平成 25 年度 産業利用トライアルユース：先端研究基盤共用・プラットフォーム形成事業
『みんなのスパコン』TSUBAME による日本再生
成果報告書

利用課題名 メソ構造を持つ高分子材料のマルチスケール・シミュレーション
Multi-scale simulation of nano-structured polymeric material

本田 隆
Takashi Honda

日本ゼオン基盤技術研究所
Foundation Technology Laboratory, R & D Center, ZEON CORPORATION
http://www.zeon.co.jp

TSUBAME2.0 のトライアルユースにより、2002 年にリリースされた高分子の SCF 法を用いる OCTA/SUSHI コードを MPI・GPU を用いて並列化することができた。この高速化により、今まで不可能であった、大規模な系の高分子の相分離構造のシミュレーションが可能となった。具体的なベンチマークにおいては、1core＋1GPU 計算において、96^3 メッシュであれば、1core に比べ約 20 倍の高速化を達成できた。また、MPI・GPU 計算においては、$256^2 \times 384$ メッシュの系において、92 コアを用いたフラット MPI 計算より約 3 倍の高速化を達成できた。この課題の成果は今後の高分子材料の研究に大きく寄与すると考える。

A trial use of TSUBME2.0 made the OCTA/SUSHI code, which is one of the SCF codes using SCF method of polymer, enable to invoke GPU and MPI・GPU parallel calculation. This parallelization enables us to study large scale phase separated structures of polymers. The bench mark of the SUHI showed the ability that about 20 times faster with 1 core + 1 GPU than 1 core in 96^3 mesh system, and about 3 times faster with 92 MPI・GPU parallelization than 92 flat MPI parallelization in such $256^2 \times 384$ mesh large system. This result will give many benefits for the research of polymers.

Keywords: SCF method of polymer/Density functional theory/diblock copolymer/
phase separation/micro-phase separation

背景と目的

高分子材料は異種のものを混合しても、高分子ゆえ混合におけるエントロピーの効果が小さくなり、一般的に相分離し多様な相分離構造を形成する。よって、異種のポリマー同士を化学的に結合したブロック・ポリマー状の高分子材料においても、その相分離の性質は変わらず、自己粗視化を伴うミクロ相分離をする。よって、高性能なポリマーアロイやブロック・ポリマーの自己組織化構造を生かした高機能高分子材料の開発において、相分離構造の制御は極めて重要な技術課題となる。

このような高分子の粗視化構造をシミュレートする手法としては、Phase Field 法の一つとみなせる高分子の SCF 法（以降では簡単に SCF 法と記述する）が精度が高く優れた方法である。この SCF 法を利用したプログラムの一つとして、名古屋大学で行われた OCTA プロジェクト（Open Computational Tool for Advanced material technology：1998~2002）で開発された SUSHI（Simulation Utilities for Soft and Hard Interfaces）が存在する。SUSHI はこのプロジェクトの成果である OCTA システムの一部として、フリーなプログラムとして 2002 年に世界に向けてリリースされた[1]。しかし、高分子の SCF 法は計算負荷が高く、リリース後、高速化が望まれていた。

本プロジェクトでは、GPU 用を利用して SUSHI を高速化し、さらに MPI 並列も行えるように改良した。この改良により、1コアの計算にくらべ、GPU を利用した計算では約 20 倍の計算速度を達成し、さらに MPI・GPU 並列により、1 コアでは不可能であった大規模な系を計算できるようになった。大規模計算が可能になったことにより、これまで不可能であったブロック・ポリマーのミクロ相分離構造の安定な相分離構造の探索を行えるようになった。

概要

SCF 法は、粗視化した高分子を Gauss 統計を満たす鎖とみなし、鎖を構成するセグメント（高分子を粗視化した場合の粗視化単位であり、モノマーが重合した場合の化学的な繰り返し単位をいくつか集合させたものである）間に働く相互作用と高分子溶融体における非圧縮条件の下、相分離構造をシミュレートできる。SCF 計算におけるスキームの概念図を図 1 に示す。Gauss 鎖の両端（青と黄色のセグメント）を、空間のある点に固定すると、鎖の途中のセグメント（赤）の空間的な存在確率は、両端からの該当のセグメントまでの酔歩により分布する該当セグメントの空間的な統計的な重みの積に比例する。このようなセグメントの空間的な統計的な重みは経路積分と呼ばれ、言い換えれば、該当セグメントまでの鎖の長さに比例する時間の間、末端位置から拡散した粒子の存在確率（図中の円に相当する）と同等なものである。以降ではセグメントの空間的な密度は $\phi(\mathbf{r})$で表すこととする。

求めた該当セグメントの空間分布 $\phi(\mathbf{r})$が、セグメント間の相互作用と非圧縮条件を同時に満たすよう求まればよいが、セグメント間相互作用はセグメントの存在分布が判明した後得られるものであるから、この計算は非線形となり、繰り返し計算による自己無撞着場（Self-Consistent Field）を求めることにより達成される。よって SCF 法と呼ばれる。詳細は参考文献[2]をご覧頂きたい。

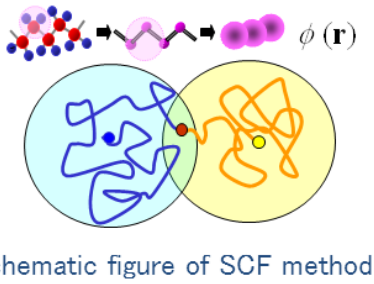

図 1 SCF 法の概念図：高分子鎖を粗視化してセグメントの繋がりとみなした後、セグメントの空間的な存在分布 $\phi(\mathbf{r})$を求める。

実際の計算では経路積分は有限差分法（FDM）を利用して計算され、その具体的なスキームは次のようになる。

$$q(s+ds,\mathbf{r}) = \exp\left[-\beta\frac{V(\mathbf{r})ds}{2}\right]\left(1+\frac{b^2}{6}\nabla^2 ds\right)\left\{\exp\left[-\beta\frac{V(\mathbf{r})ds}{2}\right]q(s,\mathbf{r})\right\} \quad (1)$$

ここで、q は経路積分であり一種の場である、s は鎖末端からの鎖の長さ、$\mathbf{r}$ は位置ベクトル、$V(\mathbf{r})$は外場（最終的に求められる自己無撞着場）、b はセグメントの有効長、$\beta=1/k_BT$ である。よって、∇^2 や exp の計算が多用される計算となり、重い計算を解くものではない。が、SCF 法により Phase Field 法で必要となるセグメントの化学ポテンシャルを求めるので、多くの繰り返し計算を必要とする。

SUSHI は SCF 法を C++にて実装したプログラムであり、実際に SUSHI の計算プロファイルを取ったところ、多用される関数は ∇^2 や、exp であることがわかった。本課題における主な成果はこれらの演算を GPU を利用して高速に行えるようにしたことにある。さらには、MPI 計算も可能とし、最終的には MPI･GPU による大規模計算を達成した。詳細を次に述べる。

1. 開発環境

開発に利用したプラットフォームは TESLA2070,2050,K20 を挿した OS が Windows7 の 64bit CPU の PC である。Visual Studio 2010 を開発環境として用いた。計算のテストや、大規模計算は TSUBAME2.0 を利用した。

2. SUSHI における場の演算の GPU への対応

SUSHI において、場は一次元ベクトル、場の演算子である ∇ や ∇^2 は行列として実装されており、計算の高速化のためには一次元ベクトル演算の関数や行列とベクトル積の高速化が主用なターゲットになる。SUSHI は C++で実装されていたので、C 言語の派生とみなせる CUDA との馴染みは良く、CUDA 導入に大きな支障は発生しなかった。また、オブジェクト指向で開発が進んでいたので、CUDA で記述されたプログラムの導入は、クラスの関数の実装の変更や新たなクラスの追加等で済ませ、元となったコードに及ぼす影響をできるだ

け小さくすることができた。

GPU を利用した改良における設計指針としては、種々のトライをした後、ホスト・デバイス間通信が遅いことを認識した後、次のようなプログラム設計方法を用いた。

(a) ホスト・デバイス間通信の低減

場のデータはホスト側と GPU 側と両方で持つこととし、基本的に演算は GPU 側で行い、最終的な結果が得られた段階でホスト側に GPU 側のデータを転送し計算を終了する。つまり、場を取り扱う ScalarField クラス(実際の場を取り扱う SUSHI における C++のクラスである)においては、実際のデータは次のように実装することとした。

```
class ScalarField {
……
    double* p_value; // ホスト上に確保
    double* p_valueD;// デバイス上に確保
};
```

さらに、これまでのプログラムの実装を変えなくても済むように、ScalarField クラスの関数内で GPU を利用するようにし、露わな GPU の利用はプログラムの主要部分には表れないように隠蔽した。例えば、ScalarField クラスには種々のオペレーターを実装するが、"+="演算子のような場合、ScalarField オブジェクトである aField, bField を利用して次の演算を行える。

```
> aField += bField;
```

しかし、演算は p_valueD を用いて全て GPU 上で行なわれ、ホスト側の p_value は利用しないように演算子の実装を変更した。

さらに、場の最大、最小、和を求めるような演算には CUDA の reduction 計算を用いて対応した。

(b) ホスト・デバイス間通信

場のデータを通信するためには、ScalarField の関数として、次のようなものを用意した。

```
class ScalarField {
……
    void putToCUDA() // GPU へ送信
    void getFromCUDA() // GPU から受信
};
```

つまり、putToCUDA()は、初期条件設定等で変更されたホスト側データ p_value を GPU 側に送る場合に利用し、getFromCUDA()はその逆の場合利用する。例えば、デバイス側で行われた計算の結果を出力したい場合、次のようにすればよい。

```
> aField.getFromCUDA();
> cout << aField;
```

このように、ホスト・デバイス間通信の記述はできるだけ簡素にし、元のコードの変更を抑え、実装の効率化をはかった。

(c) 場の演算子の実装の GPU への対応

SUSHI では∇や∇^2の演算は FreePropagator というクラスが担当している。これらの演算は規則格子上のデータに係数を掛け和をとる演算となり、CUDA が得意とする演算となる。例えば、∇^2の 3 次元空間での演算は図 2 に示すようになる。まず、3 次元空間を CUDA のスレッドが担当する角柱状の領域に分割する。分割された領域では末端より∇^2の演算が開始されるが、3 面の GPU 上の Shared メモリーの値として確保され 1 回の演算が終了した後、不要となった 1 枚の面の更新を行い、他の面はポインタの移動のみで済ませ、計算の高速化をはかった[3]。

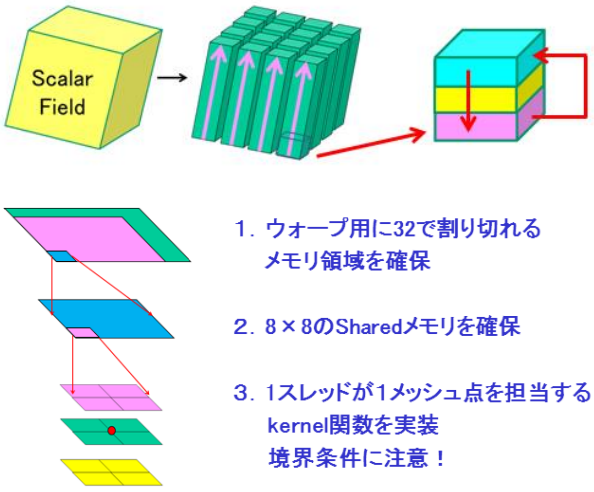

図 2 ∇^2の CUDA を利用した実装の概念図:上図は大規模な 3D 領域をスレッド用の小領域にする概念図。下図は、2D 平面をからスレッド用の小領域を取り出す概念図。

3. SUSHI における MPI 並列の実装

CUDA 実装と同時に MPI 並列による実装を行うと開発に混乱を来すので、CUDA を利用した 1GPU での並列化と並行して、フラット MPI のみによる並列化を行った。具体的には、図 3 に示すように規則格子の大規模な 3D の系を小領域の系に分割し、境界部分の糊代データを MPI 通信する FDM で利用される通常の方法を導入した。この実装においても、元のコードへの影響をできるだけ減らすため、次のようなプログラム設計方法を用いた。

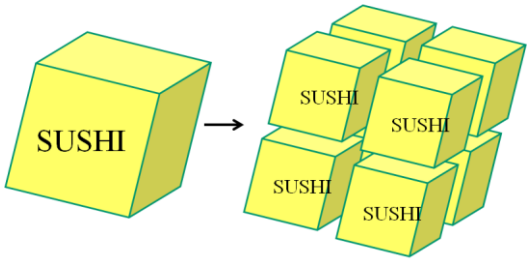

図 3 大規模領域を分割する MPI 並列の概念図

(a) MPI に関する演算の集中実装

MPIに関する実装を集中するため、MessagePassingInterfaceクラスを新たに設け、主要な通信に関する演算は全てこのクラス内のメソッドにより実行することとした。また、このクラスの派生としてMessagePassingInterfaceOfGlobalFieldクラスも実装し、通信の代表プロセスとなるrank0でのグローバル領域のマネージメントを行った。

(b) MPI 通信の隠蔽

CUDA の実装と同様、MPI 通信の実装もできるだけ隠蔽するように努め、元のコードの変更をできるだけ少なくする努力をした。基本的に小領域に分割した SUSHI プロセスで利用される ScalarField が隣接領域との間で糊代データの通信を行なう場合、communicate 関数を実装し、主要コード部分では、例えば,次のような一行で通信を終了するようにした。

```
> aField.communicate()
```

また、グローバル領域の送受信には次のような関数を用意し、簡易に通信が行えるようにした。

```
> vector< ScalarField* > phiG, phi;
> sendGlobalFields( phiG, phi );
> receiveGlobalFields( phiG, phi );
```

ここで、phiG はグローバル領域の ScalarField のポインタが収容された C++で利用される STL(Standard Template Library)の vector コンテナであり、rank0 以外のプロセスでは null が収容される。

C++を利用した並列化において、しばしば STL コンテナを利用したデータ構造の複雑化による問題が指摘されるが、大規模領域のコンテナとしてのみ利用すれば、メリットは大きいが、グスタフソンの法則からして並列化に対するデメリットにはならないと考える。

4. SUSHI における MPI・GPU 並列の実装

MPI 並列において、領域分割した場合の小領域の境界条件は、グローバル領域で設定した周期境界条件は利用できないので、Dirichlet 境界条件（意味的には壁の条件）にして対応した。よって、GPU の実装を MPI 下で利用しようとした場合、GPU の実装もやはり周期境界条件であったので変更する必要があった。そこで、FreePropagator の実装を変更することで対応した。

当初、TSUBAME2.0 で、MPI・GPU 並列を行う場合、全 GPU を利用するのに各ノードでの GPU の ID の設定を露わに行っていたため実行に失敗していたが、GPU の ID は自動認識である情報を得てコードを改良し、最終的には各ノードの GPU を全て利用する MPI・GPU 並列を行うことができた。この実装の成功により、一つの GPU では不可能であった大規模な系の計算も可能となった。

5. SUSHI の実行モジュールのバリエーション

表 1 に、Make のオプションで作り分けられるようになった 6 種類の SUSHI をまとめた。Make での有効なオプション（ON/OFF）は、(1)PTL: 以前に実装した Pthread ライブラリを用いた共有メモリー型並列、(2)GPU:CUDA を利用した 1 コア GPU 並列、(3)MPI:MPI ライブラリを利用した分散メモリー・CPU 型並列である。PTL と MPI,GPU と MPI の 2 つのオプションは同時に利用でき、本課題の成果として、SUSHI の実行モジュールは、利用環境に合わせて 6 種類の中から適当なものを選択できるようになった。但し、並列化版は、1 コア版で実現できる全ての計算方法を実行できる訳ではない。

表 1　Make オプションで make 可能な 6 種類の SUSHI

Type	PTL	GPU	MPI
1 one core			
2 PTL (Pthread Lib.)	ON		
3 GPU(CUDA)		ON	
4 MPI			ON
5 MPR(Hybrid)	ON		ON
6 MPU(MPI+GPU)		ON	ON

結果および考察

1.　GPU による大規模計算

図 4 に示すのは、GPU 並列で可能となった流体力学的効果を導入した動的な SCF 計算[4]によるジブロック・コポリマーの相分離過程のスナップショットである。無秩序状態から、ジブロック・コポリマーがミクロ相分離構造を形成する過程がシミュレートできている。これまでの、1 コアで現実的に計算可能であったシステムサイズの 3^3=27 倍の規模が計算可能となった。

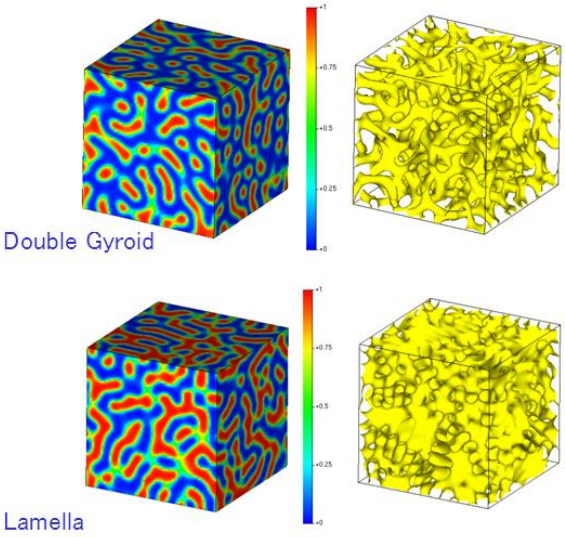

図 4　GPU を利用した流体力学的効果を含む動的な相分離過程の 1 スナップショット：鎖長 N=20 上図は Gyroid(f=0.35, χN=20)構造、下図は Lamella(f=0.5, χN=20)構造が最安定構造である。それぞれ、左図がセグメント密度のコンター図、右図がセグメン密度の等値面図である。システムサイズ 51.6^3、メッシュサイズ 96^3。χ は Flory-Huggins の相互作用パラメータである。

このような大規模計算のメリットは、構造からセグメント密度の散乱関数が得られることである。つまり、セグメント密度は電子密度に相当するので、X 線の小角散乱実験で得られる散乱関数とシミュレーションの散乱関数が比較できる。これまでの 1 コアで可能な計算では、規模が小さく、精度の良い散乱関数は得られなかった[4]。図 5 に得られた散乱関数の例を示す。緑で示すのが、SCF 法で得られている最終的な gyroid 構造の散乱関数で、安定構造に向けて相分離進んでいることがシミュレーションによっても確認することができた。

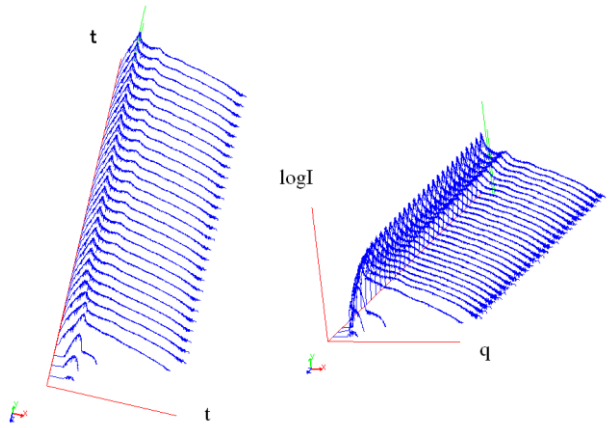

図 5　動的な SCF 計算から得られる時間発展の散乱関数：図 4 の gyroid 構造の計算から得たものである。

次に、図 6 に示すのは、静的な SCF 計算によって得られたジブロック・コポリマーの gyorid 構造と HPL（穴あきラメラ）構造の界面の構造である。このような特徴的なミクロ相分離構造は3D TEM 観察で実験的に得られているが、GPU 並列計算により、容易にシミュレーションできるようになった。

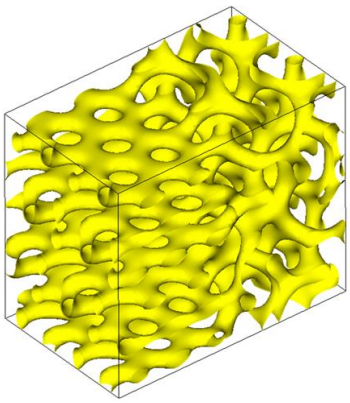

図 6　Gyroid と HPL の界面構造：計算のパラメータは図 4 の gyroid と同じである。メッシュサイズ 56×64×72

2. MPI・GPU 並列による大規模計算

図 7 に示すのは、MPI・GPU 並列(64 コア・64GPU)を用いた SCF の静的計算により得られた、ジブロック・コポリマーの相分離構造の1スナップショットである。計算は無秩序状態から開始したので、多数の異方性があるミクロ相分離構造の集合体であるローカル・ミニマム構造にトラップされている。しかし、これまで、このような大規模な系を静的な SCF 計算では実現できなかったので、初めて計算された例である。システムサイズは 137.6^3 であり、体積が図 4 の構造(GPU のメモリー容量から計算できるほぼ最大の大きさ)の約 19 倍である。

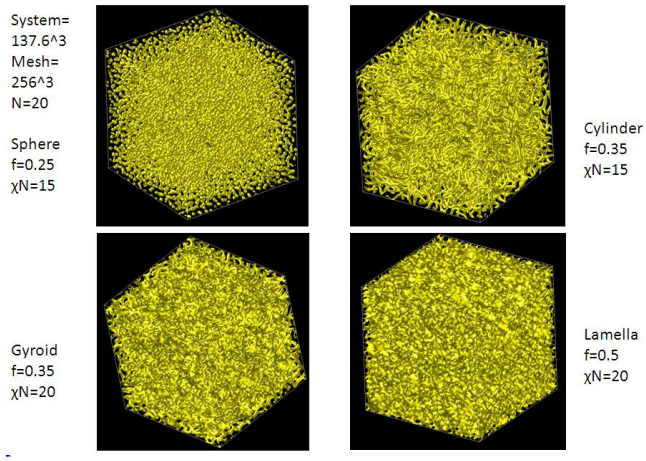

図 7 MPI・GPU 並列で計算した SCF 法の静的な構造最適化計算によるジブロック・コポリマーの相分離構造

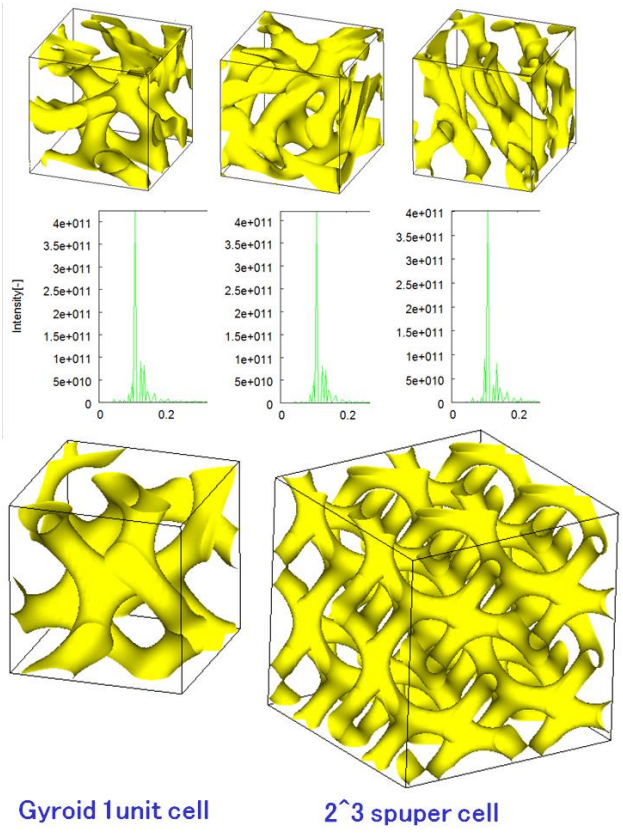

図 8 大規模な静的 SCF 計算の結果から抽出した局所的な高秩序小領域構造とその散乱関数、また、散乱関数が最大となる小領域構造の最適化構図:上図が左から散乱関数強度が1～3位の構造、下図が最適化構造(右図は 2^3 のスーパーセルとして gyroid を確認したもの)

このような大規模計算のメリットとして、ミクロ相分離構造の最安定構造を探索できることにある。図 7 は、大規模なローカル・ミニマム構造であるが、この中にミクロ相分離構造の結晶核に相当する構造が生成されているはずである。そこで、この構造の中から 32^3 メッシュの大きさの小領域を各 X,Y,Z 軸上で 8 メッシュおきに計 32^3 個抽出し、周期境界条件を仮定して局所的な散乱関数を計算した。散乱関数が最大強度となる小領域では、周期性のよい秩序構造ができているはずであるとの仮定ができるので、散乱関数が最大強度となる小領域を周期境界条件下で構造最適化すると図 8 に示すように、最安定構造である gyroid 構造が得られた。つまり、実空間の SCF 計算により、ミクロ相分離構造の安定構造が探索可能であることを示せた。

3. マルチスケール・シミュレーションへの対応

本課題は、当初マルチスケール・シミュレーションへの対応も計画していたが、途中より計画を変更し、MPI・GPU 並列に注力したため、TSUBAME2.0 では当初計画していた粗視化分子動力学計算は行わなかった。しかし、SCF 法の MPI 並列計算の実現を受けて、北大の SR1600/M1 と海洋研究開発機構の地球シミュレータを利用した熱可塑性エラストマーSIS のマルチスケール・シミュレーションを行った。つまり、SCF 法で計算した SIS の相分離構造を初期構造として、SIS のビーズ・スプリング初期構造を発生させ、引張の MD 計算を行い、実材料の応力歪曲線の評価をした[5]。

4. SUSHI のベンチマーク結果

図4で示した gyorid 構造の 1 時間当たりの SCF 回数を速度の指標として、並列化した SUSHI のベンチマーク結果を図9に示す。結果は全て TSUBAME2.0（インテル製 CPU Westmere-EP 2.93GHz, GPU NVIDIA Tesla M2050 515GFlops）を利用した結果である。残念ながら、SUSHI の MPI 並列化において、通信の隠蔽までは達成されていない。よって、ホスト・デバイス間通信が遅い MPI・GPU 並列において、それ程の高速化は達成されていないことを先に述べておく。また、SUSHI は全ての浮動小数点型のデータを倍精度で扱っているので、GPU を利用した単精度の高速化のメリットも享受できていないことも同じく述べておく。

図9の左側は 96^3 メッシュ（1GPU のメモリーの制約から計算できる SCF 計算での最大の大きさ）の結果である。MPU は MPI・GPU 計算の意味である。このベンチマーク時は 1 ノードに存在する 3 つの GPU を利用できないコードを利用しており、MPU 計算は1プロセス1core+1GPU での計算を行っている。最も高速なのは 64core フラット MPI 計算であり、これは、MPU 計算と比較するため、1core/node の計算を行っている。次に速いのは 64core(8core/node) フラット MPI 計算である。64MPU 計算が次に速く、MPI・GPU 計算のメリットはない。次が 1core＋1GPU 計算であり、1GPU のメモリーを最大に利用できる計算規模であるので、この計算の C/P が良い。次は 8MPU 計算であり、1GPU 計算より劣り、やはり MPU・GPU 計算のメリットはない。

96^3 メッシュの結果の右隣に示すのは、その約8倍の規模の 192^3 メッシュでの結果である。これだけの規模になると MPU 計算のメリットが現れる。最も高速なのは 64MPU 計算であり、次が 128MPU 計算である。これらは、128core フラット MPI 計算より高速である。注目すべきは 128MPU より 64MPU の方が高速である点で、ホスト・デバイス間通信速度が遅く GPU の演算能力が高いので、分割した小領域の規模をある程度大きくすると計算が高速になることがわかる。

図の左の方に示すのは、96^3 メッシュのそれぞれ約 19 倍、28 倍の 256^3、$256^2 \times 384$ メッシュの系でのベンチマーク結果である。このベンチマークは（3core＋3GPU）/node により1ノードの全ての GPU を利用するコードでの MPU 計算の結果である。それぞれ、フラット MPI 計算に比べ高速化がはかられている。例えば、最後の最も大規模な計算の場合、96MPU 計算により、96^3 メッシュ 64core フラット MPI 計算に比べ 1/6 倍程度の速度が出ている。単純に系の規模と core 数で概算すると、96core フラット MPI 計算の場合の速度は

$1/28*96/64 \fallingdotseq 1/18$

となるはずであるから、MPU 計算により速度が約3倍向上していることになる。通信のオーバーヘッドが増加しているので、実際にはさらなる高速化が実現されているといえる。大規模計算において数倍の速度の高速化が図れれば、そのメリットは非常に大きいといえる。

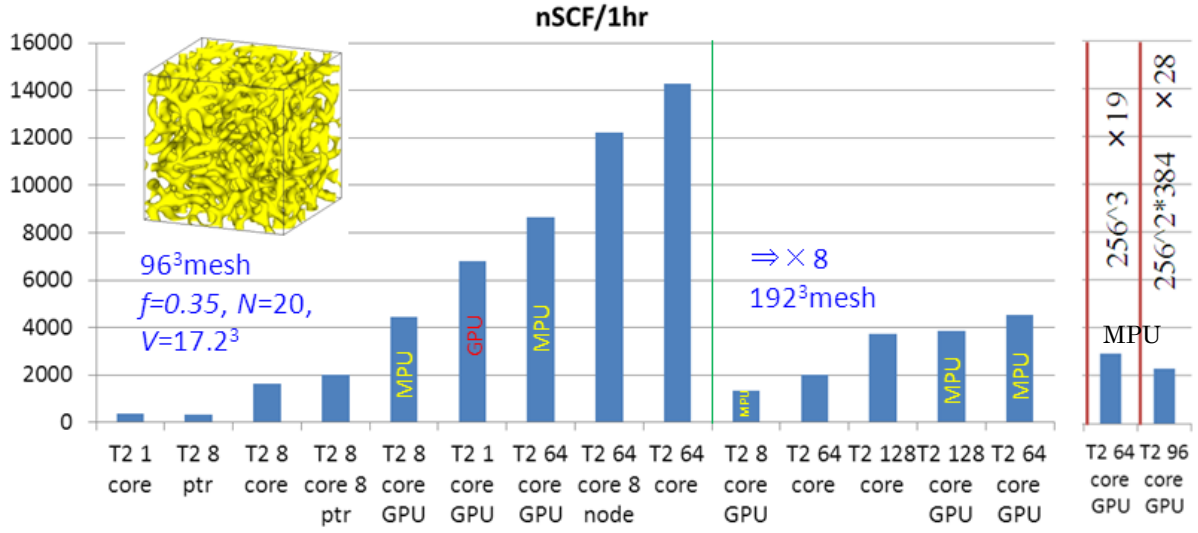

図9　SUSHI のベンチマーク結果：ptr は Pthread 並列、MPU は MPI+GPU 並列を意味する。

まとめ、今後の課題

1. まとめ

OCTA/SUSHI を TSUBAME2.0 を利用したトライアルユースにより、GPU および、MPI・GPU を利用した並列化コードとすることができた。この改良により、これまで不可能であった高分子の SCF 法を用いた高分子材料の大規模で高精度な相分離構造シミュレーションが可能となり、今後、高機能な高分子材料の開発に大きく寄与すると考える。

この成果としての SUSHI は OCTA2013 として、ソースコードを含めて 2013/6 より世界に向けて公開されている。

2. 今後の課題

(a) MPI 計算による通信の隠蔽

残念ながら、通信の隠蔽の実装までは到達せず、ホスト・デバイス間通信が遅いことも併せて、MPI・GPU 並列が実現できても、大規模な系でなければそのメリットを認められなかった。今後、小規模な系でも MPI・GPU 並列のメリットが出るよう、コードを改良する必要がある。

(b) GPU Direct によるデバイス間通信の高速化

通信の隠蔽以外にも CUDA の新バージョンで可能となっている GPU Direct によるデバイス間データ通信の高速化手法を導入すれば、より並列化速度は向上するといえる。

(c) 並列化部分の拡大

本課題での並列化は、周期境界条件の系のみに対応しており、今後、壁（Dirichlet 境界条件）や、反射壁（Neumann 境界条件）等に体操した境界条件のバリエーションを増やす必要がある。また、その他 1core にて実装した SUSHI の機能で、並列化に対応してないものが多く、今後さらなる並列化部分の拡大が必要である。

(d) マルチスケール手法の実装

大規模な高分子の SCF 計算により、精度良く高分子の相分離構造が得られるようになった。今後はそのデータをよりミクロ・マクロなスケールへ受け渡す実装をすることにより、シームレスなズーミングが可能となり、高分子材料の開発により寄与できるものと考える。例えば MPI・GPU 並列が既になされている汎用 MD コードである LAMMPS[6]等とのインターフェースが取れれば、より高分子シミュレーションにおける研究分野が拡大すると考える。

謝辞

本課題実行におきまして、多大なご支援を頂きました東京工業大学学術国際情報センターの皆様方には紙面を借りまして厚く御礼申し上げます。また、MPI 並列化・マルチスケールモデリングに関して多大なご支援を頂きました海洋研究開発機構、北海道大学情報基盤センター 大宮先生、防衛大学校 萩田先生、および OCTA コミュニティーの皆様には同じく紙面を借りまして厚く御礼申し上げます。

参考文献

[1] http://octa.jp

[2] T.Honda and T.Kawakatsu "Computer simulations of nano-scale phenomena based on the dynamic density functional theories: Applications of SUSHI in the OCTA system " in "Nanostructured Soft Matter", A.V.Zvelindovsky, ed., (Springer-Verlag, Berlin, 461-493 (2007).

[3] 青木 尊之,額田 彰 “はじめての CUDA プログラミング”、第二ＩＯ 編集部 (編) (2009).

[4] T.Honda and T.Kawakatsu, *J. Chem. Phys.*, **129** 114904-1 (2008).

[5] 平成 24 年度地球シミュレータ産業利用報告書

[6] http://lammps.sandia.gov