HA8000における並列分子動力学法コードの性能

渡辺宙志

東京大学情報基盤センター

伊藤伸泰

東京大学工学系研究科

1. はじめに

我々は気液混相流の全粒子計算を目指し、大規模な計算な可能が分子動力学法コードの開発、及びそれを用いた研究を行っている。気液混相流の振る舞いの理解は、軽水炉の安全確保やガソリンエンジンの効率向上などの工学応用上、非常に重要であるが、熱伝導や流動などの輸送現象に加え、沸騰という相転移も複雑にからみあう強い非平衡現象であることから理論的な解析が難しい。数値計算課題として見た気液混相流は、相転移というミクロな挙動と輸送・流動というマクロな挙動が同居する典型的なマルチスケール問題であり、さらに相転移による大きな密度変化、相界面の移動および生成消滅などが計算を難しくする。これまでこのようなマルチスケールな問題については、現象をそのスケールにより分類し、計算するスケールよりも小さいスケールについては粗視化された構成式を用いることで研究が行われてきた。粗視化に用いるパラメータは実験と比較することで決められるが、気泡と流体の間の摩擦など実験で決定することは難しいパラメータも存在し、それらは経験的な扱いがなされている。それに対し、流体の構成分子を全て陽に計算する全粒子計算が注目されている。構成粒子のミクロな振る舞いは、粒子間のポテンシャルを与えることにより決まるニュートン方程式に支配されている。このニュートン方程式を数値積分することで系の時間発展させ、その振る舞いを調べる手法が分子動力学法 (Molecular Dynamics method, MD) である。気液混相流の全粒子計算とは、流体を構成する粒子を分子動力学法により計算することで、マルチスケールな現象をミクロな相互作用から直接再現しようとする試みであり、これまで人為的に導入されてきた粗視化階層の妥当性や、各階層における構成方程式の検討、そしてミクロな輸送現象がマクロにどのようにつながっていくかをシームレスに研究することを目指している。マクロな系をミクロから計算するためには必然的に計算は大規模なものとなり、本質的に並列計算が必須となる。いま開発がすすめられている次世代スーパーコンピュータをはじめ、今後の大規模計算資源は超並列スカラ型 (Massive parallel processing, MPP) が主流になると考えられる。分子動力学法は通信に比べて計算が重いという特徴を持つために、比較的 MPP 型の計算機に向いていると思われるが、並列度が増えると既存のアルゴリズムでは高い計算効率を得られない可能性がある。その場合には大規模並列化に向けて新たなアルゴリズムを開発しなければならない。そこで、まず単純な並列化を実装した分子動力学法コードについて、大規模な並列計算においてどのような困難があり、今後どのようにコード開発を行うべきかについての知見を得ることを目的とし、東京大学 HA8000 システム 512 ノード、8192 プロセスにて動作させた場合の性能を測定した結果について報告する。

本稿では、まず分子動力学法の並列化アルゴリズムについて説明する。粒子の相互作用が系のサイズに比べて十分短い場合には、並列化は領域分割を用いるのが効率的である。単純な領域分割であればさほど複雑なアルゴリズムは要求されないが、シリアル版で採用している高速

化アルゴリズムが並列化に影響するので、まず 2 章にてシリアル版でのアルゴリズムを概観し、その後 3 章にて並列化の方針を説明する。4 章に計測結果について報告し、考察と今後の展望を 5 章にまとめる。

2. シリアル版のアルゴリズム

2.1. ポテンシャル形状

粒子間ポテンシャルには、6-12 型の Lennard-Jones ポテンシャルを用いる。Lennard-Jones ポテンシャルはもともと希ガスの分子間力の近似式として提案されたものであり、粒子が持つ「近距離で斥力、遠方で引力」という性質をモデル化したものである。気体・液体・固体の三相を表現することができ、計算量も比較的小さいためにモデル粒子として広く用いられている。Lennard-Jones 型のポテンシャルは遠方で急速に 0 に近づくため、すべての粒子対について力を計算するのは非効率的である。そこで適当な距離 r_c でカットオフを入れることで計算効率を上げる。具体的には二粒子間距離を r として以下の関数形を用いる [2]。

$$\phi(r) = 4\varepsilon \left(\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} + c_2 \left(\frac{r}{r_c}\right)^{2} + c_0 \right) \tag{1}$$

ただし、ε はポテンシャル深さを、σ は粒子の半径を表し、それぞれエネルギーと長さの次元を持っている。係数 c_2、c_0 は、カットオフ距離において力とポテンシャルの双方がゼロになる条件から決める。なお、カットオフのために一次ではなく二次の項を付け加えるのは、粒子間距離の二乗のみを用いて力の計算を行うためである。

2.2. 相互作用粒子探索

カットオフを入れた場合、力の計算をする前に相互作用している粒子のペアのリストを構築する必要がある。粒子のペアを直接探すと計算量が $O(N^2)$ となるため、メッシュ探索を用いることで $O(N)$ に落とす [3]。メッシュ探索とは、全系をある長さの格子に区切り、その格子に粒子を登録することで近所の粒子を探す方法であり、大きく分けて**排他的格子 (Exclusive Mesh)** と、**非排他的格子 (Non-exclusive Mesh)** の二種類がある。排他的格子は、一つの格子にはたかだか一つの粒子しか入らないようにする方法であり、純粋に格子の数だけ配列を用意して、もし粒子がその格子に存在していたら粒子番号を登録する。非排他的格子は、相互作用距離程度の格子を作成し、一度に複数の粒子が入ることを許す方法である。一般に剛体粒子のような相互作用距離が短いケースでは排他的格子が、Lennard-Jones ポテンシャルのように相互作用距離が比較的長いケースでは非排他的格子の方が効率が良いことが多い。しかし、相互作用距離や次元、密度、さらにキャッシュやメモリバンド幅にも依存するため、実際にはどちらが良いかは実装してみないとわからないことも多い。

2.3. ペアリストの有効期限判定

粒子数が多くなると、相互作用ペアリストの構築にかかる時間も長くなる。そこで、相互作用をしている粒子を探す範囲を長くして、ペアリストが有効な時間内はリストを再構築しないことで高速化を図る [1]。粒子の検索距離を r_s、相互作用距離 (カットオフ長) を r_c とすると、その差 $r_s - r_c$ を粒子が横切る時間の間はペアリストは有効、すなわちペアリストに登録されていない粒子対が相互作用する可能性はない。したがって、ペアリストを構築した時の時刻を t、系の中で最速の粒子の速度を v_{max} とすると、有効期限 t_{e} は

$$t_{\mathrm{e}} = \frac{r_s - r_c}{2v_{max}} + t \tag{2}$$

で与えられる。v_{max} ではなく $2v_{max}$ で割るのは同じような速度を持つ粒子が近づいていくような状況を考慮に入れているからである。有効期限内はペアリストを再構築する必要はないが、系

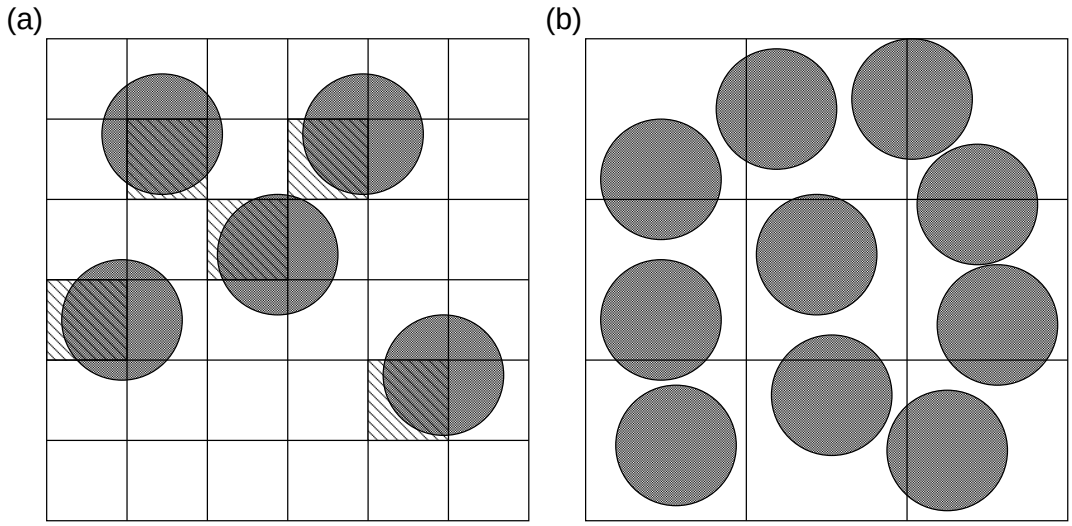

図 1: グリッドによる隣接粒子探索。(a) 排他的格子。一つの格子には高々一つの粒子のみが含まれるように格子長さを設定する。(b) 非排他的格子。一つの格子に複数の粒子が含まれる場合がある。探索距離を元に格子長さを設定する。

の最高速度 v_{max} が変わるたびに有効期限を修正する必要がある。これまでの最高速度を $v_{\rm old}$、新しい最高速度を $v_{\rm new}$ とすると、新しい有効期限は

$$t_{\rm e} = (t_{\rm e} - t)\frac{v_{\rm old}}{v_{\rm new}} + t \tag{3}$$

で表される。容易にわかるように、系の最高速度がこれまでよりも早くなれば有効時間は短くなり、遅くなれば長くなる。また、検索距離 r_s を長くするほど相互作用ペアリストの有効期限が長くなるが、それだけ登録する粒子が多くなり、力の計算において実際には相互作用をしていない粒子間の距離を計算することが多くなる。ペアリストの有効期限は検索範囲 r_s に比例し、検索にかかる時間は r_s^3 に比例するため、r_s には最適な長さが存在する。

なお、リストを作ったときの粒子の位置を記録しておき、粒子の変位が有効範囲を超えないという条件で有効期限を決める方法もある。密度がある程度高い場合、粒子は頻繁に衝突し、速度に比して変位はさほど増えないため、位置によって有効期限を決めたほうが効率的である。ただし、最高速度のチェックに比べて粒子の変位のチェックは計算量が大きいため、両方を組み合わせて使ったほうが効率が良い。

3. 並列化

3.1. 並列化の方針

並列化は実装が容易な単純領域分割を採用する。計算領域をプロセスの数と同数の単純立方格子に分割し、各プロセスは自分に割り当てられた空間領域に存在する粒子の計算を担当する。通信は、「境界からはみ出した粒子の交換」「境界領域の粒子の情報」「ペアリストの有効期間確認」の三種類が存在する。

3.2. 境界外粒子の交換

各プロセスは、自分が担当する領域から外れた粒子を、適切なプロセスに受け渡す必要がある。あまり大きくはみ出さなければ問題ないことが多いため、毎ステップ受け渡すことはせず、後述するペアリストの更新時に同時に行う。

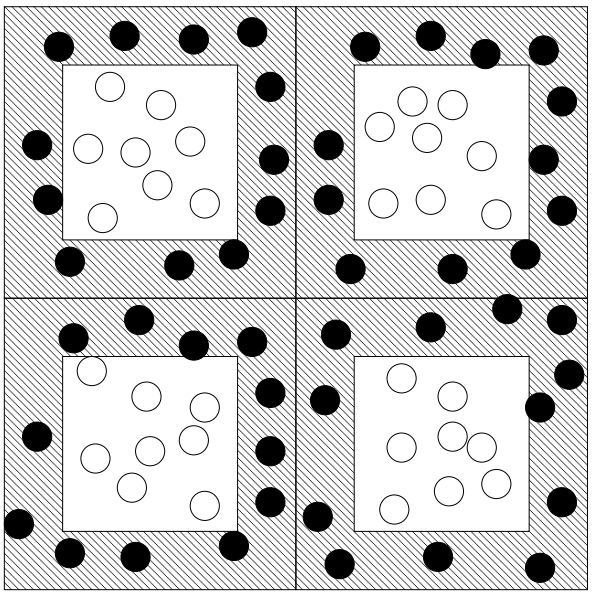

図 2: 領域分割の概念図。相互作用距離で決まる境界領域内 (図の斜線部分) にいる粒子は隣接領域の粒子と相互作用する可能性があるため、毎ステップ情報を送る必要がある。

3.3. 境界領域の粒子情報

各プロセスが担当する粒子に働く力を計算するためには、隣接する領域の粒子の位置情報が必要となる。送るべき粒子の量は、検索距離 r_s により決まる。一つのプロセスが担当する領域を大きく取ることで全体の計算に占める通信時間を減らすことができるが、実際にはメモリ容量や計算時間の兼ね合いからあまり一つのプロセスに大きな領域を担当させることができない。Lennard-Jones 粒子の場合、およそ 1 プロセスに 100 万粒子程度までが目安となる。これは一辺の長さが $L = 100\sigma$ 程度の立方体領域に対応する。検索範囲 r_s はカットオフ距離にもよるが、だいたい $r_s = 3.0\sigma$ 程度に取る。すると、境界に存在する粒子の全体に占める割合は

$$1 - \left(\frac{L - 2.0r_s}{L}\right)^3 \sim 0.169 \tag{4}$$

すなわち、およそ 17%となる。密度を ρ とすると、一つの方向、たとえば右に位置情報を送るべき粒子の数は $L^2 r_s \rho$ 個となる。数密度を 0.5、つまり領域あたり 50 万粒子を担当させたとすると、情報を送るべき粒子数は 1 万 5 千個となり、一つの粒子あたり 24 バイト (8 バイト × 3 次元) であるから片方向に 360KB を送信する必要がある。これを 6 方向行う必要があるため、プロセスあたり片道 2.16MB を毎ステップ送信し、同量を受信する必要がある。他の通信量が少ない手法、たとえば差分の階数が小さい格子法などでは通信レイテンシがボトルネックとなることが多いが、相互作用距離を持つ粒子系の場合には通信のバンド幅もボトルネックとなりうる。

3.4. ペアリストの有効期間確認

各プロセスは、相互作用をする粒子のペアのリストを独立に保持、管理している。リストが有効な期間内は同じリストを使いまわすが、有効でなくなったら再構築する必要がある。このとき、各プロセスが独立に有効期限を管理し、勝手に再構築を行うことにすると無駄な待ち時間が生じる。粒子に働く力を計算するためには最新の粒子の位置情報が必要となるが、ペアリストの構築が終わるまで他のプロセスからの Send 要求に答えることができないため、別のプロセスはその再構築が終わるまで待たなくてはいけないからである (図 3 参照)。そこで、各プ

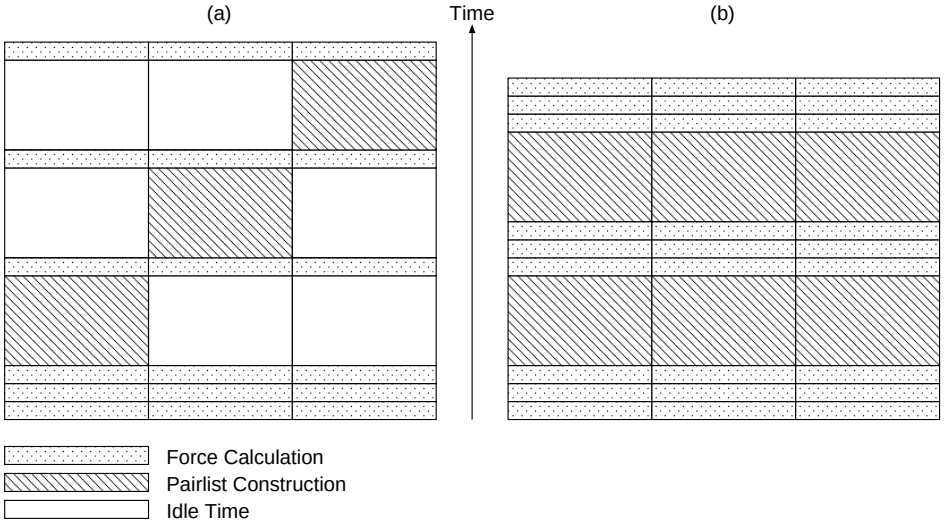

図 3: ペアリストの同期。プロセス 1,2,3 が、それぞれ 3,4,5 ステップ目にペアリストの有効期限がきれる場合。(a) 各プロセスが独立にペアリストを管理していると、ペアリストの再構築時に別のプロセスに無駄な待ち時間が発生する。(b) ペアリストの有効期限の同期を取ると、無駄な待ち時間が減る。

ロセスはペアリストの有効期限について同期をとることでこの無駄な待ち時間をなくす。すなわち、ある時刻に一つでもペアリストの再構築が必要なプロセスがあったら、全体でペアリストの再構築を行うことにする。すると、全体としてペアリストの平均寿命は短くなるが、無駄な待ち時間がなくなり、結果として計算が高速となる。ここで、他の通信が二つのプロセス間の通信であるのに対し、このペアリストの有効期限確認は全体通信であることに注意しておきたい (MPI_Allreduce にて実装されている)。物理的に隣接するプロセス間の通信は並列数とは関係ないはずであるが、全体通信は並列数に強く依存する。なお、送る情報は整数一つ (4 バイト) であるため、実際に問題となるのはレイテンシのみである。

4. 結果

以上のアルゴリズムを MPI を用いて実装した分子動力学法コードについて、その並列化効率を HA8000 にて測定した。各プロセスは一辺 100σ の立方体の領域を担当し、プロセスあたりの粒子数を 50 万個に固定したままプロセス数のみを増やす、いわゆるウィークスケーリングを行った。計算領域はなるべく立方体に近い直方体とし、全方向について周期境界条件を課した。時間刻みは 0.001 とし、二次のシンプレクティック積分法により時間積分を行った。カットオフ距離を 2.5σ、探索範囲を 3.0σ としたため、ペアリストの有効期限を決める余剰長さは 0.5σ である。最初の 150 ステップの緩和の後の 1000 ステップの計算にかかった時間から計算速度 (MUPS) を求めた。MUPS は Millions Update per second(百万粒子更新毎秒) の略であり、100 万粒子の系を 1 秒間で一度数値積分できたら 1 となる単位である。1 ノードあたり 16 プロセス、最大 512 ノード 8192 プロセスを用いて約 41 億粒子の計算を行った。測定結果を表 1 にまとめる。粒子数と速度の関係を図 4 に、1 ノード占有の場合を基準とした並列化効率を図 5 に示す。8 ノードから 64 ノードまではおおむね 85%前後の並列化効率であるが、256、512 ノードではそれぞれ 82%、76%と効率が若干低下している。1 ノードに比較して 2,4,8 ノードと単調に効率が下がるのは、ノード間にまたがる通信が 2 ノードでは x 方向のみ、4 ノードでは x,y 方向、8 ノード以上では x,y,z の全方向と増えることに対応する。8 ノードから 64 ノードまであまり並列化効率が変わらないのは、基本的に律速がノード間通信のバンド幅であり、ノード数が増えても通信時間が増えないためと考えられる。また、256 ノードや 512 ノードで効率が下がるの

は、少ないノード数では無視できていた全体の同期の時間コストが影響しているのではないかと思われるが、まだ詳細は不明である。なお、512 ノードにおいてまったく同じ計算を三回繰り返したところ、690 秒、727 秒、710 秒と、相対誤差にしておよそ 2%程度のゆらぎがあった。

ノード数	粒子数	実行時間 [SEC]	速度 [MUPS]	並列化効率
1	7913588	522.303	15.1513	1.00
2	15877262	556.774	28.5165	0.94
4	31855013	599.074	53.1738	0.87
8	63710026	608.586	104.685	0.86
16	127420052	618.909	205.878	0.84
32	254840104	623.032	409.032	0.84
64	510082164	606.344	841.243	0.86
256	2043548109	637.908	3203.51	0.82
512	4088706579	689.990	5925.75	0.76

表 1: 測定結果。1 ノードあたり 16 プロセス (コア数と同数)。150 ステップの緩和の後、1000 ステップの計算にかかった実行時間を示してある。並列化効率は 1 ノード占有計算を基準としている。

前述したように、プロセス間はステップ毎に 1 方向につき 320KB の情報を送受信する必要がある。ノード内 16 プロセスであるため、全方向にノード間通信を行ったとしても、通信量は全体で 16*6*320KB=34.56MB である。HA8000 はノード間 5GB/s の双方向同時通信が可能であるので、34.56MB の転送にかかる時間は 7msec 程度となる。今回のベンチマークでは 1 ステップに 0.5 秒ほど時間がかかる計算を行っているため、ノード間通信はほぼ無視できるはずであるにもかかわらずノードが増えるにつれて有意に計算速度が遅くなっている。そこで、通信量を減らした場合の影響を見るため、ノードあたりのスレッド数を 4 とした場合の計算も行った (表 2)。すると、256 ノード、512 ノードとも、ノード間通信を伴わない 1 ノード占有の計算よりも実行時間が短くなった。HA8000 は、1 ノードに AMD Quad Core Opteron プロセッサを 4 個（合計 16 コア）搭載した構成となっており、ノードあたりのスレッド数を減らすことで、ノード間通信のみならず、コアとメモリ間の転送量も減り、結果として計算時間が短くなったものと考えられる。これは、ノード間通信以外に、コアとメモリの間の情報転送がボトルネックとなっていることを示唆する。ノード間通信とコア－メモリ間通信のどちらにどれだけの時間がかかっているかはこれだけではわからないため、今後検討する必要がある。

ノード数	N_p	粒子数	実行時間 [SEC]	速度 [MUPS]
256	4	2043548109	637.908	3203.51
256	16	510082164	493.409	1033.79
512	4	1020968874	507.382	2012.23
512	16	4088706579	689.990	5925.75

表 2: ノードあたりのプロセス数の影響。N_p はノードあたりのプロセス数。

5. まとめ

分子動力学法による並列コードを実装し、HA8000 にて並列化効率の測定を行った。最大 512 ノード 8192 プロセスにて 41 億粒子の計算を行い、1 ノード占有計算と比較した並列化効率は 76%であった。ノード数を増やしたときの実行時間の増加は、通信量と通信バンド幅から想定

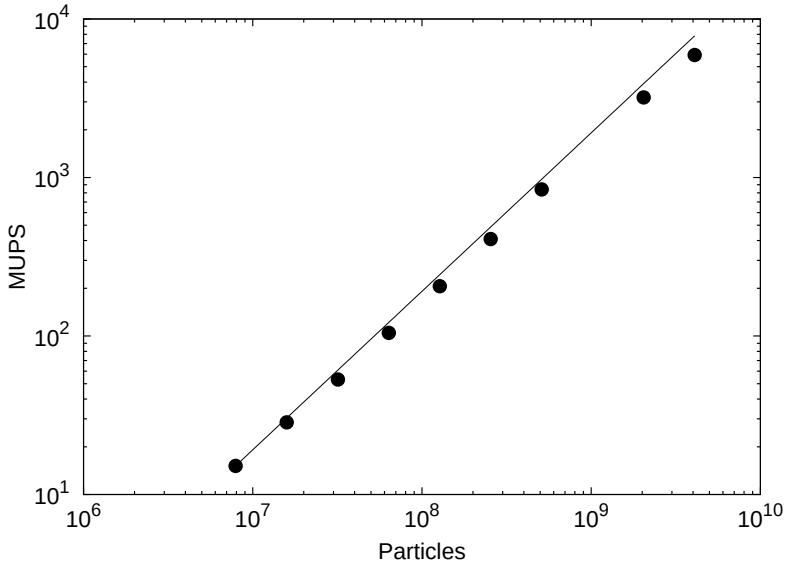

図 4: 粒子数と速度の関係。1 ノードに約 800 万粒子、512 ノードで約 41 億粒子まで計算している。実線が 1 ノードを基準とした理想的な速度。

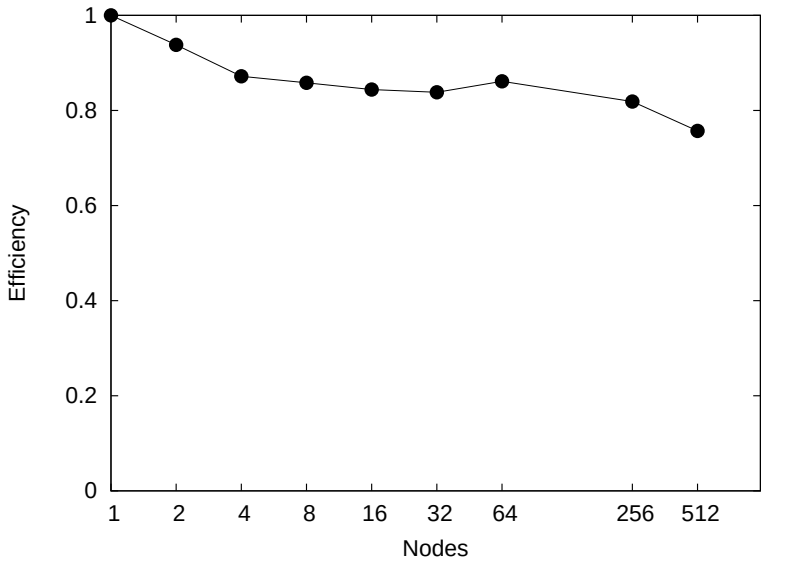

図 5: 並列化効率。1 ノードに比べて 64 ノードでおよそ 86%、512 ノードで 76%の効率であった。

される通信時間では説明できないほど大きく、その原因究明は今後の課題である。また、キャッシュ効率が悪いことが示唆されたため、並列化効率のみならず単体コードのさらなるチューニングも行う必要があることがわかった。次世代スーパーコンピュータに代表されるように、今後の計算の大規模化はコア数の増大によって達成されていくと考えられる。今回の計測に用いたコードは、1 万プロセス程度では実用的な速度が出ることが分かったが、10 万プロセスを超えるような超並列計算機においても実用的な計算を実行するためには、より高い並列化効率を目指すことが必須となろう。そのためには、ノード増加に対する速度低下の原因を究明した上で、動的なロードバランス調整など、より大規模な並列化に向けたアルゴリズムの開発を行う必要がある。

最後に、今回の大規模計算で気がついたことをいくつかまとめておく。まず、ベンチマークだけではなく、大規模なプロダクトランも行う予定であったが、ファイル入出力に手間取り、規定時間内に終了することができなかった。これは、元のコードが一つのファイルに対して各プロセスが順番に情報を書き出すようになっており、8192 プロセスが同期しながら一つのファイルに書こうとして非常に時間がかかったためである。そこで、各プロセスが自分の管理する情報を別々のファイルに出力する方針に切り替えたが、ファイルを出力する途中でプログラムが異常終了してしまった。一度に多数のファイル出力要求にファイルシステムが耐えられなかったのか、それともプログラムの問題かはまだわかっていないが、ある程度まとめた (たとえばノードごと) に情報を集約してファイルを出力するなどの階層的処理を実装したほうがよさそうである。

今回の並列化には MPI を用いたが、使っている関数は隣接通信に MPI_Sendrecv、全体通信に MPI_Allreduce の二種類のみである。今回はウィークスケーリングによるベンチマークを取ったため、並列化効率が下がる原因は主に全体通信にあると予想されたが、512 ノード 8192 プロセスにおいても MPI_Allreduce が 1 ミリ秒もかかるとは思えないため、この速度低下は説明できない。隣接通信には MPI_Send と MPI_Recv を明示的に使う方法や、MPI_Isend と MPI_Irecv を用いて計算を隠蔽する方法などがある。別の計算機で試したところ、通信をペアワイズして MPI_Send と MPI_Recv を用いて実装した場合よりも MPI_Sendrecv のみを用いたほうが速いことがわかっている。しかし、HA8000 では通信順序を考慮してプロセスをペアリングしたほうが速くなる可能性があり、これは今後調べる予定である。さらに、計算規模が大きくなるとペアリストの有効期限が早く切れるプロセスが出現しやすくなり、その分だけ計算速度が遅くなる可能性がある。これは通信や並列化とは関係がないため、並列化効率の低下としてカウントするのは不適切である。今後、計算規模とペアリストの寿命についても調べる必要がある。

今回到達した 40 億粒子という粒子数は、液体中に気泡が多数発生し、かつマクロな流れとカップルしはじめる物理系を再現可能な規模である。すなわち、我々は沸騰流の全粒子計算に手が届きつつある。今後、より大規模な分子動力学計算により、これまで未解明であった相間摩擦や気泡間相互作用などの物理現象が解明されることが期待される。

謝辞

本研究は King Abdullah University of Science and Technology (KAUST) による Global Research Partnership (Award No. KUK-I1-005-04) の支援を受けて行われた。

参 考 文 献

[1] M. Isobe, Int. J. Mod. Phys. C, **10**, 1281 (1999).

[2] S. D. Stoddard, and J. Ford, Phys. Rev. A **8**, 1504 (1973).

[3] W. Form, N. Ito, and G. A. Kohring, Int. J. Mod. Phys. C **4**, 1085 (1993).