ISSN 1344-9567

スーパーコンピューティング ニュース

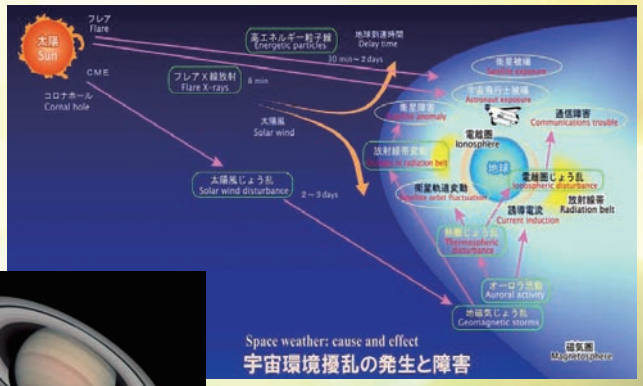
SUPERCOMPUTING NEWS

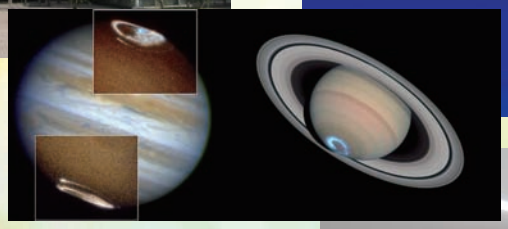
東京大学情報基盤センター スーパーコンピューティング部門

Vol.15, No.Special Issue 1

2013.2

特集：Oakleaf-FX「大規模 HPC チャレンジ」





東京大学情報基盤センター
Information Technology Center, The University of Tokyo

表紙の画像について、以下のとおり提供を受けています。

- 様々な宇宙天気現象（提供：情報通信研究機構）
- 木星と土星におけるオーロラ発光（提供：NASA）
- さまざまなスケールで存在するダークマターハローとそこでできる天体
  （提供：筑波大学計算科学研究センター 石山智明 研究員）

目　　次

特集：Oakleaf-FX「大規模 HPC チャレンジ」

平成 24 年度特集号発行にあたって …… 1
中島研吾（東京大学情報基盤センター）

急減圧液体における気泡分布関数の数値的解析…… 3
渡辺宙志（東京大学物性研究所）
鈴木　将（九州大学大学院工学研究院）
伊藤伸泰（東京大学工学系研究科）

電磁流体コードによる惑星磁気圏シミュレーション性能測定…… 15
深沢圭一郎・南里豪志（九州大学情報基盤研究開発センター）
梅田隆行（名古屋大学太陽地球環境研究所）

2 次元フラストレート系の計算科学的研究…… 33
中野博生（兵庫県立大学大学院物質理学研究科）
藤堂眞治（東京大学物性研究所）
坂井　徹（原子力研究開発機構）

超並列重力多体問題シミュレーションコードの性能測定…… 47
石山智明（筑波大学計算科研究センター）

100 億超格子を用いた自動車の大規模流体解析への挑戦 …… 59
小野謙二（理化学研究所計算科学研究機構）
大西順也（東京大学生産技術研究所）

ポストペタスケール環境における大規模疎行列解法のための
数値計算・通信ライブラリに関する研究…… 71
林　雅江・中島研吾・石川　裕（東京大学情報基盤センター）

平成24年度特集号発行にあたって

中 島 研 吾

東京大学情報基盤センター

1. はじめに

本特集号は、平成24年度に東京大学情報基盤センター（以下、本センター）で実施した公募プロジェクト「FX10スーパーコンピュータシステム「大規模HPCチャレンジ」[1]」（以下、「大規模HPCチャレンジ」）について、平成24年4月～6月の試験運転期間中に実施された課題の報告をまとめたものです。

2. FX10スーパーコンピュータシステム「大規模HPCチャレンジ」

「大規模HPCチャレンジ」は、平成24年4月より稼働を開始したFX10スーパーコンピュータシステム（愛称：Oakleaf-FX、ピーク性能：1.13 PFLOPS）を使用し、大規模科学技術シミュレーション、先端的スーパーコンピューター環境を目指した研究の促進を目的としたものです。

年3～4回公募を実施し、本センタースーパーコンピューティング研究部門の教職員及び外部委員により構成される審査委員会における審査の上、年間で12件程度の優れた研究課題を採択します。採択された研究グループは、Oakleaf-FXの全計算ノード数である4,800ノード（76,800コア）を最大24時間占有利用[2]することができます。

対象は、最大 4,800 ノードを使用する大規模計算を実施する研究とし、自作プログラム、オープンソースプログラムを利用した課題に限定しています。

申込者及び研究グループのメンバーは、国内外の並列計算機を利用した大規模計算の実績があることを前提とし、以下の様な「High-Performance Computing」に関連した幅広い分野の研究を対象としています。

・大規模シミュレーション
・大規模データ処理
・大規模ベンチマーク、演算・通信システム性能評価
・その他、大規模計算に関係するソフトウェア実行

「大規模 HPC チャレンジ」に採択された課題は、終了後、得られた成果をもとに、スーパーコンピュータの利用に関連する課題での科学研究費補助金（科研費）の採択や「学際大規模情報基盤共同利用・共同研究拠点[3]」（JHPCN）の公募型共同研究等への進展が期待されます。

第 1 回（試験運転期間）の公募では、表 1 に示す 7 件の課題が採択されました。

---

1 http://www.cc.u-tokyo.ac.jp/service/4800hpc/
2 試験運転期間中は、1 グループで最大 48 時間占有利用とした。
3 http://jhpcn-kyoten.itc.u-tokyo.ac.jp/

表１　「大規模 HPC チャレンジ」第１回（試験運転期間）採択課題

課題名	代表者氏名	代表者所属
急減圧液体における気泡分布関数の数値的解析	渡辺宙志	東京大学物性研究所
電磁流体コードによる惑星磁気圏シミュレーション性能測定	深沢圭一郎	九州大学情報基盤研究開発センター
2次元フラストレート系の計算科学的研究	中野博生	兵庫県立大学大学院物質理学研究科
超並列重力多体問題シミュレーションコードの性能測定	石山智明	筑波大学計算科学研究センター 神戸分室
大規模グラフ処理ベンチマークGraph500のスケーラブルな探索手法による性能評価	鈴村豊太郎	東京工業大学
100億超格子を用いた自動車の大規模流体解析への挑戦	小野謙二	理化学研究所計算科学研究機構 （採択時：東京大学生産技術研究所）
ポストペタスケール環境における大規模疎行列解法のための数値計算・通信ライブラリに関する研究	林　雅江	東京大学情報基盤センター

急減圧液体における気泡分布関数の数値的解析

渡 辺 宙 志

東京大学物性研究所

鈴　木　将

九州大学大学院工学研究院

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、大規模な計算が可能な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。気液混相流の研究とは、典型的には熱いパイプの中を流れる流体の振る舞いを理解しようというものである。外からの熱流入を受け、流体は沸騰をしながら流れることになり、相転移と流動がカップルした現象を起こす。こういった気液混相流は冷却システム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扱うことで、(古典的な意味で) 非経験的な知見を得ようとするのが我々が目指す全粒子計算である。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。しかし、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。並列計算の観点で言えば、短距離力古典 MD は並列化は比較的容易であり、コアあたりの粒子数が多い計算を行えば並列化の粒度も疎となり、良好なスケーリングが期待される。しかし、単なるベンチマークにとどまらず、具体的に相転移を伴う大規模なプロダクトランを実行するためには観測ルーチンも含めた並列化、計算実行カーネルの高速化など、解決すべき問題が多い。本稿では、東京大学情報基盤センターの PRIMEHPC FX10 に向けたカットオフ付き Lennard-Jones 粒子の計算の高速化手法、及び急減圧による多重気泡核生成の計算結果について報告する。

2. アルゴリズムと高速化手法

2.1. 基本アルゴリズム

本研究で用いる粒子間ポテンシャルは、カットオフのある 6-12 型 Lennard-Jones ポテンシャルである。その関数形は以下で与えられる [1]。

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

ただし、r は粒子間、σ は粒子直径、ε はエネルギースケールである。以下、ボルツマン定数 k_{B} を 1 とし、そのほかの物理量も σ、ε を用いて無次元化した単位系を取る。二つの追加係数 c_2, c_0 は、カットオフ距離 r_{c} においてポテンシャルと力がちょうどゼロになる、すなわち $V(r_{\mathrm{c}}) = V'(r_{\mathrm{c}}) = 0$ の条件から定められる。カットオフ距離は、ベンチマークで $r_{\mathrm{c}} = 2.5$、多重気泡核生成過程の計算では $r_{\mathrm{c}} = 3.0$ とした。短距離相互作用 MD シミュレーションのアルゴリズムとして、グリッド探索 [2, 3, 4, 5](もしくはリンクリスト探索 [6, 7]) により相互作用粒子を探索し、ペアリスト構築後は Bookkeeping 法 [8] によりリストの再利用を行い、リストの寿命判定に dynamic upper time cutoff (DUTC) 法 [9] を拡張したものを採用した。詳細については文献 [10] にまとめたので、ここではいくつかのポイントのみ述べる。

メモリアクセスの最適化のため、相互作用する粒子のペアを探索した後に粒子についてソートを行っている。後の便利のため、相互作用している粒子対のうち番号の若い方を Key 粒子、その相手を Partner 粒子と呼ぼう。なお、これはいわゆる i 粒子、j 粒子に対応する。Key 粒子によりソートをすると、同じ粒子と相互作用する粒子がまとめられる。これは、Key 粒子の情報をレジスタに載せることで、必要メモリバンド幅を削減することと、近い番号の Key 粒子と相互作用する Partner 粒子には重なりが多いことによるキャッシュ効率の向上を目的としている。

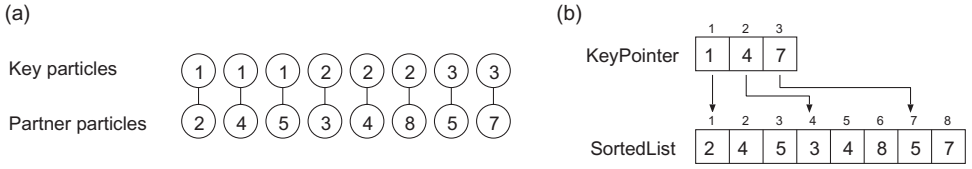


図 1: (a) Key 粒子の粒子番号によりソートしたペアリスト。(b) Partner 粒子のデータフォーマット。ひとつの一次元配列で Partner 粒子の番号を全て格納し、どこからどこまでがどの Key 粒子と相互作用をしているのかをポインタで表現する。例えば、i 番目の Key 粒子と相互作用する最初の Partner 粒子は SortedList[KeyPointer[i]] に、その次の Partner 粒子は SortedList[KeyPointer[i]+1] に格納されている。

並列化アルゴリズムは、単純空間分割法を採用する。また、ロードバランスなどは考慮せず、全ての領域の体積を同じにする。空間分割法は、相互作用距離にくらべて分割された領域が十分大きければ通信時間がほぼ無視できるため、大規模短距離 MD に向いた方法である。通信は基本的には隣接する領域でしか発生しないが、ペアリストの有効期限判定には注意を要する。それぞれの領域において、独立したペアリストを保持するが、そのリストの有効期限は領域により異なる。また、一般にリストの再構築は時間積分 1 ステップに比べて数倍から 10 倍以上時間がかかる。従って、それぞれが独立にペアリストの再構築を行うと、再構築中は別のプロセスからの通信要求に答えることができず、その間は全てのプロセスが待たされてしまう (図 2 (a))。そこで、どこか一つでもペアリストの有効期限が切れたら、他の全てのプロセスのペアリストも同時に再構築を行うことにする (図 2 (b))。これにより、ペアリストの平均再利用回数は同期しない場合よりも少なくなるが、ペアリスト再構築中のアイドル時間がなくなるため、

全体としては効率が良くなる。これは MPI_Allreduce で整数を一つ送ることで実装しており、時間積分の 1 ステップに比べて通信時間はほぼ無視できるが、全体が毎ステップ同期する必要があるため、ロードインバランスの影響を強く受けることになる。

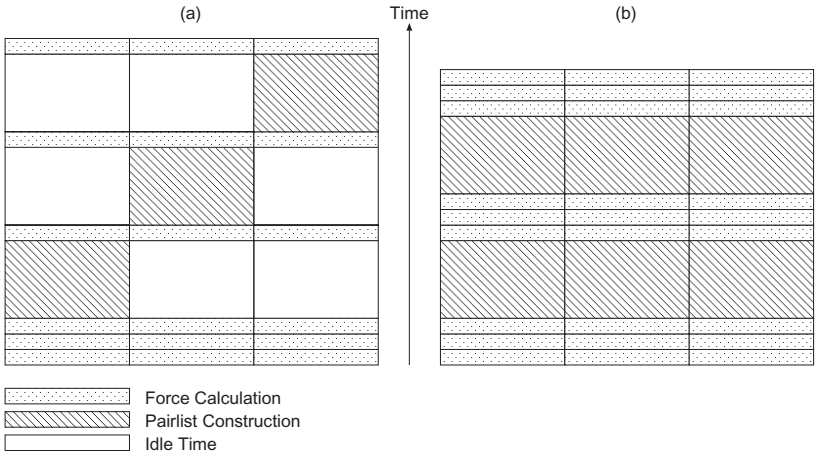


図 2: (a) ペアリストの同期を行わなかった場合。あるプロセスがペアリストの再構築中に別のプロセスが待たされる。これが繰り返されることにより、全体のアイドル時間が増加してしまう。(b) ペアリストの同期を行った場合。どこかひとつでもペアリストの有効期限が切れたら、全てのプロセスのペアリストを同時に再構築する。ペアリストの平均再利用回数は減るが、アイドル時間はなくなる。

2.2. 並列化の実装

並列化は、flat-MPI とするか、MPI/OpenMP のハイブリッドとするかの選択がある。短距離古典 MD においては、ハイブリッドの flat-MPI の方が計算が速い場合が多いが、flat-MPI はメモリ消費が大きく、特に大規模計算においてプロダクトランに必要なメモリ確保が難しくなる場合がある。そこで我々は、MPI/OpenMP のハイブリッドコードを開発した。ハイブリッド並列化を行う際、一般にはノード間を MPI を用いた空間分割、ノード内を OpenMP を用いたループ分割にすることが多いが、今回我々はノード間、ノード内のどちらも空間分割とする、疑似 flat-MPI 法を採用した。これは、OpenMP のスレッド番号を MPI のランク番号のように扱う事で実現する。疑似 flat-MPI 法は、共有メモリを分散メモリとして扱う点で flat-MPI と同様な計算手法であるが、MPI のプロセス数が減る分、消費メモリの低減が期待できる。しかし、環境によってはスレッド間の MPI 通信ができないか、制限を受けることが多いため、通信は一度プロセス単位でまとめて行い、再度スレッドに分配する必要がある (図 3)。全ての物理量の観測ルーチンを、この二段階通信にあわせて用意しなければならないのが、ループ分割によるハイブリッド並列に比べた、疑似 flat-MPI による方法の大きなデメリットである。しかし、ループ分割による方法では、ペアリスト作成などもループ分割しなければ全体の性能が向上せず、かつデータの排他制御も正しく扱わなければならない。何より、計算のホットスポットである力の計算で、OpenMP 並列と SIMD 化を同時に考慮しなければならない点がコードの最適化の大きな障害となる。疑似 flat-MPI では、OpenMP 並列が上位の階層で済んでいるため、力の計算はシングルコアの最適化のみ考慮すれば良い。

既に flat-MPI のコードがあるのなら、疑似 flat-MPI コードを実装するのは比較的容易である。具体的には、計算を行う MDUnit クラスと、通信を行う MDManager クラスにより実装している (開発言語に C++ 言語を用いている)。MDMangaer クラスにおいて、MDUnit クラス

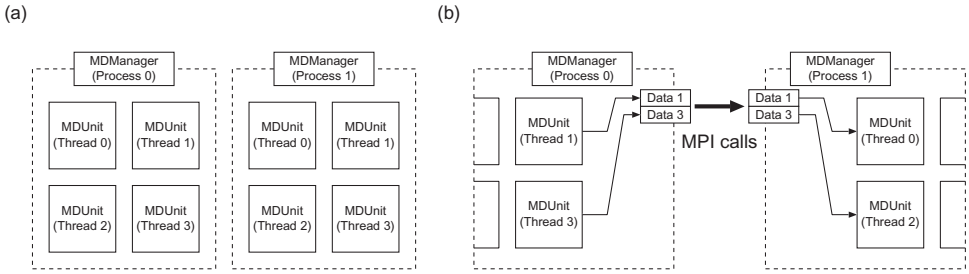


図 3: 疑似 flat-MPI 法。見やすさのため、二次元系で表示してある。2 つの MPI プロセスが、それぞれ 4 つの OpenMP スレッドを持っている。(a) MDUnit と MDManager がそれぞれ MPI プロセスと OpenMP スレッドに対応する。計算領域は MDUnit に割り当てられ、時間積分を行う。MDManager は MDUnit 間の通信を管理する。(b) プロセス間通信。環境によってはスレッドからの MPI 通信が許可されていないか制限を受けるため、一度プロセス単位でデータをまとめてからプロセス間で通信し、その後スレッドに分配する。

List 1: MDUnit クラスの初期化

```
int tid;
MDUnit *mdp;
std::vector <MDUnit *> v;
#pragma omp parallel shared(v) private(tid,mdp)
{
  tid = omp_get_thread_num();
  mdp = new MDUnit();
  #pragma omp critical
  v.push_back(mdp);
}
mdv.resize(num_threads);
for(unsigned int i = 0; i < v.size(); i++) {
  mdv[v[i]->ID] = v[i];
}
```

の初期化を行うサンプルコードは List 1 の通りである。ここで、MDUnit の vector である mdv は MDManager のメンバ変数であるが、OpenMP の制約でクラスのメンバ変数を shared 指定できないため、一度テンポラリ領域に確保してからソートしてコピーしている。MDUnit へのポインタは全てマスタースレッドに確保されるが、実際に計算に使われる配列は MDUnit 内で確保されるため、NUMA 最適化は自動的に満たされている。スレッド数は、MDManager クラスが保持する MDUnit クラスの数である。MDUnit がひとつしかなければ flat-MPI、複数あればハイブリッド実行となる。OpenMP 並列化、例えばたとえば力計算の並列化は以下の様になされる。

```
#pragma omp parallel for schedule(static)
for(int i = 0; i < num_threads; i++) {
  mdv[i]->CalculateForce();
}
```

すなわち、flat-MPI と疑似 flat-MPI の違いは、MDUnit クラスのメソッドが MPI プロセスから呼ばれるか、OpenMP スレッドから呼ばれるかだけしかない。従って、適切な領域配置を行えば、flat-MPI とハイブリッドで (足しあげの順序等も含めて) 全く同じ計算を実行することができる。

2.3. FX10 向けのチューニング

今回の計算に向け、力の計算ルーチンの FX10 向けのチューニングを行った。要点は、作用反作用を使わないことによるソフトウェアパイプライニングの促進、SIMD 化、逆数近似命令の利用とその精度補正である。準備的計算により、Intel Xeon などのアーキテクチャに比べて、FX10 での実行効率が悪いことがわかっていたが、その理由の一部は間接参照を伴う運動量の書き戻しにあることがわかった。通常、力の計算を行う際、作用反作用を用いて、i 粒子から j 粒子への力積を計算した後、i 粒子だけでなく、j 粒子の運動量にもその力積を反映させる。しかし、i 粒子と相互作用する j 粒子番号は間接参照となっており、これがコンパイラによるソフトウェアパイプライニングを阻害していただけなく、store 命令そのもののレイテンシが強く見えているようであった。そこで、作用反作用を使わないことで間接参照を伴う運動量の書き戻しを削除した。これにより計算量は二倍となるが、速度向上は二倍以上となり、トータルで計算速度が大きく向上した。また、コンパイラが効率的に SIMD 命令を出力できていないようだったため、intrinsic 命令により SIMD 命令を明示的に記述した。基本的にはループを二倍展開することで独立な命令を作る、自明 SIMD 化を行ったが、除算の SIMD 命令が見つからなかったため、代わりに逆数近似命令を利用した。逆数近似命令 (frcpad) は、高速に逆数を計算できる代わりに、その相対誤差は 8bit、すなわち最大で 1/256 程度に達する。通常、コンパイラがこの命令を出力する場合は、精度補正命令も同時に発行してくれるのだが、明示的に frcpad を実行したため、精度補正も明示的に行う必要がある。具体的には、以下のような手順で、二次の補正を行った。粒子間距離を r としよう。実際に計算するのはその二乗、r^2 である。その逆数近似値を $\tilde{r}^{-2}$ とする。すなわち、$\tilde{r}^{-2} = \mathrm{frcpad}(r^2)$ である。逆数近似と、倍精度で計算した場合の精度差、すなわちエラー ε は以下で表される。

$$\tilde{r}^{-2}r^2 = 1 + \varepsilon, \qquad (|\varepsilon| < 1/256) \tag{2}$$

力の計算には r^{-8}、および r^{-14} の計算が必要となるが、以下に r^{-8} の精度補正を示す。式 (2) を以下のように変形する。

$$\begin{aligned} r^{-8} &= \frac{(\tilde{r}^{-2})^4}{(1+\varepsilon)^4} && (3)\\ &= (\tilde{r}^{-2})^4(1 - 4\varepsilon + O(\varepsilon^2)) && (4)\\ &= (\tilde{r}^{-2})^4(5 - 4(1+\varepsilon) + O(\varepsilon^2)) && (5)\\ &= (\tilde{r}^{-2})^4(5 - 4\tilde{r}^{-2}r^2 + O(\varepsilon^2)). && (6) \end{aligned}$$

すなわち、r^{-8} の逆数近似値 $(\tilde{r}^{-2})^4$ に対し、$(\tilde{r}^{-2})^4(5 - 4\tilde{r}^{-2}r^2)$ と補正項をかけてやれば、その精度は $O(\varepsilon^2)$ となる。同様に、r^{-14} への補正項は $(8 - 7\tilde{r}^{-2}r^2)$ と求められる。どの程度の精度を行う必要があるかは現象により異なるが、今回の計算では二次の補正で十分であることを確認している。実際のコードではさらに二倍に展開し、一部のソフトウェアパイプライニングを手で行う事で、コンパイラによるソフトウェアパイプライニングを促進している。最終的にループは 4 倍展開されているが、2 倍が自明 SIMD 化のために、2 倍がソフトウェアパイプライニングのために行われている。

3. ベンチマーク

作成したコードを用いて、ベンチマークシミュレーションを行った。計算条件は以下の通りである。

- コアに割り当てる領域のサイズは $100 \times 100 \times 100$ の立方体。
- 数密度は 0.5、すなわちコアあたり 50 万粒子、ノードあたり 800 万粒子。

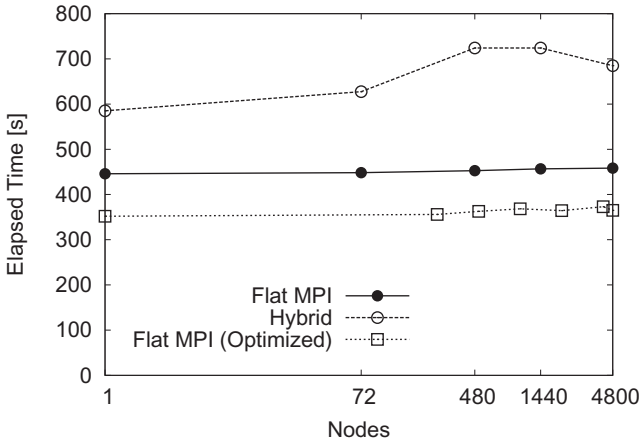


図 4: ベンチマーク結果。1000 ステップの時間積分にかかった時間。Intel Xeon 向けのコードの flat-MPI、ハイブリッド実行の結果、及び FX10 向けコードの flat-MPI の結果。

- 初期条件は FCC 格子とし、全方向に周期的境界条件。
- 時間積分は二次のシンプレクティック積分 (陽解法)。
- 時間刻みは 0.001。カットオフは 2.5。
- 初期速度は絶対値 0.9 とし、方向はランダム。
- 150 ステップのあと、1000 ステップの時間積分にかかった時間を計測。
- ノード数あたりの粒子数を固定したままノード数を増やす、ウィークスケーリング解析。

計算速度は MUPS (millions of updates per second) という単位を用いて定義する。これは百万粒子を 1 秒間で 1 ステップ計算できたら 1 となる単位である。flat-MPI と、ハイブリッド両方の計算を行った。ハイブリッド計算では、ノードあたり 16 スレッドとした。flat-MPI の計算は、Intel Xeon 向けのチューニングを施したコードと、FX10 向けのチューニングを施したコードの両方について、ハイブリッドは Intel Xeon 向けのチューニングを施したコードのみについて計算を実行した[1]。それぞれの実行結果を表 1、表 2、表 3、及び図 4 にまとめた。最大で 4800 ノード、76800 プロセス/スレッドの計算で、最大 384 億粒子の計算を行った。flat-MPI のスケーリングはほぼ完璧であるにもかかわらず、ハイブリット実行の結果は揺らぎが大きく、かつ flat-MPI の計算より二倍近く遅い。また、FX10 向けのチューニングを行ったコードは、Intel Xeon 向けのチューニングを行ったコードを FX10 で実行した場合に比べて 25%程度速度が向上している。なお、4800 ノードの実行においてノードあたりの消費メモリは flat-MPI の場合で 14GB、hybrid 実行で 10GB であり、ほぼ同じ計算を行っているにもかかわらず、flat-MPI の方が 4GB 余計にメモリを消費していた。

最大の計算 (4800 ノード) における計算効率は、Intel Xeon 向けコード+flat-MPI で 56.5 TFLOPS (ピーク性能比 5%)、FX10 向けコード+flat-MPI で 193 TFLOPS (ピーク性能比 17%) であった。FX10 向けのコードは作用反作用を利用していないために FLOPS の絶対値には意味がないが、二倍の計算量を、二倍以上の効率で計算していることから、トータルの計算実行速度は 25%程度向上している。

[1]FX10 向けのチューニングではスレッドローカルな配列を利用する必要があったが、そのハイブリッド版の用意が実行に間に合わなかったため。

ノード数	粒子数	実行時間 [s]	速度 [MUPS]	並列化効率
1	8,000,000	445.815	17.9447	1.0
72	576,000,000	448.275	1284.93	0.99
480	3,840,000,000	452.781	8480.93	0.98
1440	11,520,000,000	456.641	25227.7	0.98
4800	38,400,000,000	458.399	83769.9	0.97

表 1: Intel Xeon 向けチューニング+flat-MPI の実行結果。並列化効率はシングルノードの実行時間と比較した値。

ノード数	粒子数	実行時間 [s]	速度 [MUPS]	並列化効率
1	8,000,000	585.160	13.6715	1.00
72	576000000	627.449	918.003	0.93
480	3840000000	724.139	5209.34	0.79
1440	11520000000	724.139	15908.5	0.81
4800	38,400,000,000	684.871	56068.9	0.85

表 2: Intel Xeon 向けチューニング+ハイブリッドの実行結果。

4. 多重気泡生成シミュレーション

4.1. 多重気泡生成

気液混相流の全粒子計算への第一歩として、急減圧による多重気泡生成シミュレーションを行った。液相からの気泡生成には、温度上昇による沸騰 (boiling) と、減圧によるキャビテーション (cavitation) に区別される。キャビテーションは船のスクリューの腐食などに関与する。シミュレーションで実現が容易なのは、定積条件下において急減圧することである。これは、ある瞬間に系を一様膨張させることで実現される。減圧の度合いが弱いと、系は準安定状態となり、一定時間後に気泡が生成するが、減圧の度合いが強い場合には、系はスピノーダル分解を起こし、減圧後直ちに多数の気泡が生成する。今回は、強い急減圧を行う事で多重気泡生成を再現する。多重気泡生成では、初期ステージにおいて多数の気泡が生成した後、圧力を介した気泡間相互作用により、大きな気泡がより大きく、小さい気泡がより小さくなる Ostwald 的成長を起こす。その後、十分な時間の後に単一気泡へと収束する。同様な現象に、液滴核生成がある。これは純粋な気相状態から条件を変えることで過飽和状態とし、液滴が成長する過程であ

ノード数	粒子数	実行時間 [s]	速度 [MUPS]	並列化効率
1	8,000,000	351.821	22.7389	1.00
256	2,048,000,000	355.829	5755.57	0.99
512	4,096,000,000	362.775	11290.7	0.97
1024	8,192,000,000	368.275	22244.2	0.96
2048	16,384,000,000	364.272	44977.4	0.97
4096	32,768,000,000	373.007	87848.3	0.94
4800	38,400,000,000	364.920	105229.0	0.96

表 3: FX10 向けチューニング+Flat-MPI の実行結果。

る。どちらも気液相転移による緩和過程であるが、まわりが液体であるが故に気泡間相互作用は液滴間相互作用よりも強く、かつ気泡が成長する際に液体にする仕事を考慮しなければならないなど、気泡生成過程の方が液滴生成過程よりも考慮すべき項目が多く、解析が難しくなる。

4.2. 計算の詳細

まず平衡状態にある液相を用意する。これは、相図上で気液共存線に近い液相 (温度 0.9、密度 0.7) において、NVT シミュレーションを行うことで実現する。温度制御には Nose-Hoover 法を用いた [11]。温度を制御している間の時間積分は、二次の指数分解法 (reversible system propagator algorithm, RESPA) を用いた [12]。カットオフ距離は 3.0、時間刻みはシミュレーションを通して 0.005 とした。系の大きさはコアあたり一辺 30 の立方体とし、全体の大きさは $1440 \times 1200 \times 1200$、数密度が 0.7 であることから粒子数は 14 億 5 千万である。温度制御付きで 10000 ステップ計算して緩和させた後、密度が 0.65 となるように系を一様膨張させ、以後は温度制御をせず NVE シミュレーションにより時間発展を行った。計算は緩和も含めて 17800 ステップ実行し、減圧後は 1000 ステップに一度、以下に述べる局所密度情報を保存した。なお、計算は flat-MPI ではメモリ不足で失敗したため、ノードあたり 1MPI プロセス ×16 OpenMP スレッドのハイブリッドで実行した。実行時間は 15 時間であった。

気相、液相の判定のため、空間を小さなセルに区切り、一定密度以下であるセルを気相、それ以上の密度を持つ系を液相と定義した [13]。隣接する気相セルは同じ気泡と定義し、その体積を計算した。今回の計算ではセルの大きさを一辺 3.0 の立方体とし、気相判定の密度の閾値を 0.2 とした。なお、気泡判定アルゴリズムは原理的にパーコレーションにおけるクラスタリングアルゴリズムと同じものであるが、これを全系で行うと FFT 型の全体通信が必要となり、通信路の輻輳により極めて時間がかかる。そこで、各プロセスにおいて局所密度情報を計算した後、その情報をルートノードに集約し、クラスタリング処理はルートノードのみで行うことにした。各コアの担当領域が一辺 30 の立方体であるため、各コアあたり 1000 個のセルを持つ。もし密度を倍精度実数で表現すると全体で 614MB の情報が必要となるが、通信量、およびスナップショットのデータ量削減のため、それぞれのセルについて何個の粒子が含まれるかを 1 バイト (unsigned char) で表現することとした。各セルの体積が 27 であり、液相密度が 0.7、液相がほぼ非圧縮であることから局所数密度が 1 を超える可能性は極めて低く、密度情報を精度を全く失うことなく保存可能である。これにより、密度スナップショットに必要な容量は 76.8MB となった。スナップショットは 169 回保存したため、計算全体で利用したストレージは 13GB 程度であった。

4.3. 計算結果

図 5 に圧力の時間発展の様子を、図 6 に気泡の可視化のスナップショットを示す。液相状態で緩和させたあと $t = 50$ において急減圧を行った結果、系は負圧となる。減圧直後から気泡が生成することで系の圧力が上昇していく様子がわかる。また、最初は気泡が多数生成するが、気泡成長にともない圧力が上がることでその圧力下で成長できる気泡、できない気泡にわかれる。成長できない気泡は圧力に負けてつぶされていき、大きな気泡はより大きく、小さな気泡はより小さくなっていく (Ostwald 成長)。

系に存在する気泡の数の時間発展を図 7 (a) に示す。減圧後に急激に増えた気泡は、途中から減少に転じる。これは気泡成長により圧力が上昇し、新たな気泡生成が抑制された上に、小さな気泡がどんどんつぶされて減って行くことにより、大きな気泡がより大きく成長していることによる。減圧後、1 万ステップ後の気泡のサイズ分布関数を図 7 (b) に示す。分布関数が十分な精度で観測できていることがわかる。

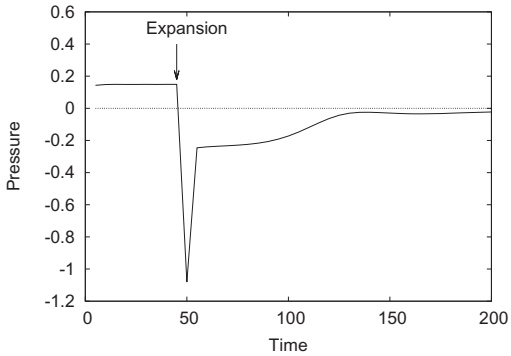


図 5: 圧力の時間発展。最初は純粋な液相 (温度 0.9、密度 0.7) に保たれているが、$t = 50$ において急減圧を行った。圧力は一度負圧となり、その後気泡生成により徐々に上昇する。

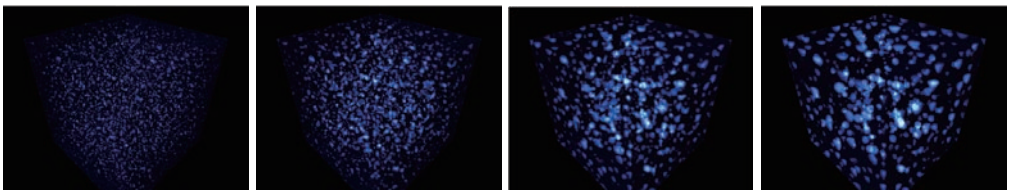


図 6: 気泡生成過程。左から右に時間発展する。減圧直後から多数の気泡が発生し、その後 Ostwald 的成長が見られる。なお、可視化には POV-Ray を用いた [14]。

4.4. ロードインバランス

多重気泡生成シミュレーションでは、減圧後に気泡が成長する過程で密度揺らぎが大きくなる。気泡が領域をまたぐレベルに成長すると、各コアの担当する領域が気相のみ、液相のみとなる場合があり、大きなロードインバランスが発生する。毎ステップ、グローバルな同期を取っているため、ロードインバランスは全体の計算効率に直結する。ここではロードインバランスによる計算速度低下の程度を見積もる。短距離 MD において、計算負荷は大雑把に粒子数に比例する。今回は全ての分割領域の体積が等しいため、計算負荷は密度に比例する。理想的なロードバランスが達成された場合、計算負荷は平均密度 ρ_{ave} に比例する。また、気泡生成が始まったあと、最も計算負荷が重いのは、領域内がすべて液相となる場合である。液相はほぼ非圧縮であることを考慮すると、もっとも計算負荷が重い領域の計算負荷は液相密度 ρ_{l} に比例し、これが全体の計算速度を決める。今回の計算では、初期密度 0.7 から 0.65 に減圧したため、平均密度 $\rho_{\mathrm{ave}} = 0.65$ である。また、減圧後の温度は 0.88 となった。この温度での気液共存密度は、気相が $\rho_{\mathrm{g}} = 0.0344$、液相が $\rho_{\mathrm{l}} = 0.687$ である [15]。ロードインバランスによる性能劣化は、平均密度 ρ_{ave} と液相密度 ρ_{l} で決まるため、今回の計算では $1 - \rho_{\mathrm{ave}}/\rho_{\mathrm{l}} \sim 0.05$、すなわち 5%程度しか劣化しない。これは気泡が成長しきった、最終ステージでの最悪の値であるため、実際の計算ではロードインバランスはほとんど問題とならない。気相と液相の共存密度の比は $\rho_{\mathrm{l}}/\rho_{\mathrm{g}} \sim 20$ であるため、計算が軽い領域と重い領域の計算負荷の差は最大で 20 倍近くになるが、計算が軽くなった領域の分の計算負荷を、多数の別の領域が分担することで全体の計算時間はほとんど増えていないという構図になっている。ただし、これは減圧シミュレーションによる特徴であるため、例えば過飽和気相から液滴生成を計算する場合にはロードインバランスは極めて深刻な問題となる。

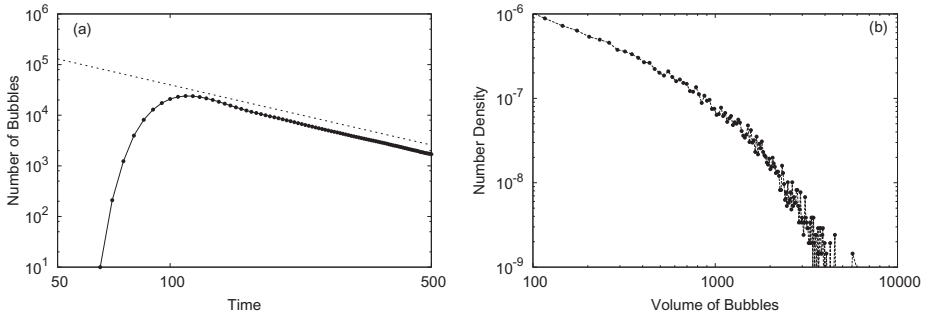


図 7: (a) 気泡数の時間発展 (両対数プロット)。点線は傾き-1.7。(b) $t = 100$ (減圧後 1 万ステップ後) の気泡数密度分布関数 (両対数プロット)。

5. ハイブリッド実行時の速度向上

HPC チャレンジ後、ハイブリッド実行時の速度が flat-MPI よりも遅い件について富士通側に調査を依頼したところ、STL の std::vector が原因であることが判明した。具体的には、隣接粒子探索をするルーチンにおいて std::vector を利用していたのだが、そこがマルチスレッド実行で遅くなっていた。問題となった std::vector は以下のように関数内のテンポラリバッファとして用いられていた。

```
void
SomeClass::SomeFunction(void){
  std::vector<int> v;
  // v を用いた処理
}
```

ただし、SomeFunction はそれぞれのスレッドから独立に実行される関数であり、この領域では排他制御の必要はない。この変数宣言を以下のようにスタックからヒープに確保した上で、スレッドローカルな変数となるように変更した。

```
void
SomeClass::SomeFunction(void){
  static __thread std::vector<int> v;
  // v を用いた処理
}
```

なお、変更箇所は一カ所のみである。以上の変更による実行速度の変化を表 4 及び図 8 にまとめる。1 ノードの計算で、flat-MPI の場合は 16 プロセス、ハイブリッドの計算は 1 MPI プロセス 16 OpenMP スレッド、粒子数は 100 万粒子 (コアあたり 6 万 2500 粒子)。1000 ステップ計算するのにかかった時間。flat-MPI では、vector をスタックに取ってもヒープに取っても実行速度に影響はなかった。しかし、1 MPI プロセス、16 OpenMP スレッド実行をした場合では、vector をスタックに取ると flat-MPI に比べて 40%程度遅くなり、ヒープに取ると flat-MPI と遜色ない実行速度となった。

6. まとめ

疑似 flat-MPI による並列化がなされた短距離古典分子動力学法コードを開発し、最大 4800 ノード、384 億粒子の計算を実行、193 TFLOPS の性能 (ピーク性能比 17%) を達成した。FX10 向けのチューニングを行うことで、Intel Xeon 向けのコードを利用した場合よりも 25%程度の性能向上が得られた。既にキャッシュ効率の最適化が進んでいるコードであったため、最適化による速度向上はさほど大きいとは言えないが、同様な計算を Intel Xeon と実行した場合、コ

	実行時間 [s]
flat-MPI (stack)	47.73
flat-MPI (heap)	47.28
hybrid (stack)	65.29
hybrid (heap)	47.78

表 4: flat-MPI、hybrid のそれぞれについて、vector をスタックに取った場合 (変数宣言無指定) と、ヒープに取った場合 (static 指定)。

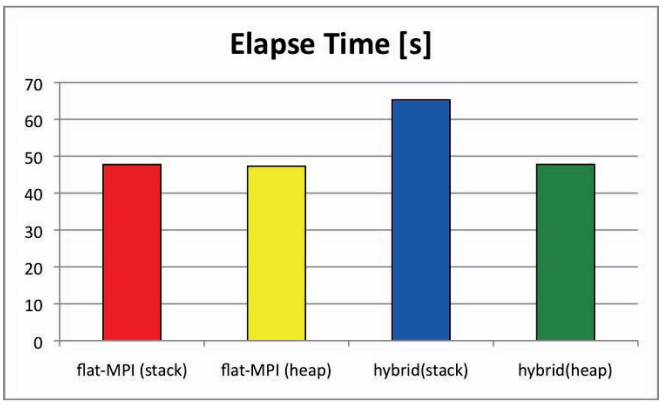


図 8: flat-MPI、hybrid のそれぞれについて、vector をスタックに取った場合 (変数宣言無指定) と、ヒープに取った場合 (static 指定)。

アあたりの性能は FX10 の方が 2 倍近く低く [10]、メモリバンド幅、キャッシュヒット率、力の計算アルゴリズム等から予想される理論ピーク性能に比べて実行性能が低いため、まだ性能向上の余地があると思われる。

また、flat-MPI 実行とハイブリッド実行の比較も行った。コアあたりの計算量及びノード間通信のデータ量は同じまま、ノード間通信回数はむしろハイブリッド実行の方が少ないにもかかわらず、flat-MPI の方がハイブリッド実行よりも二倍近く速かった。その後の調査により、std::vector に問題があることが判明し、コードを修正することで現在では flat-MPI と遜色のない実行速度が得られている。並列化のスケーリングに関しては、1 ノード実行を基準とした 4800 ノード実行の並列化効率は 96%と、計算規模を考えれば極めて良好であった。他の計算資源においては、システム由来と思われるノイズにより、大規模計算実行時に並列化効率の劣化が見られたが、本システムにおいてはシステム由来のノイズはほぼ観測されなかった。これは FX10 のシステムノイズ対策が効果的になされていることを示唆する。

FX10 の 4800 ノード全体を用いたプロダクトランとして、14 億粒子を用いた多重気泡生成シミュレーションを行った。これは気液相転移を伴う粒子計算としては (筆者の知る限りにおいて) 世界最大の計算である。粒子間相互作用の結果として気泡が生成し、気泡間相互作用の結果として Ostwald 成長が観測された。これは、少なくとも二階層にわたるマルチスケール構造が、ミクロなモデルから直接計算できたことを意味する。また、数億粒子を超える計算を行うと、気泡の数が 1 万を超えるため、気泡分布関数の時間発展を十分な精度で追うことができる。これにより、ミクロな相互作用から推定した分布関数の時間発展と直接比較することが可能となった。今後は、気泡間相互作用の直接推定や、系内の圧力分布の解析などにより、多数の気泡が存在する系の物性を解明し、気液混相流の解析へとつなげていきたい。

なお、本研究に用いたコードはオープンソースとして公開しているため、興味のあるかたは参照されたい [16]。

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)、及び科研費 若手研究 (B) (課題番号: 23740287) の支援を受けて行われました。

参 考 文 献

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

[2] M. P. Allen and D. J. Tildesley, Computer Simulation of Liquids (Clarendon Press, Oxford, 1987).

[3] D. E. Knuth, The Art of Computer Programming (Addison–Wesley, Reading, MA, 1973).

[4] G. S. Grest and B. Dünweg, Comput. Phys. Commun. **55** (1989) 269 – 285.

[5] D. M. Beazley and P. S. Lomdahl, Parallel Comput. **20** (1994) 173 – 195.

[6] B. Quentrec and C. Brot, J. Comput. Phys. **13** (1975) 430 – 432.

[7] R. W. Hockney and J. W. Eastwood, Computer Simulation Using Particles (McGraw-Hill, New York, 1981).

[8] L. Verlet, Phys. Rev. **159** (1967) 98 – 103.

[9] M. Isobe, Int. Mod. Phys. C **10** (1999) 1281–1293.

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

[11] W. G. Hoover, Phys. Rev. A **31** (1985) 1695 – 1697.

[12] M. Tuckerman, B. J. Berne, and G. J. Martyna, J. Chem. Phys. **97** (1992) 1990 – 2001.

[13] H. Watanabe, M. Suzuki, and N. Ito, Phys. Rev. E **82** (2010) 051604.

[14] ⟨http://www.povray.org⟩

[15] H. Watanabe, N. Ito, and C.-K. Hu J. Chem. Phys. **136** (2012) 204102.

[16] コードは MDACP という名前で公開している。今回の計算で用いたバージョンは 2.00。最新版は以下の URL から取得可能。⟨http://mdacp.sourceforge.net⟩.

電磁流体コードによる惑星磁気圏シミュレーション性能測定

深沢圭一郎、南里豪志

九州大学 情報基盤研究開発センター

梅田隆行

名古屋大学 太陽地球環境研究所

１．はじめに

１．１　磁気圏、宇宙天気とは

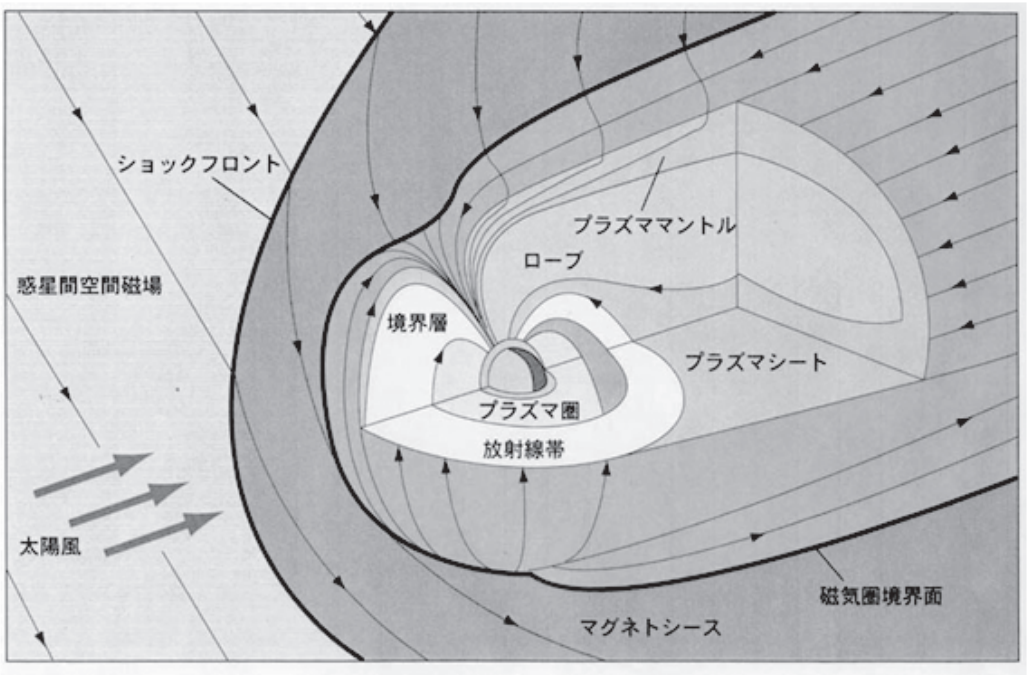
宇宙空間は真空と思われているが、我々人間が把握している99%はプラズマで満たされている。プラズマとは電離した気体のことであり、帯電している電子とイオンが分かれて存在する状態である。しばしば個体、液体、気体に続く物質の第４の状態とも呼ばれている。宇宙空間、特に我々が暮らす太陽系においては太陽から太陽風と呼ばれるプラズマの風が常時吹き出しており、太陽系全体にそのプラズマが充満している。このような宇宙プラズマは導電率が高いため、プラズマは磁力線に沿って動きやすく、また磁力線を横切る動きを取りにくい特徴がある。そのため、太陽風プラズマは太陽の磁場を伴って超音速で吹き出しており、地球のような磁場を持つ惑星に衝突すると、その磁場を伴ったプラズマの風が惑星の固有磁場と相互作用する。その結果、惑星磁場が変形し、磁気圏という第１図に示すような形をとる。

惑星磁気圏の太陽側は太陽風の圧力により圧縮された形をしており、反太陽側は太陽風によって引き延ばされた形をしている。図の左側から太陽風が流れ込み、磁気圏の前面には、弓形の衝撃波面(bow shock)が形成され、その内側にはマグネトシースと呼ばれる中間領域が存在する。磁気圏はその磁場構造により、内部磁気圏(中低緯度に根ざす閉じた磁力線からなる領域)と外部磁気圏(高緯度側に根ざす開いた磁力線からなる領域)の２つに分けられる。その内部磁気圏と外部磁気圏の昼側境界にあたるのが、カスプ領域である。ローブ領域は外側磁気圏で開いた磁力線領域であり、希薄なプラズマが存在している。ローブに挟まれた閉じた領域がプラズマシートと呼ばれる部分で地球の極域電離圏（電離した大気で満たされた領域）と磁力線を通してつながっている。その境界領域はエネルギーが高くなっている。特に、このプラズマシートは、南と北のローブ領域の磁場で囲まれているため、エネルギーが高く、また、地球に向かう高速の流れが存在し、かつ密度粒子が高く、地球の極域電離圏で起こる様々な現象の源となっている。磁気圏境界と呼ばれる部分が、地球磁気圏の殻である。その内側において、尾部の子午面では、カスプからの延長のプラズママントルが、赤道面では、低緯度境界層が存在し、これらの領域では、100～150 km/s の粒子の流れが観測されている。この磁気圏は基本的に太陽風プラズマに対するシールドとして働いているが、いくつかの条件下では太陽風プラズマが磁気圏内に侵入することがある。その結果、例えば身近な現象としては極域地方でのオーロラ発光という形で表れる。より詳細な紹介については、参考文献[1]などを参考にしていただきたい。

宇宙プラズマ研究において、我々は主にこのような太陽から吹いてくる磁場を伴ったプラズマの風（太陽風）と惑星の磁場が相互作用して起こる様々な現象を研究ターゲットにしている。これらは宇宙空間で起きる現象であるため探査機を打ち上げて観測を行うが、基本的に“その

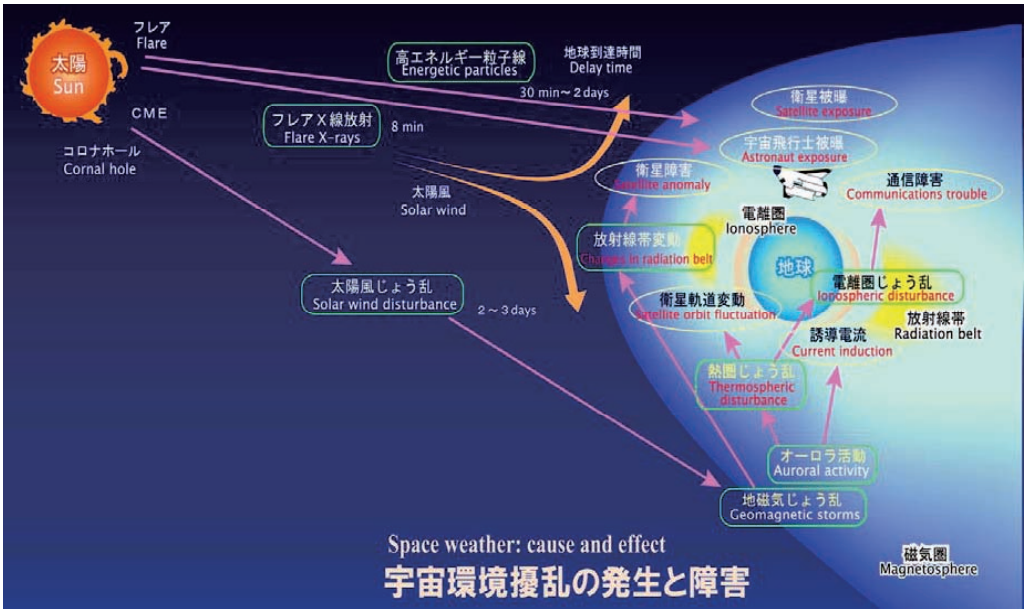
場”の観測しか行えない（立体空間情報を得ることができない）。そのため、３次元空間構造、さらにその時間発展などを調べることのできる宇宙プラズマ計算機シミュレーションがこの分野の理論の発展、また観測結果の理解の促進に非常に重要な役割を果たしてきている。

地球周辺の宇宙空間において、前述のような宇宙プラズマに起因する様々な現象を気象にならい、宇宙天気と呼んでいる。その宇宙天気と呼ばれる現象を第 2 図にまとめた。基本的にこれらの現象は太陽の活動により生じる。例えば、太陽表面でのフレア爆発、コロナから高速にガスの固まりが放出される現象（コロナ質量放出 : CME）等がある。それら太陽の活動の結果、身近なものとしては、衛星に障害が生じ、衛星放送に不具合が生じることや、国際宇宙ステーションなどで活動している宇宙飛行士が被爆することなどがある。また磁気圏より地球に近いところに存在する電離圏という領域では太陽活動により、電子密度が変動し、地上で GPS 衛星からの電波をうまく受信できないことも起こる。このような現象を引き起こす太陽活動だが、実はその活動度は１１年周期で変動している。つまり１１年毎に高い活動度（極大期）、低い活動度（極小期）を繰り返している。現在太陽活動が活発な時期に入ってきており、GPS に代表される様々な衛星や宇宙ステーションへ甚大な影響を及ぼし、不具合を引き起こす可能性が高まっている。



第１図：地球磁気圏の構造。

図の左側から超音速の太陽風が惑星間空間磁場を伴って吹いており、それが地球に達すると、地球の前面には衝撃波面が形成される。その衝撃波面より地球側に磁気圏の境界を表す磁気圏境界面が存在し、その中が磁気圏になる。磁気圏は惑星固有磁場の勢力範囲であり、太陽風から地球をシールドする働きを持つ。磁気圏内は、いろいろな領域に分けられており、それぞれ図中にあるような名前が付いている。



第２図: 様々な宇宙天気現象（情報通信研究機構提供）。

太陽で起きる、フレア、CME（コロナ質量放出）や太陽風により、様々な現象が地球周辺で起こる。例えば、高エネルギー粒子線であれば、宇宙空間にいる宇宙飛行士に被爆をもたらし、太陽風の擾乱により、磁気圏で擾乱が起こり、それに伴い地球周辺の様々な領域で擾乱が起こり、オーロラ活動が活発になったり、衛星の軌道に影響を与えたりする場合がある。日本の宇宙天気研究、予測は歴史的に独立行政法人情報通信研究機構で行われている。

このような宇宙天気現象は地球だけでなく、磁場を持つ木星や土星でも起こる。第１表に太陽系内の代表的な磁化惑星の特徴を示すが、惑星に固有磁場が存在すると、前述のように太陽風プラズマとの相互作用が起こり、磁気圏が形成される。そのため、木星、土星でもオーロラが観測されている（第３図を参照）。また第１表にあるように木星、土星は高速自転しており、地球とは異なった磁気圏構造をしていると考えられる。木星は巨大な磁場を持ち、更に磁気圏内に大量のプラズマを保持したまま高速自転しているために、磁気圏が遠心力によりディスク状に伸びていると考えられている（第４図参照）。また固有磁場が非常に強力なためその勢力範囲である磁気圏も非常に大きく、土星磁気圏が木星磁気圏の尾部（反太陽側）に入っている観測結果もある。土星はその特徴からよく地球と木星の中間の惑星と言われる。それは、地球程度の磁場しか無く、一方でガス惑星であり、木星と同様に高速自転し、大きさも太陽系では木星の次の規模であることからきている。しかしながら、最近の研究では２つの惑星の中間という特徴ではなく、土星固有の現象、特徴も見つかってきている。

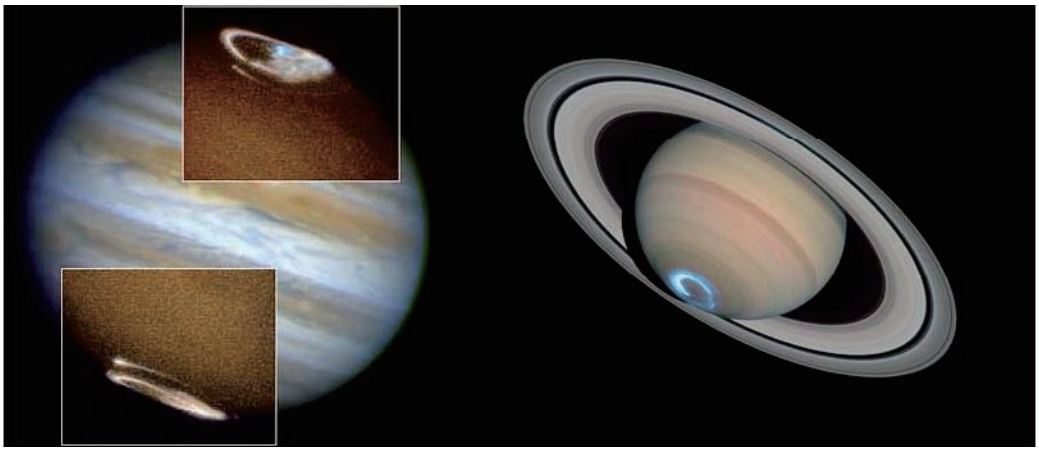
地球においては今までに、また現在もたくさんの衛星が磁気圏を観測し、宇宙天気現象について理解が進んできているが、その他の惑星ではそれほど進んでいない。それでも木星においては、今まで８つの探査機（Voyager 1、2、 Pioneer 10、11、 Ulysses、 Galileo、 Cassini、 New Horizon）が観測を行い、その中でも Galileo 探査機は１９９７年から２００２年まで木星を周回観測し、さまざまな情報を与えてくれた。我々のグループにおいてもこの Galileo 探査機の結果に対するシミュレーションを行い、その構造、現象の理解につなげている[2][3][4]。一方で土

星においては、今までに４機の探査機（Voyager 1、 Pioneer 10、11、Cassini）が観測を行っており、２００２年から今現在も Cassini 探査機が土星を周回観測している。Cassini 探査機は Galileo 探査機以上に様々な現象を観測しており、現在我々もその理解のため、シミュレーションを行っている[5][6]。第５図に我々の土星磁気圏シミュレーション結果を載せる。ここでは土星磁気圏赤道面におけるプラズマ対流の磁力線方向に対して平行の渦度、左下図で極域における沿磁力線電流を表している。数字が付いているプラズマ対流の渦構造が磁力線により極域沿磁力線構造とつながっていることが示されている。近年 Cassini 探査機がこの対流構造に似た観測を行い、シミュレーションの正当性が強くなってきている。

第１表: 地球、木星、土星の特徴。

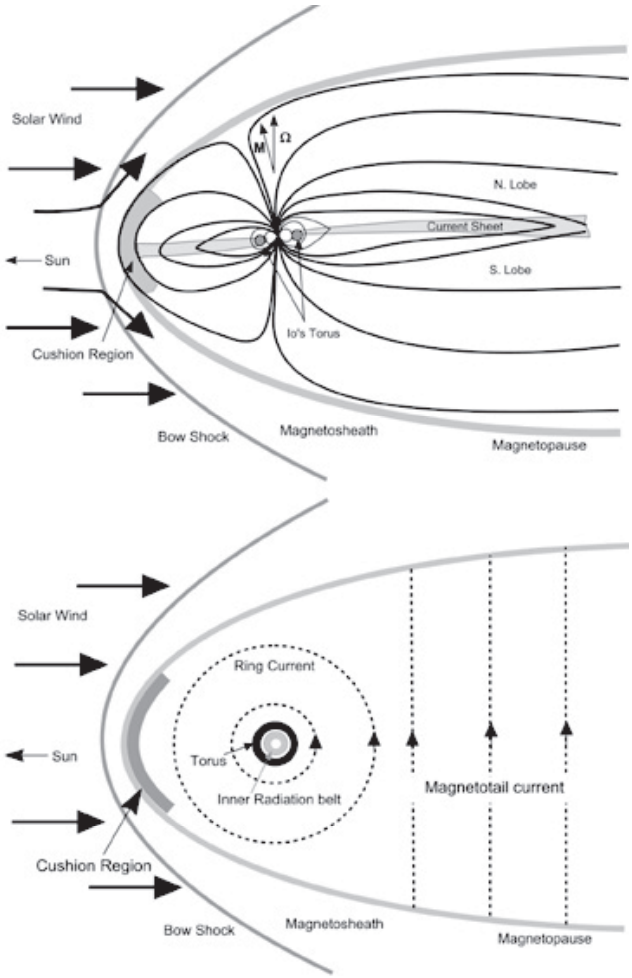
固有磁場は木星が飛び抜けて大きく、地球と土星ではそれほど変わらない。自転速度、赤道半径は木星、土星で同程度であるが、地球に比べると遙かに大きい。プラズマ源は各惑星磁気圏内に広がるプラズマの発生源を示している。イオ、エンケラダスは衛星であり、地球で言うところの月に当たるが、大気を持っている等、その特徴は異なる。木星はおよそ太陽から、地球と比べて 5 倍以上離れており、土星はさらに 9.5 倍離れている。

	固有磁場 [nT]	磁極	自転周期 [hr]	プラズマ源	赤道半径 [km]	太陽からの距離 [A.U.]
地球	31,000	N 極が南	24	電離圏	6,378	1
木星	420,000	N 極が北	10	イオ、電離圏	71,492	5.2
土星	21,000	N 極が北	10.65	エンケラダス、電離圏	60,268	9.55



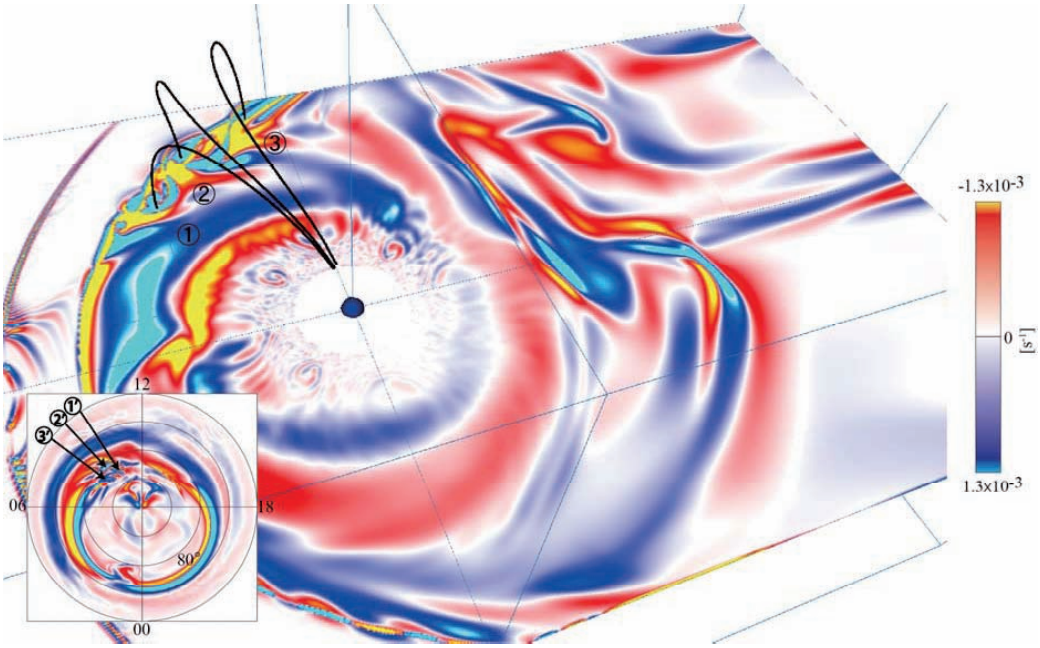
第 3 図: 木星と土星におけるオーロラ発光[NASA 提供]。

左側がハッブル宇宙望遠鏡で撮像された木星におけるオーロラ発光（紫外線領域）の写真であり、右図が土星におけるオーロラ (紫外線領域) の写真である。青白く光って見えるのは紫外線領域での撮像のためである。



第４図: 木星磁気圏の概略図[7]。

上図が子午面（縦に切った面）における木星磁気圏であり、遠心力によって磁力線が左右に伸びているのがわかる。この構造を磁気ディスクと呼ぶ。下図は赤道面における磁気圏構造を示している。木星の衛星イオから吹き出したプラズマが作るイオトーラスが木星の囲むように輪を描いている。これらは地球にはない木星の特徴である。これらによって木星磁気圏独特の性質ができあがっている。



第５図: 土星磁気圏シミュレーション結果[6]。

赤道面における磁場に垂直なプラズマ対流の渦度と極域における沿磁力線構造。左側から太陽風が吹いてきている。数字の付いた場所でプラズマ対流が乱れており、渦構造になっていることがわかる。その構造が沿磁力線電流の構造と磁力線によってつながっていることが図から分かる。

１．２　プロジェクトの目的

一般に磁気圏の大規模構造をシミュレーションする場合、プラズマを電磁流体と近似した、電磁流体力学（MagnetoHydroDynamics : MHD）シミュレーションを行う。このシミュレーションでは数値計算的な意味においてスケールフリーであり、解像度は計算機の資源によって決まる。現在の MHD シミュレーションでは、計算機の性能限界により、宇宙天気に影響を与えるプラズマの磁気圏界面における渦構造等を解像できない、MHD スケールからミクロスケールへの遷移領域が扱えないなど、解像度不足を指摘されている。これらを解像するためには 0.01 R_E (1 R_E は地球半径 ＝6,380 km)の解像度が必要と見積もられており、$15{,}000^3$ 以上のグリッドが必要となる。今回は実用化へ向けた超大規模並列計算の第一歩として FX10 で扱える 10,000×8,000×8,000 グリッドを利用した計算を行い、その性能評価を行う。前述のように惑星磁気圏は巨大で有り、太陽風との相互作用により構造が変わるため、準定常状態に達するまでに非常に時間がかかり、意味のある結果を得るためには、シミュレーションを長時間走らすことが必要となる。そのため、計算機の実行効率が非常に重要になってくる。

２．磁気圏シミュレーションモデルの概要と特徴

２．１　電磁流体方程式

宇宙プラズマの密度はとても低いために、その平均自由行程が非常に長くなる。例えば、太陽プラズマの平均自由行程は１天文単位（太陽と地球の距離）にも達する。そのため宇宙プラ

ズマは基本的に衝突が無いと見なされる。その無衝突プラズマの振る舞いは以下の Vlasov（無衝突 Boltzmann）方程式によって記述される。

$$\frac{\partial f_s}{\partial t}+\vec{v}\frac{\partial f_s}{\partial \vec{r}}+\frac{q_s}{m_s}(\vec{E}+\vec{v}\times\vec{B})\frac{\partial f_s}{\partial \vec{v}}=0 \tag{1}$$

ここで $\vec{E}$ 、$\vec{B}$ 、$\vec{r}$ と $\vec{v}$ はそれぞれ電場、磁場、距離、速度を表す。また、$f_s(\vec{r},\vec{v}_s,t)$ は位置-速度位相空間における分布関数であり、s はイオンや電子など種類を示す。q_s は電荷を m_s は質量を表す。電磁場の時空間発展は以下のように Maxwell 方程式によって記述される。

$$\begin{aligned}
&\nabla\times\vec{B}=\mu_0\vec{J}+\frac{1}{c^2}\frac{\partial\vec{E}}{\partial t}\\
&\nabla\times\vec{E}=-\frac{\partial\vec{B}}{\partial t}\\
&\nabla\cdot\vec{E}=\frac{\rho}{\varepsilon_0}\\
&\nabla\cdot\vec{B}=0
\end{aligned} \tag{2}$$

ここで $\vec{J}$ は電流密度、ρ は電荷密度、μ_0 は真空中の透磁率、ε_0 は真空中の誘電率、c は光速を示す。電流密度 $\vec{J}$ は帯電した粒子の動きによって以下のように記述できる。

$$\vec{J}=\sum_s q_s\int f_s\vec{v}d\vec{v} \tag{3}$$

また、電流密度 $\vec{J}$ は以下の電荷保存則を満たしている。

$$\frac{\partial\rho}{\partial t}+\nabla\cdot\vec{J}=0 \tag{4}$$

ここまでにあげた式が無衝突プラズマの第一原理の方程式である。

次に電磁流体力学（MHD）方程式を求めていく。MHD 方程式は Vlasov 方程式(1)の０次、１次、２次のモーメントをとり、運動論的効果を無視することで得られる。まず Vlasov 方程式の０次のモーメントをとる（Vlasov 方程式(1)を速度空間について積分する）と連続の式が求まる。

$$\frac{\partial n_s}{\partial t}+\nabla\cdot(n_s\vec{u}_s)=0 \tag{5}$$

$\vec{u}_s$ は各プラズマの平均速度を表す。次に Vlasov 方程式(1)に $m\vec{v}$ を掛けてから速度空間で積分する（１次のモーメントをとる）と、運動方程式が求まる。

$$\frac{\partial}{\partial t}(m_s n_s\vec{u}_s)+\nabla\cdot(m_s n_s\vec{u}_s\vec{u}_s+\vec{P}_s)-\rho_s\vec{E}-\vec{J}_s\times\vec{B}=0 \tag{6}$$

$\vec{P}_s$ は圧力テンソルを表す。更に、粒子の運動エネルギー $\frac{1}{2}m|\vec{v}|^2$ を Vlasov 方程式(1)に掛けて速度空間で積分する（２次のモーメントをとる）と、エネルギー方程式が求まる。

$$\frac{\partial}{\partial t}(\frac{1}{2}m_s n_s|\vec{u}|_s^2+\frac{3}{2}p_s)+\nabla\cdot(\frac{1}{2}m_s n_s|\vec{u}|_s^2\vec{u}_s+\frac{3}{2}p_s\vec{u}_s+\vec{P}_s\cdot\vec{u}_s+\vec{h}_s)-\vec{E}\cdot\vec{J}_s=0 \tag{7}$$

p_s は $p_s \equiv \frac{1}{3}\sum_{i=1\sim3} P_{i,i,s}$ で定義される圧力である。$\vec{h}_s$ は熱流束密度である。ここで以下の操作を行って、単一流体近似をこれらの流体方程式に適応すると、

$$\sum_s m_s n_s \equiv \xi、\frac{\sum_s m_s \vec{u}_s}{\sum_s m_s} \equiv \vec{U}、\sum_s \rho_s \equiv 0、\sum_s \vec{J}_s \equiv \vec{J} \tag{8}$$

さらに、いくつかのテンソル計算を行って、以下の式を得る。

$$\frac{\partial \xi}{\partial t} + \nabla \cdot (\xi \vec{U}) = 0 \tag{9}$$

$$\xi \frac{\partial \vec{U}}{\partial t} + \xi(\vec{U} \cdot \nabla)\vec{U} + \nabla p - \vec{J} \times \vec{B} = 0 \tag{10}$$

$$\frac{\partial p}{\partial t} + (\vec{U} \cdot \nabla) p + \gamma p \nabla \cdot \vec{U} = 0 \tag{11}$$

ここで $\gamma = \frac{5}{3}$ は比熱比であり、等方対角テンソル $\sum_s \vec{P}_s \equiv pI$（$I$ は単位テンソル）と仮定したため、熱流束密度は無視できる。更に、磁場凍結条件により、

$$\vec{E} + \vec{U} \times \vec{B} = 0 \tag{12}$$

が求まり、Darwin 近似を用いて、

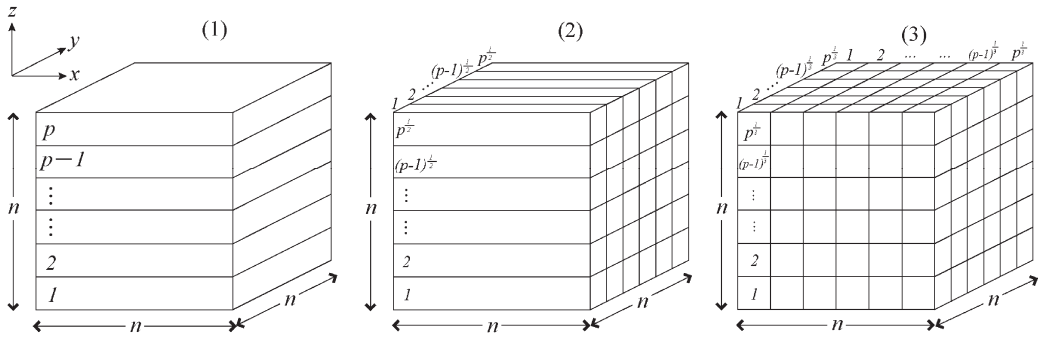
$$\nabla \times \vec{B} = \mu_0 \vec{J} \tag{13}$$

が求まる。これら(9)～(13)式が MHD 方程式となる。より詳細な導出は参考文献[9]を参考にされたい。

２．２　シミュレーションモデル

２．２．１　MHD モデル

MHD 方程式を解く数値計算法としては、Ogino et al. [10]によって開発された Modified Leap Frog 法を使用する。これは最初の 1 回を two step Lax-Wendroff 法で解き、続く（l − 1）回を leap-frog 法で解き、その一連の手続きを繰り返す。l の値は数値的に安定の範囲で大きい方が望ましいので、1 次精度の中心空間差分を採用するとき、数値精度の線形計算と予備的シミュレーションから $l = 8$ に選んでいる。Modified leap frog 法は、two step Lax-Wendroff 法の数値的安定化効果を一部取り入れて、leap frog 法の数値的減衰と分散の小さい効果をより多く取り入れた、数値的減衰と分散にバランスの良くとれた一種の組み合わせ計算方法となっている。また、パラメータを変化させることによって、性質の良く分かった２つの計算方法に一致させることができるので、結果に与える数値誤差の影響も理解し易い利点を持っている。更にこの手法を用いた計算で、今まで様々な計算機で性能評価を行ってきたこともあり、同様の手法をもちいることで、過去の結果と比較できる利点もある。



第６図: ３種類の領域分割法。

左から 1 次元領域分割、2 次元領域分割、3 次元領域分割の概要図を示す。n^3 の配列を並列 p で分割している。全並列数を p としているため、2 次元領域分割では各次元で $p^{1/2}$ 並列、3 次元領域分割の場合、x, y, z 方向に $p^{1/3}$ 並列を適用している[11]。

並列化にはMPIを使用する。並列化手法としては3次元空間を分割する領域分割法を用いる。領域分割には、第６図に示すように、１次元、２次元、３次元分割が考えられ、本性能評価ではこれらすべての評価を行う。分散メモリ型の並列計算機を用いた並列計算では、３次元配列に対して領域分割を用いるのが通常である。３次元モデルの場合、領域分割の次元を１次元、２次元、３次元に選ぶことができる。その場合の計算時間(T_{S1}, T_{S2}, T_{S3})と通信時間(T_{C1}, T_{C2}, T_{C3})は大まかに次の様に見積もることができる。

i) １次元領域分割

$$T_{S1} = \frac{k_1 n^3}{p}、\quad T_{C1} = k_2 n^2 (p-1) \tag{14}$$

ii) ２次元領域分割

$$T_{S2} = \frac{k_1 n^3}{p}、\quad T_{C2} = 2k_2 n^2 (p^{\frac{1}{2}} - 1) \tag{15}$$

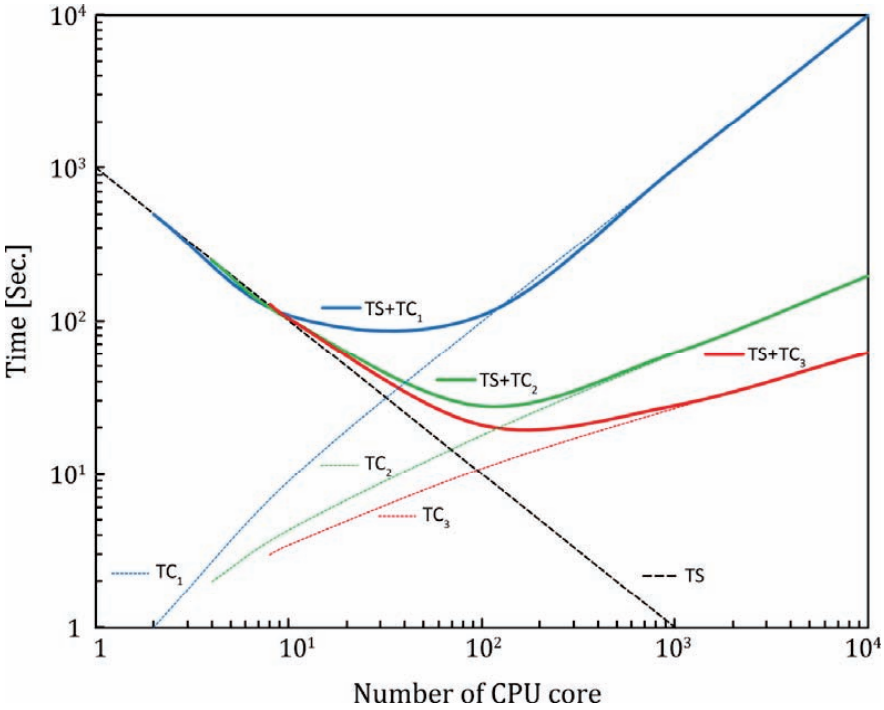
iii) ３次元領域分割

$$T_{S3} = \frac{k_1 n^3}{p}、\quad T_{C3} = 3k_2 n^2 (p^{\frac{1}{3}} - 1) \tag{16}$$

ここに、k_1 と k_2 は一定の係数、n は３次元配列における１方向の変数量、p は総並列数である。ここでは簡単のため、領域分割は第５図に示すように x, y, z 方向に同じ数(n)の配列を使用し、各次元を並列化する場合は等しい並列化数を設定している。計算時間と通信時間の和が並列計算に要する時間と考えられる。これらの式を第７図にグラフで示した。式、図より明らかに計算時間 T_{S1} T_{S2} T_{S3} は並列数 p に反比例して短くなるが、通信時間 T_{C1}, T_{C2}, T_{C3} は p の増加に伴って長くなる。 しかし、その通信時間の長くなる様子は、T_{C1}, T_{C2}, T_{C3} によって大きく異なる（第７図参照)。即ち、３次元領域分割が最も通信時間を短くでき、また、１次元と２次元領域分割の間でも通信時間の差は大きくなることが理解できる。ただし、この比較では簡単のため、通信時間を決める係数 k_2 が同じであると仮定した。これは通信部分のプログラムの工夫によりある程度小さくすることが可能である。こうして、スカラ並列機では３次元領域分割が、一方ベクトル並列機では、１つの次元方向はベクトル化に利用する必要があるために２次元領域分割

が最も効率的であろうと予想できる。

スカラ機で性能を出すにはキャッシュの有効活用が重要である。基本的な動作としてはデータアクセス時に、その前後含めて数 KB のデータをキャッシュに格納する。キャッシュの量や、一度にキャッシュに格納するデータ量は CPU アーキテクチャ毎に変わるので、最高のパフォーマンスを出すにはそれぞれの調整が必要である。MHD シミュレーションにおいては、物理変数がプラズマ密度、速度 3 成分、圧力、磁場 3 成分の計 8 変数となる。そのため、配列を u(i,j,k,m) と定義し（Type A）、m = 8 としている。数値計算時に、同じ場所の物理変数を何度も使うことになるので、一般に u(m,i,j,k,)と定義した方がキャッシュヒット率は上がると考えられる（Type B）。そのため、本性能評価においてもこの配列定義を使った評価も行う。



第 7 図:並列数に対する各領域分割における計算時間の変化[11]。

式(14)～(16)によって描かれたグラフ。横軸は並列数、縦軸は時間を示す。点線はそれぞれ計算時間、通信時間を示し、実線が各領域分割にかかる時間（T_S+T_C）を示している。ここでは簡単のため、k_1=1、k_2=0.01 としている。k_n の値に依るが、ある並列数から、計算時間の上昇が見られる。しかしながら、３次元領域分割の場合、その上昇を最小限に抑えられている。

３．性能評価結果

３．１　MHD シミュレーション

ベンチマークとして MHD シミュレーションでは 1 GB/node サイズの配列を計算する weak scaling だが、MHD 方程式を Modified Leap Frog 法で解くためのワーク配列として 5 GB/node を追加で使用する。この設定で１次元、２次元、３次元領域分割（配列並び２種類）の性能測定を行った。基本的に１次元領域分割は、小並列のベクトル計算機に最適な並列化手法と言われ

ている。ベクトル化のためにdoループを長くとる取る必要があったためと、並列化数が少ない分、通信コストも低いので、次に述べる２次元、３次元領域分割の場合のように通信用のバッファーにデータを集める動作の方が、コストがかかるからである。この手法はFlat MPIを使う上では超並列計算には対応していない（並列化される次元の配列数を並列化数が越えてしまうため）が、自動並列やOpenMPを併用することで、ある程度の並列計算に対応することは可能である。今回の性能評価では科学的に意味のある計算ではなく、ある次元の配列数を増やしていくことで、１次元領域分割を評価した。１次元領域分割は昔のベクトル計算機（富士通VPP5000、NEC SX-6など）によく使われていた手法で、今回１次元領域分割の評価に使用した我々のコードベクトル計算機向けに最適化されたコードでもある。

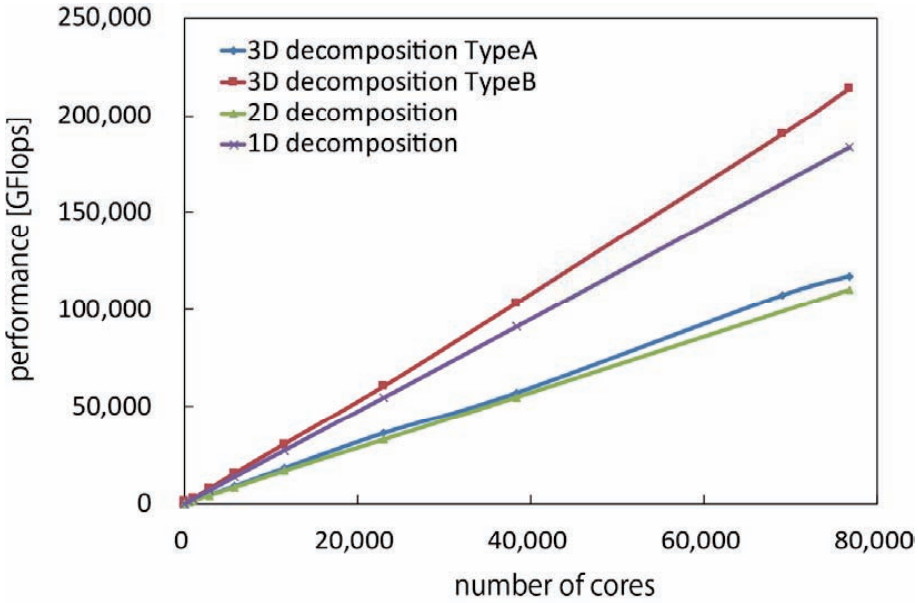
２次元領域分割は１次元領域分割と同様にベクトル計算機向けの手法と考えられているが、特に並列化数の多いベクトル計算機向けである。理由は１次元領域分割の場合とベクトル化向けにある次元向けのdoループを長くとることができることと、並列化数が多くなりすぎて、通信コストが無視できなくなっているからである。この場合には、並列化プロセスで通信データをまとめて、各プロセス間で一気に通信する方がコストを低くできると考えられる。この手法は地球シミュレータで特に有効で、２次元領域分割の評価に使うコードは地球シミュレータで最適化されたコードでもある。

３次元領域分割は、１次元、２次元領域分割と比べて、並列化数の上昇に対する通信コストが低いために、超並列計算機向けの手法と言われる。３次元領域をすべて並列化するために、基本的にはスカラ計算機向けの手法である。今までにスカラ計算機（富士通HPC2500、FX1、RS200S6、日立SR16000/L2、HA8000 など）を利用し、３次元領域分割による最適化を行い良い性能を得ている[12]。

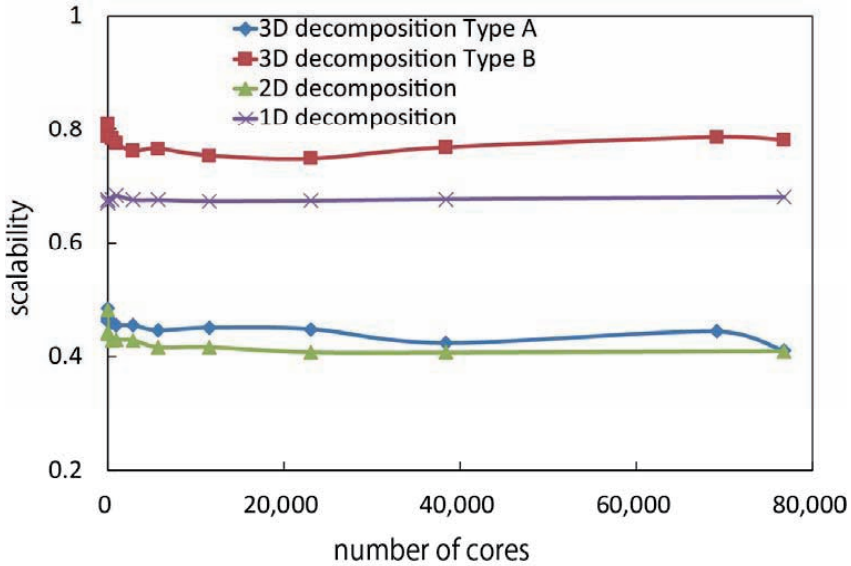
第８図にそれぞれの並列化数（使用core数）に対する実効性能、第９図に並列化効率を示している。これらはFlat MPIでの性能評価となる。4,800ノード（76,800コア）利用時の実効性能では３次元領域分割Type B（並びを入れ替えた配列）が最も良く（約210 TFlops）、その次に１次元領域分割が良く（約180 TFlops）、２次元、３次元領域分割TypeAはそれらに比べて、かなり性能が悪くなった（約110 TFlops）。このように並列の分割法、配列の並び方が違うだけで、２倍の性能差が見えた。実行効率としては最高で約19 %となった。この結果は今までのSPARC系の結果、POWER系の結果と同じ傾向である。これはキャッシュヒットを上げることが、これらのCPUでは性能を上げる上で重要であることを示している。一方で、スカラCPUであるXeon系、Opteron系では、２次元、３次元領域分割の性能が高くなることが分かっている[12]。これらではSSE系によるベクトル化率を上げることが性能向上に重要なことを示している。

並列化効率ではどの並列化手法においても、フラットな結果になっており、高並列数になることでの性能劣化は見られなかった。しかし、ノードを越えた並列化を行った時点で、１次元を除くすべてで急に並列化率が下がっている。この原因を調べるために、通信に伴う操作（通信データのパック/アンパックとMPI通信）を含まない３次元領域分割TypeBの性能を測定すると、実行効率が約25%になり、MPI通信だけを除く（パック/アンパックは含む）場合を測定すると実行効率が22 %となり、データをまとめるだけで2 %の実行効率を失っていることが分かった。これはデータ並びが飛び飛びになっている配列成分のコピーのため、ストライドアクセスになっており、効率が出ないことが原因となっていた。この影響を最低限に抑えるために

通信を非同期にし、パック/アンパックと最大限にオーバーラップさせると、実行効率が 21 %になり、約 230 TFlops を達成することが出来た。このコードを利用し、10,000×8,000×8,000 グリッドで計算を行い、実際に計算が行えること、同様の性能が出ることを確認した。



第 8 図:並列数に対する各領域分割の実効性能。



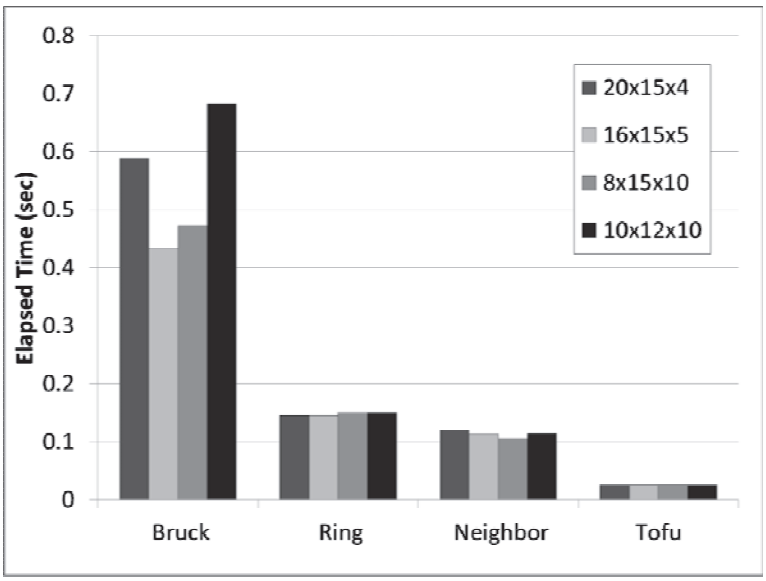
第 9 図:並列数に対する各領域分割の並列化効率。

３．２　ノード間通信性能評価

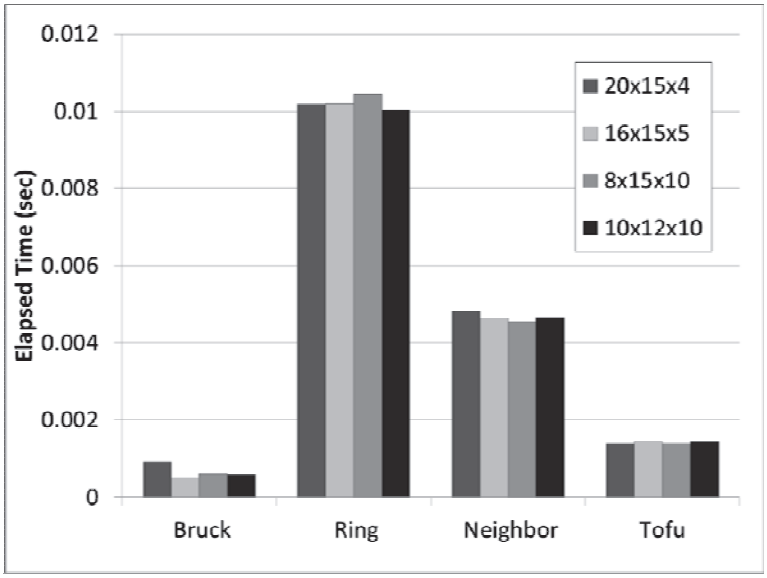
FX10 では Tofu インターコネクトが導入されており、ノード間通信トポロジーも考慮する必要があるため、ノード間の集団通信アルゴリズムの形状による性能変動調査も行った。実験に用いたプログラムは、同じ集団通信を 5 回呼び出すプログラムで、それぞれについて所要時間を計測した。プログラムは C 言語と MPI で書かれており、実行時のノードあたりプロセス数は 1 とした。また、今回の実験では、使用ノード数を 1200 とし、同じノード数に対して、20x15x4、16x15x5、8x15x10、16x12x10 という３次元形状を指定した。

各プロセスの所有データ量が 2MB と 2KB の場合の、Allgather、Allreduce、Alltoall の各アルゴリズムの所要時間の形状による変化を、図１０～１４にそれぞれ示す。各グラフの横軸は集団通信アルゴリズムである。なお, Tofu とは、Tofu 向けに開発された専用アルゴリズムである。どの集団通信でも、形状による性能への影響はアルゴリズムによって異なる。ここで、形状によってアルゴリズムの性能が変動するのは通信衝突が原因であると考えられる。また、衝突の発生はアルゴリズムに依存するため、性能変動の度合いはアルゴリズムによって異なる。この事から、集団通信のアルゴリズム選択においては、形状とアルゴリズムに応じた衝突の影響予測が重要であることが分かる。

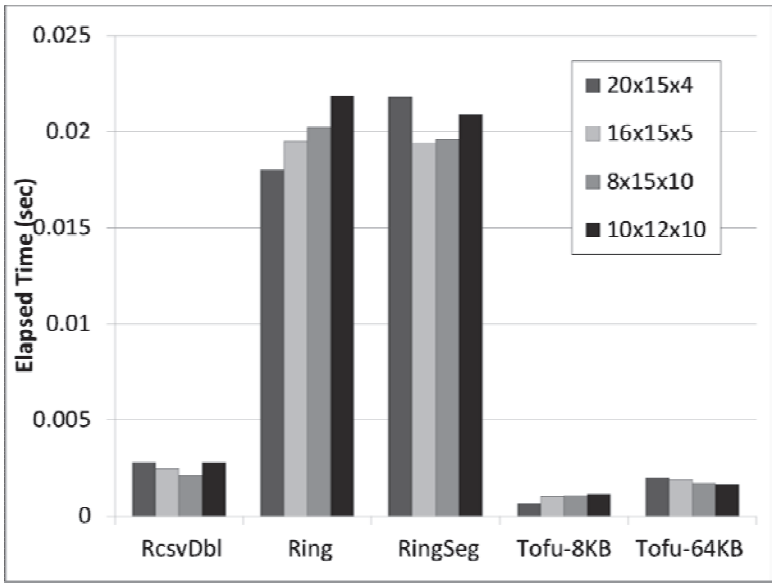
また、どの集団通信でも Tofu 専用アルゴリズムは、メッセージサイズが大きい場合、他のアルゴリズムより高速である。これは、これらのアルゴリズムが主に隣接ノードとの通信で構成されており、衝突が発生しにくくなっているためである。一方、メッセージサイズが小さい場合、Bruck や Recursive Doubling のように通信回数の少ないアルゴリズムの方が高速になる。これは、小さいメッセージサイズでは衝突の影響が少なくなり、一回の通信当たりに必要となる遅延時間が問題となるためである。そのため、状況に応じて適切なアルゴリズムを選択する技術が必要である。



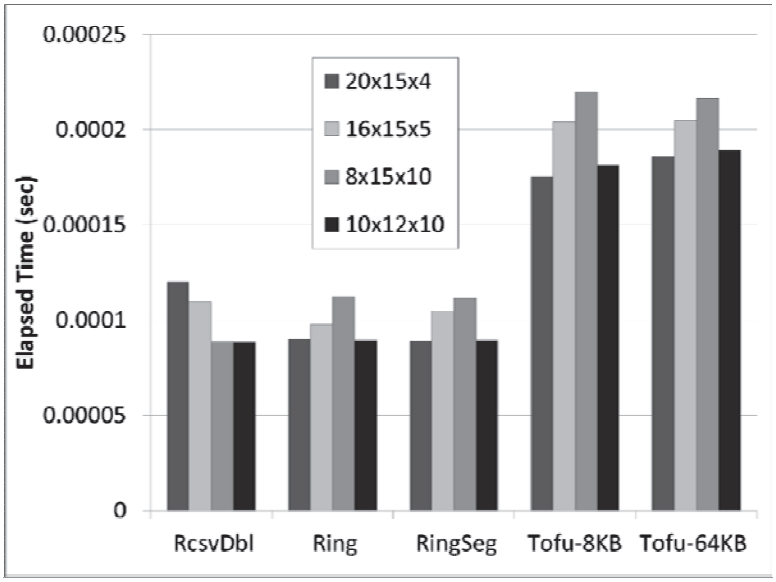
第１０図: Allgather 2MB。



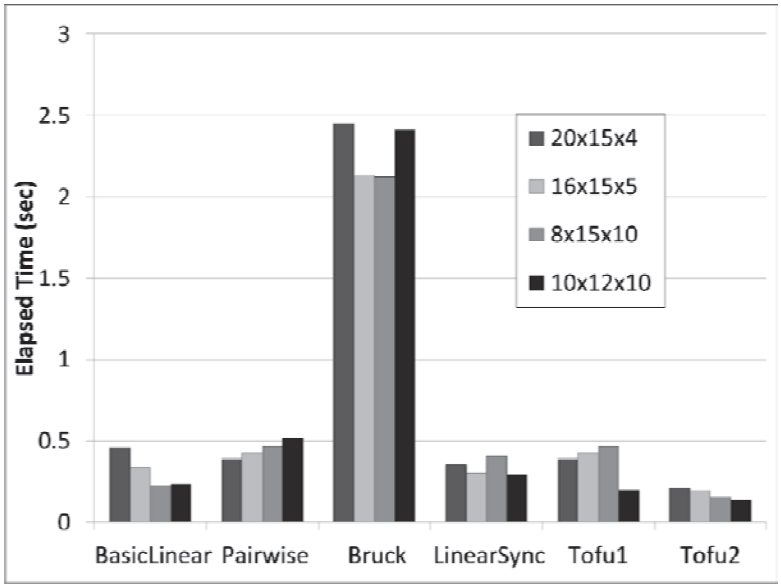
第１１図: Allgather 2KB。



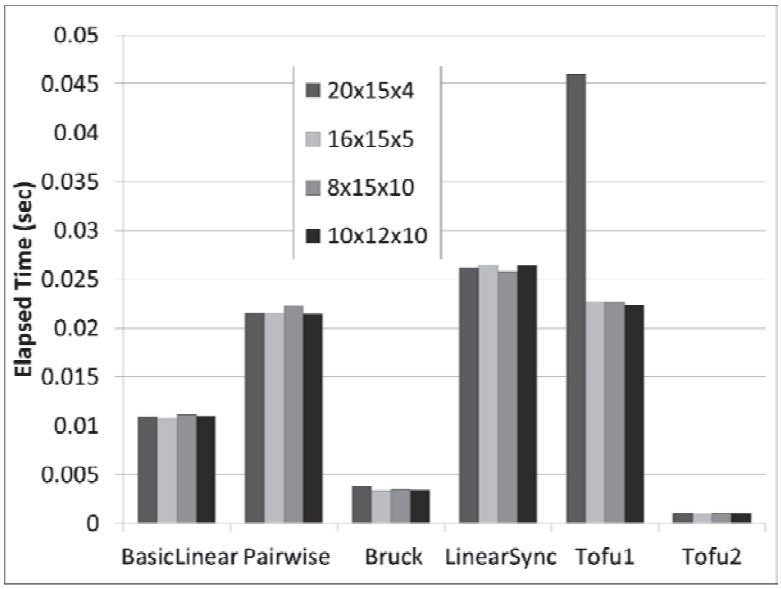
第１２図: Allreduce 2MB。



第１３図: Allreduce 2KB。



第１４図: Alltoall 2MB。



第１５図: Alltoall 2KB。

４．まとめと問題点

本研究では東京大学 FX10 4800 ノードを利用し、惑星磁気圏シミュレーションを行う電磁流体力学（MHD）コードの性能評価、Tofu ノード間通信の性能評価を行った。この惑星磁気圏シミュレーションは宇宙天気と呼ばれる地球周辺の電磁気環境などの変動を調べる研究で利用されている。MHD コードでは 3 種類の領域分割と配列の並びを変更したコードで性能評価を行った。その結果、配列の並びを変更した３次元領域分割の時に最高の実効性能を得ることが出来た。並列化効率では並列数の増加による性能低下は見られなかったが、ノード間並列を行うと急に性能が下がることが分かった。その原因は通信回数を減らすために行っているデータのパック/アンパックであった。今回はそのパック/アンパックと通信をできるだけオーバーラップさせることで、性能低下を出来るだけ抑え、実効性能 230 TFlops、実行効率 21 %を達成することが出来た。

ノード間の通信性能評価として、ノード間の集団通信アルゴリズムの形状による性能変動調査を行い、通信性能が集団通信のアルゴリズムにより変わることが分かり、それは通信衝突によるものだと分かった。このように超並列計算時では状況に応じて適切なアルゴリズムを選択する技術が必要である。

問題点としては、低並列時では OpenMP と MPI でスレッド並列した場合は実行効率で 22%を達成しているが、4800 ノードでは 17 %の実行効率しか出ていないこと、データのパック/アンパックの高速化が挙げられる。スレッド並列の問題はどこが原因かまだ分かっていないが、九州大学の FX10 も利用しながら、調べていく。また 1PFlops クラスの計算であれば、データのパック/アンパックで変わってくる実行効率 2%は非常に大きい性能の違い（20TFlops）であり、これからのエクサ級時代に向けて、解決するべき問題である。一方で、この問題は、x86 系ではそれほど見えないので、SPARC 系のチューニング手法の 1 つとして、調べていきたい。

参 考 文 献

[1] Margaret G. Kivelson，Christopher T. Russell 編『Introduction to space physics』，Cambridge University Press，1995.

[2] Ogino, T., R. J. Walker, and M. G. Kivelson, A global magnetohydrodynamic simulation of the Jovian magnetosphere, *J. Geophys. Res., 103*, 225, 1998.

[3] Fukazawa, K., T. Ogino, and R.J. Walker, “Dynamics of the Jovian magnetosphere for northward interplanetary magnetic field (IMF)”, Geophys. Res. Lett., 32, doi:10.1029/2004GL021392, 2005.

[4] Fukazawa, K., T. Ogino, and R.J. Walker, "The Configuration and Dynamics of the Jovian Magnetosphere", J. Geophys. Res., 111, A10207, 2006.

[5] Fukazawa, K., T. Ogino, and R.J. Walker, "Magnetospheric Convection at Saturn as a Function of IMF B_Z", Geophys. Res. Lett., 34, L01105, 2007a.

[6] Fukazawa, K., T. Ogino, and R.J. Walker, "Vortex-associated reconnection for northward IMF in the Kronian magnetosphere", Geophys. Res. Lett., 34, L23201, 2007b.

[7] Khurana, K. K., M. G. Kivelson, V. M. Vasyliunas, N. Krupp, J. Woch, A. Lagg, B. H. Mauk and W. S. Kurth, The Configuration of Jupiter's Magnetosphere, In:Bagenal, F., T. Dowling and W. McKinnon (Ed.), Jupiter, Cambridge University Press, New York, 2004.

[8] 様々なスパコンにおける MHD コードの実効性能.
http://center.stelab.nagoya-u.ac.jp/web1/simulation/hpfja/comput04.html

[9] R. O. Dendy，『Plasma Dynamics』，Oxford University Press，1990.

[10] T. Ogino, R. J. Walker, M. Ashour-Abdalla, A global magnetohydrodynamic simulation of the magnetopause when the interplanetary magnetic field is northward, IEEE Trans. Plasma Sci.20, 817.828, 1992.

[11] Fukazawa, K., T. Umeda, T. Miyoshi, N. Terada, Y. Matsumoto, and T. Ogino, Performance measurement of magneto-hydro-dynamic code for space plasma on the various scalar type supercomputer systems, IEEE Transactions on Plasma Science, Vol. 38, No. 9, pp2254, 2010.

[12] Fukazawa, K., T. Umeda, Performance measurement of magnetohydrodynamic code for space plasma on the typical scalar type supercomputer systems with the large number of cores, International Journal of High Performance Computing, doi:10.1177/1094342011434813, 2012.

２次元フラストレート系の計算科学的研究

中野博生

兵庫県立大学大学院物質理学研究科

藤堂眞治

東京大学物性研究所

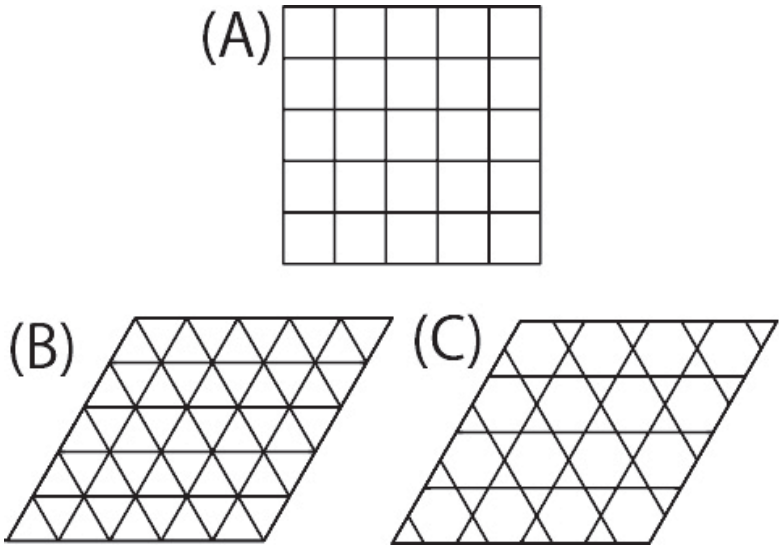
坂井徹

原子力研究開発機構

１．背景

「あちらを立てればこちらが立たず」は，人間関係の複雑さと難しさを言い表したことわざであるが，その状況は人間社会に限らない。自然現象でも同じようなことはしばしば起こり，その典型的な事例にフラストレート磁性体がある。

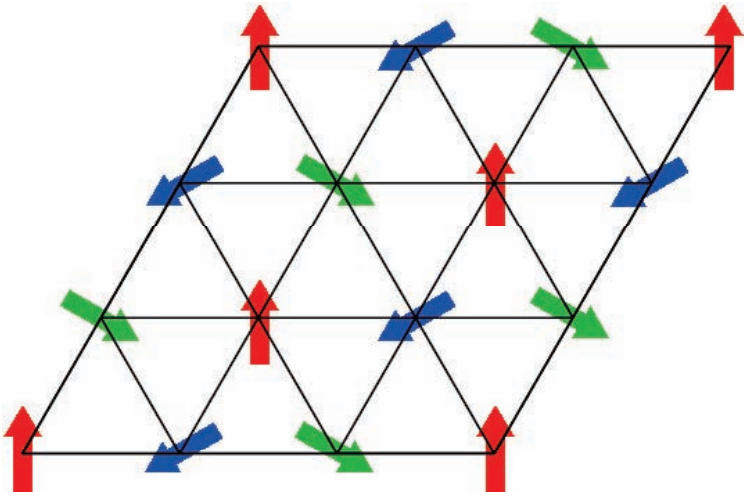
磁性体は一般に，小さな磁石の集まりと見なせる。小さな磁石の１個ずつは，電子が持つ特性の一つであるスピンが担う。電子は，物質中に事実上無限個と言ってよいくらい多数存在しており，互いに影響を受け合いながら全体として状態がどのような振舞いを示すか，ということで磁性体の性質が決まる。絶縁性磁性体物質は，電子が何らかの理由で動きまわることが出来ずに，固体中の原子配置で決まる特定の箇所に局在化した結果として出来たものである。電子自体が動きまわることが出来なくても，スピンの向きに関する自由度は残り，スピンの向きの状況によって絶縁性磁性体の性質が決まる。このような系は量子スピン系と呼ばれている。固体の原子配置には様々な格子形状があり，２次元系では正方格子が一番シンプルなものであろうが，局所的な形状に正方形でなく三角形を含む物質がある。この三角形の頂点にはスピン



第１図：(A)正方格子，(B) 三角格子，(C) カゴメ格子。

があって，隣接スピン間に互いに反対向きになる方がエネルギーを得するような相互作用（反強磁性相互作用と呼ばれる。）が働く場合，上述の「あちらを立てれば…」の状況が実現する。こういった三角形を基本構造として，何らかの規則で空間に配置することで出来るのがフラストレート磁性体である。正三角形が，隣接ペア同士で辺を共有する形で平面を敷き詰める規則で構成されるものが三角格子と呼ばれる格子である(第１図参照)。 他にも，隣接正三角形が頂点を共有して平面を敷き詰めることで得られるカゴメ格子や，3 次元的に配置されたパイロクロア格子も良く知られている。

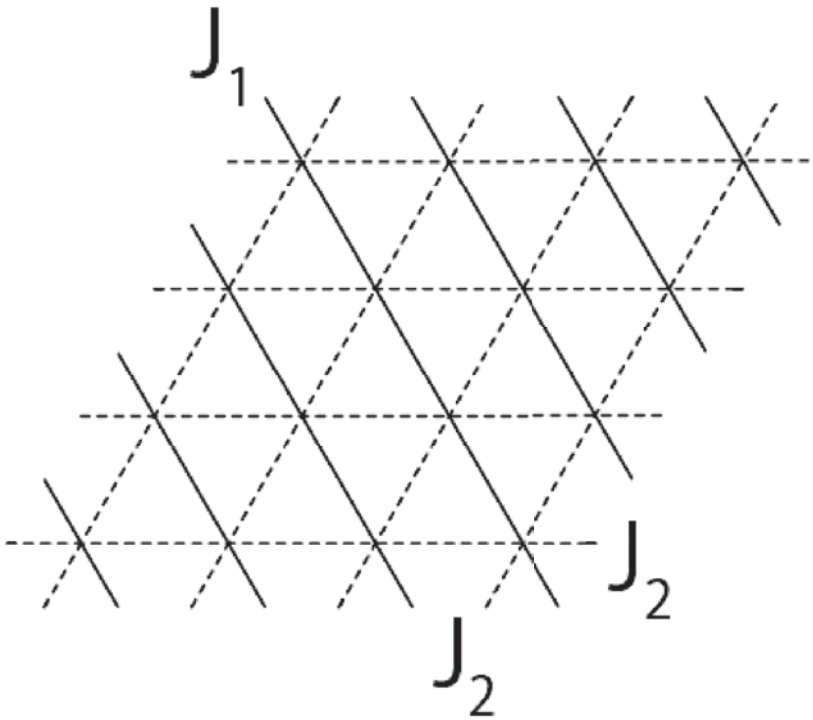
そのような三角格子反強磁性体は，その基底状態が「スピン液体状態」となっている可能性を P. W. Anderson が約 40 年前に指摘[1]して以降，理論と実験の両面から精力的に研究が行われてきた。その結果，今では，後で説明するハイゼンベルクスピンで構成される反強磁性体の基底状態は，Anderson の予想は的中しなかったが，第 2 図で示した１２０度構造とも呼ばれる 3 副格子構造の長距離秩序状態が実現していると広く信じられている。冒頭のことわざの後には，ことわざ辞典を見ると「双方立てれば身が立たぬ」が続く。磁性体の中では「身が持たぬ」という訳にはいかず，各々の隣接スピンペアが少しずつ妥協した状況を作りだす。その結果として最終的に微妙なバランスのもとに選ばれる状態がこの１２０度構造の状態である。そのような状態は，落語に登場する，いわゆる「三方一両損」の大岡裁きに似ている。



第 2 図：三角格子反強磁性体の 3 副格子構造（120 度構造）の概念図。

量子スピン系は，それを記述する理論模型の解が数学的に厳密な形で得られる例外的な場合もあるが，一般には解析困難な多体問題の典型である。三角格子反強磁性体に対して理論的研究[2-7]が盛んに行われてきたと述べたが，それらの研究の多くは，多体問題の困難さのために何らかの近似を用いた計算に立脚している。近似を用いれば，解析困難な問題も取り扱いが可能となるが，一方で，近似に由来する効果が現れてしまい，真の姿とは異なる様子を見ている可能性を完全に避けることはできない。それを解決する手段として，量子力学的効果を丸ごと取り込んだ直接数値シミュレーションがあり，その方法に立脚した研究ももちろん行われているものの，様々な制約から十分な知見を得ることが難しい状況にある。量子スピン系で良く使われる直接シミュレーションの方法には，量子モンテカルロ法，密度行列繰り込み群法，数値

的厳密対角化法があり，それぞれ一長一短の特徴を持つ。後で具体的に言及するが，量子モンテカルロ法と密度行列繰り込み群法は 2 次元フラストレート系の計算にはあまり有効ではない。それに対し，厳密対角化法は 2 次元フラストレート系に有効で，その計算によって我々は様々なフラストレート磁性体の性質解明を進めてきた[8-12]。本報告では，上述の三角格子 S=1 ハイゼンベルク反強磁性体が空間異方性を有する相互作用を持つ場合（第 3 図）に得られた研究成果[13]について紹介する。



第 3 図：空間異方性を持つ三角格子と反強磁性相互作用。

２．模型

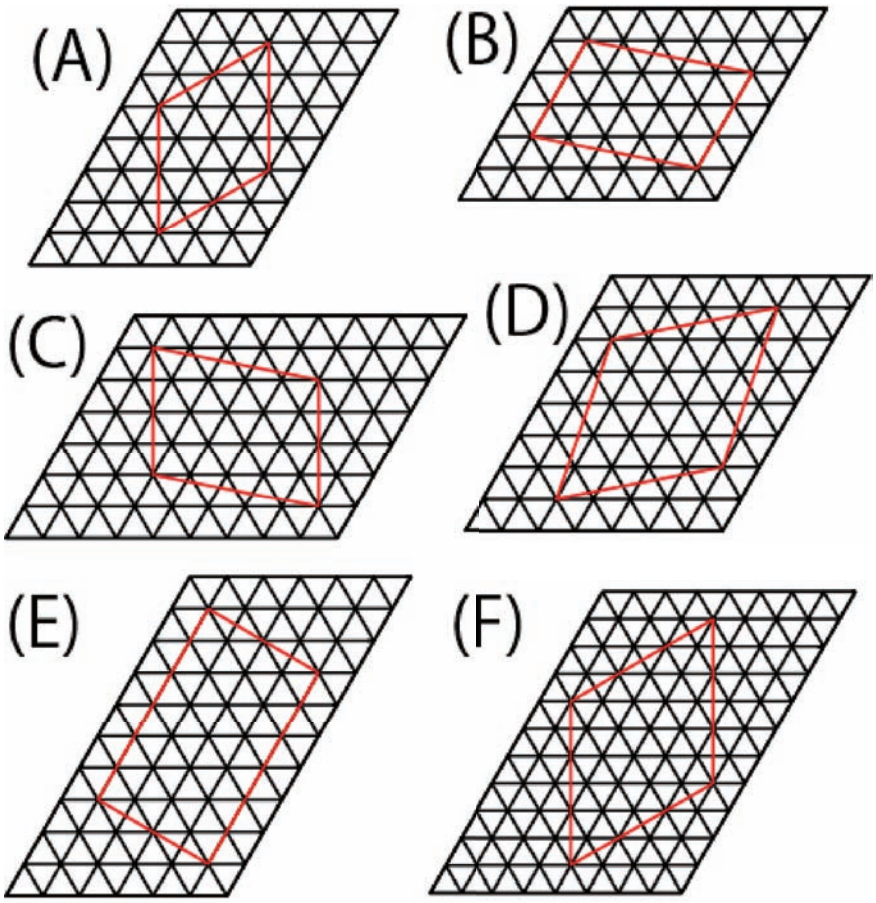
２－１．ハミルトニアン

我々が調べた，異方的三角格子上の S=1 ハイゼンベルク反強磁性体を記述するハミルトニアンは，次の式で与えられる。

$$\mathcal{H} = J_1 \sum_{\langle i,j \rangle} \boldsymbol{S}_i \cdot \boldsymbol{S}_j + J_2 \sum_{[i,j]} \boldsymbol{S}_i \cdot \boldsymbol{S}_j$$

ここで，$\boldsymbol{S}_i$はサイト i での S=1 スピン演算子であり，第 3 図の格子点に乗っている。隣接スピン間の相互作用の強さ $J_1(>0)$ と $J_2(>0)$ のボンドも第 3 図の実線と破線で示した位置にあるものとする。有限サイズクラスターは N_s 個のスピンを含むものとし，かつ，その N_s は 3 の倍数に限る。なお，J_2=0 の時に J_1 のボンドだけの 3 サイト系の正三角形クラスターが出来てしまう場合は除外するものとする。可能な有限サイズクラスターは，等方的な場合に第 4 図として示した。

N_s=12, 21, 27 では 120 度の内角を持つ菱形を取ることが出来て，有限サイズクラスターとしては系の 2 次元性を良く反映していると考えられる。一方で，N_s=15, 18, 24 ではそのような菱形を取ることが出来ない。有限サイズクラスターの端は周期境界条件を課しておく。



第 4 図：(A) N_s=12, (B) N_s=15, (C) N_s=18, (D) N_s=21, (E) N_s=24, (F) N_s=27 の各場合に対する三角格子有限サイズクラスター。

我々が厳密対角化法で計算した，これらの系の物理量は，系の磁化に当たる$\sum_j S_j^z = M$で分類された部分空間の最低エネルギー

$$E(N_s, M)$$

と，その時の固有状態に対するスピン相関関数

$$\langle S_i^z S_j^z \rangle$$

である。直接シミュレーションで得た，上のハミルトニアンに対するこれらの物理量から，上述の模型の基底状態について議論を進めていく。今回我々は，特に $0 \leqq J_2/J_1 \leqq 1$ の領域に焦点を当てる。$J_2/J_1=1$ の等方的な場合に長距離秩序が実現していることが確認できれば，$J_2/J_1=0$ の場合に系が，反強磁性 1 次元鎖の系となり，その 1 次元性から長距離秩序は存在しないこと

が分かっているため，$0 \leqq J_2/J_1 \leqq 1$ の範囲のどこかで長距離秩序が消失する相転移の発現が確実だからである。

２－２．これまでの研究報告

三角格子 S=1 ハイゼンベルク反強磁性体を考える前に，スピンの大きさが異なる場合についての状況を復習しておこう。量子系を理解する前に，取り扱いが相対的に容易な（S が無限大の場合に対応する）古典系について調べておくのは有用である。その古典系では，$J_2/J_1<1$ の領域で，隣り合うスピン同士が成すキャント角で特徴づけられるスパイラル基底状態が実現している。これは，古典系のスピンはその実効的な長さが変動出来ず，エネルギーを下げる要素はスピン間の角度しかない状況で選ばれた結果である。$J_2=J_1$ の等方的な場合に選ばれるのが，キャント角が 120 度となる第 2 図のスピン構造である。ところが量子系では，量子揺らぎの結果，一般にスピンの実効的な長さがスピン演算子の大きさ S から縮む。そのため，量子的基底状態は，個々のスピンの実効的な長さとスピン間の角度が調整されて選ばれた結果として実現することになる。もっとも，Anderson が指摘したような「スピン液体状態」が実現していれば，長距離秩序としてはスピンの実効的な長さがゼロに潰れてしまうような状況となっている可能性もあった訳で，どのような基底状態なのかは，全く自明でない問題である。

S=1/2 の系[14-22]については，古典系と類似のスパイラル状態が J_2/J_1～1 近傍で実現しているという研究報告が複数あるが，スパイラル相の範囲は計算方法によって様々な異なる結果が得られている。級数展開法[14]では，古典系と同じ J_2/J_1～0 までの全領域にスパイラル相が広がっているという結果もあれば，変分モンテカルロ法[17]によるとスパイラル相は J_2/J_1～0.85 までの領域に留まっているという。さらには，汎関数繰り込み群法[19]によれば，等方的な場合に現れる長距離秩序は異方性が入ると急速に失われるという報告もある。磁気秩序を定量的に捉えるにはスピン相関関数の振る舞いから秩序変数を引き出すのが直接的な方法であるが，$J_2/J_1=1$ の直上であっても，現時点で厳密対角化法によって取り扱える範囲の小さいクラスターサイズのデータから外挿手続きによって秩序変数を定量的に精度良く求めることは難しいという報告[6,7]もある。このように，様々な方法で調べられている S=1/2 の系ですら，その性質の理解にコンセンサスが得られている状況とは言えない。

それに対して今回調べた S=1 の系は，研究報告自体その数は少ない。そもそも，S=1 の系は，$J_2/J_1=0$ の 1 次元系の基底状態が，S=1/2 の場合と大きく異なっている。S=1/2 の系の基底状態はスピン励起がギャップレスであるのに対して，S=1 の系は，基底状態のすぐ上にハルデンギャップと呼ばれるエネルギーギャップが開いているからである。この $J_2/J_1=0$ におけるエネルギーギャップの存在が，$J_2/J_1=1$ の等方的な場合に現れると考えられる磁気秩序の相のあり方にどのような影響を与えるのかという疑問は非自明な問題と考えられる。そのような状況で，S=1 の系に対しては，連結クラスター級数展開法[23]と結合クラスター法[24]による研究が行われている。前者の方法では，スパイラル状態の磁気秩序相が $J_2/J_1=0.33$ まで広がっていて，そこで無秩序相への相転移が起こる結果となっている。一方，後者では秩序相がさらに広く，少なくとも $J_2/J_1=0.25$ まで広がっていることを報告している。そういう状況で，特定の近似を超えた理解を深める目的で，近似を伴わない直接シミュレーションの重要性が高まっていると言える。そこで本研究では，ランチョス法に基づく数値的厳密対角化法の計算によってスピン相関関数を得て，そこから磁気的長距離秩序の定量的情報を引き出すことを試みた。その第一段階

として行ったことは，J_2/J_1=1 の等方的な場合に，厳密対角化法のスピン相関関数の結果で 3 副格子構造の長距離秩序の実現を確認したことである。さらにその次のステップとして，その長距離秩序が，相互作用に異方性が入った場合にどのように変化していくことになるのかを，古典系での議論からくる予断を排除し，もっぱら厳密対角化の計算だけに基づいて調べていくことにする。

３．計算手法

既に述べたように，量子スピン系に対する直接数値シミュレーションとしては，量子モンテカルロ法と密度行列繰り込み群法，厳密対角化法が代表的な方法である。このうち，量子モンテカルロ法と密度行列繰り込み群法は，比較的大きな系の計算が可能であるが，量子モンテカルロ法は，系にフラストレーションがある場合には負符号問題と呼ばれる計算上の困難が知られている。密度行列繰り込み群法は，フラストレーションの有無に関係なく適用できるが，1 次元的な系に大変強力な方法であるものの，高次元系に対する有効な処方箋は今のところ見つかっていない。これらの状況から，2 次元フラストレート系である三角格子反強磁性体に対してこれらの手法をそのまま適用するのは難しい。一方で，厳密対角化法は，これらの中で最も古くから使われている方法ながら，フラストレーションの有無や系の形状に依らずに計算が可能であることが大きな利点である。取り扱える範囲の大きさの系であれば，統計誤差のない，非常に高精度な結果を得ることが出来て，三角格子反強磁性体の研究に向いていることが分かる。唯一の短所は，取り扱える系の大きさが非常に小さいものに限られることである。その短所を何らかの方法で少しでも克服し，過去に行われていない大きさの系の計算が出来れば，新たな知見を得ることが期待される。その克服の手段として，近年のスーパーコンピュータの特徴を活かした並列計算を行う方向性が考えられ，我々は，そのような並列計算を行う厳密対角化法のプログラムを独自に開発してきた。

ランチョス法に基づく数値対角化では，ベクトルに対してハミルトニアン行列を繰り返し作用させながら，3 重対角行列を作る計算を行うが，ベクトルに行列を作用させる手続きが高コストの中心部分である。この計算を行うには，行列の次元に対応する長さを持つベクトルを数本，コンピュータのメモリに格納することが必要となる。ハミルトニアンが記述する系の大きさ（スピンの数:N_s）が小さいうちは，行列次元がそれほど大きくないので問題ないが，N_s の増加に対して指数関数的に行列次元が増大するため，あっという間に利用可能なコンピュータの資源量を超えてしまう。このような困難を少しでも解消する方法の一つに，MPI を用いて計算を並列化し，各計算ノードが有するメモリに分散してベクトルを格納（それによって単一ノード使用時では使えない大きさの実メモリを複数ノードで同時に使用）することが考えられる。あるノードでの計算には，他のノードのメモリ上にあるデータを使って計算しなければならない状況が発生するが，通信コストの増大を抑えるバタフライ型のノード間データ転送を行うことで，これを実現することが出来る。

今回の大規模チャレンジジョブでは，N_s=27 の S=1 スピン系が最大規模の計算となった。これまでの S=1 スピン系に対する数値対角化法による研究報告では，1 次元系のハルデンギャップの精密評価に関する研究で N_s=24 の報告事例[25]があり，著者が知る限り，これが最大である。なお，N_s=27 の S=1 スピン系については，ごく最近，大きな M に対する部分空間に対する研究報告[26]があるが，行列次元が M=0 の場合に比べて小さな行列次元にとどまる部分空間に対する

計算に限られていて，空間次元が最大となる M=0 の部分空間までは到達できていない。それに対して我々は，この M=0 の部分空間に対する計算を大規模チャレンジジョブで成功させた。行列次元の具体的な数字としては，S=1 スピン系の N_s=27，M=0 の部分空間の次元が 712,070,156,203 である。この場合に対して，我々は，59049 (=3^{10}) 個のプロセス並列で計算を実行し，その際のランチョスステップ 1 回分の時間は約 167 秒であった。

４．数値計算の結果

４－１．等方的な場合

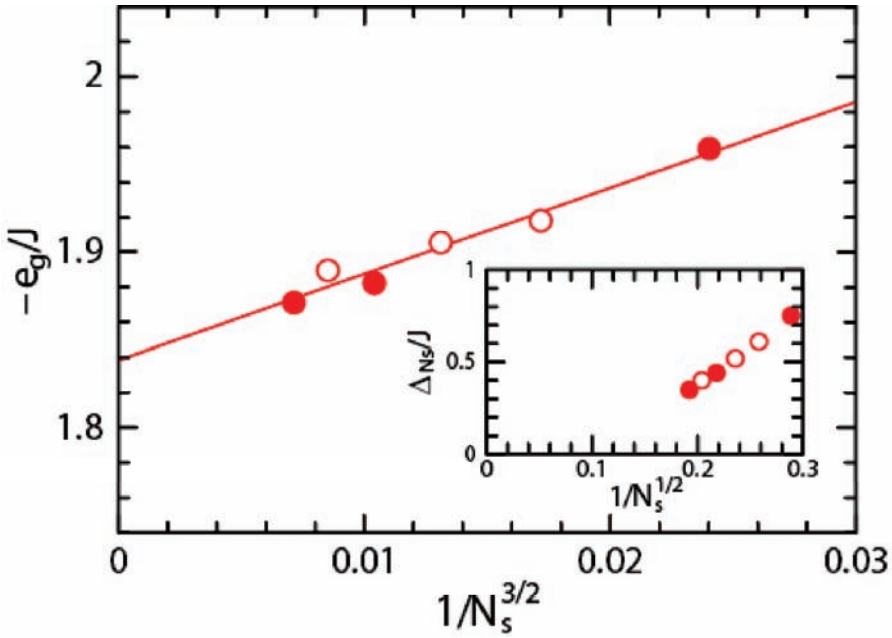
我々がまず調べたのは，サイト当たりの基底状態エネルギー

$$e_g = E(N_s, M=0)/N_s$$

と，基底状態の上のスピン励起エネルギーギャップ

$$\Delta_{Ns} = E(N_s, M=1) - E(N_s, M=0)$$

である。サイト当たりの基底状態エネルギーは，$N_s^{3/2}$ の関数としてプロットし，第 5 図に示した。システムサイズ依存性として良い直線性が見られ，最小二乗法でフィッティングすると



第 5 図：等方的な場合の基底状態エネルギーのシステムサイズ依存性。

挿入図は，基底状態のすぐ上のスピン励起エネルギーギャップのシステムサイズ依存性。塗り潰された丸印は菱形クラスターの場合で，白抜きの丸印は菱形でない平行四辺形のクラスターの場合である。

e_g/J_1=1.838±0.007 という値が得られる。この値は，スピン波近似で得られる値および結合クラスター法の近似で得られるものと良く合致している。また，第 5 図の挿入図では，エネルギーギャップのサイズ依存性を $N_s^{1/2}$ の関数としてプロットした。熱力学極限では，このエネルギーギャップが消失するように見える。ギャップの消失は，基底状態に長距離秩序があることと

矛盾のない振舞である。これらの状況は，第 4 図の有限サイズクラスターが，サイズ外挿によって熱力学極限の値を知るのに適切な系列となっていることを意味している。

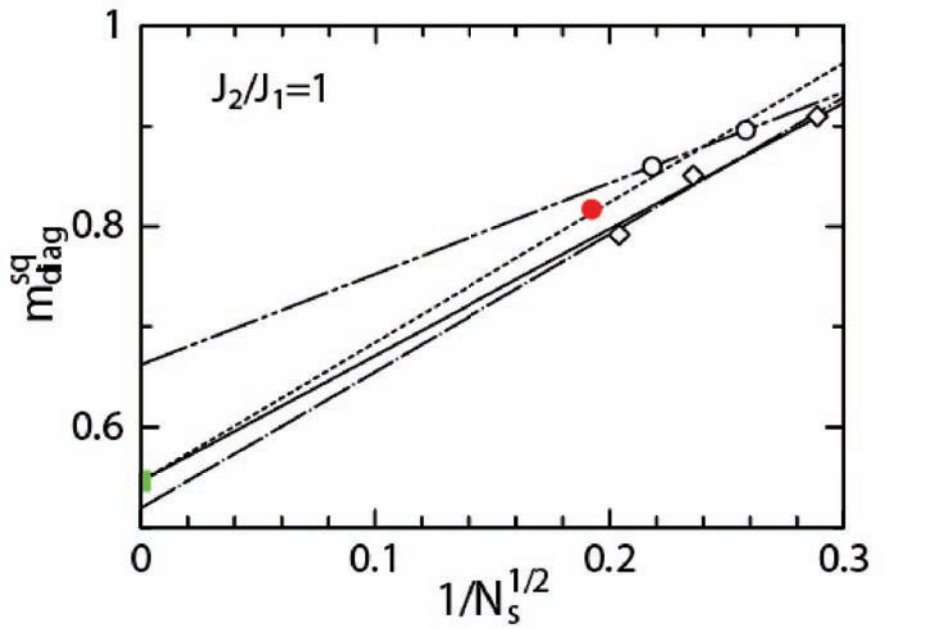
次に我々は，三角格子反強磁性体の有限サイズクラスターの磁気秩序を調べるために，以下の量を各サイズで見積もった。

$$m_{\mathrm{diag}}^{\mathrm{sq}} = \frac{1}{N_\alpha}\sum_{\alpha}\frac{1}{N_{\mathrm{s}}(N_{\mathrm{s}}/3-1)}\sum_{i,j}{}'\langle \boldsymbol{S}_i\cdot\boldsymbol{S}_j\rangle$$

この時，$\langle \boldsymbol{S}_i\cdot\boldsymbol{S}_j\rangle$ はスピン空間について等方的な相互作用を考えていることから，厳密対角化法によって計算されたスピン相関関数を用いて$3\langle S_i^z S_j^z\rangle$ によって得られるものである。また，$m_{\mathrm{diag}}^{\mathrm{sq}}$を見積もる上で，サイト i と jに関するプライム付きの和記号は，i と jが（3 個の副格子のうち）同じ副格子に属し，i と jが同じサイトにならない場合に和を取ることを意味する。さらに，αは，どの方向に相互作用の強い J_1 のボンドを選ぶか，ということに対するラベルであり，$m_{\mathrm{diag}}^{\mathrm{sq}}$の計算では$\alpha$に関する平均を取っている。この平均は，第 4 図の中で菱形でない平行四辺形のクラスターとなる場合を考慮して行うものである。この$m_{\mathrm{diag}}^{\mathrm{sq}}$は，系が磁気秩序している場合に熱力学極限で副格子磁化の 2 乗に対応する物理量である。等方的な場合のスピン波近似では，古典的な長さから量子性に起因して発生するスピンの縮みによって副格子磁化が

$$m_{\mathrm{sw}}=S-\Delta_0+O[1/S]$$

となる。ただし，Δ_0=0.261 である。したがって，m_{sw}^2=0.546 という値が得られるが，それは等方的な場合$m_{\mathrm{diag}}^{\mathrm{sq}}$の比較対象となるものであることが分かる。なお，この$m_{\mathrm{diag}}^{\mathrm{sq}}$は，S=1/2 スピ

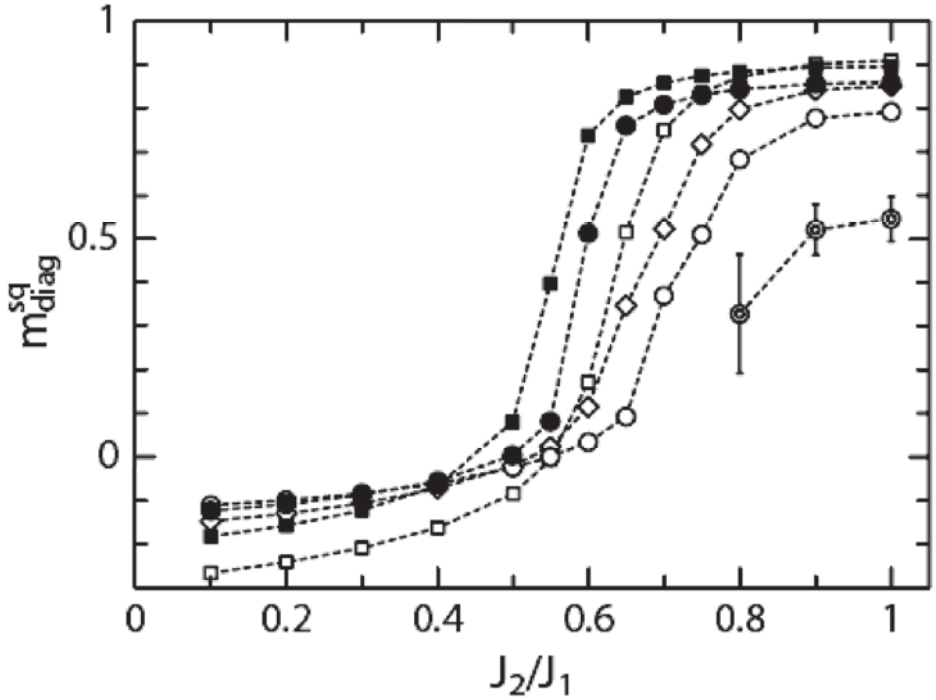


第 6 図：等方的な場合の$m_{\mathrm{diag}}^{\mathrm{sq}}$の外挿。

丸印は N_s が奇数，ダイヤ印は偶数の場合である。赤丸は大規模チャレンジジョブで得た N_s=27 の結果である。緑印はスピン波近似で得られる副格子磁化の値である。

ン系について調べた文献[6]における式(2.11d)と同じものである。また，iとjが同じ副格子に属している場合のみ和に考慮されていることから，この$m_{\mathrm{diag}}^{\mathrm{sq}}$は，異なる副格子に属するスピンの間の関係についての情報を直接与えるものではないことに留意する必要がある。

この$m_{\mathrm{diag}}^{\mathrm{sq}}$に対する数値計算結果は第6図に$N_s^{1/2}$の関数として示した。$N_s$の偶奇で振舞に振動が見られることから，$N_s$が偶数の場合と奇数の場合とで分けて取り扱うべきであることが伺える。N_sの偶奇で異なる系列のデータとなっているとは言え，N_sが無限大の熱力学極限では二つの系列は同じ値に収束しなければならない。このことを踏まえ，第6図での解析の一つとして，N_sの偶奇の系列を別々に直線フィットの最小二乗法を行うが，縦軸切片の値としては同じ値となるような拘束条件を付けてフィットを行うことが考えられる。それをN_s=24までのデータで行ったのが，第6図の実線と点線である。共通の縦軸切片の値として，0.536±0.051が得られる。さらに，拘束条件付きの同じ解析をN_s=27までのデータで行ったところ，縦軸切片は0.551±0.035となった。これらの外挿値は，互いに一致しており，また，スピン波近似の結果とも合致している。したがって，拘束条件付きの解析は，N_s=27のデータが無く，N_s=24までのデータを用いて外挿した場合でも，誤差はやや大き目の結果となるものの，定量的に信頼性の高い値を得ることができることを示している。第6図では，拘束条件付き解析と併せて，N_s=24までのデータの範囲内で，N_sの偶奇の各系列で拘束条件を使わずに直線フィットを行った結果も1点鎖線と2点鎖線で示した。縦軸切片の値として得られるものは，拘束条件付きの同じ解析で得られた外挿値と比べて，N_sが偶数の場合は相対的に小さく，N_sが奇数の場合は相対的に大きくなることが分かった。



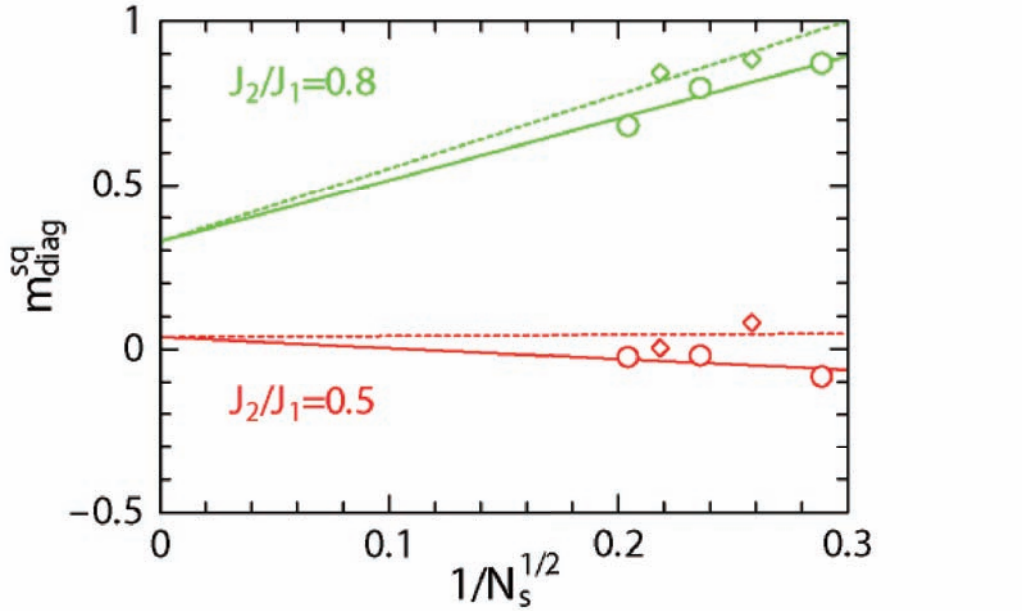
第7図：異方的な場合の$m_{\mathrm{diag}}^{\mathrm{sq}}$の$J_2/J_1$依存性。

□，■，◇，●，○は，N_s=12，15，18，21，24の各場合である。エラーバー付きの二重丸は外挿の結果得られたものである。

４－２．異方的な場合

それでは，空間異方性がある場合（$J_2/J_1<1$）について見てみよう。有限サイズ N_s での$m_{\mathrm{diag}}^{\mathrm{sq}}$を J_2/J_1 の関数として第７図に示した。J_2/J_1 を１から小さくするにつれて，J_2/J_1=0.8 から 0.5 付近の範囲で磁気秩序が急激に減少していく様子が見て取れる。この J_2/J_1=0.8 と 0.5 において，上で述べた拘束条件付き解析を行ったのが，第８図である。J_2/J_1=0.5 では共通の縦軸切片として 0.04±0.17 となり，３副格子構造の長距離秩序が消失していることが示唆される。一方，J_2/J_1=0.8 では，共通の縦軸切片として有意に零とは異なる値が得られた。その結果は，第７図の中で J_2/J_1=0.9 と１の場合と共に示した。J_2/J_1=0.8 から１の領域では，３副格子構造の長距離秩序が確かに現れていることが分かる。

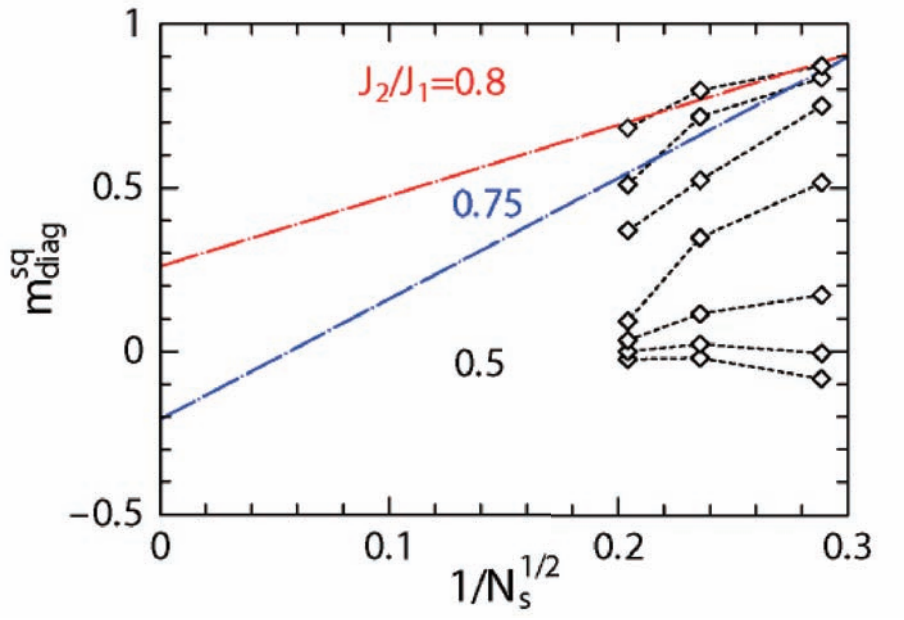
J_2/J_1=0.8 と 0.5 の間では，有限サイズ効果が深刻な形で現れるため，拘束条件付き解析の適用は困難である。その代わりに，我々は拘束条件を課さずに N_s の偶奇の各系列で直線フィットを試みた。第９図は N_s が偶数の系列に対しての結果，第 10 図は N_s が奇数の系列に対しての結果である。第９図で縦軸切片が零となるのは，J_2/J_1=0.75 と 0.8 の間で現れる。等方的な相互作用の場合にみたように，偶数の系列にこの解析を行うと磁気秩序が小さ目に評価されると考えられるので，縦軸切片が有限な値に留まる領域はやや狭めな形で得られることになる。一方，第 10 図で縦軸切片が零となるのは，J_2/J_1=0.6 と 0.65 の間である。N_s が奇数の系列に対しては，偶数の場合と反対で磁気秩序を大き目に評価していると見られ，縦軸切片が有限な値に留まる領域はやや広めの結果となると考えられる。これらの様子から，今回調べた長距離秩序が消失する J_2/J_1 の比は 0.7±0.1 と見積もられる。



第８図：J_2/J_1=0.5 と 0.8 の場合に対する$m_{\mathrm{diag}}^{\mathrm{sq}}$の外挿。

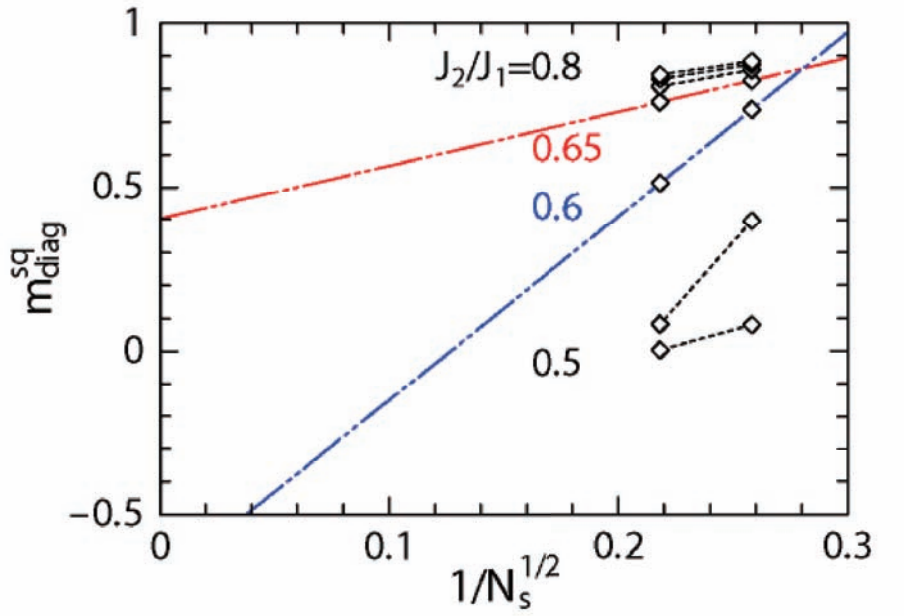
以上のことから，直接シミュレーションで得られた数値データを調べた結果として分かることは，等方的な三角格子の S=1 スピン系に発現している磁気秩序は，相互作用に空間異方性が入った場合でもただちに消失するのではなく，３副格子構造を保ったまま副格子磁化が徐々に

小さくなっていくということである。



第 9 図：J_2/J_1=0.5 と 0.8 の間の偶数サイト系に対する$m_{\mathrm{diag}}^{\mathrm{sq}}$の外挿。

有限サイズのデータは，J_2/J_1について 0.05 刻みで与えている。



第 10 図：J_2/J_1=0.5 と 0.8 の間の奇数サイト系に対する$m_{\mathrm{diag}}^{\mathrm{sq}}$の外挿。

有限サイズのデータは，J_2/J_1について 0.05 刻みで与えている。

5．まとめと展望

我々は，空間異方性を持つ三角格子上の S=1 ハイゼンベルク反強磁性体の基底状態について，

3 副格子構造の磁気秩序の安定性を，近似を伴わない直接シミュレーションである数値対角化の方法（ランチョス法）で調べた。異方性がない等方的な場合には，スピン相関関数の計算データを外挿し，スピン波近似で得られる副格子磁化の結果と定量的に合致することを確認できた。この長距離秩序は，相互作用が空間異方性を持つようになる場合，3 副格子構造を保ったまま徐々に減少し，J_2/J_1 の比が 0.7 付近で消失する結果を得た。磁気秩序相の範囲としては，近似を用いた研究でこれまでに報告されているものと比べるとかなり狭い領域となっている。J_2=0 の場合に現れるハルデンギャップに関連して，J_2/J_1 の比が小さい領域ではスピン励起にギャップが開いていることが予想され，J_2/J_1 の比が大きい場合の秩序相に近づくにつれてどのようにギャップが小さくなっていくのかという問題は，次なる課題である。エネルギーギャップを調べるには，系統的な系列の情報が必要となるが，それには大きな系のデータが必要となるため，スパコンを用いた大規模計算の重要性はますます高まってくる。

三角格子上の S=1 ハイゼンベルク反強磁性体に対する実験としては，これまでに，NiGaS[27] や $Ba_3NiSb_2O_9$[28] が等方的な事例の良い候補物質として報告されている。とは言え，これらの物質に対する実験で観測されていることと理論的な予測が，全ての面で合致している状況ではない。空間異方性を持つ系の候補物質が見つかれば，フラストレーション系の現象を深く理解していく上で非常に有効であることが期待される。

謝辞

本研究は，科学研究費助成事業学術研究助成基金助成金（基盤（C）２４５４０３４８，基盤（C）２３５４０３８８）および文部科学省科学研究費補助金（基盤（B）２３３４０１０９）の支援を受けました。東京大学情報基盤センターのスーパーコンピュータの大規模チャレンジジョブとして実行した大規模数値計算に加えて，数値計算の一部は，東京大学物性研究所および自然科学研究機構核融合科学研究所，東北大学金属材料研究所，九州大学情報基盤センターのスーパーコンピュータの共同利用によって得られたものです。

参　考　文　献

[1] P. W. Anderson: Mater. Res. Bull. **8** (1973) 153.
[2] D. A. Huse and V. Elser: Phys. Rev. Lett. **60** (1988) 2531.
[3] Th. Jolicour and J. C. Le Guillou: Phys. Rev. B **40** (1989) 2727(R).
[4)] R. R. P. Singh and D. A. Huse: Phys. Rev. Lett. **68** (1992) 1766.
[5] B. Bernu, C. Lhuillier, and L. Pierre: Phys. Rev. Lett. **69** (1992) 2590.
[6] P. W. Leung and K. J. Runge: Phys. Rev. B **47** (1993) 5861.
[7] B. Bernu, P. Lecheminant, C. Lhuillier, and L. Pierre: Phys. Rev. B. **50** (1994) 10048.
[8] H. Nakano and T. Sakai: J. Phys. Soc. Jpn. **79** (2010) 053707.
[9] H. Nakano, T. Shimokawa, and T. Sakai: J. Phys. Soc. Jpn. **80** (2011) 033709.
[10] T. Sakai and H. Nakano: Phys. Rev. B **83** (2011) 100405(R).
[11] 中野博生，坂井徹: 日本物理学会誌 **66** (2011) 832.
[12] H. Nakano and T. Sakai: J. Phys. Soc. Jpn. **80** (2011) 053704.
[13] H. Nakano, S. Todo, and T. Sakai: submitted to J. Phys. Soc. Jpn.
[14] W. Zheng, R. H. McKenzie, and R. P. Singh: Phys. Rev. B **59** (1994) 14367.

[15] S. Yunoki and S. Sollera: Phys. Rev. B **74** (2006) 014408.
[16] O. A. Starykh and L. Balents: Phys. Rev. Lett. **98** (2007) 077205.
[17] D. Heidarian, S. Sollera, and F. Becca: Phys. Rev. B **80** (2009) 012404.
[18] J. Reuther and R. Thomale: Phys. Rev. B **83** (2011) 024402.
[19] J. Reuther and R. Thomale: Phys. Rev. B **83** (2011) 024402.
[20] A. Weichselbaum and S. R. White: Phys. Rev. B **84** (2011) 245130.
[21] S. Ghamari, C. Kallin, S. S. Lee, and E. S. Sørensen: Phys. Rev. B **84** (2011) 174415.
[22] K. Harada: Phys. Rev. B **86** (2012) 184421.
[23] T. Pardini and R. R. P. Singh: Phys. Rev. B **77** (2008) 214433.
[24] P. H. Y. Li and R. F. Bishop: Euro. Phys. J. **85** (2012) 25.
[25] H. Nakano and A. Terai: J. Phys. Soc. Jpn. **78** (2009) 014003.
[26] J. Richter, O. G¨otze, R. Zinke, D. J. J. Farnell, H. Tanaka: arXiv:1210.7594.
[27] S. Nakatsuji and Y. Nambu, H. Tonomura, O. Sakai, S. Jonas, C. Broholm, H. Tsunetsugu, Y.Qiu, and Y. Maeno: Science. **309** (2005) 1697.
[28] Y. Shirata, H. Tanaka, T. Ono, A. Matsuo, K. Kindo, and H. Nakano: J. Phys. Soc. Jpn. **80** (2011)093702.

超並列重力多体問題シミュレーションコードの性能測定

石 山 智 明

筑波大学計算科学研究センター

概要

本研究の目的はスーパーコンピューター「京」で最適化された重力多体問題シミュレーションコード、”GreeM” を後継機である FX10 上で性能評価し、「京」上で実施してきた最適化が FX10 上でも有効かどうかを検証することである。また多スレッド多並列環境においての問題点を抽出し、きたるエクサスケールに備えた最適化とアルゴリズム改良のための準備を行うことである。シミュレーションコードはすでに「京」上で最大限チューニングされており、フルシステム(82944 ノード)を用いて 55%程度の対ピーク性能と良好なスケーラビリティが得られている。

ダークマター粒子数 4096^3 の宇宙論的 N 体シミュレーションを 4800 ノードで行った場合、「京」では対ピーク性能 51%の実行効率が得られたが、FX10 でも 45%とそれほど遜色のない性能が得られた。パート毎に見ると、ノードあたりのネットワーク性能が同じであるにも関わらず、全対通信、隣接通信ともに「京」が速いことがわかった。また並列性が自明な部分は、「京」、FX10 双方でピーク性能に比例するような性能が得られた。一方、自明でない、または難しい部分は、予想通り並列度が増えた恩恵をさほど受けられないばかりか、性能が逆転するケースもあった。

1. ダークマター構造形成

最新の宇宙の構造形成理論では、普段われわれが目にする原子や分子のような**バリオン**と呼ばれる物質は、宇宙の物質全体の 15%程度の割合で存在し、残りは**ダークマター**[1]と呼ばれる、重力を通してのみ相互作用する物質であると考えられている。ダークマターは宇宙初期にはほとんど一様に存在したが、ごくわずかな密度の揺らぎが存在した。この揺らぎは重力により成長して、宇宙の至るところにダークマターハローとよばれるさまざまなサイズの高密度なダークマター天体を形成した。はじめに小さいハローができて、それらが繰り返し合体することで、より大きなハローができていった。そしてハローの中でバリオンがダークマター重力によって集まり、現在の宇宙で観測される銀河や銀河団のような多種多様な構造が生まれてきた。この意味で、ダークマターは天体形成に必要不可欠な存在である。またハローの構造形成過程や分布は、銀河や銀河団の形成進化や分布をも決定づける。

こういった宇宙の構造形成過程は、重力の非線形性が本質的に重要な役割を果たしているため、そのダイナミクスを研究するには数値シミュレーションが非常に有用である。

---

[1] さまざまな観測から間接的に存在が確かめられているが、ダークマターの素粒子としての正体そのものはよくわかっていない。ダークマター粒子が「冷たい」、つまり速度分散が小さいモデルは現在の宇宙を非常に良く再現する。冷たいダークマターに基づいた構造形成理論は CDM (cold dark matter) 理論と呼ばれる。

図 1: さまざまなスケールで存在するダークマターハローとそこでできる天体

宇宙ではじめにできるダークマターハロー（最小ハロー）は地球質量（10^{-6} 太陽質量）程度であると考えられている。初代星はおよそ 10^{5-6} 太陽質量のハローの中でバリオンが集まってできたと考えられている（画像は初代星ではなく太陽である。初代星は太陽の数十倍の質量であると考えられている）。銀河は 10^{12} 太陽質量、銀河団は 10^{15} 太陽質量程度のハローの中に存在している。

ダークマターハローは最小のものでは地球質量程度（10^{-6} 太陽質量）[2]、最大のものでは銀河団質量程度（10^{15} 太陽質量）の 20 桁以上にもおよぶ非常に広い質量幅に渡って存在すると考えられており、それぞれの中で初代星や銀河などの対応する天体が形成する（図 1）。宇宙の構造形成過程は、こういったマルチスケールな天体が宇宙約 140 億年という長い時間において相互作用するという極めて複雑な過程である。したがって、考慮すべき質量・空間・時間スケールが広大なため、大規模シミュレーションが必要不可欠である。また宇宙空間でみられる多くの物理現象は地上で実験して検証することが不可能であるため、シミュレーションでのみ扱うことのできる対象も数多い。時にシミュレーションは「理論の望遠鏡」とも呼ばれ、宇宙の研究において非常に重要な位置を占めている。

[2] ダークマター粒子として質量が 100GeV 程度の、「超対称性粒子」、「ニュートラリーノ」を仮定した場合。最小ハローのスケールは、ダークマター粒子による無衝突減衰によって密度揺らぎがならされるスケールに対応している。他のダークマターの候補として「アクシオン」が挙げられる。

2. シミュレーション

本研究で行うシミュレーションの対象は、宇宙の重力的な構造形成に主要な役割を担ってきたダークマターである。シミュレーション手法は**重力多体シミュレーション**、いわゆる N 体シミュレーションである。重力多体シミュレーションでは、N 個の粒子間にはたらく重力相互作用を計算し、粒子の時間発展を追う。ダークマターの他に、星団や銀河、ブラックホールなど適用範囲は非常に幅広い[3]。

ダークマターシミュレーションで、宇宙初期の密度揺らぎを持ったダークマターの分布を離散化して粒子として表現し、重力相互作用による時間発展を追うものを、特に**宇宙論的 N 体シミュレーション**と呼ぶ。無限遠方からの重力を取り入れるために周期境界条件が用いられる。初期条件は宇宙背景放射の観測などから非常に良く制限されている。対象とする天体によって、扱う領域はさまざまである。銀河の空間分布を調べたい場合は、観測できる宇宙の全領域にせまる領域を扱い、銀河の微細構造を調べる時は一つのハローだけを扱う。また初代星形成や初代ハロー形成を追う場合は、宇宙初期のごく限られた狭い領域を取り扱う。図 2 に広い領域を扱った宇宙論的 N 体シミュレーションの一例を示す。ダークマター粒子数 2048^3 を用いた宇宙の大規模構造形成シミュレーションであり、はじめは小さかった密度揺らぎが重力不安定で成長していき、現在の宇宙まで至る様子がわかる。これは観測された宇宙の大規模構造と非常に良く似ており、二点相関関数などの手法を用いて統計的に比較することで、構造形成過程そのものの研究を行うことが可能となる。

宇宙論的 N 体シミュレーションでは、前述のように大規模化が必要不可欠であるため、大規模並列コードの開発が世界のいくつかのグループで進められてきた。我々のグループでも C++ による OpenMP+MPI 並列シミュレーションコード、”GreeM” を開発してきた（Ishiyama et al. 2009）。もともとは国立天文台のスーパーコンピューター Cray-XT4（Opteron 2.2 GHz, 740 ノード、2960 コア）上で開発を進めていたが、さらに理化学研究所計算科学研究機構の「京」上で、フルシステムである 8 万ノードでも効率良く動作するように最適化を行った。その結果良好なスケーラビリティと、「京」のフルシステムを用いた場合、最大 5.8Pflops の実効性能（55% の対ピーク性能比）を達成した。この成果が認められ、「ハイ・パフォーマンス・コンピューティングに関する国際会議 SC12」（2012 年 11 月、米国ソルトレークシティー開催）において、**ゴードン・ベル賞**を単独受賞した。

ゴードン・ベル賞ファイナリストには、ピーク性能が 20 ペタフロップス（「京」の約 2 倍）の「セコイア」（米国ローレンス・リバモア国立研究所）を使い、同様のダークマターシミュレーションで 14 ペタフロップスを達成した米国のグループがあった。ところがわれわれのグループのアプリケーションが実際の計算速度で上回り、1 粒子あたり 2.4 倍の速さでシミュレーションをすることが可能であった（米国と同じ計算機を用いた場合は 5 倍近く速い）。こういった点が評価され、ゴードン・ベル賞の受賞につながった。

---

[3] ダークマター構造形成や銀河は二体緩和時間が対象とする時間スケールより十分長い。このような系は無衝突系と呼ばれ、高精度な重力計算や時間積分よりも、粒子数をできるだけ大きくすることが本質的に重要である。一方、星団やブラックホールは二体緩和時間が対象とする時間スケールより短く、粒子の近接遭遇が系全体の進化に影響を与える。このような系は衝突系と呼ばれ、高精度な重力計算や時間積分が要求される。無衝突系ではツリー法のような近似を用いて重力を計算し、2 次程度の時間積分公式が用いられることが多いが、衝突系では式(1)を直接解き、4 次や 6 次の時間積分公式が用いられることが多い。

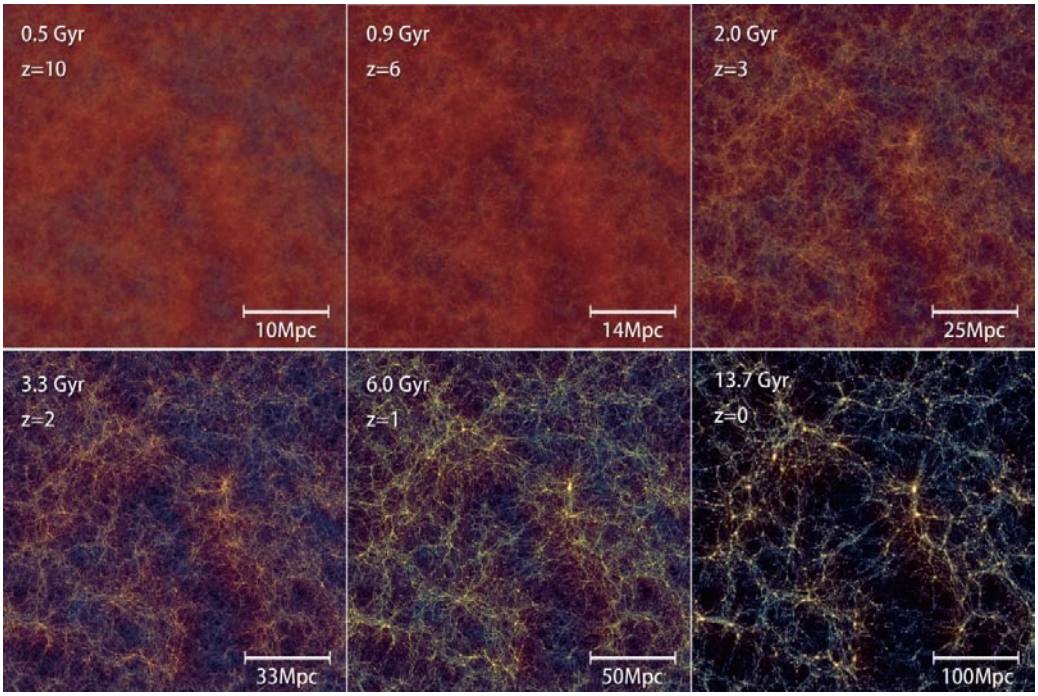


図 2: 宇宙論的 N 体シミュレーションによる、宇宙の大規模構造形成シミュレーション。

各パネルは左上→右上→左下→右下の順に時間進化を表す。左上の数値は宇宙年齢で z=0 は現在の宇宙(宇宙誕生からおよそ 137 億年後)に対応する。右下の数値は物理的な空間スケールを表しており、pc(パーセク)は約 3.26 光年である。

以下 GreeM で採用している重力相互作用計算アルゴリズム、TreePM 法や、並列化手法について述べる。

2.1 TreePM 法

粒子間に働く重力加速度 **f** は通常次のように表される。

$$\mathbf{f}(\mathbf{r}') = \frac{m(\mathbf{r}-\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|^3}. \tag{1}$$

ここで、**r'**、**r**、m はそれぞれ力を受ける粒子の座標、力を及ぼす粒子の座標、質量である。万有引力定数 G は省略してある。全粒子の相互作用をこのまま直接計算すると、相互作用数はペア数に比例するので、粒子数 N に対して演算量は $O(N^2)$ となり非常に大変である。そこで開発されたのが、ツリー法 (Barnes and Hut, 1986) や PM (Particle-Mesh) 法 (Hockney and Eastwood, 1981) などの近似法である。

ツリー法では、近傍の粒子からの力は直接計算するが、十分遠方の粒子群からの力はまとめて多重極展開を使って計算する[4]。このため 8 分木構造が良く使われる(図 3)。ツリー法では計算量を $O(N \log N)$ に落とすことが可能である。

[4] monopole (重心)まで計算することが多いが、コードによっては、quadrupole、octopole、hexadecapole まで計算することもある。

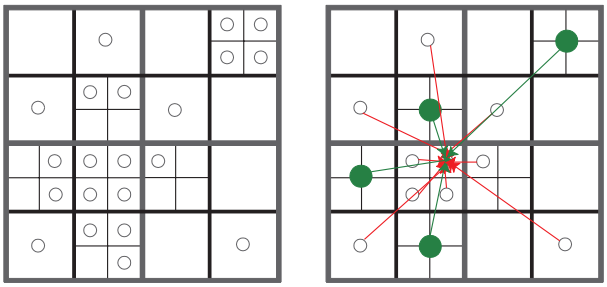


図 3: ツリー法の概念図。

(左)8 分木構造。(右)ある粒子が受ける力。緑線は多重極展開として力を計算していることを表す。

PM 法では、まず粒子の位置と質量から一様格子状の密度場を計算し、FFT とポアソン方程式から一様格子状のポテンシャルを計算する。そして差分と補間から、各粒子位置への加速度を計算する。PM 法では計算量を O(MlogM) (M はメッシュ数)に落とすことができ、宇宙論的 N 体シミュレーションに必要な周期境界条件を自然に組み込むことができるという点で非常に強力であるが、分解能がメッシュ数によって制限される[5]。粒子分布が一様に近い場合はともかく、構造が発達した箇所はより細かいメッシュが必要となり、粒子数 N に対して M>>N が要求されるため、PM 法だけでは計算量の観点からも、メモリ使用量の観点からも不都合である。

そこで考え出されたのが P^3M 法と TreePM 法である[6] (図 4)。まず力を近距離力 $\mathbf{f}_{sr}$ と遠距離力 $\mathbf{f}_{lr}$ に分割する。すなわち

$$\mathbf{f}(\mathbf{r}') = \mathbf{f}_{\rm sr}(\mathbf{r}') + \mathbf{f}_{\rm lr}(\mathbf{r}'), \tag{2}$$

とし、近距離力は式(1)に遠距離で 0 になるようなカットオフがついたカーネル g_{p3m} を掛けたものを用いる。

$$\mathbf{f}_{\rm sr}(\mathbf{r}') = \frac{m(\mathbf{r}-\mathbf{r}')}{|\mathbf{r}-\mathbf{r}'|^3} g_{\rm P3M}(|\mathbf{r}-\mathbf{r}'|/\eta), \tag{3}$$

ここ η はスケール長で、粒子間の距離を規格化する。カーネルはいくつか流儀があるが、ここでは以下の多項式型のものを用いる。

$$g_{\rm P3M}(R) = \begin{cases} 1 - \frac{1}{140}(224R^3 - 224R^5 + 70R^6 \\ +48R^7 - 21R^8) \quad \text{for} \quad 0 \le R < 1 \\ \\ 1 - \frac{1}{140}(12 - 224R^2 + 896R^3 - 840R^4 + 224R^5 \\ +70R^6 - 48R^7 + 7R^8) \quad \text{for} \quad 1 \le R < 2 \\ \\ 0 \quad \text{for} \quad 2 \le R. \end{cases} \tag{4}$$

このカーネルでは 2η がカットオフ長となる。近距離力を直接解いて、遠距離力を PM 法で計算

[5] PM を再帰的に適応して、高分解能を実現するアルゴリズムも考案されており、たとえば AMR 法 (Adaptive Mesh Refinement) などがある。AMR も宇宙論的 N 体シミュレーションに良く用いられる。

[6] PM の代わりに分子動力学で良く用いられる Ewald 法が使われることもある。

するのが P^3M (particle-particle paricle-mesh)法である。構造が発達し、いたるところに高密度な構造ができると、直接計算での計算量が非常に大きくなるので、P^3M 法ではやはり大変である。そこで近距離力をツリー法で解くことでそれを防ぐのが TreePM 法であり、現在宇宙論的 N 体シミュレーションで最も良く用いられているアルゴリズムである。比較的新しいアルゴリズムであり、原案が提唱されたのは1995年である (Xu 1995)。広く使われるようになったのは 21 世紀に入ってからで、いくつかの並列コードが開発されている (GOTPM: Dubinski et al. 2004、GADGET-2: Springel 2005、GreeM: Ishiyama et al. 2009 など)。

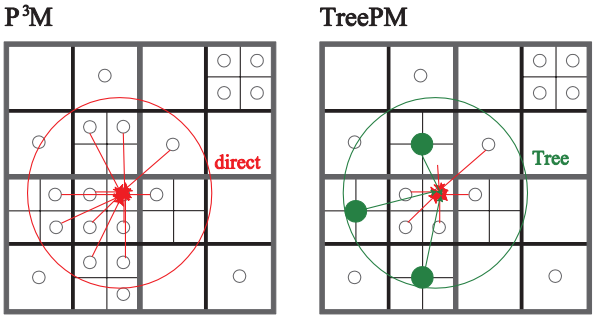


図 4: P^3M 法と TreePM 法の概念図。
(左)P^3M 法。(右)TreePM 法。円は近距離力のカットオフを表し、この外からの力は PM の寄与のみとなる。

最適な誤差と計算量の観点から、カットオフ半径は PM のメッシュ間距離の 3 倍程度にすることが多い。また PM のメッシュ数を全粒子数と等しくするか、全粒子数の 1/8 (メッシュ間距離が平均粒子間距離の 2 倍)とすることが多い。TreePM 法の最大のホットスポットは、ツリー法で重力相互作用を計算する部分である。つまり全粒子の加速度を式(3)、(4)を用いて計算する相互作用カーネル部であり、我々のコード GreeM では、全計算の 6〜7 割を占める。歴史的には、この部分は重力多体問題専用計算機、GRAPE (GRAvity PiPE: Sugimoto et al. 1990 など多数)や、GPU (Hamada et al. 2009 など)を用いて加速されてきた。一方、汎用計算機上でも、SSE や AVX などの SIMD 拡張命令を用いて高速に計算することを可能とするライブラリ、Phantom-GRAPE (Nitadori et al. 2006, Tanikawa et al. 2012, 2013) が開発され、WEB 上で公開されている[7]。「京」と FX10 でのシミュレーションでは、共同研究者の似鳥啓吾氏(2013 年 1 月現在、理化学研究所計算科学研究機構粒子系シミュレータ研究チーム研究員)が開発した HPC-ACE に対応したものを用いている。式(3)、(4)を 1 相互作用あたり 51 演算で計算することができ、FMA 部と非 FMA 部の割合が 1:1 (つまり 17 命令ずつ)であるので、理論ピーク性能は、CPU のピーク性能の 75%となる。カーネルだけの単純なベンチマークテストでは、72%程度の実行効率(理論ピークの 96%)を達成した。

2.2 並列化

並列化は領域分割に基づく。シミュレーション空間をノード数で分割し、各ノードは領域内にある粒子が受ける力を計算する。我々のコードでは、再帰的多段分割法(Makino 2004)に基づいた 3 次元領域分割を行う。この領域分割では使用するノード数が、任意の 3 つの数の積で表されれば何でも良い。またネットワークトポロジーに合致するように空間分割を設定すること

[7] http://code.google.com/p/phantom-grape/
AVX に対応したものや、精度が異なるいくつかのバリエーションが存在する。

で通信を最適化することが可能である[8]。各ステップにおいて、近距離力の計算に必要なカットオフ半径内のツリー情報は隣接ノードからもらい、遠距離力の計算に必要な自領域をカバーするローカルメッシュ上のポテンシャルは、全対通信が含まれる FFT により計算される。

宇宙論的 N 体シミュレーションでは、いたるところに高密度な構造が発達する。したがってできる構造に応じた空間分割や、動的分割が必要不可欠である。プログラム中最大のホットスポットはツリー法による重力相互作用計算部であるが、構造が発達した箇所は計算量も多くなるので、ツリー部分の相互作用の計算量がいっそう大きくなる。したがって粒子数が均等になるような領域分割では、構造が発達するとロードバランスが非常に悪化する。これを解決するために、各ノードが計算する相互作用の総数が均等になるように領域分割をする方法が多用される(Dubinski et al. 2004、 Springel 2005 など)。この方法では、構造が発達した箇所は相互作用数が多くなるので、粒子数を減らして相互作用数を均等にしようとする。したがって粒子数の不均一が大きくなる傾向にある。これはツリー構築や PM 部分で若干のロードバランスの低下を招く。

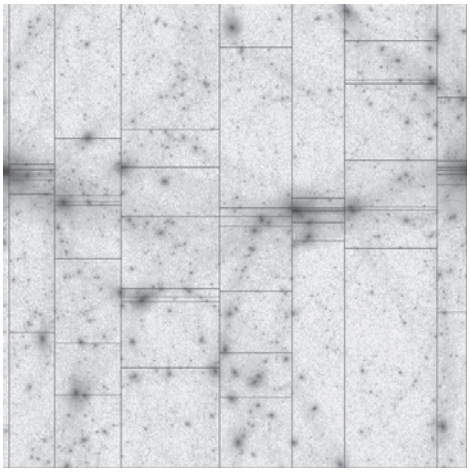


図 5: 領域分割の例。

8x8 分割されている。

そこで GreeM では完全に相互作用の総数が均等になるようにするのではなく、それに若干の補正を加え全体のロードバランスの低下を抑えている。それにはまずツリー法と PM 法の計算全体にわたってプログラム中でかかった時間を計測する。そして領域分割の際にこの時間が均等になるように分割を行っている。単純にホットスポットの情報のみを用いるのではなく、他の部分の情報を用いることでロードバランスの低下を最大限防いでいる。

領域分割の例を図 5 に示す。構造ができる箇所で細分化されているようすがわかる。

なお宇宙論的 N 体シミュレーションのために、世界で最も良く使われている公開コードは GADGET-2(Springel 2005)[9]である。このコードは我々と同じ TreePM 法を用いているが、領域分割に peano-hilbert-ordering を用い、ロードバランスの調整に相互作用数を用いている。このコードと我々のコードの並列化効率の違いを図 6 に示す。なお Phantom-GRAPE を用いて、相

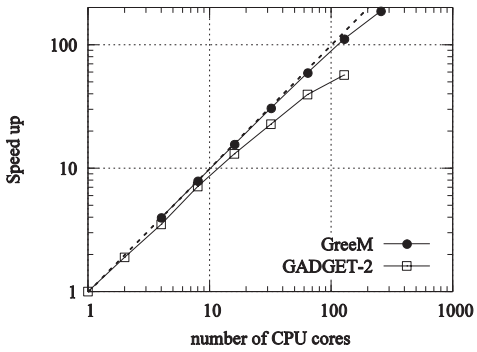


図 6: 我々のコード GreeM と GADGET-2 の並列化効率の比較。

使用した CPU コア数に対する加速率を表す。

[8] 「京」や FX10 では各ノードに 1〜3 次元の仮想トーラス座標が設定される。たとえば、「京」のフルシステムでは、全 82944 ノードを 48x54x32 の 3 次元の仮想トーラスとして割り当てることが可能である。

[9] http://www.mpa-garching.mpg.de/gadget/

互作用計算部をSIMD化していることにより、そもそも我々のコードの方が数倍早い。一般的に遅いコードは並列化効率を出しやすいが、我々のコードはGADGET-2と比較して圧倒的に良い並列化効率を示していることがわかる。

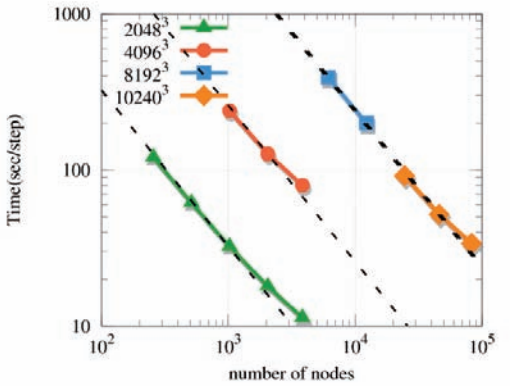


図7:「京」でのストロングスケーリング。使用したノード数に対する計算時間

図7は「京」でのストロングスケーリングを表したものである。粒子数が 2048^3～10240^3 のシミュレーション結果で、非常に良いスケーラビリティを示していることがわかる。我々のコードはスケーラビリティだけでなく実行効率も高く、たとえば2兆粒子のシミュレーションを「京」のフルシステムで実行した時の実行性能は5.8Pflops（55%の対ピーク性能）であった。

3. 「京」とFX10での性能評価

		京	FX10
ノード	CPU	SPARC64™ VIIIfx	SPARC64™ IXfx
	コア数	8	16
	クロック	2.0 GHz	1.848 GHz
	ピーク性能	128 GFlops	236.5 GFlops
	共有L2キャッシュ	6 MB	12 MB
	メモリ容量	16 GB	32 GB
	メモリ帯域	64 GB/s	85 GB/s
	インターコネクト	Tofu	
システム全体	総ノード数	82944	4800
	ピーク性能	10.62 PFlops	1.135 PFlops

表1:「京」とFX10

「京」とFX10でのコードの性能評価に移る。本研究では、「京」で最適化されたコードをそのままFX10で動かしたときに、どれくらい性能が異なるかを評価する。その前に両者の違いを表1にまとめた。FX10のCPUは「京」で用いられているものの後継であり、クロックが若干低く、1CPUあたりのコア数が16と2倍になっているのが大きな違いである。したがって「京」と比べて、スレッド並列に向かずシングルスレッドで処理せざるをえない部分による性能低下がより顕著になる。またスレッド間でのロードバランスを良好に保つプログラミング技術が要求される。さらにノードあたりのネットワーク性能が「京」とFX10では同じであり、スレッド

数の違いによる通信性能への影響も懸念材料の一つである。

そこで本研究では主に主要演算部であるPMとツリー部分の、

(i)　並列度が自明な部分の性能

(ii)　並列度が自明でない部分の性能

(iii)　隣接通信・全対通信の性能

に着目して性能の比較を行う。

3.1. シミュレーション

「京」とFX10ではノードあたりのピーク性能とメモリ帯域が異なる一方、ネットワーク性能が同じと、単純に同じシミュレーションから両者の性能を議論することは難しい。とりあえず同じ規模で同じノード数のシミュレーションで、「京」とFX10の比較を行う。ダークマター粒子数 4096^3 のシミュレーションを、4800 ノード、2048^3PM メッシュで計算したときに各パートにかかった時間を表2にまとめた。10ステップ目から数えて20ステップ分で平均してある。ノード間はMPI並列で、ノード内はOpenMPのハイブリッド並列である。

まず全体の実行性能から述べる。計算した重力相互作用数は1ステップ、1ノードあたりおよそ 3.8×10^{10} であった。これに51(ひとつの重力相互作用計算あたりに必要な演算数)をかけて、1ステップにかかった計算時間で割ると、おおよその実行性能値が計算できる。結果は

- FX10: 507Tflops、45%の対ピーク性能比
- 「京」: 314Tflops、51%の対ピーク性能比

であった。FX10は対ピーク性能比が若干落ちているものの、FX10の性能は「京」と比べ遜色の無いものと言える。

以下各パートについて詳しく述べる。

3.1.1. PM

表2にある演算1はPM部で並列性が自明な箇所を合計したものである。差分、補間などが含まれる。演算2は並列性が自明でない箇所の合計で、送信リスト作成などが含まれる。FFTは

項目(演算1は並列性が自明な部分、演算2はそれ以外)		計算時間 sec/step	
		京	FX10
PM	全対通信	0.15	0.93
	FFT	0.66	0.82
	演算1	0.57	0.30
	演算2	0.092	0.13
ツリー	隣接通信	0.56	1.01
	演算1	24.72	13.76
	演算2	1.23	0.96

表2: 「京」とFX10で、各パートにかかった時間。

両者ともに 4096^3 粒子、4800ノードでのシミュレーションにおいて、1ステップあたりに要した時間である。演算1は並列性が自明な部分、演算2は自明でない部分を合計したものである。仮に演算速度が完全にCPUのクロックに比例するならば、「京」の数値の 128/236.5=0.54倍がFX10での理想値である。

ここではFFTW3ライブラリ[10]を用いた。スレッド並列をしない、1次元の分散並列に対応したものを用いている。いったん各ノードが3次元分割された自領域を覆うローカルメッシュ上の密度場を計算した後に、1 次元分割になるように密度データを全対通信している。このシミュレーションでは、各ノードが1x2048x2048のメッシュデータを持つように通信を行う。厳密に言うと部分対部分通信であるが、極めて非一様な通信であるためMPI_Alltoallvで実装してある[11]。また今回、FFTはノードあたり1コアを使用するため、FFTの計算の際の並列度は「京」とFX10は同じで、2048ノード、2048コアである。

まず全対通信であるが、FX10 は「京」よりも大きく時間を費やしていることがわかる。FFTもFX10が少し遅い。FFTはメモリ帯域リミットであると考えられるため、シングルスレッド処理をしているFFTは、ノードあたりのメモリ帯域が大きいFX10のほうが良い性能が出そうであるが、逆の結果が得られた。全対通信の性能値とあわせて考えると、「京」のほうがネットワークまわりのライブラリのチューニングが進んでいる可能性がある。

演算1に関しては、ほぼ理想的にピーク性能に比例する結果となった。演算2はピーク性能に対して逆の結果となっている。ここは主にループ長が100程度、またはそれ以下と少ないループに対してスレッド並列を行っている。さらにスレッド毎で処理するデータ量が大きく異なるため、ロードバランスの調整が難しく、同期処理に時間がかかったためだと考えられる。

3.1.2. ツリー

次にツリー計算部について述べる。PM部の時と同様に並列性が自明な箇所は演算 1 として、自明でないところは演算2としてまとめてある。前者はある粒子群に対してツリー構造をたどり、相互作用を及ぼす粒子リストを作成するところと、作成した粒子リストから力を計算するところが含まれている。後者はツリー構造の生成である。より具体的には、まず粒子ごとにmorton key を作成し、ソートする。そしてそれに基づき構造体の配列となっているツリー構造を生成していく。各ノードに含まれる粒子全てに対する、グローバルなツリー構造が必要であること、より深い階層のツリー構造を予測することが難しいことから、スレッド並列で性能を得るにはきわめて困難な部分である。我々のコードでは、はじめにあるひとつのスレッドでルートを含む数階層分のツリー構造を生成した後に、それより深い階層のツリー生成をスレッド並列で行っている。

隣接通信が発生するのは、近傍のノードから近距離力の計算に必要なツリー情報をもらうところである。各ノードの領域形状から、そのノードに送るべきツリー構造が生成できる。我々のコードではこの通信と通信リスト作成をオーバーラップさせているが、ここはシングルスレッドで処理をしている。

隣接通信の性能であるが、シングルスレッドで処理、つまりノードあたりのネットワーク性能は同じにも関わらず、FX10の方が2倍近く遅い結果が得られた。本コードのホットスポット

---

[10] http://www.fftw.org/
「京」では富士通製の3次元分散並列FFTを使うことも可能であるが、ここでFFTW3を用いた。

[11] MPI_IsendやMPI_Irecvのような非同期通信が良いと思うかもしれないが、メッシュ形状が板状で、領域分割は3次元形状なので、ある板メッシュを担当するノードは数百～数千のノードから密度データを受け取る必要がある。したがって非同期通信は向かない。また数万ノードではこの部分がボトルネックになり得るため、通信を階層化している。詳しくは Ishiyama et al. 2012を参照。

である演算 1 はほぼピーク性能に比例した理想的な結果が得られた。そのおかげで、「京」と FX10 での実行性能はそれほど遜色の無いものとなっている。演算 2、ツリー構築はやはり原理的な難しさからか、FX10 では 1.3 倍程度の加速であった。

4. まとめ

「京」でチューニングされた超並列重力多体問題シミュレーションコードをそのまま FX10 上で動かした時に、どれくらい性能が異なるかを調べた。ダークマター粒子数 4096^3 の宇宙論的 N 体シミュレーションを 4800 ノードで行った場合、「京」では対ピーク性能 51%の実行効率が得られたが、FX10 でも 45%とそれほど遜色のない性能が得られた。これは「京」上で実施してきた最適化が FX10 上でも有効であることを意味している。つまり FX10 は「京」の後継として優れたシステムであり、プログラムが相互に容易に移植できる利用性の高い環境であるといえる。

その一方、パート毎に見ると、両者で優位な性能差を示す部分が存在することがわかった。まず通信性能であるが、ノードあたりのネットワーク性能が同じであるにも関わらず、全対通信、隣接通信ともに「京」が優位に速いことがわかった。「京」のほうがネットワークまわりのライブラリのチューニングが進んでいる可能性がある。

また並列性が自明な部分は、スレッド並列数が 8(「京」) → 16(FX10)になっても、ほぼ理想的にピーク性能に比例するような性能が得られた。一方、自明でない、または難しい部分は、予想通り並列度が増えた恩恵をさほど受けられないばかりか、性能が逆転するケースもあった。今回は全体の性能にさほど悪影響を与えなかったが、よりノードあたりのコア数やピーク性能が増大した場合は問題になる可能性がある。

今回は時間の都合により表面的な比較しかできなかったが、機会があれば FX10 でノードあたりのスレッド数を変化させた時の性能や、並列性が自明でないところについて、「京」と FX10 でスレッド数を変えた時の振舞いなどを調べていきたいと考えている。

参 考 文 献

(1) J. Barnes and P. Hut, “A hierarchical O(N log N) force-calculation algorithm,” Nature, vol. 324, pp. 446-449, Dec. 1986.

(2) J. Dubinski, J. Kim, C. Park, and R. Humble, “GOTPM: a parallel hybrid particle-mesh treecode,” New Astronomy, vol. 9, pp. 111-126, Feb. 2004.

(3) T. Hamada, T. Narumi, R. Yokota, K. Yasuoka, K. Nitadori, and M. Taiji, “42 Tflops hierarchical n-body simulations on gpus with applications in both astrophysics and turbulence,” in Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, ser. SC ’09. New York, NY, USA: ACM, 2009

(4) R. W. Hockney and J. W. Eastwood, Computer Simulation Using Particles (New York: McGraw-Hill), J. W. Hockney, R. W. & Eastwood, Ed., 1981

(5) T. Ishiyama, K. Nitadori, J. Makino, “4.45 Pflops Astrophysical N-Body Simulation on K computer - The Gravitational Trillion-Body Problem”, Proceedings of the 2012 ACM/IEEE conference on Supercomputing (SC ’12), Saltlake, Utah, USA, Nov 2012 (ACM Gordon Bell Prize Winner)

(6) T. Ishiyama, T. Fukushige, J. Makino, GreeM：Massively Parallel TreePM Code for Large Cosmological N-body Simulations, 2009, PASJ, 61, 1319
(7) J. Makino, “A Fast Parallel Treecode with GRAPE,” Publ. of the Astron. Society of Japan, vol. 56, pp. 521-531, Jun. 2004
(8) K. Nitadori, J. Makino, and P. Hut, “Performance tuning of N-body codes on modern microprocessors: I. Direct integration with a hermite scheme on x86 64 architecture,” New Astronomy, vol. 12, pp. 169-181, Dec. 2006.
(9) A. Tanikawa, K. Yoshikawa, T. Okamoto, and K. Nitadori, “N-body simulation for self-gravitating collisional systems with a new SIMD instruction set extension to the x86 architecture, Advanced Vector eXtensions,” New Astronomy, vol. 17, pp. 82-92, Feb. 2012.
(10) A. Tanikawa, K. Yoshikawa, K. Nitadori, and T. Okamoto, “Phantom-GRAPE: numerical software library to accelerate collisionless N-body simulation with SIMD instruction set on x86 architecture,” New Astronomy, vol. 19, pp.74-88, Feb. 2013
(11) V. Springel, “The cosmological simulation code GADGET-2,” Mon. Not.R. Astron. Soc., vol. 364, pp. 1105-1134, Dec. 2005.
(12) D. Sugimoto, Y. Chikada, J. Makino, T. Ito, T. Ebisuzaki, and M. Umemura, “A special-purpose computer for gravitational many-body problems,” Nature, vol. 345, pp. 33-35, May 1990.
(13) G. Xu, “A New Parallel N-Body Gravity Solver: TPM,” Astrophys. J.Supp., vol. 98, pp. 355-+, May 1995

100 億超格子を用いた自動車の大規模流体解析への挑戦

小 野 謙 二

理化学研究所 計算科学研究機構

大 西 順 也

東京大学 生産技術研究所

１．はじめに

ものづくりの現場において，物理シミュレーションによる設計支援への期待は大きい。自動車設計では，衝突安全性の予測技術として構造解析が用いられるが，熱マネジメントや燃費，走行安定性などの空力性能設計には計算流体力学（Computational Fluid Dynamics, CFD）シミュレーションが用いられる。CFD は流動現象の非線形性をより精確に再現する場合，非常に多くの計算格子点を必要とするため，大規模な計算機資源を必要とするシミュレーションの一つである。

CFD は計算機の発展とともに，学術的・技術的にも大きく進展してきた。CFD のプロセスは，図１に示すように形状データや格子生成に関連する前処理，流体解析，可視化や統計処理などのポスト処理などのサブプロセスから構成される。現実の工業製品の設計に CFD を適用する場合には，現在でも多くの課題がある。まず，最初の課題として現象の数理的なモデリングが挙げられる。流動現象のマクロな運動は Navier-Stokes 方程式の型式で表現されるが，構成方程式は経験的に決められている。特に，混相流の場合には，構成方程式の形やパラメータなど未定なものも多い。比較的多くの研究がなされている乱流現象でさえ，種々のモデルや計算方法が提案されており，未だに決着はつかない。しかしながら，最近の巨大な計算機資源を用いて，できるだけ任意のパラメータを用いず，高い信頼性の結果を得ようとする試みがある。その手法の一つがラージエディシミュレーション（Large Eddy Simulation, LES）である[1]。乱流現象は大きな渦から小さい渦まで，さまざまなスケールの渦から構成されている。LES は，計算格子を細かくしていった極限で，格子で捉えられるスケールの渦は直接計算し，格子スケールより小さな渦は統計的に普遍性のあるモデルを利用する計算方法である。LES は経験的なパラメータが少なく，高い信頼性の結果が得られる計算方法として注目を集めている。

次の課題としては，計算対象の形状の複雑性がある。自動車は非常に多くの部品から構成され，形状も非常に複雑である。これらの形状データは CAD により作成されるが，流体計算を行う場合には，この CAD データから計算格子を作成する必要がある。この前処理プロセスにかかる時間と労力が大きな課題である。

３つめの課題は，シミュレーションにかかる時間である。シミュレーションは，流体計算時間だけでなく，前述の格子生成，および後述のポスト処理の時間も含む。CFD の全プロセスの時間を十分に短くすることが要求される。このため，各サブプロセスに必要な時間を短縮する並列処理のアプローチが必要となる。

大規模な計算機資源を活用したシミュレーションを実行するにあたり，CFD の全てのプロセスについて上記の課題がある。本研究では，各課題についてどのような方針で臨めば良いかを

明確にし，今後の大規模シミュレーションの普及促進を図る礎にすることを目的とする。

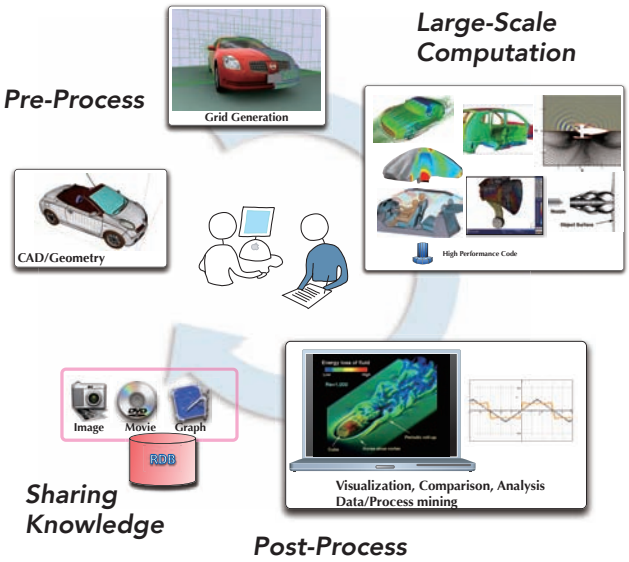


図 1　実務における CFD のプロセスの例

CAD データから計算格子作成，流体計算，可視化や統計処理，その後シミュレーション結果を活用して設計を行う。実務においては，これらの各プロセスをシームレスに実施し，短期間で設計ができるようにすることが要求される。

２．計算格子生成

従来の実用的な計算手法の主流は，非構造格子や BFC 格子を用いた計算手法であり，計算格子の作成に大きな労力を必要とする。場合によっては，計算格子生成に全シミュレーションプロセスの 50%を超える時間を要する場合もある。

大規模な流体計算には，非常に多くの格子点をもつ計算格子が必要となる。例えば，10 億（1G）点の BFC 構造格子の生成を考えてみよう。3 次元の座標点データを単純に単精度で保持すると，x，y，z 各 4B で 1 点あたり 12B で，10 億点では 12GB となる。これに，計算に必要な識別子や付加的な情報もあるので，少なくとも 20GB 程度になる。非構造格子（四面体とする）の場合には，要素あたりの 4 頂点と接続情報，識別子などを含めて少なくとも 70B/要素は必要なので，10 億要素では 70GB のオーダーとなる。エンジニアが利用する PC での格子生成を考えると，現在でも 70GB のメモリ空間はかなり高額となるので現実的とは言えない。今後の計算規模は，100 億以上となる見込みであり，もはや PC での格子生成は難しい領域となる。したがって，前処理として計算格子生成はできなくなるため，計算時に格子生成を行う方法へ移行せざるを得ない。

シミュレーション利用の観点からも，計算時の格子生成には大きな利点がある。まず，大規模計算を実行する計算機は，殆どの場合，バッチ運用の計算機となる。キューの混み具合によって，利用するキュー（大抵，並列数と利用時間の上限があるため，混んでいるキューとそうでないキューなどがある）を選択することができる。状況に応じて，並列数を変更する場合，事前に計算格子を作成する方法では対応できない。また，最初に 512 プロセスで並列計算をして，リスタート時に 1,982 並列で計算をするという利用方法も考えられる。この様な柔軟な利

用を行うためにも計算開始時に格子を生成する技術は必須である。

計算精度の点からは，格子の直交性や解像度，形状の再現性などの課題がある。まず，格子の性質や後に述べる格子生成アルゴリズムと記憶容量の点から，直交格子を採用する。形状の再現性については，直交格子上で任意形状を反映する計算スキーム[2]の導入により対応する。

計算時の初期化段階で格子生成を実施するためには，格子生成アルゴリズムに幾つかの要件が必要になる。一つは，格子生成アルゴリズムのロバスト性である。入力となる形状データから格子を作成する場合にアルゴリズム的に破綻せず，かつ安定であることが必須である。この要請を考慮し，データ構造として単純な直交格子を利用する。直交格子を用いると記憶領域も小さくなり，ソルバーの要求メモリ量も低下するため，メリットは大きい。次の要求項目は，格子生成アルゴリズムの並列性である。分散並列処理で格子生成を行う必要があるためである。

上記の要件を満たす格子生成法として，直交格子データ構造を用いた交点計算に基づく格子生成法を開発した。その概要を図２に示す。計算領域と境界条件については，指定を支援するGUI アプリケーション FXgen を利用する。FXgen[1]は計算対象となる形状データを読み込み，計

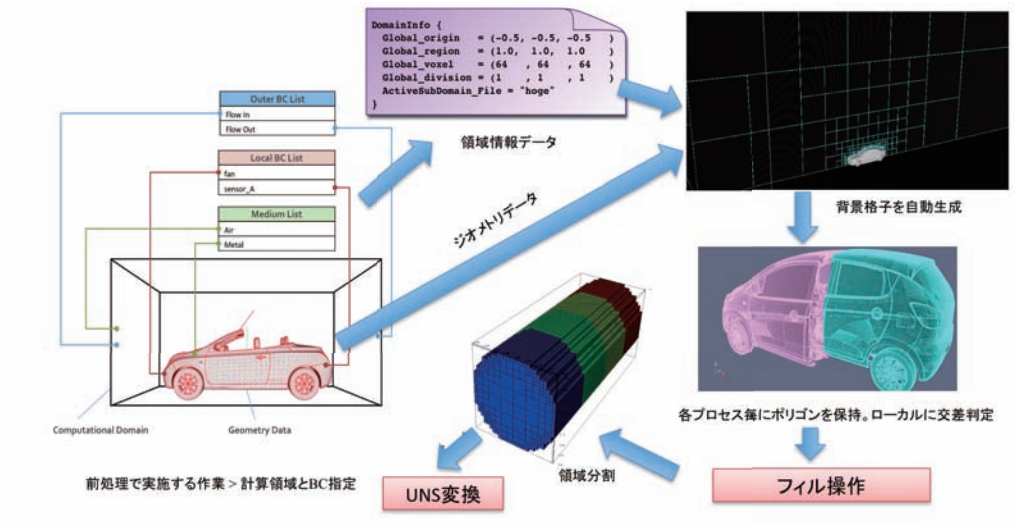


図２　計算時の初期化段階で格子を生成する方法

流体計算の初期化段階で，計算格子を自動生成する。入力データとして，形状を表す STL データと計算領域や境界条件を指定するメタファイルを用意する。メタファイルは，専用の GUI アプリケーションを利用して，事前にテキストデータとして作成する。自動生成された格子データは，オプションにより非構造格子データへも変換でき，非構造格子ソルバーの入力データとしても利用できる。

算領域を空間座標から適切に決めることができる。境界条件については，ロードしたポリゴンデータに識別子を付けて出力する。この識別子は，ソルバーでの境界条件の指定に利用する。流体解析ソルバーは，識別子を付与された形状データと計算領域情報などを記述したメタデータを読み込む。その後，計算領域情報から背景格子を自動生成し，物理量の定義点座標間を結ぶ線分と物体形状との交点計算を行う[2]。この交点計算はロバストであり，部分領域毎に独立に計算可能である。計算された定義点から物体表面までの距離を用いて，計算スキームを構成

[1] 東京大学生産技術研究所からオープンソースソフトウェアとしてリリース予定。

することにより，格子生成の自動化を実現した。

３．流体ソルバー

利用した流体解析ソルバーはFrontFlow/Violet Cartesian(FFV-C)である。このソルバーは，直交等間隔格子を用いた三次元非定常の熱流体解析プログラムである。理化学研究所VCADシステム研究プログラムで開発してきた熱流体解析システム V-Sphere::CBC システム[3]を参考にして，大規模並列計算用に再設計したものである。また，機能モジュール化を進め，下記に示すライブラリ群を利用したプログラミングを行っている。

3.1.　並列計算ミドルウェア CPMlib

FFV-CはCBCシステムと同様に，オブジェクト指向設計で開発している。V-Sphere::CBCの場合には,ミドルウェアとしてフレームワーク的な要素の強いミドルウェアV-Sphereを用いていたが，FFV-C ではより柔軟なプログラミングができるように設計したライブラリ CPMlib（Cartesian Partition Management Library）を開発して利用している。CPMlibは直交等間隔格子を用いた並列計算アプリケーションを記述するために必要な，領域分割処理，並列領域の情報管理，領域間通信の機能を提供する軽量のクラスライブラリである[2]。

3.2.　ポリゴン管理，交点計算ライブラリ Polylib，Cutlib

計算格子の自動生成のためには，ソルバーで形状データのロードと管理を行う必要がある。Polylib[3]は並列領域におけるポリゴンデータの管理機能を提供する。ポリゴンデータのロードと部分領域情報に基づくデータの分散保持，出力，移動，マイグレーションなどの機能をもつ。Cutlib[4]は，ポリゴンと任意線分との交点計算を行う機能を提供する。現在は，直交等間隔格子のみに対応しているが，不等間隔格子への拡張も予定している。

```
Steer {
   . . .
   TimeControl {
       AccelerationType  = "Time"                  //  加速時間の単位指定
       Acceleration       = 1.0
       DtType            = "CFLReferenceVelocity" //  参照速度基準のクーラン数
       DeltaT            = 0.2
       PeriodType        = "step"                 //  計算時間の単位指定
       CalculationPeriod = 1000
     }
   . . .
 } //Steer
```

図３　TextParserによるパラメータの記述

キーワードとキーワードに対する値（数値と文字列）を内部ツリーに保持し，保持したパラメータにアクセスするAPIを提供する。ブレースにより階層的なパラメータ記述ができる。また，自由形式の記述やコメントの挿入なども可能。

3.3. パラメータパーサーライブラリ TextParser

計算パラメータはテキストで行われることが多い。最も簡単な方法としては，平文でフラットなテキスト記述を行う方法があるが，パラメータの意味づけや記述性の点で柔軟ではない。

---

[2] CPMlibは，東京大学生産技術研究所からオープンソースソフトウェアとしてリリース予定。
[3] V-Polylibとして，http://vcad-hpsv.riken.jp/jp/からダウンロード可能。
[4] V-Cutlibとして，http://vcad-hpsv.riken.jp/jp/からダウンロード可能。

一方，汎用的な方法として，XML による記述がある。階層性や任意記述性が高い一方で，煩雑で見通しがわるい点もある。加えて，XML ライブラリはかなり大規模であり，スパコンへの対応などの点でインストールに手間取る面もある。これらの点を考慮し，シミュレーションのパラメータを見通しよく，簡便に記述するため，階層性と自由記述性を主とした軽量パーサーを開発[5]した。図３に TextParser によるパラメータ記述の一例を示す。ブレース”{}”により階層化を図り，パラメータのグルーピングが可能になる。また，C ライクなコメントもいれることもできる。

```
-----------------------------------------------------------------
Report of Timing Statistics PMlib version 1.2

Operator  : Kenji_Ono
Host name : Strontium.local
Date      : 2012/07/11 : 20:31:24

Parallel Mode                  :   OpenMP (24 threads)

Total execution time           = 5.960922e+00 [sec]
Total time of measured sections = 5.461881e+00 [sec]

Statistics per MPI process [Node Average]
Lavel       |call |            accumulated time            |       flop | messages[Bytes]
            |     |  avr[sec]  avr[%]   sdv[s] avr/call[s] |   avr         sdv         speed
------------------+------------+--------------------------+-------------------------------------
Search Vmax :  25   6.0695e-02   1.11  0.000e+00  2.4278e-03   4.325e+08  0.000e+00    6.64 Gflops
assign BC   :  75   2.1376e-03   0.04  0.000e+00  2.8502e-05   0.000e+00  0.000e+00    0.00 Mflops
Copy Array  :  50   3.1872e-01   5.84  0.000e+00  6.3744e-03   0.000e+00  0.000e+00    0.00 Mflops
Pseudo Vec  :  25   3.0864e+00  56.51  0.000e+00  1.2345e-01   2.180e+10  0.000e+00    6.58 Gflops
P Vec FluxBC:  25   1.3652e-02   0.25  0.000e+00  5.4609e-04   1.055e+07  0.000e+00  737.20 Mflops
Pvec. EE    :  25   1.3234e-01   2.42  0.000e+00  5.2938e-03   2.595e+08  0.000e+00    1.83 Gflops
Pvec. BC    :  25   2.2888e-03   0.04  0.000e+00  9.1552e-05   3.870e+04  0.000e+00   16.12 Mflops
PvecFace BC :  25   8.4519e-04   0.02  0.000e+00  3.3807e-05   1.325e+04  0.000e+00   14.95 Mflops
assign Array:  50   3.9125e-02   0.72  0.000e+00  7.8251e-04   0.000e+00  0.000e+00    0.00 Mflops
Div CC      :  25   2.7460e-01   5.03  0.000e+00  1.0984e-02   8.939e+08  0.000e+00    3.03 Gflops
Poi Src. BC :  25   9.2988e-03   0.17  0.000e+00  3.7195e-04   5.499e+06  0.000e+00  563.93 Mflops
Poi Setup   :  25   1.5258e-05   0.00  0.000e+00  6.1035e-07   0.000e+00  0.000e+00    0.00 Mflops
Poi SOR2SMA :  50   1.7074e-01   3.13  0.000e+00  3.4149e-03   7.370e+10  0.000e+00  401.97 Gflops
Poi BC      :  50   9.1314e-05   0.00  0.000e+00  1.8262e-06   0.000e+00  0.000e+00    0.00 Mflops
Prj Vec CF  :  25   5.7449e-01  10.52  0.000e+00  2.2979e-02   1.398e+09  0.000e+00    2.27 Gflops
Prj Vec CFBC:  25   1.0159e-02   0.19  0.000e+00  4.0636e-04   1.930e+04  0.000e+00    1.81 Mflops
Prj Vec CC  :  25   3.0391e-01   5.56  0.000e+00  1.2156e-02   1.038e+09  0.000e+00    3.18 Gflops
Vec. BC     :  25   4.4822e-05   0.00  0.000e+00  1.7929e-06   0.000e+00  0.000e+00    0.00 Mflops
Div CF      :  25   1.0764e-01   1.97  0.000e+00  4.3057e-03   2.019e+08  0.000e+00    1.75 Gflops
P Norm Dmax :  25   4.0273e-02   0.74  0.000e+00  1.6109e-03   8.651e+07  0.000e+00    2.00 Gflops
Updt Vec.   :  25   7.0383e-03   0.13  0.000e+00  2.8153e-04   0.000e+00  0.000e+00    0.00 Mflops
Oflow Vel   :  25   2.9467e-02   0.54  0.000e+00  1.1786e-03   1.455e+07  0.000e+00  471.01 Mflops
Avr Space   :  25   2.7546e-01   5.04  0.000e+00  1.1018e-02   6.632e+08  0.000e+00    2.24 Gflops
H Stdout    :  25   1.0623e-03   0.02  0.000e+00  4.2495e-05   0.000e+00  0.000e+00    0.00 Mflops
H Base      :  25   7.7629e-04   0.01  0.000e+00  3.1051e-05   0.000e+00  0.000e+00    0.00 Mflops
H DomainFlux:  25   3.5381e-04   0.01  0.000e+00  1.4152e-05   0.000e+00  0.000e+00    0.00 Mflops
Monitoring  :  25   1.3113e-05   0.00  0.000e+00  5.2452e-07   0.000e+00  0.000e+00    0.00 Mflops
H Component :  25   2.0313e-04   0.00  0.000e+00  8.1253e-06   0.000e+00  0.000e+00    0.00 Mflops
------------------+------------+--------------------------+-------------------------------------
Total       |              5.461881e+00                          1.005e+11             17.14 Gflops
```

図４　PMlib の出力結果

3.4. 性能情報ライブラリ PMlib

PMlib (Performance Monitor Library) は並列計算時の性能情報をサンプリングする機能を提供する。一般的には，gprof などのチューニングツールを利用することが多いが，チューニ

[5] TextParser は東京大学生産技術研究所からオープンソースソフトウェアとしてリリース予定。

ングツールの利用は，実行性能への影響が少なくないためデバッグ時に限られる。ハードウェアカウンタ情報を用いた機能を提供しているシステムもあるが，ハードウェアカウンタはシステム依存である。そこで，ユーザがある区間の浮動小数点演算数を申告し，その区間の時間を計測することで演算性能のログをとる機能を開発した。図４に示すようなレポートを実行終了時に表示する。また，詳細モードでは，プロセス毎の統計情報も表示できる。このライブラリは，ユーザがループ内の浮動小数点演算数をカウントして，区間ラベルを指定するという手間がかかるが，どの計算機でも利用可能であり，異なるプラットフォーム間の性能評価ができる。また，タイミング測定にかかるコストは無視できるほど小さく，実際の問題を計算している場合でも利用できる利点もある。

3.5. FFV-C ソルバー

FFV-C ソルバー[6]は非圧縮清流体を対象としているので，解析方法は Fractional Step 法に基づいている。有限体積法を用いて離散化しており，対流項スキームには形式的に MUSCL スキームを用いている。反復解法には，RedBalck-SOR，CG，CGS，BiCGStab などを持つ。プログラムはオブジェクト指向技術を利用しており，C++のメイン関数から前述のライブラリ群をインスタンスして利用，コードのカーネル部分は Fortran90 を用いた混合言語で記述している。

４．ポスト処理

ポスト処理については，今回は市販の可視化ソフトウェア FieldView を用いた。利用した可視化環境を図５に示す。今回は，計算結果を Server のローカルストレージ（SSD:900GB）にダウンロードし，Client と Server 間で通信しながら並列可視化を実施した。なお，可視化用のファイルフォーマットとして PLOT3D 形式を用いた。

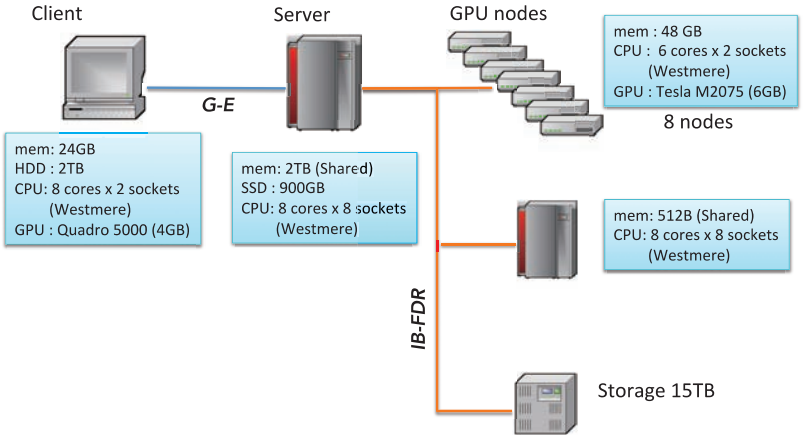


図５　可視化環境

５．計算結果

計算に利用したデータは，自動車会社から借用した実車の形状データである。STL 形式で 1,241 万ポリゴン，3.3GB のファイルサイズである。図６に自動格子生成アルゴリズムにより作成したエンジンルーム内断面の様子を示す。このケースは 4mm 幅の格子を用いたケースである。

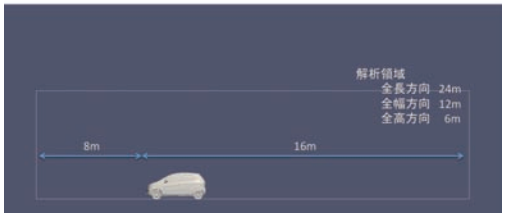
[6] FFV-C は東京大学生産技術研究所からオープンソースソフトウェアとしてリリース予定。

図 7 には車体形状と計算領域の大きさを示す。計算条件は，時速 100km/h 相当である。計算格子は 4mm の格子幅を用いた 290 億格子の計算を想定しているが，クーラン数 0.1 では，意味のある時間平均結果を得るまでに 290 時間を要する試算となる。この計算時間では，4800 ノードを 24 時間しても計算できない。そのため，異なる格子幅の３段階の計算格子を用いて計算する方法を採用した。これは粗い格子を用いて現象の時間を進め，準定常的になって順次細かい格子で計算する方法である。つまり，粗い格子を用いた計算結果からリスタートする場合に，細かい格子に補間する。これにより，計算初期の過渡的な現象を粗く近似し，意味のある時間領域で細かい計算を行い，精度の高い結果を短い計算時間で得ることができる。今回の一連の計算では，利用できる計算機資源の都合上，図 7(b)に示すジョブ１とジョブ２を FX10 Oakleaf 上で計算し，ジョブ３は京の上で計算した。

まず，ジョブ１とジョブ２の Poisson 方程式の反復履歴を図 8 に示す。補間リスタート時には，保存則を満たしていないことによる反復回数の増加が見られるが，時間とともに収束に向かうことが確認される。参考に，最終ステップのジョブ３（京で実行）の履歴を図 9 に示す。リスタート後，しばらく反復回数の多い時間があるが，これは格子が細かくなったことにより細かい渦が解像されるため，その圧力変動の影響と考えられる。この場合もある程度の時間の後，収束に向かう。

図 6　自動生成アルゴリズムにより作成した 4mm 幅格子の計算格子の断面図（車体近傍のみ）
ポリゴンと背景格子との交点計算の結果から固体と流体の識別を行い，そのごフラッドフィルアルゴリズムにより固体内部をフィルする。

	X	Y	Z	備考
解析領域サイズ	24.5 m	12.2 m	6.14 m	ブロッケージ比：約0.4%



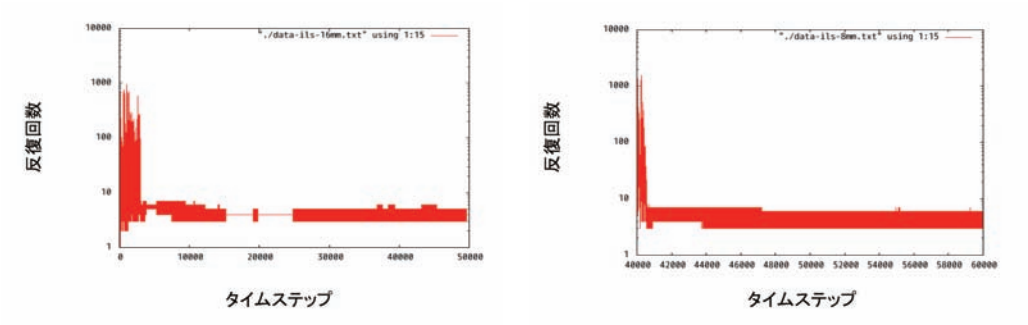
(a) 計算領域と対象

	X	Y	Z	備考
詳細モデル (F, ⊿x=4mm)	6144	3072	1536	約29G格子
中間モデル (C1, ⊿x=8mm)	3072	1536	768	約3.6G格子
疎視化モデル (C2, ⊿x=16mm)	1536	768	384	約0.45G格子

	モデル（格子点数）	ノード数【台】	ステップ数	解析時間【時間】	実時間【秒】	
ジョブ1	C2(0.45G)	4608	50000	2.0	2.87	初期値スタート
ジョブ2	C1(3.6G)	4608	20000	2.0	0.57	補間付きリスタート
ジョブ3	F(29G)	9216	10000	27.4	0.14	補間付きリスタート

(b) 計算格子

図 7　計算領域と計算格子



(a) Δx=16mm 格子（t=0〜2.3 sec.）　　(b) Δx=8mm 格子（t=2.3〜2.87 sec.）

図 8　計算格子幅の違いによる反復履歴

反復解法には，RB-SOR（反復 4 回固定）を前処理に用いた CG 法を用いた。（b）のリスタートでは，リスタート時に保存則を満たしていないので反復回数が多くなるが，時間とともに収束していく。

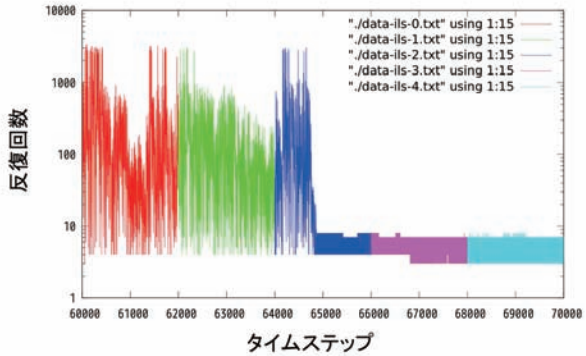


図 9　最も細かい格子での収束履歴（京での参考計算結果）

計算結果の可視化結果を図 10 に示す。解像度により，計算された流れ場の様子が異なることが分かる。両ケースとも，低解像度の計算結果である。最終的な 4mm での計算結果を図 11 に示す。非常に細かい渦が車体周りに生じていることが判る。

非表示

(a) Δx=16mm のケース

ルーフ後端で剥離が生じている。床下の流れの挙動は緩やか。

非表示

(b) Δx=8mm のケース

ルーフ後端での剥離がなくなり，床下にも渦が見える。車体後部で細かな渦が解像されている。

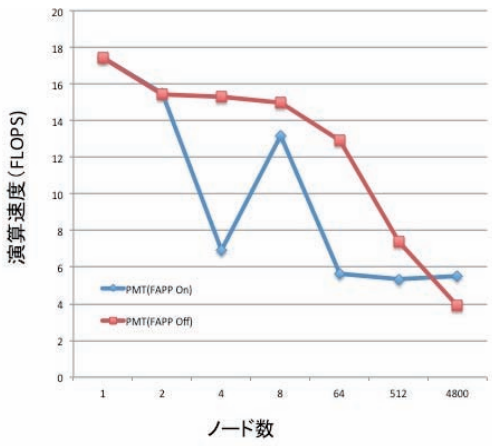
図 10　解像度による結果の違い

非表示

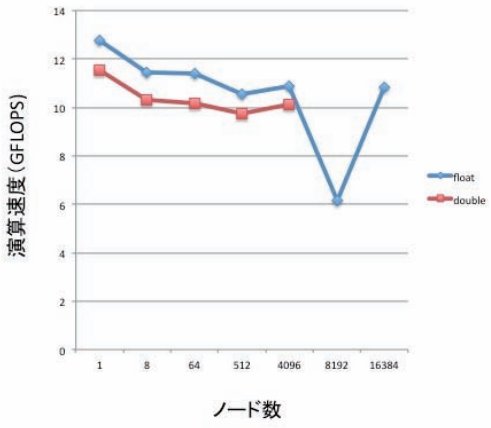
図 11　290 億格子を用いた車体周りの流れの様子

６．ベンチマーク

本節では,大規模計算のベンチマーク結果を述べる。ベンチマーク問題を FX10 と京で実施し，結果を比較する。ベンチマークは，FFV-C の組み込み例題として提供される三次元キャビティフロー問題を固定回数で反復する計算である。ノードあたりの要素数は，256x256x256(16M 点)とする Weak Scaling ベンチマークである。



(a) FX10　ノードあたり要素数は 256x256x256
倍精度，FAPP ON/OFF



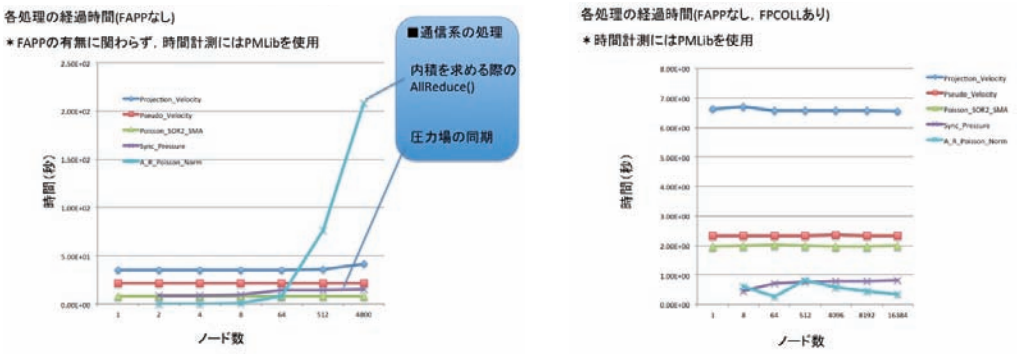
(b) 京　ノードあたり要素数は 128x128x128
FAPP OFF，単精度/倍精度

図 12　PMT ベンチマークのノードあたりの演算速度比較

まず，図 12 にノードあたりの演算性能を示す。問題規模は，FX10 では 16M 点，京では 2M 点とした。演算性能と通信性能のバランスの点では，FX10 の方が有利である。注目すべき点の一つは，FAPP を ON にしてハードウェアカウンタを用いた性能測定を行うと，測定性能が低下するケースがあった。FAPP については無視できないオーバーヘッドが生じていると思われる。そこで，FAPP を使わず PMlib での測定を実施した。FX10 と京を比較すると，コア数とメモリバンド幅の性能は FX10 の方が高いため，実行性能も高くなっていることが確認できた。京では，単精度と倍精度の比較も実施した。単精度の方がバンド幅を有効に利用できるため，若干性能が高くなっている。

次に，通信性能の比較結果を図 13 に示す。FX 10 ではノード数が増加すると，同期通信系の処理時間が増える。特に，集団通信の Allreduce 処理が指数的に増加している。一方，京では通信時間はノード数に依らずほぼ一定である。これはハードウェア機構が働いていると推察できる。

スケーラビリティの点では，FX10 が有利な問題設定になっているが，結果は逆に京の方が良いという結果であった。これは，テスト期間中の FX10 は，通信にコストがかかっているためと思われる。



(a) FX10　　(b) 京

図 13　通信性能の比較

7．まとめ

本研究では，大規模な実用計算を行う上で課題となる点として，格子生成，流体ソルバーの並列化，可視化処理の 3 点について検討した。格子生成については，計算時の初期化段階で自動格子生成を行う方法を提案し，290 億の自動格子生成を 9,216 プロセスで実行し，計算可能な格子を生成できることを示した。計算については，FFV-C ソルバーは十分なスケーラビリティを示すことが判った。また，可視化については，直交等間隔データの場合，290 億格子程度であれば，時間は多少かかるが，市販の可視化ソフトウェアでも可視化が可能であることが判った。

また，ベンチマークプログラムを FX10 と京で実施した。この結果，テスト期間中の FX10 は通信処理が何かの原因で遅くなっているという結果となった。これについては，再度評価したい。また，計時については，開発した PMlib はばらつきの少ないよい測定結果が得られたことを付け加えておく。

参　考　文　献

[1] 梶島岳夫，「乱流の数値シミュレーション」，養賢堂，1999.

[2] 赤坂 啓，小野 謙二，「非水密なポリゴン要素で構成された任意形状周りの直交格子を用いた非圧縮粘性流れ解析」，日本機械学会論文集 B 編，第 76 巻 764 号，pp. 536-545，2010.

[3] http://vcad-hpsv.riken.jp/jp/

ポストペタスケール環境における大規模疎行列解法のための数値計算・通信ライブラリに関する研究

林雅江　中島研吾　石川裕

東京大学情報基盤センター

1 はじめに

有限要素法，有限差分法等の科学技術計算は最終的には大規模な疎行列を係数とする線形方程式を解くことに帰着される．大規模問題向けの解法として，クリロフ部分空間法に基づく前処理付反復法が広く使用されている．並列計算においては，隣接領域境界における一対一通信と Allreduce（内積）による通信が発生し，コア数（MPI プロセス数）の増加とともに，通信によるオーバーヘッドは無視できないものになる．本研究では，並列有限要素法による三次元弾性静解析プログラム(GeoFEM)ならびに並列マルチグリッド法による三次元地下水シミュレーションプログラム(pGW-MG)の二つのアプリケーションを具体例とし，計算科学と計算機科学の両分野の緊密な協力のもとに，数値計算アルゴリズムおよび通信の両方の観点から，ポストペタスケール環境における疎行列解法の確立を目標として実施した．具体的には 1 対 1 通信オーバーヘッドを削減した並列前処理付共役勾配法アルゴリズムを開発するとともに，石川らが開発中の Fujitsu FX10 のインターコネクトが有する RDMA（Remote Direct Memory Access）機能に基づく Persistent Communication をサポートする MPI ライブラリを適用し，更なる最適化を図る．本稿では 2 章で RDMA 機能に基づく MPI ライブラリについて述べ，3 章で実験に用いた二つのアプリケーションプログラムの概要を述べ，4 章で Oakleaf-FX を利用した大規模数値実験について示す．最後に 5 章で本稿をまとめる．

2 RDMA 機能に基づく Persistent Communication をサポートする MPI ライブラリ

2.1 Global Persistent Communication

多くの科学技術計算の並列化では，プロセスローカルな計算とプロセス間通信を繰り返す．繰返し中のプロセス間通信ではその相手先と送受信データ領域は固定していることが多い．

MPI 通信ライブラリにおける Persistent Communication はこのような通信パターンにおいて使われることが想定されて規格化されたものであり，MPI 1.0 通信規格から存在する．図 1 に Persistent Communication の利用例を示す．

```
for (i = 0; i < N; i++)    MPI_Recv_init(rbuf[i], count, MPI_DOUBLE, src[i], tag, com, &req[i]);
for (I = 0; i < N; i++)    MPI_Send_init(sbuf[i], count, MPI_DOUBLE, dest[i], tag, com, &req[i+N]);
do {
        /* Computation */
        MPI_Startall(N*2, req);
        /* Computation */
        MPI_Waitall(N*2, req, stat);
        / **** /
} while (…);
```

図 1　Persistent Communication 使用例

主ループ内で同じ送信パターン，受信パターンを発行する通信処理に対して，主ループに入る前にそれら要求を初期化し(MPI_Send_init, MPI_Recv_init)， MPI_Request 構造体の配列である req に要求を格納している．複数の要求を発行する MPI_Startall 関数を使い，MPI_Waitall 関数で処理の完了を待つ．ループの中で MPI_Isend/MPI_Irecv 関数を使うよりも簡素に記述できプログラムの見通しが良くなるのがわかる．Persistent Communication では，MPI_Send_init および MPI_Recv_init 時に通信バッファを獲得しておくなどの最適化が可能となるが，現在のハードウェアではその効果は限定的である．MPI_Send_init，MPI_Recv_init によって送受信間が関連付けられないため，送受信端それぞれローカルにできる最適化のみとなる．しかし，実際のアプリケーションは通信相手が固定されていることが多く，また，MPI_Startall 時には集団通信と同様コミュニケータに属する全てのプロセスが関与することが多い. そこで以下を仮定する MPI Persistent Communication を Global Persistent Communication と呼ぶことにする （[1]では Persitent Remote DMA と呼んでいる）.

1） MPI_Send_init, MPI_Recv_init によって通信相手が固定される．MPI_Recv_init では，送信元が特定される（MPI_ANY_SOURCE を使用しない）.
2） MPI_Startall は集団通信同様コミュニケータが属する全てのプロセスは MPI_Startall を呼び出す.

Global Persistent Communication では以下のような最適化が可能となる.

1） MPI_Startall 時，ひとたび送受信の通信データ領域が通知されれば，リモート DMA 通信機構を利用することができる. 最初の MPI_Startall 時には通信データ領域の授受が必要になるが 2 回目以降はその情報を再利用できる．ループ内で MPI_Isend/MPI_Irecv を使い大量のデータを通信する場合，MPI 通信ライブラリはランデブプロトコルという通信処理を行う．このプロトコルでは，受信側は受信アドレスを送信側に伝え，送信側はそのアドレスを受け取るとリモート DMA 通信機構を用いて直接受信メモリ領域にデータを転送する. 現在のほとんどの MPI_Startall 実装は, この実装を踏襲しているため, Persistent Communication においてもランデブプロトコルが使われ, 性能に大きな差がない．一方 Global Persistent Communication の実装では，同期は必要であるが，ランデブプロトコルのオーバヘッドを生じさせない.
2） MPI_Startall 時に同時に通信する相手と通信量が確定しているため，通信ハードウェアを効率よく使用することができる. FX10 では計算ノードあたり通信 DMA を 4 基搭載している．通信量の少ない順にこれら 4 基を利用して送信し，4 基の DMA の使用率を上げるよう送信スケジューリングを行うことが可能となる.

2.2 Global Persistent Communication の実装

富士通 MPI が有する MPI 拡張機能の一つである RDMA 機能を用いて Global Persistent Communication を実装した．富士通 MPI で利用できる RDMA 機能を表 1 に示す.

表 1　富士通 MPI の RDMA 拡張機能

FJMPI_Rdma_init	RDMA 機能の初期化
FJMPI_Rdma_finalize	RDMA 機能の終了
FJMPI_Rdma_get_remote_addr	メモリ ID のリモートアドレスの取得
FJMPI_Rdma_reg_mem	RDMA 可能メモリ領域の登録（メモリ ID が返値）
FJMPI_Rdma_put	リモートメモリに DMA による書き出す
FJMPI_Rdma_get	リモートメモリのデータを DMA により取り出す
FJMPI_Rdma_poll_cq	RDMA 完了確認

MPI 通信ライブラリは，ユーザが MPI 関数を上書きできる機能を有している．上書きされた関数の中で MPI 通信ライブラリのオリジナル関数が呼び出せるように, MPI_XXX の代わりに PMPI_XXX という名前の関数が定義されている．本実装では，以下の関数を再定義している．

- MPI_Init/MPI_Finalize
  Global Persistent Communication 実装のために必要な初期化，終了処理の記述

- MPI_Send_init/MPI_Recv_init
  メッセージサイズが小さい場合にはオリジナル関数を呼び出し，そうでなければ，Global Persistent Communication のための初期化を行う．本拡張のために独自の MPI_Request 構造体が定義される．アプリケーションプログラムに渡す MPI_Request 構造体は，オリジナル構造体と本拡張の構造体で区別できるようにする必要がある．オリジナル MPI_Request 構造体はアドレス値として渡される．本拡張の構造体のアドレスを渡すのではなく，アドレス値よりも小さい整数値を渡し区別する．本拡張では，メモリ領域が RDMA できるように登録するとともに, 送信側受信側の同期のためのメモリ領域の確保およびその領域が RDMA できるように登録する．
  - ➢ MPI_Recv_init は MPI_Send 関数を使って送信側に受信メモリ領域およびデータ送信完了を受け取るための同期メモリ領域を送信する．MPI_Send_init 側から送信された同期メモリ領域アドレス通知メッセージが到着していれば受信する．
  - ➢ MPI_Send_init は MPI_Send 関数を使って受信側との同期メモリ領域アドレスを送信する．受信側からのメモリ領域情報が受信できれば受信する．受信できなければ，MPI_Start/MPI_Startall 時に受信する．

- MPI_Startall/MPI_Start
  MPI_Request 構造体がオリジナルであれば，オリジナル関数を呼び出す．そうでなければ，以下の処理をする．
  - ➢ 受信の場合は，送信側からの同期メモリ領域アドレス通知メッセージが受信されていなければ，その到着を待つ．既にアドレスが分かっていれば，RDMA 機能を用いて送信側の同期メモリ領域に Ready to Send フラグを立てる．
  - ➢ 送信の場合，受信側からのメモリ領域情報が到着していなければ，その到着を待つ．

既にアドレスが分かっていれば，送信側の同期メモリ領域に Ready to Send フラグが立った後, RDMA 転送行い, その後受信側同期メモリ領域に送信終了フラグを立てる.

- MPI_Wait/MPI_Waitall/MPI_Test/MPI_Testall
  - MPI_Request 構造体がオリジナルであれば，オリジナル関数を呼び出す．そうでなければ，送信側は送信終了の確認を，受信側は同期メモリに送信終了フラグが立ったかの確認を行う.

Persistent Communication を使用したアプリケーションコードを変更することなく，本ライブラリをリンクするだけで Global Persistent Communication を利用することができる.

3 テストアプリケーションの概要

3.1 並列有限要素法による三次元弾性静解析プログラム「GeoFEM」

ターゲットアプリケーションの一つ目は 3 次元立方体形状における静弾性問題を扱う並列有限要素解析 GeoFEM[2] で，境界条件を図 2 の(1)に示す．また，ヤング率・ポアソン比は一様とする．各節点が 3 つの自由度を持つことから，剛性マトリクスにおいては 3x3 ブロック化がなされている．対称・正定な係数行列であることから解法には 3x3 ブロック対角スケーリング付き共役勾配法（Conjugate Gradient Method，CG）を用い，残差ノルム $|\{b\}-[A]\{x\}|/|b|$ が 10^{-8} 未満となるときを CG 法の収束条件とする．並列プログラミングモデルには FlatMPI と OpenMP/MPI Hybrid 並列の両方を実施した．また，4，800 ノードを用いた大規模並列計算に向け，三次元弾性静解析のメッシュデータをあらかじめ用意するのではなく，計算の始めに各 MPI プロセスが自身の解析領域データおよび隣接領域との情報（通信テーブル）を生成し並列計算を開始するようにした．一辺が 100 要素の立方体（100 万要素，約 300 万自由度）を一ノードに割り当てるとし，図 2 (2) で示すように利用するノード数に合わせて立方体を x, y, z の 3 方向に並べたものが全解析領域となる.

FlatMPI や Hybrid 並列のケースによっては一ノード当たりに割り当てる MPI プロセス数が異なるため，ノード当たりに起動する MPI プロセス数によって 100^3 の立方体がさらに分割され，各 MPI プロセスに割り当てられる．各部分領域は直方体となるため，隣接領域との接し方は面・辺・頂点の 3 パターンとなり，隣接数は最大 27，最少 7 となる．通信量は面で接する隣接ペアの間で最大となり, 本モデルでは最も大きいケースでも 240KB 程度の通信量となる．OpenMP レベルの並列化については，CG 法同様に並列化が容易な対角スケーリングを用いていることからソルバー全体を通し，行に関するループをスレッド数で分割している.

本プログラムにおける計算コストの 90%以上は連立一次方程式の解法部分（対角スケーリング付き CG 法）で占められ，そのほとんどが疎行列ベクトル積部分となっている．そこで疎行列ベクトル積部分の最適化を主眼とし，それに係る計算と通信のオーバーラップを実施した．その疑似プログラムを図 3 に示す．計算と通信のオーバーラップは，疎行列ベクトル積において局所データのみで計算できる行(indep_row)と演算を完結させるには通信が必要な行(dep_row)にあらかじめわけ，indep_row の計算と通信をオーバーラップさせる.

さらに，MPI_Isend・MPI_Irecv を使用した通信部分を 2 章で述べた Persistent Communication 通信である MPI_Send_init，MPI_Recv_init，MPI_Start に置き換えたもの

を図 4 に示す．MPI_Send_init，MPI_Recv_init は，これに 2 章で開発されたライブラリをリンクさせ実行ファイルを作成する．反復のループの中では常に同じ送信パターン，受信パターンで送受信が実行されるため，反復ループに入る前に MPI_Send_init, MPI_Recv_init にてそれら要求を初期化している．ループの内部で MPI_Start 関数を使い Receive 側の reqest, Send 側の request の順に発行している．MPI_Waitall 関数で処理の完了を待つ．

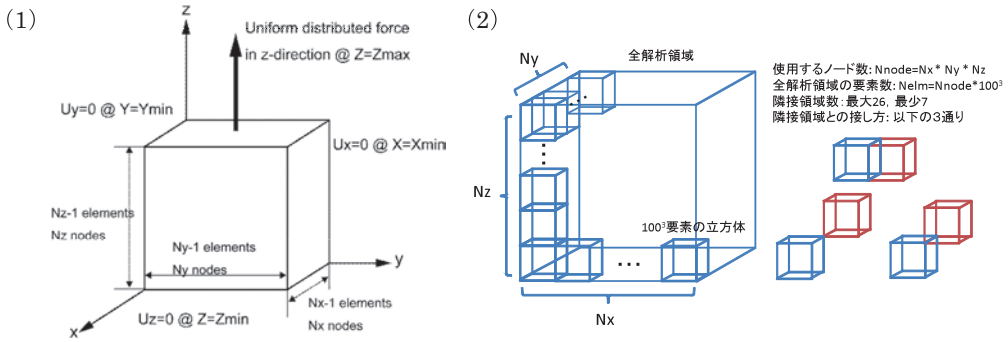
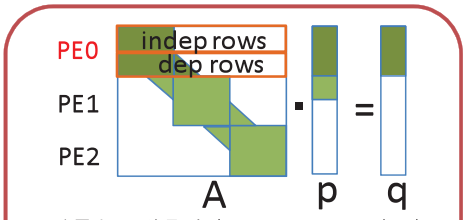


図 2　テストモデルの境界条件（1）と全体解析領域の形状（2）

```
//SEND
for( neib=1;neib<=NEIBPETOT;neib++){
        istart=EXPORT_INDEX[neib-1];
        inum  =EXPORT_INDEX[neib]-istart;
        for( k=istart+1;k<=istart+inum;k++){
            ii=3*EXPORT_ITEM[k-1];
            WS[3*k-3]=WW[P][ii-3];
            WS[3*k-2]=WW[P][ii-2];
            WS[3*k-1]=WW[P][ii-1];
        }
        MPI_Isend(&WS[3*istart], 3*inum, MPI_DOUBLE,
        NEIBPE[neib-1], 0, MPI_COMM_WORLD, &req1[neib-1]);
}
//RECEIVE
for( neib=1;neib<=NEIBPETOT;neib++){
        istart=IMPORT_INDEX[neib-1];
        inum  =IMPORT_INDEX[neib]-istart;
        ii=3*(IMPORT_ITEM[istart]-1);
        MPI_Irecv(&WR[3*istart], 3*inum, MPI_DOUBLE,
          NEIBPE[neib-1], 0, MPI_COMM_WORLD, &req2[neib-1]);
}

//BEGIN calculation for independent rows
#pragma omp parallel for private(ip, iS, iE, j, X1, X2, X3, WVAL1, WVAL2, WVAL3,
k, i, in)
for(ip=1;ip<=PEsmpTOT;ip++){
        iS=STACKindep[ip-1]+1;
        iE=STACKindep[ip ];
    for(in=iS;in<=iE;in++){
        j=row_indep[in-1];
        X1=WW[P][3*j-3];
        X2=WW[P][3*j-2];
        X3=WW[P][3*j-1];

            (indep_rowについてq=A･pの演算)

        WW[Q][3*j-3]=WVAL1;
        WW[Q][3*j-2]=WVAL2;
        WW[Q][3*j-1]=WVAL3;
    }
}
```

```
MPI_Waitall (NEIBPETOT, req2, sta2);
MPI_Waitall (NEIBPETOT, req1, sta1);

//BEGIN calculation for dependent rows
#pragma omp parallel for private(ip, iS, iE, j, X1, X2, X3, WVAL1, WVAL2, WVAL3,
k, i, in)
for(ip=1;ip<=PEsmpTOT;ip++){
        iS=STACKdep[ip-1]+1;
        iE=STACKdep[ip ];
    for(in=iS;in<=iE;in++){
        j=row_dep[in-1];
        X1=WW[P][3*j-3];
        X2=WW[P][3*j-2];
        X3=WW[P][3*j-1];

            (dep_row についてq=A･pの演算)

        WW[Q][3*j-3]=WVAL1;
        WW[Q][3*j-2]=WVAL2;
        WW[Q][3*j-1]=WVAL3;
    }
}
```



上図はPE0に注目した時のdep_row/indep_rowを示す．
割り当て行（濃い緑）の計算のうちdep_rowの計算には
ベクトルpの袖領域成分が必要となる。

図 3　CG 法における疎行列ベクトル積部分にて実施した計算と通信のオーバーラップ

```
//SEND
for( neib=1;neib<=NEIBPETOT;neib++){
        istart=EXPORT_INDEX[neib-1];
        inum  =EXPORT_INDEX[neib]-istart;
        for( k=istart+1;k<=istart+inum;k++){
            ii=3*EXPORT_ITEM[k-1];
            WS[3*k-3]=WW[P][ii-3];
            WS[3*k-2]=WW[P][ii-2];
            WS[3*k-1]=WW[P][ii-1];
        }
        MPI_Send_init(&WS[3*istart], 3*inum, MPI_DOUBLE,
        NEIBPE[neib-1], 0, MPI_COMM_WORLD, &req1[neib-1]);
}
//RECEIVE
for( neib=1;neib<=NEIBPETOT;neib++){
        istart=IMPORT_INDEX[neib-1];
        inum  =IMPORT_INDEX[neib]-istart;
        ii=3*(IMPORT_ITEM[istart]-1);
        MPI_Recv_init(&WR[3*istart], 3*inum, MPI_DOUBLE,
          NEIBPE[neib-1], 0, MPI_COMM_WORLD, &req2[neib-1]);
}


for(ITER=1; ITER<=MAXIT; ITER++){  //ITERATION START

                    (CG法アルゴリズム中略)

/***SpMV***/
//SEND&RECV
for( neib=1;neib<=NEIBPETOT;neib++){
        MPI_Start(&req2[neib-1]);
}
for( neib=1;neib<=NEIBPETOT;neib++){
        istart=EXPORT_INDEX[neib-1];
        inum  =EXPORT_INDEX[neib]-istart;
        for( k=istart+1;k<=istart+inum;k++){
            ii=3*EXPORT_ITEM[k-1];
            WS[3*k-3]=WW[P][ii-3];
            WS[3*k-2]=WW[P][ii-2];
            WS[3*k-1]=WW[P][ii-1];
        }
        MPI_Start(&req1[neib-1]);
}
```

```
//BEGIN calculation for independent rows
#pragma omp parallel for private(ip, iS, iE, j, X1, X2, X3, WVAL1, WVAL2, WVAL3,
k, i, in)
for(ip=1;ip<=PEsmpTOT;ip++){
        iS=STACKindep[ip-1]+1;
        iE=STACKindep[ip ];
    for(in=iS;in<=iE;in++){
        j=row_indep[in-1];
        X1=WW[P][3*j-3];
        X2=WW[P][3*j-2];
        X3=WW[P][3*j-1];
            (途中略)
        WW[Q][3*j-3]=WVAL1;
        WW[Q][3*j-2]=WVAL2;
        WW[Q][3*j-1]=WVAL3;
    }
}
MPI_Waitall (NEIBPETOT, req2, sta2);
MPI_Waitall (NEIBPETOT, req1, sta1);
//BEGIN calculation for dependent rows
#pragma omp parallel for private(ip, iS, iE, j, X1, X2, X3, WVAL1, WVAL2, WVAL3,
k, i, in)
for(ip=1;ip<=PEsmpTOT;ip++){
        iS=STACKdep[ip-1]+1;
        iE=STACKdep[ip ];
    for(in=iS;in<=iE;in++){
        j=row_dep[in-1];
        X1=WW[P][3*j-3];
        X2=WW[P][3*j-2];
        X3=WW[P][3*j-1];
            (途中略)
        WW[Q][3*j-3]=WVAL1;
        WW[Q][3*j-2]=WVAL2;
        WW[Q][3*j-1]=WVAL3;
    }
}

                    (CG法アルゴリズム中略)

/*** check torelance***/
if (RESID< eps) break;

}
```

図 4　CG 法における疎行列ベクトル積部分の通信に Persistent Communication の MPI 関数を適用

3.2 三次元地下水シミュレーションプログラム「pGW-MG」

本研究では，図 5 に示すような不均質な多孔質媒体中の三次元地下水流れを並列有限体積法（Finite Volume Method，FVM）によって解くアプリケーションを扱う．対象とする問題は以下に示すような，ポアソン方程式および境界条件である：

$$\nabla \cdot \left(\lambda(x, y, z)\nabla\phi\right) = q, \phi = 0 \; at \; z = z_{\max}$$

ここで，ϕは水頭ポテンシャル，$\lambda(x,y,z)$は透水係数で位置座標の関数であり，セル（cell）ごとに異なっている．透水係数は，地質統計学の分野で使用される Sequential Gaussian アルゴリズム[3]により発生させた値を使用した（図 5（a））．q は体積フラックスであり，本研究では

一様（=1.0）に設定されている．透水係数の最小値，最大値，平均値はそれぞれ 10^{-5}，10^{+5}，10^{0} となるように設定されている．有限体積セルは一辺長さ 1.0 の立方体である．
このような問題設定では，条件数が 10^{10} のオーダーとなるような対称，正定な悪条件マトリクスを係数とする線形方程式を解く必要がある．本研究で対象とするモデルは，各々 128^3 セルから構成される同じ不均質場に基づく部分モデルの集合である．したがって，x，y，z 各方向に周期的に同じ不均質パターンが繰り返される．

（a）　　（b）

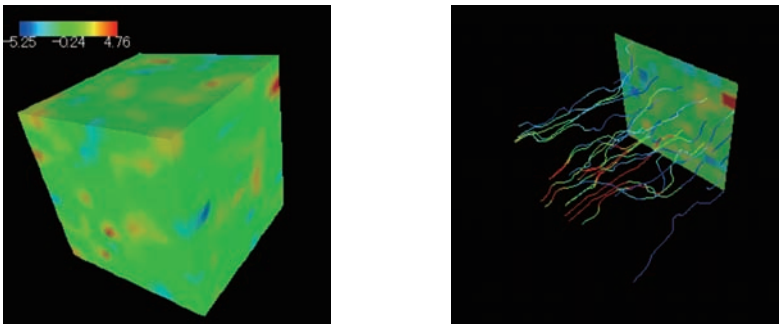


図 5　不均質多孔質媒体中の地下水流れの例，（a）透水係数分布，（b）流線

本研究では，ポアソン方程式を有限体積法によって離散化して得られる対称，正定（Symmetric Positive Definite，SPD）な行列を係数行列とする連立一次方程式を，多重格子法（Mulrigrid）による前処理を施した共役勾配法（Conjugate Gradient Method，CG）によって解く．このような前処理付き共役勾配法を MGCG 法[4]と呼ぶ．残差ノルム $|\{b\}-[A]\{x\}|/|b|$ が 10^{-12} 未満となるまで反復が繰り返される．多重格子法は大規模問題向けのスケーラブルな解法として注目されている．Gauss-Seidel 法などの古典的反復法はセルサイズに相当する波長をもった誤差成分の減衰には適しているが，誤差の成分のうち，長い波長の成分は緩和を繰り返しても中々収束しない．多重格子法は，長い波長の成分が粗い格子上で効率的に減衰するという考え方に基づいている[4]．多重格子法は，細かい格子において対象とする線形方程式の残差を計算し，修正方程式を粗い格子へ補間（制限補間，restriction）して解き，その結果を細かい格子に補間（延長補間，prolongation）して誤差を補正するというプロセスを，再帰的に多段階に適用することによって構築可能である．各レベルの計算が適切に実施されれば，誤差のあらゆる長さの波長をもった成分を一様に減衰させることができるため，計算時間が問題規模に比例するいわゆる「scalable」な手法の実現が可能である．本研究では，図 6 に示すように，8 個の「子（children）」セルから 1 個の「親（parent）」セルが生成されるような等方的な幾何学的多重格子法に基づき，格子間のオペレーションとしては，最密格子と最疎格子の間を直線的に動く V サイクル[4],[5],[6]を採用した．本研究では，各レベルにおける多重格子法のオペレーションは並列に実施されるが，最も粗い格子レベル（図 6 における Level=k）では 1 コアに集めて計算を実施する．

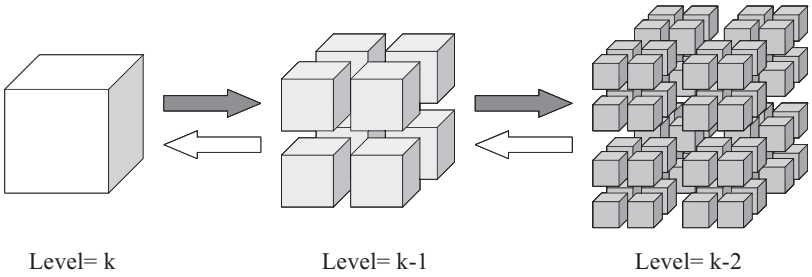


図 6　幾何学的多重格子法のプロセス（8 children=1 parent）

多重格子法では，各レベルにおける線形方程式を緩和的に計算するための演算子を緩和演算子（smoothing operator，smoother）と呼んでいる．緩和演算子として代表的なものはGauss-Seidel 法であり多くの研究で使用されているが，悪条件問題向けには不完全 LU 分解，不完全コレスキー分解が有効である[4]．本研究では，フィルインを生じない不完全コレスキー分解（IC(0)）を緩和演算子として採用した．IC(0)のプロセス（分解，前進後退代入）は大域的な処理を含むため，並列化は本来困難である．各領域において独立に IC(0)処理を実施するような，ブロック Jacobi 型の局所処理によって並列化は可能であるが，特に悪条件問題の場合，領域数が増えると収束が悪化する．ここで，加法シュワルツ法（Additive Schwartz Domain Decomposition，以下 ASDD）[4]を組み合わせることにより，並列計算においても安定した解を得ることが可能となる（図 7）．

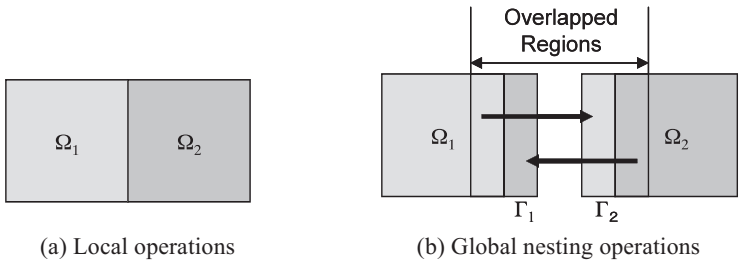


図 7　加法シュワルツ法（Additive Schwartz Domain Decomposition，ASDD）

OpenMP/MPI ハイブリッド並列プログラミングモデルで，FVM によるアプリケーションを並列化する場合，領域分割された各領域に MPI のプロセスが割り当てられ，各領域内で OpenMP による並列化が行われる．各領域においては，不完全コレスキー分解のように大域的な依存性を含むプロセスについては，各要素の並べ替え（reordering）により依存性を排除し，並列性を抽出する手法が広く使用されている[4]．本研究では，並列性が高く悪条件問題に対して安定な CM-RCM 法による並び替えを適用している[4]．本手法は，Reverse Cuthill-McKee（RCM）法とサイクリックに再番号付けする Cyclic マルチカラー法（cyclic multicoloring，CM）を組み合わせたものである．CM-RCM 法では各「色」内の要素は独立で，並列に計算を実行することが可能である．CM-RCM 法の色数の最大値は RCM におけるレベル数の最大値である．本研究では多重格子法の各レベルにおいて CM-RCM 法を適用している．

4 数値実験

4.1 GeoFEM

実験では 1200 ノード，2400 ノード，4800 ノードの三つの規模で性能評価を行った．一ノードあたりに割り当てる要素数は 100^3 要素に固定され，使用するノード数に対して規模を増加させていく Weak Scaling による性能評価を行った．Oakleaf-FX の最大 4800 ノードでの実行時には約 48 億要素（約 144 億自由度）の問題規模となった．OpenMP/MPI ハイブリッド並列プログラミングとしては，

- **Hybrid 2×8（HB 2×8）**:Oakleaf-FX 各ノードにスレッド数 2 の MPI プロセスを 8 つ起動する
- **Hybrid 4×4（HB 4×4）**:各ノードにスレッド数 4 の MPI プロセスを 4 つ起動する
- **Hybrid 8×2（HB 8×2）**：各ノードにスレッド数 8 の MPI プロセスを 2 つ起動する
- **Hybrid 16×1（HB 16×1）**：各ノードにスレッド数 16 の MPI プロセスを 1 つ起動する

のパターンを実施し，Flat MPI と比較した．

（1）最適化前

図 8 は，最適化前のプログラムを用いて 1200 ノードから 4800 ノードまでの Weakscaling による実験結果である．前処理付き反復解法における 1 コア当たりの Flops 値を示す．ノードあたりで適用するプログラミングモデルとしては Hybrid16x1（以下 Hybrid を HB と省略），HB8x2，HB4x4，HB2x8，FlatMPI の全ケースを実施した．4800 ノードでは最も性能がよいケース HB2x8 でコア当たり 7.75GFlops に達し，全体では 61TFlops が得られた．

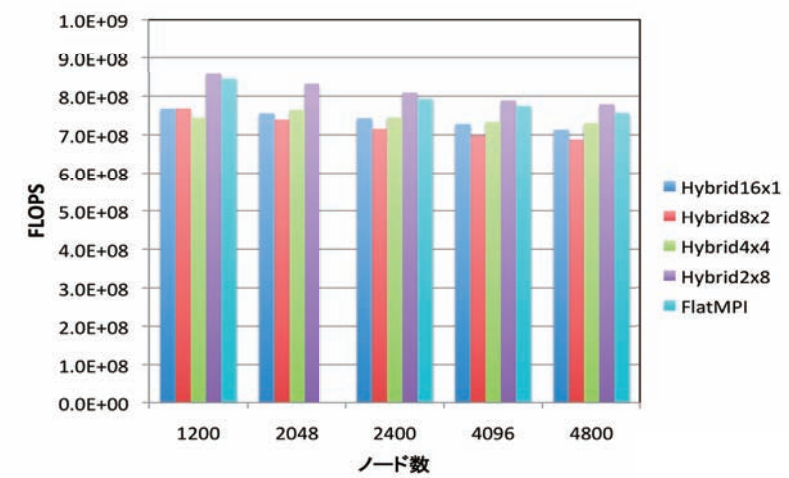


図 8　並列プログラミングモデルによるテストアプリケーションの性能評価

（2）通信のオーバーラップと RDMA 機能に基づく MPI ライブラリの適用

次に，計算と通信のオーバーラップさせた最適化プログラムを用い，4800 ノードで実施した際に RDMA 機能に基づく MPI ライブラリの適用効果を見たものを図 9 に示す．FlatMPI および全ハイブリッドプログラミングモデルのケースで実施し，1 コア当たりの FLOPS 値を測定した．微小ではあるが RDMA 機能をもつ通信ライブラリの効果が得られた．その効果が大きく現れなかった一因としては， 3.1 節で述べたように，本テストモデルにおいては通信ペア

間での通信量が比較的小さいことが挙げられる．RDMA 対応 MPI ライブラリがその効果を十分に発揮すると考えられる通信量に対して小規模であったためと考えられる．

また，通信量が小さいことで，今回実施した計算と通信のオーバーラップも大きな効果には結びつかず，結果的には RDMA 通信ライブラリを有効にした HB2x8 のケースが最高で 6.57GFlops となった．この性能後退の原因には，通信と計算のオーバーラップの実装において，indep_row と dep_row についてのリオーダリングを実施しなかったために間接参照を増やす実装となっていたこともある．そのため，Oakleaf-FX 上において RDMA 通信ライブラリの効果は確認できたものの，最適化前より Flops 値を下げることになった．リオーダリングにより間接参照を行わないような改善の後，再度検証実験が不可欠とされる．

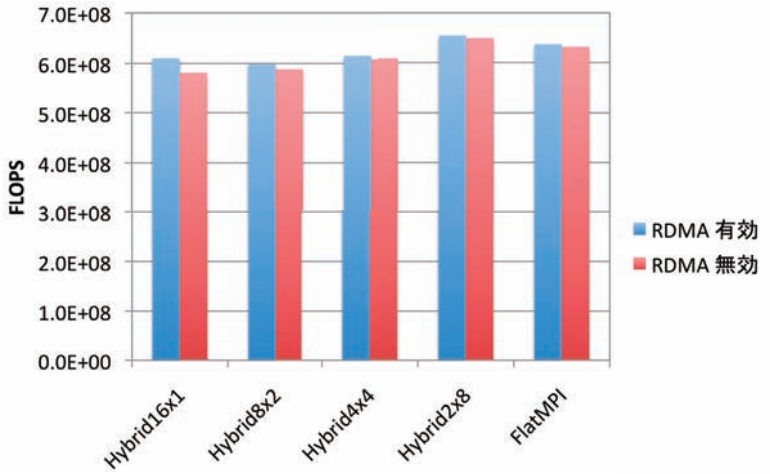


図 9　RDMA 通信ライブラリの効果

4.2　pGW-MG

（1）Coarse Grid Aggregation

既存研究[5]では V サイクルによる多重格子法は図 10 に示すような手順で実施されている．図 11 は図 10 の 1)，2)，3) のプロセスをまとめたものである．

1) Starting from finest level (level=1), a smoothing operation by IC(0), and *restriction* to coarser levels are applied. Operations at each MPI process are done by a parallel manner with some communication.
2) At the coarsest level, information on each process is gathered into a single MPI process, and a “coarse grid problem” is formed. The size of this coarse grid problem corresponds to the number of MPI processes.
3) **(Coarse Grid Solver):** A serial multigrid solver by IC(0) is applied for solving the coarse grid problem on a single core.
4) After the coarse grid problem is solved, the results of the coarse grid solver are scattered to each MPI process.
5) Starting from the coarsest level, *prolongation* to a finer level with smoothing at each level, is applied, until the finest level (level=1).

図 10　V サイクルによる多重格子法

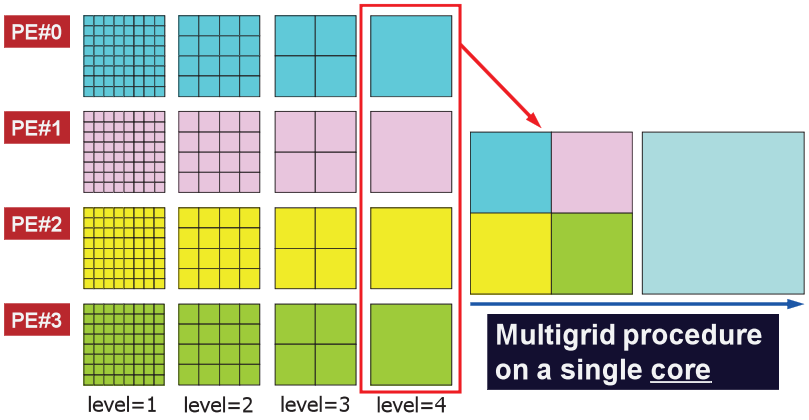


図 11　V サイクルによる並列多重格子法（4 MPI プロセス）における Coarsening／Restriction

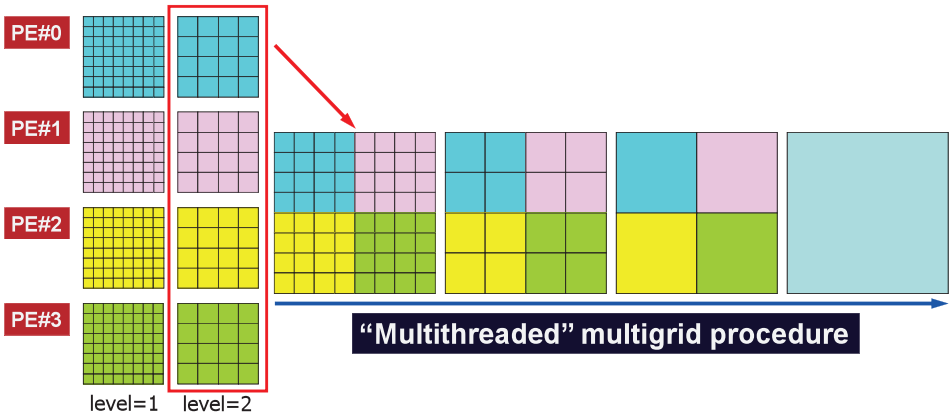


図 12 Coarse Grid Aggregation の例：level=2 の計算の後，各プロセスの結果は 1 MPI プロセスに集められる

中島によって提案された「Coarse Grid Aggregation」[6]では図 10 の 2)，4）における MPI プロセスの統合（aggregation）／分割（disaggregation）がより細かい（finer）レベルで実施されている.「Coarse Grid Solver」[4]への移行をより細かいレベルで実施することによって，収束の安定性，通信オーバーヘッドの削減が期待されるが，Coarse Grid Solver の解く問題サイズはオリジナルの設定と比較してより大きくなる．既存研究[5]では Coarse Grid Solver は 1 コアのみを使用しているが，[6]では各 MPI プロセス上の全コアを使用して OpenMP による並列化を実施している．ポストペタ／エクサスケールシステムでは，各ノードは $O(10^2)$規模のコアを含んでいるものと予測され，各ノードの多数のコアの計算能力を利用することを考慮する必要がある．図 12 は Coarse Grid Aggregation の例であり，ここでは level=2 の計算後に各 MPI プロセスにおける情報は 1 つの MPI プロセスに集められるため，Coarse Grid Solver は図 11 の場合よりも早い段階で開始している．

（2）Weak Scaling

各コアにおける問題サイズ（セル数）は 262,144（=643）を固定した Weak Scaling による性能評価を Oakleaf-FX の最大 4,096 ノード（65,536 コア）を使用して実施した．最大問題サイズは 17,179,869,184 である．以下に示す，3 種類の OpenMP/MPI ハイブリッド並列プログラミングモデルを適用し，全コアに独立に MPI プロセスを発生させる Flat MPI と比較した：

- **Hybrid 4×4（HB 4×4）**:Oakleaf-FX 各ノードにスレッド数 4 の MPI プロセスを 4 つ起動する
- **Hybrid 8×2（HB 8×2）**:各ノードにスレッド数 8 の MPI プロセスを 2 つ起動する
- **Hybrid 16×1（HB 16×1）**:各ノードにスレッド数 16 の MPI プロセスを 1 つ起動する

図 13（a），（b）は HB 8×2（CM-RCM（90））及び HB 16×1（CM-RCM（100））の 4,096 ノード使用時の Coarse Grid Aggregation の効果である．Coarse Grid Solver へ移行するレベルの効果が示してある．表 2 は各ケースにおける収束までの反復回数である．

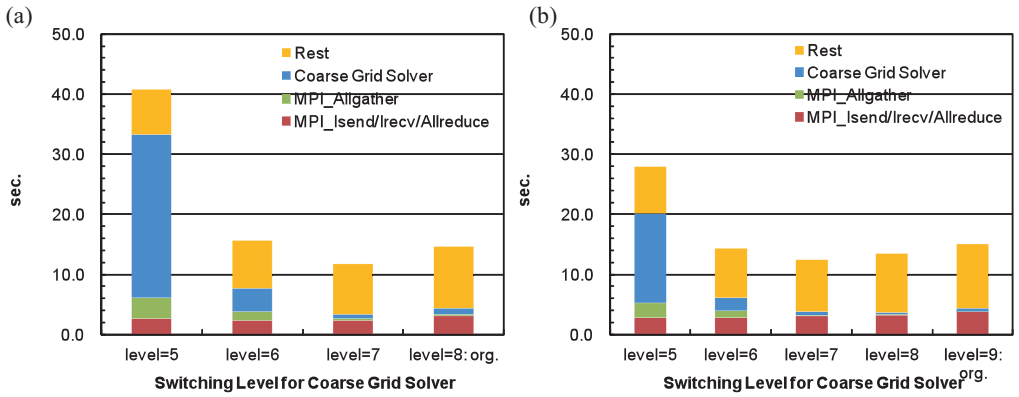
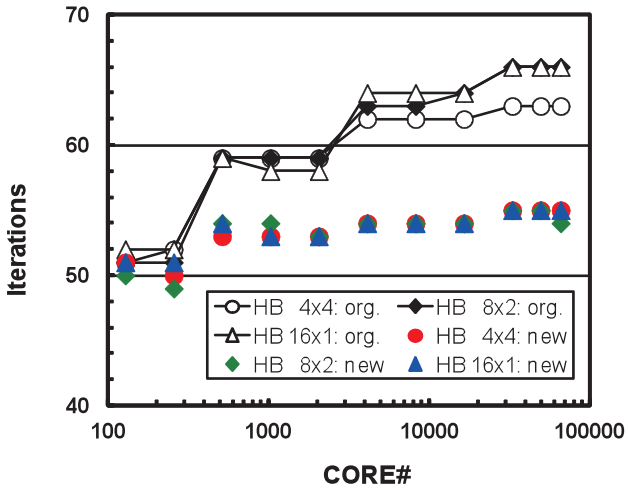


図 13　Oakleaf-FX における CM-RCM 法による MGCG 法の計算時間，4,096 ノード（65,536 コア），最大問題サイズ：17,179,869,184，Coarse Grid Solver への移行レベルの効果（a）HB 8×2（CM-RCM（90）），（b）HB 16×1（CM-RCM（100））

表 2　Oakleaf-FX における CM-RCM 法による MGCG 法の反復回数，4,096 ノード（65,536 コア），最大問題サイズ：17,179,869,184，Coarse Grid Solver への移行レベルの効果

Switching level for coarse grid solver	HB 8×2	HB 16×1
level=9	-	66
level=8	66	61
level=7	54	55
level=6	51	51
level=5	49	50

(a)



(b)

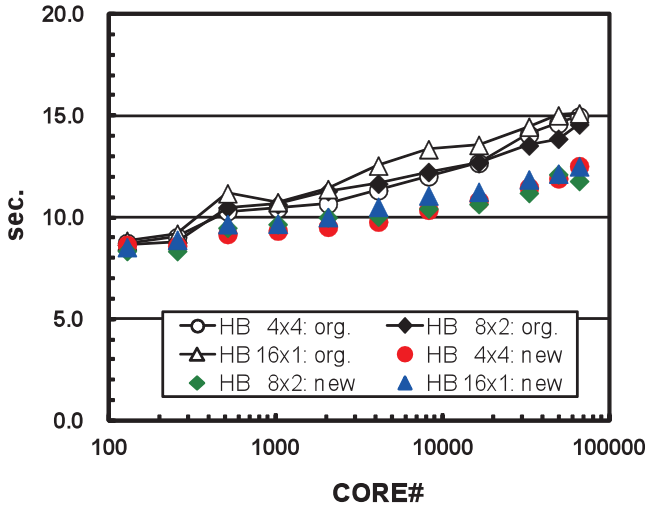


図 14　Oakleaf-FX における CM-RCM 法による MGCG 法の計算性能，Coarse Grid Aggregation の効果，最大 4,096 ノード（65,536 コア），Weak Scaling：コア当たり問題サイズ=262,144（=64^3），最大問題サイズ：17,179,869,184（a）MGCG 法の反復回数，（b）MGCG 法の計算時間 HB 4×4（CM-RCM（70）），HB 8×2（CM-RCM（90）），HB 16×1（CM-RCM（100））

HB 8×2 の level=8，HB 16×1 の level=9 では Coarse Grid Solver は，[5]では 1 コア上で動いている．[5]では ASDD による「局所型」IC(0)が各 MPI プロセスにおいて適用されている．本研究では Coarse Grid Aggregation により Coarse Grid Solver が 1 MPI プロセス上で適用されているため，IC(0)による緩和が局所型 IC(0)と比較してより効果的に行われている．

図 14 に示すように，より細かいレベル（level が小さい場合）において Coarse Grid Solver

へ移行することにより収束は安定し，反復回数は減少しているが，level=5, level=6 では Coarse Grid Solver の問題サイズが大きくなり，むしろ計算時間がかかっている．HB 8×2，HB 16×1 ともに level=7 の場合が最も計算時間が短い．また，HB 4×4 においても level=7 が最適である．

図 14（a），（b）は Weak Scaling における Coarse Grid Aggregation の効果である．全ケースで level=7 としている．図 14（a）に示すように，Coarse Grid Aggregation による収束の改善効果は顕著である．MGCG 法の Scalability は図 14（b）に示すように改善されている．4,096 ノード（65,536 コア）における計算時間は 12.5 秒（HB 4×4），11.8 秒（HB 8×2），12.5 秒（HB 16×1）である．既存手法[5]と比較して計算時間は 20%程度改善されている．

5 まとめ

ポストペタスケール環境における疎行列解法の確立を目標として，計算科学と計算機科学の両分野の緊密に協力し，数値計算アルゴリズムおよび通信の両方の観点から，大規模 HPC チャレンジに臨んだ．GeoFEM, pGW-MG の二つのアプリケーションが最大で 4,800 ノードおよび 4,096 ノードで行われた．一つ目のアプリケーション GeoFEM においては FX10 上で RDMA 機能に基づく Persistent 通信の利用が試みられた．アプリケーション側からだけのアプローチであれば，計算と通信のオーバーラップをはじめとするプログラム上での最適化が限界になってしまうことも多く，Isend/Irecv などの既存の通信関数以外を試みようとすることもなかなか難しい．MPI1.0 通信規格からあり，想定される通信パターンが適確であるにも関わらず使われていなかった Persistent Communication を実装に取り入れられたこと，さらにハードウェアに最適化された通信機能をもつ MPI ライブラリを活用しながらアプリケーションの最適化が試みられた事は大変有意義なことであったと同時に，今回のような通信分野とアプリケーション分野の緊密な連携が不可欠であると感じた．二つ目のアプリケーション pGW-MG では収束の安定性，通信オーバーヘッドの削減が期待される提案手法 Coarse Grid Aggregation が実施され既存手法と比較された．提案手法による収束の改善効果が得られ，20%程度の計算時間の改善が得られた．

最後に，Oakleaf-FX の全 4800 ノードが 24 時間利用できたことは規模・時間ともに大変恵まれた機会であった．HPC チャレンジの実施期間中，Oakleaf-FX が稼働開始してから間もなかったこともあり，昼夜を問わず富士通のサポートの方に対応いただいた．心より御礼を申し上げて，本稿のまとめとする．

参　考　文　献

[1] Yutaka Ishikawa, Kengo Nakajima, and Atsushi Hori, “Revisiting Persistent Communication in MPI,” EuroMPI 2012, 2012 (Poster).

[2] GeoFEM プロジェクト http://geofem.tokyo.rist.or.jp/

[3] Deutsch, C.V., Journel, A.G., GSLIB Geostatistical Software Library and User's Guide, Second Edition. Oxford University Press (1998).

[4] Nakajima, K., Parallel Multigrid Solvers using OpenMP/MPI Hybrid Programming Models on Multi-Core/Multi-Socket Clusters, Lecture Notes in Computer Science 6449, pp.185-199 (2011).

[5] Nakajima, K., New strategy for coarse grid solvers in parallel multigrid methods using OpenMP/MPI hybrid programming models, ACM Proceedings of the 2012 International Workshop on Programming Models and Applications for Multicores and Manycores, ACM Digital Library (DOI: 10.1145/2141702.2141713) (2012).

[6] Nakajima, K., OpenMP/MPI Hybrid Parallel Multigrid Method on Fujitsu FX10 Supercomputer System, IEEE Proceedings of 2012 International Conference on Cluster Computing Workshops, 199-206, IEEE Digital Library: 10.1109/ClusterW.2012.35 (2012).

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.Special Issue1 (2013.2)

スーパーコンピューティングニュース編集スタッフ
編　集　長　　中島研吾
編集幹事　　中村昇平
編集委員　　佐藤周行, 片桐孝洋, 田浦健次朗, 大島聡史, 實本英之,
鴨志田良和, 平野光敏, 石﨑　勉, 小川大典, 佐島浩之,
宮嵜　洋, 山本和男, 佐藤孝明, 高橋朝一, 諏佐賢司
編集・発行　　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部門
〒113-8658　東京都文京区弥生2-11-16
（電話）　03-5841-2717（ダイヤルイン）
（FAX）　03-5841-2708