

プログラムモデルの異なる PIC コードの性能測定

梅田 隆行^{†1}

Particle-In-Cell (PIC) プラズマ粒子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。PIC シミュレーションでは、電磁場による加速を記述する荷電粒子の運動方程式（ニュートン・ローレンツ式）を、電磁場の時間発展を記述するマクスウェル方程式と共に解き進めている。膨大な数の荷電粒子が、格子と与えられた電磁場を自由に動き回ることから、PIC コードにおいて高い計算性能を出すためには様々な工夫が必要である。本研究では、プログラム構造の異なるいくつかの PIC コードについて、単一ノードにおける OpenMP スレッド並列の強スケーリング性能を測定した。

Performance measurement of particle-in-cell (PIC) code with various program structures

TAKAYUKI UMEDA^{†1}

Particle-In-Cell (PIC) plasma simulation code is a first-principle method for collisionless space plasma. The PIC code solves the Newton-Lorentz equation (equation of motion for charged particles) for individual charged particles together with the Maxwell equations for electromagnetic fields. Since a huge number of charged particles move freely in the grid cells of electromagnetic fields, it is difficult to achieve a high performance of the PIC code on a massively-parallel scalar supercomputer. In the present study, the performance measurement of PIC codes with various program structures is made on the strong scaling of OpenMP thread parallelism on a single compute node.

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およ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、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およ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ブラソフ（無

衝突ボルツマン）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 つの手法があるが、本研究では、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 PIC (Particle-In-Cell) 法に注目する。格子点 (Cell) 上に定義された電磁場はマクスウェル方程式により解き進められ、Cell 中を粒子が自由に動きまわることから、古くから PIC 法と呼ばれている。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、PIC 法では、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 (super-particle) として扱う。PIC 法は 1960 年代より電波科学、プラズマ科学および地球惑星科学分野の研究者らにより開発されており [1,2]、その数値解法の完成度は非常に高く、近年ではプラズマ科学以外の分野にも幅広く応用されている。しかし、

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

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や、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などの欠点がある。

PIC 法では、ラグランジュ変数である粒子の位置および速度と、オイラー変数である電磁場が混在しており、スカラ型超並列計算機において高い性能を得るのは容易ではない。従来のプログラム構造では、粒子の番号に対して繰り返し演算を行っている。粒子は加速場に従って動くため、粒子が居る格子の番号は粒子の番号に無関係となる。したがって、粒子が格子上の場のデータにアクセスするときにはランダムアクセスとなり、キャッシュミスによる演算性能の低下の原因となる。一方で、分子動力学シミュレーションや天文学の N 体（重力多体）シミュレーションでは、粒子のデータを格子の番号順に並べ替え、かつ格子の番号順に繰り返し演算を行うことにより、高い演算性能を実現している[3]。

本研究では、格子の番号順に繰り返し演算を行う新しいプログラム構造をプラズマ PIC コードに導入し、その性能を測定する。

2. 計算手法の概要

2.1 基礎方程式

無衝突プラズマの運動は、以下の荷電粒子の運動（ニュートン・ローレンツ）方程式によって記述される。

$$\frac{d\mathbf{r}_p}{dt} = \mathbf{v}_p \quad (1.1)$$

$$\frac{d\mathbf{v}_p}{dt} = \frac{q_p}{m_p} (\mathbf{E} + \mathbf{v}_p \times \mathbf{B}) \quad (1.2)$$

ここで $\mathbf{E}$, $\mathbf{B}$, $\mathbf{r}$ および $\mathbf{v}$ はそれぞれ電場、磁場、位置、速度を表す。また、 q と m はそれぞれ電荷と質量を表し、添え字は p 番目の粒子であることを示す。

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マクスウェル方程式によって記述される。

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

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

$$\nabla \cdot \mathbf{E} = \frac{\rho}{\epsilon_0} \quad (2.3)$$

$$\nabla \cdot \mathbf{B} = 0 \quad (2.4)$$

ここで $\mathbf{J}$ 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。式(2.1)の発散をとり、

式(2.3)を代入すると、以下の電荷保存則が得られる。

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

マクスウェル方程式(2.1)に含まれる電流密度 $\mathbf{J}$ 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 $\mathbf{J}$ 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。

以上の方程式は、PIC コードにおいて解いている基礎方程式であり、無衝突プラズマの第一原理である。

2.2 数値解法の概要

PIC コードの計算は大きく分けて、式(2)の電磁場の更新（以下、Maxwell という）、式(1.2)の粒子の速度更新（以下、Velocity という）および、式(1.1)の粒子の位置更新と式(3)による電流密度の計算（以下、Current という）の3つのカーネルに分類できる。電流は荷電粒子の移動によって生じるものなので、式(3)を満たす電流密度は粒子の位置の更新に基づいて計算する必要がある。

Maxwell カーネルは、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[4]と呼ばれる staggered 格子を用いており、中心差分において式(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、時間と空間共に精度は2次である。

2.3 従来のプログラム構造

Velocity カーネルでは、Boris 法[5]と呼ばれる手法を用いており、leap-frog アルゴリズムに基づいて電磁場と速度が半タイムステップずれた、2次の時間精度になっている。以下は2次元コードにおけるプログラムの概要である。

```
!$OMP DO
DO p=1,np
  i = nint(x(p))
  j = nint(y(p))
  wx1 = ...
  wx2 = ...
  wx3 = ...
  wy1 = ...
  wy2 = ...
  wy3 = ...
  pex = ex(i-1,j-1)*wx1*wy1 &
    + ex(i,j-1)*wx2*wy1 &
    + ex(i+1,j-1)*wx3*wy1 &
    + ex(i-1,j)*wx1*wy2 &
    + ex(i,j)*wx2*wy2 &
    + ex(i+1,j)*wx3*wy2 &
    + ex(i-1,j+1)*wx1*wy3 &
```

```

+ ex(i ,j+1)*wx2*wy3 &
+ ex(i+1,j+1)*wx3*wy3
pey = ...
pez = ...
pbx = ...
pby = ...
pbz = ...
vx(p) = vx(p) + ...
vy(p) = vy(p) + ...
vz(p) = vz(p) + ...
END DO
!$OMP END DO

```

まず, p 番目の粒子が居る格子点の番号 (i, j) を計算し, その隣接格子への粒子の重み $(wx1, wx2, \dots, wy3)$ を計算する (具体的な重み計算については省略するが, $x(p)$ および $y(p)$ に依存した量となる). 次に, 各格子点の電磁場 (ex, ey, ez, bx, by, bz) に対してそれぞれ重みを掛け, 粒子の位置での電磁場を計算する. 最後に, 粒子の位置での電磁場より Boris 法[6]を用いて加速度を計算し, 速度を更新する (具体的な加速度の計算については省略する). 以上の作業を np 個の粒子に対して行っている. 粒子の番号 p と粒子が居る格子点の番号 (i, j) は無関係であるため, 格子点上の電磁場の配列へのアクセスはランダムロードになる. また OpenMP の DO 指示節によりスレッド並列化を簡単に行うことができる.

Current カーネルでは, 粒子の位置を式(1.1)に基づいて 2 次の時間精度の leap-frog アルゴリズムで更新する. また式 (3)に基づいて電流密度を計算する際に, 電荷保存と呼ばれる手法[7]を用いている. 以下は 2 次元コードにおけるプログラムの概要である.

```

!$OMP DO REDUCTION(+:jx,jy,jz)
DO p=1,np
  i = nint(x(p))
  j = nint(y(p))
  wx1 = ...
  wx2 = ...
  wx3 = ...
  wy1 = ...
  wy2 = ...
  wy3 = ...
  jx(i-1,j-1) = jx(i-1,j-1) + q(p)*wx1*wy1
  jx(i ,j-1) = jx(i ,j-1) + q(p)*wx2*wy1
  jx(i+1,j-1) = jx(i+1,j-1) + q(p)*wx3*wy1
  jx(i-1,j ) = jx(i-1,j ) + q(p)*wx1*wy2
  jx(i ,j ) = jx(i ,j ) + q(p)*wx2*wy2
  jx(i+1,j ) = jx(i+1,j ) + q(p)*wx3*wy2
  jx(i-1,j+1) = jx(i-1,j+1) + q(p)*wx1*wy3
  jx(i ,j+1) = jx(i ,j+1) + q(p)*wx2*wy3

```

```

jx(i+1,j+1) = jx(i+1,j+1) + q(p)*wx3*wy3

jy(i-1,j-1) = jy(i-1,j-1) + ...
jy(i ,j-1) = jy(i ,j-1) + ...
jy(i+1,j-1) = jy(i+1,j-1) + ...
:
jz(i-1,j-1) = jz(i-1,j-1) + ...
jz(i ,j-1) = jz(i ,j-1) + ...
jz(i+1,j-1) = jz(i+1,j-1) + ...
:
END DO
!$OMP END DO

```

Velocity カーネルと同様に, まず p 番目の粒子が居る格子点番号 (i, j) を計算し, その隣接格子への粒子の重み $(wx1, wx2, \dots, wy3)$ を計算する (具体的な重み計算については複雑なため省略する). 次に, 各格子点の電流密度 (jx, jy, jz) に対して, 粒子の位置での電流密度に重みを掛けて足し込む. 以上の作業を np 個の粒子に対して行っている. Velocity カーネルとは異なり, Current カーネルの格子点上の電流密度の配列へのアクセスはランダムロードおよびランダムストアになる. また OpenMP の DO 指示節によりスレッド並列化を行うが, 電流密度 (jx, jy, jz) の配列に対する各スレッドの書き込みが競合しないように REDUCTION 指示節を付ける必要がある.

従来のプログラム構造では, 粒子の番号 p と格子点の番号 (i, j) は無関係であり, 粒子の番号順に繰り返し演算を行うと電磁場や電流密度の配列へのアクセスがランダムになり, キャッシュミスが頻発する. 格子点の番号 (i, j) が連続的になるように粒子の配列を並べ替えることによりキャッシュミスを大幅に減らし, コードの性能を向上することができている[8,9].

2.3 新しいプログラム構造

以下は新しいプログラム構造を持つ 2 次元コードにおける Velocity カーネルの概要である.

```

!$OMP DO COLLAPSE(2) SCHEDULE(static,1)
DO j=1,ny
DO i=1,nx
  DO p=psta(i,j),psta(i,j)-1+np(i,j)
    wx1 = ...
    wx2 = ...
    wx3 = ...
    wy1 = ...
    wy2 = ...
    wy3 = ...
    pex = ex(i-1,j-1)*wx1*wy1 &
          + ex(i ,j-1)*wx2*wy1 &
          + ex(i+1,j-1)*wx3*wy1 &
          + ex(i-1,j )*wx1*wy2 &

```

```

+ ex(i ,j )*wx2*wy2 &
+ ex(i+1,j )*wx3*wy2 &
+ ex(i-1,j+1)*wx1*wy3 &
+ ex(i ,j+1)*wx2*wy3 &
+ ex(i+1,j+1)*wx3*wy3
pey = ...
pez = ...
pbx = ...
pby = ...
pbz = ...
vx(p) = vx(p) + ...
vy(p) = vy(p) + ...
vz(p) = vz(p) + ...
END DO
END DO
END DO
!$OMP END DO

```

新しいプログラム構造では、粒子のデータが格子の番号順に並べ替えられていることは必須である。ここで、 $psta(i,j)$ は格子点 (i,j) における先頭の粒子の番号を表し、 $np(i,j)$ 格子点 (i,j) における粒子数である。新しいプログラム構造では格子の番号順に繰り返し演算を行うため、格子点上の電磁場の配列へのアクセスは連続的になる。なお、OpenMP の COLLAPSE(2) 指示節は、 i および j に対する 2 重の繰り返し演算をスレッド化する。また、チャンクサイズを 1 とした静的スケジューリングを用いているが、これは、各格子点における粒子数は一般的には一様ではないのと、動的スケジューリングのオーバーヘッドが大きいためである。

以下は新しいプログラム構造を持つ 2 次元コードにおける Current カーネルの概要である。

```

!$OMP DO COLLAPSE(2) SCHEDULE(static,1) &
!$OMP REDUCTION(+:jx,jy,jz)
DO j=1,ny
DO i=1,nx
  cjx1=0.0; cjx2=0.0; cjx3=0.0; ...
  c jy1=0.0; c jy2=0.0; c jy3=0.0; ...
  cjz1=0.0; cjz2=0.0; cjz3=0.0; ...
  DO p=psta(i,j),psta(i,j)-1+np(i,j)
    wx1 = ...
    wx2 = ...
    wx3 = ...
    wy1 = ...
    wy2 = ...
    wy3 = ...
    cjx1 = cjx1 + q(p)*wx1*wy1
    cjx2 = cjx2 + q(p)*wx2*wy1
    cjx3 = cjx3 + q(p)*wx3*wy1

```

```

  cjx4 = cjx4 + q(p)*wx1*wy2
  cjx5 = cjx5 + q(p)*wx2*wy2
  cjx6 = cjx6 + q(p)*wx3*wy2
  cjx7 = cjx7 + q(p)*wx1*wy3
  cjx8 = cjx8 + q(p)*wx2*wy3
  cjx9 = cjx9 + q(p)*wx3*wy3

```

```

  c jy1 = c jy1 + ...
  c jy2 = c jy2 + ...
  c jy3 = c jy3 + ...
  :
  cjz1 = cjz1 + ...
  cjz2 = cjz2 + ...
  cjz3 = cjz3 + ...
  :

```

```

END DO
jx(i-1,j-1) = jx(i-1,j-1) + cjx1
jx(i ,j-1) = jx(i ,j-1) + cjx2
jx(i+1,j-1) = jx(i+1,j-1) + cjx3
jx(i-1,j ) = jx(i-1,j ) + cjx4
jx(i ,j ) = jx(i ,j ) + cjx5
jx(i+1,j ) = jx(i+1,j ) + cjx6
jx(i-1,j+1) = jx(i-1,j+1) + cjx7
jx(i ,j+1) = jx(i ,j+1) + cjx8
jx(i+1,j+1) = jx(i+1,j+1) + cjx9

jy(i-1,j-1) = jy(i-1,j-1) + c jy1
jy(i ,j-1) = jy(i ,j-1) + c jy2
jy(i+1,j-1) = jy(i+1,j-1) + c jy3
:
jz(i-1,j-1) = jz(i-1,j-1) + cjz1
jz(i ,j-1) = jz(i ,j-1) + cjz2
jz(i+1,j-1) = jz(i+1,j-1) + cjz3
:

```

END DO

END DO

!\$OMP END DO

Velocity カーネルと同様に、格子の番号順に繰り返し演算を行うため、格子点上の電流密度の配列へのアクセスは連続的になる。またこのプログラムでは、 p に対する繰り返し演算において先にスカラ量ヘータを書き込むことにより、メモリに対するアクセスを削減している。

この新しいプログラム構造は、分子動力学シミュレーションや天文学の N 体シミュレーションにおいて既に導入されており、高い演算性能を実現している[3]。また近年では、プラズマ PIC シミュレーションにも導入されている[10]。

3. 性能測定

性能測定は、Xeon E5-2697 v4 (Broadwell, 2.3 GHz, 18 cores)プロセッサを 2 基搭載する単一の計算ノードを用いて行った。コンパイラは Intel Parallel Studio XE Ver.17 であり、コンパイルオプションは“-ipo -ip -O3 -xCORE-AVX2 -qopenmp”である。

格子点数は $N_x \times N_y = 500 \times 500$ 、粒子は 2 種（電子および正電荷を持つイオン）、粒子数は粒子種あたり $N_p = 25,000,000$ 個（格子あたり 100 個）であり、計算ジョブのサイズは作業配列を含めて約 4 GB である。各粒子種に 1 プロセスを割り当て（プロセス数を 2 に固定し）、スレッド数を 1 から 18 まで変化させ強スケーリング測定を行った。

図 1 の○, □, ×マークはそれぞれ、粒子の番号 p が格子点の番号 (i, j) に関係なく従来のプログラム構造を持つコード、粒子の番号 p を格子点の番号 (i, j) に基づいて並べ替えを行った従来のプログラム構造を持つコード、新しいプログラム構造を持つコード（粒子の番号は並べ替えられている）の時間ステップ数 100 に対する実行時間を表す。なお、粒子の番号の並べ替えにはスレッド並列化した計数ソート[11]を用いた。

粒子の番号 p が格子点の番号 (i, j) に関係ない場合（ランダムアクセス）においても Velocity カーネルは非常に高いスケーラビリティを示した。一方で、Current カーネルは 8 スレッドまではスケールしたが、スレッド数を増やすと性能が劣化した。Current カーネルはランダムロードとストアの両方を含むが、Velocity カーネルはランダムロードのみであり、メモリ上のデータを更新しないランダムロードのスケーラビリティは高いことが示された。全体としてスレッド数が 8 の場合が最も高速であったが、最適なスレッド数はキャッシュメモリのサイズやメインメモリのバンド幅に依存する。

粒子の番号を格子点の番号順に並べ替えた場合、Velocity カーネルと Current カーネル共に約 2 倍の性能向上が見られた。また、スレッド数を増やしたときの Current カーネルの性能劣化も少なかった。ただし、粒子データの並べ替え（Sort カーネル）に Velocity カーネルと同等の計算負荷がかかっているため、プログラム全体としては粒子の番号を並べ替えない場合（8 スレッド）に対して粒子の番号を並べ替えた場合は約 2 倍高速であった。

新しいプログラム構造を持つコードの Velocity カーネルは、粒子の番号を格子点の番号順に並べ替えた従来のプログラム構造のコードに対して約 2 倍高速であった。一方で、新しいプログラム構造を持つコードの Current カーネルは、粒子の番号を格子点の番号順に並べ替えた従来のプログラム構造のコードに対して約 5 倍高速であったが、スレッド数を増やすと性能が劣化した。この性能劣化の原因は OpenMP の reduction のオーバーヘッドが挙げられる。

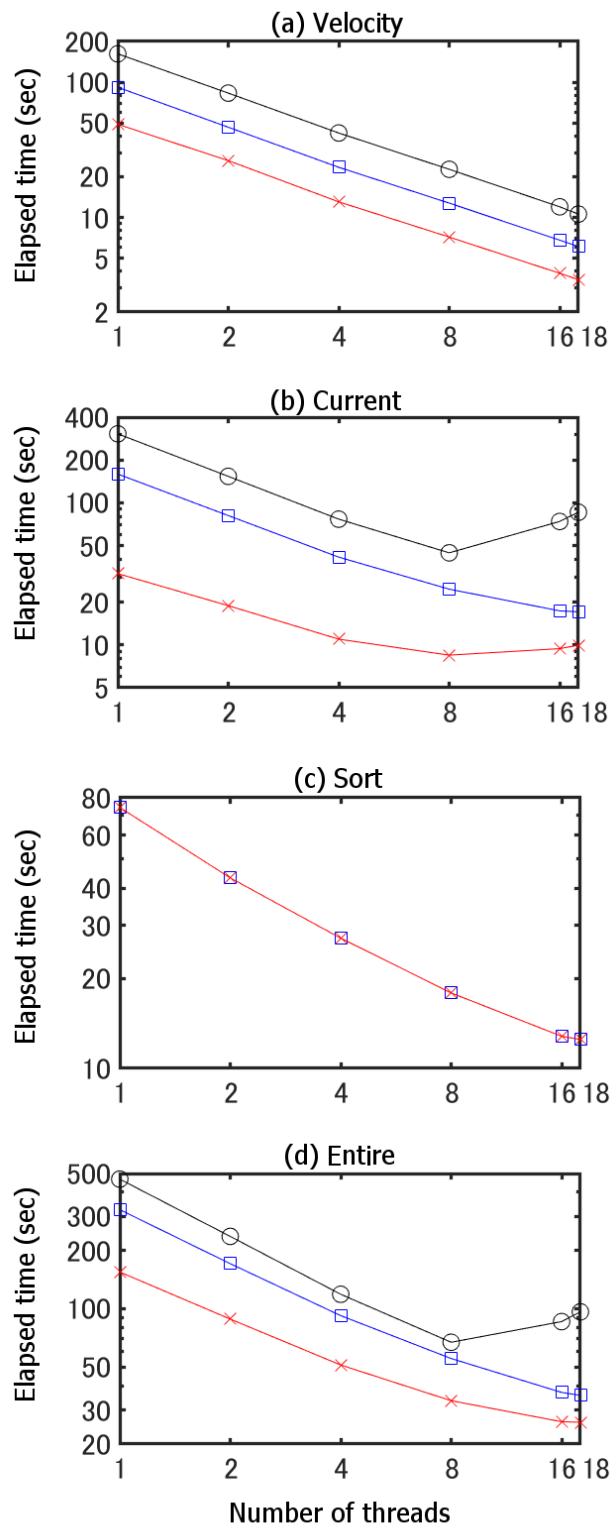


図 1 : 50,000,000 個の粒子を時間ステップ数 100 で解いた場合の各カーネルおよびプログラム全体の経過時間。○, □, ×マークはそれぞれ、粒子の番号が格子点の番号に関係なく従来のプログラム構造を持つコード、粒子の番号を格子点の番号に基づいて並べ替えを行った従来のプログラム構造を持つコード、新しいプログラム構造を持つコードの計測結果を表す。

本性能測定では, **reduction** に単位時間ステップあたり約 0.1 秒掛かっていることが推定される. また **Sort** カーネルは, 粒子の番号を格子点の番号順に並べ替えた従来のプログラム構造のコードと同じであり, プログラム全体としては, 粒子の番号を格子点の番号順に並べ替えた従来のプログラム構造のコードに対して約 1.5 倍高速であった. したがって, 粒子の番号を並べ替えない場合 (8 スレッド) に対して新しいプログラム構造を持つコードは約 3 倍高速であった.

4. おわりに

PIC 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のみならず, プラズマ科学分野外でも広く用いられている. プラズマは通常の荷電粒子の集団をまとめた超粒子として定義される. PIC シミュレーションはラグランジュ変数とオイラー変数が混在しており, 計算性能の向上が困難である. 本研究では特に, PIC コードのスカラ性能およびスレッド性能に着目した性能測定を行った.

格子データへのランダムアクセスを無くすように粒子

データをソートすることにより, スカラ性能のみならずスレッド性能も向上することが分かった. また粒子データのソーティングの効果は, ランダムロードよりもランダムストアに対してより有効であることが分かった.

格子に対して繰り返し演算を行う新しいプログラム構造は, 粒子のみに対して繰り返し演算を行う従来のプログラム構造に対して数倍高速であることが示された. 一方で, 新しいプログラム構造は粒子データのソートが必須であり, 計算カーネル部の高速化に伴ってソートカーネルの負荷が目立つようになった. 今後, より高速かつスケーラビリティの高いスレッド並列ソーティングアルゴリズムを探す必要がある. またプログラムの高速化に伴って, 電流密度の計算における OpenMP の **reduction** 演算のオーバーヘッドも目立つようになった. 今後, ループタイリングなどによる演算順序の変更により, **reduction** 演算を用いない計算法に変更する必要がある.

謝辞 本研究は科学研究費補助金・基盤研究(B) No.JP19H01868 によりサポートを受けている. また, 本研究は名古屋大学宇宙地球環境研究所の計算機利用共同研究課題として行った.

参考文献

1. Hockney, R. W., Eastwood, J. W.: *Computer Simulation Using Particles*, McGraw-Hill, New York (1981).
2. Birdsall, C. K., Langdon, A. B.: *Plasma Physics via Computer Simulation*, McGraw-Hill, New York (1985).
3. Mattson, W., Rice, B. M.: Near-neighbor calculations using a modified cell-linked list method, *Comput. Phys. Commun.*, Vol.119, 135—148 (1999).
4. Yee, K. S., Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media, *IEEE Trans. Antenn. Propagat.*, Vol.AP-14, 302—307 (1966).
5. 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).
6. Umeda, T., Omura, Y., Tominaga, T., Matsumoto, H.: A new charge conservation method in electromagnetic particle-in-cell simulations, *Comput. Phys. Commun.*, Vol.156, 73—85 (2003).
7. Decyk, V. K., Karmesin, S. R., deBoer, A., Liewer, P. C.: Optimization of particle-in-cell codes on reduced instruction set computer processors, *J. Comput. Phys.*, Vol.10, 290—298 (1996).
8. Bowers, K.: Accelerating a particle-in-cell simulation using a hybrid counting sort, *J. Comput. Phys.*, Vol.173, 393—411 (2001).
9. Nakashima, H., Summura, Y., Kikura, K., Miyake, Y.: Large scale manycore-aware PIC simulation with efficient particle binning, *Proc. 2017 IEEE Int. Parallel Distrib. Proc. Symp.*, 202—212 (2017).
10. Umeda, T., Oya, S.: Performance comparison of parallel sorting with OpenMP, *Proc. 3rd Int. Symp. Comput. Network.*, 334—340 (2015).