMO 法では以下のように近似する．

$$E_{\text{total}}^{\text{FMO}} = \sum_{I>J}^{N_{\text{frag}}} E_{IJ} - \left(N_{\text{frag}} - 2\right) \sum_{I}^{N_{\text{frag}}} E_I \qquad (1)$$

$$\mathbf{D}_{\text{total}}^{\text{FMO}} = \sum_{I>J}^{N_{\text{frag}}} \mathbf{D}_{IJ} - \left(N_{\text{frag}} - 2\right) \sum_{I}^{N_{\text{frag}}} \mathbf{D}_I$$

ここで，N_{frag} はフラグメント（モノマー）数，$\{E_I\}$（$\{E_{IJ}\}$）は，モノマー（ダイマー）のエネルギー，また，$\{\mathbf{D}_I\}$（$\{\mathbf{D}_{IJ}\}$）は，モノマー（ダイマー）の密度行列を，それぞれ表わす．モノマーの電子状態計算を行うためには，計算しているモノマー自身の密度行列のほかに，その近傍にあるモノマーの密度行列も必要となる．

$$E_I^{n+1}\left(\mathbf{D}_I^{n+1}\right) \leftarrow f\left(\mathbf{D}_I^n, \left\{\mathbf{D}_K^n\right\}_{K \in \text{neighborhood of monomer } I}\right) \qquad (2)$$

式(2)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の密度行列の

関数であることを表わしている．一般に，式(2)の引数として与えているモノマーIの密度行列$\mathbf{D}_I^n$と，出力として得られる$\mathbf{D}_I^{n+1}$は異なる．FMO 法では，この関数の入力$\{\mathbf{D}_K^n\}$と出力$\{\mathbf{D}_K^{n+1}\}$の差が十分に小さくなる（モノマーの密度行列が収束する）まで，各モノマーの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 計算では，ダイマーの電子状態計算を，SCC 処理で収束したモノマー密度行列を用いて行う．

$$E_{IJ}(\mathbf{D}_{IJ}) \leftarrow f\left(\mathbf{D}_I, \mathbf{D}_J, \{\mathbf{D}_K\}_{K \in \text{neighborhood of monomer } I \text{ and } J}\right) \quad (3)$$

ダイマーの電子状態計算も，モノマーの場合と同様に，ダイマーのエネルギー，密度行列は，ダイマーを構成する２つのモノマーI，Jと，その近傍にあるモノマーの密度行列の関数である．FMO 法では，式(2)，(3)で得られたモノマー，ダイマーのエネルギー, 密度行列, および, 式(1)を用いて，分子全体の電子状態を求める．

FMO 法では計算精度を犠牲にすることなく計算時間を削減するために，ダイマーの電子状態計算に対する近似を行う．この近似では，ダイマーを構成する２つのモノマー間の距離が離れている場合に，繰り返し計算が必要で計算負荷の大きな Self-Consistent Field (SCF)計算を行わずに，２つのモノマーのエネルギーと，モノマー間の静電相互作用で，ダイマーの電子状態を近似する．

$$E_{IJ} \leftarrow f\left(E_I, E_J, \mathbf{D}_I, \mathbf{D}_J\right)$$
$$\mathbf{D}_{IJ} \simeq \mathbf{D}_I \oplus \mathbf{D}_J \quad (4)$$

この近似のことを**ダイマーES 近似**, この近似を行うダイマーのことを **ES ダイマー**と呼ぶ．一方，負荷の重い SCF 計算を行うダイマーを **SCF ダイマー**と呼ぶ．ダイマーES 近似を適用することで，ES ダイマーを構成する 2 つのモノマーI，Jの密度行列のみを用いてダイマー電子状態計算を行うことができる. **図１**に FMO 計算の概略を示す．計算のはじめに，モノマー電子状態計算で必要となるモノマー密度行列の初期値を計算する．その密度行列を用いて，各モノマーの電子状態計算を行い，エネルギーや新たな密度行列を求める．この操作を，得られた密度行列と与えた密度行列とが一定の収束条件を満たすまで繰り返す．モノマーの密度行列が収束したら，その結果を用いて，ダイマーの電子状態計算（ダイマーSCF 計算，ダイマーES 近似計算）を行い，(1)式を用いて分子全体の電子状態（エネルギー，密度行列）を計算する．

【並列 FMO プログラムの基本構造】

前述のように，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）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

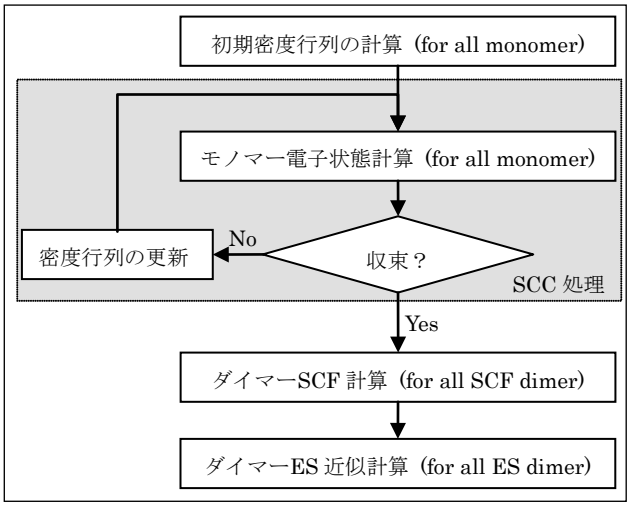

図１：FMO 計算の流れ

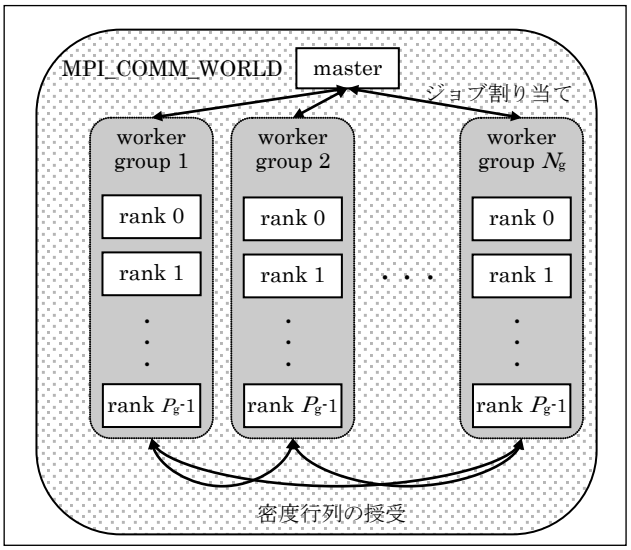

図２：並列 FMO プログラムの基本構造

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２**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，残りが各モノマー，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_g 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_g 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.

超並列実行時のターゲット分子では，フラグメント分割した場合にモノマー数（フラグメント数）が数 1000～数万になると考えているが，その場合には，データサイズの観点から全モノマー密度行列のデータを各ノードで保持することが困難であり，一部プロセス，あるいは，全プロセスで分散保存することが必須となる．その場合には，各モノマー，ダイマーの計算で必要となるモノマー密度行列データが,一般には,計算を担当する worker group 内のプロセスに存在しないため，該当データを保存しているプロセスとデータが必要な worker group のプロセスとの間でデータのやり取りが必要となる.

【モノマー密度行列の取り扱いについて】

これまで述べてきたように，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ができると考えられる．モノマー密度行列の全データ量は入力データの規模に比例して増加するため，入力規模に対する scalability を保つためには，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，複数のノードで分散保存して，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．そのために，今回はモノマー密度行列データの保存方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に入力に対する scalability も高い，という特徴がある．**図３**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〇で示されており，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を用いて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

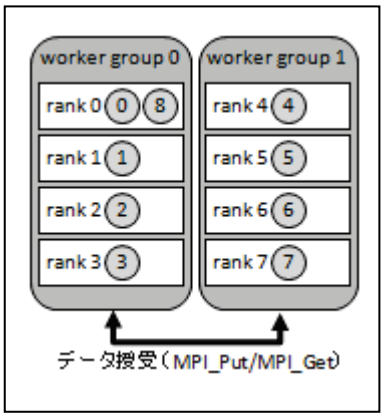

図３：方法１によるモノマー密度行列データの分散保存方法
（ワーカプロセス数＝８，モノマー数＝９の場合）

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．ただし，MPI-2 の片側通信機構を用いてこの方法を実装して性能評価を行った結果,後述のように，片側通信の実装によって，性能が非常に悪くなることが分かった．

そこで片側通信を用いずに，しかも，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**）を考えた．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４**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 関数で待ち，参照要求の場合には要求元へデータを送信し，更新要求の場合には要求元からデータを受信して該当モノマー密度行列データを更新する．一方，ワーカプロセスは，モノマー密度行列データが必要になると，worker group 内の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，データを送信することで更新処理を行う．この方法では，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る．また，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,入力に対する scalability も保つことができる．一方で，ストレージプロセスとワーカプロセスのプログラムを別々に準備する必要があること，また，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ことが欠点である．

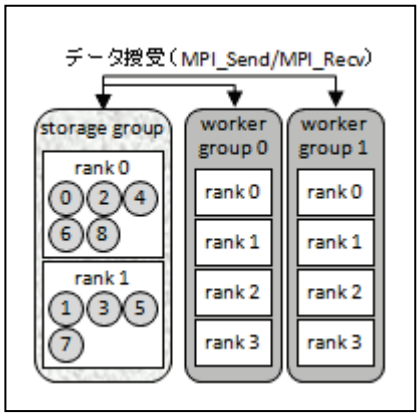

図４：方法２によるモノマー密度行列データの分散保存方法
（ワーカプロセス数＝８，ストレージプロセス数＝２，モノマー数＝９の場合）

今回は，ここで示した２つの方法をモノマー密度行列データの保存，および，アクセスに用いて性能評価を行った．

【性能評価の方法】

実装したモノマー密度行列データの保存，および，アクセス方法に対する性能評価方法を以下に示す．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（version 1.3.1），コンパイラとして Intel C++ compiler（version 12.0.0）を用いた．入力データとして，アクアポリン分子(PDB ID=2f2b，492 フラグメント）とアデノウィルスの KNOB ドメイン（PDB ID=1NOB，575 フラグメント）の 2 種類を用いた．これらの入力データを用いて OpenFMO による並列計算を行い，MPICH2 に付属しているログ取得ツール MPE を利用して，実行プロファイルを取得した．以下に２つの方法それぞれに対する評価結果を示す．

【方法１の評価結果】

まず，ワーカプロセスに分散保存されたモノマー密度行列データへのアクセスに対して MPI-2 の受動的片側通信機構を用いた場合（方法１）の実

行プロファイルを図５に示す．この図は MPE に付属しているプロファイルビューワ jumpshot を用いてモノマー電子状態計算部分のプロファイルの一部を抜粋したものである．横軸は経過時間，縦軸はプロセスのランク番号を表しており，横方向に幅の広い色付きの帯はブロッキング通信や集団通信，あるいは，片側通信を含む非同期通信の同期処理の同期待ちなどの通信処理を表している．また，縦方向の白い線は，ブロッキング 1 対 1 通信や片側通信の要求先と要求元を結んでいる．この例は 13 プロセスでの並列実行であり，ランク 0 のプロセスはマスタプロセスとして動作しているため，ほとんどがワーカからのジョブ要求待ち（MPI_Recv）である．残りの 12 プロセスを６プロセスずつ,2 つの worker group に分割している（ランク１～６のプロセスを worker group 0，ランク７～１２のプロセスを worker group 1 とする）．ランク１とランク７の 2 つのプロセスが，分子積分計算の動的負荷分散で用いている global counter の専用プロセスとして動作しているため，MPI_Recv によるジョブ要求待ちが生じていることが分かる．図５に示した結果で最も特徴的なことは,MPI-2(MPICH2)の片側通信を用いた場合には，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５の経過時間約 2.5 秒～7.5 秒あたりで worker group 1 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 1 の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 0 に含まれていたため，そのターゲットプロセスが MPI 関数を呼び出している見られる経過時間約 7.5 秒付近まで，worker group 1 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の実装を用いると，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が分かった．MPICH2 は OpenMPI と並んで多くの並列計算機で利用できる MPI ライブラリであるため，プログラムの可搬性を考慮すると，MPI-2 の片側通信機能の利用を前提とした方法 1 によるモノマー密度行列データの取り扱いは，現状では好ましくない．

【方法２の性能評価】

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専用プロセスであるストレージプロセスを用いた方法２を用いた場合における性能評価結果を図６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１プロセスをモノマー密度行列データを保存して，アクセス要求に応答する

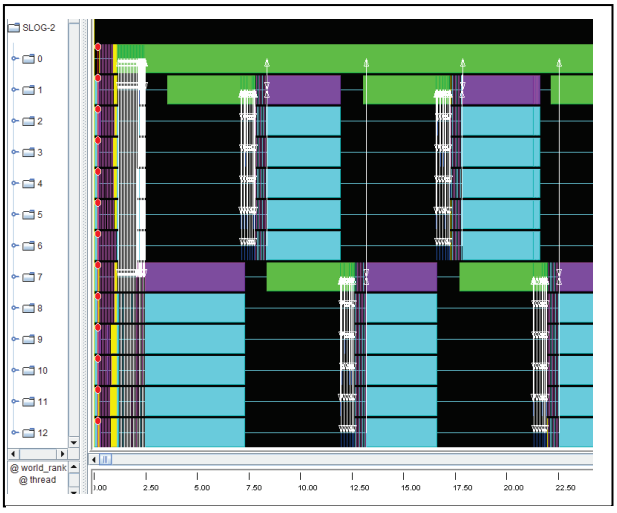

図５：方法１を用いた場合の MPI プロファイル（抜粋）

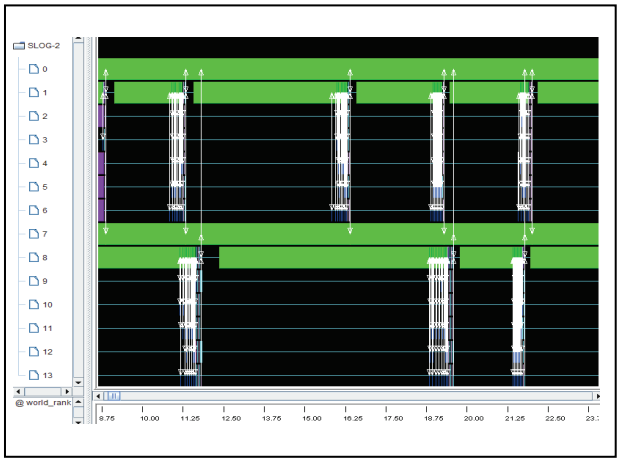

図６：方法２を用いた場合の MPI プロファイル（抜粋）

ためのストレージプロセスであり，残りの 12 プロセスをランク１～ランク６までの worker group 0 とランク８～ランク 14 までの worker group 1 の２つの worker group に分割している．この図を見ると，ランク０のマスタープロセスは，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 global counter のマスタプロセスとなっているランク１とランク 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ランク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．

図７は，モノマー密度行列データへのアクセスを行っている部分を拡大したものである．この図を見ると，worker group 0 のマスタプロセス（ランク１のプロセス）からランク７のストレージプロセスへのアクセス要求（MPI_Send/MPI_Recv に対応する縦方向の矢印）があると，ストレージプロセスがすぐに該当モノマー密度行列データを worker group 0 のマスタプロセスへ送信していることが分かる．また，この部分で worker group 0 に割り当てられたモノマー電子状態計算に必要であった該当モノマー，および，周辺モノマーの密度行列数は 6 つであり，そのデータの転送に，経過時間 16.402 秒～16.404 秒の約 0.002 秒しかかかっていないことが分かる．このように，モノマー密度行列データの保存，アクセスにストレージプロセスを導入した方法 2 を用いると，モノマー密度行列データへのアクセスに伴うワーカプロセスの待ち時間が非常に小さくなることが分かる．さらに，この方法による大規模並列実行時の性能を調べるために，グループ数をさらに増やした実行を行った．使用した計算機は Riken Integrated Cluster of Clusters (RICC) の超並列 PC クラスタ部で，worker group 数 63，worker group あたり 256 並列（＝32 プロセス×４スレッド/worker group）での計算を行った．その際に，ストレージプロセス数が１と２の場合で，モノマー密度行列へのアクセスにかかる時間がどのように変化するかを調べた．入力データはアデノウィルス KNOB ドメイン（PDB ID=1nob，575 フラグメント）で，基底関数として 6-31G(d)を用いた．その結果を表 1 に示す．表１には，worker group 数が１と 63 の場合に，それぞれ，ストレージプロセス数を１，２と変化させた際の，モノマー密度行列データ取得にかかる時間の合計を示している．この結果を見ると，worker group 数が 1，63 のいずれの場合でも，ストレージプロセス数が 1 から２へと増えると，データアクセスが分散するため，密度行列データ取得時間が短くなることが分かる．また，worker group 数が 63 でストレージプロセス数が１という，通信負荷の点で最も厳しい場合でも，

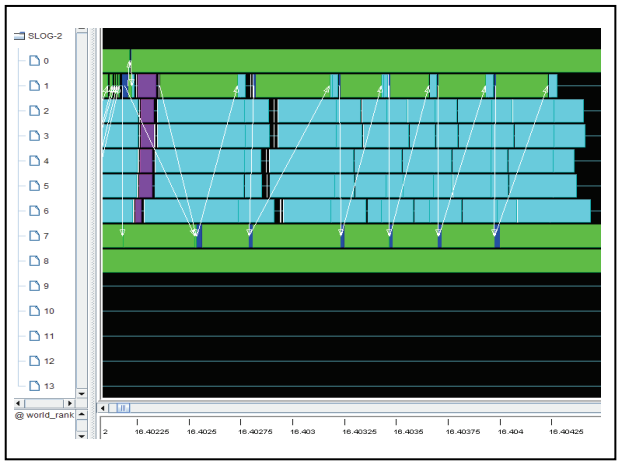

図７：モノマー密度行列データへのアクセスの様子

表 1：密度行列取得時間と worker group 数，ならびに，ストレージプロセス数の関係

ストレージプロセス数	worker group 数	密度行列データの取得に要した時間*（秒）
1	1	37.8(37.8)
2	1	33.2(33.2)
1	63	44.2(0.70)
2	63	37.3(0.59)

*；()内は worker group あたりの平均値

今回の実験で最も通信負荷が小さいストレージプロセス数２，worker group 数１の場合に比べて，密度行列データへのアクセス時間の増加が 11（44.2-33.2)秒と軽微なものであることが分かった．したがって，モノマー密度行列データの分散保存，アクセス方法として，方法 2 を用いた場合では，大規模並列実行時でも大きな性能低下が見られないことが明らかとなった．

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．

(2) 当初計画の達成状況について

今年度の当初計画では，前年度行った小規模電子状態計算（細粒度並列化）部分の高並列化に引き続き，FMO 計算の超並列実行を行う際に性能に影響を及ぼすと考えられるモノマー密度行列データの扱いに対する最適化を行うことにしていた．入力データへの scalability を考えると密度行列データを分散して保存する必要があるため，その分散保存されたデータに対する参照，および，更新処理をどのように行うかを検討して，2 つの方法で実装して，まず，簡単な性能評価を行った．その結果,MPI-2 の片側通信機構を用いた場合には，データアクセスのための待ち時間が非常に大きくなるため，全体の計算効率を著しく低下させる可能性があることが分かった．一方で，モノマー密度行列の保存，アクセス要求を専門に行うストレージプロセスを導入して MPI-1 のブロッキング 1 対 1 通信で記述する方法では，ワーカプロセスからのデータアクセス要求に短時間で応答できることが示された．また，大規模並列時の性能評価の結果でも，方法 2 で効率の良いモノマー密度行列データへのアクセスができることが明らかとなった．

この結果から，モノマー密度行列データへの効率的なアクセス方法を検討して，OpenFMO の並列性能を向上させる，という目標が達成できたと考えている．

4. 今後の展望

今後は,FMO 計算のカーネルコードの一つである，分子積分プログラムの最適化を行い，スカラ性能と並列性能の両方で効率のよいプログラムにしていく必要がある．また，Post Hartree-Fock 法や密度汎関数法などを導入して，より高精度な計算ができるように改良していく必要がある．これらの改良を行い，大規模生体分子に対する精度のよい FMO 計算が高速に行えるようになれば，創薬などの分野での応用が可能になり，より短期間での医薬品開発に貢献できるようなると考えている．

5. 研究成果リスト

(1) 学術論文

なし

(2) 国際会議プロシーディングス

なし

(3) 国際会議発表

1. Yuichi Inadomi, ″Optimization of FMO Code (OpenFMO) for Effective Massively Parallel Execution″, poster P02, The 4th French-Japanese Workshop on Computational Methods in Chemistry,2012.3.5～6

(4) 国内会議発表

1. 稲富雄一，高見利也，本田宏明，小林泰三，あ青柳睦，眞木淳，南一生「OpenFMO におけるフラグメント電子状態計算部分の高並列化」,日本コンピュータ化学会春季年会 2011，ポスター発表３P10,2011.6.15～17,東京工業大学大岡山キャンパス

2. 稲富雄一，眞木淳，本田宏明，高見利也，小林泰三，西田晃，青柳睦，南一生,「並列 FMO プログ

ラム OpenFMO の超並列化への取り組み」, 第 5 回分子科学討論会 2011 札幌, ポスター発表 1P108, 2011.9.20～23, 札幌コンベンションセンター
3. 稲富雄一, 眞木淳, 本田宏明, 高見利也, 小林泰三, 西田晃, 青柳睦, 南一生,「並列 FMO プログラム OpenFMO の超並列化への取り組み」, 日本コンピュータ化学会秋季年会 2011, ポスター発表 1P02, 2011.11.4～5, 福井商工会議所
4. 稲富雄一, 眞木淳, 高見利也, 小林泰三, 青柳睦, 南一生,「並列 FMO プログラム OpenFMO の性能最適化」HPCS2012, ポスター発表 P1-3, 2012.1.25, 名古屋大学
5. 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,「並列 FMO プログラム OpenFMO の性能最適化」, 情報処理学会研究報告 2012-HPC-133, 口頭発表, 2012.3.26～27, 有馬温泉
(5) その他（特許, プレス発表, 著書等）
なし