1P-110　メニーコアプロセッサによる量子化学とその応用

（名大エコトピア[1]，㈱クロスアビリティ[2]）安田 耕二[1]，古川 祐貴[2]，古賀 良太[2]

Quantum Chemistry on Many-core Processors

（Nagoua Univ.[1]，X-Ability Co.Ltd.[2]）Koji Yasuda[1]，Yuki Furukawa[2]，Ryota Koga[2]

【序】微細加工技術に伴って計算機は高速化してきたが、消費電力が問題になりつつある。現在では 1 語をメモリーに転送すると、数百演算分の電力を消費する。つまり電力面からはメモリーを読み書きしないアルゴリズムが有利になる。その結果量子化学計算では、一個の二電子積分はおよそ数十演算で計算できるので、direct SCF が電力面から望ましくなる。更に積分計算アルゴリズムは、計算量よりメモリーの読み書き回数で優劣が決まる。また計算機も、単純なコアを多数集積したメニーコア型が、従来型より一桁電力効率が良いため、主流になると思われる。我々はメニーコア型計算機(GPU)での量子化学計算法を研究してきたが、今回はメモリー読み書きを減らすアルゴリズムと、d 基底関数への実装を報告する。

【方法】漸化式による二電子積分計算では、多数の中間積分ができる。これがレジスタ上に保持できないと、キャッシュやメモリーに待避される。このレジスタ溢れを押える事が、速度向上と電力削減に重要である。メモリー操作回数で漸化式の優劣を測る指標として、MOpコストが知られている。これは漸化式の右辺に現れる異なる中間積分の総数を示す。MOpコストは、レジスタ上に必要な中間積分を全て保存できる場合、レジスタ溢れや電力消費の良い指標になる。が現在のCPUはもっと少ないレジスタしか持たない。そこで我々は、レジスタ溢れを押える積分計算法について検討した。具体的にはMcMurchie- Davidsonの一中心積分$[r]^{(m)}$から、静電ポテンシャル行列要素をdirect法で計算する際、①どの順序で積分を計算するか ②どう並列化するか、について検討した。

① 一中心積分$[r]^{(m)}$をBoys関数$[0]^{(m)}$に還元するには、最小の非ゼロ指数を持つ軸（x, y, zどれか）方向の漸化式を使えば良い。ある中間積分は、漸化式の右辺に複数回現れる。計算した中間積分はレジスタから溢れる前に、早く使い切るのが望ましい。そこでどの順序で中間積分を計算すると、中間積分を保持するレジスタ数を最小化できるか調べた。それには中間積分のデータ依存関係を木構造に表し、どの部分木から評価するのが良いか考えればよい。これは、部分的にはコンパイラが最適化時にやるのだが、この最適化はNP困難な問題で解けない。我々はこの最適解を総当りで調べた。

② 二電子積分は普通 shell 四つ組単位で計算される。そこで複数の shell 四つ組を同時に計算して、並列化やベクトル化を行う。この時静電ポテンシャル行列要素をどこに保持するかも問題である。d 関数では 35 個の角運動量成分があるが、レジスタに置くには数が多く、メモリーに置くと読み書きが増える。一中心積分計算は x, y, z 方向に対称的なので、我々は 3 個のスレッドで 1 つの shell 四つ組の計算をし、必要なレジスタ数を減らす方法を検討した。また静電ポテンシャル行列要素も 3 個のスレッドのレジスタに分散して割り付けた。必要な積分を他のスレッドが持つ場合、再計算した。

【結果】中間積分を保存するのに必要な最小レジスタ数は、中間積分の総数よりかなり小さく、例えばL=4, 8 でそれぞれ 6, 12 だった。適切な計算順序ではレジスタ数は少なくてすむ可能性を示

す。次にmが小さい順に$[r]^{(m)}$を計算したソースコードと、最適順序で計算したソースコードをnvccでコンパイルし、使用レジスタ数を調べた。その結果角運動量Lが小さいなら差はなく、Lが大きいと差があった。このレジスタ数には、静電ポテンシャル行列要素やループカウンターなども含んでいる。また 3 スレッドでshell四つ組を計算する場合、Lが大きくてもレジスタ数はかなり少なくなった。また、これらのソースコードで 1024 個のshell pair間の二電子積分から静電ポテンシャル行列要素をGPUを用いて計算した時の性能を測定した。使用レジスタ数 63 以下では、理論性能の約 5 割を示し、それ以上では約 2 割に性能は減少した。

次にこの積分計算法を Gaussian と GAMESS-FMO に組み込んだ。

Gaussian : Linda で並列化した上で、各 node の計算は OpenMP と GPU で高速化した。次の問題があった。(1) GPUで各nodeの計算速度を 10 倍程度加速した結果、Lindaの通信時間が無視できなくなった。Gaussian09 では、二電子積分計算時に、全てのサブルーチン引数を通信で送っている。そこでできるだけ通信をやめ、各nodeで引数を再計算した。(2) Gaussian09 の高速多重極計算ルーチンには、競合と思われるバグがあり、OpenMP並列で大きな計算をするとたまに間違える。(3) GPU計算がまれにunspecified launch failure（原因不明のでエラー）で終了する事があった。この再現困難なエラーはGPUのスタック領域不足が原因のようだ。(4) Second order SCF (Newton法)をGPUで加速する場合、連立方程式の反復解法が数値誤差の影響を受け易い事が分かった。そこでDIIS法が失敗する場合は、scaled steepest decent法を用いた。水和白金クラスター[$Pt_{13}(H_2O)_{15}$]、基底関数はD95 とStuttgart/Dresden ECPを用い（801 AOs）、2 node×6 threadsで計算した時間を表 1 に示す。全体で 9 倍程度加速された事、Linda並列化されていない高速多重極計算や、通信時間が無視できなくなっている事が分かる。

表 1 ：SCF10 反復にかかった時間（秒）

	original	+GPU
高速多重極	21.8	93.6
二電子積分(CPU)	2440.2	28.8
二電子積分(GPU)	-	55.4
交換相関項(CPU)	211.5	1.2
交換相関項(GPU)	-	80.8
通信	50.7	27.8
計 (SCF10 反復)	2744.4	313.9

GAMESS-FMO：フラグメント分子軌道法(FMO 法)の環境静電ポテンシャル(ESP)の計算の加速を行った。モデル分子はヒトインシュリン(PDBID:2HIU)、CPU は Intel Core i7-3930K 3.2GHz (6 core)、GPU は NVIDIA GeForce GTX680, コンパイラには Intel FORTRAN/C++ Compiler と CUDA4.0 を用いた。この計算には、クロスアビリティが販売している GPU 計算エンジンモジュール XA-CUDA-QM [1] を用いた。

表 2 : ESP 部分と計算全体の加速率（GAMESS を 1 とする）

	ESP	全体
6-31G, FMO2, 一点計算	14	4.1
6-31G*, FMO2, 一点計算	17	4.4
6-31G, FMO3, 一点計算	20	3.7
6-31G, FMO2, 勾配計算	11	2.6

表 2 に種々の条件での計算全体とESP部分のみの加速率をまとめた。どの場合も全体でGAMESSを 2.6～4.4 倍加速できた。また、エネルギー値の差は 10^{-6} a.u.以下であり、十分小さかった。また同様の計算をGeForce GTX580 GPUで行ったが、加速率はほぼ同じだった。GTX680 の理論性能がGTX580 の 2 倍である事を考えると、改善の余地がまだある事が分かる。詳しい解析は当日報告する。

[1] Furukawa, Y., Koga, R., Yasuda, K. JSST 2011 Tokyo. (2011)