Implementation of Parallel Fragment Molecular Orbital Calculation Program Using One-Sided Communication Functionality

稲富, 雄一
九州大学情報基盤研究開発センター

真木, 真木
九州大学情報基盤研究開発センター

本田, 宏明
九州大学情報基盤研究開発センター

薄田, 竜太郎
福岡県産業・科学技術振興財団

他

出版情報：情報処理学会研究報告, 2007-HPC-111. 2007 (80), pp.85-90, 2007-08. Information Processing Society of Japan
バージョン：

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

稲富雄一*，真木淳*，本田宏明*，薄田竜太郎***，井上弘士**，青柳睦*
*；九州大学情報基盤研究開発センター　　**；九州大学システム情報科学研究院
***；福岡県産業・科学技術振興財団

フラグメント分子軌道（FMO）法に基づいた量子化学計算を，数万～数 10 万プロセッサを用いた超並列計算機で効率よく動作するソースコード作成を目的としている．計算で用いる大容量のデータを分散して保存し，それに対するアクセスを効率よく行うために，MPI-2 で実装されている片側通信機構を用いてコーディングを行っている．作成中のコードとペタスケールインターコネクト技術開発プロジェクトで開発された性能評価ツールなどを用いたベンチマークテストの結果や，通信コストを推定した結果から，データの分散保存を行った並列 FMO コードが，通信コストの面で，超並列計算機でも効率よく動作することが確認できた．

Implementation of Parallel Fragment Molecular Orbital Calculation Program Using One-Sided Communication Functionality

Yuichi Inadomi*, Jun Maki*, Hiroaki Honda*, Ryutaro Susukita***, Koji Inoue** and Mutsumi Aoyagi*
*; Research Institute for Information Technology, Kyushu University, 6-10-1 Hakozaki, Higashi-ku, Fukuoka 812-8581
**; Department of Informatics, Kyushu University, 6-1 Kasuga-Koen, Kasuga, Fukuoka 816-8580
***;Fukuoka Industry, Science & Technology Foundation, Acros Fukuoka Nishi Office 9F, 1-1-1, Tenjin, Chuo-ku, Fukuoka 810-0001

The fragment molecular orbital (FMO) method is one of the promising technique to calculate the electronic structure of large-scale molecules such as proteins and DNAs, and it is suitable for parallel processing. Considering the effective execution of FMO program on massively parallel computers, it is desired to store the density matrix data appearing in FMO calculation in distributed manner. We've been trying to implement the FMO program with the distributed density matrix storage using the one-sided communication functionality in MPI-2 standard. The results of benchmark test using the intermediate code showed that the FMO program with the distributed density matrix storage is more effective than one with non-distributed storage in massively parallel execution.

1. はじめに

フラグメント分子軌道（FMO）法[1-3]は，従来の電子状態計算手法では取り扱いが困難であったタンパクや DNA，糖鎖などの大規模分子の電子状態計算のために開発された計算手法である．FMO 計算では，分子の小片（フラグメント）に対する電子状態計算を行うことで，求めたい分子全体の電子状態を近似的に求める．各フラグメントの電子状態計算は独立に行うことができるために，並列処理向きの計算手法であり，既存の実装である ABINIT-MP[3]や GAMESS[4]を用いた計算で，効率よく並列 FMO 計算が行えることが確認されている．

FMO 計算では，各フラグメントの密度行列というデータを保存しておき，それを繰り返し更新，参照する必要があるが，次世代スパコンでの計算対象となる巨大分子の計算を考えると，密度行列データの総量が数十 GB になる．次世代スパコンは数万～数 10 万プロセッサを用いた超並列計算機であることが想定されるが，そのような計算機で効率よく，スケーラブルに大規模 FMO 計算を行うためには，大容量の密度行列データの保存・参照の方法を工夫して，プログラムを作成する必要がある．今回，従来と同様に，各プロセスがすべての密度行列データを保持する方法と，全プロセスで分散し

て保存する方法の 2 種類の密度行列データの保存・参照方法を考えて，それに基づいた FMO コードの作成を目論んでいる．

ところで，次世代スパコンでのアプリケーションプログラムの性能予測を行うためのツールの開発を，筆者らが携わっているペタスケールインターコネクト技術開発プロジェクト（以下，PSI プロジェクト）で行っている[5]．この開発中のツールは，アプリケーションプログラムから計算部分を削除して通信を含む骨格部分だけの構造にした「スケルトンコード」と呼ばれるプログラムを用いて，次世代のペタスケールスパコンでの実行時性能を予測するものである．我々は，密度行列の保存・参照方法の異なる 2 種類の FMO プログラムに対応するスケルトンコードを作成して，その通信コストの違いを評価した．

2. フラグメント分子軌道（FMO）法の概要

FMO 法は，計算対象となる大規模分子を，30～40 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 $E_{\text{total}}^{\text{FMO}}$，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．

$$E_{\text{total}}^{\text{FMO}} = \sum_{I>J}^{N_{\text{frag}}} E_{IJ} - \left(N_{\text{frag}} - 2\right) \sum_{I}^{N_{\text{frag}}} E_I$$
$$\mathbf{D}_{\text{total}}^{\text{FMO}} = \sum_{I>J}^{N_{\text{frag}}} \mathbf{D}_{IJ} - \left(N_{\text{frag}} - 2\right) \sum_{I}^{N_{\text{frag}}} \mathbf{D}_I \qquad (2.1)$$

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．

$$E_I^{n+1}\left(\mathbf{D}_I^{n+1}\right) \leftarrow f\left(\mathbf{D}_I^n, \left\{\mathbf{D}_K^n\right\}_{K\in\text{neighborhood of monomer } I}\right) \qquad (2.2)$$

式(2.2)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(2.2)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．

$$E_{IJ}\left(\mathbf{D}_{IJ}\right) \leftarrow f\left(\mathbf{D}_I, \mathbf{D}_J, \left\{\mathbf{D}_K\right\}_{K\in\text{neighborhood of monomer } I \text{ and } J}\right) \qquad (2.3)$$

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(2.2)，(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(2.1)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．

$$E_{IJ} \leftarrow f\left(E_I, E_J, \mathbf{D}_I, \mathbf{D}_J\right)$$
$$\mathbf{D}_{IJ} \simeq \mathbf{D}_I + \mathbf{D}_J \qquad (2.4)$$

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．

3. 片側通信を用いた FMO コードの必要性

現在は 1,000 フラグメント～2,000 フラグメントの分子に対する FMO 計算が行われているが, 次世代スパコンが利用可能になれば, 数万～10 万フラグメントの巨大分子に対する FMO 計算を行うことができると考えている．FMO 法は並列処理向きの計算手法であり，すでに，いくつかのプログラムパッケージに MPI などの並列化ライブラリを用いた FMO 計算の実装が行われている．今回，次世代スパコンで効率よく動作する FMO 計算プログラムの開発を１つの目標としている．前述したように，FMO 計算で行う各モノマー，ダイマーの計算の際に，他のモノマーの密度行列データを参照する必要があるので，計算中には，すべてのモノマーの密度行列データが，計算に参加しているすべてのプロセスから参照可能でなくてはならない．従来の実装では，各プロセスがモノマー密度行列データすべてを保存している形式を採っている（以降，**非分散保存**と呼ぶ）．この方法には，

- 電子密度行列を参照する際のコストがほとんどかからない（メモリコピー程度）

という利点があるが，一方で，

- 各プロセスが必要とするメモリ量が，計算対象分子の大きさに比例して増加する
- SCC 計算で必要な, モノマー密度行列データの更新に多くの通信コストを要する

という欠点がある. 特に, 前者は深刻である．たとえば, 10 万フラグメントの通常のタンパク分子に対する FMO 計算を行うことを考えると，モノマー密度行列１つあたりの平均データ量は約 450KB であるため，10 万フラグメントの場合には，すべてのモノマーの密度行列を保存するために約 40GB のメモリが必要となる．これだけのメモリをすべてのプロセスが使用するのは困難である．また，多くの計算ノードをつないだ形式をとるクラスタ型計算機を利用する場合，使用可能メモリの総量が，利用するノード数に比例して増えていく恩恵を得ることが出来ない. したがって, 超並列アーキテクチャを持つと思われる次世代スパコンで実行することを考慮した FMO プログラムでは, 計算に参加しているプロセス全体で, この大容量の密度行列データを分散保持する必要がある．また，フラグメント（モノマー, ダイマー）の電子状態計算を行う際には, 前節の式(2.2), (2.3)で示したとおり，近傍にある，別のモノマーの密度行列が必要になるが, 密度行列の分散保存をした場合には, その必要なデータが，一般には，計算を担当していないプロセスが実行されている計算機に存在するため, 密度行列を持っている計算機の邪魔をできるだけせずに, 必要なデータを得る必要がある. そのために, MPI-2 規格[6]にある MPI_Put, MPI_Get 関数などを用いた片側通信機構を利用することにする．このように，密度行列を分散保持する手法（以降，**分散保存**，と呼ぶ）には,

- 効率を考えると片側通信が必須になる
- 密度行列参照時の通信コストが増大する

といった欠点があるが,

- 計算規模が大きくなっても, メモリ不足になる心配がほとんどない
- FMO 計算のモノマー，ダイマーの電子状態計算では, 対象フラグメント近傍のモノマーの密度行列データしか必要でないため，通信量増加が緩やかだと予想される

という利点もある．次節で，通信コストの推定を行うことにする．

4. 非分散保存, 分散保存の各 FMO プログラムにおける通信コスト比の推定

ここでは，密度行列を非分散保存, 分散保存した場合の通信時間の比を推定する. 密度行列データに関係する通信は，（１）モノマーの電子状態計算で新たに得られた密度行列を保存する更新処理と，（２）保存された密度行列データをフラグメント（モノマー，ダイマー）電子状態計算時に参照する処理，の２つがある．以下の考察は, MPI で並列化された FMO プログラムが，全 MPI プロセスをいくつかのグループに分割して，そのグループに対して，フラグメント（モノマー，ダイマー）の計算を割り振る，という構造を持つ，と仮定して行う．

まず, 非分散保存について考える. この場合，更新処理は，SCC 繰り返し回数だけ，すべて

のモノマー密度行列を broadcast する必要がある．また，参照時には，メモリコピー以外の処理以外は行わないので，それに伴う通信時間は 0 である．従って，非分散保存バージョンの通信時間 $T_{\text{total}}^{\text{non-distributed}}$ は，次式で表わされる．

$$T_{\text{total}}^{\text{non-distributed}} = \bar{N}_{\text{SCC}} \cdot N_{\text{frag}} \cdot \bar{t}_{\text{p2p}} \cdot \log_2 \left(N_{\text{group}} \times P_{\text{group}} \right) \quad (4.1)$$

ここで，$\bar{N}_{\text{SCC}}$，N_{frag}，$\bar{t}_{\text{p2p}}$，N_{group}，および，P_{group} は，それぞれ，SCC 繰り返し回数，フラグメント（モノマー）数，モノマー密度行列 1 つの 1 対 1 通信の平均通信時間，グループ数，および，グループ当たりの MPI ランク数，をそれぞれ表わす．

次に，分散保存を行う場合について考える．分散保存バージョンにおけるモノマー密度行列の更新処理では，グループが担当するモノマー個数分の Put 処理（リモートメモリに対する更新処理）を SCC 繰り返し回数分行うことが必要である．参照時には，参照回数だけ，リモートメモリからの Get 処理（リモートメモリからのデータ取得）とグループ内 broadcast が必要になる．参照回数は，モノマー電子状態計算，ダイマーSCF 計算，および，ダイマーES 計算において，それぞれ，

$$\bar{N}_{\text{SCC}} \cdot N_{\text{frag}} \left(1 + \bar{N}_{\text{ifc4c, monomer}} \right) / N_{\text{group}},$$

$$N_{\text{frag}} \cdot \bar{N}_{\text{SCF dimer}} \left(2 + \bar{N}_{\text{ifc4c, dimer}} \right) / 2N_{\text{group}},$$

$$\left\{ N_{\text{frag}} \left(N_{\text{frag}} - 1 \right) - N_{\text{frag}} \cdot \bar{N}_{\text{SCF dimer}} \right\} / N_{\text{group}},$$

である．ここで，$\bar{N}_{\text{ifc4c, monomer}}$，$\bar{N}_{\text{ifc4c, dimer}}$ は，それぞれ，モノマー計算時，ダイマーSCF 計算時に，近傍にあるモノマー数の平均値であり，$\bar{N}_{\text{SCF dimer}}$ はダイマーSCF 計算を行う相手モノマーの平均数（モノマーあたり）である．従って，分散保存バージョンの通信時間 $T_{\text{total}}^{\text{distributed}}$ は，次のように表わされる．

$$T_{\text{total}}^{\text{distributed}} = \bar{N}_{\text{SCC}} \cdot \frac{N_{\text{frag}}}{N_{\text{group}}} \cdot \bar{t}_{\text{p2p}} + \frac{\bar{t}_{\text{p2p}}}{N_{\text{group}}} \left(1 + \log_2 P_{\text{group}} \right) \times \left\{ \bar{N}_{\text{SCC}} \cdot N_{\text{frag}} \cdot \left(1 + \bar{N}_{\text{ifc4c, monomer}} \right) + \frac{N_{\text{frag}} \cdot \bar{N}_{\text{SCF dimer}} \cdot \bar{N}_{\text{ifc4c, dimer}}}{2} + N_{\text{frag}} \left(N_{\text{frag}} - 1 \right) \right\} \quad (4.2)$$

以上の結果から，非分散保存バージョンと分散保存バージョンの通信時間の比は，次式で表わされる．

$$\frac{T_{\text{total}}^{\text{non-distributed}}}{T_{\text{total}}^{\text{distributed}}} = N_{\text{group}} \bar{N}_{\text{SCC}} \left(\log_2 N_{\text{group}} + \log_2 P_{\text{group}} \right) / \left\{ \bar{N}_{\text{SCC}} + \left(1 + \log_2 P_{\text{group}} \right) \times \left[\bar{N}_{\text{SCC}} \left(1 + \bar{N}_{\text{ifc4c, monomer}} \right) + \frac{\bar{N}_{\text{SCF dimer}} \bar{N}_{\text{ifc4c, dimer}}}{2} + \left(N_{\text{frag}} - 1 \right) \right] \right\} \quad (4.3)$$

この式を見ると，フラグメント数が大きくなると分散保存したほうが不利であるが，グループ数が増えると，逆に有利になることが分かる．

5. スケルトンコードと BSIM Logger

PSI プロジェクトで開発中の性能評価環境を用いた性能評価では，実際のアプリケーションプログラム（以降，実コード）で計算負荷の大きい部分を削除して，代わりに削除された部分の推定計算時間を埋め込んだコード（以降，スケルトンコード）を作成して，そのスケルトンコードを基に，計算性能や通信遅延の評価を行う．実コードのスケルトンコード化の例を**図 1** に示す．図 1 の左側が実コードで，for ループが計算負荷の高い部分だと仮定する．これをスケルトン化する場合には，図 1 の右のコードのように for ループ部分を無効化して，その代わりに，無効化した部分の推定計算時間（ここでは，etime(n)）を引数とした BSIM_Add_time 関数を追加する．

このスケルトンコードを，性能評価ツールの 1 つである BSIM Logger を用いて実行すると，MPI プログラムの実行ログ取得のための MPE 環境を用いた場合と同様の，実行ログ（clog2 形式）が得られる．通常の MPE 環境では，各種イベントの発生時間は，実際に消費した計算時間や通信時間を基にして決められるが，BSIM Logger 環境で得られたログに記録されるイベント発生時間は，スケルトンコード中の BSIM_Add_time で与えられた推定実行時間を基に生成される．さらに，PSI プロジェクトで開発中のネットワークシミュレータ（NSIM）で通信遅延（通信時間）を見積もる予定であるため，BSIM Logger 環境で得られたログには通信遅延 0 として，記録される．

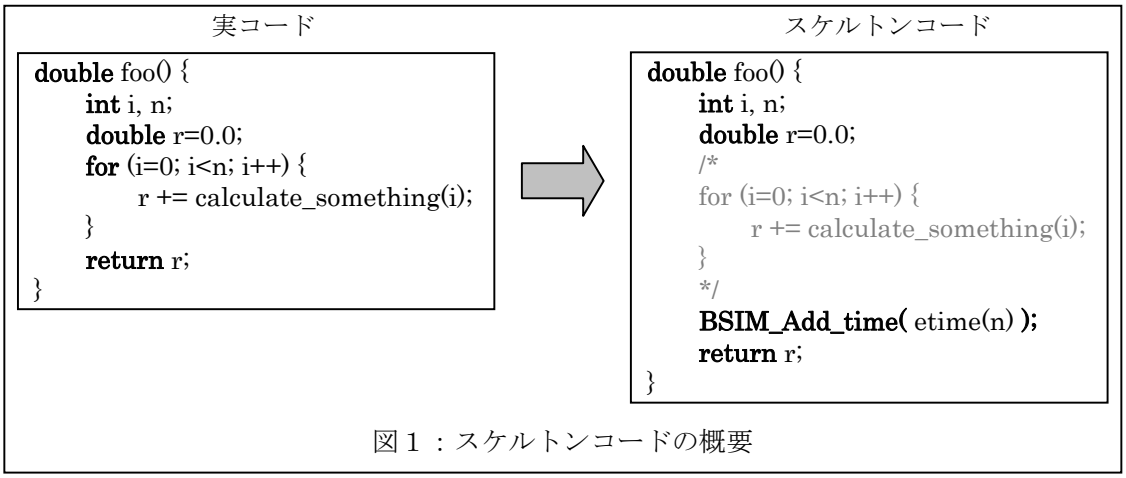

図１：スケルトンコードの概要

6. 並列 FMO プログラムのスケルトンコードとその性能評価

MPI-2 の片側通信機構を用いて，密度行列データを分散保存する FMO スケルトンコードを作成した．このコードを **Open-FMO2** と呼ぶ．このスケルトンコードでは，FMO 計算で行うモノマー，ダイマーの電子状態計算を行う代わりに，BSIM_Add_time 関数で推定実行時間を与えている．推定実行時間には，NAREGI[7]プロジェクトで報告されているフラグメント電子状態計算時間の推定式で得られた値を用いた．この推定式は，実機（Xeon クラスタマシン）での実測値を基にして，フラグメントの基底関数数，および，原子数の関数として決められている．これとは別に，NAREGI プロジェクトで，密度行列データを非分散保存する FMO スケルトンコード（以降，**Open-FMO1**）が作成されているので，この両者を用いて BSIM Logger 環境で実行ログを取ることにより，その性能の差を調べた．ただし，この 2 つの FMO プログラムでは，同じ推定式から得られた値を BSIM_Add_time に与えているので，通信遅延が 0 のままでは，性能差が見られない．そこで，ネットワークシミュレータ NSIM を通す前に，簡易的に通信遅延時間が付加された推定実行時間を求めることにした．この操作を，様々なサイズのデータの 1 対 1 通信における通信遅延時間を，利用可能なネットワークを利用して測定し，その実測データを基に推定通信遅延時間を求めて，その推定値を BSIM_Add_time 関数で MPI プログラムの実行時間に加算する，という手順で行った．MPI_Bcast や MPI_Reduce，あるいは，MPI_Allreduce などの集団通信は，binomial tree ライクな 1 対 1 通信の組み合わせで実装されていることを仮定して，推定遅延時間を求めた．また，MPI_Put や MPI_Get を用いた片側通信の遅延時間（通信開始から MPI_Unlock 関数からの復帰までの時間）も，MPI_Send や MPI_Recv の様な 1 対 1 通信の通信遅延時間と同じである，と仮定した．

このように，経験的な通信遅延時間を付加するように変更した 2 つの FMO コードを用いて，BSIM Logger 環境でのログ採取を行った．この評価の際の入力データはアクアポリン分子（PDB ID=2F2B，14,432 原子，492 フラグメント）で，基底関数は 6-31G*とした．MPI ライブラリは BSIM Logger 用コードが組み込まれた MPICH2[8]を用いた．

図 2 に Open-FMO1 と Open-FMO2 の通信遅延時間の比を示す．この計算では，グループあたり MPI ランク数を 16 に固定して，グループ数を 8～64 まで変化させた．図 2 の横軸は，グループ数，縦軸は 2 つの FMO プログラムの通信遅延時間の比を，それぞれ表わしている．この値が 1 未満だと，非分散保存を行っている Open-FMO1 が通信コストの面で有利であり，1 より大きいときには，逆に，分散保存している Open-FMO2 のほうが有利であることを表わす．また，実線はスケルトンコードと BSIM

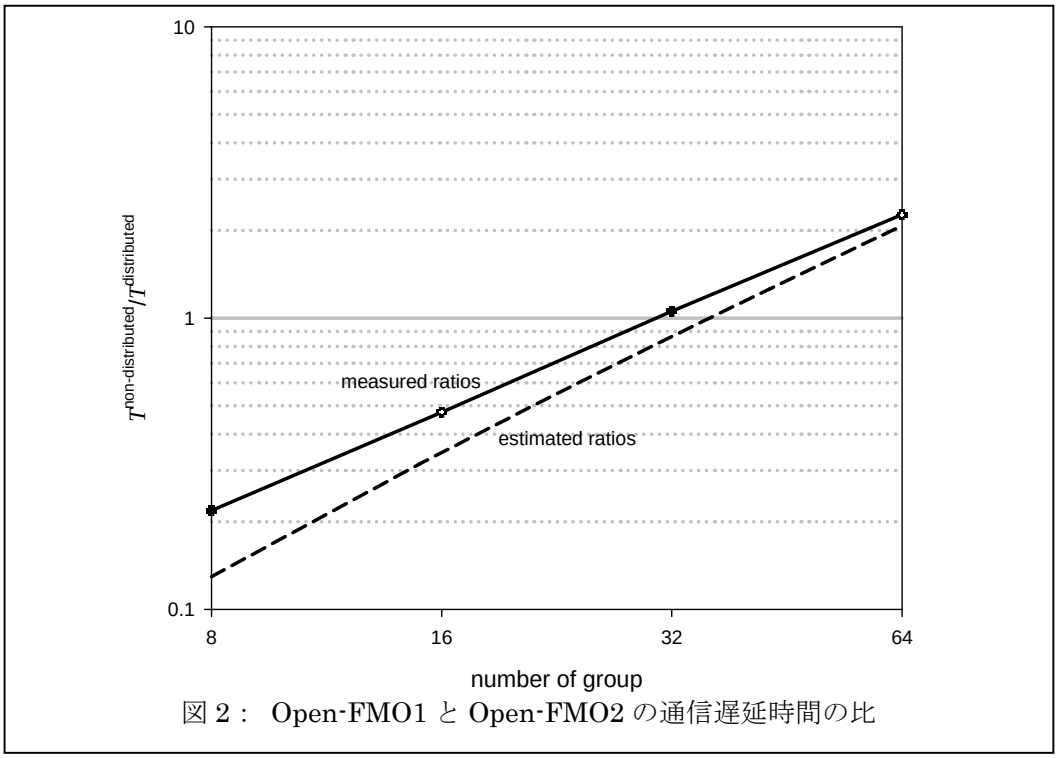

図 2： Open-FMO1 と Open-FMO2 の通信遅延時間の比

Logger を用いて得られた実測値の基づいた結果を，破線は通信コストの比の理論式(4.3)に基づいた結果を，それぞれプロットしたものである．実測値（実線）をみると，グループ数 8 の時には Open-FMO 2 は Open-FMO1 の 5 倍程度の通信コストがかかっているが，グループ数が増すにつれて，両者の差が縮まっていき，32 グループ以降は，Open-FMO2 の方が，通信遅延が小さくなっていることが分かる．多少，数値に差があるが，密度行列の通信遅延時間の比の推定値（破線）の結果も，よく傾向が一致している．

これらの結果から，FMO 計算で用いる密度行列データを分散して保存・参照する実装を行っても，通信コストの増加による性能低下の影響が軽微であり，使用する計算機資源が大きい場合には，むしろ，非分散保存の場合よりも効率が良くなることが分かった．

謝辞

本研究は，文部科学省「次世代 IT 基盤構築のための研究開発」，研究開発領域「将来のスーパーコンピューティングのための要素技術の研究開発」（平成 17 年度～19 年度）における研究開発課題「ペタスケール・システムインターコネクト技術の開発」による．また，計算機を用いた実験は，すべて，理研スーパーコンバインドクラスタ（RSCC）の Linux クラスタを用いて行った．

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]「ペタスケール・システムインタコネクト技術の開発」平成 18 年度報告書
[6]Message Passing Interface Forum, URL=http://www.mpi-forum.org/
[7] National Research Grid Initiative, URL=http://www.naregi.org/
[8] MPICH2, URL=http://www-unix.mcs.anl.gov/mpi/mpich/