[2-2] 短距離古典分子動力学法の超並列計算に向けて

東京大学物性研究所物質設計評価施設
渡辺 宙志 hwatanabe@issp.u-tokyo.ac.jp

概要

近年のスーパーコンピュータの計算能力の増加は主に CPU 内での SIMD 化，ノード内のコア数の増加，そしてノード数の増加による．これらの計算資源を使いこなすためには，多階層化されたシステムに対応した計算コードを開発する必要があり，プログラム開発者にとって大きな負担となっている．本稿では，短距離古典分子動力学法を題材に，1 万コアを超えるような超並列計算にどのように取り組めば良いかを紹介する．

キーワード：分子動力学法，超並列計算，高速化

1 はじめに

二期にわたり TOP500 の首位であった「京コンピュータ」は，その名の通り 10 ペタ (1 京)FLOPS の計算能力を誇る．その構成は，1 ノードに 8 コアの CPU を 8 万ノード以上束ねたものになっており，使うためには 8 万プロセス以上の超並列化が必須となる．さらに並列化だけではなく，SIMD 化，メモリ最適化など，超並列計算では考慮すべきことが多く，プログラム開発者の大きな負担となっている．本原稿ではサイエンスはさておき，筆者が大規模並列プログラムを組むにあたって苦労したこと，そこから分かった注意すべきことについてまとめてみたい．

2 並列化，その前に

大規模並列を行う理由は，解きたい問題サイズが大きすぎて一つのノードに収まりきらない場合と，並列化することで，より早く結果を得たい場合の二通りに分類されるが，多くの場合は後者であろう．その際，並列化を行う前にシングルノードでの性能が十分であるかどうかを確認するべきである．プログラムの組み方によっては，チューニングにより実行速度が数倍から数十倍速くなる場合もある．「シングルノードではこれ以上性能がでないことがわかっており，並列化必須である」と分かってから並列化を行うべきである．また「並列化」と言うと，既存のシリアル版のプログラムを改造して並列プログラムにするというイメージとなりがちだが，並列化に適したデータの保存方法とシリアル版に適したデータの保存方法は異なるため，よほど自明並列に近くない限り「改造による並列化」では 100 並列程度で性能が頭打ちになることが多い．超並列化向けのプログラムはゼロから何度も作り直す覚悟が必要となる．

3 メモリ最適化

プログラムの高速化において最も重要なのは，メモリまわりの最適化である．データはメモリに保存されているが，計算を行う為にはメモリからレジスタまでデータを持ってこなければならない．近年の計算機は，計算能力の発展に比して CPU －メモリ間のデータ転送能力が相対的に貧弱になっており，キャッシュをうまく使えないとそれだけで数倍から数十倍の計算速度劣化の原因になる．CPU の計算能力に対してどれだけデータ転送能力があるかを表す指標として，データ転送能力 (Byte/s) と計算能力 (FLOPS) の比を取った値がよく用いられる．これを Byte per Flops，通称 B/F 値と呼ぶ．B/F 値は計算機によって異なるが，典型的な値は 0.5 ～ 1 程度である．この値が数値計算にとってどのくらい厳しい値であるか，例を挙げて考えてみよう．A=B*C という計算を考える．この計算を実行するには，B と C という倍精度実数を二つ取ってきて，一度掛け算し，A に書き戻す必要がある．倍精度実数は，ひとつ 8Byte である．2 回の読み込み (load) と，1 回の書き込み (store) があるから，全体で 24Byte の load/store が必要となる．一方，計算は 1 度しかしていない．独立な乗算をひたすら繰り返すような計算でピーク性能を出すためには B/F 値は 24 以上でなければならない[1]．逆に B/F 値が 0.5 である計算機で A=B*C をただ繰り返すような計算をしようとすると，理論ピーク性能の 2%強しか使えず，残りの 98%の時間は CPU が遊んでしまうことになる．さらに，実際の計算ではメモリバンド幅だけではなく，レイテンシも重要である．メモリバンド幅とは，通信が始まった状態から通信が終わるまでにどれだけのデータを流すことができるかを示す値であり，レイテンシとは，CPU が必要なデータを要求してから実際に届くまでの時間 (サイクル数) であ

[1] ここでの議論ではレイテンシや演算器の数などは無視している．

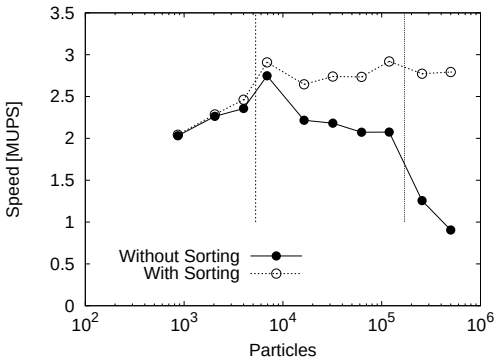

図 1: 性能の粒子数依存性．L2 と L3 キャッシュ容量に対応する粒子数を点線で示す．キャッシュをうまく使えていない場合 (黒丸) は，キャッシュからデータが溢れる度に性能が急激に劣化していくのが分かるが，キャッシュを有効に使えている場合 (白丸) では性能の粒子数依存性がほとんどなく，キャッシュが有効に使えていることがわかる．

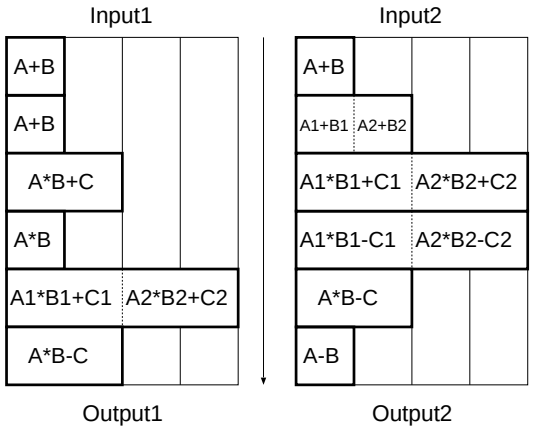

図 2: SIMD の実行イメージ．幅が 4 のベルトコンベアが 2 ラインあり，それぞれに幅 1 から 4 の荷物 (命令) を一つずつ流す事ができる．全て幅 4 の荷物でコンベアを埋めた時がピーク性能である．また，荷物を乗せてから製品として出てくるまでの時間がレイテンシに対応する．

る．システムによってはレイテンシが数百サイクル以上かかる場合もあり，広大なメモリ空間をランダムにアクセスするようなプログラムでは計算時間がほとんどメモリレイテンシ，ということになりかねない．

自分のコードがどれだけキャッシュを効率的に使えているかを知るのにはプロファイラなどを利用する方法もあるが，粒子数 (自由度) を増やした時の粒子数あたりの性能の変化を見るのが簡単である．図 1 は分子動力学法において粒子数を増やしていったときのプログラムの実行速度の変化である．百万粒子を 1 秒に一度更新できたら 1 となる MUPS (Million Update Per Second) という単位を使っている．キャッシュを効率的に使えていない場合は，データサイズがキャッシュから溢れたところで急激に性能が劣化するが，キャッシュを効率的に使えている場合は，性能がほとんど劣化しない．これは，利用すべきデータがだいたいキャッシュにのっていることを示す．キャッシュ効率を上げる方法に一般論はないが，空間的に近い粒子を，メモリ上でも近くなるように再配置したり，同じ粒子と相互作用する粒子をまとめるなど，粒子系ではソートが有効である場合が多い．詳細は文献 [1] にまとめたので参照されたい．

4 CPU チューニング

メモリまわりの最適化はある程度汎用性があるのに比べ，CPU チューニングは利用アーキテクチャに強く依存しており，また苦労の割には加速率は高くないことが多いので，一般論としてはメモリ関連の最適化が済んだらプロダクトランの準備を行い，研究を進めながら片手間でチューニングも行う，といった程度で良い．CPU チューニングは多岐にわたるが，ここでは SIMD 化と条件分岐削除について説明する．

4.1 SIMD 化

京コンピュータの CPU コアは動作クロックが 2GHz, コアひとつあたりの性能が 16GFLOPS である．1 サイクルで発行できる演算命令は 2 つだけであるため，1 つの命令で 4 つの浮動小数点演算ができる計算となる．この，一つの命令で複数の演算を実行する仕組みが SIMD (Single Instruction Multiple Data) である．近年の数値計算用に利用される CPU アーキテクチャのほとんどは SIMD 機能を備えているため，性能を引き出すためにはプログラムの SIMD 化は必須技術となっている．京コンピュータや IBM POWER アーキテクチャでは，積和演算を行う命令がある．積和命令とは $A \times B + C$ のような計算を一度に行うような命令で，この命令一つで二つの浮動小数点演算と数える．さらに，京コンピュータでは，独立な積和演算を二つ同時に計算することで，1 命令で 4 つの浮動小数点演算ができる仕組みとなっている．この SIMD の実行イメージを図 2 に示す．

京コンピュータでは毎サイクル 4 つの独立な積和演算を実行して初めてピーク性能がでる．逆に，もし 1 サイクルで一つの浮動小数点演算 (乗算，加減算) しか実行できなければ，ピーク性能の 8 分の 1 しか出せない事になる．コードの性能が低い主原因が，SIMD 化されていないスカラ命令が多数発行されていることに依るものとわかった場合，性能向上のためにはプログラム

の SIMD 化を進める必要がある．しかし経験的に，コンパイラが自動で SIMD 化できなかった場合に手動で SIMD 化することは極めて難しい．そこで，なるべくコンパイラが SIMD 化しやすいコードを書く必要がある．例えばループのインデックス間で依存性のないループなどは SIMD 化しやすいため，依存性があるループを多少のオーバーヘッドはあっても依存性のないループに書き換えるなどの修正は有効であることが多い．

4.2 条件分岐削除

一般に，プログラムの開発は Intel 系の CPU を搭載した PC で行うことが多いであろう．PC ではそこそこ性能が出ているのに，京コンピュータや，IBM POWER などの CPU を搭載したスパコンに持って行くと性能がでない，ということはよくあるが，その原因は条件分岐に依る場合が多い．分子動力学計算ではカットオフを設けることが多いため，粒子間距離を計算し，カットオフ距離以上であればループの次へジャンプ (continue) させることが多いが，この条件分岐によるジャンプは Intel 系以外のアーキテクチャでは性能劣化を招きやすい．簡単な対処法は，粒子間距離がカットオフ以上の時に計算した力をゼロにクリアし，大きさゼロの力積を足す方法に変更することである．計算量には無駄が生じるが，ループがジャンプフリーになることで様々な最適化が効くようになり，場合によっては数倍以上高速化される場合もある．

5 並列化で起きる問題

シングルノードや研究室のクラスタでのテストも終わり，いざ大規模計算，となったときに初めて出会う，大規模計算特有の問題がいくつか存在する．これらは大規模計算で初めて現れるため，知らなければ事前の対策が難しい．以下は筆者が経験した問題である．

5.1 Flat-MPI か，ハイブリッドか

超並列計算で知っておくべきことは，MPI プロセスは OpenMP スレッドに比べてメモリ利用量が大きいことである．まったく同じ規模の計算を行った場合でも，flat-MPI(全て MPI プロセスによる並列化) の場合とハイブリッド (MPI+OpenMP による並列化) ではメモリの使用量が大きく異なる．東大物性研の SGI Altix ICE 8400EX の 1024 コアにおいて，flat MPI (1024 プロセス) とハイブリッド (128 プロセス× 8 スレッド) の計算を行った結果を表に示す．Flat-MPI の方が実行速度は早いが，69TB だけ余計にメモリを使っていることがわかる．詳細は実装に依存するが，一般に並列数が大きくなればなるほど MPI の「プロセスあたり」のメモリ使用量が大きくなる傾向にある．従って，100 並列では大丈夫だった計算が，ウィークスケーリング (ノードあたりの計算規模を固定してノード数を増やす方法) であるにもかかわらず 1000 並列ではメモリ不足で失敗する，ということがおきる．その場合は環境変数を修正することで MPI の利用メモリを減らしたり，ハイブリッド化により MPI のプロセス数を減らすなどの工夫が必要となる．

並列化手法	実行時間 [s]	利用メモリ [TB]
Flat-MPI	249.36	268
Hybrid	257.09	199

表 1: Flat-MPI 並列 (1024 プロセス) と Hybrid 並列 (128 プロセス× 8 スレッド) の性能比較．カットオフ付き Lennard-Jones ポテンシャルの計算．直径を 1 として 800 × 800 × 1600，密度 0.5 で 5 億 1 千万粒子を 1000 ステップ計算するのにかかった時間と利用実メモリ量．

また，MPI の利用でよくおきる問題として資源の枯渇がある．通信時間の隠蔽のため，MPI_Isend などのノンブロッキング通信を使うことも多いだろう．しかし，一般にノンブロッキング通信はブロッキング通信よりも多くのメモリ，多くの種類のバッファを使うため，大規模並列時に問題を起こしやすい．例えば小規模並列では問題なかったプログラムを大規模並列実行した時に「MPI_REQUEST_MAX が足りない」といったエラーメッセージとともにジョブが失敗することがある．MPI_REQUEST_MAX は同時に行うことができるノンブロッキング通信の最大数であり，これが不足する場合には，通信を小分けにするか，一部をブロッキング通信に置き換えるなどの工夫が必要となる．また，REQUEST_MAX 以外にも「MPI_○○○_MAX が足りない」というタイプのエラーはよく出現する．その際は対応する環境変数を上げる必要があるが，その背景にはメモリ不足が疑われるため，ノードあたりのプロセス数を減らしたり，問題サイズを小さくしたりする必要がある．

5.2 システムノイズの影響

並列化効率を決める指標として，計算に対する通信の頻度 (粒度：granularity) がある．一般に計算時間に対して通信頻度と通信時間が小さければ，大規模並列時でも並列化効率は劣化しないはずである．しかし，意味のある並列計算ではあるステップごとに同期をとる必要があり，この同期によって並列化効率が下がる場合がある．それが OS ジッタと呼ばれるシステムノイズである．システムには計算プログラムの他，OS (Operating System) の様々なプロセスが実行されている．これら

のシステムプロセスがたまに数値計算に割り込んでくるが，これが実効的な負荷バランスを悪くすることがある．図 3 に概念図を示す．プログラム上は負荷バランスがほぼ均等であり，かつ単位時間あたりのシステムプロセスの割り込みも各プロセスで同じであったとしても，その割り込みと同期のタイミングによっては見かけ上のロードバランスを悪化させ，実行効率を下げる方向に働く．この問題はユーザ側ではどうしようもないため，システム側で対処してもらうしかない．

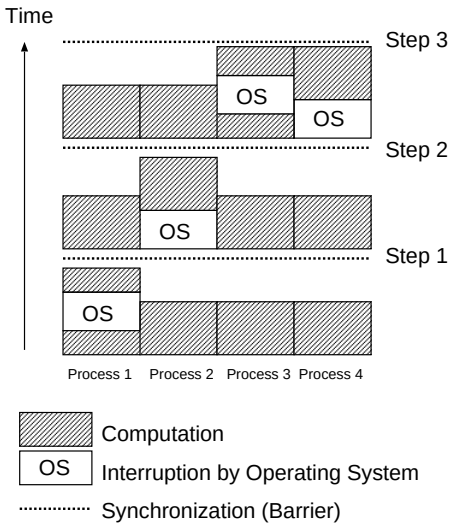

図 3: システムノイズの模式図．各プロセスのステップごとの計算負荷 (網掛け部分) はほぼ一定であるにもかかわらず，ランダムに割り込むシステムノイズにより見かけ上負荷バランスが悪くなる．

6 プロダクトランに向けて

シングルノードでのチューニングも十分に済み，大規模並列でも満足できる性能が出たとして，それでもまだ科学論文を書く事はできない．分子動力学法であれば，温度制御や圧力制御，境界条件なども正しく並列化されていなければならない．さらに重要なのは観測・解析ルーチンである．大規模計算を実行中，その全ての情報をディスクに出力していくのは現実的ではないため，一般には物理量を粗視化しながら出力する操作が必要になる．例えば気泡検出は，もっとも単純には空間を小さく分割して局所密度がある閾値以下であれば気相と判断するなどの手法を取るが，気泡サイズ分布を得るためには，クラスタリングの並列化を行う必要がある．粗視化した密度分布だけ出力しておいてクラスタリングをポスト処理で行うべきか，クラスタリングまで実行中に行ってしまうべきかは場合による．もし並列化を試みるならば，完成した並列化プログラムをプロダクトラン用に改良する作業量は，並列化プログラムを作るのと同じくらいかかると見ておいた方が良い．

7 終わりに

以上，駆け足で大規模計算機に向けたプログラムのチューニングや，並列化において注意すべき点を述べた．経験的に，並列数が一桁増えるたびに質的に異なる困難が待ち受けている．現在，最下層では SIMD 化，ノード内ではスレッド並列，ノード間ではプロセス並列と，異なる複数の並列パラダイムを扱わなければならない．またシングルノードだけ見ても，メモリが L1, L2, L3 キャッシュにメインメモリと多階層になっており，現在でも開発の負担は大きいが，その負担は今後増えることはあっても減ることはないと思われる．その先にあるのは「プログラムを組む人」と「論文を書く人」の分業化であり，既に一部の分野では分業が進んでいるように思われる．ある種の分業は不可避であることは事実であるが，その一方でプログラム開発から論文を書くところまで同じ人がやる，ということは大事にしたいものである．本稿，及び公開しているコード [2] が，そのような「ゼロから大規模並列コードを書いて計算し，論文を書く人」の助けになれば幸いである．

謝辞 本稿で紹介した内容は，理化学研究所計算科学研究機構の京速コンピュータ「京」の試験利用，東京大学情報基盤センター PRIMEHPC FX10，及び東京大学物性研究所のシステム B (SGI Altix ICE 8400EX) を使わせていただいた結果を含みます．

参考文献

[1] H. Watanabe, M. Suzuki, and N. Ito, Prog. Theor. Phys. **126** 203–235 (2011).

[2] http://mdacp.sourceforge.net/

著者紹介

渡辺宙志氏 (博士 (工学))：〔経歴〕2004 年東京大学工学研究科博士課程修了，同年名古屋大学大学院情報科学研究科，東京大学情報基盤センターを経て，現在東京大学物性研究所.〔専門〕分子動力学計算，統計力学.〔趣味〕プログラム.