TSUBAME 共同利用　令和 6 年度　学術利用　成果報告書

TSUBAME 4.0 における LAMMPS の性能評価
Performance analysis of LAMMPS in TSUBAME 4.0

太田 幸宏
Yukihiro Ota

高度情報科学技術研究機構 神戸センター 利用支援部
Department of HPC support, Kobe center, Research Organization for Information Science and Technology
https://www.hpci-office.jp/ristkobe/

TSUBAME 4.0 において、汎用分子動力学シミュレーションコードである LAMMPS の GPU 利用に関する性能評価を行う。LAMMPS の開発者が公開している様々な基礎的ベンチマークデータを対象に、多様な計算機能を確認する形で NVDIA GPU 向けの実装である GPU パッケージと KOKKOS パッケージの比較をする。結果、今回の調査の範囲では、様々力場への対応と性能の観点から KOKKOS パッケージの利用が優位ということがわかった。さらに、LAMMPS における GPU 実装を理解するため、Kokkos を利用した単純な GPU 向けカーネルコードの作成に取り組む。

We examine the performance of a molecular-dynamics simulator, LAMMPS, with use of GPUs in TSUBAME 4.0. LAMMPS has two different kinds of implementations for NVIDIA GPU, GPU and KOKKOS packages. We explore the characteristics of these two implementations, focusing on various kinds of the benchmark data sets prepared by the developers in LAMMPS. Within our surveys, we find that use of KOKKOS package is preferable since this package supports various kinds of force fields and the performance is better than that in GPU package in many cases. Furthermore, we write simple kernel programs for GPUs with Kokkos library, leading to a better understanding of GPU implementation in LAMMPS.

Keywords: 分子動力学法, LAMMPS, Kokkos, 性能評価, スケーラビリティ

背景と目的

LAMMPS [1, 2]は物質・材料分野で広く利用されている汎用分子動力学（MD）シミュレーションコードである。NVDIA GPU 向けの実装として、CUDA で実装された GPU パッケージ、および Kokkos [3]を通して CUDA をバックエンドとして利用する KOKKOS パッケージの 2 種類が利用できる。これらの 2 種のパッケージは別々に実装されているため、対応する計算機能や性能などの相違点について調査することは、GPU を搭載する計算機で LAMMPS を利用する上で有用な情報と考えられる。

本利用課題では、NVIDA H100 を搭載した HPC 向け大型計算機である TSUBAME 4.0 において、LAMMPS の GPU 利用に着目した性能評価を行う。LAMMPS 開発者が公開している基礎的なベンチマークデータ [4]を中心に、GPU パッケージと KOKKOS パッケージを比較した。その結果、今回の調査の範囲では、計算機能（力場）への対応状況、そして性能の観点から、KOKKOS パッケージの利用が優位ということを見出した。また、KOKKOS パッケージの詳細を調べるため、マルチ GPU 利用によるスケーラビリティを調査した。さらに、LAMMPS における GPU 実装を理解するため、Kokkos を用いた単純なカーネルコードの作成を行った。こうして、TSUBAME 4.0 で LAMMPS を効率よく実行するための基本的な知見と機能拡張に必要となる技術的知見を得ることができた。

概要

TSUBAME 4.0 で LAMMPS の性能評価を行う。バージョンは 2Aug2023.3 とする。GPU 利用では、GPU パッケージと KOKKOS パッケージを比較する。また、LAMMPS における GPU 実装を理解するため、Kokkos を利用した単純な GPU 向けカーネルコードの作成にも取り組む。

表 1: LAMMPS の性能評価で使用するパッケージ

凡例	パッケージ名	詳細
pure	-	pure-MPI 並列向けの標準的なパッケージ（デフォルト）。
opt	OPT	CPU 向けのチューニング版パッケージ。
omp	OPENMP	CPU 向け OpenMP スレッド並列版パッケージ
gpu	GPU	CUDA、HIP、OpenCL で実装された GPU 向けパッケージ。
kkg	KOKKOS	KOKKOS で実装されたパッケージ。バックエンドで CUDA を使用。
kkghn	KOKKOS	kkg に-pk kokkos neigh half newton on を追加 [6]。全原子 MD 計算(例: rhodo)に関するベンチマークの実行で必要。

TSUBAME 4.0 では GPU パッケージ対応の LAMMPS がプリインストールされている。しかし KOKKOS パッケージは有効ではない。そこで、GPU および KOKKOS パッケージを含むように LAMMPS をソースからコンパイルする。

LAMMPS の性能評価では MD 計算の典型例をデータの対象とする。用いるデータは、公開ベンチマークデータ [4]やソースファイルに同梱されているサンプルデータとする。基本ベンチマークとして chain, chute, eam, lj, rhodo の 5 種類、力場ポテンシャル・ベンチマーク から 12 種類、サンプルデータから機械学習ポテンシャルの一種である SNAP 力場 [5]を評価する。また、LAMMPS の replicate 機能を用いることでデータサイズの拡張を行い、複数 CPU コアやマルチ GPU 利用時の性能を調査する。計測は複数のパッケージについて行う（パッケージの詳細については [6] の Package command を参照)。表 1 に性能評価で使用したパッケージ名を凡例として示す。

性能分析は LAMMPS の標準出力ログに表示される時間計測情報に基づく。すなわち、MD 計算のメインループの経過時間、(Time steps)/sec.（単位秒で進む MD ステップ数)、そして計測区間に応じた時間内訳に着目する。時間内訳においては、MD の典型的な高負荷部分である力場の特性、および GPU 利用で生ずるボトルネックを調べる。計測時のコアバインディングは、OpenMPI のオプションで--bind-to core を指定して制御する。ただし、MPI+OpenMP のハイブリット並列では、CPU コアのタイムシェアリングを回避するため、--bind-to none に設定し、OMP_PLACES と numactl コマンドにコア ID を陽に指定する。具体的には、スレッド 0（つまり MPI のランク 0）が割り当てられたコア ID の近傍にスレッドが生成するよう設定する。

Kokkos を利用した GPU 向けカーネルコードの作成では、Kokkos 4.3.01 を使用する。LAMMPS の KOKKOS パッケージで典型的に使用されているパターンを念頭に置き、Kokkos を用いた GPU コーディングの基本的な技術の習得を目指し、行列行列積、STREAM、MPI 通信における PingPong の通信パターンなどのカーネルを作成する。

結果および考察

本報告書では、LAMMPS の性能評価の結果を主に示す。Kokkos を利用したカーネルコード作成については、LAMMPS の性能評価の後で簡潔に示す。

LAMMPS の性能評価の結果を示す前に、本利用課題で使用した LAMMPS の構築設定をまとめる。構築環境は NVIDA HPC SDK (nvhpc) 24.1 および OpenMPI 5.0.2 を用いた。CPU 向けコードの最適化レベルは-O3 とした。また、高速フーリエ変換のライブラリとして、GPU 向けに cuFFT、CPU 向けに FFTW3.3.10 (システム側で提供) を利用する。有効にする LAMMPS の計算機能は、外部ライブラリとのリンクが最小限になるものとした（インストール時設定の most に相当)。また、CPU 向けスレッド並列に対応する OPENMP パッケージ、NVDIA GPU を使用するために GPU および KOKKOS パッケージを有効にする（表

1)。なお、GPU パッケージでは NVIDIA 以外の GPU 利用もサポートされている（詳細は[6]の GPU Package を参照)。 Kokkos は LAMMPS のソースファイルに内包されているものを使用した（バージョンは 3.7.2)。Kokkos におけるアーキテクチャ設定を TSUBAME 4.0 に対応させる。すなわち、CPU は Zen3（-DKokkos_ARCH_ZEN3=yes, つまり nvc++における-tp=zen3 に相当。TSUBAME 4.0 の CPU としては Zen4 が適切だが、今回使用した Kokkos では対応していないため Zen3 とした)、GPU は HOPPER90 (-DKokkos_ARCH_HOPPER90=yes, つまり nvcc における-arch=sm_90 に相当）とした。

まず、5 種類の基本ベンチマークの計測結果を示す（図 1)。計測では TSUBAME 4.0 の 1/4 ノード(node_q キュー）を使用した。基本ベンチマークのデータサイズについて、全て原子数 32000 の小規模データである。横軸に示された pure, opt, omp は CPU のみの実行であり、gpu, kkg, kkghn が GPU を利用した実行に対応する。性能を(Time steps)/sec.で評価する。すなわち、大きな数値ほど高速に MD 計算を実行できることを意味する。1 CPU コアでの実行である pure に対し、gpu, kkg, kkghn について、1 GPU でデータ lj, eam, rhodo において 50~70 倍程度の速度向上となることがわかる。データ chain, chute では 10 倍程度の向上となる。GPU パッケージと KOKOS パッケージを比較すると、データ eam については GPU パッケージ(gpu）が最速、それ以外は KOKKOS パッケージ(kkg あるいは kkghn）が最速となる。 kkg と kkghn の差はほぼ見られなかった。pure による 1 CPU コア実行に対する GPU 利用の性能向上が顕著であったデータの中から lj と rhodo に着目し、計測区間に応じた経過時間内訳を調べる（図 2)。CPU 利用では Pair が全体の 80%程度を占める高負荷部である。これは MD 計算における力場の処理に対応し、演算のみで構成されている。GPU 利用により Pair が大幅に削減されていることがわかる。また、GPU 利用時における高負荷部がパッケージに依存して大きく変わることを把握できた。

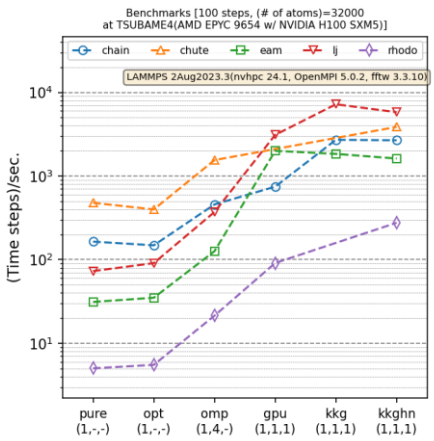

図 1: 基本ベンチマークの性能。横軸はパッケージ名（表 1)、縦軸は性能指標である(Time steps)/sec.を示す。パッケージ名の下の三組（np,nt,ng)は並列設定を示す（np は MPI プロセス数、nt は OpenMP スレッド数、ng は GPU 数。記号”-“は未実装を意味する)。データにパッケージが対応していない場合、プロットは表示されない。

次に、力場ポテンシャル・ベンチマークの結果を示す(図 3)。図 1 の設定と同様に、TSUBAME 4.0 の 1/4 ノードを使用して計測した。横軸は力場の種類を示す。12 種類の力場について、gpu および kkghn の pure による 1 CPU コア実行からの速度向上が示される。パッケージで対応していない力場では、グラフは示されていない。図 2 から、gpu に比べ kkghn が対応する力場が豊富であることがわかる。性能はデータ eam を除けば、gpu より kkghn が優勢である。

図 2 で示した力場以外にも、GPU パッケージでは未対応だが KOKKOS パッケージで対応している力場がある。その中で、機械学習ポテンシャルである SNAP 力場について計測した。1 例として、材料モデル InP の 32768 原子データに対する計測結果を示す。48 MPI プロセスの pure の実行で 5.27 (Time steps)/sec.、1 MPI プロセス、1 スレッド、1 GPU の kkghn の実行で 81.0 (Time steps)/sec.であり、48 CPU コアの実行に対して 15 倍程度の速度向上となる。他の材料モデル(WBe, Ta)においても同様な傾向だった。

これまでは GPU 利用時に 1 CPU コアのみを利用していた。ここで、1 枚の H100 に対し、CPU コア数を増加させたときの挙動から、GPU パッケージと KOKKOS パッケージを比較する。データは基本ベンチ

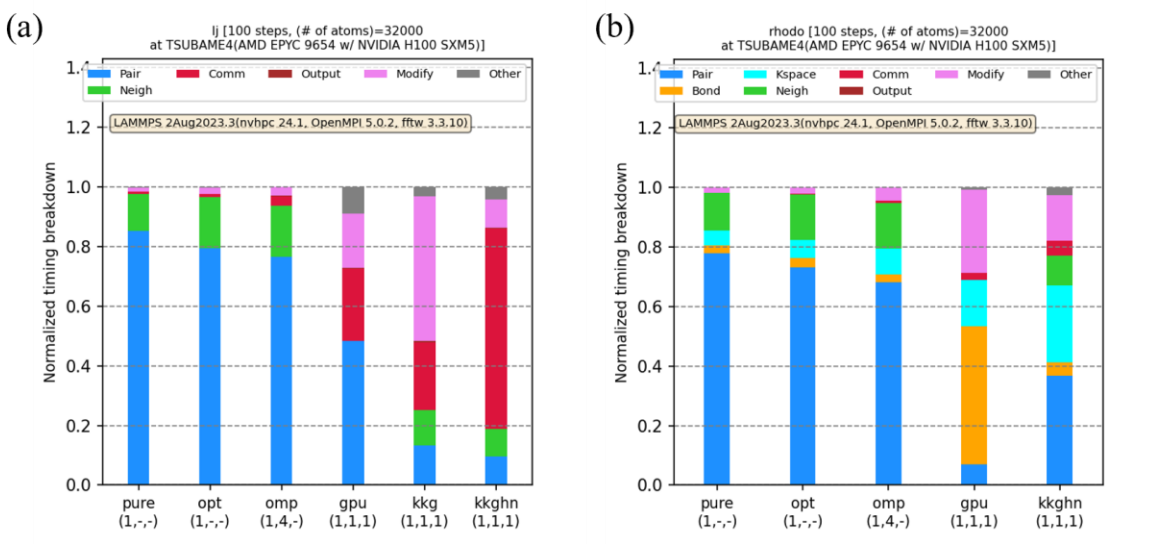

図 2: 基本ベンチマークにおける経過時間内訳。横軸は図 1 と同じ。縦軸はメインループの経過時間で規格された時間。(a) lj、(b) rhodo。

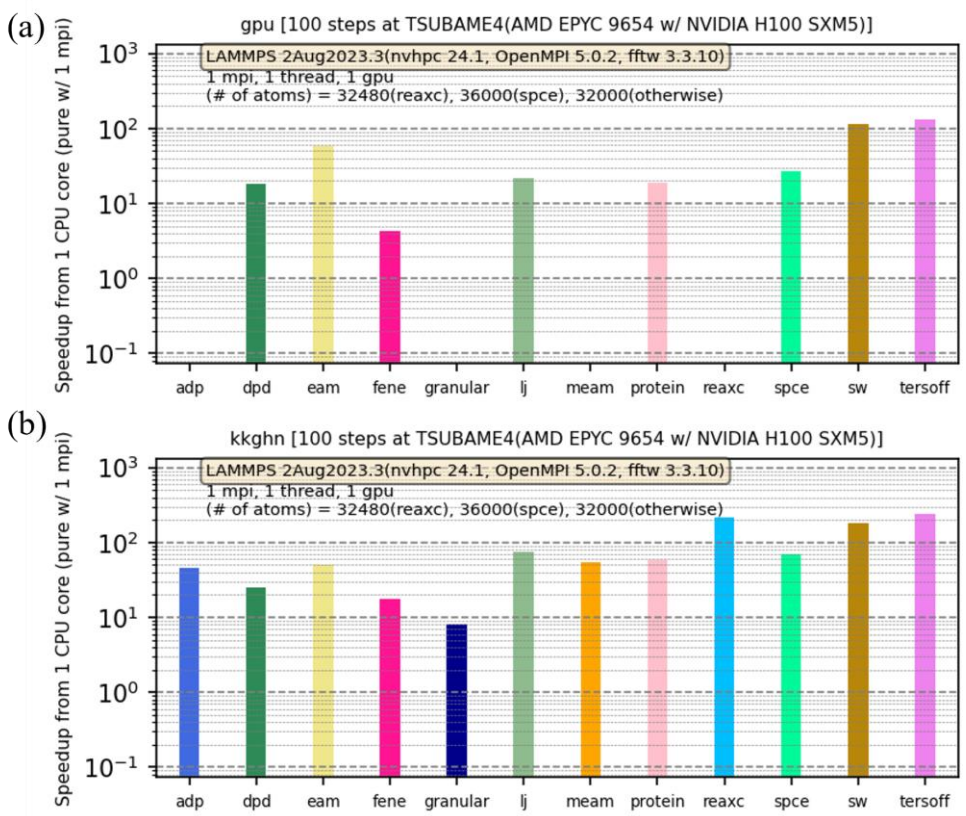

図 3: 力場ポテンシャル・ベンチマークに関する GPU 利用による速度向上。横軸はベンチマーク名（力場の種類に対応）、縦軸は 1 CPU コア利用（pure で 1 MPI プロセス）からの速度向上。(a) gpu、(b) kkghn。パッケージ名（pure, gpu, kkghn）の詳細は表 1 を参照。データにパッケージが対応していない場合、グラフは示されない。

マークから lj と rhodo に着目する。データサイズは replicate コマンドで拡張した。オリジナルのデータについて、3 次元空間の x 軸、y 軸、z 軸方向をそれぞれ 2 倍拡張した（256000 原子データ）。図 1 における計測

表 2: 1 MPI, 1 スレッド, 1 GPU 使用時からの速度向上。データはサイズ拡大した lj (256000 原子)。性能のベースラインは、(Time steps)/sec.で、gpu について 506、kkghn について 2064。

	MPI	スレッド数	Speedup
gpu	1	2	1.3
gpu	2	1	1.2
kkghn	1	2	1.1
kkghn	2	1	0.11

表 3: 1 MPI, 1 スレッド, 1 GPU 使用時からの速度向上。データはサイズ拡大した rhodo (256000 原子)。性能のベースラインは、(Time steps)/sec.で、gpu について 10.1、kkghn について 105。

	MPI	スレッド数	Speedup
gpu	1	2	1.0
gpu	2	1	1.9
kkghn	1	2	1.0
kkghn	2	1	0.20

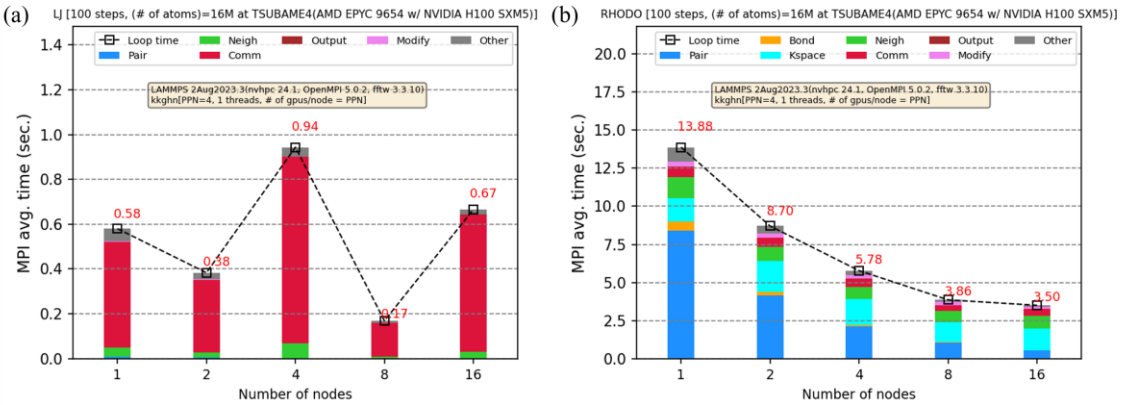

図 4: データ拡張された基礎ベンチマークに対するストロングスケール測定（16384000 原子)。横軸はノード数、縦軸は経過時間。ノードあたりの MPI プロセス数は 4 に固定し、GPU はノードあたり 4 枚使用する。OpenMP スレッド数は 1 に固定。(a) lj、(b) rhodo。

と同様に、TSUBAME 4.0 の 1/4 ノードを使用する。まず、1 CPU コア、1 スレッド、1 GPU の設定で、GPU パッケージより KOKKOS パッケージのほうが高速であることが確認された（lj で 4 倍、rhodo で 10 倍の性能差)。それぞれのパッケージについて、この性能をベースラインとし、CPU コア数増大による速度向上を調べる（表 2、表 3)。GPU 1 枚に対し、CPU コア数を 2 とする。CPU コアは OpenMP スレッド並列か MPI プロセスかのどちらかに割り当てる。gpu ではスレッド数と MPI プロセス数のどちらについても、増大で正の効果を得た。特に、データ rhodo について、MPI プロセス数の増大は理想的な性能向上が見られた（なお、4 MPI プロセスで性能向上は飽和した)。LAMMPS のドキュメント[6]の GPU Packages において、GPU パッケージでは 1 GPU に対して MPI プロセス数の増大による性能向上の可能性が指摘されており、その記述と無矛盾な結果と言える。一方、kkghn については、スレッド数の増大で性能は変わらず、MPI プロセス数増大は性能劣化を引き起こすことがわかった。MPI ランクと GPU のデバイス ID に関する 1 対 1 のバインディングを破る実行は、LAMMPS のドキュメント[6]の Kokkos Packages において非推奨とされており、その記載を計測で確かめることができたと言える。このように、GPU パッケージと KOKKOS パッケージとで性能向上に寄与する CPU コアの設定方法が異なることがわかった。GPU パッケージは、CPU コアを余らせることなく使用できる可能性がある。ただし、今回の調査の範囲では、KOKKOS パッケージの性能に迫る設定を見出すことはできなかった。一方、KOKKOS パッケージでは、さらなる高速化を狙う場合は、MPI ランクとデバイス ID の 1 対 1 バインデ

表 4: Kokkos によるカーネル作成例: 行列行列積カーネルの性能（倍精度浮動小数点，行列次元: 1024）。

配列次元	実装方法	Gflop/s
2	Kokkos: TeamPolicy, TeamVectorRange	2918.253
1	OpenACC	3411.408
1	cuBLAS	34775.811

ィングを保ちつつ、GPU 数（すなわちノード数）を増やす方策が考えられる。

ここまでの結果から、多様な力場への対応と性能の観点で、NVIIDA GPU を使った LAMMPS の実行では KOKKOS パッケージの利用が優位と考えられる。

LAMMPS の性能評価の最後として、マルチ GPU 利用でのスケーラビリティを調べる。この調査では KOKKOS パッケージのみに着目する。データは lj および rhodo を x 軸、y 軸、z 軸方向それぞれに 8 倍に拡張した 16384000 原子数データを対象とする。上記の調査結果に従い、ノード内における MPI プロセス数(PPN) はノード内 GPU の数と一致させておく。TSUBAME 4.0 では 1 ノードあたり 4 枚の H100 が搭載されているため、PPN は 4 に設定することになる。OpenMP スレッド数は 1 に設定する。計測では、node_f キューを使用し、ノード数を 1 から 16 まで増大させた。結果を図 5 に示す。データによりスケーラビリティが大きく異なる。データ lj では、Pair の寄与はほぼ消えており、性能のボトルネックは通信関数を含む計測区間である Comm であることがわかった。小規模実行の段階で（図 2)、lj は単一 GPU 利用で十分に Pair の寄与が削減されている。こうして、データ lj は KOKKOS パッケージによるマルチ GPU 利用に適するものではなかったと判断される。一方、データ rhodo については、8 ノードまで良好なスケーラビリティを示しており、Pair の寄与がノード数の増大とともに順調に削減している。高ノード側におけるスケーラビリティの阻害要因は高速フーリエ変換を含む計測区間である Kspace となることがわかった。

最後に、Kokkos を利用したカーネルコード作成とその検証についてまとめる。今後の LAMMPS の KOKKOS パッケージおける機能拡張やチューニングなどを念頭に置き、Kokkos を利用した簡単なカーネルを作成した。MD 計算自体とは直接関係はないものの、Kokkos を利用したコーディングで必要となる要素技術を習得できるようなカーネルとして、行列行列積、STREAM、MPI 通信関数の呼び出しなどを実装し、OpenACC や CUDA との実装の結果と比較した。本報告書では、倍精度密行列に対する行列行列積の実装と計測結果を示す(表4)。リスト 1 に Kokkos を使った実装の例を示す。レファレンスとして実装した OpenACC の例をリスト 2 に示す。Kokkos と OpenACC とで同程度の性能となることがわかる。素朴な実装であり、cuBLAS による実行の 1/10 程度の性能となった。リスト 1 のような階層的並列は LAMMPS の力場カーネルでも使われているパターンであり、そのコーディング方法を把握できた。

まとめ、今後の課題

TSUBAME 4.0 における LAMMPS による MD 実行において基礎的な性能評価を実施した。GPU 向けコードとして異なる実装である GPU パッケージと KOKKOS パッケージを比較し、今回の調査の範囲では、力場への対応状況と性能から、KOKKOS パッケージの利用が好ましいことがわかった。ノード内実行および複数ノードを使った実行についても性能上の知見を得ることができた。また、Kokkos を用いた単純なカーネルコードを実装し、LAMMPS の機能拡張に必要となるコーディング上の基礎知識を習得した。作成したカーネルコードは、RIST が開催する講習会等でサンプルとして公開したいと考えている。

リスト 1: Kokkos による行列行列積の実装例（Kokkos: TeamPolicy, TeamVectorRange）。

```
// Kokkos: TeamPolicy, TeamVectorRange
typedef Kokkos::Cuda ExecSpace;   struct TagA {};
void MyKernel::mm_A () {
   Kokkos::parallel_for("Kernel_A",
      Kokkos::TeamPolicy<TagA,ExecSpace>(NN, Kokkos::AUTO, 32), *this);
}
KOKKOS_INLINE_FUNCTION
void MyKernel_kk::operator() (TagA,
   const typename Kokkos::TeamPolicy<ExecSpace>::member_type & team) const
{
   int i = team.league_rank();
   // use of copy ([=]) would correctly work.
   Kokkos::parallel_for(Kokkos::TeamVectorRange(team, 0, MM), [&] (const int & j ) {
      double tmp = 0.0;
      for ( int k = 0; k < LL; ++k ) {
         tmp += d_A( i, k ) * d_B ( k, j );
      }
      d_C( i, j ) = tmp;
   });
}
```

リスト 2: OpenACC による行列行列積の実装例。

```
// OpenACC
#pragma acc loop gang independent
for ( int i = 0; i < NN; ++i ){
#pragma acc loop vector independent
for ( int j = 0; j < MM; ++j ) {
   double mctmp = 0.0;
   #pragma acc loop seq
   for ( int k = 0; k < LL; ++k ) {
      mctmp += ma[k + LL*i]*mb[j + MM*k];
   }
   mc[j + MM*i] += mctmp;
}}
```

課題として、以下の 3 点が考えられる。第一に、KOKKOS パッケージの利用で複数 CPU コアを有効に利用できていない点が挙げられる。検討として、例えば、CPU コアのスレッド並列の効果について、スケールしない原因がデータに起因するか、実装の問題なのかを明らかにすることは重要と考えられる。1 ジョブの中で MPMD モデル的に複数のプログラムを動かす方策も CPU コアの有効活用の観点で重要と思われる。第二に、KOKKOS パッケージで実装されていない機能について検証を進めることが挙げられる。KOKKOS パッ

ケージは GPU パッケージに比べれば多くの力場に対応しているが、例えば自由エネルギー計算（FEP パッケージ）には対応していない。こうした計算機能の GPU 対応を進めたい。第三に、今回の検証から外した外部ライブラリを使用する計算機能について GPU 利用時の性能を検証することが挙げられる。例えば、QM/MM 計算で用いられる QMMM パッケージなどが考えられる。

参考文献

[1] A. P. Thompson, H. M. Aktulga, R. Berger, D. S. Bolintineanu, W. M. Brown, P. S. Crozier, P. J. in't Veld, A. Kohlmeyer, S. G. Moore, T. D. Nguyen, R. Shan, M. J. Stevens, J. Tranchida, C. Trott, and S. J. Plimpton, Comput. Phys. Commun., 271, 10817 (2022).

[2] LAMMPS Molecular Dynamics Simulator; https://www.lammps.org/

[3] Kokkos: https://kokkos.org/

[4] LAMMPS Benchmarks; https://www.lammps.org/bench.html

[5] A. P. Thompson, L.P. Swiler, C.R. Trott, S.M. Foiles, and G.J. Tucker, J. Comput. Phys. 285, 316-330 (2015).

[6] LAMMPS Documentation; https://docs.lammps.org/