3P10

OpenFMOにおけるフラグメント電子状態計算部分の高並列化

○稲富 雄一[1], 高見 利也[1], 本田 宏明[2], 小林 泰三[1], 青柳 睦[1], 眞木 淳[3], 南 一生[4]

[1] 九州大学情報基盤研究開発センター（〒812-8581 福岡市東区箱崎6－10－1）

[2]九州大学システム情報科学研究院（〒814-0001福岡市早良区百道浜 3-8-33）

[3] 九州先端科学技術研究所（〒814-0001 福岡市早良区百道浜2-1-22）

[4] 理化学研究所次世代スーパコンピュータ開発実施本部（〒650-0047 兵庫県神戸市中央区港島南町7-1-26）

【はじめに】

フラグメント分子軌道（FMO）法[1-3]は，タンパク質や核酸，糖鎖などの巨大生体分子に対する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．分子を小規模なフラグメントに分割して，各フラグメント（モノマー），ならびに，フラグメントペア（ダイマー）に対する小規模電子状態計算を行うことで，巨大分子全体の電子状態を近似する．各小規模電子状態計算を独立に処理できるため並列処理向きの計算手法であり，既に実装されている並列FMO計算プログラムのGAMESS[4]やABINIT-MP[3]などを用いた計算によって，1000並列程度であれば効率よく計算できることが示されている．FMO法では，複数の小規模電子状態計算を並列に処理（疎粒度並列処理）できるだけでなく，各小規模電子状態計算も更に並列処理（細粒度並列処理）ができるため，数万～数10万並列といった超並列処理も効率的に実行できると考えられている．しかし，並列計算に使用するプロセッサ数（並列度）をこれまで以上に増やした場合，並列処理を行っている部分の計算時間が短くなることによって，通信処理部分や非並列処理部分の処理時間が目立つようになり，並列化効率が低下する．したがって，超並列実行を効率的に行うためには，使用する計算機に適した精緻な最適化を行う必要がある．本研究では並列FMO計算プログラムの1つであるOpenFMOに対して，超並列実行を行うための最適化を行っている．OpenFMOは九州大学と九州先端研で開発された並列FMOプログラムで，Hartree-Fock法を基にしたFMO計算に特化しているため，ソースコードが小規模（約5万行，うち約3万行は分子積分プログラム）である．対象となるコードの規模が小さいほうが，超並列化のための最適化作業がしやすい，ということがOpenFMOを最適化対象とした大きな要因である．今回は，最適化作業の現状を報告する．

【最適化① ハイブリッド並列化】

現在，神戸に作られているスーパーコンピュータ「京」[5]は，8コアのプロセッサSPARC64 VIIIfx（Venus）を1つ搭載したノード8万台以上を高速ネットワークTofuインターコネクトで接続したクラスタ型並列計算機である．近年，プロセッサの「メニーコア化」が進んでいるため，最近の大型並列計算機は，「京」コンピュータと同様のSMPクラスタ型アーキテクチャを持つものが多い．このような並列計算機は，ノード内とノード間の通信速度が大きく異なるため，並列性能を向上させるためには，その通信性能差を考慮したプログラミングが必須となる．そこで，ノード内並列化にはOpenMPによるスレッド並列化を，また，ノード間並列化にはMPIによるプロセス並列化を，それぞれ用いたハイブリッド並列化をOpenFMOに適用した．

```
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 1：小規模電子状態計算部分の模擬コード

【最適化② 共有カウンタを用いた部分的な動的負荷分散の導入】

FMO計算では小規模電子状態計算を多数回，行う必要があり，この部分が計算時間の多くを占めている．したがって，まず，小規模電子状態計算部分についての最適化を行うことにした．この小規模電子状態計算部分の模擬コードを（図 1）に示す．この部分は，2電子積分計算と4中心，3中心，および，2中心のクーロン積分計算などの各種分子積分が主な計算要素となっており，最適化前は静的負荷分散を用いた並列処理を行っていた．静的負荷分散は，計算を各プロセス（スレッド）へ割り当てる際に余分なコストがほとんどかからない，という利点がある一方で，計算時間の予測が難しい場合には，並列性能低下を招く要因の1つである負荷不均衡を生じやすい，という欠点がある．そこで超並列化に向けた最適化の1つとして，負荷バランスを保つために，共有カウンタを用いた動的負荷分散を適用することにした．共有カウンタは，計算に参加しているすべてのプロセス（スレッド）からの排他的な参照，更新が可能なカウンタであるが，そのアクセスには通信を伴うため，多用すると通信オーバヘッドによる性能低下を招く．そこで，今回は，2中心クーロン積分部分にのみ共有カウンタを用いた動的負荷分散を適用することで，過度の通信処理の増加を抑えながら負荷バランスを保つことにした．動的負荷分散導入の効果を調べるために，並列実行時の性能評価を行った．用いた計算機は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 1.3.2p1），コンパイラとしてIntel C++ compiler（version 12.0.0）を用いた．ワーカプロセス数12，プロセスあたり4スレッド（=48ワーカスレッド）での実行を行い，その実行プロファイルを取得した．その結果を図 2に示す．左図と右図は，それぞれ，動的負荷分散導入前と導入後の実行プロファイルである．図の横軸は経過時間，縦軸はプロセス（のrank番号）を表しており，通信に関連している処理をしている部分には，縦線（または矩形）が描かれている．これを見ると，動的負荷分散導入前には各プロセスの分子積分計算終了時刻にばらつきがあり負荷不均衡が生じているが，導入後には，すべてのプロセスがほぼ同時に分子積分計算を終了しており，負荷バランスが非常に良くなっていることが分かる．小規模電子状態計算における分子積分計算の並列化率は動的負荷分散導入前の99.76%から99.99%へと向上し，小規模電子状態計算部分を数100～1000並列で効率的に実行する，という目標に近づいた．

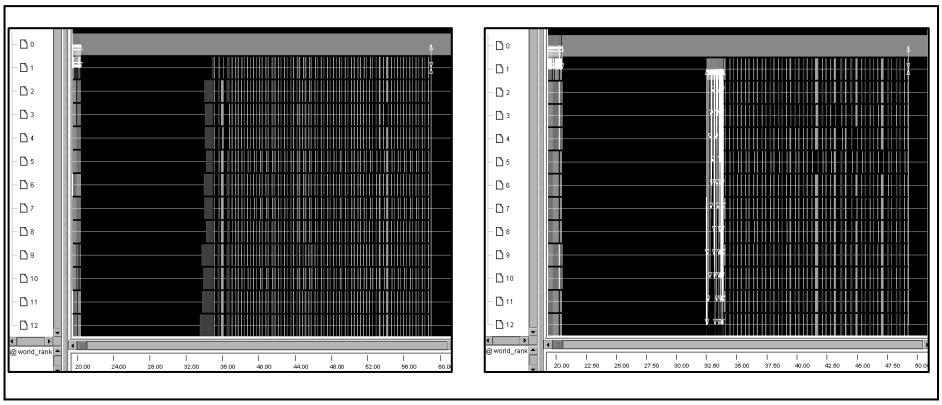

図 2：動的負荷分散導入の効果（左図＝導入前，右図＝導入後）

【謝辞】

本研究は九州大学と理化学研究所との共同研究「次世代スーパーコンピュータシステム向けOpenFMO計算性能最適化に向けた基本設計」，および，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）の支援による．

参考文献：[1] K. Kitaura et al., Chem. Phys. Lett., Vol.312, pp.319-324 (1999) [2] K. Kitaura et al., Chem. Phys. Lett., Vol.313, pp.701-706 (1999) [3] T. Nakano et al., Chem. Phys. Lett., Vol.318, pp.614-618 (2000) [4]M.W.Schmidt et al., J. Comput. Chem., Vol.14, pp.1347-1363 (1993) [5] URL : http://www.nsc.riken.jp/index_j.html