Vol.2018-HPC-167 No.1
2018/12/17

Xeon プロセッサにおけるブラソフコードの性能チューニング

梅田 隆行[†1]

Vlasov コードは宇宙空間を満たす無衝突プラズマの第一原理シミュレーション手法である．Vlasov シミュレーションでは，位置及び速度で与えられる超多次元位相空間における荷電粒子の分布関数の時間発展を，運動論方程式により Euler 型の数値解法を用いて直接解き進めている．4 次元以上の空間を扱うシミュレーションでは，ノードあたり，あるいはコアあたりに使用できるメモリ容量の制限から，数値解法や性能チューニングにおいて様々な工夫が必要である．本研究グループはこれまでに様々な HPC 関連プロジェクトと通じて，Vlasov コードの性能チューニングを行ってきた．本講演では，Xeon Broadwell プロセッサ上において効果があったチューニングをいくつか紹介する．

Performance tuning of Vlasov code on the Xeon processor

TAKAYUKI UMEDA[†1]

Vlasov code is a first-principle simulation method for collisionless space plasma. The Vlasov code solves the time development of phase-space distribution functions of charged particles in hyper-dimensions based on fully kinetic equations with the Eulerian grids. Since the distribution functions are defined in more than four dimensions, the Vlasov code requires high-resolution and high-performance numerical schemes which should work in limited computational memory per node or per core. Performance of our Vlasov code has been tuned on various scalar CPU architectures under Japanese HPC projects. In the present study, the performance tuning of our Vlasov code is made on the Xeon Broadwell processor. Some tips for the performance tuning will be reported.

1. はじめに

我々が住む宇宙の 99.99%以上の体積はプラズマと呼ばれる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（無衝突プラズマ）を理解することは，宇宙の本質的な理解につながる．

我々が住む地球周辺の宇宙環境は，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）と，地球の固有磁場との相互作用によって複雑な磁気圏構造を形成している．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，これを宇宙天気とよぶ．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，宇宙天気の予報・予測に繋がる宇宙プラズマ研究は極めて重要である．

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．グローバル磁気圏構造に対して，境界層不安定性は中間（メゾ）スケール現象と呼ばれる．これらのグローバル及び中間スケールの現象は，粒子運動論を扱う方程式である Vlasov（無衝突 Boltzmann）方程式の 0 次・1 次・2 次のモーメントを取ることによって求められる磁気流体力学（MHD）方程式によって記述される．しかし，近年の科学衛星による高精度な「その場」観測では，中間スケールの不安定性において MHD 方程式で記述できる物理過程と粒子の運動論方程式によって記述できる物理過程が結合していることを示唆している．これらの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ためには，全てのスケールをシームレスに扱える運動論方程式（第一原理）によるシミュレーションが本質的である．

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，Newton－Coulomb－Lorentz 方程式により解き進める PIC（Particle－In－Cell）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，このように呼ばれている．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 法はその数値解法の完成度が高く，プラズマ科学分野では広く用いられている．しかし，プラズマを超粒子として扱うことにより熱雑音が大きくなること，電荷密度や電流密度などの荷電粒子の運動に起因する場の量を格子上に割り振る際に生じる高波数モードが数値誤差として蓄積すること，さらに並列化の際に負荷のバランス（各プロセス内の粒子数の均一性）を保つために特殊なデータの分割が必要になることなどの欠点がある．

†1 名古屋大学宇宙地球環境研究所
Institute for Space-Earth Environmental Research, Nagoya University

一方もう 1 つの手法である Vlasov 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 Vlasov 方程式により直接解き進める方法である．格子点上に定義された分布関数は熱雑音を持たず，また流体シミュレーションと同様に並列計算も容易である．しかし，Vlasov 方程式は実空間 3 次元及び速度空間 3 次元の計 6 次元を扱う方程式であり，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，その手法の開発はあまり進んでいない．実際，ここ数年の HPC プロジェクトによる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．

本研究の最終的な目的は，プラズマシミュレーションとしては「次々々」世代の技術にあたる第一原理 Vlasov シミュレーション手法を世界に先駆けて確立し，プラズマ科学に基づいた宇宙天気の実現に貢献することにある．そのための準備として，現存する超並列計算機上のおける 5 次元 Vlasov コードの性能評価及び性能チューニングを行っている．

これまでの研究において様々な超並列計算機での Vlasov コードの性能評価を行ってきた．本研究では，メニーコアプロセッサである Xeon Phi KNL（Knights Landing）における Vlasov コードの性能測定を行う．また，Xeon Broadwell プロセッサにおける性能との比較を行った．

2. 計算手法の概要

2.1 基礎方程式

無衝突プラズマの振る舞いは，以下の Vlasov（外力を電磁力とした無衝突 Boltzmann）方程式によって記述される．

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

ここで **E**，**B**，**r**および**v**はそれぞれ電場，磁場，位置，速度を表す．また，$f_s(\mathbf{r},\mathbf{v},t)$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種類を示す．またq_sとm_sはそれぞれ電荷と質量を表す．

プラズマ粒子の分布関数は，電磁場によって変形する．電磁場の時空間発展は以下の Maxwell 方程式によって記述される．

$$\nabla\times\mathbf{B} = \mu_0\mathbf{J} + \frac{1}{c^2}\frac{\partial \mathbf{E}}{\partial t} \tag{2.1}$$

$$\nabla\times\mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} \tag{2.2}$$

$$\nabla\cdot\mathbf{E} = \frac{\rho}{\epsilon_0} \tag{2.3}$$

$$\nabla\cdot\mathbf{B} = 0 \tag{2.4}$$

ここで**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．Vlasov 方程式(1)を速度空間で積分すると，以下の電荷保存則が得られる．

$$\nabla\cdot\mathbf{J} + \frac{\partial \rho}{\partial t} = 0 \tag{3}$$

Maxwell 方程式(2.1)に含まれる電流密度**J**はプラズマの運動によって生じ，これにより電磁場が変化する．電流密度**J**は Vlasov 方程式(1)の第二項にあたる実空間の流束$\mathbf{v}f_s$を速度空間で積分することによって求まり，電流密度**J**が電荷保存則(3)を満足する限り，Poisson 方程式(2.3)は自動的に満たされる．

以上の方程式は，Vlasov コードにおいて解いているプラズマ粒子の運動論方程式であり，無衝突プラズマの第一原理とよぶ．

2.2 数値解放の概要

Vlasov 方程式は 4 次元以上の「超次元」を扱う方程式であり，そのままの形で多次元数値積分を行うのは非常に困難であるため，演算子分離（operator splitting）法が古くから用いられてきた[1]．過去の研究では，各次元(x，y，z，v_x，v_y，v_z)それぞれを 1 次元移流方程式に分解する方法が採用されていたが，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 3 つの物理的な演算子に分離する手法を用いている[2].

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

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

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

この演算子分離は，PIC 法において Newton－Coulomb－Lorentz 式（荷電粒子の運動方程式）を時間 2 次精度で解く手法として広く用いられている leap-frog アルゴリズムに基づいている．

本研究では，演算子分離による数値拡散を抑制するために，多次元の線形移流方程式に対する演算子非分離（unspliting）法を新たに開発している[2]．また本研究では，無振動性及び正値性を保証するリミッタを新たに開発し，数値振動の抑制を行っている[3,4]．ここで無振動スキームとは，ある区間において新たな極値（極大，極小）を生じず，既に存在する極値は（できるだけ）減衰させないスキームであり，ENO/WENO 法はこれに該当するが，TVD 法は極地を鈍らせるために該当しない．

式(4.3)は荷電粒子の速度が磁力線により運動エネルギーを保ったまま変化する回転方程式を表す．直交座標系における回転方程式は剛体回転問題と等価であり，線形移流問題と同様に，数値計算において最も基本的であるが，計算精度が重要となる問題である．本研究で採用している back-substitution 法[5]では，Boris アルゴリズム[6]に基づいて速

度空間での粒子の軌道をバックトレースし，v_x，v_y，v_z 方向それぞれの演算子を分離して回転運動を解いている．剛体回転問題では，系の外側，即ち速度空間において速度が速くなればなるほど移動量（加速）は大きくなり，Courant 条件の影響を受けやすくなる点に注意が必要であり，今後，陰解法や演算子非分離法の開発が必要である．

以上のように，Vlasov 方程式の数値解法は未だ発展途上である．この大きな原因は，Vlasov コードで扱う次元が多いためであり，開発やデバッグにために大容量の共有メモリ環境が必要となるからである．

一方，Maxwell 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く．FDTD 法では，Yee 格子[7]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 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．

2.3 ハイブリッド並列

Vlasov 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．Vlasov 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 5 次元 Vlasov コードにおける並列化の概念を示す．我々の目は 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）においてのみ領域分割を行い，速度空間の領域分割は行わない[8]．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 reduction 処理を行わないようにするためである．

本研究グループの Vlasov コードでは，OpenMP によるスレッド並列も併用している．経験的に，Fujitsu FX シリーズにおいては，ハイブリッド並列のほうが flat-MPI 並列よりも効率的になる場合が多い．数年前では Xeon プロセッサ（SandyBridge，IvyBridge など）においても，ハイブリッド並列のほうが flat-MPI 並列よりも効率的になるケースが出てきた．また，京コンピュータ 6144 ノードの実利用経験より，IO 処理や分散ファイルのデータ解析などの観点からプロセス数をできるだけ減らしたほうが利点は大きい．スレッド並列はそのオーバーヘッドの大きさから，できるだけより外側のループで行うのが効率的である．しかし，Vlasov モデルは 4 次元以上の超次元を扱い，メモリ使用量が非常に多いため，速度空間の格子点を 30^3-60^3 に固定してコアあたりのメモリ使用量 1-4GB に設定しつつ，使用ノード数を増やして計算領域（実空間の格子数）を拡張していくのが実際の超並列計算機の利用方法である．近年の計算機においては，ノード内の共有メモリの容量は増えずにコア数のみが増加していく傾向にあるため，単一のループのみをスレッド化する単純な方法には限界がある．本研究グループの Vlasov コードでは，OMP DO ディレクティブの COLLAPSE オプションを最外側ループに挿入することにより，多重ループのスレッド化を行う．これにより，スレッド数を増やしたときに発生するオーバーヘッドを軽減することができる[9].

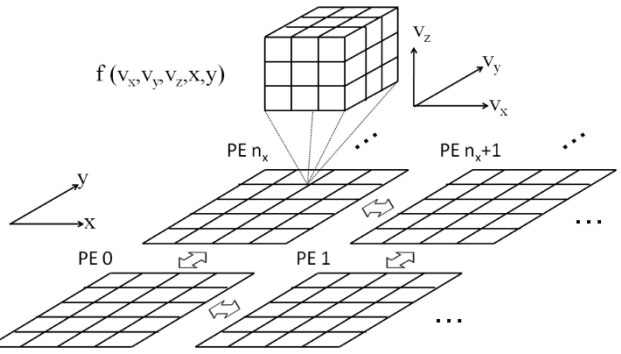

図 1　5 次元 Vlasov コードにおける空間領域分割[8].
Figure 1　The domain decomposition in the configuration space for the five-dimensional Vlasov code [8].

3. 性能チューニング

まず，本研究で使用した計算機環境は以下のとおりである．CPU として Xeon E5-2697 v4 (Broadwell, 2.3 GHz)を 2 つ搭載し，DDR4 の共有メモリを 512 GB 有する．CPU あたりのコア数は 18 であり，Hyper threading (HT)機能によりノード内で 72 スレッドを同時実行できる. またコンパイラは Intel Parallel Studio XE Cluster Edition Ver.17.0.1.132 を搭載し，コンパイラオプションとして“`-ipo -ip -O3 -xCORE-AVX2`”を使用した.

測定に使用した格子数は， Nx * Ny * Nvx * Nvy * Nvz *Ns= 126*66*40*40*40*2 であり, 使用したメモリ使用量は, 作業配列も含めると約 24 GB である．ここで，Nx,Ny,Nvx,Nvy,Nvz は x,y,z,v_x,v_y,v_z 方向の格子点数であり，Ns は粒子種(水素イオン，・電子など)の数を表す.

3.1 ループ分割

図 2 に，式(4, 2)を解くための改良前のプログラムを示す．このプログラムの前後には，実空間の x および y 座標に対する `ii` および `jj` の添え字に関するループがあり，5 重ループとなっているが，簡単のためにここでは省略している．このプログラムでは，3—29 行目において，v_x, v_y, v_z それぞれの方向に対する数値流束を，5 次精度の保存型・無振動スキーム[3,4]であるユーザ定義関数 `pic5` により計算

```
1  do nn=nvzs-1,nvze+1
2    do mm=nvys-1,nvye+1
3      do ll=nvxs-1,nvxe+1
4        ffi(ll)=1.0d0/ff(ll,mm,nn,ii,jj)
5      end do
6      do ll=nvxs-1,nvxe+1
7        hm2=ff(ll-lv-lv,mm,nn,ii,jj)
8        hm1=ff(ll-lv   ,mm,nn,ii,jj)
9        hp0=ff(ll      ,mm,nn,ii,jj)
10       hp1=ff(ll+lv   ,mm,nn,ii,jj)
11       hp2=ff(ll+lv+lv,mm,nn,ii,jj)
12       gfx(ll)=pic5(hp2,hp1,hp0,hm1,hm2,fex)
13     end do
14     do ll=nvxs-1,nvxe+1
15       hm2=ff(ll,mm-mv-mv,nn,ii,jj)
16       hm1=ff(ll,mm-mv   ,nn,ii,jj)
17       hp0=ff(ll,mm      ,nn,ii,jj)
18       hp1=ff(ll,mm+mv   ,nn,ii,jj)
19       hp2=ff(ll,mm+mv+mv,nn,ii,jj)
20       gfy(ll)=pic5(hp2,hp1,hp0,hm1,hm2,fey)
21     end do
22     do ll=nvxs-1,nvxe+1
23       hm2=ff(ll,mm,nn-nv-nv,ii,jj)
24       hm1=ff(ll,mm,nn-nv   ,ii,jj)
25       hp0=ff(ll,mm,nn      ,ii,jj)
26       hp1=ff(ll,mm,nn+nv   ,ii,jj)
27       hp2=ff(ll,mm,nn+nv+nv,ii,jj)
28       gfz(ll)=pic5(hp2,hp1,hp0,hm1,hm2,fez)
29     end do
30     do ll=nvxs-1,nvxe+1
31       gfxy  = abs((gfx(ll)*ffi(ll))* gfy(ll))               *0.5d0
32       gfyz  = abs((gfy(ll)*ffi(ll))* gfz(ll))               *0.5d0
33       gfzx  = abs((gfz(ll)*ffi(ll))* gfx(ll))               *0.5d0
34       gfxyz = abs((gfx(ll)*ffi(ll))*(gfy(ll)*ffi(ll))*gfz(ll))*inv3
35       dfx(ll+l0,mm   ,nn   ) = dfx(ll+l0,mm   ,nn   ) +(gfx(ll)+(gfxyz-gfxy-gfzx)*sx)
36       dfx(ll+l0,mm   ,nn+nv) = dfx(ll+l0,mm   ,nn+nv)         -(gfxyz-gfzx)     *sx
37       dfx(ll+l0,mm+mv,nn   ) = dfx(ll+l0,mm+mv,nn   )         -(gfxyz-gfxy)     *sx
38       dfx(ll+l0,mm+mv,nn+nv) = dfx(ll+l0,mm+mv,nn+nv)         + gfxyz           *sx
39       dfy(ll   ,mm+m0,nn   ) = dfy(ll   ,mm+m0,nn   ) +(gfy(ll)+(gfxyz-gfyz-gfxy)*sy)
40       dfy(ll+lv,mm+m0,nn   ) = dfy(ll+lv,mm+m0,nn   )         -(gfxyz-gfxy)     *sy
41       dfy(ll   ,mm+m0,nn+nv) = dfy(ll   ,mm+m0,nn+nv)         -(gfxyz-gfyz)     *sy
42       dfy(ll+lv,mm+m0,nn+nv) = dfy(ll+lv,mm+m0,nn+nv)         + gfxyz           *sy
43       dfz(ll   ,mm   ,nn+n0) = dfz(ll   ,mm   ,nn+n0) +(gfz(ll)+(gfxyz-gfzx-gfyz)*sz)
44       dfz(ll   ,mm+mv,nn+n0) = dfz(ll   ,mm+mv,nn+n0)         -(gfxyz-gfyz)     *sz
45       dfz(ll+lv,mm   ,nn+n0) = dfz(ll+lv,mm   ,nn+n0)         -(gfxyz-gfzx)     *sz
46       dfz(ll+lv,mm+mv,nn+n0) = dfz(ll+lv,mm+mv,nn+n0)         + gfxyz           *sz
47     end do
48   end do
49 end do
```

図 2　改良前の式(4,2)を解くための FORTRAN プログラムの一部.

Figure 2　A part of the “as-is” FORTRAN program for numerically solving Eq.(4.2).

している．そして 30—47 行目において，1 次元の数値流束から 3 次元の数値流束を文献[2]に基づいて合成している．

変数 lv,l0,mv,m0,nv,n0 は値として+1 または-1 のどちらかをとり，実空間の変数である電場ベクトルの符号に依存するために 1 行目の外側で計算される．ユーザ定義関数 pic5 には多くの FORTRAN 組み込み関数が含まれており，演算が重い．また 3—5 行目の割り算も重い演算であるため，演算が重いループを 1 次元で複数に分割することにより，それぞれのループに対するコンパイラの最適化を促進している．

一方で，30—47 行目にはあまり重い演算は含まれていないためにこれまではループ分割は行っていなかったが，35—46 行目にある 3 次元配列への累計計算において配列要素へのアクセスに依存関係があり，30 行目のループに対するコンパイラの最適化を阻害していた．

本研究ではまず，図 3 に示すように 30—47 行目を複数の 1 次元ループに分割した．図 3 のプログラムでは，各ループ内で配列 dfx, dfy, dfz それぞれにアクセスするのは各 1 回であり，配列要素へのアクセスに対して依存関係が無いため，それぞれのループについてコンパイラの最適化が行われた．ただし，変数 gfxy, gfyz, gfzx, gfxyz それぞれの計算結果を分割したループに渡す必要があるため，これらを新たな 1 次元配列として定義した．

図 2 のプログラムをプロセスあたりのスレッド数 18，2 プロセスで実行すると，1 タイムステップあたりの計算に約 4.16 秒掛かっていたが，この最適化によって図 3 のプログラムでは計算時間が約 2.22 秒に短縮され，約 1.87 倍の高速化に成功した．ループ分割には配列が増えることによって計算性能が劣化するデメリットがあるが，本例の場合にはコンパイラの最適化による性能改善のメリットの方が大きかった．

3.2 配列の削減

本研究では更なる高速化チューニングとして，図 4 に示すような配列の削減を行った．まず，図 2 のプログラムの 3—5 行目のループを図 3 のプログラムの 30—35 行目のループに融合し，配列 ffi をスカラ変数とした．また，変数 gfxy, gfyz, gfzx, gfxyz の計算を図 3 のプログラムの 36—55 行目の各ループ内で直接計算するようにした．これにより演算数量は増えるが，計算時間自体は約 1.75 秒に短縮され，約 1.27 倍の高速化に成功した．つまり，式(4.2)を解くプログラム全体においては，約 2.37 倍の高速化がなされた．

```
30   do ll=nvxs-1,nvxe+1
31     gfxy(ll)  = abs((gfx(ll)*ffi(ll))* gfy(ll))               *0.5d0
32     gfyz(ll)  = abs((gfy(ll)*ffi(ll))* gfz(ll))               *0.5d0
33     gfzx(ll)  = abs((gfz(ll)*ffi(ll))* gfx(ll))               *0.5d0
34     gfxyz(ll) = abs((gfx(ll)*ffi(ll))*(gfy(ll)*ffi(ll))*gfz(ll))*inv3
35   end do
36   do ll=nvxs-1,nvxe+1
37     dfx(ll+l0,mm   ,nn   ) = dfx(ll+l0,mm   ,nn   ) +(gfx(ll)+(gfxyz(ll)-gfxy(ll)-gfzx(ll))*sx)
38     dfy(ll   ,mm+m0,nn   ) = dfy(ll   ,mm+m0,nn   ) +(gfy(ll)+(gfxyz(ll)-gfyz(ll)-gfxy(ll))*sy)
39     dfz(ll   ,mm   ,nn+n0) = dfz(ll   ,mm   ,nn+n0) +(gfz(ll)+(gfxyz(ll)-gfzx(ll)-gfyz(ll))*sz)
40   end do
41   do ll=nvxs-1,nvxe+1
42     dfx(ll+l0,mm   ,nn+nv) = dfx(ll+l0,mm   ,nn+nv)         -(gfxyz(ll)-gfzx(ll))*sx
43     dfy(ll+lv,mm+m0,nn   ) = dfy(ll+lv,mm+m0,nn   )         -(gfxyz(ll)-gfxy(ll))*sy
44     dfz(ll   ,mm+mv,nn+n0) = dfz(ll   ,mm+mv,nn+n0)         -(gfxyz(ll)-gfyz(ll))*sz
45   end do
46   do ll=nvxs-1,nvxe+1
47     dfx(ll+l0,mm+mv,nn   ) = dfx(ll+l0,mm+mv,nn   )         -(gfxyz(ll)-gfxy(ll))*sx
48     dfy(ll   ,mm+m0,nn+nv) = dfy(ll   ,mm+m0,nn+nv)         -(gfxyz(ll)-gfyz(ll))*sy
49     dfz(ll+lv,mm   ,nn+n0) = dfz(ll+lv,mm   ,nn+n0)         -(gfxyz(ll)-gfzx(ll))*sz
50   end do
51   do ll=nvxs-1,nvxe+1
52     dfx(ll+l0,mm+mv,nn+nv) = dfx(ll+l0,mm+mv,nn+nv)          + gfxyz(ll)*sx
53     dfy(ll+lv,mm+m0,nn+nv) = dfy(ll+lv,mm+m0,nn+nv)          + gfxyz(ll)*sy
54     dfz(ll+lv,mm+mv,nn+n0) = dfz(ll+lv,mm+mv,nn+n0)          + gfxyz(ll)*sz
55   end do
```

図 3　改良後の式(4,2)を解くための FORTRAN プログラムの一部.

Figure 3　A part of FORTRAN program after performance tuning for numerically solving Eq.(4.2).

```
30   do ll=nvxs-1,nvxe+1
31     ffi = 1.0d0/ff(ll,mm,nn,ii,jj)
32     gfx(ll)  = abs(gfx(ll)*ffi)
33     gfy(ll)  = abs(gfy(ll)*ffi)
34     gfz(ll)  = abs(gfz(ll)*ffi)
35   end do
36   do ll=nvxs-1,nvxe+1
37     dfx(ll+l0,mm   ,nn   ) = dfx(ll+l0,mm   ,nn   ) + ff(ll,mm,nn,ii,jj) &
                              *gfx(ll)*(gfy(ll)*(gfz(ll)*inv3-0.5d0)+(1.0d0-gfz(ll)*0.5d0))*sx
38     dfy(ll   ,mm+m0,nn   ) = dfy(ll   ,mm+m0,nn   ) + ff(ll,mm,nn,ii,jj) &
                              *gfy(ll)*(gfz(ll)*(gfx(ll)*inv3-0.5d0)+(1.0d0-gfx(ll)*0.5d0))*sy
39     dfz(ll   ,mm   ,nn+n0) = dfz(ll   ,mm   ,nn+n0) + ff(ll,mm,nn,ii,jj) &
                              *gfz(ll)*(gfx(ll)*(gfy(ll)*inv3-0.5d0)+(1.0d0-gfy(ll)*0.5d0))*sz
40   end do
41   do ll=nvxs-1,nvxe+1
42     dfx(ll+l0,mm   ,nn+nv) = dfx(ll+l0,mm   ,nn+nv) - ff(ll,mm,nn,ii,jj) &
                              *gfx(ll)*gfz(ll)*(gfy(ll)*inv3-0.5d0)*sx
43     dfy(ll+lv,mm+m0,nn   ) = dfy(ll+lv,mm+m0,nn   ) - ff(ll,mm,nn,ii,jj) &
                              *gfy(ll)*gfx(ll)*(gfz(ll)*inv3-0.5d0)*sy
44     dfz(ll   ,mm+mv,nn+n0) = dfz(ll   ,mm+mv,nn+n0) - ff(ll,mm,nn,ii,jj) &
                              *gfz(ll)*gfy(ll)*(gfx(ll)*inv3-0.5d0)*sz
45   end do
46   do ll=nvxs-1,nvxe+1
47     dfx(ll+l0,mm+mv,nn   ) = dfx(ll+l0,mm+mv,nn   ) - ff(ll,mm,nn,ii,jj) &
                              *gfx(ll)*gfy(ll)*(gfz(ll)*inv3-0.5d0)*sx
48     dfy(ll   ,mm+m0,nn+nv) = dfy(ll   ,mm+m0,nn+nv) - ff(ll,mm,nn,ii,jj) &
                              *gfy(ll)*gfz(ll)*(gfx(ll)*inv3-0.5d0)*sy
49     dfz(ll+lv,mm   ,nn+n0) = dfz(ll+lv,mm   ,nn+n0) - ff(ll,mm,nn,ii,jj) &
                              *gfz(ll)*gfx(ll)*(gfy(ll)*inv3-0.5d0)*sz
50   end do
51   do ll=nvxs-1,nvxe+1
52     dfx(ll+l0,mm+mv,nn+nv) = dfx(ll+l0,mm+mv,nn+nv) +ff(ll,mm,nn,ii,jj) &
                              *gfx(ll)*gfy(ll)*gfz(ll)*inv3*sx
53     dfy(ll+lv,mm+m0,nn+nv) = dfy(ll+lv,mm+m0,nn+nv) +ff(ll,mm,nn,ii,jj) &
                              *gfx(ll)*gfy(ll)*gfz(ll)*inv3*sy
54     dfz(ll+lv,mm+mv,nn+n0) = dfz(ll+lv,mm+mv,nn+n0) +ff(ll,mm,nn,ii,jj) &
                              *gfx(ll)*gfy(ll)*gfz(ll)*inv3*sz
55   end do
```

図 4　更なる改良後の式(4,2)を解くための FORTRAN プログラムの一部.

Figure 4　A part of FORTRAN program after further performance tuning for numerically solving Eq.(4.2).

この他，同時に使用しないが形状（サイズ）が似た，以下のような複数の配列があるとき，

```
integer(kind=4) :: l0(nvxs-1:nvxe)
integer(kind=4) :: lx(nvxs-1:nvxe)
integer(kind=4) :: m0(nvxs  :nvxe)
integer(kind=4) :: my(nvxs  :nvxe)
integer(kind=4) :: n0(nvxs  :nvxe)
integer(kind=4) :: nz(nvxs  :nvxe)
```

まずこれらの配列を 1 つにまとめ，次にプリプロセッサディレクティブ`#define` を用いて，以下のように同時に使用しない配列の使いまわしを行うようにし，サイズの削減を行った.

```
integer(kind=4) :: itmp(nvxs-1:nvxe,2)
#define l0(x) itmp(x,1)
#define lx(x) itmp(x,2)
#define m0(y) itmp(y,1)
#define my(y) itmp(y,2)
#define n0(y) itmp(z,1)
#define nz(z) itmp(z,2)
```

同様の変更を式(4.1)および(4.3)を解くプログラムにも施

して計算時間を測定した結果，性能チューニング前のプログラム全体の 1 タイムステップあたりの計算時間が約 9.76 秒であったのに対し，以上のチューニングを行った後の計算時間は約 5.55 秒に短縮された．結果としてプログラム全体では約 1.75 倍の高速化に成功した．

4. おわりに

Vlasov 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である．プラズマは位置－速度位相空間における分布関数として定義され，多次元の Euler 変数として与えられる．多次元 Vlasov シミュレーションは計算負荷が非常に高く，その手法の開発やデバッグが容易ではないであるため，計算手法は未だ発展途上にある．本研究では，2 次元実空間及び 3 次元速度空間を扱う 5 次元 Vlasov コードについて，Xeon Broadwell プロセッサ上において性能測定およびチューニングを行った．

まず，配列要素へのアクセスに依存関係がある累計計算において，依存関係を無くすためのループ分割を行い，コンパイラによる最適化を促進させた．ループ分割では，ループ間のデータの橋渡しを行う新たな配列の増加により計算性能が劣化するデメリットもあるが，本例では結果として約 1.8 倍の高速化に成功した．さらに，ループ分割によって増えた配列のサイズを削減するための変更を行った結果，演算数は増えたが計算時間は 0.8 倍に短縮された．以上のチューニングの結果，プログラム全体として約 1.75 倍の高速化に成功した．

参考文献

1. Cheng, C. Z., Knorr, G.: The integration of the Vlasov equation in configuration space, *J. Comput. Phys.*, Vol.22, No.3, 330—351 (1976).
2. 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).
3. Umeda, T.: A conservative and non-oscillatory scheme for Vlasov code simulations, *Earth Planets Space*, Vol.60, No.7, 773—779 (2008).
4. 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 (2012).
5 . Schmitz, H., 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).
6. Boris, J. P.: Relativistic plasma simulation-optimization of a hybrid code, *Proc. Fourth Conf. Num. Sim. Plasmas*, ed. by J. P. Boris and R. A. Shanny, pp.3—67, Naval Research Laboratory, Washington D. C. (Nov. 1970).
7. 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).
8. 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 (2012).
9. Umeda, T., Fukazawa, K.: Hybrid parallelization of hyper- dimensional Vlasov code with OpenMP loop collapse directive, *Adv. Parallel Comput.*, Vol.27, 265—274 (2016).