課題番号 jh140029-NA15

並列フラグメント分子軌道計算プログラム OpenFMO のマルチプラットフォーム化

渡邊寿雄 （東京工業大学）

概要

京コンピュータなどの SMP クラスタ型超並列計算機に最適化された並列フラグメント分子軌道（FMO）プログラム OpenFMO をベースとして，GPGPU や Intel Xeon Phi などを搭載したアクセラレータ型スパコンへのマルチプラットフォーム化を進めた．マルチプラットフォーム化においては，JHPCN の枠組みを活用し，複数拠点の多様なアクセラレータ型スパコンを利用した．Hartree-Fock 計算ルーチンの GPGPU 化と OpenFMO への組み込みが行われ，FMO 計算での高速化を確認した．

またアクセラレータをより効率的に活用するために，分子軌道計算のカーネルコード（性能ボトルネックである 2 電子反撥積分）の簡素化を検討した．簡素化により高い SIMD 演算率が達成された一方で，計算量が増加したため，実行時間としては既存プログラムよりわずかに増加する結果となった．

1. 共同研究に関する情報

(1) 共同研究を実施した拠点名

東京工業大学 学術国際情報センター
京都大学 学術情報メディアセンター
九州大学 情報基盤研究開発センター

(2) 共同研究分野

□ 超大規模数値計算系応用分野

(3) 参加研究者の役割分担

東京工業大学・学術国際情報センター
- 渡邊 寿雄　課題とりまとめ，簡素化コードの作成/評価
- 青木 尊之　GPU 化の支援

京都大学・学術情報メディアセンター
- 中島 浩　Intel Phi 化の支援
- 河合 直聡　Intel Phi 化/性能評価

九州大学・情報基盤研究開発センター
- 青柳 睦　簡素化コード作成支援
- 本田 宏明　Intel Phi 化/性能評価，簡素化コード作成支援

九州大学・大学院システム情報科学研究院
- 稲富 雄一　OpenFMO のオリジナルコードの改良

筑波大学・計算科学研究センター
- 梅田 宏明　GPU 化/性能評価

産業技術総合研究所・ナノシステム研究部門
- 長嶋 雲兵　簡素化コード作成支援

2. 研究の目的と意義

コンピュータシミュレーションによって原子や電子を露わに扱い化学反応を解明・予測する計算化学は，2013 年ノーベル化学賞を受賞し，方法論やプログラムの開発から様々な応用まで幅広く研究・開発が行われている．その中でもフラグメント分子軌道(FMO)法は，たんぱく質などの巨大分子に対する第一原理電子状態計算を可能にする計算手法として注目されている．いくつかの FMO プログラムではスパコンへ向けた最適化も行われており，稲富(九州大学)らにより開発された FMO プログラム OpenFMO は，京をはじめとした SMP クラスタ型並列計算機で効率的な超並列実行が可能なプログラムである．

昨今，様々なアーキテクチャのスパコンが登場し，特に省電力性能に優れたアクセラレ

ータ型スパコンはエクサへ向けて重要な役割を果たすことが期待されている．しかしながら，同じ計算化学分野のプログラムである分子動力学法がアクセラレータ化に素早く対応したのとは対照的に，分子軌道法のプログラム群のアクセラレータ化はその複雑なカーネルコードが原因で遅れている．そのカーネルコードの複雑さを解消すれば，既存の GPU や Intel Xeon Phi などへのアクセラレータ化が容易になるのみならず，今後新たなアーキテクチャのアクセラレータが登場した際にも他分野のアプリケーションに後れを取ることなく，アクセラレータ化が可能になる．

カーネルコードの複雑さを解消する一つのアイディアとして，電子状態の基底関数展開を最も単純な関数(*s* 型 Gauss 関数)にて再展開する方法がある．このアイディアは非常に古く(1956 年 H. Preuss らの Gaussian Lobe 法など)，アルゴリズムやカーネルコードの簡素化とそれに伴うプログラムの高速化が可能である．この方法に於いては，計算量の増大が起こるため，現在の主要プログラムでは採用されていない．この計算量の増大を克服するには，簡素化したカーネルコードの高速化が必要であり，今後さらに SIMD（Single Instruction / Multiple Data）長が増大するアクセラレータの活用が有効であろう．分子軌道計算の性能のボトルネックは 2 電子反撥積分であるが，簡素化したカーネルコードではそのループ長が揃うため，SIMD 演算の有効利用が可能になる．

カーネルコードの複雑さを解消する他の方法としては，基底関数展開に用いる関数を縮約された基底関数ではなく原始基底関数に基づいて取り扱う方法がある．これにより，カーネルコードの最内ループが容易にアンローリング可能となると共にループボディ部にデータ依存性がなくなることにより SIMD 演算の効率的利用が可能になるであろう．

本研究課題の目的は，並列フラグメント分子軌道計算プログラム OpenFMO のカーネルコードの簡素化と，そのアクセラレータ化により，実用的な FMO プログラムの開発を行うことである．

3. 当拠点公募型共同研究として実施した意義

本研究課題の目的の 1 つであるアクセラレータ化を行うには，多用なアクセラレータ型スパコンを利用できる環境が不可欠であった．当拠点公募型共同研究においては，複数拠点を利用することで，各拠点が持つ多様なアクセラレータ型スパコンの利用が可能であり，本研究課題に必要な環境が揃っていた．また，各拠点の研究者の協力により，当初から予定していた GPU 化のみならず，Intel Xeon Phi 化も実施可能になった．

また，本研究課題のもう 1 つの目的である OpenFMO のカーネルコードの簡素化のメリットは，これまでアクセラレータ化が遅れてきた分子軌道法のプログラム群のアクセラレータ化が容易になることに加えて，計算化学者と計算機科学者によるプログラムの共同開発を容易にすることも大きなメリットである．当拠点公募型共同研究においては，このような学際的な共同研究を推奨しており，本研究課題での共同研究の実施に適していた．

また本研究課題で開発したプログラム群は公開予定であり，OpenFMO プログラムとして直接利用されることに加えて，マルチプラットフォーム化されたカーネルコードが計算化学分野の他のプログラムのマルチプラットフォーム化にも大きな波及効果を及ぼすことが期待される．

4. 前年度までに得られた研究成果の概要

新規課題のため，記入不要

5. 今年度の研究成果の詳細

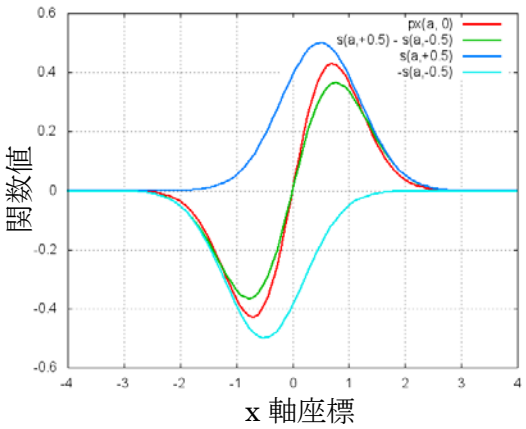

図 1a　p 型関数の s 型関数による再展開

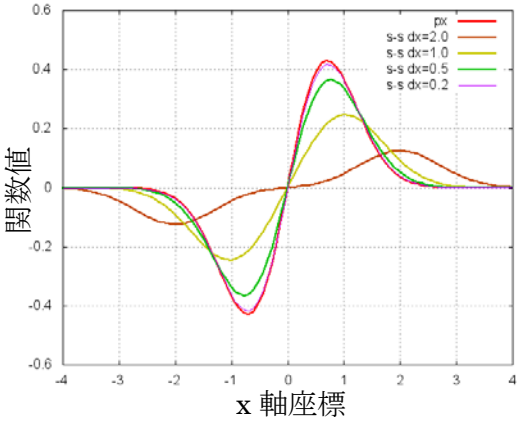

図 1b　p_x 型関数の s 型関数による再展開における軌道中心位置と誤差

（ア）　OpenFMO のカーネルコードの簡素化 1

【概要と目的】

カーネルコードが複雑である原因の 1 つは，HF 法や DFT 法におけるホットスポットである二電子反撥積分計算ルーチンがガウス型基底関数の 4 つ組の積分タイプ毎に必要となる点にある．その複雑さを解消するために，最も単純な s 型基底関数による基底関数の再展開による OpenFMO のカーネルコードの簡素化を行った．カーネルコードの簡素化においては混合精度計算の検討も不可欠であり，それに伴い，下記 4 点の検討が必要である．

(1) 再展開時の近似誤差による精度低下
(2) 再展開後の基底関数の線形従属性による二電子反撥積分の精度低下
(3) 再展開後の基底関数の球面調和性の低下
(4) 再展開後の SIMD 演算活用による高速化

今年度は，(1)～(3)による精度低下について検討を行った．

【結果】

s 型基底関数で p_x 型基底関数を再展開する場合の式を以下に示した．

$$s(\alpha,\boldsymbol{r}) = N(\alpha)\exp(-\alpha\boldsymbol{r}^2)$$
$$p(\alpha,\boldsymbol{r}) = N(\alpha)\,x\exp(-\alpha\boldsymbol{r}^2)$$
$$p'(\alpha,\boldsymbol{r},\boldsymbol{d}) = N'(\alpha)\,(s(\alpha,\boldsymbol{r}+\boldsymbol{d}) - s(\alpha,\boldsymbol{r}-\boldsymbol{d}))$$

α ：軌道指数
$N(\alpha)$ ：規格化定数
$\boldsymbol{d}$：軌道中心のずれ

p_x 型基底関数は共通の軌道指数 α と中心位置のずれが $\boldsymbol{d}$ である 2 つの s 型基底関数の符号が異なる線形結合で近似的に表される．軌道指数 α は中心位置のずれ $\boldsymbol{d}$ に依存し，中心位置のずれ $\boldsymbol{d}$ が 0 に近づくと軌道指数 α は無限大になり，$\boldsymbol{d}$ が 0 の極限では p_x 型基底関数に一致，つまり再展開時の近似誤差がゼロになる．一方で，$\boldsymbol{d}$ が増えるに従い，元々の p_x 型基底関数からのずれが大きくなる．

一方で $\boldsymbol{d}$ が減少することで p_x 型基底関数を構成する 2 つの s 型基底関数の重なりは大きくなり，結果として線形従属性による積分精度の低下が起きる．この積分精度の低下によって SCF 計算が収束しない現象は以前より報告されている．そこで p_x 型基底関数を s 型基底関数で近似する際の一定の誤差を許容し，積分精度を確保する固定値 $\boldsymbol{d}$ を採用することで，SCF 計算が収束することを確認した．

(3) の再展開後の基底関数の球面調和性の低下については，再展開時の s 型基底関数の張る空間の等方性に着目し，3 つの p 型基底関数を 6 つの s 型基底関数にて再展開する際に球面調和性の低下が最も大きいことを確認した．球面調和性の確保には，再展開時の s 型基底関数が張る空間の等方性が不可欠な

ため，s 型基底関数を追加して展開することで球面調和性の担保を行うことを検討中である．

（イ） OpenFMO のカーネルコードの簡素化２

【概要と目的】

今後開発されるメニーコアプロセッサに於いては，搭載される SIMD 演算器の SIMD 長が増大すると予想されるため，その特性を活かして効率的に計算するためには SIMD 演算の割合を高めることが必要である．しかし，現在の分子軌道法計算プログラムでは，そのボトルネックとなる二電子 Fock 行列(G 行列) 計算において効率的な SIMD 演算が難しいことが知られており，現状では効率的なメニーコアプロセッサ利用が困難である．

そこで本研究では前項にて示した通り，s 型基底関数による基底関数の再展開によるカーネルコードの簡素化によってそのループ長を揃えつつ，SIMD 演算器の WAY 数に対して十分なデータ並列性を得ることで，SIMD 演算の有効利用を目指している．

しかしながら，効率的な SIMD 演算を可能にするには，それだけでは不十分であり，条件分岐がないことやメモリ中の連続領域への被演算データの配置など更なる SIMD 演算の阻害要因の排除が必要不可欠である．

そのため，本研究では，基底関数展開に用いる関数を縮約された基底関数ではなく原始基底関数に基づいて取り扱うことによるカーネルコードの更なる簡略化により，より効率的な SIMD 演算の実現を目的とした．

【効率的な SIMD 演算に向けたコード変更】

一般に，コンパイラの自動並列化機能を利用したコード中の多重ループ箇所の SIMD 化オブジェクトコードの生成には，最内ループが容易にアンローリング可能であることならびにループボディ部にデータ依存性が少ないことが必要条件となる．しかしながら，一般的な G 行列計算コードは 8 重ループで構成され，下位 4 重ループ長が上位 4 重ループの変数に依存して 1～6 程度の範囲で変化し，それが実行時に決まる．またループボディ部では条件分岐を含むデータ依存関係のある複雑な計算が行われている．

これらの問題に対し本研究では，これまでの縮約基底ではなく原始基底に基づく方法を試み SIMD 演算の効率的利用を図る．この際，下位縮約ループをアンローリングし上位シェルループへの組み込みを行うというアプローチをとる．さらに，ループボディのコードに対し配列表記や専用ディレクティブを利用してコード変更を行うことで，より明確にコンパイラへ SIMD 演算箇所を伝える工夫をする．

【性能評価】

本研究では，九大の稲富らにより開発されているフラグメント分子軌道法プログラ

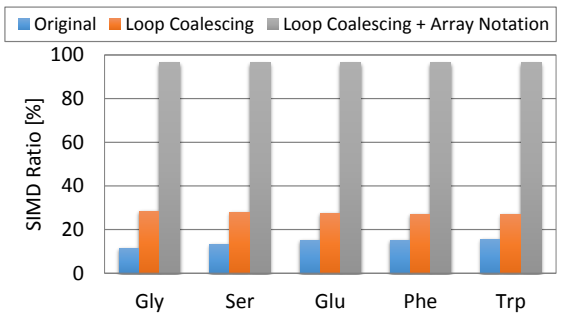

図 2 各アミノ酸における SIMD 演算率

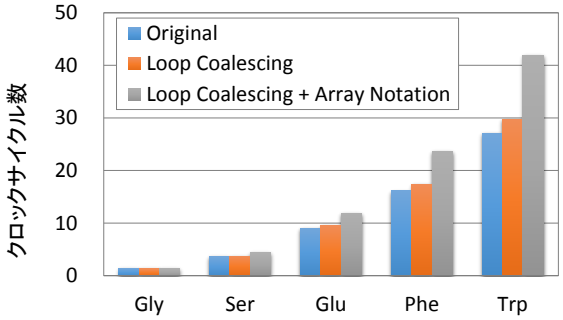

図 3 各アミノ酸でのクロックサイクル数

ム OpenFMO のカーネルコードを使用した.

既存の RHF 法プログラムに対する評価のため，アミノ酸５種(Gly，Ser，Glu，Phe，Trp）に対し上記のプログラムを用いた 1 回の繰り返しのみの RHF 法テスト計算を行った．計算には Intel Xeon E5-2620(6 core)が 2 CPU の PC を用い，2 ならびに 4 WAY の倍精度浮動小数点演算が可能である．Intel Compiler14.0.0 の-O3 最適化での PAPI ライブラリによるハードウェアカウンタの取得結果から，G 行列計算における SIMD 演算率は 5%以下であった.

次に多くの積分タイプの中から(ss|ss)積分に関わる G 行列計算に注目し，Loop Coalescing＋ループボディ SIMD 化した場合の SIMD 演算率と計算に要したクロックサイクル数をそれぞれ図 2，図 3 に示した．図 2 が示す通り，ループボディ部の変更に従って SIMD 演算率は全分子で 96%超を達成することができた．しかしながら，クロックサイクル数を見ると，オリジナル＜ Loop Coalescing＜Loop Coalescing＋SIMD 化 と実行時間は最大で 1.6 倍に増加している．これは Loop Coalescing＋SIMD 化により浮動小数点演算以外の部分によるオーバーヘッドのためである.

今回用いた手法で達成した高い SIMD 演算率が示す通り，現在ある 8 ウェイの SIMD 演算器を持つプロセッサだけでなく，今後開発が予想される 16 ウェイや 32 ウェイの SIMD 演算器を持つプロセッサなど，より長い SIMD 長を持つプロセッサを利用した場合でも効率良く SIMD 演算器を活用できることは確実である．今後メモリ上のデータ配置方法の考慮など SIMD 化コードの最適化も同時に行うことにより，更なる高速化を検討予定である.

（ウ） OpenFMO のアクセラレータ化（GPU 化）

【概要と実装】

筑波大学の梅田らを中心として，HF 計算コードの GPGPU 化を行っている．GPGPU 化を行った Fock 行列計算コードは OpenFMO においてフラグメント(及びフラグメントペア)の計算で利用される HF 計算コードをスケルトンコードとして切り出したものである．C 言語で書かれたスケルトンコードは簡潔であり機能追加や改良も容易となっている．またこのコードへの改良点を OpenFMO プログラムに導入することも比較的容易であり，これにより GPGPU 化した Fock 行列計算コードを FMO 計算に適用することが可能となる.

多くの計算コアを持つ GPU では行列への加算にコストの高い排他制御が必要となるため，これを避けるようなアルゴリズムを開発した．また GPU スレッドの SIMD 動作が効果的に実行可能となるような最適化や，CPU と GPU の同時実行についても実装している．さらに MPI による並列化と合わせることで複数 GPU を利用した高速化も実現した.

組み込みに利用したバージョンの OpenFMO プログラムは動的プロセス生成を利用する MIMD プログラムとなっており，マスタ MPI プロセスとメモリサーバ MPI プログラム，そして多数のワーカ MPI プログラムからなっている．我々のコードを実装するのは，このう

表 1　GPGPU 化 Fock 行列計算コードを用いた FMO 計算の実行時間(秒)

	(Ala)10			Crambin		
Algorithm	SCC	DimerSCF	Total	SCC	DimerSCF	Total
CPU (direct)	219	165	392	1,418	1,606	3,115
CPU (in-core)	212	112	333	1,518	1,446	3,069
GPU+CPU (direct)	196	106	320	1,342	1,269	2,746
GPU+CPU (in-core)	195	85	299	1,391	1,229	2,749

ちでワーカプログラムだけである．GPGPU 化コードの組み込みは 1)GPU の初期化，2)分子データの転送，3)Fock 行列計算ルーチンの差し替え，4)分子データの消去，5)GPU の終了処理の各コードを OpenFMO ワーカコードに挿入することで実装できる．

【ベンチマーク】

OpenFMO に実装した GPGPU 化 Fock 行列計算コードの性能評価は筑波大学の HA-PACS ベースクラスタを用いて行った．HA-PACS ベースクラスタの計算ノードには 2 台の 8 コア Intel E5 CPU(Sandy Bridge-EP, 2.6GHz)と 4 台の Fermi 世代の GPGPU(NVIDIA M2090 GPU)，および 128GB のメモリが搭載されており，それらが InfiniBand により接続されている．複数 GPU を活用するためノードごとに 4MPI プロセスを起動し，それぞれのプロセスが OpenMP 並列で 4 CPU コアと 1 台の GPU を利用することとした．コンパイルや実行には Intel コンパイラ 14.0, CUDA 5.0.35, mvapich2 1.8.1 をそれぞれ利用した．OpenFMO では動的プロセス生成や片側通信を利用するため，MPI 処理系である mvapich2 については適切に環境変数を設定している．

性能評価としてグリシンの 10 量体(112 原子，5 フラグメント)及び Crambin タンパク(642 原子，20 フラグメント)の FMO-HF/6-31G(d)計算を行った．それぞれ計算には HA-PACS ベースクラスタの 2 ノード(8MPI プロセス)および 8 ノード(32MPI プロセス)を用いている．HA-PACS クラスタではノードあたり 128GB と大きなメモリ空間を利用可能なため，FMO 計算のフラグメントサイズ程度であれば二電子積分をメモリに保存しておいて何度も利用する in-core 手法が可能になる．そこで性能評価では CPU や GPU による direct 計算だけでなく，この in-core 計算手法も比較の対象とした．表 1 にはそれぞれの計算手法による実行時間を示した．ここで表中の実行時間は Fock 行列計算が含まれる二つの計算手順(SCC 計算部分，dimerSCF 計算部分)，および FMO 計算全体に対するマスタプロセスにおける経過時間である．我々が実装した GPGPU 化 Fock 行列計算コードは CPU による direct 計算より速いだけでなく，in-core 計算手法を用いた場合よりも高速に動作していることがわかる．

In-core 計算手法よりも高速ではあるが，GPU による高速化は計算全体で 1 割程度でしかない．今回の実装では Fock 行列計算部分のみを GPGPU 化しているため，FMO 計算におけるもう一つのボトルネックである環境ポテンシャル計算部分が高速化されていないことが原因である．現在，Fock 行列計算と同様に二電子積分を利用する 4 中心項部分についての GPGPU 化を検討中であり，これによりさらに大きな高速化が期待できる．

（エ） OpenFMO のアクセラレータ化(Intel Xeon Phi 化)

OpenFMO の Intel Phi 化については，京都大学 学術情報メディアセンターが JHPCN 採択課題向けに募集し採択された大規模計算プログラムの高度化・高性能化支援制度を利用して実施した．

OpenFMO の最適化は，まず Xeon E5-2697 v3 (HSW 2.6GHz 14 コア) の２ソケットシステム上で最適化し，次に同システム上に実装した Xeon Phi 7120A (1.238 GHz 61 コア) 上での単独実行 (native) 向けに最適化した．

HSW 向けの最適化には，インテルの最新コンパイラ(15.0.2,164)を用いて OpenMP で並列化し，HAVX2 命令を使用するようにコンパイルした．Phi 向けには，同コンパイラにて OpenMP で並列化し，512 ビットの SIMD 命令を使用するようにコンパイルした．

最も時間を要する関数のループをベクトル化できるように書き換え，内部関数でベクトル化を行った結果，HSW 上で 3.3%、Phi 上

で 20.1%の性能向上が得られた.

Phi 上での実行では, HSW と同様に誤差関数テーブルがメモリ上に乗るようにできなかったため, HSW の 11 分の 1 程度の性能となった. テーブルサイズは展開項数と要求精度に依存し, 計算量を増やす代わりにテーブルサイズを小さくすることが可能なため, メモリ上に乗るように最適化する必要がある.

HSW と Phi においては同じ条件での比較ができなかったため, バッファーメモリサイズを同じ 32MB にして比較した場合でも Phi 上での性能は HSW の半分弱であった. Knights Corner には誤差関数の演算をベクトル化する機能があるため, その活用による更なる高速化を今後検討している.

6. 今年度の進捗状況と今後の展望

カーネルコードの簡素化に関しては, 当初の実施計画にあった s 型基底関数による基底関数の再展開による簡素化に加えて, 基底関数の縮約を解いた原始基底関数の露わな利用による簡素化にも取り組んだ. 現時点では, 簡素化されたカーネルコードにおける演算数の増加に SIMD 化による高速化が追い付いておらず, 実行時間の増加を招いているため, 更なる最適化の検討が必要である.

カーネルコードの GPU 化と, GPU 化されたカーネルコードの取り込みによる OpenFMO プログラムの GPU 化については計画通りに達成できた. OpenFMO プログラムの GPU 化については, カーネルコード以外の環境ポテンシャル計算部分の GPU 化による高速化を検討中であり, さらに大きな高速化が期待できる.

OpenFMO の Intel Phi 化については, 既存コードを Phi 上での単独実行 (native) 向けに最適化を行った. しかしながら, 性能的には満足いくものではなく, 今後は誤差関数テーブルサイズの最適化や, 誤差関数の演算をベクトル化するなど更なる高速化を試みる必要がある.

7. 研究成果リスト

(1) 学術論文

1. 梅田宏明, 塙敏博, 庄司光男, 朴泰祐, 重田育照, ″GPGPU クラスタ上での FMO 計算の性能評価″, Journal of Computer Chemistry, Japan, 13(6), 323-324(2014)

(2) 国際会議プロシーディングス

なし

(3) 国際会議発表

1. H. Umeda, "Fock matrix preparation with GPGPU for fragment molecular orbital (FMO) calculation", HPC connection workshop @SC14, 招待講演, 2014.11.19, ニューオリンズ(アメリカ)

(4) 国内会議発表

1. 渡邊 寿雄, 稲富 雄一, 梅田 宏明, 本田宏明, 青柳 睦, 長嶋 雲兵「ガウス型基底関数の s 型関数による再展開についての考察」第 8 回分子科学総合討論会 2014, ポスター発表 4P128, 2014.9.21-24, 広島大学 東広島キャンパス
2. 梅田 宏明, 塙 敏博, 庄司 光男, 朴 泰祐, 重田 育照「GPGPU 化 Fock 行列計算ルーチンの OpenFMO への組み込み」第 8 回分子科学総合討論会 2014, ポスター発表 4P119, 2014.9.21-24, 広島大学 東広島キャンパス
3. 渡邊 寿雄「ガウス型基底関数の s 型関数による再展開についての考察」第 2 回 CUTE シンポジウム・コンピュータ化学, 招待講演, 2014.10.30-31, 三重大学 地域イノベーション研究開発拠点
4. 梅田 宏明「FMO 計算のための Fock 行列生成ルーチンの GPGPU 化」第 2 回 CUTE シンポジウム・コンピュータ化学, 招待講演, 2014.10.30-31, 三重大学 地域イノベーション研究開発拠点
5. 今村憲亮, 本田宏明, 稲富雄一, 南里豪志「二電子 Fock 行列の原始基底計算アルゴリズムの提案」, 日本コンピュータ化学

会 2015 春季年会, ポスター発表 1P14, 2015.5.28, 東京工業大学 大岡山キャンパス

6. 梅田宏明, "Development of FMO program for recent HPC systems: K-computer and GPGPU cluster", ポスト京向けアクセラレータについての勉強会, 招待講演, 2014.4.17, 岡崎
7. 梅田宏明, "フラグメント分子軌道法における Fock 行列計算の GPGPU 化", GTC Japan 2014, 招待講演, 2014.7.16, 東京
8. 梅田宏明, 塙敏博, 庄司光男, 朴泰祐, "GPGPU 化した Fock 行列計算ルーチンの OpenFMO への組み込み",日本コンピュータ化学会 2014 春季年会, ポスター発表, 2014.5.30, 東京
9. 梅田宏明, 塙敏博, 庄司光男, 朴泰祐, "GPGPU クラスタ上での FMO 計算の性能評価", 日本コンピュータ化学会 2014 秋季年会, ポスター発表, 2014.10.19, 郡山

(5) その他（特許，プレス発表，著書等）

なし