10-NA05

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

南 一生（理化学研究所次世代スーパーコンピュータ開発実施本部）

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．プロセッサのキャッシュの効率的な利用と通信資源の有効利用と考慮するために，OpenMP と MPI を組み合わせた hybrid 並列化を行った．また，多くのプロセッサを用いた並列性能の評価を行った．その結果，負荷バランスの悪さが並列性能を低下させていることが判明したため，動的負荷分散を用いた負荷分散を行い，並列性能の向上を試みた．

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と思われる．そこで本研究では，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することにした．また，その結果を用いて，理論化学の研究者と協力して，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い，OpenFMO の高性能化を図る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

九州大学

（2）共同研究分野

量子化学，計算物理

（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

【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）に対する電子状態（分子のエネルギー，電子分布の様子など）の計算を行うことで，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 $E_{\mathrm{total}}^{\mathrm{FMO}}$，および，密度行列 $\mathbf{D}_{\mathrm{total}}^{\mathrm{FMO}}$ であるが，FMO 法では以下のように近似する．

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

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 $\left\{\mathbf{D}_K^n\right\}$ と出力 $\left\{\mathbf{D}_K^{n+1}\right\}$ 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．

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

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．

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

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．

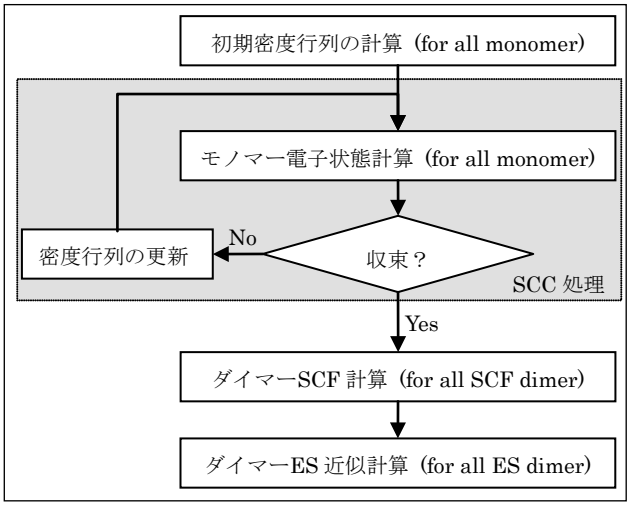

図 1：FMO 計算の流れ

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大粒度並列化と，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 での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ープロセスとなり，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）のグループに分けられる．このグループ単位で，モノマー，ダイマーの電子状態計算を行う（大粒度並列処理）．各グループには複数（図では P_g 個）のプロセスが含まれており，マスター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，マスター - 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，各グループ間でモノマー密度行列データの授受が必要となるため，純粋なマスター - ワーカ型プログラムではない点である．

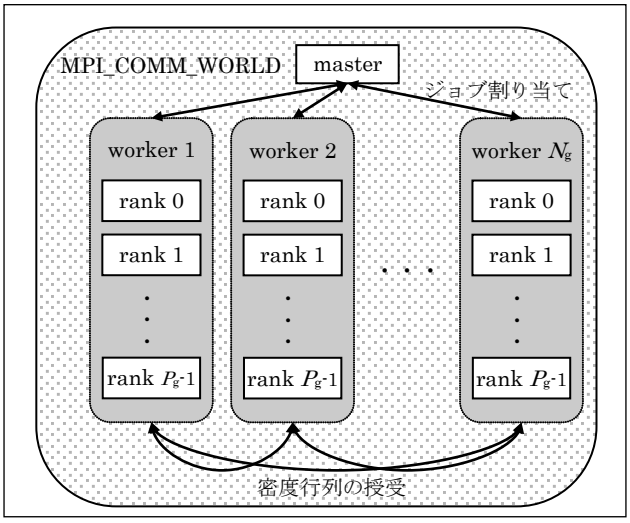

図 2：並列 FMO プログラムの基本構造

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

列のデータを各プロセスで保持することが困難であり，全プロセスで分散保存することが必須となる．モノマー密度行列データの分散保存を行った場合には，各モノマー，ダイマーの計算で必要になる近傍にあるモノマーの密度行列データが，一般には，計算を担当するグループ内のプロセスに存在しないため，グループ間でのデータのやり取りが必要となる．このグループ間のデータ通信を，データを持つプロセスが属するグループの計算の邪魔をすることなく非同期に行うために，OpenFMOプログラムではMPI-2の片側通信機構を利用している．

【フラグメント電子状態計算について】

図 3に細粒度並列部分であるモノマー，ダイマー（以降，モノマーとダイマーをまとめて「フラグメント」と呼ぶ）の電子状態計算部分の模擬コードを示す．FMO 法では，各フラグメントの電子状態計算（ES ダイマー近似を除く）を行うが，そこでは，1 電子積分，2 電子積分，および，4 中心，3 中心，2 中心クーロン積分，と呼ばれる各種分子積分を計算する必要があり，この分子積分計算がフラグメント電子状態計算のほとんどを占めている．各種分子積分計算を 1 回行うために必要な計算時間と，1 回のフラグメント電子状態計算での各種分子積分の計算回数の目安を表 1に示す．このうち，2電子積分は，一般には，繰り返し（SCF）計算の際に複数回計算するが，今回は，多数のノードを用いた並列処理を考えており，フラグメント電子状態計算時に計算した全ての 2 電子積分を保存するのに十分な主記憶が利用できることを仮定しているため，フラグメント電子状態計算 1 回あたり 1 度だけ計算することとしている．4 中心，3 中心のクーロン積分は計算 1 回あたりの計算コストは数 10～1000 秒と比較的大きいが，計算対象のフラグメントの近傍にあるモノマーとの間でしか計算を行わなこと，ならびに，立体的に限られた数のモノマーしか近くにないため，ターゲット分子の規模（フラグメント数）に依らず，計算する回数はほぼ一定である．一方で，2 中心クーロン積分は，1 回あたりの計算時間こそ 0.1～0.5 秒と短いが，ターゲット分子の分割数（フラグメント数）回計算する必要があるため，ターゲット分子の大きさに比例して計算時間が増加する．これらの分子積分は独立に計算でき，かつ，計算箇所のコードが単純なループ構造を持つため，サイクリック分割などを用いた並列処理が容易である．

```
calculation_SCF_energy(int ifrag) {
  V=0.0;
  calculate_2e_integral(ifrag, ERI_buffer);   // 2 電子積分
  for (id=0; id<Nifc4c; id++)
    V += calc_env_pot_4c(id);    // 4 中心クーロン積分
  for (id=0; id<Nifc3c ; id++)
    V += calc_env_pot_3c(id);    // 3 中心クーロン積分
  for (id=0; id<Nfrag; id++)
    V += calc_env_pot_2c(id);    // 2 中心クーロン積分
  calculate_1e_integral(ifrag, Hcore);          // 1 電子積分
  calculate_projection_operator(ifrag, P);    // 射影演算子
  H = Hcore+ V + P;
  SCF_procedure(H, ERI_buffer, Dfrag); // SCF 計算
}
```

図 3：細粒度並列部の模擬コード

各種分子積分の他に，射影演算子の計算，および，繰り返し（SCF）計算が必要になる．このうち，射影演算子の計算は，計算時間が 0.1 秒未満と非常に短いため，並列処理を行う必要はない．また，繰り返し（SCF）計算は，フラグメント電子状態計算 1 回あたり 1 回必要となり，計算時間も数 10 秒必要となるが，最も時間を要するのは，保存された 2 電子積分を用いた Fock 行列計算であるため，並列処理が非常に簡単に行える．

表 1：各種分子積分の計算時間，および，計算回数の目安（フラグメント電子状態計算 1 回あたり）

	1 回あたりの計算時間	呼び出し回数
1 電子積分	0.1～0.5 秒	1
2 電子積分	数 10～1000 秒	1
4 中心クーロン積分	数 10～1000 秒	10～20 回
3 中心クーロン積分	数 10～100 秒	20～30 回
2 中心クーロン積分	0.1～0.5 秒	フラグメント数回

【FMO 計算の超並列化の方針】

並列 FMO 計算では，並列計算に用いるプロセッサ（コア）数が決まった場合，グループ数と各グループ内プロセス数（グループサイズ）との間に任意性がある．グループ数を増やすと，各グループサイズは小さくなり，フラグメント電子状態計算（細粒度並列）部分の並列化効率は一般によくな

る．しかし，各グループに割り当てられるモノマー電子状態計算の数が少なくなるため，大粒度並列処理部分の負荷バランスがとりにくくなる．ここで，フラグメント数 65,536 のターゲット分子の FMO 計算を 64 万プロセッサコアで並列処理することを考える．大粒度並列処理部分での負荷バランスを保つために各グループが平均 32 個のモノマー電子状態計算を行うとすると，グループ数を 2,048 にする必要がある．そうすると，各グループサイズは約 300 となる．この条件で効率的な並列処理を行うためには，各グループで行うモノマーやダイマーの電子状態計算を 300 並列で効率よく処理する必要がある．

そこで本研究では，フラグメント電子状態計算（＝細粒度並列部分）を効率的に並列処理できるように最適化を行うことを主眼に置いて，OpenFMO の超並列化を行うことにした．

【OpenMP/MPI ハイブリッド並列化について】

一般に用いられている並列計算機は，小規模計算機（ノード）を Inifiniband や Myrinet などの高速ネットワークで多数結合した SMP クラスタ型アーキテクチャが大多数を占めている．このような計算機で並列プログラムを効率的に動作させるためには，ノード内とノード間の通信速度差を考慮に入れたプログラミングを行うことが必要となる．また，プロセッサアーキテクチャの主流となっているマルチコアプロセッサには，コア間同期などの機能がハードウェア実装されているものもあるため，並列プログラムにおけるワーカプロセス間の同期をとる場合には，同一プロセッサチップ内とプロセッサチップ間の同期機構の差異を考慮に入れた方が，より並列性能がよくなると考えられる．このように，同期を含む通信性能の特徴を考慮したプログラミング手法の 1 つとして，OpenMP を用いたノード内のスレッド並列と MPI を用いたノード間のプロセス並列を組み合わせた Hybrid 並列が挙げられる．Hybrid 並列化は，MPI ライブラリが用いるメモリ量を削減できること，通信資源の競合を起きにくくすること，あるいは，同一チップ内の複数のコアで共有しているキャッシュを有効利用すること，などのために必要な並列化手法であり，現在，開発が行われている次世代スーパーコンピュータで動作するためのプログラムに対す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，次世代スパコンをはじめとしたクラスタ型並列計算機での超並列実行を効率的に行うために，OpenFMO を Hybrid 並列化することにした．

先に述べたとおり，細粒度並列化部分であるフラグメント電子状態計算では，各種分子積分計算が計算時間のほとんどを占める．そのため，各種分子積分プログラムを Hybrid 並列化した．この分子積分部分は，コード量としては OpenFMO プログラムの 6 割弱（約 31,000 行）を占めるが，Hybrid 並列化で手を加える部分が少ないこと，ならびに，似たようなループ構造を持っている関数が多いため，比較的単純な変更を行うだけで済んだ．

【細粒度並列部分の性能測定】

最適化を行う前のフラグメント電子状態計算部分の並列化効率を知るために，Hybrid 並列化しただけのプログラムを用いて性能測定を行った．使用した計算機は，九州大学情報基盤研究開発センターの PRIMERGY である．コンパイラは富士通コンパイラを，また，MPI は OpenMPI を用いた．入力データとして，アクアポリン（PDB ID＝2f2b，2 アミノ酸残基/フラグメント，492 フラグメント，基底関数 6-31G**）を用いた．グループ数を 1 として，グループサイズを 64，128 で実行し，SCC の 1 ステップ（全モノマーに対して 1 回ずつ電子状態計算を実行）に対する経過時間を測定した．その結果，グループサイズが 64，128 の場合に，それぞれ，24926 秒，16716 秒，かかった．この結果をアムダールの法則に当てはめると，見かけの並列化率が 99.2%であることが分かる．これを基に 300 並列時における細粒度並列部分の並列化効率を求めると，およそ 0.29 となり，理想的な性能向上の 3 割弱の性能しか出ないことになる．したがって，このままでは 64 万並列の超並列実行時での効率的処理を行うことが期待できない．

【性能低下要因の解析】

並列性能を低下させる要因を探るために，MPI プログラムのプロファイル機能を持つ MPE を用いた解析を行った．入力データとしてアクアポリン（PDB ID＝2f2b，2 アミノ酸残基/フラグメント，492 フラグメント，基底関数 STO-3G）を用いて，全モノマーに対する電子状態計算を 1 回だけ行わせて，その際のプロファイルを採った．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, 5 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．得られたプロファイルの一部を図 4 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 1 つのモノマー電子状態計算部分を拡大して出力した結果であり，横軸は経過時間，縦軸はプロセスのランク番号を表している．ランク０のプロセスはジョブの割り振りを行うマスタープロセスであり計算には参加していないため，計算を行っているワーカプロセスの数は 10 である．また，この計算はグループ数を１にしているため，グループサイズは 10 である．この結果を見ると，経過時間 10 秒くらいから 10 個のワーカプロセスがほぼ同時に計算を開始しているが，積分計算が早く終わったプロセスが，計算に時間がかかっているプロセス（ランク２，３，４のプロセス）の計算が終わるまで待っており，

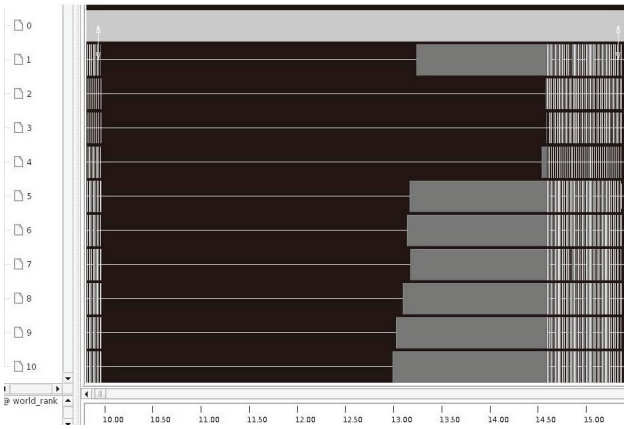

図 4：最適化前の OpenFMO プログラムの実行プロファイルの一部

負荷バランスがとれていないことが分かる．このコードでは，計算時間のかかる各種分子積分計算をサイクリック分割による静的負荷分散により並列処理している．しかし，計算時間削減のためにカットオフ処理を利用しているため分子積分の計算量を予め推定するのが難しいこともあり，静的負荷分散だけで各プロセスに均等に負荷を割り当てることは困難である．図 4 の結果は，静的負荷分散だけを用い負荷バランスを保つことの難しさ示すものであり，この負荷の不均衡が並列性能低下の主な要因の１つであることが分かった．

【共有カウンタを用いた動的負荷分散の導入】

フラグメント電子状態計算部分の並列化効率を上げるためには，各種分子積分の負荷バランスを保つことが必要である．サイクリック分割などの静的負荷分散は，負荷バランスを保つための通信の必要がないため，低コストで負荷分散を行うことができるという利点があるが，各プロセスに割り当てる計算量を事前に推定することが難しい場合には，負荷バランスがとりづらい．このような場合には，グローバル（共有）カウンタを用いた動的負荷分散を用いることで負荷バランスを保つことが考えられ，広く用いられている量子化学計算プログラムパッケージである GAMESS での負荷分散でも採用されている．共有カウンタはグループに含まれるプロセスが共有するカウンタであり，このカウンタの値に排他的にアクセス（atomic fetch and add）することで計算を割り当てて負荷バランスを保つ．この共有カウンタを用いた負荷分散は，ほぼ確実に良好な負荷バランスを保つことを可能にするが，共有カウンタへのアクセスに通信を伴うために，多用すると負荷分散処理自体の通信コストがかさむことになる．そこで，静的負荷分散と共有カウンタを用いた動的負荷分散を組み合わせることで，負荷分散のための通信コスト増加を抑えながら，負荷バランスを保つことにした．具体的には，2 電子積分，4 中心クーロン積分，および，3 中心クーロン積分は静的負荷分散による並列処理を行い，2 中心クーロン積分計算を，動的負荷分散により，並列計算することにした．表 1 に示した通り，2 中心クーロン積分１つあたりの計算時間は 0.1 秒～0.5 秒と非常に短い

ために，１つの 2 中心クーロン積分を並列処理することは考えずに，複数の 2 中心クーロン積分を並列に処理することにした．

共有カウンタを MPI で実装する方法はいくつか考えられるが，カウンタ専用のプロセス（スレッド）を起動しないで済むように，MPI-2 の片側通信機構を利用した実装を行った[1]．グループのランク 0 のプロセスに，グループ内の各プロセスがインクリメントした数を記憶する配列を用意して，この配列の各要素を足し合わせてカウンタの値とする方法である．この実装では，カウンタの値を得るためにグループのプロセス数個の整数データを通信する必要があるため，グループサイズが大きくなるとカウンタの性能が低下するという欠点があるが，スレッド並列化と併用することで，グループサイズがそれほど大きくならないと考えているため，今回は，この方法で実装した共有カウンタを利用することにした．

【共有カウンタ利用の OpenFMO の性能評価】

MPI-2 の片側通信を用いた共有カウンタを利用して 2 中心積分の動的負荷分散を行った OpenFMO プログラムの実行プロファイルの一部を図 5 に示す．実行環境，入力データなどは，最適化前の OpenFMO プロファイル取得時と同じである．この結果は，我々が予期していたものと大きく異なり，通信待ちの時間が非常に大きくなることが分かった．jumpshot を用いて調べたところ，7 秒付近でいくつかのプロセスが 2 中心積分の割り当てを受けるための共有カウンタへのアクセスの際に MPI_Win_unlock 関数で長い時間待たされていることが分かった．1 回のカウンタ値の取得，更新には，「グループのプロセス数 - 1」個の整数データの MPI_Get 操作と１つの整数データに対する MPI_Accumulate 操作が必要になる．今回の場合はグループのプロセス数が高々10 であるため，通信にかかる時間は非常に短いと考えていた．ところが，図 5 の結果を見ると，ワーカグループのランク 0 以外のプロセスでかなりの待ち時間が生じており，中には 0.3 秒以上待たされているプロセスも存在することが分かった．このように，長時間，MPI_Win_unlock 関数で待たされている原因を調べるために，共有カウンタへアクセスしている部分のプロファイルを詳細に調べた．その部分の拡大図を図 6 に示す．整数配列の実体が存在するランク１（ワーカグループにおけるランク０）のプロセスが MPI_Get や MPI_Accumulate などの MPI 関数を呼ぶのとほぼ同時に他のプロセスの MPI_Win_unlock 関数が終了していることがこの図から見てとれる．これは，今回の共有カウンタ実装で用いている受動的片側通信機構が，アクセス対象のメモリの実体があるターゲットプロセスが MPI 関数を呼ぶときに，他のプロセスから要求されている片側通信に応答する，という手段で MPICH2 に実装されているためだと思われる．

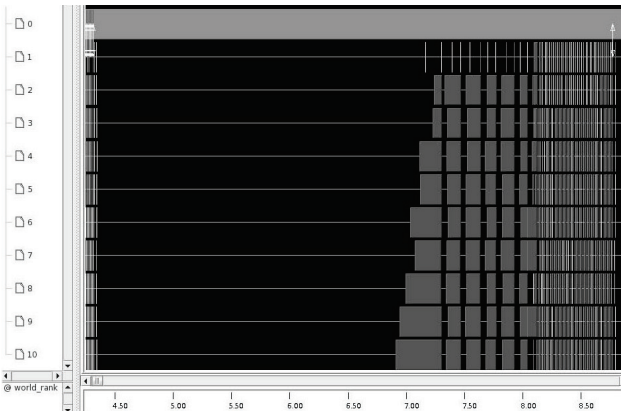

図 5：MPI-2 の片側通信を用いた共有カウンタによる動的負荷分散を行った OpenFMO の実行プロファイル

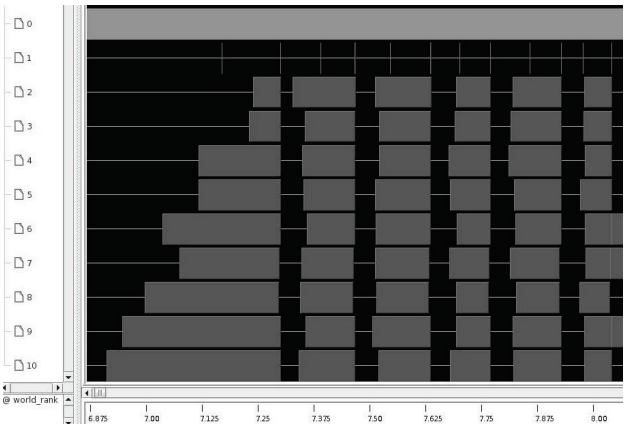

図 6：共有カウンタのアクセス部分の拡大図

参考文献

[1]「実践 MPI-2（メッセージパッシング・インターフェイスの上級者向け機能）」ウイリアム・グロップら著，畑崎隆雄訳，ピアソン・エデュケーション

4. これまでの進捗状況と今後の展望

今回の研究では，FMO 計算プログラム OpenFMO の超並列化を目指して，hybrid 並列化と並列性能向上のための最適化を行うことを目的としていた．既に hybrid 並列化を適用する作業を終え，大規模な並列実行を行うことで，並列性能を低下させる原因の１つが負荷のインバランスにあることを突き止めた，負荷分散を適切に行うための最適化作業を行っているところである．

今後は，負荷分散関連部分のさらなる最適化を行い，並列性能を向上させる必要がある．最適化を行い，1000 並列程度でもフラグメント電子状態計算（細粒度並列化）部分の効率的な実行が可能になれば，疎粒度並列部分を含めた FMO 計算全体の効率的な超並列実行ができる．FMO 計算の超並列実行が可能になり，大規模生体分子に対する 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

なし

(4) 国内会議発表

1. 稲富雄一，眞木淳，本田宏明，西田晃，高見利也，小林泰三，青柳睦，南一生「FMO法の超並列化への取り組み」第4回分子科学総合討論会2010大阪，ポスター発表 2P095，2010.9.14～17，大阪大学豊中キャンパス

2. 稲富雄一，眞木淳，本田宏明，西田晃，高見利也，小林泰三，青柳睦，南一生「並列FMOプログラムOpenFMOの超並列化」日本コンピュータ化学会2010秋季年会，ポスター発表 2P03，2010.10.22～23，長岡技術科学大学（長岡市）

3. 稲富雄一，眞木 淳，高見利也，小林泰三，青柳 睦「OpenFMO の階層化実装と「京」での超並列実行に向けた性能評価について」'物性研・CMSI・次世代ナノ情報合同研究会 「計算物質科学の課題と展望」ポスター発表 C-17 (2011.01.05-07，東大物性研)

(5) その他（特許，プレス発表，著書等）

なし