Vol.2020-HPC-173 No.14
2020/3/17

プログラムモデルの異なる PIC コードの性能測定（２）

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 (Newton-Lorentz 式)を，電磁場の時間発展を記述する Maxwell 方程式と連立させて解き進めている．膨大な数の荷電粒子が格子上に与えられた電磁場中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．前回は，並べ替えられた粒子のリスト持ち，かつ格子および粒子のリストに沿った多重ループ構造を持つ PIC コードが，従来の粒子のリストに沿ったループ構造を持つ PIC コードに対して，ノードあたり 4 倍以上高速であることを示した．しかし，OpenMP の reduction が非常に高いオーバーヘッドを持つことも分かった．本研究では，multi-color ordering により reduction 演算を用いないプログラムを新たに採用し，単一ノードにおいて更なる性能測定を行う．

Performance measurement of particle-in-cell (PIC) code with various program structures. 2

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 previous study, performance of a new program structure with multiple loop iterations over lists of both grids and particles together with sorted list of particles has been measured. It is shown that the new program structure is four times firster than the conventional program structure with unsorted list of particles on a single compute node. On the other hand, a large overhead at the OpenMP reduction has been seen. In the present study, further performance measurement is conducted on a single compute node by using a new program, in which reduction operations are not used by adopting multi-color ordering.

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メゾスケールの現象は,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ッ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

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

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，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]．また近年では，この格子の番号順に繰り返し演算を行うプログラム構造はプラズマ PIC シミュレーションにも導入されており[4]，本研究グループにおいても，従来のプログラム構造に対してノードあたり 4 倍以上高速であることを示した[5]．一方で，後述する OpenMP の reduction 演算のオーバーヘッドにより，スケーラビリティの劣化が見られた．

本研究では，OpenMP の reduction 演算のオーバーヘッドを無くて性能向上を図るために，multi-color ordering を導入する．また，reduction 演算を用いるプログラムとの性能比較を行う．

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（Newton-Lorentz）方程式によって記述される．

$$\frac{\mathrm{d}\boldsymbol{r}_p}{\mathrm{d}t} = \boldsymbol{v}_p \tag{1.1}$$

$$\frac{\mathrm{d}\boldsymbol{v}_p}{\mathrm{d}t} = \frac{q_p}{m_p}\left(\boldsymbol{E} + \boldsymbol{v}_p \times \boldsymbol{B}\right) \tag{1.2}$$

ここで $\boldsymbol{E}$，$\boldsymbol{B}$，$\boldsymbol{r}$ および $\boldsymbol{v}$ 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 Maxwell 方程式によって記述される．

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

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

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

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

ここで $\boldsymbol{J}$ 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．

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

マックスウェル方程式(2.1)に含まれる電流密度 $\boldsymbol{J}$ 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 $\boldsymbol{J}$ 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[6]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[7]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,..., 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[7]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[8]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,...,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ることが知られている[9,10].

2.4 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; ...
  cjy1=0.0; cjy2=0.0; cjy3=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

    cjy1 = cjy1 + ...
    cjy2 = cjy2 + ...
    cjy3 = cjy3 + ...
    :
    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) + cjy1
  jy(i  ,j-1) = jy(i  ,j-1) + cjy2
  jy(i+1,j-1) = jy(i+1,j-1) + cjy3
  :
  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へデ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．

2.5 Multi-color ordering

前節で示した Current カーネルには OpenMP の reduction 演算が含まれ，各スレッドが有する粒子が作る電流密度の累積値の配列を全スレッドで合計している．

図 1 は，粒子の番号 p が格子点の番号(i,j)に無関係かつ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（□）および，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）について，スレッド数を 1 から 18 まで変化させて時間ステップ数 100 に対する実行時間を表した強スケーリング測定の結果である[5]．ここで，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．また，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．

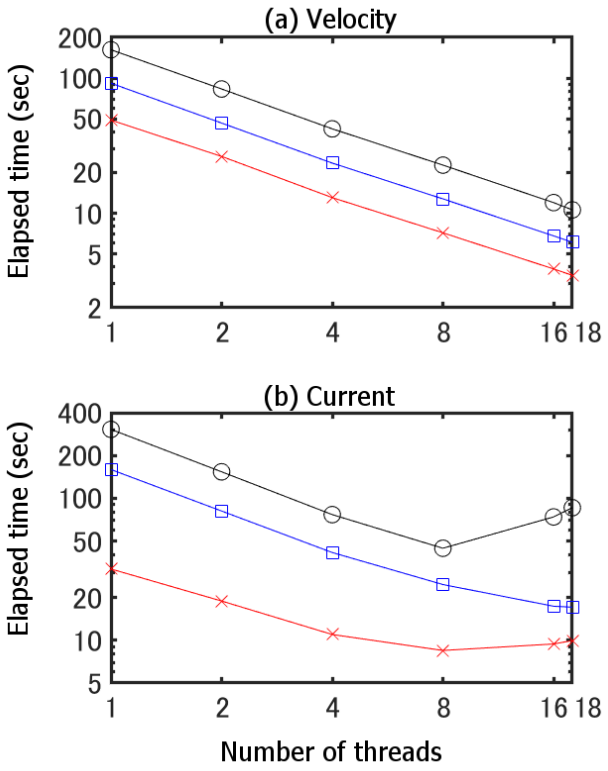

図 1：50,000,000 個の粒子を時間ステップ数 100 で解いた場合の Velocity および Current カーネルの経過時間．文献[5]より抜粋．○，□，×マークはそれぞれ，粒子の番号が格子点の番号に無関係かつ従来のプログラム構造を持つコード，粒子の番号を格子点の番号に基づいて並べ替えを行った従来のプログラム構造を持つコード，新しいプログラム構造を持つコードの計測結果を表す．

メモリからの配列データの読み込みのみである Velocity カーネルでは，全てのプログラムにおいて性能がスケールしている一方で，配列データの読み書きがある Current カーネルでは 8 スレッドあたりで性能の劣化が見られる．特に，並べ替えを行った粒子のリストを持つ 2 つのプログラム（□および×）では，実行時間 20 秒付近においてスケーラビリティの低下が見られた．つまり，時間ステップあたり 0.1 秒のオーダーで OpenMP の reduction 演算のオーバーヘッドが存在することを示唆している．

前節で示した Current カーネルでは，格子点(i,j)に対して前後 1 点ずつの計 9 点（3 次元では計 27 点となる）の電流密度配列のデータを更新している（前後に何点を更新するのかは重みの空間精度の次数に依存しており，前節のプログラムでは空間 2 次精度である）。このため，格子点(i,j)と(i+1,j)を別のスレッドで同時に計算するとアクセス競合が起こるため，reduction 演算が必要である．本研究では，格子点(i,j)と(i+3,j)のように 3 点ごとに計算を行うことによりアクセス競合を無くす．以下に示すプログラムのように，2 次元では格子点を 9 色に分け，配列へのアクセス競合が起こらないように計算順序を変更する．

```
DO j0=0,2
DO i0=0,2
!$OMP DO COLLAPSE(2) SCHEDULE(static,1)
DO j=1+j0,ny,3
DO i=1+i0,nx,3
  cjx1=0.0; cjx2=0.0; cjx3=0.0; ...
  cjy1=0.0; cjy2=0.0; cjy3=0.0; ...
  cjz1=0.0; cjz2=0.0; cjz3=0.0; ...
  DO p=psta(i,j),psta(i,j)-1+np(i,j)
    wx1 = ...
    wx2 = ...
    wx3 = ...
    wy1 = ...
    wy2 = ...
    wy3 = ...
  :
END DO
END DO
!$OMP END DO
END DO
END DO
```

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$N_x \times N_y = 1000 \times 1000$，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$N_p =$ 150,000,000個（格子あたり 150 個）であり，計算ジョブのサイズは作業配列を含めて約 24 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 2 の×および○マークはそれぞれ，2.4 節で示した reduction 演算を用いる Current カーネルおよび 2.5 節で示した multi-color ordering を用いた Current カーネルの時間ステップ数 100 に対する実行時間を表す. また参考値として，プログラム全体の実行時間を破線で示す．

図 1 の計測に対して 4 倍の格子を用いているため，reduction 演算を用いた Current カーネルは実行時間 40 秒付近において性能が飽和しており，時間ステップあたりの reduction のオーバーヘッドが約 0.4 秒であることを示唆している．一方で，multi-color ordering を用いた Current カーネルは，シリアル実行のときは reduction 演算を用いたカーネルよりもやや遅くなっており，3 重ループから 5 重ループになったことによりオーバーヘッドが増えた可能性や配列へのストライドアクセスが原因として考えられる．しかし，multi-color ordering を用いた Current カーネルは良くスケールしており，18 スレッドを用いた場合に，reduction 演算を用いたカーネル（の 8 スレッドを用いた場合）に対して約 2 倍高速となった．

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 コードのマルチコア CPU スカラにおける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し，また格子に対して繰り返し演算を行うプログラム構造は，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 4 倍以上高速である．一方で，多くのスレッドを用いた場合に，電流密度の計算における OpenMP の reduction 演算のオーバーヘッドにより性能の飽和が見られた．本研究では，multi-color ordering の採用により，reduction 演算を用いない電流密度の計算方法を提案し，スレッド数を増やしても性能がスケールすることを示した．

最後に，本研究で提案した multi-color ordering を用いる電流密度の手法は，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は用いることはできず，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においてのみ用いることができるを追記する．

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実施された．

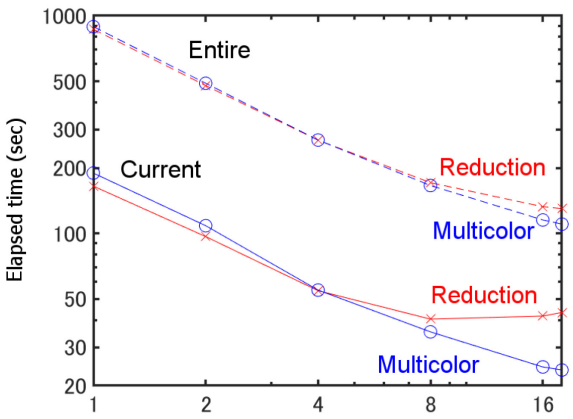

図２：300, 000, 000 個の粒子を時間ステップ数 100 で解いた場合の Current カーネルおよびプログラム全体の経過時間．×および○マークはそれぞれ，reduction 演算を用いるプログラムおよび multi-color ordering を用いるプログラムの計測結果を表す．

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. 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).
5. 梅田 隆行: プログラムモデルの異なる PIC コードの性能測定, *研究報告ハイパフォーマンスコンピューティン(HPC)*, 2019-HPC-172(16), 1—6 (2019).
6. 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).

7. 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).
8. 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).
9. 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).
10. Bowers, K.: Accelerating a particle-in-cell simulation using a hybrid counting sort, *J. Comput.Phys*, Vol.173, 393—411 (2001).
11. Umeda, T., Oya, S.: Performance comparison of parallel sorting with OpenMP, *Proc. 3rd Int. Symp. Comput. Network.*, 334—340 (2015).