Vol.2010-HPC-127 No.5
2010/10/13

GPUを用いた分子動力学法の高速化と省電力化

宇田川　拓郎[†1]　関嶋　政和[†2,†3]

スーパーコンピュータや大規模クラウドにおける消費電力は年々増加しており、エネルギー効率の高いスーパーコンピュータや、このようなコンピュータを活用するアプリケーションプログラムの開発が求められている。近年、画像処理に用いられて来た GPU の高い演算性能を科学技術計算に用いる事で、消費エネルギーを削減する試みが為されている。分子動力学シミュレーションは、多原子系における原子の運動を、原子間の相互作用を計算しながら個々の原子に対する Newton の運動方程式を積分することにより求める方法であり、大規模で複雑な計算が必要となる。本研究では、分子動力学法のシミュレーションプログラムを GPU 向けに最適化し、CPU における実行と計算速度、消費電力、消費エネルギーに関して比較を行った。

Energy Consumption and Acceleration of GPU of Molecular Dynamics Simulation

TAKURO UDAGAWA[†1] and MASAKAZU SEKIJIMA[†2,†3]

Molecular dynamics simulations are widely used for simulating the motion of molecules in order to gain a deeper understanding of chemical reactions, fluid flow, phase transitions, and other physical phenomena due to molecular interactions. However, these simulations require huge computer resources. In addition, the problem of energy consumption must be solved. Recently, GPGPU has attracted attention as a possible solution to these problems. In this paper, we performed molecular dynamics simulations on a CPU and GPU, and compare calculation time, power consumption and energy consumption results between them.

†1 東京工業大学 工学部 情報工学科
Department of Computer Science, Tokyo Institute of Technology
†2 東京工業大学 学術国際情報センター
Global Scientific Information and Computing Center, Tokyo Institute of Technology
†3 東京工業大学 情報理工学研究科 計算工学専攻
Department of Computer Science, Tokyo Institute of Technology

1. はじめに

分子動力学法 (Molecular Dynamics; MD) は、多原子系における原子の運動を、原子間の相互作用を計算しながら、個々の原子に対する Newton の運動方程式を積分することにより求める方法である。このような積分は解析的に計算することが不可能であるため，有限差分法を用いて数値的に計算する。現在、分子動力学シミュレーションは様々な研究分野において広く用いられている。バイオインフォマティクスの分野を例にとってみると、タンパク質やペプチド、ＤＮＡのような生体分子のダイナミクス、構造最適化、熱力学的な解析、機能予測などに用いられている[1)2)]。

MD は、例えば生体分子においては数千から数万原子について femto 秒単位の時間刻みを積み重ねて nano 秒から micro 秒までの計算を繰り返し、挙動を追跡しなければならない。そのため、生体分子の MD の実行には、スーパーコンピュータ等を用いて数日から数ヶ月の日数が必要とされてきた。我々は超並列計算機やスーパーコンピュータを用いてこれまでに大規模な MD を実行してきたが[4)3)]、一般的には特定の機関を除いてそのような計算資源を十分に用意することが容易ではない。

近年、GPU(Graphics processing units) の利用が、この問題の解決策の一つとして挙げられる。GPU は本来画像処理を行うために開発されてきたデバイスであるが、コンピュータグラフィックス技術の発展などに伴い、強力な計算能力をもつようになった。この GPU の強力な計算力に着目し、GPGPU(general purpose computation on graphics processing units) と呼ばれる汎用計算に GPU を利用する技術は、流体計算[6)] や、データーベース操作[7)]、天気予報[8)] など、現在幅広い分野において応用されるようになってきている。

GPU はスーパーコンピュータにも用いられ始めており、中国のスーパーコンピュータ Nebulae は NVIDIA 社の GPU、Tesla C2050 を使い、2010 年 6 月に発表された TOP500 で 2 位となった[10)]。さらに、2010 年 11 月に稼働する予定の東京工業大学のスーパーコンピュータ TSUBAME2.0 は NVIDIA 社の Tesla M2050 を使用している。

一方で、GPU はその強力な計算力だけでなく、環境負荷の小さいデバイスとしても期待されている。スーパーコンピュータ・クラウドを始め、計算機における電力消費は年々増加しており、電力や排出する二酸化炭素の制限によって計算が制限されてしまう可能性も危惧されている。環境に配慮したグリーンコンピューティングは今後 IT 分野で重要なテーマになると考えられる[5)]。

本論文では、分子動力学法のシミュレーションプログラムを GPU 向けに最適化し、ア

ルゴン原子の振る舞いについて CPU における実行と計算時間、消費電力、消費エネルギーに関して CPU との比較を行った。

2. 分子動力学法

分子動力学シミュレーションは femto 秒のような非常に短い時間幅のステップの繰り返しで行われる。それぞれのステップは、原子の受ける力の計算と原子座標の更新という 2 つの部分に大きく分けられる。以下で今回用いた分子動力学シミュレーションの流れを示す。

(1) 原子のデータのロード
(2) 他の原子から受ける力の計算
(3) この原子の、次の座標と速度の計算
(4) 1-3 を全ての原子反復
(5) 全ての原子の座標の更新
(6) 系のエネルギー、運動エネルギー、温度の計算
(7) 1-6 を規定時間まで反復

原子間のエネルギーの計算には式 (1) のレナード・ジョーンズポテンシャルを用いた。

$$\varphi(r_{ij}) = \sum_{ij} 4\varepsilon \left[\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^{6} \right] \tag{1}$$

ただし、r_{ij} は原子 i と原子 j の距離、ε は結合の強さ、σ は原子半径を表す。次のステップでの原子の座標と速度を得る計算には Verlet 法を使った。Verlet 法は分子の現在の情報と、一つ前のステップの情報から、原子の速度と次のステップにおける座標を計算する。その式は (2)(3) で示される。

$$r_i(t_+) = 2r_i(t) - r_i(t_-) + a_i(t)(\Delta t)^2 \tag{2}$$

$$v_i(t) = \frac{1}{2\Delta t}[r_i(t_+) - r_i(t_-)] \tag{3}$$

相互作用によるポテンシャルエネルギーは式 (4)、運動エネルギーは式 (5)、温度は式 (6) で求めた。

$$\Phi = \sum_{i<j} \sum \varphi(r_{ij}) \tag{4}$$

$$K = \sum \frac{1}{2} m_i v_i^2 \tag{5}$$

$$T = \frac{2K}{3k_B N} \tag{6}$$

m_i は原子 i の質量、v_i は原子 i の速度、k_B はボルツマン定数、N は原子数を表す。本研究におけるシミュレーションは周期境界条件を用いて行った。基本セルに原子を配置し、その周囲を基本セルと同じ原子配置をもった仮想的なセルが囲っていると仮定する。基本セルから粒子が飛び出した場合は、中心について点対称の位置から同じ速度を持って新たにセルに飛び込んでくる粒子を作る。このようにしてセル内の原子数を一定に保ち、境界面の影響を緩和し、精度の高いシミュレーションを実行している。

3. GPU について

3.1 NVIDIA CUDA

GPU 計算を行うためには Brook+[9],ATI Stream[11] などの方法があるが、本研究では C 言語の拡張である、NVIDIA の CUDA を用いてプログラムを実装した。

CUDA をサポートする NVIDIA の GPU は G80、GT200、Fermi の 3 つの世代がこれまでに発表されている。いずれの世代でも、NVIDIA の GPU は SM(Streaming Multiprocessor) を一つの演算ユニットとして構成されている。SM は CUDA コア、レジスタ、シェアードメモリなどから構成され、SM の構成や GPU がもつ SM の数は世代や GPU の種類により異なっている。例えば、最新世代の Fermi アーキテクチャの GTX480 は、15 個の SM をもち、SM あたり 32 個の CUDA コアを搭載し、合計で 480 個の CUDA コアをもつ。

CUDA のソースコードはホストコードとデバイスコードの二つの部分に分けられる。ホストコードには CPU 側での処理が記述される。従来の C 言語のプログラミングに加え、GPU 上で実行される関数の呼び出しや、GPU のメモリの確保、GPU と CPU 間のデータ転送などを記述する。デバイスコードには GPU 上で実行される処理が記述される。CPU から呼び出され GPU 上で計算を行うカーネル関数、カーネル関数から呼び出され GPU 上で計算を行うデバイス関数などがそれにあたる。

CUDA では計算スレッドが階層的に管理されている (図 1)。それぞれのスレッドが論理的にブロックにまとめられ、ブロックはグリッドにまとめられる。スレッドの実行は CUDA コアなどの実行ユニットが担い、スレッドブロックの実行は SM が担う。同一ブロックに所属するスレッドは一つの SM で計算され、32 個のスレッドがウォープという単位を作り同じ命令を実行する。同じブロックに所属するスレッドはスレッド間で同期をとることや、シェアードメモリを用いた通信を行うことができる[14]。

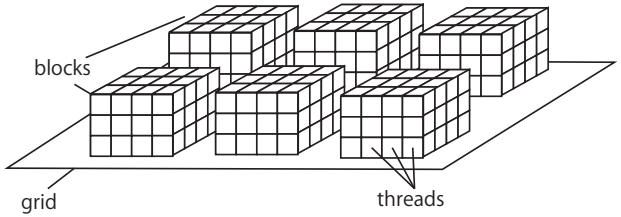

図 1 CUDA のスレッドの階層構造の模式図

3.2 GPU 利用によるエネルギー消費

GPU を使う計算を行うためには、CPU と GPU の両方を稼働させる必要がある。そのため電源からの供給電力という観点では GPU を計算に用いた場合の方が不利となる場合がある。しかし、電力効率や計算に必要なエネルギーを比較した場合では GPU の方が有利となる結果が報告されている。例えば Mahsan Rofouei らは separate convolutions の計算を LEAP-Server を用いて計算を行った。この研究で彼らは GPU を、GPU に適した並列アプリケーションに対して用いることは計算速度やエネルギー消費の観点から有利であることを示した[12]。

さらに GPU の電力効率についてはスーパーコンピュータに対する利用においても見ることができる。1 章でも述べたように中国のスーパーコンピュータ Nebulae は TOP500 において 2 位を獲得した。それだけでなく Nebulae は 2010 年 6 月に発行された Green500 では 4 位になった[13]。Green500 は MFLOPS/watt によってスーパーコンピュータの電力効率性能を評価したランキングである。一方で TOP500 で 1 位の Jugar は Green500 では 56 位となっている。

4. CUDA による分子動力学法の実装

我々が CUDA を用いて実装した分子動力学法の流れは以下のとおりである。

(1) GPU のメモリを確保
(2) 原子のデータを CPU から GPU へ転送
(3) カーネル関数 1 を実行
(4) カーネル関数 2 を実行
(5) 計算結果を GPU から CPU へ転送
(6) 2 から 5 を繰り返す

3 章で記述したとおり GPU のメモリを最初に確保しなければならない。次にアルゴン原子の初期座標のデータを GPU に転送する。そして一つ目のカーネル関数を実行する。このカーネル関数では、原子数と同じ数のスレッドを立てる。各スレッドがそれぞれ一つの原子のデータをロードし、他の原子から受ける作用を計算し、次のステップでの座標と、現在の速度、ポテンシャルエネルギー、運動エネルギーを求める。一つ目のカーネル関数の実行が終わると、二つ目のカーネル関数を実行する。ここでは、カーネル関数 1 で求めた結果を元に、原子のもつエネルギーの総計を計算する (図 2)。総和を求めるには全てのスレッドでカーネル関数 1 の実行が終わっていなければならないため、この計算はカーネル関数 2 として独立させた。関数の実行が終わったら計算結果を CPU へと送る。本研究では、この計算を 100 ステップ実行した。

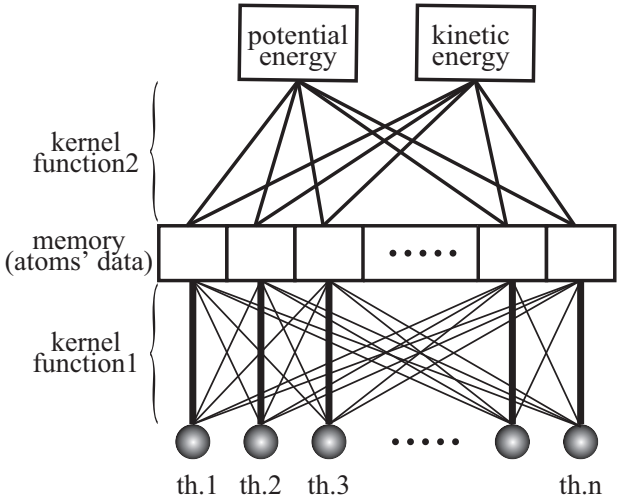

図 2 カーネル関数の実行の模式図

5. 実　　験

実験を行った環境について記述する。CPU は Intel の Core i7-860 を使用した。これは動作周波数 2.80GHz の CPU コアを 4 つ持ち、256KB の L2 キャッシュ、8MB の L3 キャッシュを搭載している。メモリは 6 GB の DDR3 の SDRAM メモリ。GPU は NVIDIA の GT240 を用いた。GT240 は 96 個の CUDA コアを持つ。OS には Ubuntu9.04、CUDA ツールキットはバージョン 3.0 を使った。

実験では MD のプログラムを CPU のみの場合と、GPU も用いた場合で実行し、計算に要した時間と消費したエネルギーを求めた。計算時間は相互作用の計算の開始直前から、最後のステップの結果の出力直後までを計測した。電源からの電力供給を測るのには SYSTEM ARTWARE 社の WATT-HOUR METER SHW3A を使用した。それぞれの場合でシミュレーションに用いた原子数を 8、64、128、512、1024、4096、8192、32768、65536 として実験値を計測した。なお、シミュレーションは NEV アンサンブルで行った。

6. 結果と考察

実験で得られた結果を表 1, 図 3、図 4 に示す。シミュレーションで用いる原子が非常に少ないときは CPU のみの場合の方が計算速度が速いことが分かる。原子数が 8 の時、CPU のみの計算時間は 0.003 秒、GPU を使用した場合の計算時間は 0.0089 秒であった。この原因は原子が少なすぎて並列化の恩恵を十分に得られず、却って CPU-GPU 間の転送などのために計算時間がかかってしまったものと考えられる。シミュレーションに用いる原子数が増加するに連れて GPU を用いた方が計算速度が速くなる。原子数が 64 個の時、GPU の実行速度が CPU の実行速度を逆転し、それ以降は一方的に差が広がっていく。MD のシミュレーションでは通常数千から数万程度の原子を用いるため、GPU を用いることで大幅に計算時間を減らすことができる。

表 1 results

atoms	value	CPU	GPU
8	sec	0.0030	0.0089
	watt	92.5	114
	joule	0.2775	1.015
64	sec	0.160	0.038121
	watt	94.3	115
	joule	15.088	4.384
512	sec	8.280	0.2941
	watt	94.8	121
	joule	784.944	35.59
65536	sec	106508	1554.98
	watt	98.5	136
	joule	10491038	211477

電力消費に目を向けてみると、供給電力の大きさという観点では GPU の方が常に CPU よりも不利な結果となった。しかし、計算時間と供給電力の積からある計算にかかったエネ

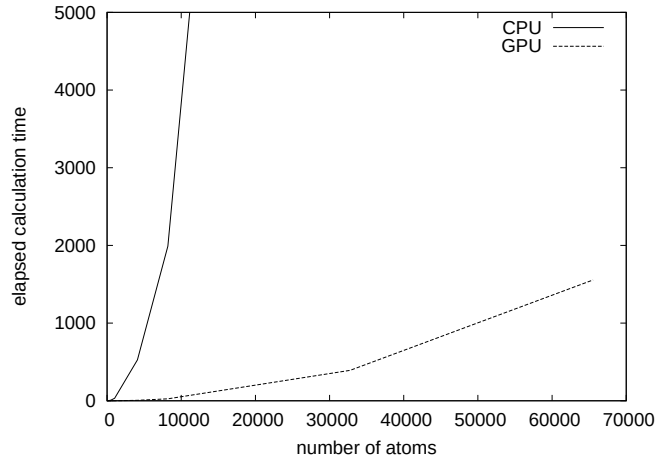

図 3 原子数と計算時間のプロット

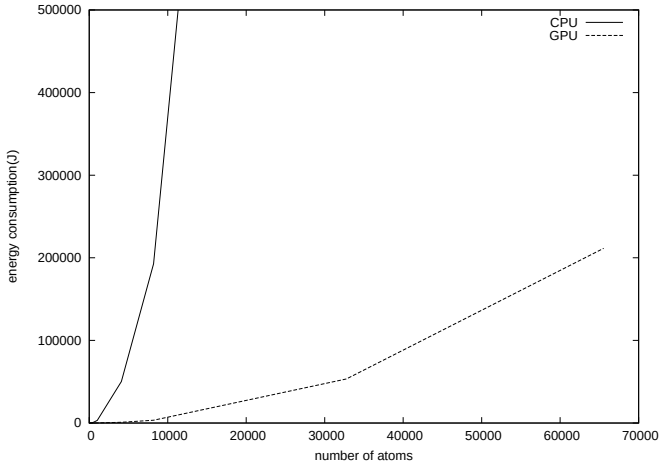

図 4 原子数と計算で消費したエネルギーのプロット

ルギーを求めると、GPU を利用した場合の方が圧倒的に小さくなる。これは GPU 利用による計算時間の短縮が、供給電力の増加をはるかに上回る影響をもつ結果を示したからである。このことから GPU の利用は従来の行われてきた計算をより少ないエネルギーで実行することができるといえる。

7. ま と め

CPU のみの場合と GPU を用い場合の二つの場合について MD のプログラムを実行し、得られた結果を比較した。GPU を用いた場合の方が供給される電力は大きくなる。しかし、並列化が十分になされると供給電力の増加の影響を大きく上回って計算時間が短縮されるため、総消費エネルギーは GPU を用い場合の方が小さくなる。このことから GPU の利用は既存のプログラムを短時間で実行できるだけでなくグリーンコンピューティングにも大きく貢献できる可能性があると結論づけられる。

謝辞 本原稿を執筆するにあたり、GPU 計算におけるエネルギー計測について助言をくださった東京工業大学 遠藤敏夫特任准教授、丸山直也助教に感謝致します。

参 考 文 献

1) M. Sekijima, C. Motono, S. Yamasaki, K. Kaneko, and Y. Akiyama, ”Molecular dynamics simulation of dimeric and monomeric forms of human prion protein: Insight into dynamics and properties”Biophysical Journal ,85, pp.1176-1185, 2003.
2) K. Sugawara, S. Saito, M. Sekijima, K. Ohno,Y. Tajima, M.A. Kroos, A.J. Reuser, H. Sakuraba, ”Structural modeling of mutant alpha-glucosidases resulting in a processing/transport defect in Pompe disease”, J Hum Genet, 54, pp. 324-330, 2009.
3) M. Sekijima, J. Doi, T. Noguchi, and S. Shimizu,”Optimization and Evaluation of Parallel Molecular Dynamics Simulation on Blue Gene/L”, In Proceedingsof the IASTED International Conference on Parallel and Distributed Computer and Networks (PDCN2007), pp.257-262, 2007.
4) M. Sekijima, S. Takasaki, S. Nakamura and K Shimizu, ”Automatic Improvement of Scheduling Policies in Parsley Parallel Programming Environment” In Proceedings of the 14th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2002) , pp.380-385, Cambridge, Massachusetts, USA, November 2002.
5) Michael Feldman, ”The Greening of HPC”, http://www.hpcwire.com/features/The_Greening_of_HPC.html
6) W.Y. Liang, T.J. Hsieh, M.T. Satria, Y.L. Chang, J.P. Fang, C.C. Chen, and C.C. Han, ”A GPU-Based Simulation of Tsunami Propagation and Inundation” Algorithms and Architectures for Parallel Processing, pp.593-603, 5574, 2009.
7) Peter Bakkum et al, ”Accelerating SQL database operations on a GPU with CUDA” ACM International Conference Proceeding Series; Vol. 425, Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units, pp.94-103
8) J. Michalakes and M.Vachharajani, ”GPU Acceleration of Numerical Weather Prediction”, Parallel Processing Letters, 18, pp. pp.531–548, 2008.
9) AMD Brook+ Presentation. In SC07 BOF Session, 2007.
10) http://www.top500.org/
11) Advanced Micro Devices, Inc., ”GPU and CPU Technology for Accelerated Computing”, http://www.amd.com/US/PRODUCTS/TECHNOLOGIES/STREAM-TECHNOLOGY/Pages/stream-technology.aspx
12) Mahsan Rofouei, Thanos Stathopoulos, Sebi Ryffel, William Kaise, and Majid Sarrafzadeh, ”Energy-Aware High Performance Computing with Graphic Processing Units”, In Proceedings of the Workshop on Power Aware Computing and Systems 2008 (HotPower’08), 2008.
13) http://www.green500.org/
14) NVIDIA CUDA C Programming Guide Version 3.1.1 http://developer.nvidia.com/object/cuda.html