4P119

GPGPU 化 Fock 行列計算ルーチンの OpenFMO への組み込み

(筑波大学*, 東京大学**) ○梅田宏明*、塙敏博**、庄司光男*、朴泰祐*、重田育照*

Porting GPGPU-enabled Fock matrix preparation routine into OpenFMO

(Univ. Tsukuba*, Univ. Tokyo**) ○H. Umeda*, T. Hanawa**, M. Shoji*, T. Boku*, Y. Shigeta*

序

近年の大規模高性能並列計算機の発展において、アクセラレータを用いた高性能科学技術計算は大きなトピックとなっている。量子化学計算においても例外ではなく、最新の計算機による大規模高性能計算を実現するためにはアクセラレータへの対応が不可欠である。これまで我々は Hartree-Fock(HF)計算のホットスポットである Fock 行列計算について、アクセラレータの一つである GPU への実装を試みてきた[1]。この成果を大規模分子軌道計算に応用するために、フラグメント分子軌道(FMO)計算[2]プログラムの大規模計算機向け実装である OpenFMO [3]に我々の GPGPU 化 Fock 行列計算コードを導入した。本発表では、GPGPU 化手法の説明とこれを用いた FMO 計算についての性能評価を報告する。

実装

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て提供されているものである。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、これにより我々の実装を 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[1]。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[4]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)及びクランビンタンパク(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、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。

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)

	$(Gly)_{10}$			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

謝辞: 本研究で使用した HF 計算コードは九州大学の稲富らによる OpenFMO プログラムの一部を抜粋して提供していただいている．また本研究の一部は文部科学省特別経費「エクサスケール計算技術開拓による先端学際計算科学教育研究拠点の充実」事業，および 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．

参考文献

[1] 梅田宏明, 塙敏博, 庄司光男, 朴泰祐, 稲富雄一, 情報処理学会論文誌コンピューティングシステム(ACS), **6**, 26(2013).
[2] K. Kitaura et al., Chem. Phys. Lett., **312**, 319(1999).
[3] OpenFMO; http://www.openfmo.org/
[4] HA-PACS ベースクラスタ;
http://www.ccs.tsukuba.ac.jp/research/research_promotion/project/ha-pacs/cluster