1P108

並列 FMO プログラム OpenFMO の超並列化への取り組み

（1; 九大情基セ、2; 九州先端研、3; 九大シス情、4; 理研）○稲富雄一[1]、眞木淳[2]、
本田宏明[3]、高見利也[1]、小林泰三[1]、西田晃[1]、青柳睦[1]、南一生[4]

【はじめに】フラグメント分子軌道（FMO）法は、タンパク質や DNA、糖鎖などの巨大生体分子に対する量子化学に基づいた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、巨大分子を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。各モノマー、ダイマーの電子状態を独立に計算できるため、複数の小規模電子状態計算を並列に処理できる（疎粒度並列処理）。また、各小規模電子状態もさらに並列処理（細粒度並列処理）できるため、FMO 法は大規模並列処理向きの計算手法と考えらており、現在開発中の「京」コンピュータにおけるターゲットアプリケーションとしても取り上げられている。しかしながら、1 万～数 10 万並列といった超並列処理時に効率的な FMO 計算を行うためには、超並列実行を行うための最適化が必須である。

我々は、FMO 法がどの程度までの並列処理に耐え得るのか、という計算機科学的な観点から、九大などで開発された FMO 計算プログラム OpenFMO の超並列処理に向けた最適化を行っている。本発表では、その現状を報告する。

【最適化の方針】

並列処理ライブラリ MPI を用いた並列 FMO 法のプログラムの基本構造を図 1 に示す。計算を行うプロセス全体を複数のグループ（ワーカ）に分割して、各ワーカがモノマーやダイマーに対する小規模電子状態計算を並列に処理する。各ワーカにも複数のプロセスが含まれており、ワーカに含まれるプロセスで各小規模電子状態計算を並列処理する。FMO 計算の超並列実行を効率的に行うためには、各ワーカに含まれるプロセス（スレッド）数を数 100～1,000 にした場合でも高い効率で小規模電子状態計算を行う必要がある。

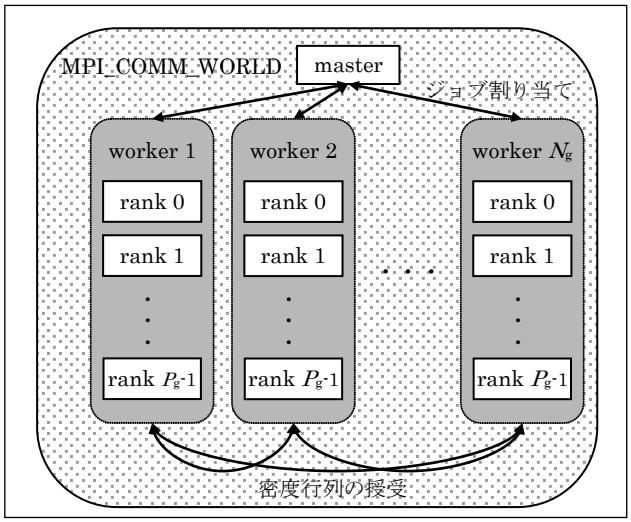

図 1：並列 FMO プログラムの基本構造

図 2 に小規模電子状態計算部分の模擬コードを示す（ダイマー計算における近似ダイマー計算部分は除く）。通常の SCF 計算でも見られる、2 電子積分や 1 電子積分の計算、および、SCF 処理部分に加えて、FMO 計算に特有な各種積分（(4 中心、3 中心、2 中心）クーロン積分）や射影演算子の計算が含まれているが、対象とする分子の規模が大きくなる、すなわち、フラグメント数が多くなると、その計算時間のほとんどが、分子積分計算に

```
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 2：細粒度並列部の模擬コード

なる。そこで今回は、小規模電子状態計算で最も計算量の大きい各種分子積分計算部分の並列処理を最適化することにした。

【最適化前性能評価】まず、最適化前の性能を調べるために、九州大学情報基盤研究開発センターの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）を用いて最適化前のコードの性能評価を行った。使用した MPI は MPICH2(version 1.3.1) でコンパイラは Intel C++ compiler (version 12.0.0) を用いた。入力データはアクアポリン (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*) であり、マスタープロセスを除くワーカプロセス数は 12, プロセスあたり 4 スレッドの計 48 並列で実行した。その実行プロファイルを図 3 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。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 12 である。この結果を見ると、経過時間 20 秒くらいから 12 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、ランク 1 のプロセスの計算が終了するまで他のプロセスに待ち時間が生じており、負荷不均衡を生じていることが分かる。小規模電子状態計算部分で高い並列化効率を達成するには、負荷バランスを改善する必要がある。

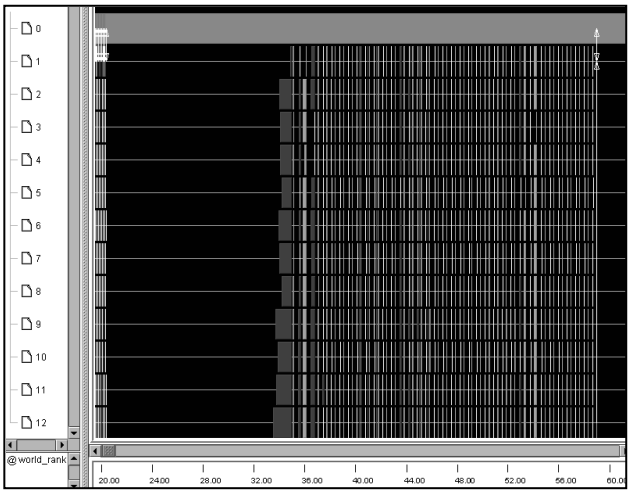

図 3:最適化前の OpenFMO の実行プロファイル（抜粋）

【最適化内容とその結果】負荷バランスを保つためには GAMESS などでもグローバルカウンタを用いた動的負荷分散が用いられているが、カウンタへのアクセスに低速な（ノード間）通信を伴うため、多用すると通信負荷の増加に伴う性能低下が起こる。そこで、分子積分の多くの部分には通信を伴わない静的負荷分散を用いて、2 中心クーロン相互作用項の計算部分にのみ、グローバルカウンタを用いた動的負荷分散（部分動的負荷分散）を適用した。その結果を図 4 に示す。この結果をみると、経過時間 20 秒過ぎから一斉に計算を開始して、32 秒過ぎまで静的負荷分散による 2 電子積分、4 中心クーロン積分, 3 中心クーロン積分を行っているため、通信なしで計算していることが分かる。その後 32.5 秒あたりから動的負荷分散に伴う通信が発生しているが、動的負荷分散を行ったことで、すべてのプロセスがほぼ同時に分子積分を終了していることが分かる。今回のこの最適化で分子積分部分の並列化率が 99.99%に向上していることが分かり、小規模並列処理部分の 1,000 並列での効率的な実行に目星がついた。

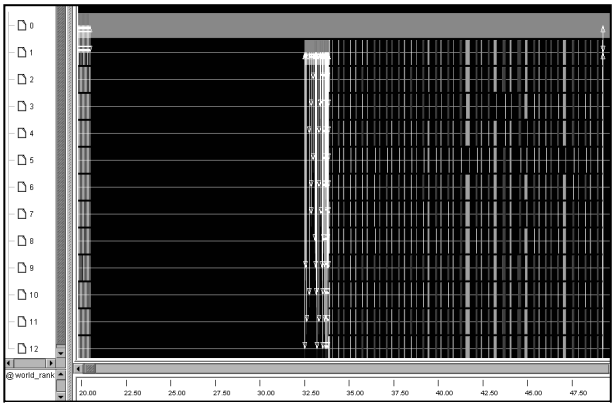

図 4：動的負荷分散導入後の OpenFMO の実行プロファイル（抜粋）