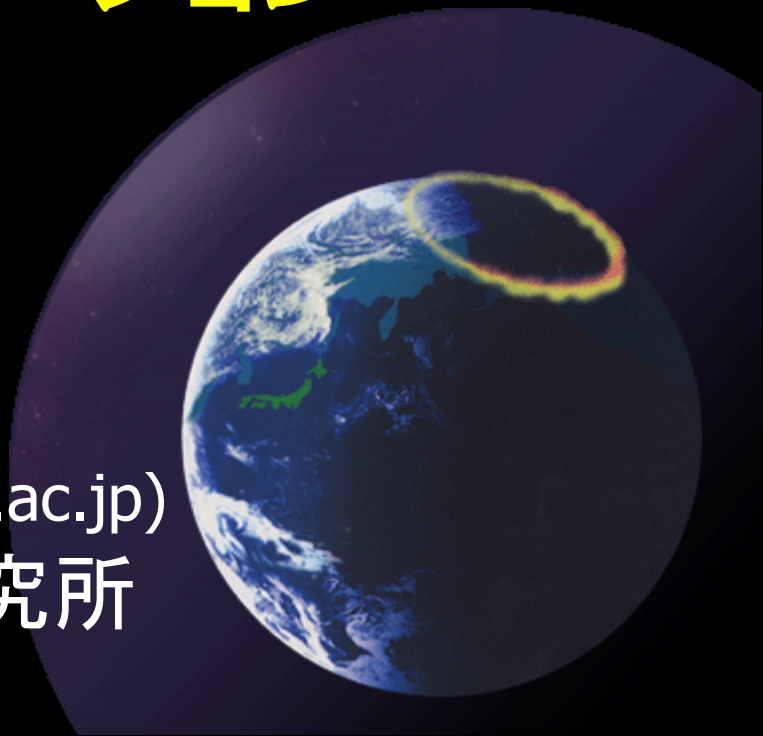
宇宙プラズマの第一原理ブラソフシミュレーション



梅田 隆行 (umeda@isee.nagoya-u.ac.jp)
名古屋大学 宇宙地球環境研究所

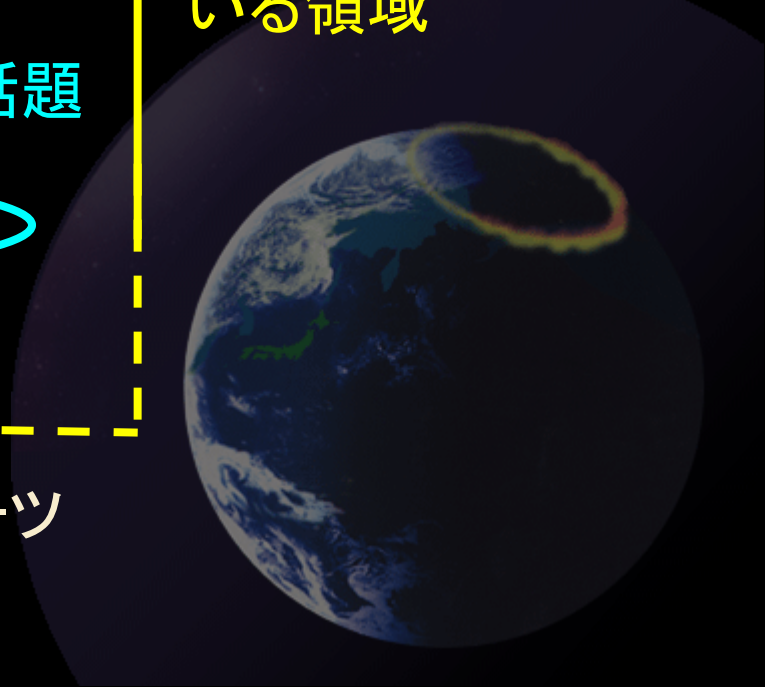
HPCに関わる様々な領域

学者(scientist) and/or 技術者(engineer)

- 工学、理学、気象学、化学、薬学、etc
  - アプリケーションユーザ
- 計算科学(ソフトウェア)
  - 計算スキーム(アルゴリズム)
  - プログラム開発
  - 並列化
  - チューニング
- 計算機科学(ハードウェア)
  - スーパーコンピュータシステム
  - CPU, メモリ, ネットワーク等のパーツ

本日の主な話題

私が関わっている領域



計算機利用歴

2000 2004 2006 2012 2016

NEC SX-5
RASC Kyoto Univ.

Fujitsu HPC2500
(SPARC64V)
RISH Kyoto Univ.

Fujitsu FX1
(SPARC64VII)
Nagoya Univ.

Fujitsu 京
(SPARC64VIII)
RIKEN

Fujitsu FX100
(SPARC64XI)
Nagoya Univ.

IBM Data Star
(POWER4)
UCSD

Fujitsu FX600
(AMD Shanghai)
Nagoya Univ.

Fujitsu FX10
(SPARC64IX)
U. Tokyo/Kyushu U.

ベクトル機は
学生時代だけ

SGI Columbia
(Intel Itanium 2)
NASA

Fujitsu CX400
(Intel SandyBridge)
Kyushu Univ.

Fujitsu CX400
(Intel Haswell)
Nagoya Univ.

Hitachi HA8000
(AMD Barcelona)
Univ. Tokyo (T2K)

Intel Pentium III Xeon
Clovertown (Core)
Nehalem
SandyBridge
Haswell
P4 Northwood/Prescott (NetBurst)
Westmere
IvyBridge
Sun UltraSPARC II
AMD Magny-Cours
Bulldozer

ソフトウェア技術レベルの壁

- 既存のプログラムを動かす（ユーザ）

壁

- プログラムが書ける

  （計算スキーム（アルゴリズム）を開発したことがある）
- プログラムを並列化したことがある
- プロファイラでプログラムの性能測定をしたことがある
- プログラムをチューニングして速くしたことがある

私はこの辺

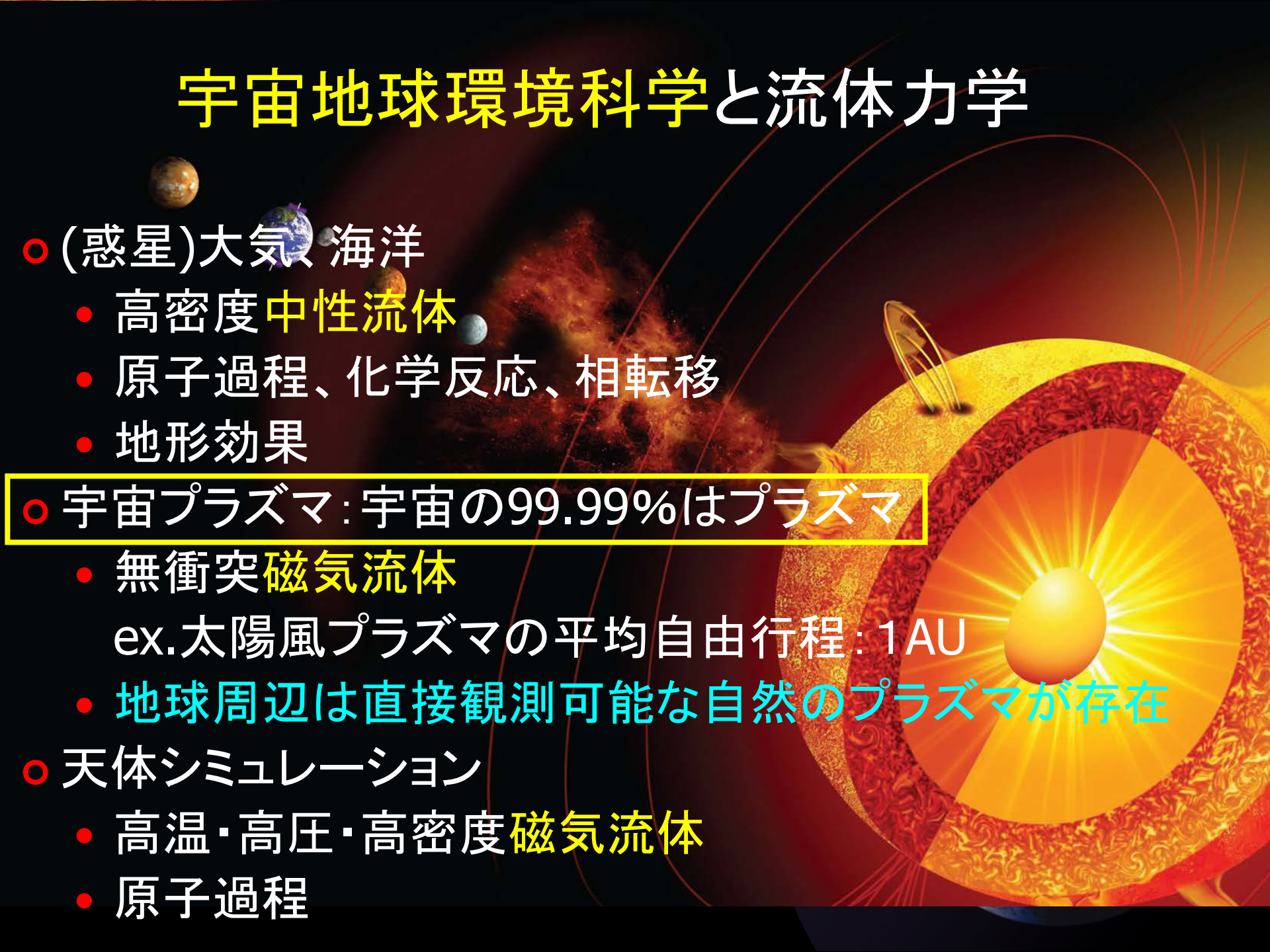
壁

- アセンブラを見たことがある
- アセンブラを書いたことがある

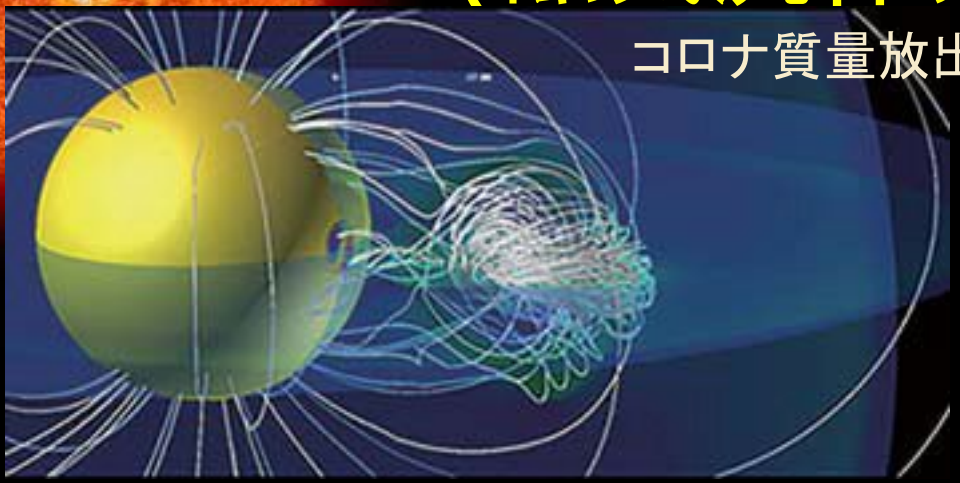
宇宙地球環境科学と流体力学

- (惑星)大気、海洋
  - 高密度中性流体
  - 原子過程、化学反応、相転移
  - 地形効果
- 宇宙プラズマ：宇宙の99.99%はプラズマ
  - 無衝突磁気流体

    ex.太陽風プラズマの平均自由行程：1AU
  - 地球周辺は直接観測可能な自然のプラズマが存在
- 天体シミュレーション
  - 高温・高圧・高密度磁気流体
  - 原子過程



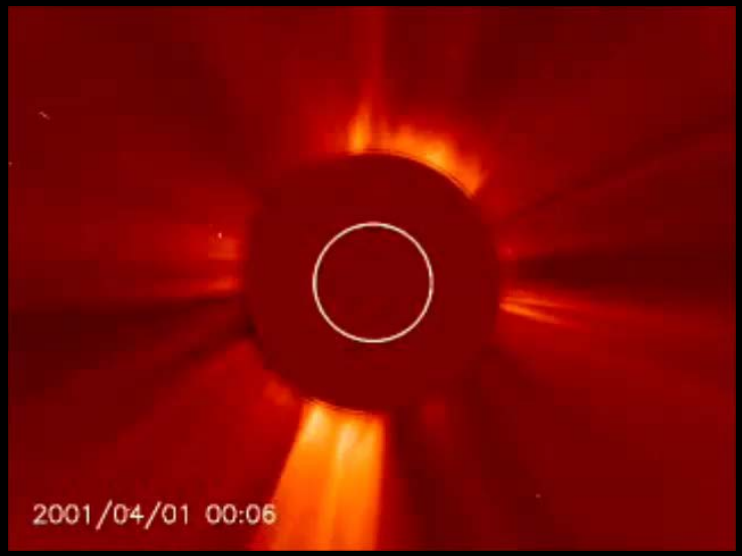
MHD(磁気流体力学)シミュレーション



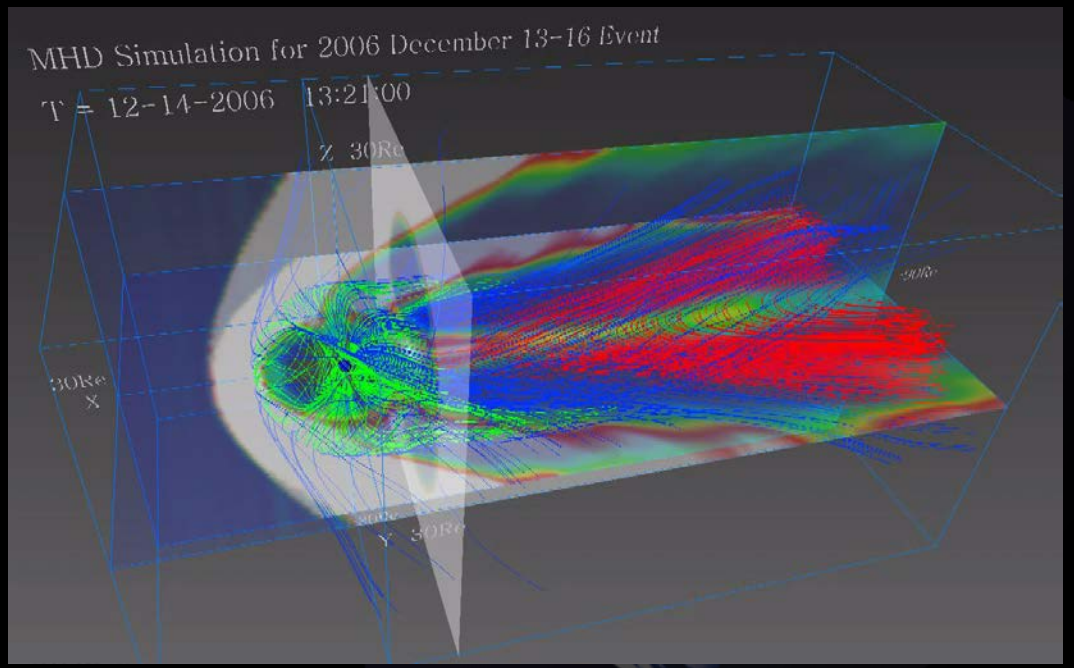
コロナ質量放出　塩田大幸氏(名大)提供

地球半径:　6,371 km
太陽半径:695,700 km
⇒同じMHDシミュレーションでも、
計算の解像度が100倍違う

SOHO衛星の紫外線映像 ©NASA

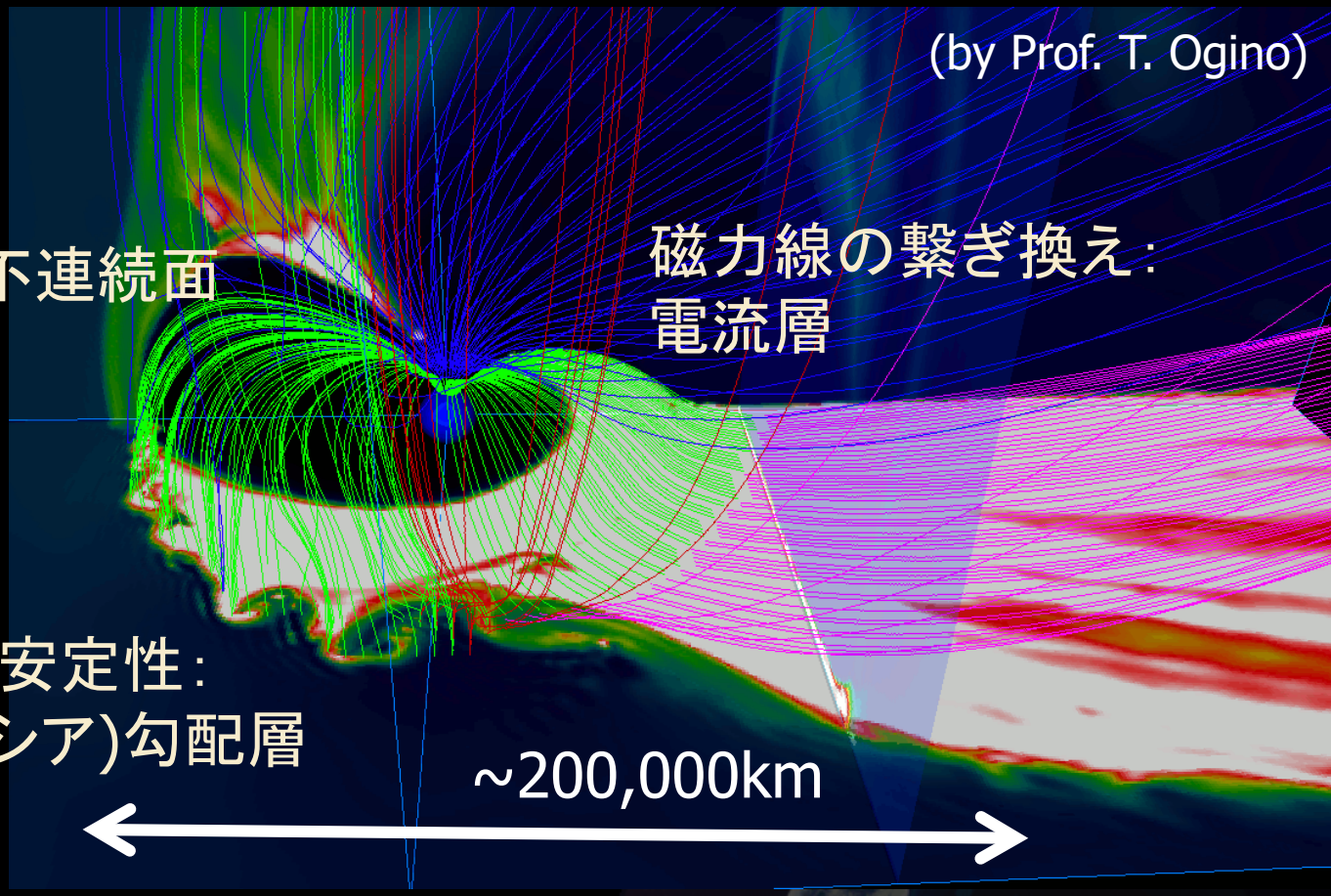


定常的な吹き出し(太陽風)と
突発的な放出(コロナ質量放出)



地球磁気圏 荻野竜樹氏(名大)提供

地球磁気圏の3D MHDシミュレーション



900 x 600 x 600 格子点(~80GB)　　Δ=0.1Re ~640km

- MHD（磁気流体力学）で記述できるグローバルな磁気圏構造の中に、様々なメソスケールの境界層が存在

流体力学と宇宙プラズマ

- 流体力学と同じく、大規模計算（計算領域を広く or 空間解像度を高く）が新しい宇宙プラズマ科学を拓く
  - 計算機性能が上がれば、地球磁気圏シミュレーションの空間解像度で太陽からシームレスに解く事も可能に(?)
  - 1AU~210太陽半径 ⇒ 230,000^3格子点 ~ 3EB
- 空間解像度を高くしていくと、MHD近似が破れる空間スケールに到達
  - 流体力学では平均自由行程が最小特性長
  - 宇宙プラズマでは、慣性長、ジャイロ半径、デバイ長など

⇒近似のない第一原理方程式系の重要性

- 衛星観測（軌道上の1点観測）では、グローバルなMHDスケールは見えない

流体方程式vs.第一原理

Vlasov方程式 = 無衝突Boltzmann+電磁力

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

同じ物理を扱う

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

Euler系(格子系)
空間上の速度分布関数で考える

Lagrange系(粒子系)
粒子の軌道上で考える

流体方程式：Boltzmann方程式から導出される保存則

質量保存

$$0=\frac{\partial N}{\partial t}+\frac{\partial}{\partial \boldsymbol{x}}N\boldsymbol{U}$$

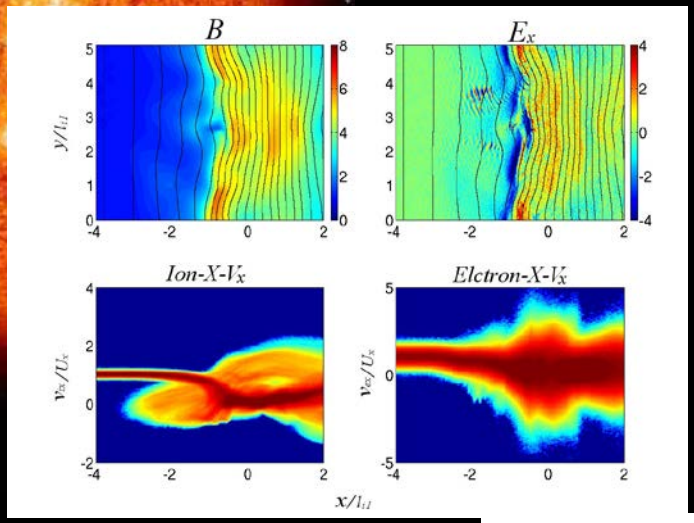
運動量保存

$$0=\frac{\partial}{\partial t}mN\boldsymbol{U}+\frac{\partial}{\partial \boldsymbol{x}}\left(mN\boldsymbol{U}\boldsymbol{U}+P\right)-\rho\boldsymbol{E}-\boldsymbol{J}\times\boldsymbol{B}-mN\boldsymbol{g}$$

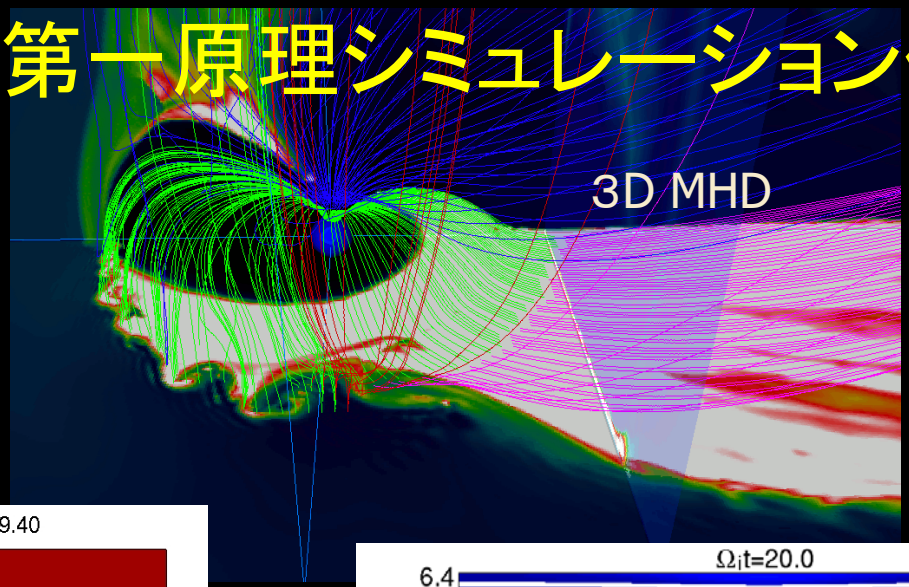
エネルギー保存

$$0=\frac{\partial}{\partial t}\frac{mN\left|\boldsymbol{U}\right|^2+3P}{2}+\frac{\partial}{\partial \boldsymbol{x}}\frac{mN\left|\boldsymbol{U}\right|^2+5P}{2}\boldsymbol{U}-\boldsymbol{E}\cdot\boldsymbol{J}-mN\boldsymbol{g}\cdot\boldsymbol{U}$$

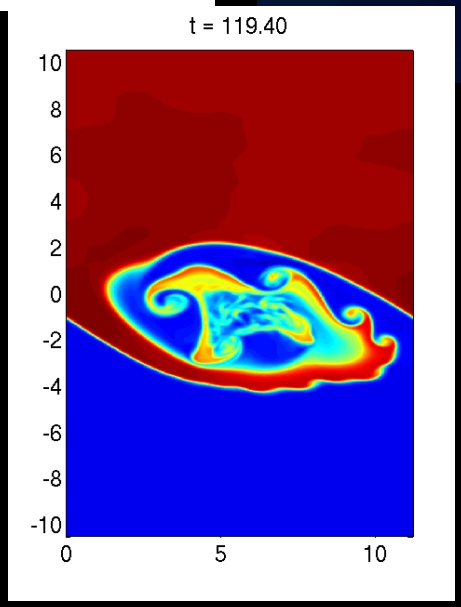
第一原理シミュレーション例



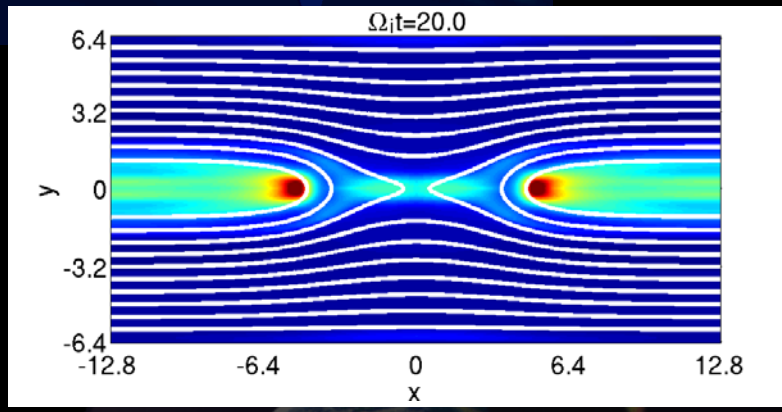
衝撃波
粒子(PIC)



磁力線
繋ぎ変え



ブラソフ
KH渦



ブラソフ

イオン慣性
イオンジャイロ

プラズマ振動

>数百万km　　数万～数千km　　数百km～数十m

マクロ:
グローバル磁気圏

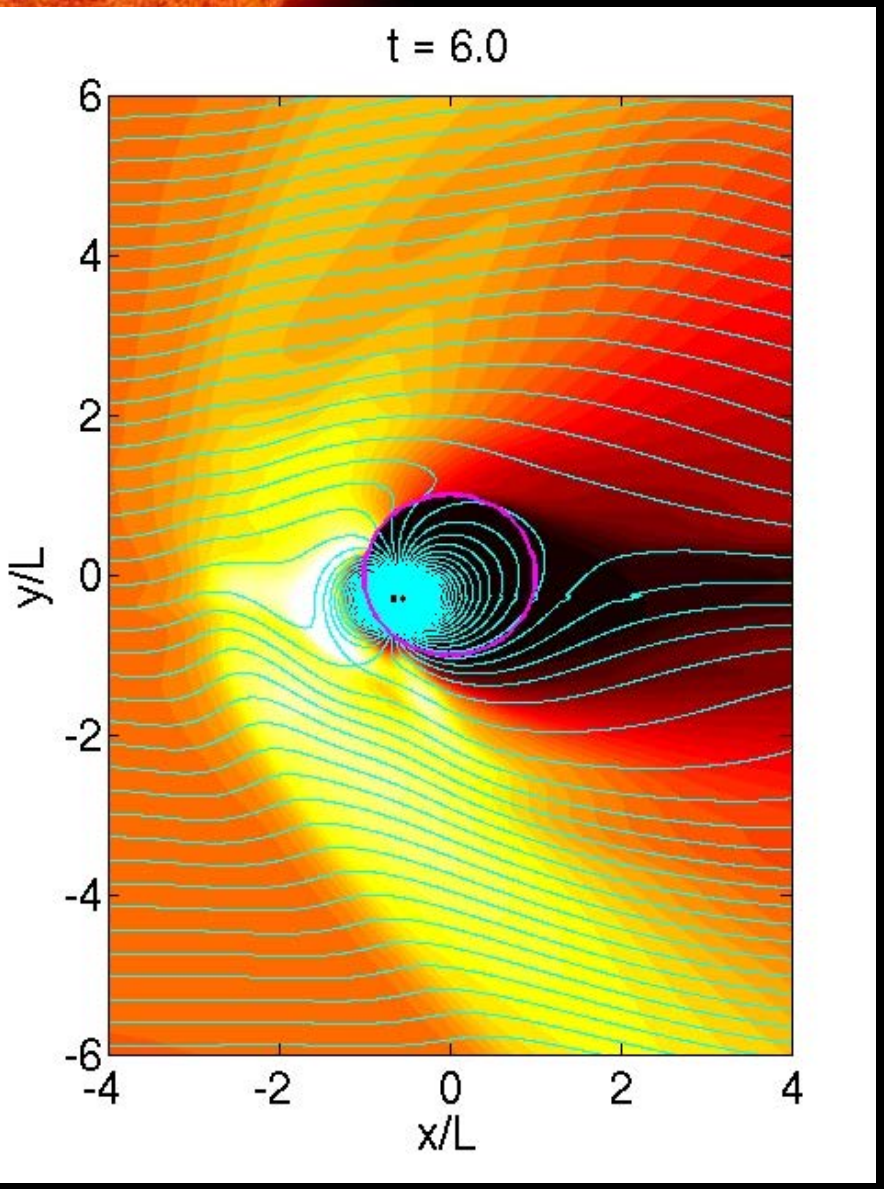
メソ:ローカル境界層
MHD

ミクロ:粒子性
運動論

"ミニ"磁気圏のブラソフシミュレーション

空間2次元・速度3次元(2.5D)

	この計算	月
天体半径	2.72km	1738km
イオンジャイロ半径	1.36km	85km
質量比: m_i/m_e	25	1836
太陽風速度	400km/s	



Umeda & Fukazawa EPS 2015

i+ ⊙ Bz

磁力線 とプラズマ密度

(1920*2560*60^3 ~ 50TB)

6144 nodes @ K-computer

フル6次元計算には

あと2500倍必要！

ブラソフコード開発の構想

- 高解像度のMHD計算には“物理”の限界がある（MHD近似が破たんする）

⇒第一原理計算の重要性

- 2つの第一原理計算法

✓粒子法(PIC: Particle-In-Cell) ⇒ 主流

✓ブラソフ法 ⇒ 誰もやってない

- 何故誰もやってない？

⇒大量のメモリが必要 40^6 ~ 160GB

開発当時(2006年前後)の計算機では実行が困難

次元を減らせばなんとかなる？ 6D ⇒ 5D

⇒省メモリ型の新しいスキームを開発 40^5 ~ 4GB

- 並列化はPIC法に比べると超簡単

ブラソフコードのスキーム

- ブラソフコードのスキームの論文は(1960年後半から)2000年代前半で既に100編以上あった
  - ほぼすべての論文は、どの手法でも同じ結果が得られる“標準(ベンチマーク)問題”しか扱われていなかった

  ⇒新しいスキームを開発する必要性はどこに？
  - 標準問題以外の実用計算例が稀

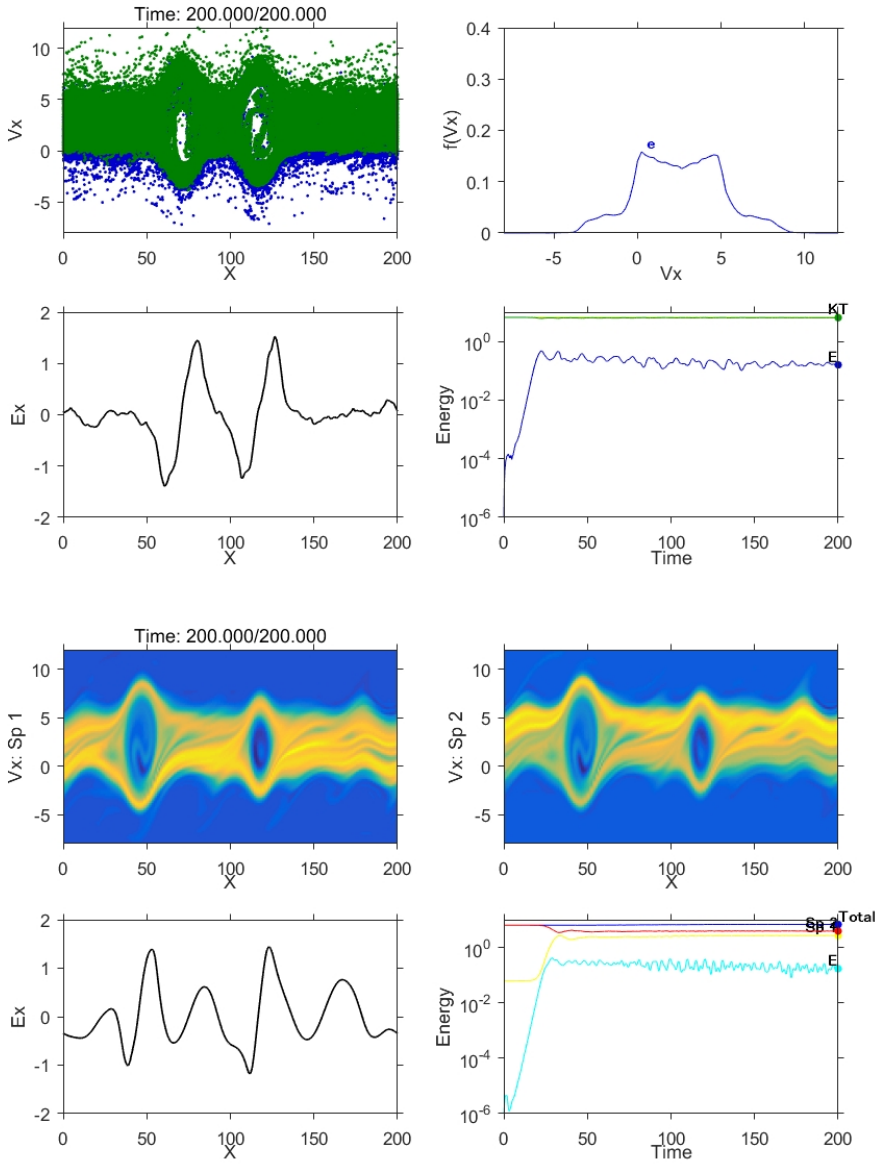
  ⇒問題点は、メモリ量なのか、スキームなのか？
- 2006年頃の主流

  ・スペクトル法、3次スプライン ⇒ 1970年代からの現役

  ・CIP法 ⇒ 2000年代前半では広い分野で使われていた

  Ex. Umeda et al. PoP 2003
- 本日のベンチマーク
  - 時間・空間変化が激しく、かつ長時間の問題に適用

コード比較(粒子vs.ブラソフ)



- 二流体不安定：速度が異なる2つの流体の混合
  - プラズマの基本的問題
  - 中性流体⇒衝突による
  - 宇宙プラズマ⇒衝突がないため、音波（電場）を介して

粒子(PIC)コード
⇒プラズマ業界の主流
Nx=200, Np=100,000*Nx
全ての粒子を表示

ブラソフコード with CIP法
⇒2006年頃の主流の1つ
Nx=200, Nv=100
速度分布関数f(x,v)を表示

新しいスキームの開発

時間・空間変化が激しく、かつ長時間の問題に耐えうる

- プラズマ物理からの制約

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

○ 保存性 ⇒ ×(非保存型)CIP法

○ 正値性　負の質量からのエネルギー供給を防ぐ

○ 無振動性（既存の極値を保ち新たな極値は抑える）

⇒× TVD法、スペクトル法

- メモリ消費からの制約

×多段時間積分法 (e.g., R-K法)

×マルチモーメント(マルチデータ)法 (e.g., CIP法)

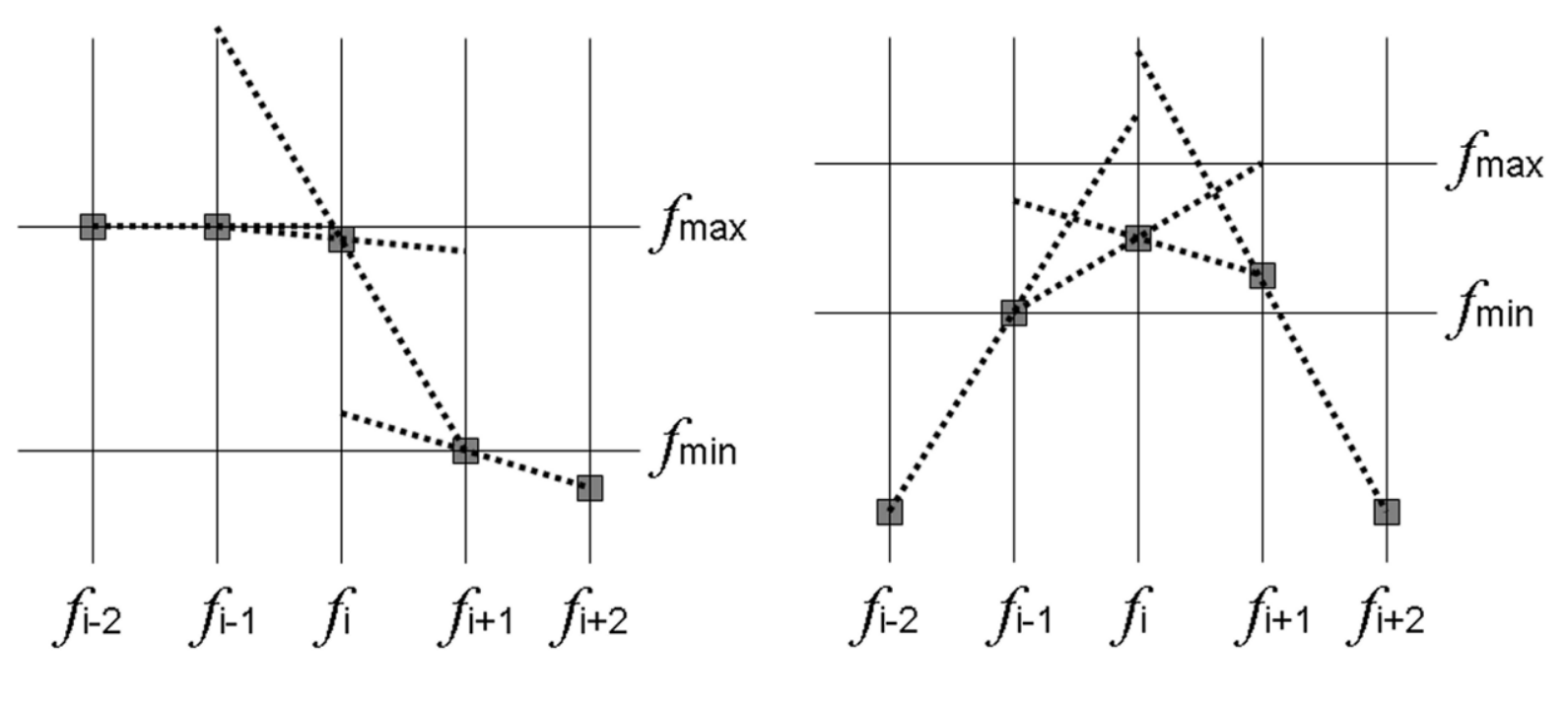
○ 高次空間精度

⇒高次・保存型セミラグランジュ法＋無振動リミッタ

空間精度＝時間精度、中間データ必要なし

無振動スキーム

区間[i-1 i+1]内の極値を検出するには5点必要

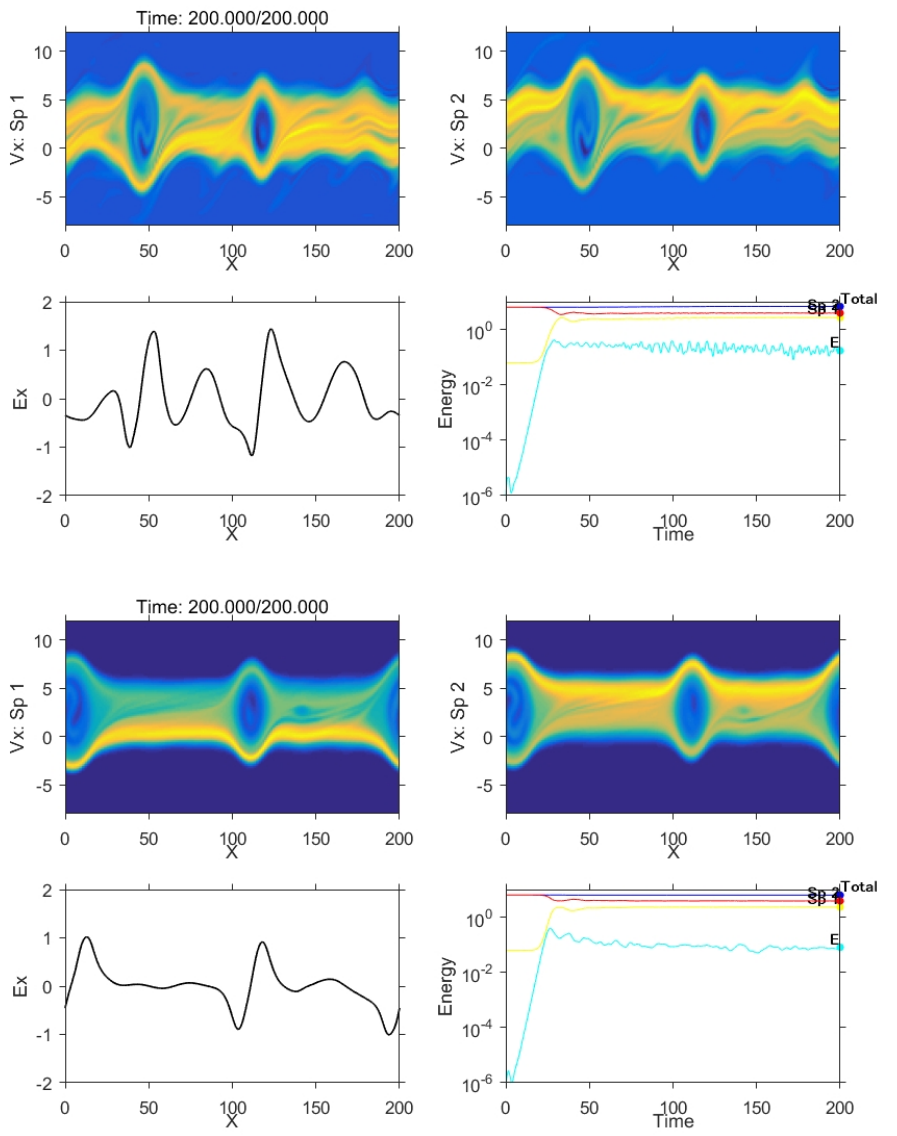


５点の情報から不連続面（極大・極小）を判定し、既存の極値を残す
極小 fmin=0 とすれば、正値性も保証

3次精度 *Umeda EPS 2008*
4次精度 *Umeda et al. CPC 2012*

本日の計算例は5次精度（未出版）

コード比較(CIP vs.保存型無振動)



CIP法⇒2006年頃の主流の1つ
Nx=200, Nv=100

- 物理量が急激に変化するところで空間的数値振動が生じる
- 負の質量からのエネルギー供給で、非物理的な時間振動が生じる

5次精度保存型無振動スキーム
Nx=200, Nv=100

- 数値的な空間振動と非物理的な時間振動の両方を抑制
- 2桁以上の物理量の変化には精度が足りず解が鈍る

並列化・性能測定・チューニング

- ブラソフコードでは、オイラー系でよく用いられる領域分割法で高い並列性能が出る
- 基本的には、“袖”領域のデータ交換（mpi_isend/irecv or mpi_sendrecvによる隣接通信）でよい
- 極たまに、収束法（mpi_allreduceなどの全体通信）が用いられ、プロセス数が増える(>1000)と並列効率の劣化が顕著になる
- Xeon Phiでは、経験的にflat-MPIが最速
- x86, SPARCなどでは、CPU(ダイ)内はスレッド並列を用いたハイブリッド並列のほうが速い

本日は、いくつかのスパコンでの性能測定結果を紹介

“なんちゃって”マルチコア

©Intel

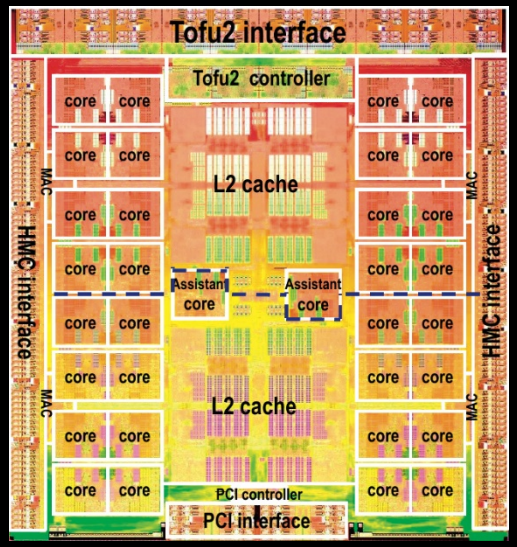
Intel Pentium D

パッケージ内に、CPUダイを2つ搭載


©AMD

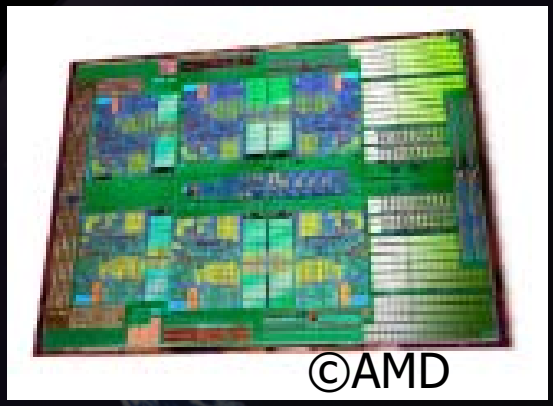
AMD Opteron (Barcelona) versus Intel Xeon (Clovertown) 2006年頃

Intelの2ダイ(なんちゃって)に対して、native quad-coreを主張



©Fujitsu

SPARC64 XI (FX100)も
なんちゃって32コア
⇒経験的に、32スレッド
よりも16スレッドx2プロセス
のほうが高速


©AMD

後に、なんちゃって12コアを作成(笑)
AMD Opteron (Magny-Cours = Istanbul x 2)

日本の国家プロジェクト

Earth Simulator

- NEC SX-6 - 8CPUs: 64GFlops /node
- 16GB memory /node
- 640nodes: クロスバー
- 40.96TFlops peak

K computer

- SPARC64 VIII - 8cores: 128GFlops /node
- 16GB memory /node
- 82,944nodes: Tofu
- 10.62PFlops peak

Exa

- >5TFlops
- 据え置き?
- >200,000 nodes

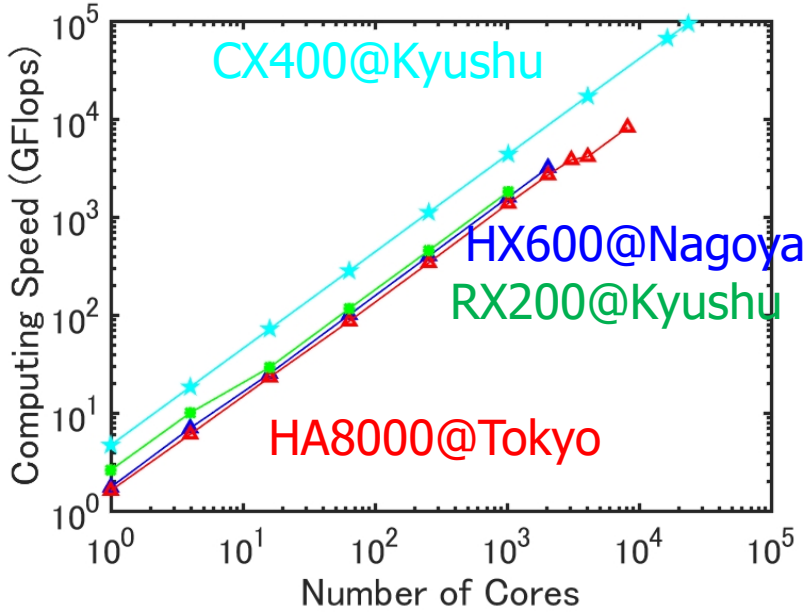
2002/03-2009/03

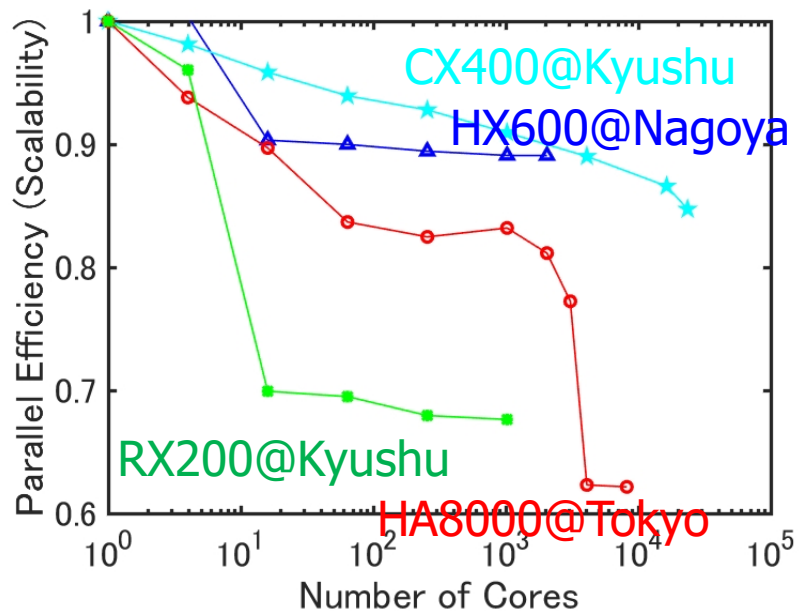
2011/11-

FX100(名大)

- SPARC64 XI: 1126.4GFlops
- 32GB
- 2880 nodes
- 3.2PFlops

Opteron vs. Xeon 性能比較





弱スケーリング測定(1GB/core)

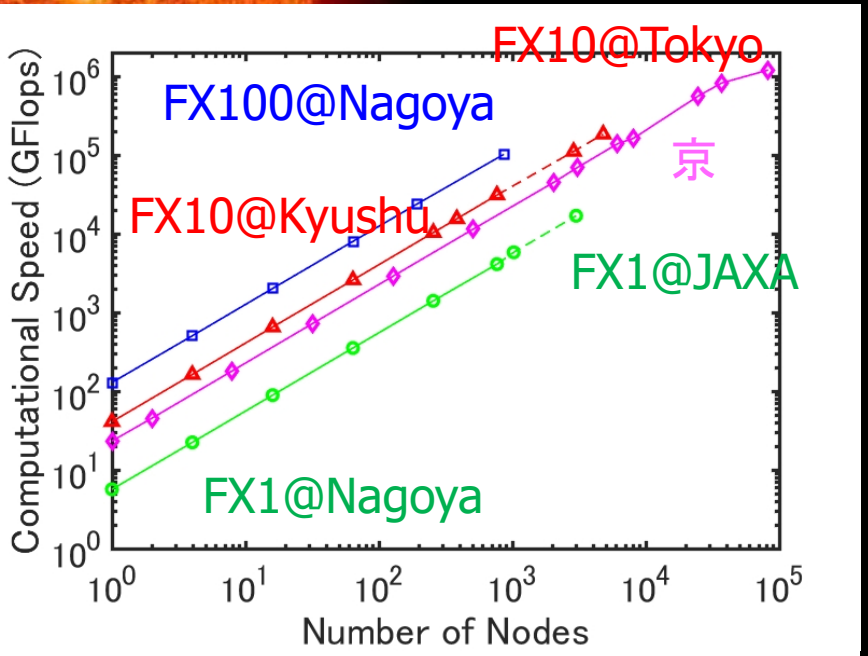
- HA8000: AMD Barcelona (4cores) x 4CPUs x 512nodes (Myrinet)
- HX600: AMD Shanghai (4cores) x 4CPU x 160nodes (DDR Infiniband)
- RX200: Intel Westmere (6cores) x 2CPUs x 384nodes (QDR Infiniband)
- CX400: Intel SandyBridge (8cores) x 2CPUs x 1476nodes (FDR Infiniband)
- HA8000とCX400は全ノードまで計測
- 2011年頃まではOpteronとXeonの実効速度はそれほど変わらなかった

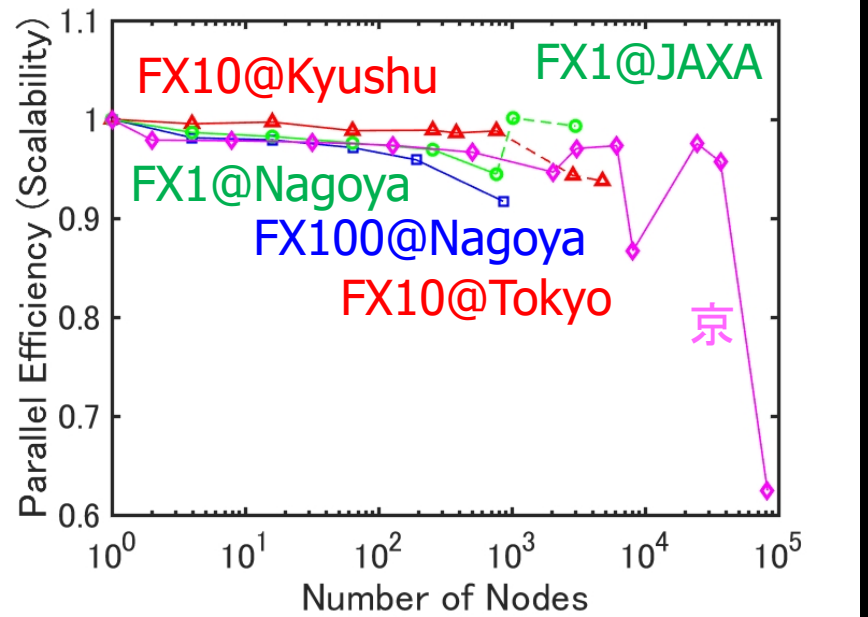
⇒実効効率~15%

- SandyBridgeで性能差が開く

⇒実効効率~20%

FXシリーズ性能比較





弱スケーリング測定(1GB/core)

- FX1: SPARC64VII (4cores) x 768/3008nodes (DDR Infiniband)
- 京: SPARC64VIII (8cores) x 82,944nodes (Tofu)
- FX10: SPARC64IX (16cores) x 768/4800nodes (Tofu)
- FX100: SPARC64XI (16x2cores) x 2880nodes (Tofu2)

- 実効効率:

FX1:京/FX10:FX100=13%:17%:10%

- FX1はJAXA>名大
- 京の通信性能は、Tofuの形状に依存 (2次元は、384*2^nで性能大)
- FX100は全ノード未実行

チューニングの話

- 「こうすれば（経験的に）速くなる（かも）」というチューニング法がいくつか存在

（例）

- ループ分割・ループ融合

⇒一見、全く逆のことを言っている・・・

- コンパイラが最適化できないくらい、ループ内が長い（重い演算が多い）場合は、分割するほうが良い
- ループ内が短い（演算が軽すぎる）場合は、融合したほうが、速くなるかも
- 基本的に trial & error、「必ず速くなる」保証は無し

⇒計算機（CPU）が変わる毎に、性能測定をすべき

- 実効性能(Flops値)ではなく、計算時間が重要！

変更前のas-isコードの例

```
do nn=nvzs-1,nvze+1
  do mm=nvys-1,nvye+1
    do ll=nvxs-1,nvxe+1
      ffi(ll,mm)=1.0d0/ff(ll,mm,nn,ii,jj)

      hm2=ff(ll-lv*2,mm,nn,ii,jj)
      hm1=ff(ll-lv  ,mm,nn,ii,jj)
      hp0=ff(ll     ,mm,nn,ii,jj)
      hp1=ff(ll+lv  ,mm,nn,ii,jj)
      hp2=ff(ll+lv*2,mm,nn,ii,jj)
      t_dfx(ll,mm)=pic5(hp2,hp1,hp0,hm1,hm2,vvx)

      hm2=ff(ll,mm-mv*2,nn,ii,jj)
      hm1=ff(ll,mm-mv  ,nn,ii,jj
      hp0=ff(ll,mm     ,nn,ii,jj)
      hp1=ff(ll,mm+mv  ,nn,ii,jj)
      hp2=ff(ll,mm+mv*2,nn,ii,jj)
      t_dfy(ll,mm)=pic5(hp2,hp1,hp0,hm1,hm2,vvy)

      hm2=ff(ll,mm,nn-nv*2,ii,jj)
      hm1=ff(ll,mm,nn-nv  ,ii,jj)
      hp0=ff(ll,mm,nn     ,ii,jj)
      hp1=ff(ll,mm,nn+nv  ,ii,jj)
      hp2=ff(ll,mm,nn+nv*2,ii,jj)
      t_dfz(ll,mm)=pic5(hp2,hp1,hp0,hm1,hm2,vvz)
    end do
  end do
  do mm=nvys-1,nvye+1
    do ll=nvxs-1,nvxe+1
      a = **; b = **; c = **; d = **;
      dfx(ll+l0,mm   ,nn   ) = dfx(ll+l0,mm   ,nn   ) + a
      dfx(ll+l0,mm+mv,nn   ) = dfx(ll+l0,mm+mv,nn   ) + b
      dfx(ll+l0,mm   ,nn+nv) = dfx(ll+l0,mm   ,nn+nv) + c
      dfx(ll+l0,mm+mv,nn+nv) = dfx(ll+l0,mm+mv,nn+nv) + d
```

2重ループで分割

外側にさらにii,jjの2重ループが存在

lv, mv, nv, l0, m0, n0, vvx, vvy, vvz
は(ii,jj)に依存
⇒nnループの外で計算

1次元フラックスを計算し、
最後に3次元フラックスを合成

a,b,c,dはffi, t_gfx, t_gfyに依存

同様にdfy, dfzを計算

変更後のコード

```
do nn=nvzs-1,nvze+1
  do mm=nvys-1,nvye+1
    do ll=nvxs-1,nvxe+1
      ffi(ll)=1.0d0/ff(ll,mm,nn,ii,jj)
    end do
    do ll=nvxs-1,nvxe+1
      hm2=ff(ll-lv*2,mm,nn,ii,jj)
      hm1=ff(ll-lv  ,mm,nn,ii,jj)
      hp0=ff(ll     ,mm,nn,ii,jj)
      hp1=ff(ll+lv  ,mm,nn,ii,jj)
      hp2=ff(ll+lv*2,mm,nn,ii,jj)
      t_dfx(ll)=pic5(hp2,hp1,hp0,hm1,hm2,vvx)
    end do
    do ll=nvxs-1,nvxe+1
      hm2=ff(ll,mm-mv*2,nn,ii,jj)
      hm1=ff(ll,mm-mv  ,nn,ii,jj
      hp0=ff(ll,mm     ,nn,ii,jj)
      hp1=ff(ll,mm+mv  ,nn,ii,jj)
      hp2=ff(ll,mm+mv*2,nn,ii,jj)
      t_dfy(ll)=pic5(hp2,hp1,hp0,hm1,hm2,vvy)
    end do
    do ll=nvxs-1,nvxe+1
      hm2=ff(ll,mm,nn-nv*2,ii,jj)
      hm1=ff(ll,mm,nn-nv  ,ii,jj)
      hp0=ff(ll,mm,nn     ,ii,jj)
      hp1=ff(ll,mm,nn+nv  ,ii,jj)
      hp2=ff(ll,mm,nn+nv*2,ii,jj)
      t_dfz(ll)=pic5(hp2,hp1,hp0,hm1,hm2,vvz)
    end do
    do ll=nvxs-1,nvxe+1
      a = **; b = **; c = **; d = **;
      dfx(ll+l0,mm   ,nn   ) = dfx(ll+l0,mm   ,nn   ) + a
      dfx(ll+l0,mm+mv,nn   ) = dfx(ll+l0,mm+mv,nn   ) + b
      dfx(ll+l0,mm   ,nn+nv) = dfx(ll+l0,mm   ,nn+nv) + c
      dfx(ll+l0,mm+mv,nn+nv) = dfx(ll+l0,mm+mv,nn+nv) + d
```

重い演算（割り算とフラックス関数）がループ内に混在
⇒1次元ループ分割

データの受け渡しを1次元配列で

経験的に、2次元ループ分割よりも1次元ループ分割のほうが速い

京/FX10では、約1.2倍の性能向上

性能比較@x86

測定条件：128x64x40x40x40x2格子点 ~ 24GB
サブルーチンの5タイムステップの経過時間を
1ノードで測定

ifort ver.17.0.1.132

- SandyBridge (16スレッド・2プロセス)
  - (前)43.6sec ⇒ (後)27.7sec (1.57倍)
- Haswell (18スレッド・2プロセス)
  - (前)27.7sec ⇒ (後)15.8sec (1.75倍)
- Knights Landing (Xeon Phi: 16スレッド・16プロセス)
  - (前)40.9sec ⇒ (後)30.7sec (1.33倍)

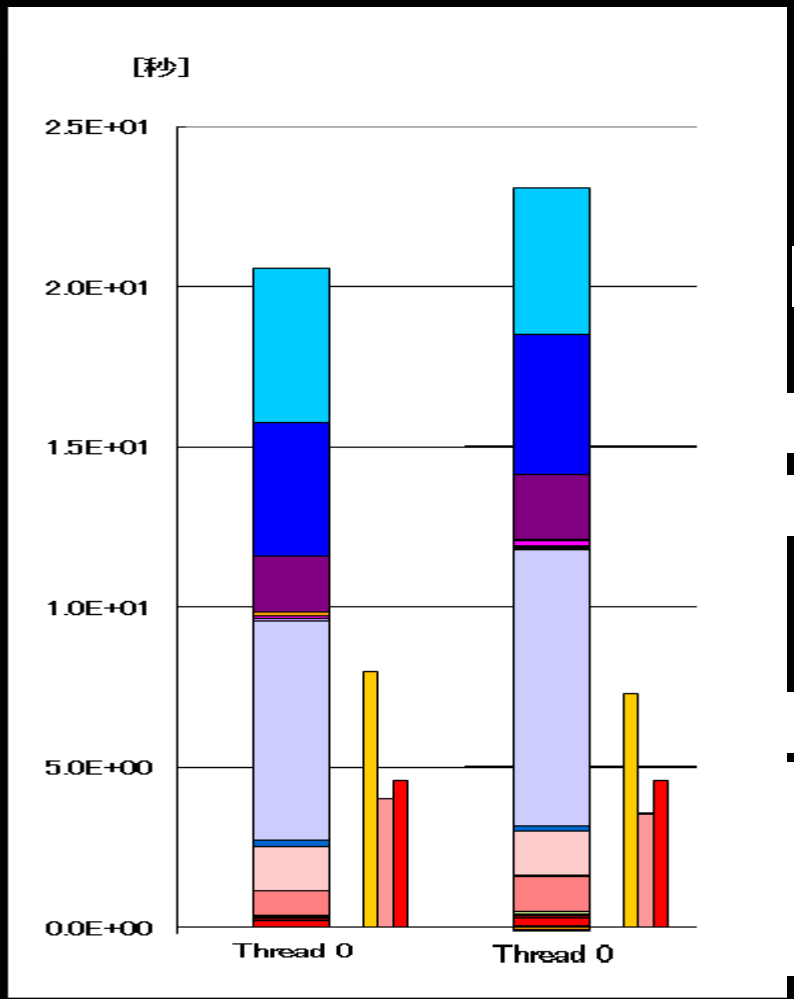
x1.75

x0.9

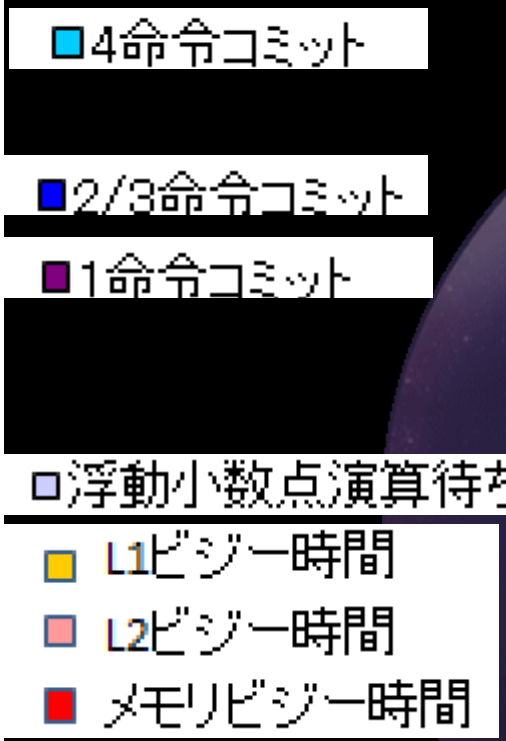
Sandy Bridge : 8 flops/clock = 8 SIMD (single instruction multiple data) AVX
Haswell : 16 flops/clock = 8 SIMD x 2 FMA (fused multiply-add)
Knights Landing : 32 flops/clock = 16 SIMD x 2 FMA

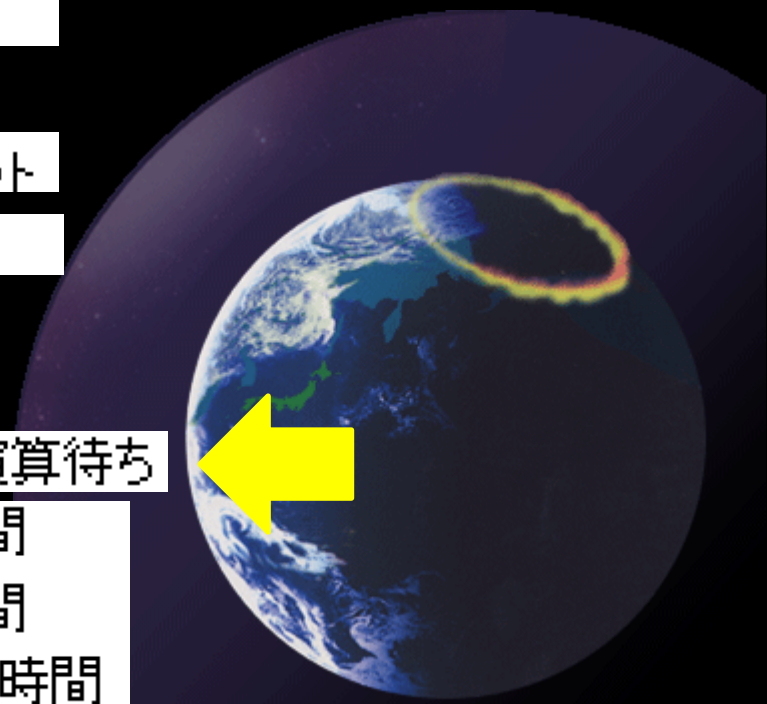
Performance Analysis ツール @FX100 での性能解析

変更前　　変更後



- 京/FX10では有効だった チューニングが効かない！
- 「演算待ち」が増加





チューニングの話

- CPUアーキテクチャや、コンパイラのバージョンによって、「最適なコード」はさまざま

⇒ trial & error!　“後継機種”でも全く異なる

- 演算の種類でも、CPU/コンパイラの得手不得手がある
- 計算機（CPU）が変わる毎やコンパイラのバージョンアップ毎に、性能測定を行ったほうが良い

今後のCPUの動向

- クロック周波数は増ず(減り)、FMAやベクトル演算器(SIMDなど)が増えていく　⇒チューニングしにくい

◎コンパイラが対応するまで時間がかかる

  - SandyBridgeとHaswellは実効速度が変わらなかった・・・
  - Xeon Phiはflat MPIが最速だった・・・

おわりに

学者としてのコメント

- 工学、理学、気象学、化学、薬学、etc
  - アプリケーションユーザ → ソフト・ハードの知識があっても損はない
- 計算科学（ソフトウェア）　アプリ分野の知識があっても損はない
  - 計算スキーム（アルゴリズム）・プログラム開発 → 必要があるかが重要
生業にするのはしんどい
  - 並列化 → 必要不可欠な技術
  - チューニング → 大規模計算では必須
生業にするのはしんどい
無理しなくてよい
- 計算機科学（ハードウェア）
  - スーパーコンピュータシステム → ソフト屋の(チューニングの)しんどさも知ってほしい
  - CPU, メモリ, ネットワーク等のパーツ