高分子鎖の大規模なLAMMPS粗視化MDのデータ解析の高速化

（GPGPUと高性能IO環境を活かした高分子系緩和現象の解析手法の研究）

萩田克美

防衛大学校　応用物理学科

１．はじめに

スーパーコンピューターの性能進化とともに，高分子材料・高分子物理の分野においても，生体高分子の分子動力学（MD）シミュレーションと同様に，より大規模な系を扱うことが望まれている。現在，MDシミュレーションと言えば，一般に，「古典MD法」と呼ばれ，原子レベルで高分子を構成し汎用ポテンシャルとクーロン力を計算するシミュレーションを思い浮かべる方が多いのが現状である。一方で，高分子材料や高分子物理の分野では，機械的特性の発現メカニズムを理解するために，化学的な詳細を無視した「実在した紐」である性質や抽象化した相互作用の効果を扱う「バネビーズ模型」(Kremer-Grest模型[1])がよく用いられる。例えば，ゴムやプラスチックのような柔らかい材料が粘弾性を持つのは，ある長さ以上の長い高分子鎖が互いに絡まり合うことで，その機械的特性を発現していることが知られている[2]。特に，高分子物理学の立場では，線状の高分子鎖，星形の高分子，複雑に分岐した高分子などが，互いに交差しない鎖（排除体積効果）を持ち，互いに絡まり合うことで，複雑な機械的特性を発現することに興味がある。この特徴的な長さは，高分子の密度に依存する。例えば，鎖を構成するLJ粒子の数密度0.85としたバネビーズ模型は，1本の鎖のビーズ（粒子）の数が，100～200以上の鎖が互いに絡まり合う必要があり，自分自身の周期境界条件の鏡像と直接的な相互作用をしないようにするためには，大きなシステムサイズが必要になる。また，タイヤゴム材料などのナノ粒子を配合した高分子材料のバネビーズ模型では，ナノ粒子の空間構造・配置や凝集状態が機械的特性と関係があると考えられるため，それらの特徴を含んだシミュレーションモデル（初期配置）を扱うためにシステムサイズは巨大となる。

これらの大規模な系のシミュレーションでは，1つの計算ノードのメモリ上に，全座標データを格納することが困難であるため，分散したファイルへの出力を行うことになる。高分子鎖に対してのデータ解析を行う場合，高分子鎖毎（粒子番号毎）にデータを扱う必要が発生する。その場合，空間的に領域分割された出力ファイルから，効率良く処理することが望まれる。今回は，その処理ルーチンの開発検討と，別の計算資源で計算した結果（分散出力ファイル）の処理ために，東京大学 情報基盤センターのスーパーコンピューター「若手・女性利用」の制度で，東大GPUクラスタ等を利用させていただいた。

本稿では，バネビーズ模型の大規模な粗視化分子動力学シミュレーションにおいて，分散出力されたデータの処理を効率良く行うための検討を行った結果（プログラム等の事例）を報告するとともに，LAMMPSの利用に関する紹介も行う。特に，LAMMPSでの初期データセットの作成についても詳しく説明し，多くの方々の利用の参考になるように利用方法を詳しく記載した。（高分子学会の学会誌「高分子」2015年3月号の3ページの紹介[3]では触れることができなかった利用の細部について，本稿にて紹介したいと考えている。）

２．LAMMPS について

高分子材料や高分子物理の分野における粗視化分子動力学シミュレーションでは，大規模な系を扱う場合,米国製の汎用分子動力学シミュレーションコード LAMMPS が用いられることが多くなっている。LAMMPS のメリットについて，先にまとめると，次の点が挙げられる。

・大規模計算での並列性能が高い。国内外の多くのスパコンでの利用実績がある。
・コード改良が比較的容易である。
・粗視化 MD（バネビーズ模型）の計算ができる。ナノ粒子等を剛体として扱える。
・OCTA と連携して利用できる。(OCTA/cognac-LAMMPS コンバーター)

LAMMPS（Large-scale Atomic/Molecular Massively Parallel Simulator）は，米国サンディア研究所で開発されているオープン・ソース・ソフトウェア（OSS）である。ライセンスは，GPL Ver.2 である。LAMMPS は，高分子だけではなく，金属材料や炭素材料などの材料研究や熱流体解析を目的とした汎用 MD シミュレーション・ソフトウェアである。すでに，デファクトスタンダードとして国内外の多くのスパコンの大規模計算で利用されている。プログラムは，データの並列分散性と処理のブロック化を高めたオブジェクト指向で設計され,C++で実装されている。並列性能が高いとともに，改造の容易性で定評があるコードである。(特に，大規模並列計算では，計算機間の通信やデータと処理の分散化が性能に影響するため，改造の容易さを担保しながら並列性の高さを実現できている点が特筆すべき点である。加えて，GPGPU や XeonPhi での高速化を進める取り組みが，比較的活発に行われている点も，注目すべき点である。)

LAMMPS では，全原子模型・フルアトム模型と呼ばれる古典 MD 法に加えて，ランダム力で駆動されるバネビーズ模型，散逸粒子法（DPD），液晶・高分子の計算で用いられる GayBerne 粗視化ポテンシャルなどの大規模並列計算も実装されている。また，高分子中のナノ粒子などの粒子構造体を剛体として設定することができる。さらに，変形などの機械材料系の計算で使われる機能が実装されている。加えて，MD 計算で用いられるオンラインでのデータ解析機能が多く整備されている。これらから，高分子材料や高分子物理での粗視化 MD シミュレーションでのデファクトスタンダードなりつつある。(一方で，生体高分子などで汎用性の高い古典 MD 法については,生体高分子に良く最適化された GROMACS,AMBER や NAMD などのプログラムの方が,LAMMPS に比べて計算性能も高い場合が多い。すなわち，LAMMPS 以外でも計算できる計算内容の場合，他のプログラムとの比較評価が望ましいと考えられる。）なお，LAMMPS では，金属材料・炭素材料で用いられる多体ポテンシャル（EAM，REBO，ReaxFF 等）も利用でき，金属系材料実験結果や第一原理 DFT 計算結果と整合する力場作成を行う OpenKIM（Knowledgebase of Interatomic Models）も利用できる。これらの詳細は，LAMMPS の公式サイト等から調べることができる。

高分子材料・高分子物理の視点では，LAMMPS と，ソフトマテリアル統合シミュレータ OCTA（http://octa.jp/）との連携が進みつつあることも，ポイントの一つである。OCTA は，NEDO プロジェクトで，土井正男先生をリーダーとして開発されたシミュレーション統合パッケージである。OCTA は，gourmet という GUI インターフェースを有し，初期配置の作成，計算条件の設定などのプリ処理と，データの解析や可視化などのポスト処理が可能となっている。OCTA の MD シミュレータである OCTA/cognac の計算データ・条件を，LAMMPS のデータ形式と相互変換する機能（OCTA/cognac-LAMMPS コンバーター）が整備されている。

LAMMPS のメリットの１つは，手元の PC から，スパコンでの大規模計算まで，同一プログラ

ムコード・同一ファイルで利用できることである。また，米国を中心にして，多くのスパコンでの高効率な利用に向けた改良検討が活発であることも，注目すべき点である。さらに，我が国のスパコン（情報基盤センターや「京」等）でも，LAMMPS の利用事例が増え運用サポートが向上しつつあり，大規模計算(HPC)分野としても広範な応用利用を期待されている。最近注目されている GPGPU や XeonPhi などのアクセラレータ（計算加速ボード）への対応は，Sandia 研究所内の他プロジェクトと連携して，精力的に進められている。詳細は LAMMPS ホームページ等を参照して頂きたい。

３．LAMMPS の初歩的な利用方法の紹介

本稿では，高分子学会誌の 2015 年 3 月号の原稿[3]では紙面の都合で記述出来なかった初心者向けの LAMMPS 利用入門について，広い計算機科学分野の方々への紹介も兼ねて，記述する。ここでは，短い高分子鎖の簡単な計算事例を示す。長い高分子が互いに絡まりあった多数の本の系でも，初期配置の作成に工夫を要するものの，基本的には同じ計算条件のファイルを利用できる。(ここでの初期配置作成は，PDB データベースからデータを取得し専用ツールを駆使して準備する生体高分子 MD などの場合とは大きく異なる点を，あらかじめ注意しておく。)

3.1 ソフトウェアの入手

LAMMPS は，公式サイト http://lammps.sandia.gov/で，ソースコード等が配布されている。利用者が自らコンパイル(make)して利用することを基本としている。Linux や Mac ユーザーは，公式ページの手順に従って，簡単にソフトウェアの準備ができる。Windows の場合，やや複雑な手順のため，Windows 用コンパイル済バイナリが，開発者（Axel Kohlmeyer 氏）により，http://rpm.lammps.org/windows.html で配布されている。なお，コンパイル(make)にあたっては，package と呼ばれるプログラム拡張を，必要に応じて導入することになる。(安易に不必要な package を導入すると，不要な苦労の要因になることがあるので注意されたい。利用したい機能に応じて，マニュアルの記述をもとに，パッケージを追加していくスタイルが望ましいと考えられる。) 例えば，剛体の機能のパッケージ(rigid)と波数空間計算機能のパッケージ(kspace)を用いる場合，下記のコマンドでパッケージが導入される。

```
% make yes-rigid
% make yes-kspace
```

3.2 ドキュメント／マニュアル

LAMMPS では，丁寧で詳細なマニュアルとコマンドの辞書が用意されている。配布ソースコードの doc ディレクトリ（pdf 形式，html 形式）や，公式サイトで，提供されている。多くの利用者においては，マニュアルの「3.Commands」の「3.1 LAMMPS input script」を閲覧することが頻繁になるであろう。

3.3 一般的な利用手順　(プログラム実行方法)

上記で準備した，実行モジュール（lmp_linux，lmp_mac，lammps-64bit-latest.exe など）を，下記のように，「計算条件ファイル」を引数に指定して，コマンドライン実行する方法が基本的な使い方である。(下記は，MPI 並列ではない，シリアル実行の例。)

```
% ./lmp_linux -in (計算条件ファイル名)
```

「計算条件ファイル」では，力場ポテンシャル，アンサンブル，計算ステップ数，出力内容とその間隔などを設定するとともに，高分子の構造を設定した「初期構造ファイル」の呼び出し設定を行う。

3.4 練習課題 examples の利用推奨

LAMMPS の配布ソースコードでは，examples ディレクトリに，練習問題が内包されている。「melt: 多数の LJ 粒子の系」「micelle: ミセルの系」「peptide: ペプチドの系」などの，「計算条件ファイル」と「初期構造ファイル」が用意されている。本格利用前に，試験利用してみることを推奨する。「計算条件ファイル」に記述されたコマンドの意味をマニュアルで調べる作業を通じて，LAMMPS 利用の理解が深められると期待される。

3.5 孤立鎖の計算事例 (最も簡単な例)

本稿では，比較的単純な鎖を扱う高分子物理の視点で最も簡単な例を，理解のために紹介したい。ここでは，ビーズ数 5 の 1 本の孤立高分子鎖について扱うことにする。そして，その鎖の情報を，周期境界条件中で複製(replicate)し，凝集させた状態を作成する手順を紹介する。これにより,ベンチマーク評価などに利用できるデータを作成することができる。以下の順で，詳細な手順について，説明する。

1) ビーズ数 5 の 1 本の鎖のバネビーズ模型を，10000 ステップ計算する。
2) 鎖の本数を 4096 (=16^3) に複製して，一辺が 28.8 の立方体の周期境界条件の中に押し込める変形を行う。

図 1 は,「初期構造ファイル」の例である。初期構造ファイルは，原子 (粒子) やボンド数，原子やボンド等の種類の数，周期境界条件の大きさ，原子の質量，原子の配置，ボンドの設定等から構成される。(初期構造ファイルに必要な項目は,atom_style に応じて定められている。)

```
LAMMPS data file

5 atoms
4 bonds

1 atom types
1 bond types

0.0 10.0 xlo xhi
0.0 10.0 ylo yhi
0.0 10.0 zlo zhi

Masses

1 1.0
```

```

Atoms

1 1 1 0.0 0.0 0.0
2 1 1 0.0 0.0 1.0
3 1 1 0.0 0.0 2.0
4 1 1 0.0 0.0 3.0
5 1 1 0.0 0.0 4.0

Bonds

1 1 1 2
2 1 2 3
3 1 3 4
4 1 4 5
```

第 1 図: 初期構造ファイルの例

左図と右図は，1 つのテキストファイルである。ここでは，練習のために，vi 等のエディタで作成することを想定している。実際には，スクリプト言語やプログラムにより作成することが実用的である。

いま，排除体積条件を満たす鎖の模型として，Kremer-Grest 模型[1]を扱うことを考える。図２は，「計算条件ファイル」の例である。

```
units           lj
atom_style      bond
special_bonds   fene

read_data       initial_config.txt
velocity        all create 0.0 12345 dist uniform

pair_style      lj/cut 2.5
pair_coeff      1 1 1.0 1.0 2.5
bond_style      fene
bond_coeff      1 30.0 1.5 1.0 1.0

neighbor        0.4 bin
neigh_modify    every 1 delay 1 check yes

fix 1 all nve
fix 2 all langevin 1.0 1.0 2.0 48279
dump 11 all custom 1000 traj01.lammpstrj id xu yu zu

timestep        0.001

restart         10000 rst01
run             10000
```

第２図：計算条件ファイルの例

Kremer-Grest 模型の標準的な「計算条件ファイル」である。詳細の説明は，マニュアルに委ねる。「初期構造」ファイルの名称として，initial_config.txt を想定している。Dump の出力ファイルの拡張子を「.lammpstrj」としているのは，分子可視化ソフト VMD での利用を簡単にするためである。

上記の「計算条件ファイル」は，次の順に構成されている。

- units，atom_style などの基本設定
- 「初期構造ファイル」の読み込み，速度の設定
- 相互作用の設定
- MD 計算性能に影響する隣接リストに関するパラメータ設定
- fix や dump とコマンド呼ばれる機能の設定
- 時間刻みの設定
- リスタートファイルの設定
- 実行の設定

（LAMMPS がこの行を読み込んだ時点で，上記で読んだ設定で計算が実行される。）

計算を実行すると，時間刻み 0.001 で，10000 ステップの計算が実行される。dump 設定に従い，1000 ステップ毎に，粒子の軌道（トラジェクトリ）データがファイル(traj01. lammpstrj)に出力される。そして，10000 ステップ毎に，restart に必要な情報が出力される。上記の場合，「rst01」の後に，ステップが付加されたファイル(rst01.10000)が作成される。

例えば，分子可視化ソフト VMD（http://www.ks.uiuc.edu/Research/vmd/）では，LAMMPS の軌道データ(原子の座標のみ)を直に読み込むことができ，可視化確認ができる。(単純に，LAMMPS の軌道データを VMD で読み込んだ場合に表示されるボンドは，上記の「初期構造ファイル」とは関係なく，粒子間の距離の条件により設定されることに注意する必要がある。)

次に，上記のファイルを元に，4096 本の高分子鎖の複製と，系の変形を行うための「計算条

件ファイル」を示す。系の複製には，replicate コマンドを用いる。変形については，fix コマンドで deform と呼ばれる機能を設定する。

```
units           lj
atom_style      bond
special_bonds   fene

read_restart    rst01.10000
replicate       16 16 16

pair_style      lj/cut 2.5
pair_coeff      1 1 1.0 1.0 2.5
bond_style      fene
bond_coeff      1 30.0 1.5 1.0 1.0

neighbor        0.4 bin
neigh_modify    every 1 delay 1 check yes

fix 1 all nve
fix 2 all langevin 1.0 1.0 2.0 48279
fix 3 all deform 1 x final -14.4 14.4 y final -14.4 14.4 z final -14.4 14.4
dump 11 all custom 1000 traj02.lammpstrj id xu yu zu

timestep        0.001

restart         100000 rst02
run             100000
```

第３図：系を複製し系を変形させる計算条件ファイルの例

計算を実行すると，ビーズ数５の 4096 本の高分子鎖の配置データを，rst02.1000000 に得ることができる。下記コマンドを実行すると，リスタートファイルの内容が，図１の入力用の「初期構造ファイル」（に近いもの；テキストファイル）に変換することができる。

```
% ./lmp_linux -r rst02.1000000 rst02.txt
```

このファイル rst02.txt は数 MB であり，さほど大きくない。この系は，20480 粒子であり，小さい。もし，大規模計算で，サイズが１万倍になり，時間方向にも蓄積していくことを考えると，処理に工夫を要することになる。大規模データの場合，dump ファイルは，バイナリ形式で，（空間的に領域分割された）MPI プロセス毎に出力することが一般的であろう。その場合，「traj03.%.bin」のように，dump の出力ファイルを指定する。まず，末尾に「.bin」を指定した場合，バイナリで出力される。「%」を指定した場合，各 MPI 領域にその時点で空間領域分割されている粒子の座標値を出力する。なお，dump_modify コマンドの利用により，複数の MPI プロセスに対して１つ MPI プロセスが IO を担当するように設定することができる。(dump_modify 11 fileper 4　のように指定する。)

今回の GPGPU クラスター等の利用では，LAMMPS で出力された多数の空間領域分割されたトラジェクトリファイルを読み込み，高分子鎖毎の挙動を分析するために必要なデータの集約処理の高速化などについて検討した。

４．LAMMPS で空間領域分割された軌道データの集約と時間相関関数計算の高性能化検討

本検討課題では,同じ長さの多数本の高分子鎖で構成される高分子メルト中の微少成分(今回は環状高分子)が緩和現象に及ぼす影響を評価するために実施した超大規模な粗視化 MD シミュレーションのデータ解析を行うために，位置座標の時間相関行列を効率良く計算数ルためのプログラムの開発と検討を目的とした。特に，高性能 IO 環境の下で複数の分散出力ファイルを同時に読み込み，粒子番号分割データとして集約処理する方法と，粒子の位置座標の時間相関行列を効率良く計算するためのMKLやCUBLAS等のライブラリの利用と実用性について検討した事例を報告する。検討したプログラムコード等については，情報共有のために，可能な範囲で詳細に記述した。

4.1 分散出力ファイルの読み込みと粒子番号分割データ化を行う集約処理の検討

前章で記述したように，LAMMPS を用いて並列計算した大規模粗視化 MD シミュレーションでは，粒子の位置座標を，分散ファイルに記録し，記録したデータをもとに解析する。LAMMPS の分散ファイル出力では，原則，各 MPI プロセスで領域分割された空間に存在する粒子の座標を分散ファイルに記録する。(複数 MPI プロセスを束ねて，1 つの MPI プロセスで出力する機能もあるが，基本的には，空間領域分割である。）図４は，分割出力されたファイルの中身である。（テキスト形式と，バイナリ形式で記録される情報は同じである。）

```
ITEM: TIMESTEP
5500
ITEM: NUMBER OF ATOMS
2151
ITEM: BOX BOUNDS pp pp pp
-14.4 14.4
-14.4 14.4
-14.4 14.4
ITEM: ATOMS id xu yu zu
3448 -14.3071 1.58365 -13.3007
641 -13.2649 1.15181 -13.8766
19935 -14.2119 0.476177 14.8977
20235 14.6314 1.59667 14.4829
19931 -13.2106 0.0473453 14.9226
5942 -12.2679 0.645063 -14.0438
5943 -12.3801 0.384719 -13.1487
2087 -11.2703 1.39259 -13.5474
2086 -11.2393 0.472405 -13.717
```

第４図: 軌道データの dump 出力の例

8 個の MPI プロセスで 20480 粒子の系を分散ファイル出力した場合の例。ITEM: NUMBER OF ATOMS は，MPI プロセスが担当する空間領域にある粒子の数である。ITEM: ATOMS 以降に，粒子番号と座標値が出力されている。

解析の際に，１つの MPI ノードのメモリでは，全粒子の座標値を格納することが困難な場合や，解析処理のために分散処理したい場合を考える。各 MPI プロセスで分散ファイルを読み込み，粒子番号順ではなく空間領域分割で分散記録されたデータを各 MPI プロセスで巡回させ，解析処理を担当する粒子番号（高分子鎖番号）のデータを蓄積・処理する仕組みを検討する。LAMMPS の計算で得た N 個の分散出力ファイルについて，M 個の MPI プロセス（計算ノード）を用いて解析する場合を考える。ここで，m 番目の MPI プロセスが担当するファイルの数は，概ね N/M 個となる。

いま，この処理手順を考えると，次のようになる。

1) 各 MPI プロセスで，担当するファイルから，ある時刻の dump データを読み込む。
2) 読み込んだデータを，RING 状のバケツリレーで巡回させる。なお，バケツリレーしているデータの送信と受信（交換）を同時に行う。MPI プロセス数を偶数とすることで，RING の両方向で巡回させることができる。MPI プロセス数が奇数の場合は，単方向での巡回となる。（なお，メモリが足りる場合は，Allreduce を用いる方が簡単である。）
3) 受信したデータの中から，担当する粒子のデータをピックアップする。
4) 2)～3)の手順を繰り返し，データ 1 周したところで，次のファイルを読み込む。
5) 次の時刻のデータの読み込みとして，1)～4)を繰り返す。

図 5 には，上記を実装したプログラムの例を示しておく。（MPI プログラムに不慣れな方への参考として，冗長ではあるが，読み込み処理部分のプログラム全体を示している。）

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef int tagint;
typedef int64_t bigint;
#define BIGINT_FORMAT "%" PRId64

int main(int argc, char **argv) {
  int nfiles=600;
  FILE *fp[600];
  int natoms0=256*512*16*16*16;
  int natoms1=natoms0/32;
  int natoms2=natoms0/300;
  int itimemax=80;
  double posi[natoms1][3], buf0[1+natoms2];
  double buf_s[1+natoms2],buf_r[1+natoms2];

  int i,j,k,m,n,jm,itmp,itime,ibuf;
  bigint ntimestep,ndump;
  int idim,iside,size_one,nchunk,triclinic;
  double xlo,xhi,ylo,yhi,zlo,zhi,xy,xz,yz;
  int boundary[3][2];
  char boundstr[9];
  bigint maxbuf = 0;
  double *buf = NULL;

  char fname[256];
  int ifl,ifl_st,ifl_en,src, my_rank, nprocs;
  int size_s[1],tag_s,size_r[1],tag_r;
  int size_s0,size_r0;
  MPI_Status stat;

  MPI_Init(&argc, &argv);
  MPI_Comm_size(MPI_COMM_WORLD,&nprocs);
  MPI_Comm_rank(MPI_COMM_WORLD,&my_rank);

  ifl_st=(nfiles*my_rank)/nprocs;
  ifl_en=(nfiles*(my_rank+1))/nprocs-1;

for (ifl=0; ifl < ifl_en-ifl_st+1; ifl++) {

sprintf(fname,"traj04.%d.bin",ifl+ifl_st);
    fp[ifl] = fopen(fname,"rb");
    if(fp[ifl]==NULL) printf("Fail open ¥n");
}
```

```
for (itime=0; itime < itimemax; itime++) {

  ibuf=0;
  for (ifl=0; ifl < ifl_en-ifl_st+1; ifl++) {
      // detect end-of-file
      if (feof(fp[ifl])) {
        fclose(fp[ifl]);
        break;
      }

  fread(&ntimestep,sizeof(bigint),1,fp[ifl]);
  fread(&ndump,sizeof(bigint),1,fp[ifl]);
  fread(&triclinic,sizeof(int),1,fp[ifl]);
  fread(&boundary[0][0],6*sizeof(int),1,
                                  fp[ifl]);
  fread(&xlo,sizeof(double),1,fp[ifl]);
  fread(&xhi,sizeof(double),1,fp[ifl]);
  fread(&ylo,sizeof(double),1,fp[ifl]);
  fread(&yhi,sizeof(double),1,fp[ifl]);
  fread(&zlo,sizeof(double),1,fp[ifl]);
  fread(&zhi,sizeof(double),1,fp[ifl]);

  if (triclinic) {
    fread(&xy,sizeof(double),1,fp[ifl]);
    fread(&xz,sizeof(double),1,fp[ifl]);
    fread(&yz,sizeof(double),1,fp[ifl]);
  }
  fread(&size_one,sizeof(int),1,fp[ifl]);
  fread(&nchunk,sizeof(int),1,fp[ifl]);

  double tmp[size_one];

  m = 0;
  for (idim = 0; idim < 3; idim++) {
    for (iside = 0; iside < 2; iside++) {
      if (boundary[idim][iside] == 0)
                        boundstr[m++] = 'p';
      else if (boundary[idim][iside] == 1)
                        boundstr[m++] = 'f';
      else if (boundary[idim][iside] == 2)
                        boundstr[m++] = 's';
      else if (boundary[idim][iside] == 3)
                        boundstr[m++] = 'm';
    }
    boundstr[m++] = ' ';
}
  boundstr[8] = '¥0';
```

第 5 図：分散出力ファイルのデータ読み込みプログラムの例

この例は，600 個に分散されたファイルを，32 個の MPI プロセス（計算ノード）で読み込み，粒子番号分割として，1 つの配列に格納するプログラムである。解析部分の記述は，割愛している。

```
// loop over processor chunks in file
for (i = 0; i < nchunk; i++) {
    fread(&n,sizeof(int),1,fp[ifl]);
    // extend buffer to fit chunk size
    if (n > maxbuf) {
      if (buf) delete [] buf;
      buf = new double[n];
      maxbuf = n;
    }
    fread(buf,sizeof(double),n,fp[ifl]);
    n /= size_one;
    m = 0;
    for (j = 0; j < n; j++) {
      buf0[1+ibuf*4]=itmp;
      buf0[2+ibuf*4]=buf[m++];
      buf0[3+ibuf*4]=buf[m++];
      buf0[4+ibuf*4]=buf[m++];
      ibuf++;
    }
}
  buf0[0]=ibuf;
  }

MPI_Barrier(MPI_COMM_WORLD);

  for(k=0;k<nprocs;k++) {
  tag_s=100;
  tag_r=200;

  if(k==0) {
    itmp=buf0[0];
    size_r[0]=itmp;
    size_r0=size_r[0]*4+1;
    for(i=0;i<size_r0;i++){
      buf_r[i]=buf0[i];
    }
  } else {
    size_s[0]=size_r[0];
    size_r0=size_r[0]*4+1;
    size_s0=size_s[0]*4+1;
    for(i=0;i<size_r0;i++){
      buf_s[i]=buf_r[i];
    }
```

```
    if((k-1+my_rank)%2==0){
      src=my_rank+1;
      if(src==nprocs) src=0;
    } else {
      src=my_rank-1;
      if(src==-1) src=nprocs-1;
    }

    MPI_Sendrecv(&size_s,1,MPI_INT,src,tag_s,
                 &size_r,1,MPI_INT,src,tag_s,
                 MPI_COMM_WORLD,&stat);
    size_s0=size_s[0]*4+1;
    size_r0=size_r[0]*4+1;
    MPI_Sendrecv(&buf_s,size_s0,MPI_DOUBLE,
                  src,tag_s,&buf_r,size_r0,
                  MPI_DOUBLE,src,tag_s,
                  MPI_COMM_WORLD,&stat);
  }

    jm=buf_r[0];
    for(j=0;j<jm;j++){
      itmp=(int) buf_r[1+j*4];
      itmp=itmp-my_rank*natoms1;
      if(itmp >= 0 && itmp < natoms1) {
      posi[itmp][0]=buf_r[2+j*4];
      posi[itmp][1]=buf_r[3+j*4];
      posi[itmp][2]=buf_r[4+j*4];
      }
    }
  }

MPI_Barrier(MPI_COMM_WORLD);

/*
   Analysis code will be writen here.
*/

} // for (itime=0; itime < itimemax; itime++)
} // int main(int argc, char **argv)
```

第５図：分散出力ファイルのデータ読み込みプログラムの例　（つづき）

上記のプログラムにて，高性能な IO 性能をもつ分散並列計算機環境で，多数のファイルに同時にアクセスしつつ，また，１つの計算ノードのメモリの格納できないケースの解析を進めることが可能となった。さらに，従来の MPI_Allreduce の実装で計算できる規模においても，読み込んだ分散データを RING 状に巡回させる方式で性能向上の効果もあった。

4.2 緩和モード解析に用いる位置座標の時間相関行列の計算処理の高速化

我々は，高分子鎖毎の緩和挙動の分析のために，位置座標の時間相関関数を計算し，その統計平均を取得している。これは，高分子鎖の遅い緩和挙動を取り出すための手法として，緩和モード解析[4,5]を行うためである。緩和モード解析では，シミュレーション中で記録した全ての粒子の位置座標 $\boldsymbol{r}_i(t)$ に対する時間相関行列 $C_{ij}(t)=<\boldsymbol{r}_i(t)\ \boldsymbol{r}_j(0)>$ を計算し，複数の時刻の時間相関行列に対する一般化固有値問題やフィッティングにより，緩和モード・緩和率スペクトルを評価することができる。全ての位置座標の時系列データという大規模データから，遅い成分を統計的に取り出す手法である。例えば，高分子材料系の粘弾性の性質を記述する応力緩

和関数の評価では，従来の方法（直接計算した応力の時間相関行列を，単指数緩和の和の形で緩和率とその振幅をフィットする方法）に比べて，1/20 以下の MD シミュレーション長でも，精度良く，応力緩和関数の挙動を予測できる[6]。これは，理論的な裏付けに基づき，緩和モード解析で独立に評価した緩和率スペクトルを，応力の時間相関行列のフィットに利用することで，緩和率と振幅の同時フィット問題を，振幅だけのフィット問題に軽減し，応力の時間相関行列の統計精度への要求が楽になったためである。

1 本の鎖が N個の粒子で構成される高分子鎖について，1 本鎖近似（1 本の高分子鎖内の相関が系の挙動を記述し，他の鎖との相関はないと考える仮定。）を仮定した解析で用いる時間相関行列の計算は，次の式に従って行われる。この表式から，N x Mの行列 A と N x Mの行列 B から，N x Nの行列 C を計算する DGEMM 関数がそのまま利用できることが分かる。

$$C_{ij}(t) = \frac{1}{L}\sum_{l=0}^{L-1}\frac{1}{3}\sum_{r=x,y,z}\frac{1}{M}\sum_{k=0}^{M-1} r_{i+N*k}(t+l)\,r_{j+N*k}(0+l)$$

$r_{i+N*k}(t)$:　時刻 t の k 番目の鎖の i 番目の粒子の位置座標

N:　1 本の高分子鎖の中の粒子数（コード例中では，num。）

M:　高分子鎖の本数　（コード例中では，kmax。）

L:　時間方向の平均を取る時間ステップ数

ここで、$C_{ij}(t)$は、時間 t=0 から T-1 まで計算することを考えている。

この DGEMM 計算は，MKL ライブラリや CUBLAS ライブラリを用いることで，簡便に，その性能を享受できる可能性がある。(実際には，データの詰め替えコストや CPU-GPU 間のデータ転送コストなどのオーバーヘッドにより恩恵が受けられない場合がある。)本検討で利用したプログラムコードのポイントを，読者の参考のために，図６に示しておく。

ここで，1 本の高分子鎖内の時間相関関数の計算では，鎖の内の粒子数 Nは，数百程度であり，行列のサイズが小さくその性能向上の効果は薄くなる。(以前の検討では，オーバーヘッドの方が大きくなり，高速化が難しかった。Nが 128 や 256 などの数百程度では逆効果となることが多かった。）最近，MKL や CUBLAS において，小さいサイズの行列に対する DGEMM の性能向上がなされていることから高速化が期待できると考え，詳細の検討を行った。また，Fujitsu 製の SSL2 の BLAS ライブラリの利用[7]についても検討の余地があると考えられるため,参考として評価した。

本検討では，ベンチマーク評価として，512 個の粒子からなる高分子鎖が 864 本ある系について，12 個の分散ファイルに出力された軌道データを，４つの MPI プロセスで処理した場合の比較検討を行った。N=512，M=864，L=90，T=10 の条件で，1 タイムステップ辺りの時間相関関数の計算部分（データの詰め替えとライブラリの呼出実行）の経過時間を評価した。ベンチマーク評価は，東京大学 情報基盤センターの GPU クラスタを主として実施した。CPU は，Intel Xeon X5670 2.93GHz x 2 (2 ソケット 12 CPU コア)であり，GPU は，Nvidia Tesla M2050 である。参考として，東京工業大学学術国際情報センターの TSUBAME2.5 を比較参考のために，利用した。TSUBAME2.5 では，CPU は東大 GPU クラスタと同じであり，GPU が NVIDIA Tesla K20X と新しく高速なものになっている。ここで，CUDA(CUBLAS)のバージョンは，5.0 を用いた。

```
// MKL

// Memory Allocation
  mklA=(double*) mkl_malloc(num*kmax*sizeof(double),64);
  mklB=(double*) mkl_malloc(kmax*num*sizeof(double),64);
  mklC=(double*) mkl_malloc(num*num*sizeof(double),64);

// Setting Array mklA[k+i*kmax], mklB[k+j*kmax]
// Clear Array mklC[]=0.0

  alpha=1.0;
  beta=0.0;
  cblas_dgemm(CblasRowMajor,CblasNoTrans,CblasTrans,num,num,kmax,alpha,
              mklA,kmax,mklB,kmax,beta,mklC,num);
```

```
// CUDA 5

// Memory Allocation
  cudaMallocHost((void **)&cudaA,sizeof(double) * num * kmax);
  cudaMallocHost((void **)&cudaB,sizeof(double) * kmax * num);
  cudaMallocHost((void **)&cudaC,sizeof(double) * num * num);
  cublasAlloc(num*kmax,sizeof(double),(void **)&devA);
  cublasAlloc(kmax*num,sizeof(double),(void **)&devB);
  cublasAlloc(num*num,sizeof(double),(void **)&devC);

// Setting Array cudaA[k+i*kmax], cudaB[k+j*kmax]
// Clear Array cudaC[]=0.0

  LDA=kmax;  LDB=kmax;  LDC=num;
  cublasSetMatrix(kmax,num,sizeof(double),cudaA,LDA,devA,kmax);
  cublasSetMatrix(kmax,num,sizeof(double),cudaB,LDB,devB,kmax);
  cublasSetMatrix(num,num,sizeof(double),cudaC,LDC,devC,num);

  cublasDgemm('T','N',num,num,kmax,alpha,devA,LDA,devB,LDB,beta,devC,LDC);
  cublasGetMatrix(num,num,sizeof(double),devC,num,cudaC,LDC);
```

第６図: MKL ライブラリと CUDA ライブラリ利用の概要

MKL ライブラリと CUDA ライブラリで，本問題において BLAS の DGEMM 関数を使うプログラムコードのポイントを示している。

ベンチマーク評価した結果を表１に示す。最近のライブラリを用いることで，「MKL の BLAS の DGEMM」や「GPGPU 上の BLAS（CUBLAS）の DGEMM」では，単純にコーディングしたコードに比べて速かった。「MKL の BLAS の DGEMM」を 12 スレッド並列とした場合と，「GPGPU 上の BLAS の DGEMM」を Nvidia Tesla M2050 で実行した場合を比較すると，「MKL の BLAS の DGEMM」の方が，性能が高い結果になった。この結果は，東工大 TSUBAME2.5 等でも同様であり，GPGPU 性能の差よりも，本体-GPGPU 間のデータ転送コストの影響が大きいと考えられる。なお，FX10 での SSL2 を利用した BLAS の DGEMM を評価したが，行列サイズが小さいために性能向上の効果は見られなかった。

表１：　ベンチ-マーク評価の結果

並列化の条件	経過時間
4MPI x 1 OpenMP	101 msec
4MPI x 3 OpenMP	47.1 msec
4MPI x 12 OpenMP	26.2 msec
4MPI with GPU	63.0 msec
4MPI with GPU (TSUBAME2.5, K20X)	36.1 msec

いま，時間 t=0 から T-1 までの $C_{ij}(t)$ を計算することを考えているので，行列 A と B を，Nx M x T とすることで，サイズ N x N x T の行列 C を得る問題と置き換えることができる。予備的な検討では，MKL の場合には，この方針で高速化できることを確かめた。(CUDA の場合は，配列の形状が正方形から外れたために，性能が悪化した模様である。) 詳細の検討は，今後検討する予定である。また，本来目的である多数本の高分子鎖の溶融体中の微少な環状高分子を含む系の大規模シミュレーション (N=256 の鎖が約 200 万本ある系 (5.37 億粒子)) のトラジェクトリデータの解析では，本研究課題で検討したプログラムと MKL ライブラリを用いた。

５．まとめ

本稿では，LAMMPS の概要紹介や，高分子物理学で用いる粗視化 MD シミュレーション（バネビーズ模型）での基本的な利用について記述するとともに，分散出力された軌道データの解析に効率的な実施のために,比較的高性能な IO 環境と GPGPU クラスターを利用した事例について報告した。１つの計算ノードでは，メモリ不足で，全粒子の位置座標データを格納できないケースについて，複数の MPI プロセスでファイルを読み込み，RING 状にデータを巡回させることで，空間領域分割されたデータから粒子番号分割のデータに集約処理するプログラムについて検討した。粒子番号分割したデータから，位置座標の時間相関関数を計算する処理について，MKL ライブラリや CUBLAS ライブラリの DGEMM 関数を用いた高速化についても検討した。

謝辞

本検討でのプログラム開発検討にあたり,東京大学情報基盤センター　片桐准教授をはじめ，関係各位のご支援・ご協力に，感謝いたします。

参　考　文　献

[1] K. Kremer and G. S. Grest: J. Chem. Phys., 92 (1990) p.5057

[2] M. Doi and S. F. Edwards　『The theory of polymer dynamics』, Oxford University Press, New York, (1986)

[3] 萩田　克美 『高分子研究のための LAMMPS 利用の入門』, 高分子学会　学会誌「高分子」2015 年 3 月号, (2013), p.147

[4] S. Koseki, H. Hirao, and H. Takano: J. Phys. Soc. Jpn. 66 (1997) p.1631

[5] K. Hagita, H. Takano: J. Phys. Soc. Jpn. 71 (2002) p.673

[6] N. Iwaoka, K. Hagita, and H. Takano: J. Phys. Soc. Jpn. 84 (2015) in press.

[7] 片桐 孝洋『富士通 PRIMEHPC FX10 チューニング連載講座　６．数値計算ライブラリ』, 東京大学情報基盤センター スーパーコンピューティングニュース, Vol. 15, No. 6, (2013) p.15