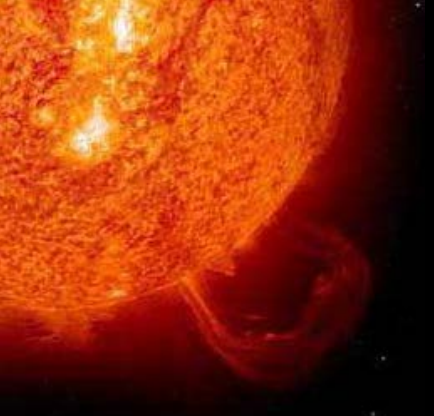
Vlasovコードの
Xeon Phi KNLでの性能評価



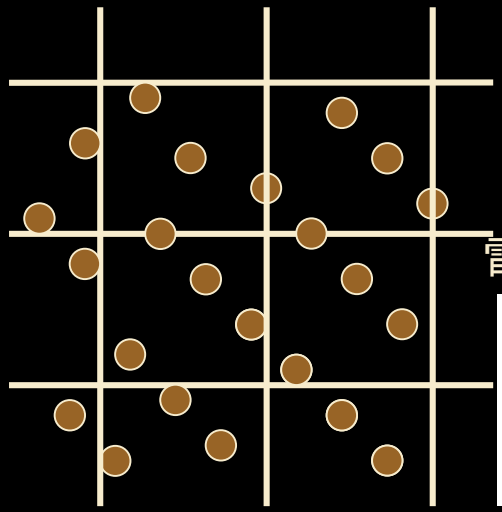
梅田　隆行　名古屋大学宇宙地球環境研究所
深沢　圭一郎　京都大学学術情報メディアセンター

運動論方程式：流体方程式の元となる第一原理
粒子間の衝突が無視できるプラズマ（電離気体）と電磁場を扱う

- 粒子(Particle-In-Cell)コード

$$\frac{d\boldsymbol{r_n}}{dt} = \boldsymbol{v_n}$$
$$\frac{d\boldsymbol{v_n}}{dt} = \frac{q_n}{m_n}(\boldsymbol{E} + \boldsymbol{v_n} \times \boldsymbol{B})$$

r = (x,y,z) と v = (vx,vy,vz) のN体
x(n), y(n), z(n), vx(n), vy(n), vz(n)
と電磁場 E(i,j,k), B(i,j,k) を扱う



電磁場：Maxwell方程式

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

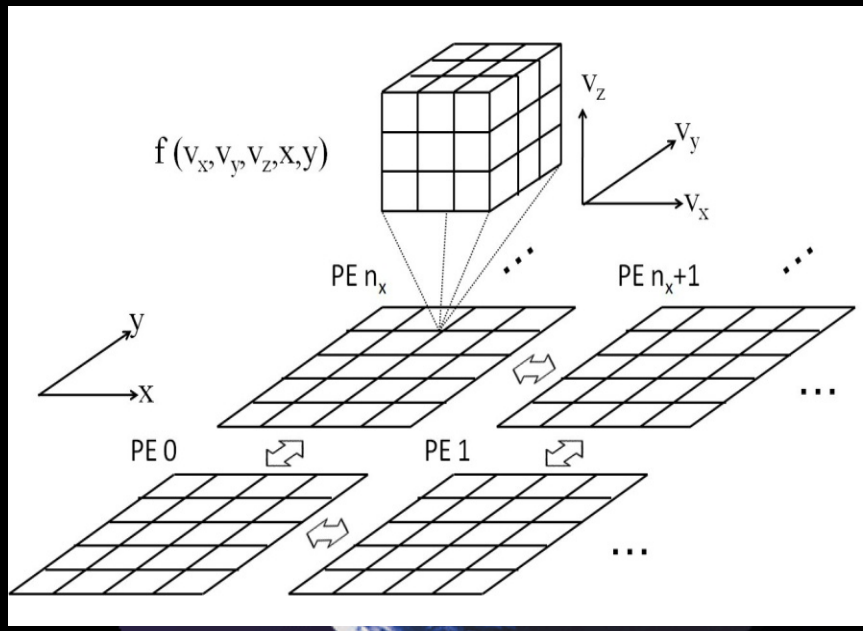
電磁場格子の中を、粒子が動き回る
ノード間並列化（ロードバランス）に課題

- 分布関数(ブラソフ)コード

$$\frac{\partial f}{\partial t} + \boldsymbol{v}\frac{\partial f}{\partial \boldsymbol{x}} + \frac{q}{m}\left(\boldsymbol{E} + \boldsymbol{v} \times \boldsymbol{B}\right)\frac{\partial f}{\partial \boldsymbol{v}} = 0$$

オイラー（流体）型

分布関数 f(i,j,k,l,m,n)と
電磁場 E(i,j,k), B(i,j,k) を扱う



電磁場格子上に分布関数が定義されている
高いスケーラビリティ

Vlasovコードの概要

40^5 ~ 4GB

40^6 ~ 160GB

解いている方程式:

- Maxwell方程式(電磁場の変化と伝搬)

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

負荷は0.1%未満

- Vlasov(無衝突Boltzmann)方程式(プラズマの運動)

$$\frac{\partial f_s}{\partial t} + \boldsymbol{v}\cdot\frac{\partial f_s}{\partial \boldsymbol{x}} + \frac{q_s}{m_s}\left(\boldsymbol{E} + \boldsymbol{v}\times\boldsymbol{B}\right)\cdot\frac{\partial f_s}{\partial \boldsymbol{v}} = 0$$

f(x, y, ~~z~~, vx, vy, vz)

6D! ⇒ 5D

3つに分解

$$\frac{\partial f_s}{\partial t} + \boldsymbol{v}\cdot\frac{\partial f_s}{\partial \boldsymbol{x}} = 0$$

(位置更新)

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

(速度更新:移流)

$$\frac{\partial f_s}{\partial t} + \frac{q_s}{m_s}\left(\boldsymbol{v}\times\boldsymbol{B}\right)\cdot\frac{\partial f_s}{\partial \boldsymbol{v}} = 0$$

(速度更新:回転)

コードの特徴

- 「京」6144ノードの経験より・・・
  ファイル数を減らしたい = プロセス数を減らしたい
- FXシリーズではflat MPIが必ずしも最速ではない

⇒ハイブリッド並列

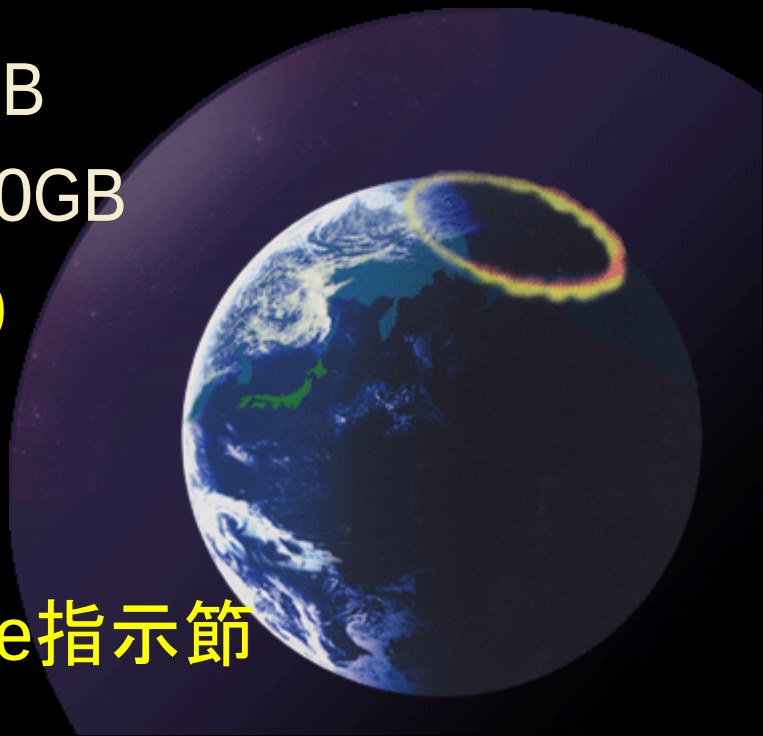
- 次元数が多い　　40^5 ~ 4GB

⇒利用メモリ量が多い　　40^6 ~ 160GB

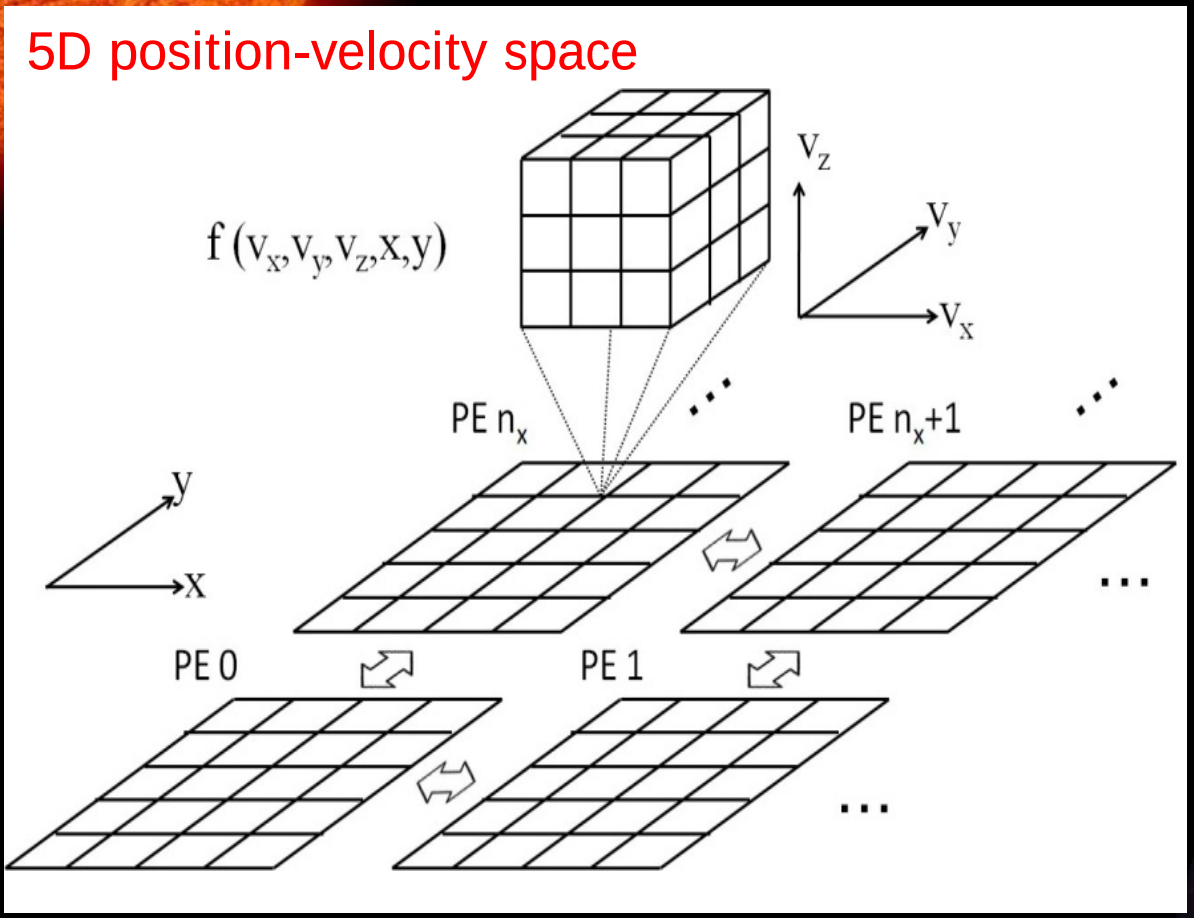
⇒各次元のループ長が短い：20-40

- メニーコア環境
  スレッド数 > ループ長

⇒多重ループのスレッド化：Collapse指示節



ハイブリッド並列



- 3D速度空間が2D/3D実空間上に定義されている ⇒ 計5D/6D
- 実空間ドメインをMPIでプロセス並列化
- サブドメインをOpenMP/自動並列でスレッド並列化

⇒将来的には速度空間もMPI化する予定

Distribution functions: f(l,m,n,i,j)
EM fields: E(i,j), B(i,j)

速度空間が内側

5次精度⇒通信の袖領域/のりしろは3点（MPI_Sendrecv）

コードの構造1

“Velocity” subroutine

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

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

```
!$OMP DO COLLAPSE(2)
DO j=1,Ny
DO i=1,Nx
DO n=1,Nvz
DO m=1,Nvy
DO l=1,Nvx
  xflux(l,m,n) = …
  yflux(l,m,n) = …
  zflux(l,m,n) = …
```

内部配列の依存性により、実空間グリッドのみOpenMP化

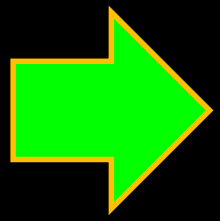
コードの構造2

"Position" subroutine

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

f(l,m,n,i,j) ⇒ ftmp(i,j,l,m,n)
実空間と速度空間の転置

```
!$OMP DO COLLAPSE(2)
DO j=1,Ny ! y
DO i=1,Nx ! x
DO n=1,Nvz ! vz
DO m=1,Nvy ! vy
DO l=1,Nvx ! vx
  xflux(l,m,n,i,j) = …
  yflux(l,m,n,i,j) = …
```

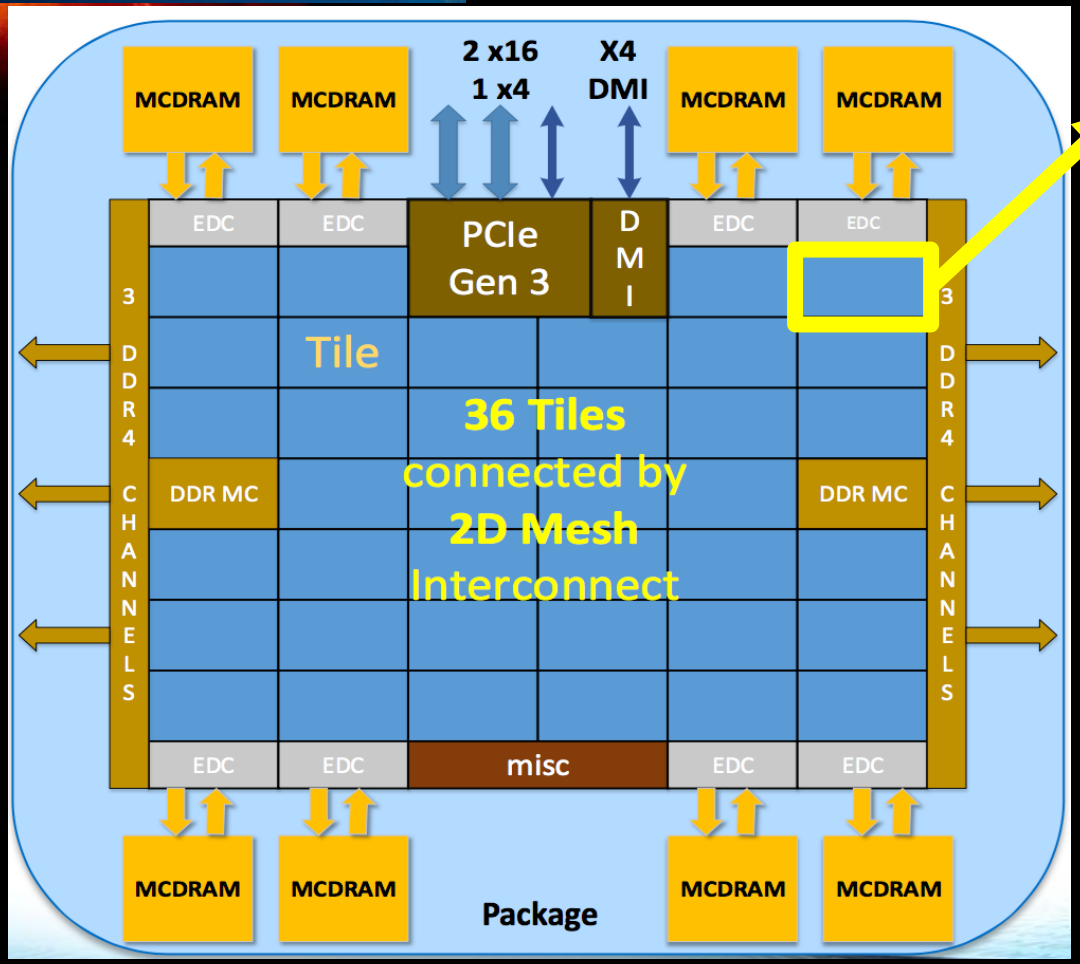


```
!$OMP DO COLLAPSE(3)
DO n=1,Nvz ! vz
DO m=1,Nvy ! vy
DO l=1,Nvx ! vx
DO j=1,Ny ! y
DO i=1,Nx ! x
  xflux(i,j) = …
  yflux(i,j) = …
```

数値流束の配列が大きくなる

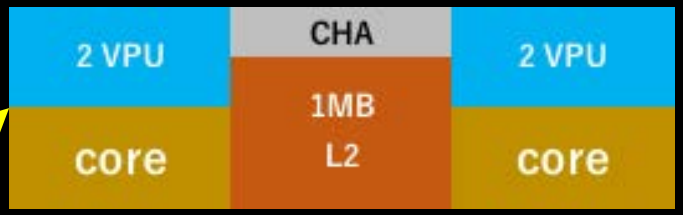
計算量は増えるが、転置をしたほうが速く、メモリ量も減る

Xeon Phi Knights Landing



Intel 資料より

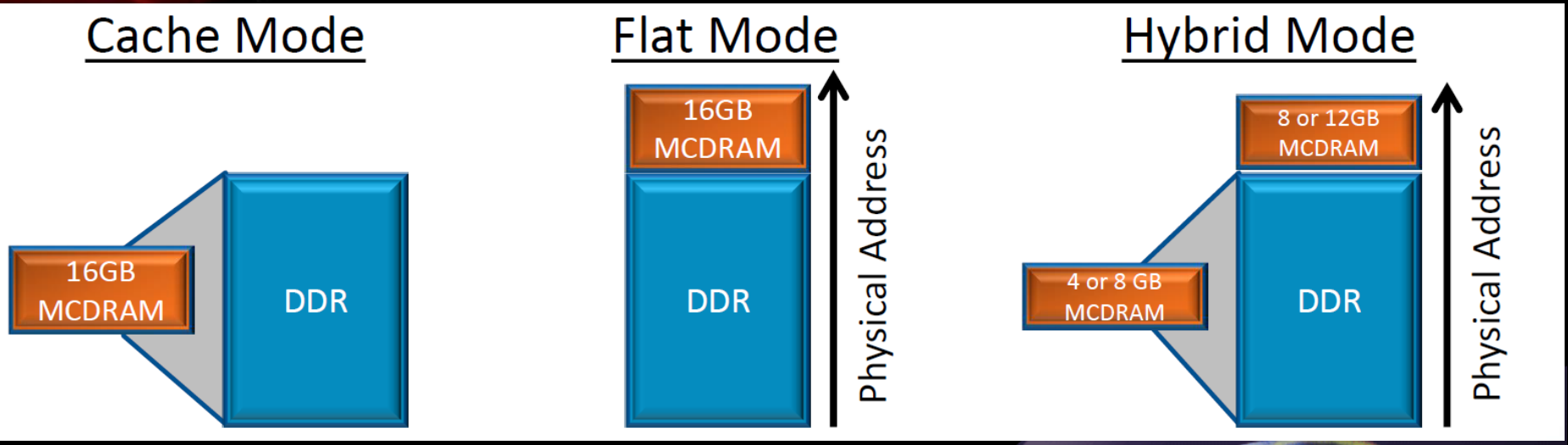
Tile



- VPU(vector processing unit)
  512bit register (AVX512)
  single ~6TFLOPS
  double ~3TFLOPS
- CHA(Caching/Home Agent)
- MCDRAM (Multi-Channel DRAM) >400GB/s
  メモリサイズ2GB×8個で計16GBが搭載
- DDR4 >96GB/s
  メモリサイズ64GB×6個で最大384GBまで搭載可

KNL Memory modes

Intel 資料より



・メインメモリ(DDR3)とL2の間の(L3)キャッシュとして使用

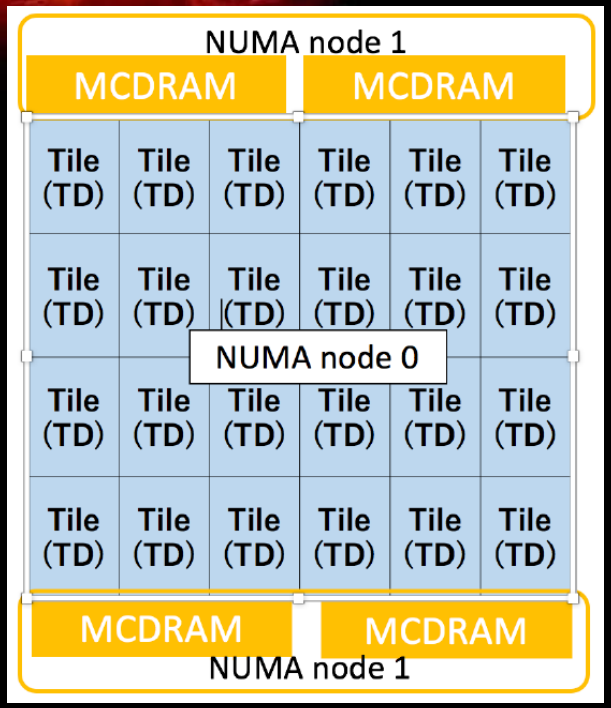
・DDR4と同じアドレスを持つメインメモリとして使用

・CacheとFlatの組合せ
本研究では使用せず

Intel 資料より

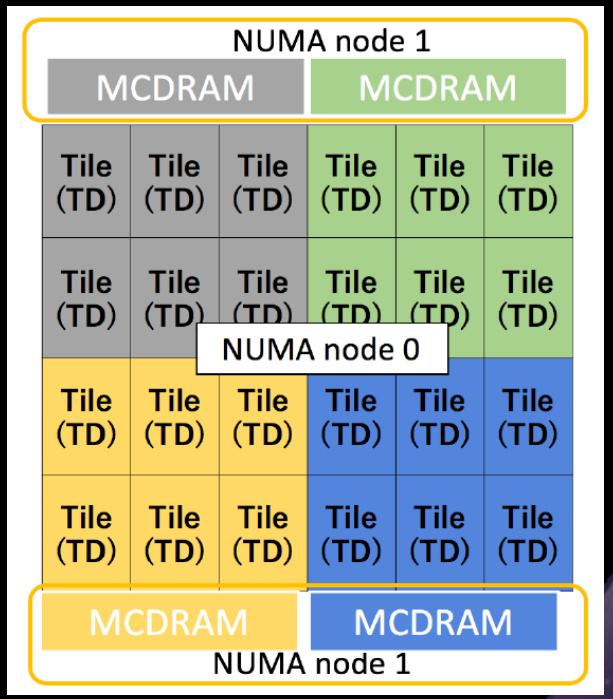
KNL Cluster modes

All2All (All to All)



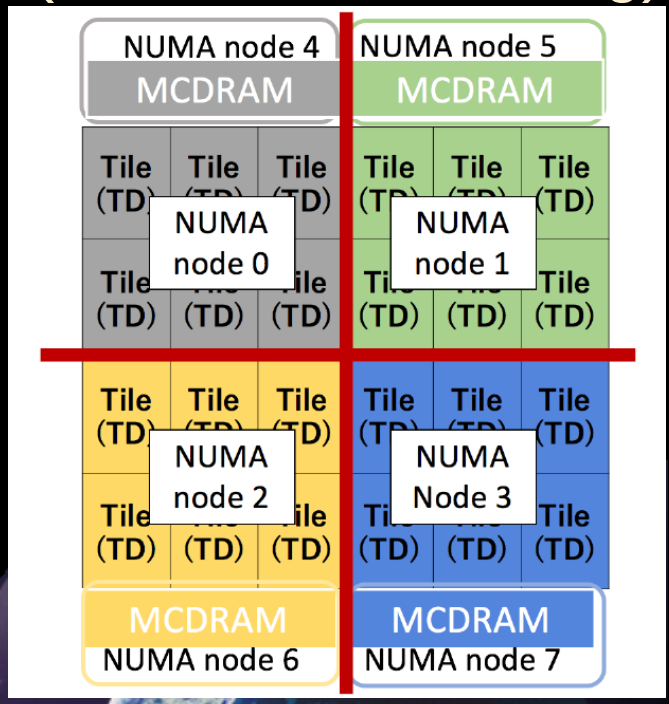
タイルとMCDRAMが一様に分布

Quadrant



タイルとMCDRAMを仮想的に4グループに分割
2分割(Hemisphere)もあり

SNC-4 (sub NUMA clustering)

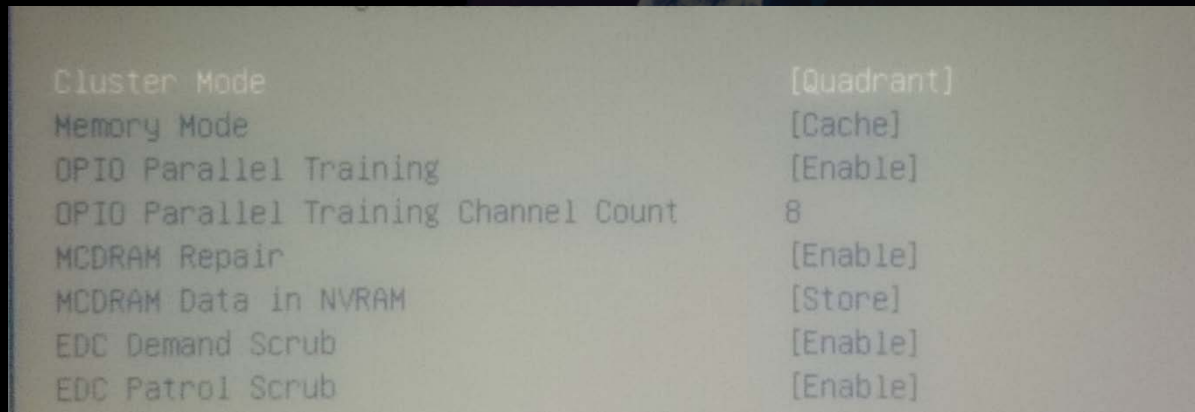


疑似的に4ノードに分割
2分割(SNC-2)もあり
本研究では使用せず

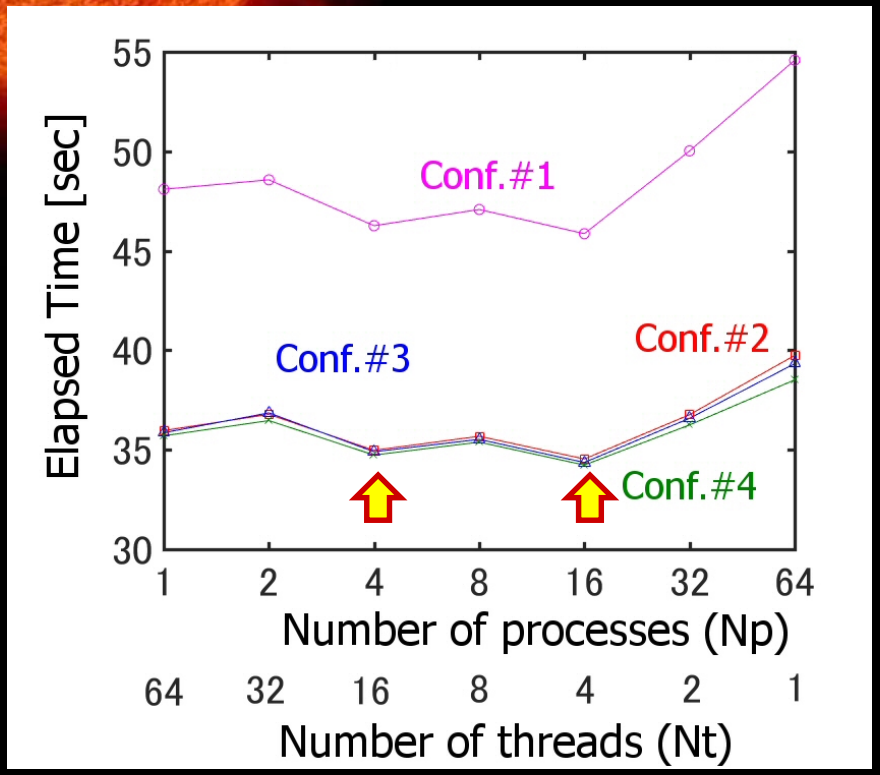
測定環境

- Xeon Phi 7250 (68コア, 16GB MCDRAM)
- 96GB DDR4
- Intel Compiler Ver.17.0.1
  - Option: -ipo -ip -O3 -xMIC-AVX512
- 格子数:40^3*128*64*2 (~28GB > MCDRAM), 5時間ステップの経過時間を計測
- Memory mode: Flat と Cache で比較
- Cluster mode: All2All, Hemisphere, Quadrant で比較

⇒Biosで設定変更



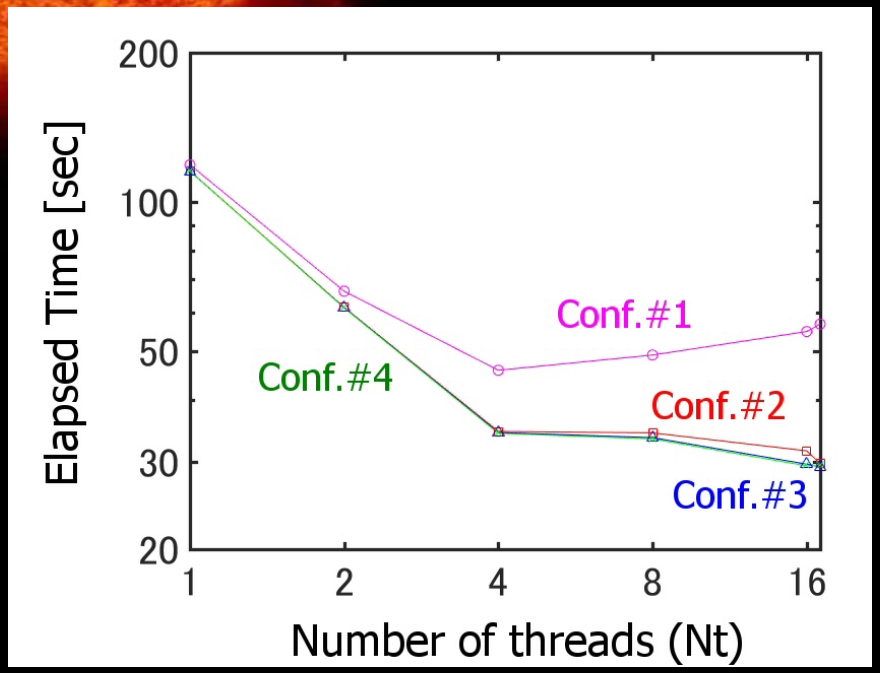
測定結果1：ハイブリッド並列



- #1 Flat-All2All
- #2 Cache-All2All
- #3 Cache-Hemisphere
- #4 Cache-Quadrant

- 64コア使用し、プロセス数Npを変化
  （スレッド数Nt=64/Np）
  4コアは遊んでいる
- Flat-MPI (Np=64)が最も遅い
- CacheモードはFlatモードに比べて1.5倍ほど高速
- Cluster modeに性能差はあまりない

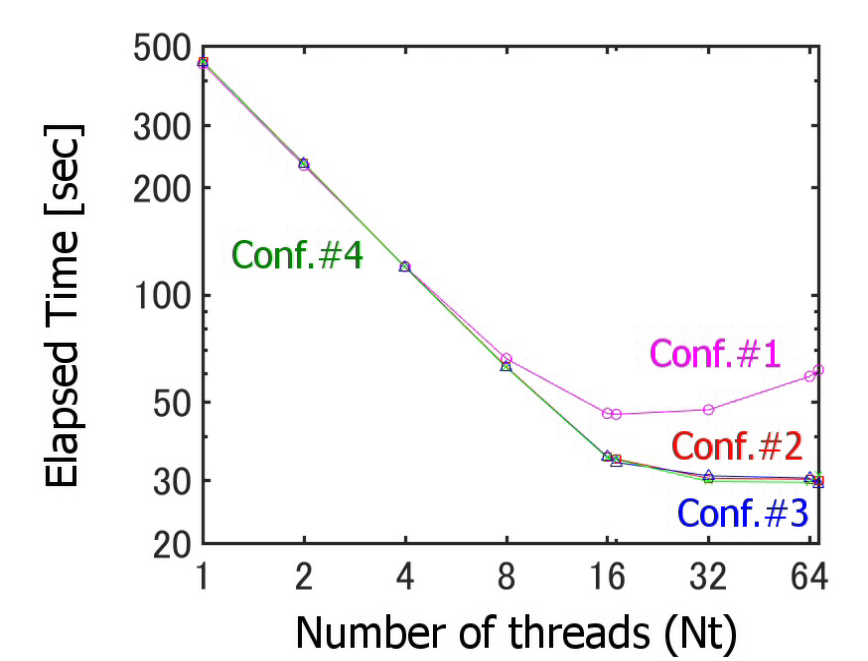
測定結果2：強スケーリング (Np=16)



- プロセス数Npを固定し、スレッド数Ntを変化
- 4スレッドまでは性能がほぼスケール
- 8, 16, 17スレッドでHyper Threads(HT)を利用
- Flatモードでは、HTで性能が劣化
- Cacheモードでは、HTで性能が若干向上
- #3と#4がほぼ同速

- #1 Flat-All2All
- #2 Cache-All2All
- #3 Cache-Hemisphere
- #4 Cache-Quadrant

測定結果3:強スケーリング (Np=4)



- プロセス数Npを固定し、スレッド数Ntを変化
- 17スレッドまでは性能がほぼスケール
- 32, 34, 64, 68スレッドでHyper Threads(HT)を利用
- Flatモードでは、HTで性能が劣化
- Cacheモードでは、HTで性能が若干向上
- #3が最速

- #1 Flat-All2All
- #2 Cache-All2All
- #3 Cache-Hemisphere
- #4 Cache-Quadrant

まとめ

- Xeon Phi Knights Landing において、Vlasovコードの性能測定を行った
- MPI-OpenMPハイブリッド並列が有効
  - 古いコンパイラバージョンでは、Flat-MPIが最速だった…
- MCDRAMはキャッシュモードで使用したほうが計算効率が高い
- クラスターモードはHemisphereがQuadrant/Flatよりも若干計算効率が高い
- HTなしで、Xeon Broadwellと同等の実効性能
  - 実効効率ではXeon Broadwellの約2/5
  - HTで性能が若干向上
- 課題：実効効率の向上：OpenMP 4.0 SIMDを試す??