Vol.2015-HPC-151 No.26
2015/10/1

GPGPU 化 OpenFMO による FMO 計算の性能評価

梅田宏明[†1]　塙敏博[†2]　庄司光男[†1]　朴泰祐[†1]　重田育照[†1]

フラグメント分子軌道（FMO）計算プログラムの一つである OpenFMO の主要部分を CUDA により GPGPU 化し，その性能評価を行った．これまで我々が実装してきた GPGPU 化 Fock 行列計算コードや，新たに GPGPU 化実装を行った 4 中心フラグメント間クーロン相互作用計算を OpenFMO に導入することにより，GPU クラスタ上での FMO 計算を可能とした．筑波大学 HA-PACS ベースクラスタでの性能評価では，CPU だけを利用した計算に比べ 3.7 倍の高速化を実現している．また 64 ノード(256GPU)を使った大規模 FMO 計算では，23,460 原子タンパク質の FMO/HF/6-31G(d)計算を約 2 時間で実行することができた．

1. はじめに

非経験的分子軌道（Molecular Orbital, MO）計算は様々な分野で分子の物性や反応の解析などのための計算化学的なツールとして利用されている．特に近年の計算機性能の向上により創薬分野でのスクリーニングを計算機上でのドッキングシミュレーション等で代用することも現実的になってきている．生体内の反応の解析には大きなタンパク質についての分子軌道計算が必要であり，これを実現するための手法の開発も進められている．フラグメント分子軌道（Fragment MO, FMO）法[1]はこのような大規模分子軌道計算のために開発された分子軌道計算手法の一つであり，京コンピュータ上での大規模並列 FMO 計算も実現されている[2, 3].

大規模分子軌道計算に向けた計算アルゴリズムの開発が進む一方で，GPGPU のような計算加速装置を利用した新しい計算機システムへの対応は未だ一般的でなく，計算化学的な研究や応用に向けて新しい計算機アーキテクチャへの対応がより一層求められている．分子軌道計算の GPGPU 化についてはいくつかのグループが開発を進めているものの[4-9]，その多くのコードは公開されておらず既存コードへの移植には問題があった．我々は計算化学分野で広く使われる非経験的分子軌道計算のベースとなる HF（Hartree-Fock）計算の GPGPU 化に取り組んでおり[10]，これを FMO 計算プログラムに実装することで GPU を用いた大規模分子軌道計算の実現を目指している．

FMO 計算では大きなタンパク質分子を比較的小規模なフラグメントに分割し，他のフラグメントによる環境静電ポテンシャル（electrostatic potential, ESP）影響下での分子軌道計算をフラグメントおよびフラグメントペアについて実行する．計算の主たる要素はフラグメントおよびフラグメントペアについての分子軌道計算部分とフラグメント間クーロン相互作用（Inter-fragment Coulomb interaction, IFC）計算部分となる．このうち分子軌道計算部分は HF 計算であり，我々の実装した GPGPU 化 HF 計算コードを移植することで実現できる．残るフラグメント間クーロン相互作用計算部分について，その一部の GPGPU 化コードを新たに CUDA[11]で実装した．実装に利用した FMO 計算プログラムは九州大学の稲富らによって開発されている OpenFMO で，C 言語で記述された比較的コンパクトなプログラムであり MPI+OpenMP 並列で動作する[12].

本報告では FMO 法の概要と HF 計算の GPGPU 化および，フラグメント間クーロン相互作用計算の GPGPU 化実装について説明する．最後に GPGPU 化した OpenFMO の性能評価として，小規模 FMO 計算によるベンチマーク計算での CPU コードとの性能比較，および大規模 FMO 計算を試みた結果を報告する．

2. フラグメント分子軌道法

2.1 計算手法

通常の分子軌道計算では分子サイズ N に対し形式的に $O(N^4)$の計算量となる HF 計算が必要である．FMO 法はターゲット分子を多数の比較的小さなフラグメントに分割することにより全分子サイズに対する HF 計算を回避する計算手法である．分子を分割した影響を相殺するため，周囲のフラグメントからの影響を環境静電ポテンシャルとして取り込み，これにフラグメント・ダイマーに対する分子軌道計算による二体の相互作用の効果を加えている．環境静電ポテンシャルの取り込みは SCC（self-consistent-charge）プロセスを通して行なわれ，個々のフラグメント（モノマー）に対する分子軌道計算の結果と環境静電ポテンシャルが自己無撞着となるまで繰り返し実行される．SCC プロセスが収束した後，得られた環境静電ポテンシャルのもとでフラグメントペア（ダイマー）に対する分子軌道計算を行う．ターゲット分子全体のエネルギーや電子密度はモノマーおよびダイマーのエネルギーや密度からそれぞれ求められる．これら FMO 計算の手順を図 1 に示した．

環境静電ポテンシャル計算はターゲットになるフラグメント(又はフラグメントペア)とそれ以外全てのフラグメントとのフラグメント間クーロン相互作用として計算される．この計算は計算アルゴリズムとしては Fock 行列計算においてクーロン相互作用項だけを抜き出したものであり（4 中心フラグメント間相互作用），スクリーニングの効果に大小はあるものの Fock 行列計算と同程度の計算量が必要と

†1 筑波大学
†2 東京大学

なってしまう. FMO 法では遠方のフラグメントからのフラグメント間クーロン相互作用を近似的に扱うこと（esp-aoc 近似，esp-ptc 近似）で計算量を削減している．

ダイマー分子軌道計算は全てのフラグメントペアについて行うため, フラグメント数の自乗回の計算が必要となる．フラグメント間の距離が大きいペアについてはフラグメント間相互作用を静電相互作用で近似（ES-dimer 近似）できるため, FMO 計算においては環境静電ポテンシャル計算と同様の近似的取り扱いにより計算量を削減できる．

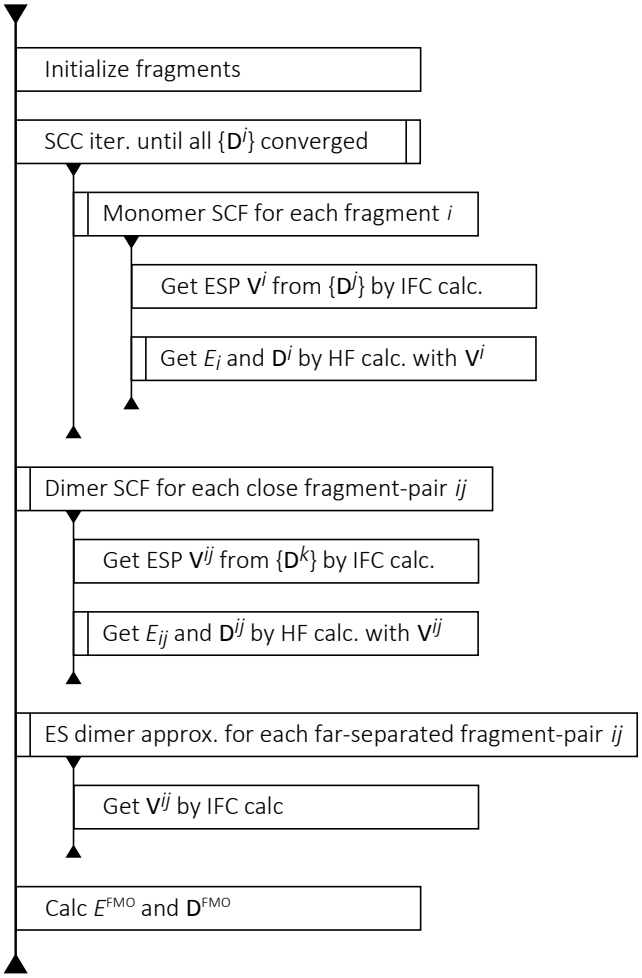

図 1　FMO 計算の手順
Figure 1　FMO calculation scheme.

2.2 FMO 法の実装

FMO 計算はフラグメント･モノマーおよびダイマーに対する HF 計算および静電ポテンシャルの計算からなっている．これらの計算は与えられた静電ポテンシャルの下で独立計算できる．HF 計算や静電ポテンシャル計算自体も並列化することができるため, FMO 計算としては二段階の並列化が可能であり大規模並列計算に適している．SCC のサイクルごとに一度だけ全体の同期を取る必要があるものの, それ以外では同期の必要がなく比較的効率の良い並列実行が可能である．

FMO 法の実装としては汎用の分子軌道計算プログラムである GAMESS[13]や FMO 計算専用の ABINIT-MP(X)[14], PAICS[15]，OpenFMO などがある．OpenFMO 以外の各プログラムでは高精度計算やエネルギー微分等もサポートされており，MD 計算と組み合わせた応用も見られる．大規模 FMO 計算の例としては京コンピュータ 24,576 ノードによる 23,460 原子のタンパク質の FMO/MP2 計算（GAMESS）[3]や地球シミュレータ 128 ノードによる 36,160 原子のタンパク質の FMO/MP3 計算（ABINIT-MP(X)）[16]等がある．BLAS ライブラリを置き換える以上の GPGPU 化が可能なものとしては GAMESS があるが，企業による商用の製品となっておりコードは公開されていない．

OpenFMO は超大規模並列計算機での FMO 計算に向けた FMO 実装であり, 九州大学の稲富らを中心に開発されている．C 言語で記述されたコードは比較的小さくまとまっており，大規模並列計算で必要な動的負荷分散の機構や耐故障性を考慮した実装が可能になっている．現在開発中の OpenFMO では MPI_Spawn による動的プロセス生成版と Falanx ミドルウェア[17]を利用した単一バイナリ版があるが, 今回は動的プロセス生成版を利用する．この OpenFMO ではマスタプロセスから複数のワーカグループとデータサーバを起動し，個々のワーカグループがフラグメント・モノマーおよびダイマーの計算を行う．各ワーカグループは複数のプロセスからなり，MPI+OpenMP による並列化がされている．

3. GPGPU 実装

前章で述べたように FMO 計算では，フラグメント・モノマーおよびダイマーに対する HF 計算とフラグメント間クーロン相互作用が重要な計算要素となっている．このうちフラグメント間クーロン相互作用計算は近距離の場合と遠距離の場合に分けられ，通常サイズの FMO 計算では近距離の 4 中心フラグメント間クーロン相互作用(4C-IFC)計算が大きくなる．一方，非常に大きな分子の FMO 計算では遠距離の部分（esp-ptc 近似）が大きくなることが予想されるため，特別な解法の実装も望まれる．現状 OpenFMO にはそのような実装はされていないため，今回は HF 計算部分と 4 中心フラグメント間クーロン相互作用部分についての GPGPU 化を試みた．OpenFMO のコードは C で書かれており，これを CUDA に移植した．

3.1 HF 計算（Fock 行列計算）

HF 計算においてボトルネックとなるのは Fock 行列の計算となる．これの CUDA による GPGPU 化は既に報告している[10]ので，ここでは簡単に概要を紹介する．

Fock 行列計算は 4 つの縮約シェル ics, jcs, kcs, lcs についての 4 重ループからなり，その内部で二電子積分(ij|kl)が計算される．この二電子積分は対応する密度行列 D の要素と積をとったのち Fock 行列 G に加算される．GPU のように

スレッドが非常に多いプロセッサではG行列をプライベート配列として持つことができず，行列加算に際して何らかの排他制御が必要になってしまう．我々は行列のアクセスパターンを3種類に分類できることを利用し，より小さなプライベート配列への加算としてこのプロセスを記述することでコストの高い排他制御を最小化することに成功した（図2）．なお，ここでijcsおよびklcsは同じフラグメント上の縮約シェルペアであり，行列Gへの加算箇所が両方の縮約シェルに依存していることに注意されたい．

```
bidx = blockIdx.x; nblk = gridDim.x;
tidx = threadIdx.x; nth = blockDim.x;
Gkl[]=0.0;
for (ijcs=bidx; ijcs<Nijcs; ijcs+=nblk) {
  Gi[]=0.0; Gj[]=0.0;
    for (klcs=tidx; klcs< Nklcs; klcs+=nth) {
    if (check_schwarz(ijcs, klcs, Dmax[])) {
      x[] = calc_2e_psss(ijcs, klcs);
      for (i=0,iao=iao0; i<3; i++,iao++) {
        Gi[i*nao+jao] += 4*x[i]*D[kao][lao];
        Gkl[klcs] += 4*x[i]*D[iao][jao];
        Gi[i*nao+kao] -= x[i]*D[jao][lao];
        Gi[i*nao+lao] -= x[i]*D[jao][kao];
        Gj[kao] -= x[i]*D[iao][lao];
        Gj[lao] -= x[i]*D[iao][kao];
      }
    }
  } // klcs
  __syncthreads();
  BlockReduce(Gi[]); BlockReduce(Gj[]);
  for (j=tidx; j<nao*3; j+=nth) {G[iao0][j] += Gi[j];}
  for (i=tidx; i<nao; i+=nth) {G[jao][i] += Gj[i];}
} // ijcs
__syncthreads();
for (klcs=tidx; klcs<Nklcs; klcs+=nth) {
  G[kao][lao] += Gkl[klcs];
}
```

図 2　(ps,ss)タイプ Fock 行列計算の疑似コード
Figure 2　Pseudocode of (ps,ss)-type Fock matrix construction .

さらにGPGPU上で高速化するためのチューニングとして，ボード内で共有するカウンタによるijcsについての動的負荷分散や，Schwarzスクリーニング（check_schwarz()）をklcsループの前に行う分岐の削減，さらに二電子積分計算（calc_2e_psss()）内でのwarp divergence削減のための縮約シェルペアのソートを行なった．

このようなGPGPU化によりHF計算単体の性能としては1.9倍，さらにCPU上での計算とオーバーラップさせることにより全体として3倍の性能を実現している（図3）．図3の性能評価は筑波大学のHA-PACSベースクラスタ1ノード（Intel Xeon E5-2680 ×2, NVIDIA M2090 ×4）による128原子分子のHF/6-31G(d)計算における経過時間である．同じ計算をLIBCCHEMライブラリ[7]によりGPGPU化されたGAMESSを用いて実行した場合に比べても高い性能を見せていることがわかる．

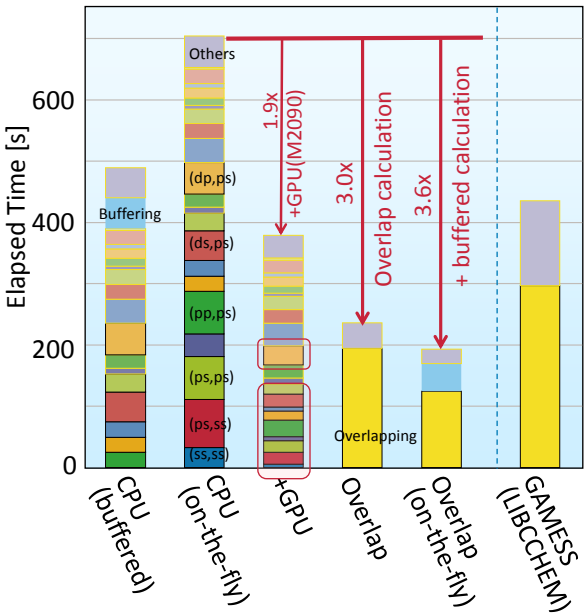

図 3　GPGPU 化 HF 計算の性能評価
Figure 3　Performance benchmark of GPU-accelerated HF calculation.

3.2 4C-IFC 計算

4 中心フラグメント間クーロン相互作用計算はフラグメントAとフラグメントBの間のクーロン相互作用項の計算であり，Fock 行列計算と同様に 4 つの縮約シェル ics, jcs, kcs, lcs についての 4 重ループからなる．ただし 4C-IFC 計算では ics および jcs はフラグメント A に属する縮約シェルであり, kcs および lcs はフラグメント B に属する縮約シェルである．また加算される行列要素はフラグメント A の IFC 行列 V[i][j]のみである．図 2 と同様，ijcs をスレッドブロック，klcs をスレッドにそれぞれ分配するよう並列化した場合，スレッドブロック内で小さい作業配列 Vij[]のリダクション処理ができれば行列加算についての排他制御は必要ないことになる（図 4）．

Fock 行列計算の GPGPU 化で見られた比較的大きなグローバルメモリ上の行配列 Gi[], Gj[]のリダクション及び G 行列への加算がなくなるため，Fock 行列計算よりは GPGPU による高速化が大きくなることが期待される．また 4C-IFC 計算についても Fock 行列計算の GPGPU 化と同様に，動的負荷分散や Schwarz スクリーニングの分離，縮約シェルペアのソート等の高速化を実装した．

```
bidx = blockIdx.x; nblk = gridDim.x;
tidx = threadIdx.x; nth = blockDim.x;
for (ijcs=bidx; ijcs<Nijcs(A); ijcs+=nblk) {
  double Vij [3]=0.0;
  for (klcs=tidx; klcs< Nklcs(B); klcs+=nth) {
    if (check_schwarz(ijcs, klcs, Dmax[])) {
      x[] = calc_2e_psss(ijcs, klcs);
      for (i=0; i<3; i++) {
        Vij[i] += 4*x[i]*D(B)[kao][lao];
      }
    }
  }
  __syncthreads();
  BlockReduce(Vij[]);
  if (tidx==0)
    {for (i=0; i<3; i++) V(A)[iao][jao] += Vij[i];}
}
```

図 4　(ps,ss)タイプ 4C-IFC 計算の疑似コード
Figure 4　Pseudocode of (ps,ss)-type 4C-IFC calculation.

4. 性能評価

本章では GPGPU 化 OpenFMO についてのベンチマーク結果について報告する．性能評価は筑波大学の HA-PACS ベースクラスタ[18]を利用した．この GPU クラスタの計算ノード 1 ノードには 16 コアの CPU と 4 台の Fermi 世代の GPU(M2090)が搭載されている．我々の GPGPU 化実装では 1 MPI ランクに 1 台の GPU を割り付ける構成を想定しているため，MPI プロセスとしては 1 ノードあたり 4MPI ランクを 4 OpenMP スレッドの環境で起動させている．コンパイル環境としては Intel C コンパイラ 15.0，CUDA Toolkit 6.5，IntelMPI 5.0 を利用した．OpenFMO は MPI_Spawn()による動的プロセス生成版を利用しており，マスタプロセスとデータサーバにそれぞれ 1 MPI ランクが割り当てられている．

4.1 GPGPU 化 4C-IFC 計算の性能評価

4 中心フラグメント間クーロン相互作用計算についての性能評価をするため，アラニン 10 量体（112 原子）の FMO-HF/6-31G(d)計算を行なった(表 1)．分子は 5 フラグメントに分割したものを 1 ワーカグループ構成で実行し，SCC 時の 4C-IFC 計算時間を積分タイプごとに積算した．この時，1 ワーカグループが 1 ノードとなるように調整している．表 1 において No.は CPU における経過時間の大きい積分タイプから順に並べた場合の順番であり，15 番 19 番となる(dp,dp)，(pp,pp)および表にないそれ以外の積分タイプについては CPU で実行することとした．量子数の大きい積分タイプでは GPU による高速化はあまり効果的でないことがわかっており，CPU で計算して GPU 計算とオーバーラップさせた方が性能的に有利だからである．今回の実装では CPU の部分と GPGPU 化された部分が同じ程度の計算時間となるよう GPGPU 化する積分タイプを選択している．表中の Speedups は 16 CPU コアによる実行時間と 4 GPU による実行時間の比である．

最も効果の大きい積分タイプ(ss,ds)では 9.6 倍の性能向上が見られ，これは 1 CPU コア/ 1 GPU 比で 38 倍に相当する．Fock 行列計算では GPU による性能向上の最も良いケースで(ss,ss)タイプの 22 倍であることを考えると高い性能となっている．一方で(pp,ds)等ではあまり性能が出ていないが，CPU 計算とのオーバーラップを考慮すれば全体としての性能の向上が期待できる．

表 1　GPGPU 化 4C-IFC 計算の性能評価
Table 1　Performance of GPU-accelerated 4C-IFC calculation.

No.	2e-Type	CPU Etime [s]	GPU Etime [s]	Speedups
1	(ps,ps)	27.36	2.91	9.4
2	(ss,ss)	18.99	2.88	6.6
3	(ss,ps)	16.05	1.95	8.2
4	(ps,ss)	15.20	2.45	6.2
5	(ds,ps)	8.67	2.48	3.5
6	(ps,ds)	7.79	1.65	4.7
7	(ps,pp)	7.30	3.06	2.4
8	(pp,ps)	6.97	3.75	1.9
9	(ds,ss)	6.54	0.89	7.4
10	(dp,ps)	6.52	3.67	1.8
11	(ss,ds)	6.16	0.64	9.6
12	(ps,dp)	5.69	3.06	1.9
13	(ss,pp)	5.50	0.75	7.3
14	(pp,ss)	4.99	1.01	4.9
15	(dp,dp)	4.70		
16	(ds,pp)	3.93	2.80	1.4
17	(dp,ss)	3.84	0.96	4.0
18	(ds,ds)	3.71	1.92	1.9
19	(pp,pp)	3.63		
20	(ss,dp)	3.60	0.61	5.9
22	(pp,ds)	3.33	2.40	1.4
36	(ss,dd)	0.80	0.29	2.8

4.2 GPGPU 化 FMO 計算の性能評価

FMO 計算全体としての性能向上を評価するため，1,961 原子のリゾチウムタンパク質（図 5）の FMO-HF/6-31G(d)計算をベンチマーク計算として実行した．フラグメントあたり 2 残基で分割し，57 フラグメントとなる．

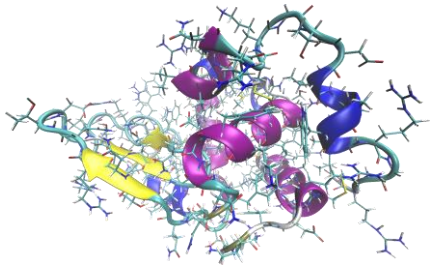

図 5　リゾチウムタンパク質（1,961 原子）
Figure 5　Lysozyme (1,961 atoms).

表 2 にベンチマーク計算における計算プロセスごとの実行時間を示した．計算にはマスタプロセスおよびデータサーバプロセスを含めて 8 ノードを利用し，15 ワーカグループ（各 2 MPI ランク）に分割して計算した．表中 on-the-fly は二電子積分を必要な時に毎回計算する手法であり，buffered は SCF 計算の直前で一度全ての二電子積分を計算しそれをメモリに保存して再利用する手法である．図 3 に見られるように，一般には FMO の HF 計算のような比較的小さな分子に対する HF 計算では全ての二電子積分をメモリ上に保存することができ，buffered 手法が有効である．しかし表 2 の SCC においては buffered 手法の十分な効果が得られていない．これは SCC が収束するに従い SCF の初期値が改善するため，全ての二電子積分を事前に計算するオーバーヘッドが大きくなったからと思われる．

表 2　GPGPU 化 FMO 計算の性能評価（リゾチウム）
Table 2　Performance of GPU-accelerated FMO calculation for lysozyme (FMO-HF/6-31G(d)).

	CPU on-the-fly	CPU buffered	CPU+GPU buffered	CPU+GPU on-the-fly	Speedups
#Node	8	8	8	8	
T(SCC) [s]	3,070.5	3,026.0	905.0	827.5	3.7
T(Dimer SCF) [s]	6,246.2	4,968.8	1,725.6	1,674.8	3.7
T(ES Dimer) [s]	407.3	407.3	77.8	77.8	5.2
T(Total) [s]	9,770.5	8,408.5	2,723.8	2,596.8	3.8

GPGPU 化した計算では，on-the-fly 計算同士の比較で 3.8 倍の高速化を実現している．特に ES Dimer 計算では 5.2 倍と高い性能を発揮している．SCC や Dimer SCF では Fock 行列計算に費やす時間が多いため，GPGPU 化 Fock 行列計算の性能である 3 倍程度の性能となっている．一方で ES Dimer はフラグメント間クーロン相互作用計算だけであり，このサイズの分子では今回新たに GPGPU 化した 4C-IFC 計算の比率が大半である．すなわちここで得られた 5.2 倍という性能は，表 1 に見られる性能向上に CPU 計算とオーバーラップしたことによる効果を加えた 4C-IFC 計算での GPGPU 化による性能向上を示している．

強スケーリング並列性能の評価のため，利用ノード数を 16 および 32 ノードにした FMO 計算も行なった．特に今回はワーカグループ数を 15 に固定することとし，ワーカグループ内の並列性能を評価した（表 3）．表中の括弧内は並列化効率を示している．16 ノード（1 ワーカグループ 4 MPI ランク）では高い並列化効率を示しているが，32 ノード（1 ワーカグループ 8 MPI ランク）では性能が落ちてしまっている．FMO 計算ではフラグメントサイズが小さいため縮約シェルペアの数が少なく，その結果 GPU のように高い並列度を要求するアーキテクチャでは十分なタスク数が確保できなくなっていると考えられる．これを避けるには積分タイプごとに別の GPU に割り振るなど，より高度な並列化が必要である．

表 3　FMO 計算の並列性能評価（リゾチウム）
Table 3　Parallelization performance of GPU-accelerated FMO calculation for lysozyme (FMO-HF/6-31G(d)).

	CPU+GPU on-the-fly	CPU+GPU on-the-fly		CPU+GPU on-the-fly	
#Node	8	16		32	
T(SCC) [s]	827.5	450.2	(0.92)	308.2	(0.67)
T(Dimer SCF) [s]	1,674.8	898.4	(0.93)	530.7	(0.79)
T(ES Dimer) [s]	77.8	42.2	(0.92)	24.8	(0.78)
T(Total) [s]	2,596.8	1,429.6	(0.91)	902.0	(0.72)

4.3 大規模 FMO 計算

我々の実装した GPGPU 化 OpenFMO を利用して大規模 FMO 計算を試みた．ターゲット計算はインフルエンザ HA3 タンパク質（23,460 原子，図 6）[19]の FMO-HF/6-31G(d) 計算である．フラグメント分割は 1 フラグメントあたり 2 残基を基準に 721 フラグメントに分割した．

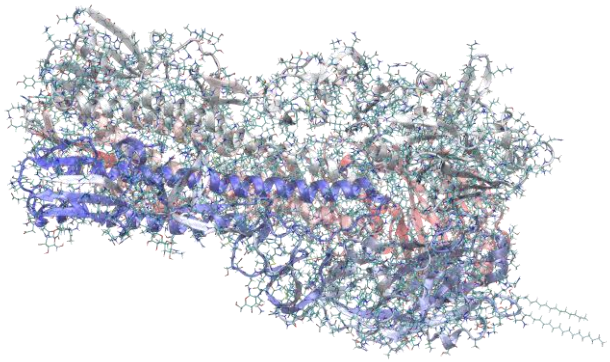

図 6　インフルエンザ HA3 タンパク質（23,460 原子）
Figure 6　Influenza HA3 protein (23,460 atoms).

計算に利用したのは筑波大学の HA-PACS ベースクラスタ 64 ノードで，GPU の数としては 256 台（256 MPI ランク）になる．ワーカとして利用できる 254 MPI ランクは 84 グループに分割し，各ワーカグループ 3 MPI ランクの構成とした．

表 4　GPGPU 化 FMO 計算の性能評価（HA3）
Table 4　Performance of GPU-accelerated FMO calculation for HA3 protein (FMO-HF/6-31G(d)).

	CPU+GPU
#nodes	64
#groups	84
T(SCC) [hr]	0.52
T(Dimer SCF) [hr]	0.90
T(ES Dimer) [hr]	0.45
T(Total) [hr]	1.97

GPGPU 化 OpenFMO による HA3 タンパク質の大規模 FMO 計算を 2 時間で実行することができた．動的プロセス生成版の OpenFMO には SCC 計算後にワーカグループ構成

を変更する機能がないためフラグメント・ダイマーの計算でも SCC 計算と同じワーカグループ構成としているが, 一般にはワーカグループ数を増加させた方が性能は良くなることが知られている．先に示したようにフラグメント内の並列化には限度があるため，フラグメント・ダイマーのようにタスク数が多い場合にはより小さなワーカグループを多数用意した方が有利になるからである．Falanx 版 OpenFMO ではこの制限が取り払われているため，さらなる高速化も期待できる．

4.4 今後の課題

前節のような大規模 FMO 計算ではフラグメント間クーロン相互作用計算の割合がかなり大きくなってくる．これはフラグメントペアの数がサイズの自乗で増加するためである．特に水などの溶媒を露に取り込んだ FMO 計算をする場合などでは，フラグメントのサイズが小さいため HF 計算等の負荷が軽く且つフラグメントの数自体も多くなり, 遠隔フラグメント間のクーロン相互作用計算の割合が大幅に増大すると予想される．この部分の高速化は OpenFMO の今後の課題の一つである．

一方でそのような大規模な FMO 計算を日常的に行うケースはあまりなく，中規模のタンパク質受容体と比較的小さなリガンドとの結合シミュレーションのようなケースを想定する方が現実的とも言える．この場合にはむしろ強スケーリング性能が重要となってくるため，ワーカグループ内での並列性能の向上が課題となる．また繰り返しごとに同期が発生する SCC プロセスでは, ワーカグループ間の負荷分散の改善も課題となる．

5. 関連研究

GPU を用いた FMO 法の実装としてはクロスアビリティ社の古川らによる GAMESS の FMO 計算の高速化の例がある[20]．この実装では HF 計算については CPU を使った buffered 手法を用い，ESP 計算部分を GPGPU 化している．彼等の GPGPU 化コードによる中規模 FMO 計算の結果では GAMESS 本体よりも 3.33 倍の高速化を実現している．使用した計算機システムや基底関数のサイズ，またフラグメント分割等の条件が異なるため，直接的な比較はできないが同程度の高速化になっていると思われる．また GAMESS 本体が OpenMP 並列に対応していないことも大規模並列計算への障害になる可能性がある．

最近の大規模 FMO 計算の例としては Alexeev らによる 17,767 原子タンパク質の FMO 計算（GAMESS, BlueGene/P, 163,840 コア）[21]や望月らによる 36,160 原子タンパク質の FMO/MP3 計算（ABINIT-MP(X)，地球シミュレータ，128 ノード）[16]，稲富らによる 16,764 原子タンパク質の FMO 計算（OpenFMO，PRIMERGY CX400，20480 コア）[22]，著者（梅田）らによる 23,460 原子タンパク質の FMO/MP2 計算（GAMESS，京コンピュータ，24,576 ノード）[3]などがある．特に最後の京コンピュータの結果は本論文で利用した HA3 タンパク質と同じものであり，計算レベルが RI-MP2 であるものの約 11 分で計算を完了している．なおこの京コンピュータ用 GAMESS-FMO プログラムは MPI+OpenMP による並列化に加えクロスアビリティ社の SIMD 化 ESP 計算コードも使用されている．

6. おわりに

本発表では大規模分子軌道計算向けアルゴリズムである FMO 計算について，その実装の一つである OpenFMO を GPGPU 化することにより高速化を試みた．GPGPU 化は FMO 計算でボトルネックになる Fock 行列計算部分と 4 中心フラグメント間クーロン相互作用計算部分について実装した．これを利用した FMO 計算は CPU のみを使った on-the-fly 計算より 3.8 倍の高速化を実現している．大規模 FMO 計算の評価として 23,460 原子のタンパク質の FMO-HF/6-31G(d)計算を HA-PACS ベースクラスタ 64 ノードを用いて実行したところ，約 2 時間で計算を完了することができた．

OpenFMO はシンプルなプログラムであるため機能が限られており，計算化学的にも計算機科学的にも多くの改良点がある．Falanx ミドルウェアによる耐故障性の実装等もされているが，さらに広範な研究者による研究・開発が必要であろう．

謝辞 本研究の一部は JST-CREST 研究領域「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出」, 研究課題「ポストペタスケール時代に向けた演算加速機構・通信機構統合環境の研究開発」による．また HA-PACS クラスタの利用は筑波大学計算科学研究センターの「学際共同利用プログラム」によるものである．

参考文献

1) Kitaura, K., et al.: Pair interaction molecular orbital method: an approximate computational method for molecular interactions, Chem. Phys. Lett. 312, pp.319-324 (1999).
2) Umeda, H. and Sato, M.: Parallelization of GAMESS to the Large-Scale Computing, International Conference on Modeling and Simulation Technology 2011(JSST2011), OS2, pp.92-95 (2011).
3) 梅田宏明: GAMESS による大規模並列 FMO 計算に向けた取り組み，日本化学会情報化学部会誌 32, pp.2-5 (2014).
4) Ufimtsev, I.S. and Martinez, T.J.: Quantum Chemistry on Graphical Processing Units. 1. Strategies for Two-Electron Integral Evaluation, J. Chem. Theory Comput. 4, pp.222-231 (2008);
5) Yasuda, K.: Two-electron integral evaluation on the graphics processor unit, J. Comput. Chem. 29, pp.334-342 (2008).
6) Wilkinson, K.A., et al.: K.J., Acceleration of the GAMESS-UK electronic structure package on graphical processing units, J. Comput. Chem. 32, pp.2313-2318 (2011).
7) Asadchev, A. and Gordon, M.S.: New Multithreaded Hybrid CPU/GPU Approach to Hartree–Fock, J. Chem. Theory Comput. 8, pp.4166-4176 (2012).
8) Kussmann, J. and Ochsenfeld, C.: Pre-selective screening for

matrix elements in linear-scaling exact exchange calculations, J. Chem. Phys. 138, pp.134114 (2013).
9) Losilla, S.A., et al.: Construction of the Fock Matrix on a Grid-Based Molecular Orbital Basis Using GPGPUs, J. Chem. Theory Comput. 11, pp.2053-2062 (2015).
10) 梅田宏明, 塙敏博, 庄司光男, 朴泰祐, 稲富雄一: フラグメント分子軌道法に現れる Fock 行列計算の GPGPU 化, 情報処理学会論文誌コンピューティングシステム（ACS） 6, pp.26-37 (2013).
11) CUDA Toolkit, http://docs.nvidia.com/cuda/index.html
12) Maki, J. et al.: One-sided Communication Implementation in FMO Method, Proc. HPCAsia07, pp.137-142 (2007);
OpenFMO, http://www.openfmo.org/.
13) GAMESS, http://www.msg.chem.iastate.edu/gamess/.
14) ABINIT-MP, http://moldb.nihs.go.jp/abinitmp/.
15) PAICS, http://www.paics.net/.
16) Mochizuki, Y., et al.: Large-scale FMO-MP3 calculations on the surface proteins of influenza virus, hemagglutinin (HA) and neuraminidase (NA), Chem. Phys. Lett., 2010. 493, pp.346-352 (2010).
17) Falanx, https://sites.google.com/site/spfalanx/.
18) HA-PACS ベースクラスタ,
http://www.ccs.tsukuba.ac.jp/research/research_promotion/project/ha-pacs/cluster
19) Sawada, T., et al.: Binding of Influenza A Virus Hemagglutinin to the Sialoside Receptor Is Not Controlled by the Homotropic Allosteric Effect, J. Phys. Chem. B 114, pp.15700-15705 (2010).
20) Furukawa, Y., Koga, R., and Yasuda, K.: Acceleration of computational quantum chemistry by heterogeneous computer architectures, International Conference on Modeling and Simulation Technology 2011(JSST2011), OS2, pp.85-91 (2011).
21) Alexeev, Y., et al.: Heuristic static load-balancing algorithm applied to the fragment molecular orbital method, Proc. Supercomputing 2012, IEEE Computer Society, (2012).
22) 稲富雄一, 井上弘士, 眞木淳, 本田宏明,青柳睦: 並列フラグメント分子軌道法プログラム OpenFMO の超並列化, 日本化学会情報化学部会誌, 32, p. 6-12 (2014).