サイエンティフック・システムズ研究会　　2013 年 10 月 23 日
科学技術計算分科会 2013 年度会合

京コンピュータを用いた宇宙プラズマの第一原理ブラソフシミュレーション

梅田　隆行
名古屋大学太陽地球環境研究所

[アブストラクト]
宇宙空間を満たす希薄な「無衝突プラズマ」の振る舞いを理解することは、私たちが住む宇宙の本質的な理解につながると共に、太陽の変動によって生じる地球周辺の環境変動の理解にとって極めて重要である。本稿では、第一原理宇宙プラズマシミュレーション手法の 1 つであるブラソフ（無衝突ボルツマン）コードについて紹介し、国内の様々な HPC プロジェクトや SS 研での活動を通じて行ってきた、超並列ブラソフコードの性能評価及び性能チューニング結果を示す。また、現在京コンピュータを用いて行っている大規模計算の実情を紹介し、エクサ及びその先に向けた展望を述べる。

[キーワード]
プラズマ、太陽地球惑星系科学、大規模シミュレーション、性能評価、京、FX10、FX1、CX400

１．はじめに

私たちが住む宇宙の 99.99%以上の体積は物質の第 4 状態であるプラズマ(電離気体)で占められている。宇宙空間に存在するプラズマの大部分は密度が非常に小さく無衝突状態にあり、宇宙プラズマという言葉は無衝突プラズマを指す。一方で、恒星や惑星などの天体近傍の高密度プラズマには衝突があり、無衝突プラズマと区別する意味で天体プラズマ等と呼ばれる。私たちが住む地球周辺の宇宙環境では、地球磁気圏(地球の固有磁場によって支配された領域)が太陽から放出された高速のプラズマ流である太陽風及び、太陽風が運ぶ惑星間空間磁場(太陽の固有磁場)によって複雑に変化している。太陽表面爆発(フレア)などのプラズマ放出現象をはじめとする太陽の様々な変動により、宇宙飛行士の被曝、人工衛星の故障や通信障害に繋がる地球磁気圏・電離圏の環境変動が引き起こされ、これを宇宙天気と呼ぶ。近年の国際宇宙ステーションでの活動や人工衛星の打ち上げなど、日本においても宇宙利用が現実的になってきており、プラズマ研究は、宇宙の構成要素の本質的に理解することのみならず、宇宙天気の予報・予測においても極めて重要である。

プラズマ科学は、ニュートン、ローレンツ、マックスウェルらの古典力学・電磁気学によって表される既知の方程式で記述できるため、応用科学的な側面を持っている。しかし、その複雑な非線形性(電磁場の変化とプラズマの物理量の変化が互いにフィードバックを掛け合う状態)ゆえに、計算機による解析は不可欠である。

太陽活動による地球磁気圏の変動は、マクロ(あるいはグローバル)スケール現象として磁気流体力学(MHD：磁場がプラズマと一緒に動く状態)方程式で記述できる。一方で地球磁気圏内には、プラズマの密度や温度などの物理パラメータが異なる様々な領域が生じる。その領域間の境界層で現れる不安定性(平衡状態の破れ)は，磁気圏の変動に大きな影響を与えていると考えられている。境界層不安定性はグローバル磁気圏構造に対して中間(メゾ)スケール現象と呼ばれ、基本的には MHD 方程式で記述できる。地球周辺のプラズマは人工衛星による科学探査よって直接観測することができ、天体等からの電磁波(光)の放射を観測する天文学に対して太陽地球惑星系科学が持つ大きな特徴と言える。近年の科学衛星による高精度な「その場(in situ)」観測では、中間スケールの不安定性におけるプラズマの粒子的振る舞いの重要性が報告されており、MHD 方程式で記述できるマクロ物理過程と粒子運動論方程式によって記述できるミクロ物理過程が結合していることを示唆している。すなわち、境界層においてプラズマの粒子的な振る舞いが境界層不安定性の成長を促進または減退し、地球磁気圏の変動に影響を与えるという、宇宙プラズマのマルチスケール／クロススケール性を意味しており、マルチスケールの磁気圏変動である宇宙天気を真に理解することが、太陽地球惑星系科学(宇宙プラズマ科学)の究極のゴールの一つと言える。このためには、全てのスケール(グローバル磁気圏からミクロ粒子まで)をシームレスに扱う運動論方程式(第一原理)によるシミュレーションが本質的であり、究極の第一原理プラズマシミュレーションの準備を行うことが本研究の位置付けである。ブラソフシミュレーション手法は、プラズマシミュレーションとしては「次々々」世代の技術にあるため、今後の*ゼタ・ヨタ*スケールコンピューティングに向けて、現存する超並列計算機上においてコードの性能評価(計算速度がどこまで速くなるか、及び、物理現象がどこまで正確に解けるか)を行うことが現段階での本研究の目的である。

２．ブラソフシミュレーション

2.1．運動論シミュレーション

プラズマの運動論シミュレーションには 2 つの手法がある。1 つは、プラズマ粒子であるイオンや電子などの個々の荷電粒子の運動をニュートン－ローレンツ方程式により解き進める粒子法である。格子点(Cell)上に定義された電磁場中を粒子が動きまわることから、PIC (Particle-In-Cell)法として広く知られている。しかし、宇宙空間に存在する膨大な数の荷電粒子を有限の計算機資源で扱うことは不可能であるため、ある程度まとまった数の荷電粒子の集団を 1 つの“超”粒子として扱う。PIC 法はその計算手法の完成度が高く、プラズマ科学分野では広く用いられている。しかし、プラズマを有限個の超粒子として扱うことにより、超粒子 1 つ 1 つが持つ雑音が大きくなることや、並列化の際にプロセス間の負荷のバランス(各プロセス内の粒子数の均一性)を保つことが困難なことなどの欠点がある。

もう 1 つの手法であるブラソフ法は、位置－速度位相空間に定義されたプラズマ粒子の分布関数の発展をブラソフ方程式により直接解き進める方法である。固定された格子点上

に定義された物理量のみを扱うため、PIC 法の問題点であった数値雑音や並列化困難さは解消される一方、最大で実空間 3 次元(x,y,z)及び速度空間 3 次元(vx,vy,vz)の計 6 次元空間を扱うため、コンピュータで解くには膨大なリソースを必要とする。このため、その計算手法の開発はほとんど進んでおらず、これまでは実用的な計算に対する動作検証がほとんどされないまま手法の提案だけが行われてきた。しかし、ここ数年の計算機環境の飛躍的に向上によって手法の開発が進み、実空間 2 次元及び速度空間 3 次元の 5 次元シミュレーションがようやく実用の域に達しつつある段階である。

2.2．計算手法の概要

無衝突プラズマの振る舞いは、以下のブラソフ(無衝突ボルツマン)方程式によって記述される。

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

ここで $\vec{E}$，$\vec{B}$，$\vec{r}$ と $\vec{v}$ はそれぞれ電場、磁場、位置、速度を表す。$f_s(\vec{r},\vec{v},t)$ は位置－速度位相空間におけるプラズマ粒子の分布関数であり、s はイオンや電子など粒子種を示す。また、q_s と m_s はそれぞれ電荷と質量を表す。

ブラソフ方程式(1)より、プラズマ粒子の分布関数が電磁場によって変形することが分かる。電磁場の時空間発展は、以下のマックスウェル方程式によって記述される。

$$\nabla \times \vec{B} = \mu_0 \vec{J} + \frac{1}{c^2}\frac{\partial \vec{E}}{\partial t} \qquad (2.1)$$

$$\nabla \times \vec{E} = -\frac{\partial \vec{B}}{\partial t} \qquad (2.2)$$

$$\nabla \cdot \vec{E} = \frac{\rho}{\varepsilon_0} \qquad (2.3)$$

$$\nabla \cdot \vec{B} = 0 \qquad (2.4)$$

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。ブラソフ方程式(1)を速度空間で積分して電荷 q_s を乗じると、以下の電荷保存則が得られる。

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \vec{J} = 0 \qquad (3)$$

電流密度 $\vec{J}$ はブラソフ方程式(1)の第二項にあたる実空間の流束 $\vec{v}f_s$ を速度空間で積分することによって求まり、電流密度 $\vec{J}$ が電荷保存則(3)を満足する限り、ポアソン方程式(2.3)は自動的に満たされる。プラズマの運動によって電流密度 $\vec{J}$ は生じ、またマックスウェル方程式(2.1)より電流密度 $\vec{J}$ によって電磁場が変化することが分かる。このようにプラズマ中では、プラズマの運動と電磁場の変化が互いにフィードバックを掛け合っている。

以上の方程式は無衝突プラズマの第一原理であり、ブラソフコードで扱う基礎方程式群である。

ブラソフ方程式は、位置－速度位相空間におけるプラズマ粒子の分布関数の保存則を表す方程式であり、数値補間を応用したセミラグランジュ法が古くから用いられてきた。しかし、4 次元以上の「超次元」をそのままの形で多次元数値積分(多次元補間)するのは非常に複雑であるため、演算子分離法(operator splitting)[Cheng and Knorr 1976]が古くから用いられてきた。本研究では、以下のように実空間移流、速度空間移流、速度空間回転の物理的に意味のある 3 つの演算子に分離する手法を開発している[Umeda et al. 2009]。

$$\frac{\partial f_s}{\partial t} + \vec{v}\frac{\partial f_s}{\partial \vec{r}} = 0 \tag{4.1}$$

$$\frac{\partial f_s}{\partial t} + \frac{q_s}{m_s}\vec{E}\frac{\partial f_s}{\partial \vec{v}} = 0 \tag{4.2}$$

$$\frac{\partial f_s}{\partial t} + \frac{q_s}{m_s}(\vec{v}\times\vec{B})\frac{\partial f_s}{\partial \vec{v}} = 0 \tag{4.3}$$

この演算子分離は、 PIC 法においてニュートン－ローレンツ式（荷電粒子の運動方程式）を時間 2 次精度で解く蛙飛(leap-frog)法と等価である。また実空間及び速度空間の移流方程式(4.1 及び 4.2)は、正値性を保証した保存型無振動補間[Umeda 2008; Umeda et al. 2012a]の 5 次精度バージョンを用いて時間積分を行っており、回転方程式(4.3)は PIC 法で広く用いられている Boris 法に基づいた back-substitution 法[Schmitz and Grauer 2006]を用いて時間積分を行っている。

一方、マックスウェル方程式(2.1)及び(2.2)は、FDTD（Finite Difference Time Domain）法と呼ばれる電磁場解析法を用いて解いている。FDTD 法では、Yee 格子[Yee 1966]と呼ばれる staggered 格子を用いており、式(2.4)が自動的に満たされるように物理量が配置されている。またブラソフ方程式を保存型解法により解く場合、電流密度 $\vec{J}$ は自動的に電荷保存則を満たすため、式(2.3)も自動的に満たされる。FDTD 法では leap-frog アルゴリズムに基づいて電場と磁場を半タイムステップずらしており、時空間精度は 2 次であるが、本研究では陰解法を用いている。

ブラソフシミュレーションでは非常に多くのメモリを必要とするため、並列計算が必須となる。ブラソフコードで使用する物理量は全て格子点上で与えられており、並列化においては領域分割法が有効である。図 1 は実空間 2 次元及び速度空間 3 次元を使用するブラソフコードにおける並列化の概念を示す。私たちの目(脳)は 4 次元以上の空間を認識できないが、2 次元実空間の各格子上に 3 次元速度空間(速度分布関数)が定義されていると考えると分かりやすい。本研究では図 1 のように実空間(x－y 平面)においてのみ MPI を用いた領域分割を行い、MPI_Sendrecv で袖領域のデータの送受を行う。一方で、速度空間は OpenMP によるスレッド並列のみ行う[Umeda et al. 2012b]。これは、電荷密度や電流密度などのモーメント量を計算する際に必要な速度空間の積分において、各実空間での MPI リダクション演算を行わないようにするためである。実際、速度空間の格子点数は 30—100 の 2—3 乗であり、並列化による高速化よりもオーバーヘッドの増加のほうが大きい可能性がある。また陰的 FDTD 法での収束レベルのチェックには MPI_Allreduce を用いる。

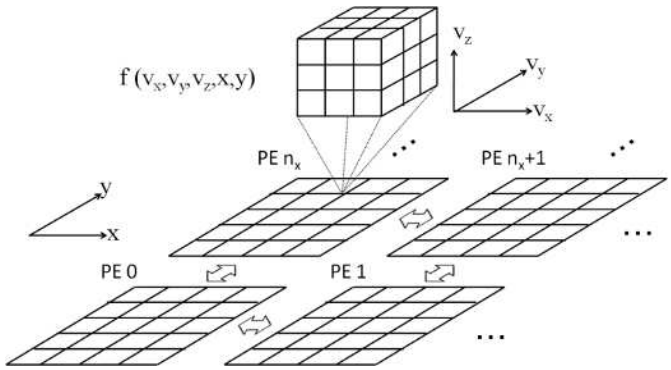

図 1：5 次元ブラソフコードにおける空間領域分割[Umeda et al. 2012b]**.**

３．性能チューニング

本節では、京/FX10 において有効であったチューニング法をいくつか挙げる。これらは、基本的には試行錯誤(trial and error)**であり、他の** CPU **とコンパイラ環境においては必ずしも有効であるとは限らないことを記しておく。**

3.1　配列のマージ

プログラム中ではしばしば、以下のように同一形状の配列を複数定義するときがある。

```
real(kind=8)::ax(ix,iy),ay(ix,iy),az(ix,iy)
```

これらの配列を図 2 **のように** 2 **通りの方法でマージする。なお、**C **プリプロセッサを用いると、プログラムの内部は変更せずに、配列の定義のみの書き換えを行うことができるので便利である。**Type A **は、配列をマージしない場合とほとんど変わらないが、マージすることにより配列へアクセスが** SIMD **化できるために積極的にマージしたほうがよい。しかし、**1 **次元目の長さが** 2 **の** n **乗前後の場合にキャッシュ競合が起こる可能性があるため、**1 **次元目の長さを試行錯誤で変える(パディングを行う)必要がある。**

一方 Type B **は、ループ内で配列** ax, ay, az **それぞれへのアクセス数が多い場合に、マージすることにより各配列へアクセスする命令の数(ストリーム数)が減り、また配列のパディングが不要になるというメリットがある。しかし、**L1 **キャッシュに読み込まれたデータがループ内で使用されない場合には、キャッシュヒット率が低下して性能が劣化する場合がある。究極的には両者は同等の性能になるが、プログラムの構造や** CPU **アーキテクチャによりどちらの性能が良いかは一概には言えない。したがって** ***マージ可能な全ての配列についてそれぞれ*****、どちらがより高速化を試す必要がある**[梅田 2013a]。

```
real(kind=8) :: array(ix,iy,3)
#define array(i,j,1) ax(i,j)
#define array(i,j,2) ay(i,j)
#define array(i,j,3) az(i,j)
```

図 2a**：配列のマージ、**Type A

```
real(kind=8) :: array(3,ix,iy)
#define array(1,i,j) ax(i,j)
#define array(2,i,j) ay(i,j)
#define array(3,i,j) az(i,j)
```

図 2b**：配列のマージ、**Type B

3.2 ループの分割

図 3 に(FX1 においてチューニングを行った)5 次元ブラソフコードのコア演算部のプログラム例を示す。実際には、図 3 のプログラムの外側に実空間座標 x,y 及び粒子種に関する 3 重ループ(ii, jj, ss)が存在するが、ここでは省略している。基本的な演算の流れは、加速度の計算(8 行目)と加速度に基づくデータの番地の計算(9-14 行目)、データの読み込み(16-20 行目)、数値フラックスの計算(22 行目) 及び、分布関数の更新(25-29 行目)である。

```
1   do nn=2,nvzp1
2      uz1=vz(nn,ss)
3      do mm=1,nvyp1
4         uy1=vy(mm,ss)
5         do ll=2,nvxp1
6            ux1=vx(ll,ss)
7
8            vv = ux1*bbx+uy1*bby+uz1*bbz
9            mv = sign(1.0d0,vv)
10           mp0=mm-floor(vv)
11           mp2=min(max(mp0+mv+mv,1),nvyp2)
12           mp1=min(max(mp0+mv   ,1),nvyp2)
13           mm1=min(max(mp0-mv   ,1),nvyp2)
14           mm2=min(max(mp0-mv-mv,1),nvyp2)
15
16           hp2=ff(ll,mp2,nn,ii,jj,ss)
17           hp1=ff(ll,mp1,nn,ii,jj,ss)
18           hp0=ff(ll,mp0,nn,ii,jj,ss)
19           hm1=ff(ll,mm1,nn,ii,jj,ss)
20           hm2=ff(ll,mm2,nn,ii,jj,ss)
21
22           dfy(ll,mm)=flux(hp2,hp1,hp0,hm1,hm2,vv)
23        end do
24     end do
25     do mm=2,nvyp1
26        ff(2:nvxp1,mm,nn,ii,jj,ss)=ff(2:nvxp1,mm,nn,ii,jj,ss) &
28                -(dfy(2:nvxp1,mm)-dfy(2:nvxp1,mm-1))
29     end do
30  end do
```

図 3：5 次元ブラソフコードのオリジナルプログラム

図 3 のプログラムでは、最内側ループ内に sign, floor, max, min などの組み込み関数の呼び出しとユーザ定義関数(22 行目)が含まれており、コンパイラの最適化を妨げている。性能チューニングとして、図 3 の 15 行目でループを分割し、図 4 のプログラムを作成した。

```
1   do nn=2,nvzp1
2     uz1=vz(nn,ss)
3     do mm=1,nvyp1
4       uy1=vy(mm,ss)
5       do ll=2,nvxp1
6         ux1=vx(ll,ss)
7
8         vv(ll,mm) = ux1*bbx+uy1*bby+uz1*bbz
9         mv = sign(1.0d0,vv(ll,mm))
10        mp0(ll,mm)=mm-floor(vv(ll,mm))
11        mp2(ll,mm)=min(max(mp0+mv+mv,1),nvyp2)
12        mp1(ll,mm)=min(max(mp0+mv   ,1),nvyp2)
13        mm1(ll,mm)=min(max(mp0-mv   ,1),nvyp2)
14        mm2(ll,mm)=min(max(mp0-mv-mv,1),nvyp2)
15      end do
16    end do
17    do mm=1,nvyp1
18      do ll=2,nvxp1
19        hp2=ff(ll,mp2(ll,mm),nn,ii,jj,ss)
20        hp1=ff(ll,mp1(ll,mm),nn,ii,jj,ss)
21        hp0=ff(ll,mp0(ll,mm),nn,ii,jj,ss)
22        hm1=ff(ll,mm1(ll,mm),nn,ii,jj,ss)
23        hm2=ff(ll,mm2(ll,mm),nn,ii,jj,ss)
24
25        dfy(ll,mm)=flux(hp2,hp1,hp0,hm1,hm2,vv(ll,mm))
26      end do
27    end do
28    do mm=2,nvyp1
29      ff(2:nvxp1,mm,nn,ii,jj,ss)=ff(2:nvxp1,mm,nn,ii,jj,ss) &
30              -(dfy(2:nvxp1,mm)-dfy(2:nvxp1,mm-1))
31    end do
32  end do
```

図 4：ループ分割を行った 5 次元ブラソフコードのプログラム

図 4 では、1 行目からのループで計算する vv, mp2, mp1, mp0, mm1, mm2 のデータを 17 行目からのループに受け渡す必要があり、nn(3 次元), mm(2 次元), ll(1 次元)のどのループ分割を行うかは重要な問題である。本研究のように比較的ループ長が短い(100×100×100 以下)場合、2 次元もしくは 1 次元でループ分割を行うのが良いことが分かった。3 次元でループ分割を行った場合は性能が大幅に劣化した。これは vv, mp2, mp1, mp0, mm1, mm2 の作業配列の領域が増え、キャッシュ領域を圧迫したことが原因だと考えている。

3.3　京／FX10 固有のチューニング[1]

図 3 のオリジナルプログラムを用いた場合、Fujitsu FX1(SPARC64 VII)では 14.%、Fujitsu CX400(Sandy Bridge)では 21%、Fujitsu FX10／京では 13%程度の実効効率であった。また図 4 のプログラムを用いた場合、Fujitsu FX1(SPARC64 VII)では 15.%、Fujitsu CX400(Sandy Bridge)では 22%、Fujitsu FX10／京では 14%程度の実効効率になり、全体的に性能が向上した。しかし、依然として FX10／京の性能が低いことが分かる。ここで、図 5 のようにプログラムを書き換えた場合、FX10／京では実効効率が 17%程度と大きく向上した。一方で、FX1 及び CX400 では図 3 のプログラムと同程度の実効効率となった。

図 4 のプログラムでは、配列の番地を計算している 11－14 行目において、整数の組み込み関数 max, min を用いている。一方で図 5 のプログラムでは、配列の番地を計算している 13－16 行目における max, min 組み込み関数は倍精度実数である。図 5 のプログラムでは、11 行目及び 13－16 行目において整数－倍精度実数の型変換があり、図 4 のプログラムと比べて演算が増加している。このため、FX1 及び CX400 では性能が劣化したが、FX10／京では元々の整数関数の性能が極端に低いため、型変換をしてでもループ内では実数関数を用いたほうが、性能が高くなることが分かった。

４．性能評価

1 コアあたりの格子数を 20×40×30×30×30 に固定し、コアあたりのメモリ使用量を 1GB に固定したときの、5 次元ブラソフコードの弱いスケーリング性能を図 6 及び図 7 に示す。図 6 はコア数に対する実効計算速度を表し、図 7 はコア数に対する実効並列化率を表す。計測に使用したシステムは、FX1(名大及び JAXA)、FX10(九大及び東大)、京(理研)及び CX400(九大)であるが、どのシステムにおいても、10^4 コアまでは性能がほぼスケールしていることが分かる。

図 7 より、FX1／京／FX10 では基本的には 95%程度の高いスケーラビリティを達成していることが分かる。一方で、所々で性能の劣化が見られる。FX1 では、全ノード(名大の 768 ノード及び JAXA の 3008 ノード)を使用した時に極端に性能が劣化する現象が見られた。一方で FX10 では、全ノード(九大の 768 ノード及び東大の 4800 ノード)を使用した時でも、性能がリニアにスケールした。ここで、九大の 768 ノード及び東大の 4800 ノードで性能に

[1] 3.2 及び 3.3 節については SS 研マルチコア性能 WG 成果報告書[梅田 2013b]にも同様の内容が記載されているが、プログラムの誤植が多いため、改めてここに記載する。

```
1   do nn=2,nvzp1
2     uz1=vz(nn,ss)
3     do mm=1,nvyp1
4       uy1=vy(mm,ss)
5       do ll=2,nvxp1
6         ux1=vx(ll,ss)
7
8         vv(ll,mm) = ux1*bbx+uy1*bby+uz1*bbz
9         sv = sign(1.0d0,vv(ll,mm))
10        mp0(ll,mm)=mm-floor(vv(ll,mm))
11        wmp0=mp0(ll,mm)
12        wnvyp2=nvyp2
13        mp2(ll,mm)=min(max(wmp0+sv+sv,1.0d0),wnvyp2)
14        mp1(ll,mm)=min(max(wmp0+sv   ,1.0d0),wnvyp2)
15        mm1(ll,mm)=min(max(wmp0-sv   ,1.0d0),wnvyp2)
16        mm2(ll,mm)=min(max(wmp0-sv-sv,1.0d0),wnvyp2)
17      end do
18    end do
19    do mm=1,nvyp1
20      do ll=2,nvxp1
21        hp2=ff(ll,mp2(ll,mm),nn,ii,jj,ss)
22        hp1=ff(ll,mp1(ll,mm),nn,ii,jj,ss)
23        hp0=ff(ll,mp0(ll,mm),nn,ii,jj,ss)
24        hm1=ff(ll,mm1(ll,mm),nn,ii,jj,ss)
25        hm2=ff(ll,mm2(ll,mm),nn,ii,jj,ss)
26
27        dfy(ll,mm)=flux(hp2,hp1,hp0,hm1,hm2,vv(ll,mm))
28      end do
29    end do
30    do mm=2,nvyp1
31      ff(2:nvxp1,mm,nn,ii,jj,ss)=ff(2:nvxp1,mm,nn,ii,jj,ss) &
32              -(dfy(2:nvxp1,mm)-dfy(2:nvxp1,mm-1))
33    end do
34  end do
```

図 5：型変換を行った 5 次元ブラソフコードのプログラム

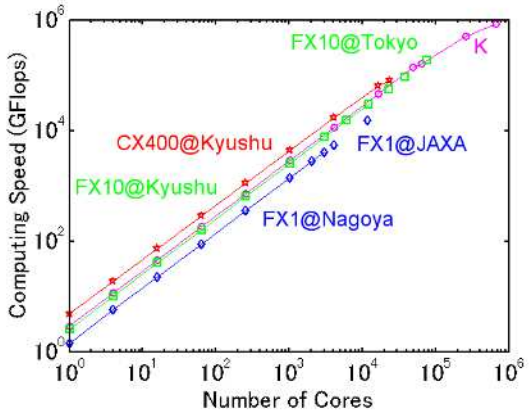

図 6：1GB/core 使用時の 5 次元ブラソフコードの弱いスケーリング性能。コア数に対する実効速度。

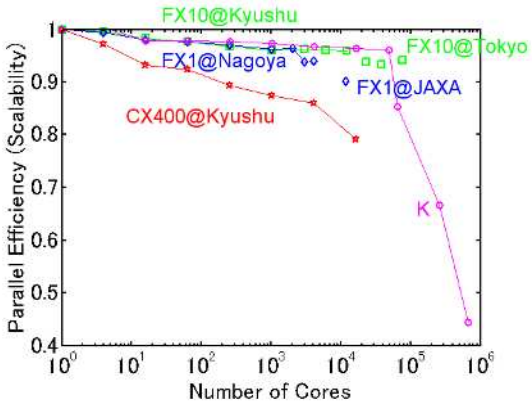

図 7：1GB/core 使用時の 5 次元ブラソフコードの弱いスケーリング性能。コア数に対する実効並列化率。

差があるのは、コンパイラバージョンの違いである。

一方、京では、実際に計算を行っている 6144 ノード(49,152 コア)までは性能がリニアにスケールしているが、8192 ノード以上で性能が徐々に低下し、82,944 ノードでは実効効率が 10%を下回り、残念ながら 1PFlops を達成できなかった。性能劣化の原因としては、電磁界ソルバ(陰的 FDTD 法)で用いている MPI_Allreduce の通信負荷が 8192 ノードから極端に高くなっていることが挙げられる[2]。

５．おわりに：「ペタ」の実際と「エクサ」に向けて

ブラソフコードは、宇宙空間に広く存在する無衝突プラズマの第一原理シミュレーション手法の１つである。ブラソフコードでは、プラズマ粒子は位置－速度位相空間における分布関数として定義され、超多次元の密度関数として与えられる。もう１つの第一原理商法である粒子コードと比べて、ブラソフコードは計算負荷が非常に高くその手法の開発やデバッグが困難であるため、計算手法は未だ発展途上にある。本研究では、新たに開発した 2 次元実空間及び 3 次元速度空間を扱う 5 次元ブラソフコードについて、国内の代表的な超並列計算機における性能評価及び性能チューニングを行っている。また、京コンピュータを用いて、隕石～小惑星サイズの“小天体”磁気圏構造に関するブラソフシミュレーションを行っている。実際の地球磁気圏のイオン粒子のパラメータを用いて 6 次元ブラソフシミュレーションを行おうとすれば、ゼタあるいはヨタスケールの計算リソースが必要である。

[2] 東大 FX10 の 4800 ノード(76,800 コア)では性能は劣化していないので、MPI_Allreduce の性能劣化の原因はノード数(の限界)であると考えられる。なお、陽的 FDTD 法を用いれば収束レベルのチェックのチェックは不要であるため、1PFlops を超えること自体は容易である。

現在行っている小天体の大規模(中規模??)計算では、京の 6144 ノード(49,152 コア)を用いている。これは、京の全計算リソースのうちの 7%未満であるが、実計算時間の最大が 24 時間であるのに対して、閑散期の待ち時間が 60 時間程度、繁盛期の待ち時間は 140 時間以上になる。運用上の問題でもあると思うが、ストレスのない待ち時間(24 時間程度)でジョブが流れるには、全計算リソースのうちの 4%未満のリソースでジョブを実行する必要があるであろう。

エクサスケールスーパーコンピュータを用いると、現在京を用いて行っている実空間 2 次元、速度空間 3 次元の計算をフル 6 次元(実空間 3 次元、速度空間 3 次元)に拡張できる。しかし、このままのハード・ミドル・ソフトウェアの構成でエクサスケール計算ができるかと問われると、答えは No である。30 の 5 乗格子点数の５次元計算には約 1GB のメモリ容量が必要となり、OS 領域を除いてコアあたり 1GB のメモリ容量を確保できる計算機上であれば動作は可能である。6 次元の場合、30 の 6 乗格子点数の計算には約 30GB のメモリ容量が必要となるため、ノードあたりのメモリ容量は OS 領域を除いて 30GB 以上(40 の 6 乗格子点数の計算には 120GB 以上)必要であり、ノードあたり共有メモリ容量がある程度確保できる計算機(少なくとも 128GB 程度)が欲しいところである。しかし、スーパーコンピュータのノードあたり共有メモリ容量は 32GB 程度のまま増えない傾向にあり、6 次元計算を行うにはループ長を減らすしかない。一方で、CPU や GPU/コプロセッサあたりコア数は増加しており、ループ長が高々20—30 しかないプログラムを高効率にスレッド化するのには限界がある。従って、多重ループを高効率にスレッド化する技術が必須であろう。

仮にノードあたり共有メモリ容量が増え、多重ループに対応したスレッド化技術が確立されたとしても、ノード数をさらに増やした計算はかなり困難である。本研究では 8000 ノード程度で MPI_Allreduce の性能劣化が見られたが、それよりもデータ処理が大きな問題となる。現在、京の 6144 ノードで行っている計算では、１物理変数あたり 6144 個のファイルが出力され、これを市販の(非並列化)数値計算ソフトウェアで図示すると、数時間かかる。即ち、解析環境として、データの読み込みとデータ処理が並列化された数値計算ソフトウェアとマルチディスプレイにそれぞれ出力できる並列分散型ワークステーションが必要となる。計算プログラムの開発・チューニングと合わせて解析環境(ソフト・ハード)の構築も同時に、個人研究者レベルで行うのは、エクサ時代ではほぼ不可能であると言える。

[謝辞]
本研究は、科学研究費補助金(No.23740367,No.25610144)よりサポートを受けた。ベンチマークテストに使用したスーパーコンピュータシステムの計算リソースは、名古屋大学太陽地球環境研究所 計算機利用共同研究、名古屋大学 HPC 連携研究プロジェクト、東京大学 大規模 HPC チャレンジ、九州大学 先端的計算科学研究プロジェクト、学際大規模共同利用・共同研究(jh130005) 及び、HPCI システム利用研究(hp120092)により提供された。また性能チューニングに際し、SS 研究会マルチコア性能 WG 及び富士通に協力を頂いた。

[**参考文献**]

Cheng, C. Z., Knorr, G. : The integration of the Vlasov equation in configuration space, *J. Comput. Phys.*, Vol.22, No.3, 330—351 (1976).

Schmitz, H., R. Grauer, R.: Comparison of time splitting and backsubstitution methods for integrating Vlasov's equation with magnetic fields, *Comput.Phys. Commun.*, Vol.175, No.2, 86—92 (2006).

Umeda, T.: A conservative and non-oscillatory scheme for Vlasov code simulations, *Earth Planets Space*, Vol.60, No.7, 773—779 (2008).

梅田隆行, **サイエンティフィック・システムズ研究会 マルチコア性能** WG **成果報告書「実践、アプリ高速化に向けて」**2.3 **章「３次元電磁界コード** FDTD3 **の測定評価」**, **サイエンティフィック・システムズ研究会** (2013a).

梅田隆行, **サイエンティフィック・システムズ研究会 マルチコア性能** WG **成果報告書「実践、アプリ高速化に向けて」**2.7 **章「宇宙プラズマ５次元ブラソフコード** Vlasov5 **の測定評価」**, **サイエンティフィック・システムズ研究会** (2013b).

Umeda, T., Fukazawa, K., Nariyuki, Y., Ogino, T.: A scalable full electromagnetic Vlasov solver for cross-scale coupling in space plasma, *IEEE Trans. Plasma Sci.*, Vol.40, No.5, 1421—1428 (2012b).

Umeda, T., Nariyuki, Y., Kariya, D.: A non-oscillatory and conservative semi-Lagrangian scheme with fourth-degree polynomial interpolation for solving the Vlasov equation, *Comput. Phys. Commun.*, Vol.183, No.5, 1094—1100 (2012a).

Umeda, T., Togano, K., Ogino, T.: Two-dimensional full-electromagnetic Vlasov code with conservative scheme and its application to magnetic reconnection, *Comput. Phys. Commun.*, Vol.180, No.3, 365—374 (2009).

Yee, K. S., Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media, *IEEE Trans. Antenn. Propagat.*, Nol.AP-14, No.3, 302—307 (1966).