2.5 LAMMPS/ReaxFFのFX100における性能評価とチューニング

株式会社豊田中央研究所
平井 宏俊

2.5.1 はじめに

LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator) [1] はアメリカのSandia 国立研究所が中心になって開発を進めているオープンソース (GPL) の古典分子動力学計算パッケージである。LAMMPS は原子、粗視化粒子、有機分子、プロテイン、DNA、金属、グラニュラーなどが取り扱える汎用分子動力学シミュレータとなっており、様々な種類の力場が取り扱える。ここでは化学反応が取り扱える多体ポテンシャルReaxFF [2] に着目し、スーパーコンピュータFujitsu PRIMEHPC FX100 での性能評価ならびにチューニングを試みた結果について報告する。

2.5.2 プログラム概要

2.5.2.1 LAMMPS

シミュレーション領域の空間分割を用いたMPI 並列による並列化プログラムとなっている。プログラミング言語にはC++が用いられており、機能はモジュール方式で基本的にソースとヘッダファイルのみで構成されている。そのため拡張性が高く、コードの80%がコア部分から拡張された部分となっている。

2.5.2.2 ReaxFF

ReaxFF は、van Duin 教授らによって開発された反応力場(Reactive Force Field) である。ReaxFF は、既存の古典分子動力学プログラムとは異なり、結合の生成と開裂を記述することができる。反応力場のパラメータは量子化学計算のデータから決められているために化学反応の高精度な記述を可能にする一方で、その動力学計算は通常の古典分子動力学シミュレーションに匹敵する計算速度を保持しており、触媒反応や燃焼反応など、材料科学分野における分子動力学シミュレーションの適用範囲を大幅に広げている。ReaxFF はいわゆるボンドオーダーポテンシャルとなっており、通常の分子力場法に比べて演算数、条件分岐が多い傾向がある。LAMMPS に実装されているReaxFF にはC++版 (reaxc) とFortran 版 (reax) があるが、本報ではより高速であるC++版を用いる。

2.5.3 手法

一般に、アプリケーションの中で多くの実行時間が費やされている箇所（ホットスポット）を見つけ出して、その部分を高速化すると、大きなチューニング効果を得ることができる。プロファイラを使用すると、実行時間の分布などのチューニング情報を得ることができる。ここでは富士通製のFX100 用のプロファイラを用いてアプリケーションの分析を行った。また、LAMMPS のプログラムが標準で書き出す各ルーチンのタイマー情報も適宜活用した。

2.5.3.1 測定方法

プロファイラには、チューニング情報をサンプリング方式で収集する基本プロファイラと、指定した区間のチューニング情報を収集する詳細プロファイラがある。基本プロファイラは、アプリケーションの実行性能情報の収集および出力を、コマンド（_pp）で実現している。これにより、時間統計情報（経過時間、ユーザCPU 時間とシステムCPU時間の内訳など）、コスト情報（アプリケーション実行中にサンプリングを行い、その回数を手続、ループおよび行単位にコストとして出力）、ハードウェアモニタ情報（アプリケーション実行時のプロセッサ動作状況を出力）などの情報を出力することができる。一方、詳細プロファイラは、アプリケーションの実行性能情報の収集および出力をコマンド(fapp)で実現している。これにより、基本情報（測定区間の呼出し回数、および時間情報）、MPI 情報（MPI 関数の実行回数、メッセージ長、実行時間、および待ち時間の平均値、最大値および最小値）、ハードウェアモニタ情報（アプリケーション実行時のプロセッサ動作状況）などを出力することができる。各プロファイラには区間指定機能があり、コスト情報を測定する範囲を指定することができる。ソースコード上で測定範囲を指定するには、コスト情報の測定開始位置と測定終了位置にサブルーチン(fipp_start、fipp_stop など）を挿入する。C/C++言語の関数を使用する場合は、関数のプロトタイプ宣言を行う、または、基本プロファイラ区間指定機能のヘッダファイルをインクルードする必要がある。

2.5.3.2 測定環境

本報では名古屋大学情報基盤センターに導入されているスーパーコンピュータ Fujitsu PRIMEHPC FX100 を用いて性能評価を行った。以下の表に上記システムの構成を示す。

使用システム	名古屋大学　FX100
プロセッサ	SPARC64 XIfx
動作周波数	2.2 GHz
コア数/ノード	32 (+アシスタントコア2)
演算性能/ノード	1.126 TFLOPS
メモリ容量/ノード	32 GiB
メモリバンド幅	480 GB/s
総ノード数	2,880（92,160コア）
総理論演算性能	3.2 PFLOPS
総メモリ容量	90 TiB
OS	LinuxOS（専用OS）

図1：測定環境

2.5.3.3 測定条件

本報では以下に示す条件において性能評価を行った。

プログラム version	lammps-10Feb15
翻訳オプション	-Kfast,preex,ocl,simd=2,optmsg=2 -Xg -Nsta -Nsrc -Nstl=500fast -Nline - Nlst=t
原子数	3600 (メタン 400 分子、酸素 800 分子、図 2 参照)
時間刻	10.1 fs
温度	2500 K
体積	100 Å× 100 Å× 100 Å
MD 手法	NVT （粒子数、体積、温度一定）
温度制御	BerendsenThermostat
MD ステップ数	2000
隣接原子リストの更新間隔	10 ステップごと
Output	原子座標ファイル（1000 ステップごと）
ReaxFF パラメータ	ReaxFF for Hydrocarbon Oxidation [3]

MDステップ数が2000 となっている以外は、通常の反応解析でよく用いられる条件になっている。通常は5 千から5 万倍の1000 万から1 億 MDステップの計算になるが、計算量が単純に5千から5万倍になるだけであり、演算の質は変わらない。スケーラビリティ以外の性能評価では1 ノード16 コアを用いた。

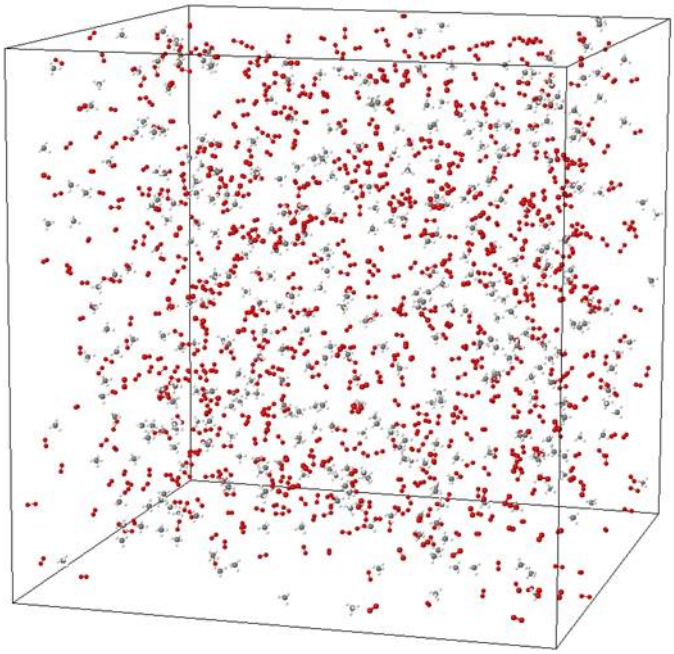

図2: 評価に用いたユニットセル

2.5.4 結果

2.5.4.1 基本プロファイリングデータ

図3 に基本プロファイリング（fipp コマンド）によって得られたProcedures プロファイルを示す。exp、pow の組込関数のコストが大きいことが分かる。これらの組込関数はインライン展開されておらず、インライン展開することで性能向上する可能性がある。Kfastオプションに含まれるKilfunc オプションでは一部の組込関数がインライン展開されることになっている。全てインライン展開すると逆に実行性能が遅くなる場合があるため、コンパイラは閾値を設定して制御している。今回インライン展開されなかったのは、この閾値を超えていたためと考えられる。そこで全ての組込関数をインライン展開するようコンパイル時にオプション指示し、上記の閾値を変更した。インライン展開後のProceduresプロファイルを図4 に示した。組込関数のインライン展開後はpow、exp などのコストが消え、トータルのコストは6322 から3652 に、実行時間（Operation）は629.3517 から362.5718 に大幅に削減されており、およそ45%の速度向上を達成した。

Cost	%	Operation (S)	Start	End	
6322	100.0000	629.3517	--	--	Application
1038	16.4189	103.3294	--	--	__ieee754_pow pow
737	11.6577	73.3904	511	557	LAMMPS_NS::CommBrick::reverse_comm()
502	7.9405	49.9580	--	--	__expl
336	5.3148	33.4408	--	--	__ieee754_exp exp
316	4.9984	31.4565	--	--	__jwe_etbf
217	3.4325	21.5924	77	416	Valence_Angles(_reax_system *, control_params *, simulation_d
214	3.3850	21.3049	368	598	BO(_reax_system *, control_params *, simulation_data *, stora
185	2.9263	18.4221	--	--	__isnan
168	2.6574	16.7235	947	980	LAMMPS_NS::CommBrick::forward_comm_fix(LAMMPS_NS::Fix *, int)
158	2.4992	15.7389	947	971	LAMMPS_NS::FixQEqReax::parallel_dot(double *, double *, int)

図3: 基本プロファイリングデータ (Procedures) インライン展開前

Cost	%	Operation (S)	Start	End	
3652	100.0000	362.5718	--	--	Application
300	8.2147	29.7835	--	--	__jwe_etbf
278	7.6123	27.5986	77	416	Valence_Angles(_reax_system *, control_params *, simulation_d
270	7.3932	26.7962	511	557	LAMMPS_NS::CommBrick::reverse_comm()
255	6.9825	25.3489	368	598	BO(_reax_system *, control_params *, simulation_data *, stora
247	6.7634	24.5143	34	202	vdW_Coulomb_Energy(_reax_system *, control_params *, simulati
202	5.5312	20.0495	260	359	BOp(storage *, _reax_list *, double, int, int, far_neighbor_d
169	4.6276	16.7732	947	980	LAMMPS_NS::CommBrick::forward_comm_fix(LAMMPS_NS::Fix *, int)
159	4.3538	15.7886	--	--	__GI_strcmp
159	4.3538	15.7906	947	971	LAMMPS_NS::FixQEqReax::parallel_dot(double *, double *, int)
147	4.0252	14.5985	49	52	rvec_ScaledAdd(double *, double, double *)

図4: 基本プロファイリングデータ (Procedures) インライン展開後

図5 にインライン展開前後の実行時間内訳を示した。ここでPair、Neigh、Comm、Output、Other はそれぞれ、力場の計算、MPI 通信処理、アウトプット出力処理、その他の処理時間を表している。インライン展開後はPair の実行時間が大幅に短縮されている。これはReaxFF の力場の計算にpow やexp が多く含まれるためである。Comm の実行時間も減っているのが確認できる。この要因は以下のように考えられる。後述の通り、Pair の計算部分にロードインバランスがあり、待ち時間が発生しComm のコストが増えてしまう。Pair の計算時間が減ったことでロードインバランスの絶対値が小さくなり、待ち時間も減ったためComm の実行時間が減ったと考えられる。

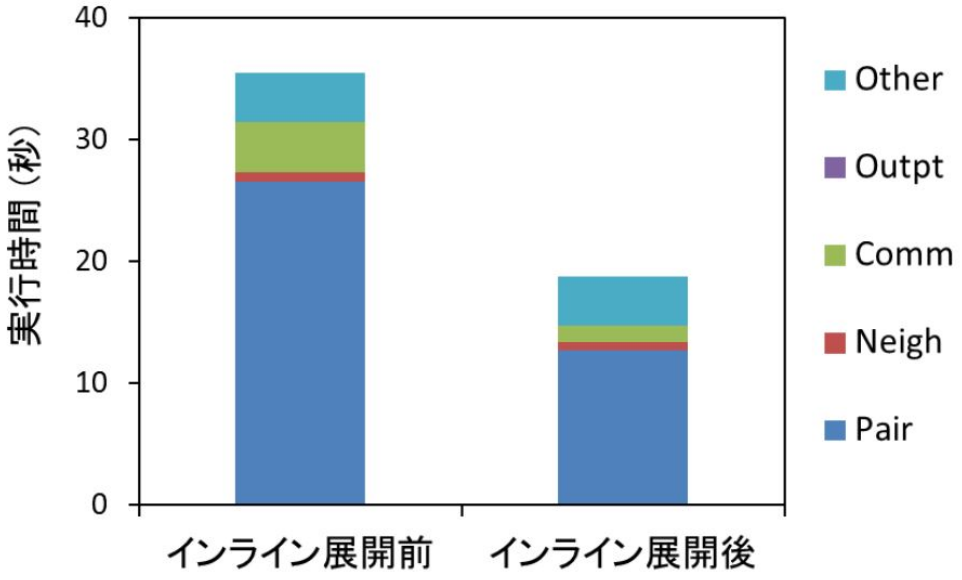

図5: インライン展開前後の実行時間内訳

2.5.4.2 ロードインバランス

全体の中で最もコストが大きい力場の計算部分（Pair）をさらに高速化することで、大幅な速度向上が望める。そこで力場の計算部分を担う関数force->pair->compute() について実行時間をプロセス毎にまとめた。結果を図6に示す。プロセス番号は実際のものではなく、実行時間でソートしている。プロセスごとに実行時間が大きく異なっており、ロードインバランスがある。ロードインバランスがあると、一部のプロセスに待ち時間が生じるため、実効効率が下がってしまう。ロードインバランスを解消することで速度向上が望める。

LAMMP にはdynamic load-balancing という機能がある（fix balance コマンドで調整可）。この機能を用いると各プロセッサに負荷が均等に分配されるように各プロセッサが担当する粒子数のバランスをとることができる。実際には各プロセッサが担当するサブ領域のサイズと形を調整することで各プロセッサが担当する粒子数を変化させている。このバランスをとる作業を分子動力学計算中に定期的（ユーザー設定）に行うことでdynamic なロードバランシングを行うことができる。図7 にdynamic load-balancing による実行時間の変化を示した。1.5 %程度の速度向上が確認できる。このときのforce->pair->compute()の実行時間をプロセス毎にまとめた結果を図8 に示す。dynamic load-balancing の適用後、ロードインバランスが改善していることが分かる。今回はプロセス数が16 と小さいために、速度向上も僅かであったが、大規模並列計算を実行した際には、ロードインバランス解消により、大幅な計算効率の向上が見込まれる。

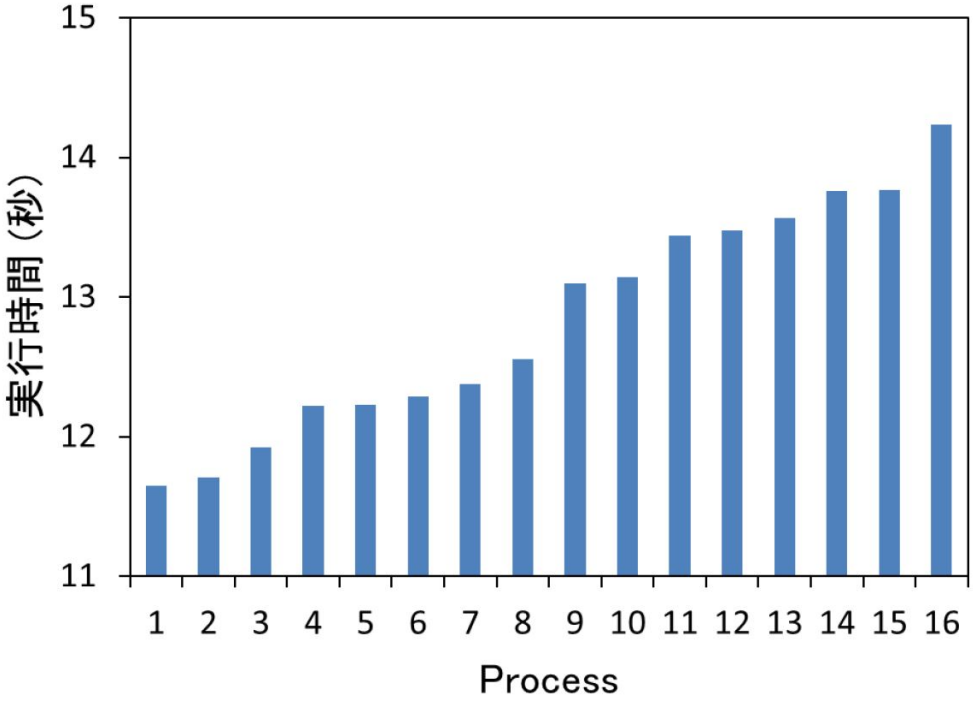

図6: 力場の計算部分のプロセス毎の実行時間

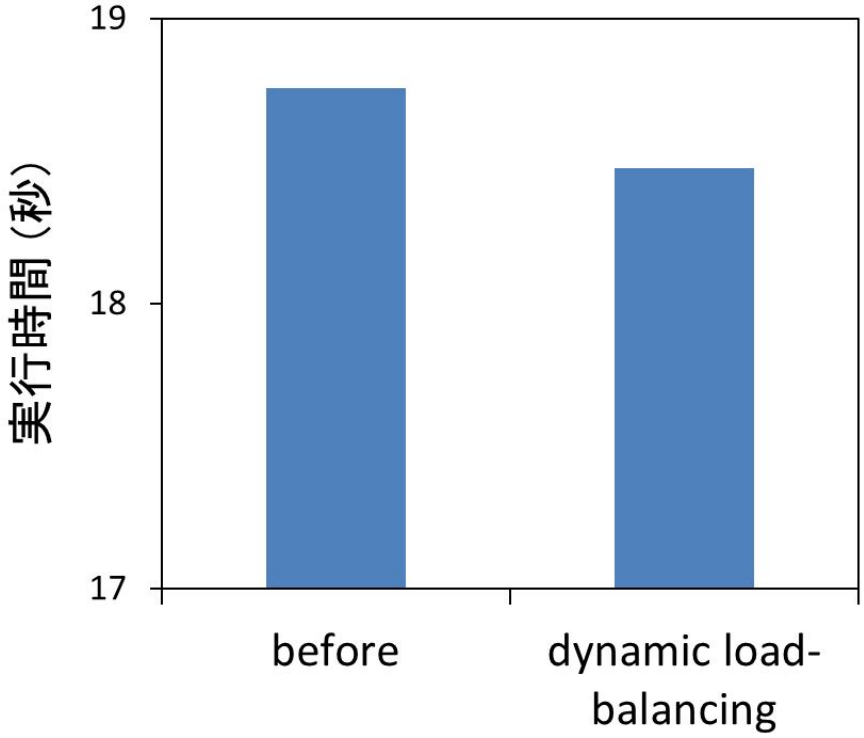

図7: dynamic load-balancing の効果

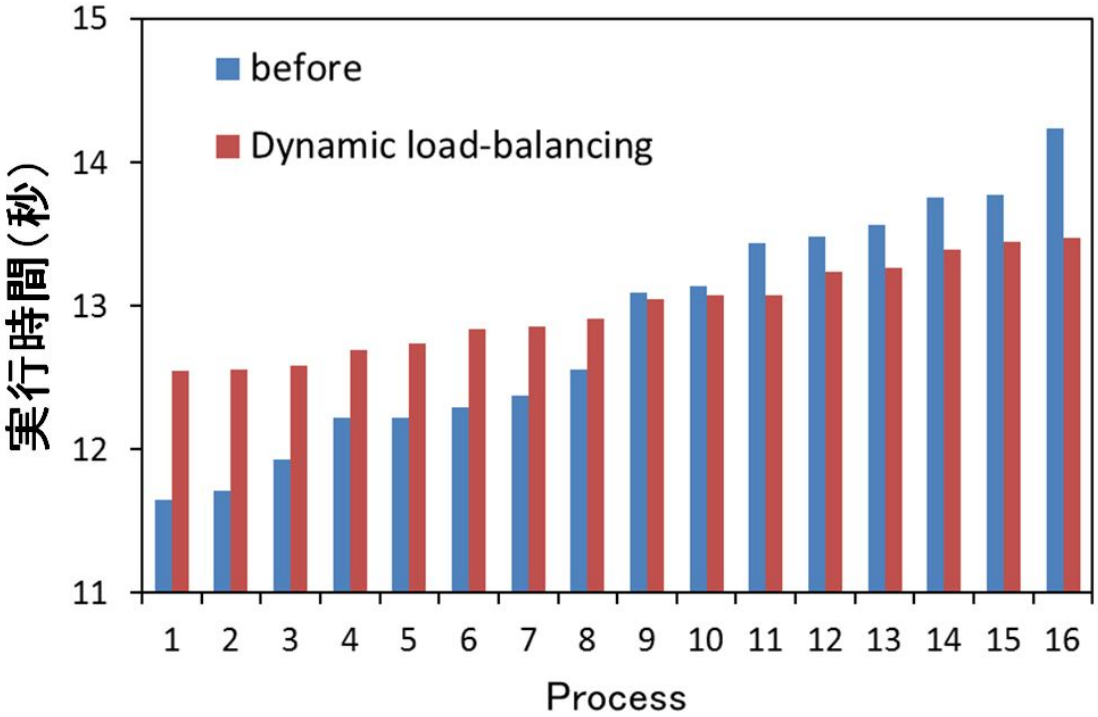

図8: 力場の計算部分のプロセス毎の実行時間(dynamic load-balancing)

2.5.4.3 詳細プロファイリングデータ

ここでは、高コスト関数force->pair->compute() の詳細プロファイリングデータ（精密PA）を取得した結果を示す。最初に実行時間の内訳を図9 に示す。浮動小数点演算待ちが大きいことが分かる。次に、メモリ・キャッシュ・SIMD 情報をまとめた結果を図10 に示す。キャッシュミス率は小さく、メモリ・キャッシュアクセスについては最適化の余地はないが、SIMD 命令率が極端に小さいのが目立つ。SIMD 化を促進することで性能向上が見込まれる。本報でコンパイル時に用いているKsimd=2 オプションは、if 文などを含むループも含めてSIMD 拡張命令を利用したオブジェクトを生成するオプションであるはずだが、上手く機能していないようである。そこで、force->pair->compute()内で呼び出されている高コストな関数の例としてボンドオーダーの計算ルーチンであるBO() のソースコードの構造を確認した（図11）。SIMD 化は最内ループに適用されるが、BO() 内のループの最内ループは条件分岐が多く複雑な構造となっている。この複雑な条件分岐のためSIMD化が適用できなかったと考えられる。

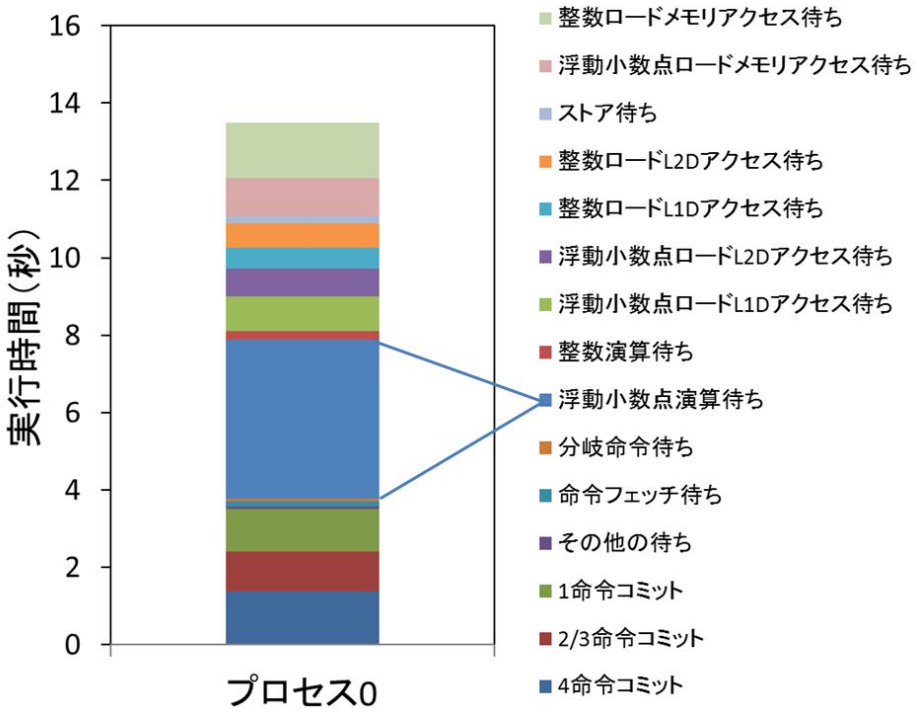

図9: 実行時間の内訳

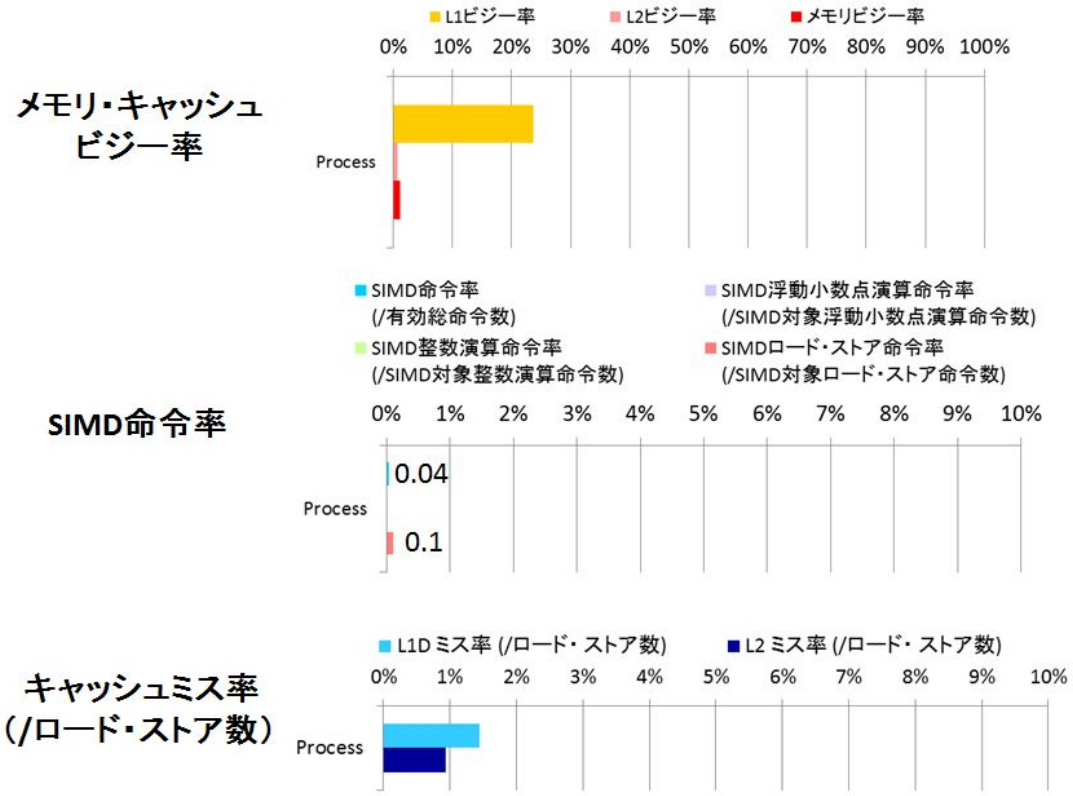

図10: メモリ・キャッシュ・SIMD 情報

```
void BO(…)
{
  .....
  for( i = 0; i < system->N; ++i ) {
    .....
    if (type_i < 0) continue;
    .....
    for( pj = start_i; pj < end_i; ++pj ) {
      .....
      if (type_j < 0) continue;
      .....
      if( i < j || workspace->bond_mark[j] > 3 ) {
        .....
        if( twbp->ovc < 0.001 && twbp->v13cor < 0.001 ) {
          .....
        }
        else {
          .....
          if( twbp->ovc >= 0.001 ) {
            ….. }
          else {
            .....
          }
          if( twbp->v13cor >= 0.001 ) {
            .....
          }
          else {
            .....
          }
          .....
        }
        .....
      }
      else {
        .....
      }
    }
  }
  .....
}
```

最内ループ

図11: ボンドオーダー計算部分のソースコードの構造

2.5.4.4 スケーラビリティ

最後にスケーラビリティを評価した結果を示す。スケーラビリティを評価することにより、ルーチン・関数毎の評価では見えにくい情報（非並列部分がどのくらいあるか？通信時間が大規模並列時にどのくらいの割合を占めるか？ロードインバランスが悪影響しないか？など）を入手することができる。スケーラビリティにはストロングスケーラビリティ（問題規模を一定にし、並列数を増やして評価する）とウィークスケーラビリティ（1 プロセッサ当たりの問題規模を一定にし、並列数を増やして評価する）がある。以下では、それぞれのスケーラビリティについて評価した結果を示す。

2.5.4.4.1 ストロングスケーラビリティ

ストロングスケーラビリティの評価ではこれまでと同様の3600 原子の系について、プロセス数を変化させて評価した。最初に実行時間のプロセス数依存性を示す

（図12）。図中の点線は理想的（実行時間がプロセス数に反比例する）な実行時間の変化を示す。プロセス数の増加と共に単調に実行時間が短縮しているが、プロセス数が大きいところで理想とのズレが大きい。このときの実行時間の内訳を図13に示す。プロセス数の増加と共に通信時間（Comm）の割合が増加している。

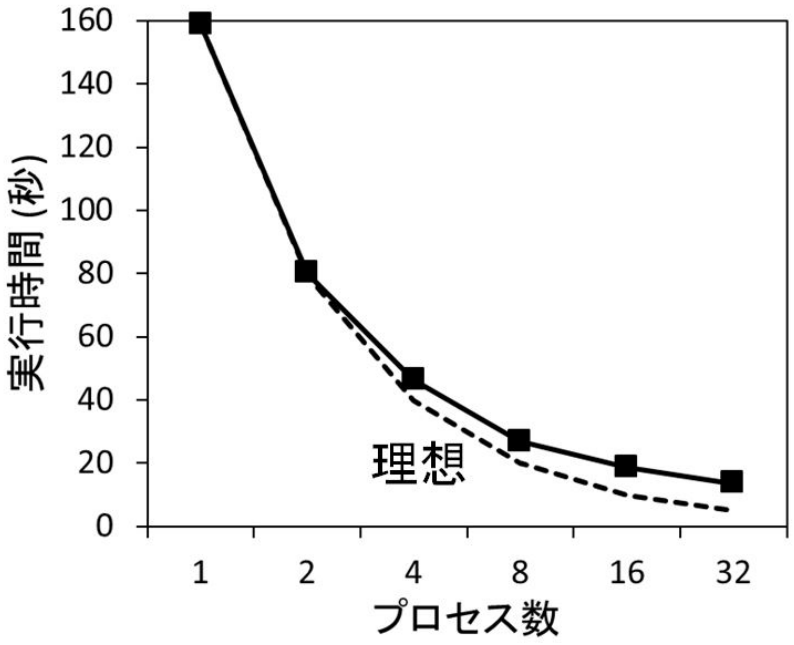

図12: 実行時間のプロセス数依存性

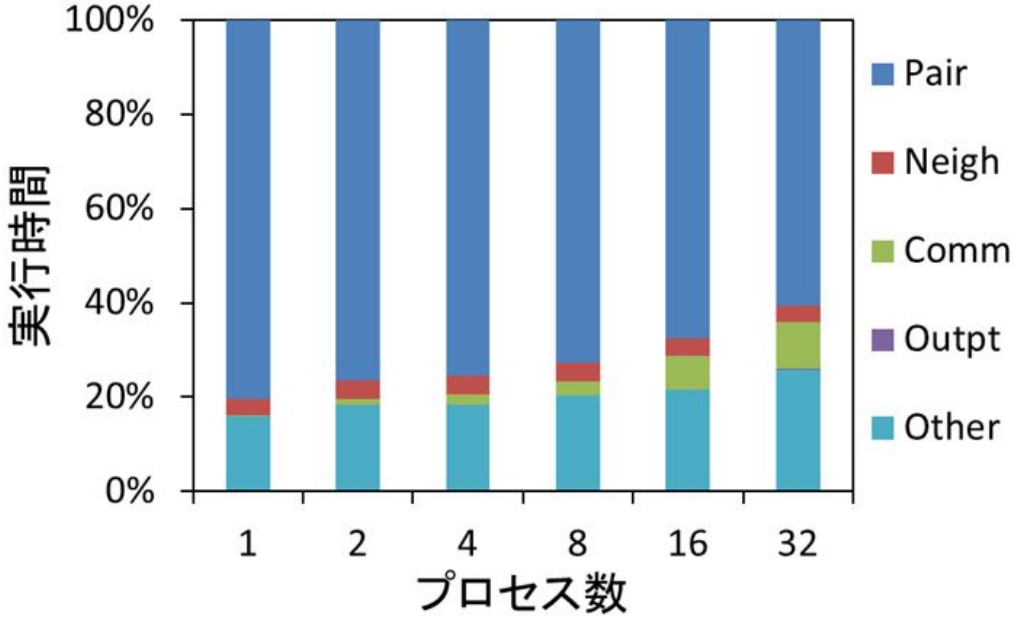

図13: 実行時間の内訳のプロセス数依存性

最後に並列化効率（使用したプロセッサの性能をどの位有効利用したか）を図14 に示す。16 並列時には並列化効率が50 %を下回っている。現在の問題規模（3600 原子）においてはこれ以上プロセス数を増やしても効率が悪い。

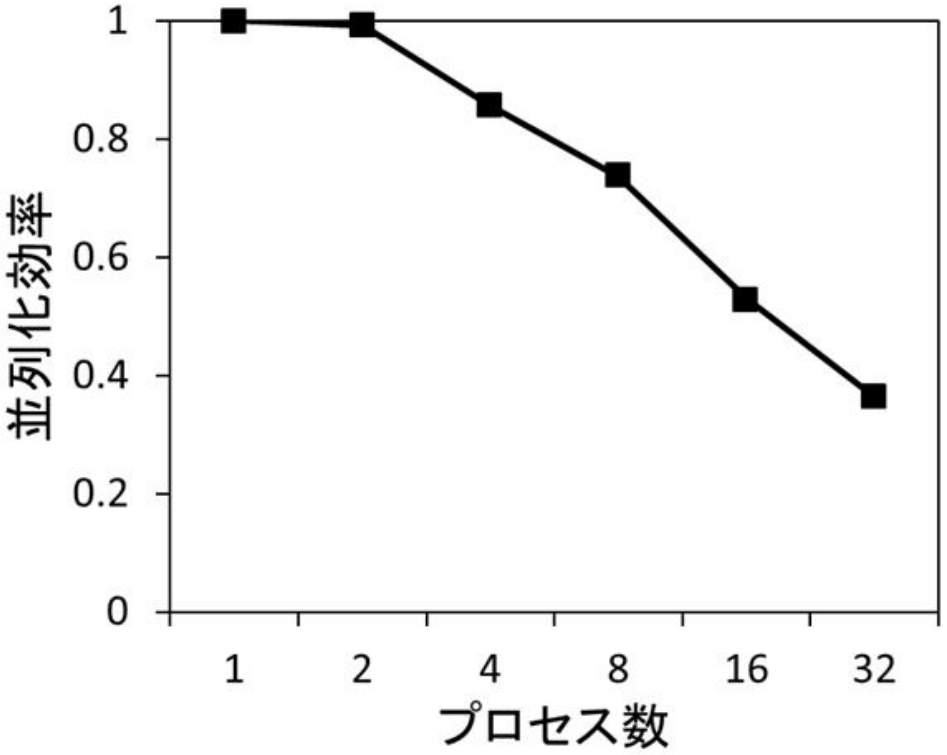

図14：並列化効率

2.5.4.4.2 ウィークスケーラビリティ

ウィークスケーラビリティの評価では450 原子の系（図15）を1 プロセッサ当たりの問題規模として、1 プロセッサ当たりの問題規模を一定にし、並列数を増やして評価した。実行時間のプロセス数依存性を図16 示す。図中の点線は実行時間の理想的な変化（実行時間一定）を示している。なだらかに実行時間が増加しているが、プロセス数が大きいところで変化が小さくなっており、問題規模を64 倍（28800 原子）にしても計算時間の増大は1.6 倍程度に収まっている。実行時間の内訳のプロセス数依存性を図17 示す。通信時間（Comm）の増加は僅かであり、大規模計算時にも通信時間はネックにならないと考えられる。一方、Pair、Other は僅かに上昇している。特にOther はプロセス数と共に単調に増加しており、非並列部が残っている可能性が高い。

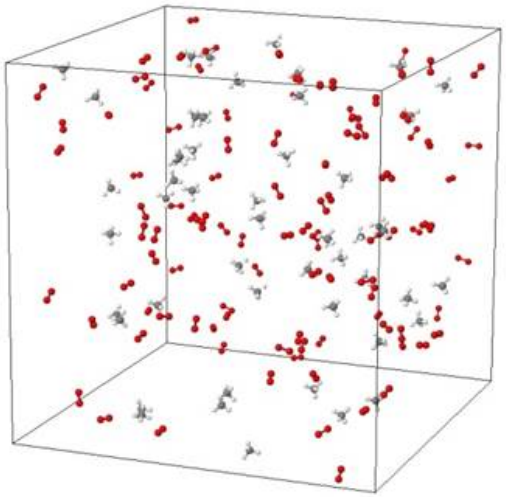

図15：1 プロセッサ当たりの問題規模

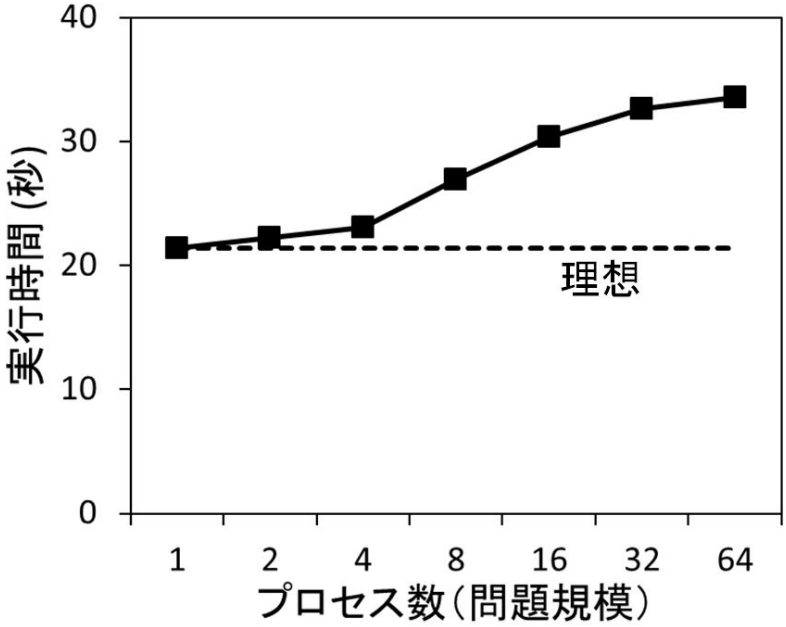

図16：実行時間のプロセス数（問題規模）依存性

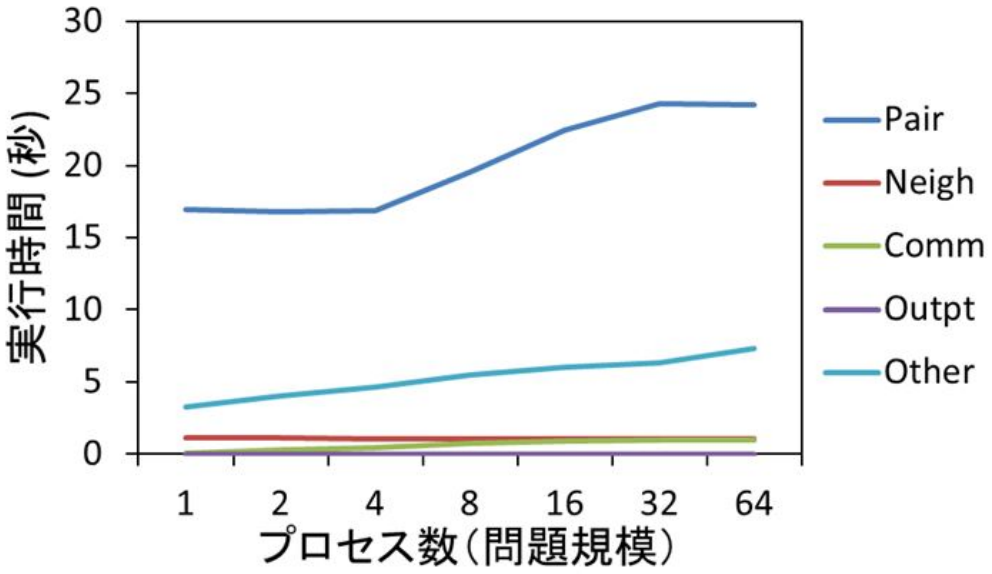

図17: 実行時間の内訳のプロセス数（問題規模）依存性

2.5.5 まとめ

LAMMPS/ReaxFF を対象としてFX100 での性能評価ならびにチューニングを実施した結果、以下の知見を得た。

・組込関数をインライン展開することで大幅に性能向上（計算速度約2 倍）

・LAMMPS のdynamic load-balancing 機能を用いることでロードインバランスが改善

・高コストループ内はコールや分岐が多く、SIMD 化が困難

・ウィークスケーラビリティは悪くなく（問題規模を64 倍にしても計算時間の増大は1.6倍程度）、大規模計算においても性能が出せる

参考文献

[1] S. Plimpton, “Fast Parallel Algorithms for Short-Range Molecular Dynamics”, Journal of Computational Physics, **117**, 1-19 (1995).

[2] A. C. Van Duin, S. Dasgupta, F. Lorant, and W. A. Goddard, “Reaxff: a reactive force field for hydrocarbons”, The Journal of Physical Chemistry A **105**, 9396 (2001).

[3] K. Chenoweth, A. C. van Duin, and W. A. Goddard, “Reaxff reactive force field for molecular dynamics simulations of hydrocarbon oxidation”, The Journal of Physical Chemistry A **112**, 1040 (2008).