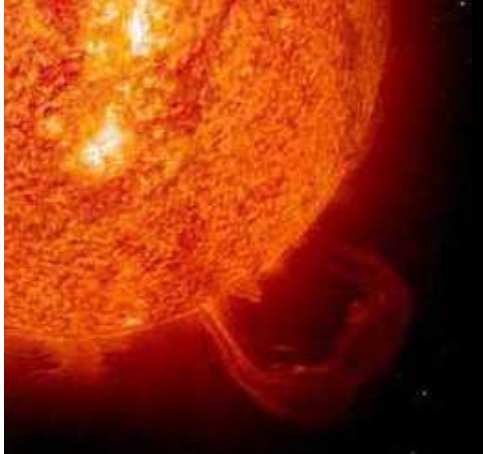


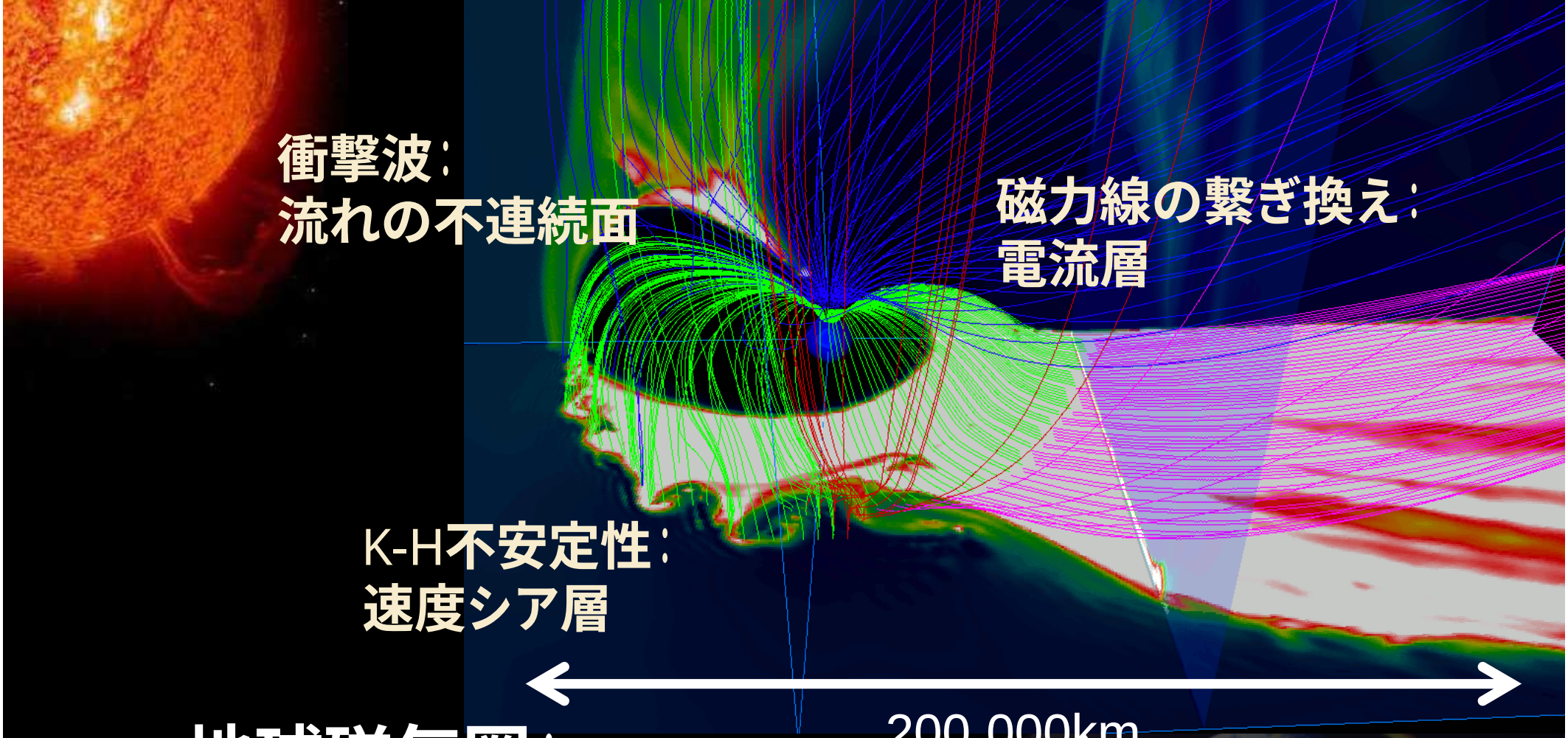
京コンピュータを用いた宇宙プラズマの第一原理ブラソフシミュレーション

梅田 隆行 (umeda@stelab.nagoya-u.ac.jp)
名古屋大学 太陽地球環境研究所

HPCI「京」若手人材育成利用
学際大規模情報基盤共同利用・共同研究(JHPCN)
九州大学 先端的計算科学プロジェクト
東京大学 大規模HPCチャレンジ
名古屋大学HPC計算科学連携研究プロジェクト
SS研マルチコアクラスタ性能WG

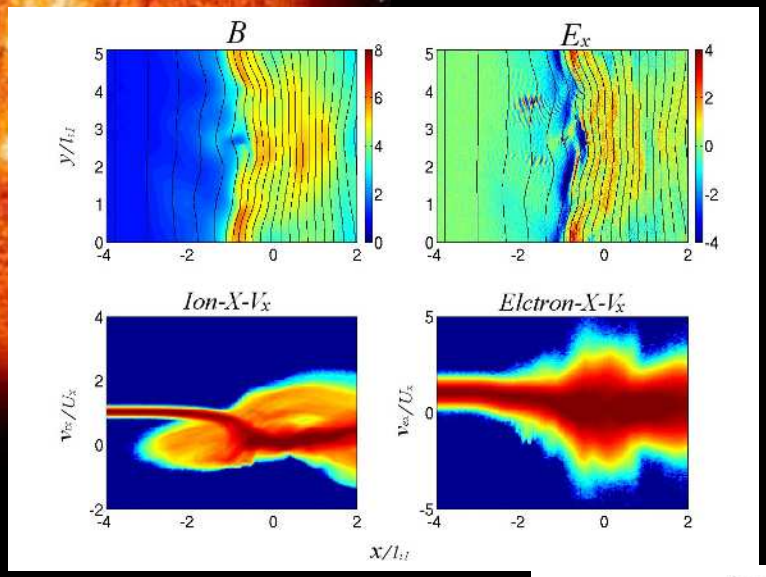
太陽地球環境(STE)のシミュレーション

- プラズマシミュレーション
  - 宇宙の99.99%はプラズマ
  - 特に「無衝突」状態を扱う
    ex.太陽風プラズマの平均自由行程：1AU
  - マルチスケール（流体性＋粒子性）
- 惑星大気シミュレーション
  - 高密度中性大気
  - 流体＋化学反応
- 天文・天体シミュレーション
  - 恒星近傍：高温・高圧・高密度
  - 基本的にMHD＋α

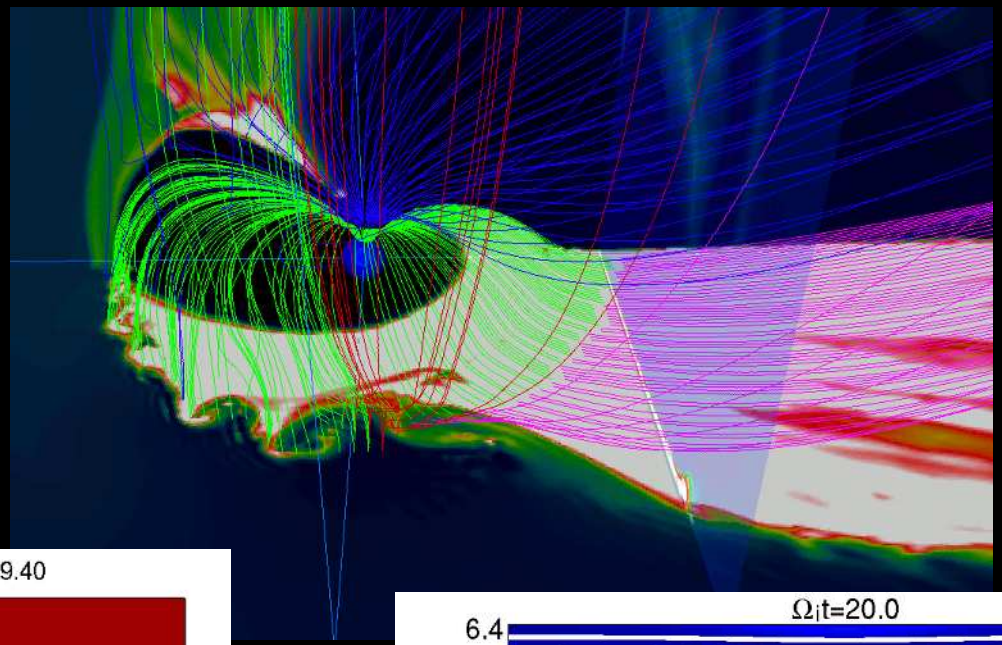


地球磁気圏：
3DグローバルMHD**シミュレーション** (T. Ogino)

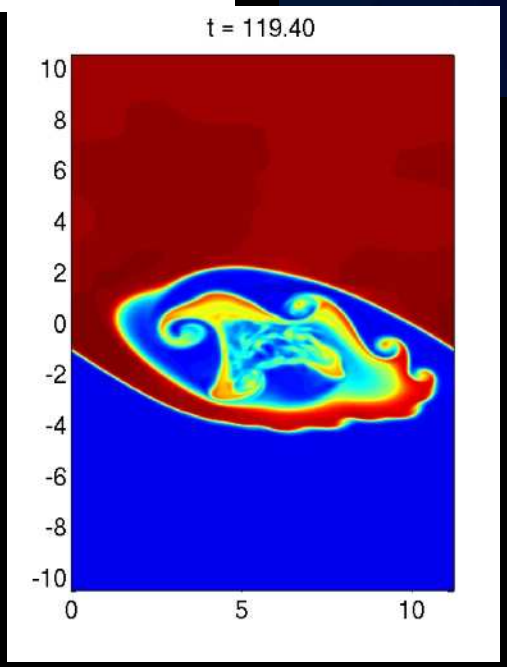
- MHD (**磁気流体力学**) **で記述できるグローバルな構造の中に、様々なメソスケールの境界層が存在**



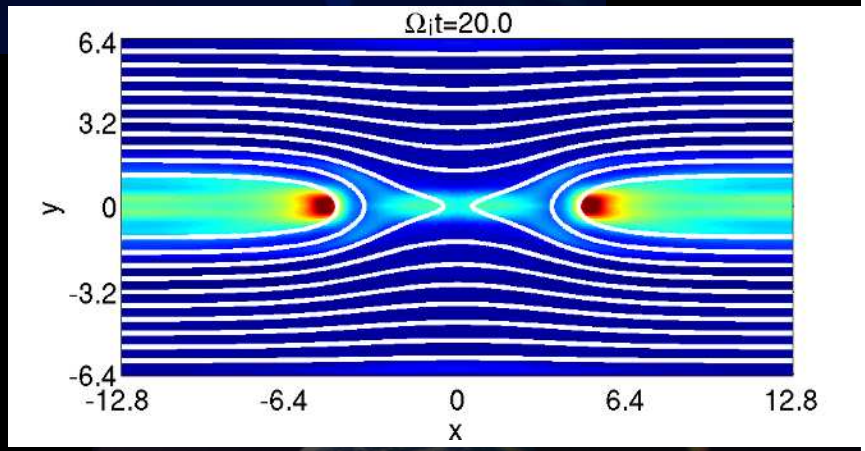
衝撃波



磁力線
繋ぎ変え



KH渦



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

ミクロ：粒子性

MHD

運動論

“グローバル”運動論シミュレーション

目的:

- 宇宙プラズマ科学的観点から、太陽地球系の変動（宇宙天気）を理解・予測する。
- 多スケール(磁気流体力学－境界層－粒子運動論)間の相互作用を理解する。

－磁気圏グローバル構造は基本的にはMHDで記述

－ローカルな境界層（不連続面・シア層・電流層など）における粒子運動論の重要性??

究極のゴール:

磁気圏全てをシームレスに運動論で解く

ゼタ or ヨタスケールコンピューティング！(20 or 30年後)

第一原理 Full Vlasov Model
基礎方程式

位相空間分布関数

$$f_s(x, y, \cancel{z}, v_x, v_y, v_z)$$

6D! ⇒ 5D

40^5 ~ 4GB

40^6 ~ 160GB

Vlasov equation

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

Charge continuity equation

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

Maxwell's equation

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

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

$$\nabla\cdot\mathbf{E} = \frac{\rho}{\varepsilon_0}$$

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

cf. Particle code

$$\frac{\partial \boldsymbol{r}_n}{\partial t} = -\boldsymbol{v}_n \qquad \boldsymbol{r} \equiv (x, y, z)$$

$$\frac{\partial \boldsymbol{v}_n}{\partial t} = \frac{q_n}{m_n}\left(\boldsymbol{E} + \boldsymbol{v}_n\times\boldsymbol{B}\right)$$

ブラソフコード概要1：スキーム

- プラズマ物理からの制約

○ 保存性

○ 正値性

○ 無振動性（新たな極値は発生しないが、既存の極値は保つ）

× TVDスキーム

- メモリ消費量が少ない

×多段時間積分法 (e.g., R-K)

×マルチモーメント(マルチデータ)法 (e.g., CIP)

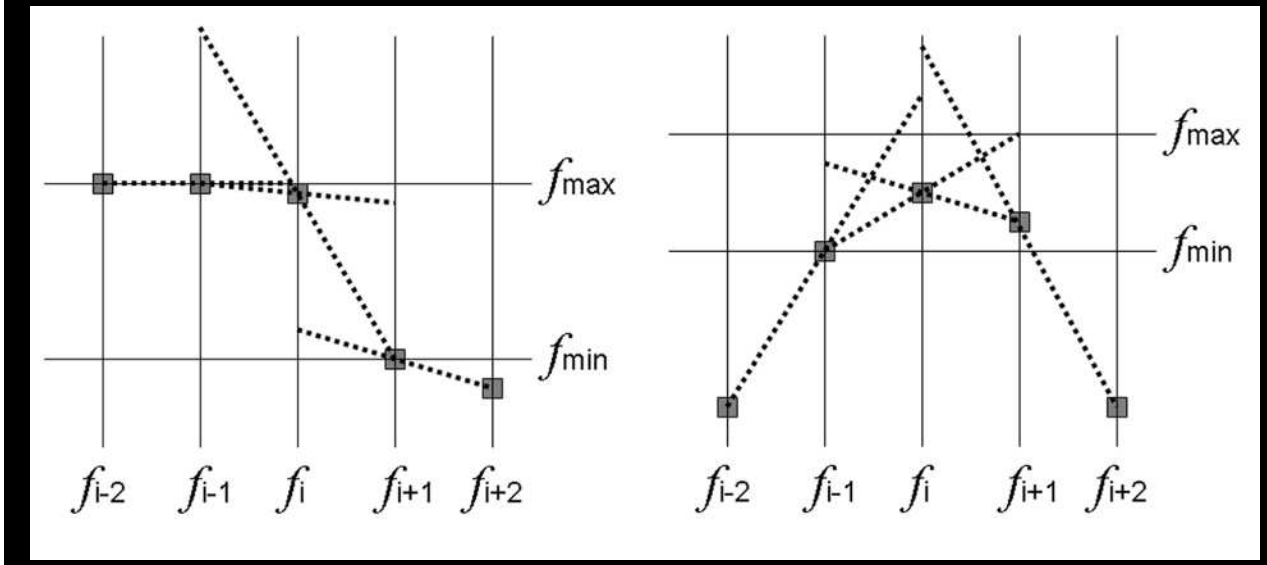
○ 高次補間

⇒高次･保存型セミラグランジュ法＋リミッタ

ブラソフコード概要2：無振動スキーム

$$\frac{\partial f}{\partial t} + V\frac{\partial f}{\partial x} = 0$$

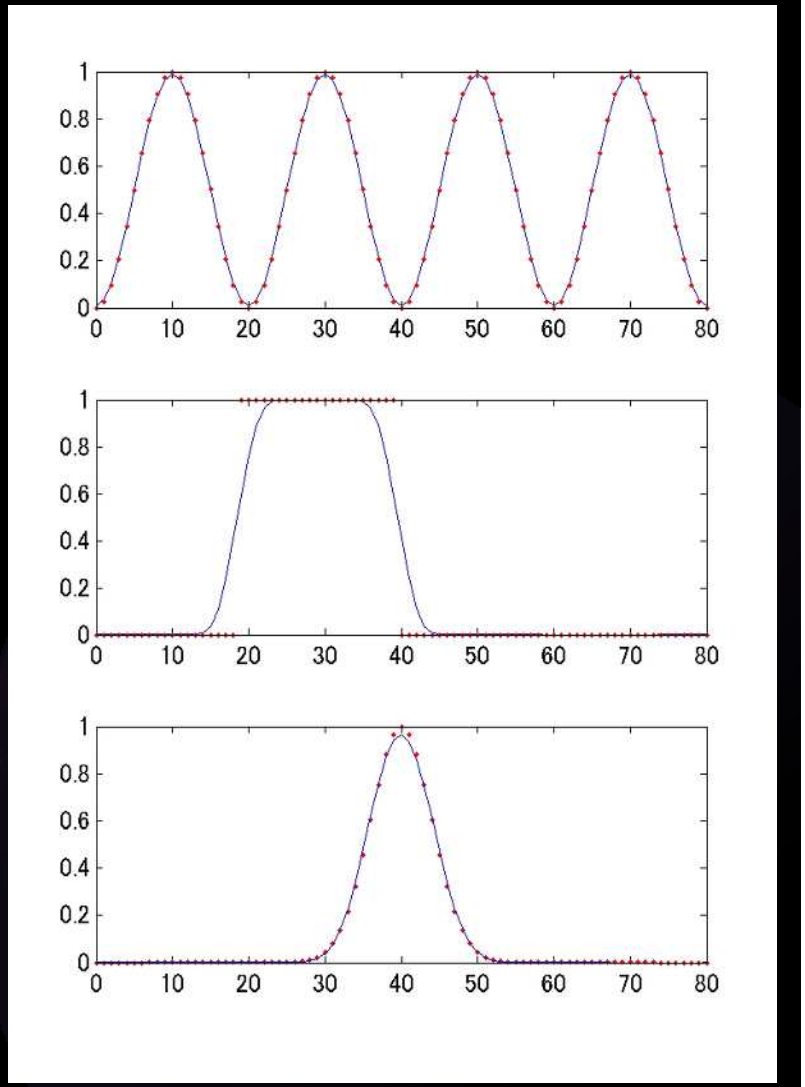
5次精度(保存型)ラグランジュ補間
＋“無振動”・正値性保証リミッタ



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

Umeda EPS 2008
Umeda et al. CPC 2012

Linear Advection Test:



ブラソフコード概要3：並列化

- **実空間格子**: MPI/OpenMP
- **速度空間格子**: OpenMP
  モーメント量 (Jx, Jy, Jz, ρ)**の計算に** reduction**処理が必要**
- 1GB/core：26×46×30×30×30
  境界の“のりしろ”（袖領域）は前後3点

Umeda et al. IEEE Plasma Sci 2012

ブラソフコード概要：まとめ

- ブラソフ方程式（分布関数）：
  - 自作の5次精度保存型・無振動スキーム
  - 通信：MPI_Sendrecv、のりしろ：前後に3点
- マックスウェル方程式（電磁場）：
  - CG法（陰解法）、だいたい30 iterationsで収束
  - 通信：MPI_Sendrecv、のりしろ：前後に1点
    MPI_Allreduce ⇒収束レベルの確認
  - CPU負荷：全体の0.1%以下
    ただし、ノード数が増えると通信負荷が急増
- 粒子コードと比べて、ノイズレスかつ並列化が容易
  ただし、必要メモリ量はケタ違いに多い

「京」でできること

- 20-30年先（実物大フル6次元計算）を踏まえつつ、今（京）でできることを考える。
- 小天体の低解像度6次元グローバルブラソフシミュレーション
  - ただし、ノードあたりのメモリが少ない「京」では、(16GB/node) 6次元コードは動作しない:30^6~40GB
- **小天体の5次元（実空間2次元・速度空間3次元）高解像度グローバルブラソフシミュレーション**
  - 低解像度計算は既に行った

「京」による弱磁化小天体の5次元“グローバル”ブラソフシミュレーション

	本研究	月
天体半径	2.72km	1738km
イオンジャイロ半径	1.36km	85km
イオンジャイロ周波数	9.36Hz	0.15Hz
マス比 m_i/m_e	25	1836

◎計算サイズ: ~50TB ⇒ ~8GB/node

・Spatial grids：1920×2560 ⇒ 20×40/node

・Velocity grids ：60×60×60 ⇒ 60×60×60/node

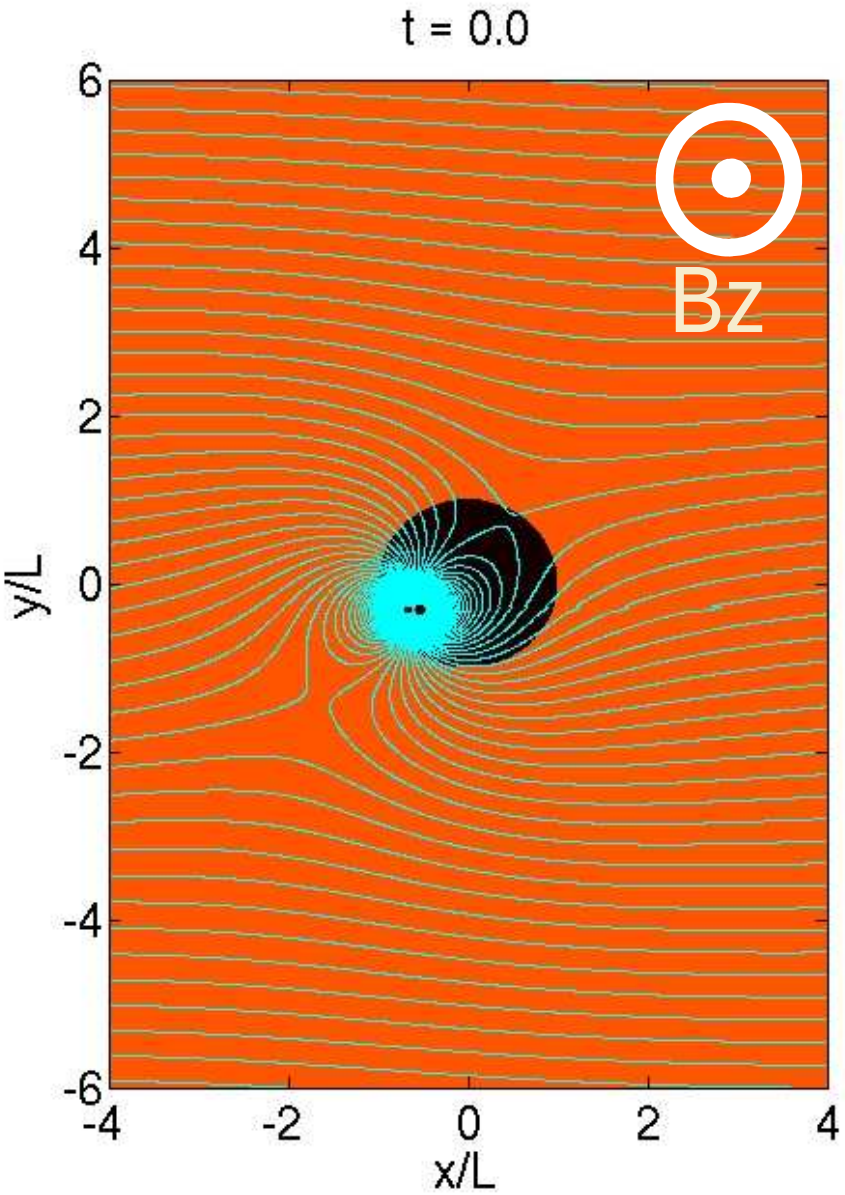
◎ノード数：6144 (96×64) ⇒全体の約14分の1

⇒FX10@東大では3072ノード必要 「京」でしかできない

◎時間ステップ数：100,000 ($10/\Omega_{ci}$) ⇒約100日

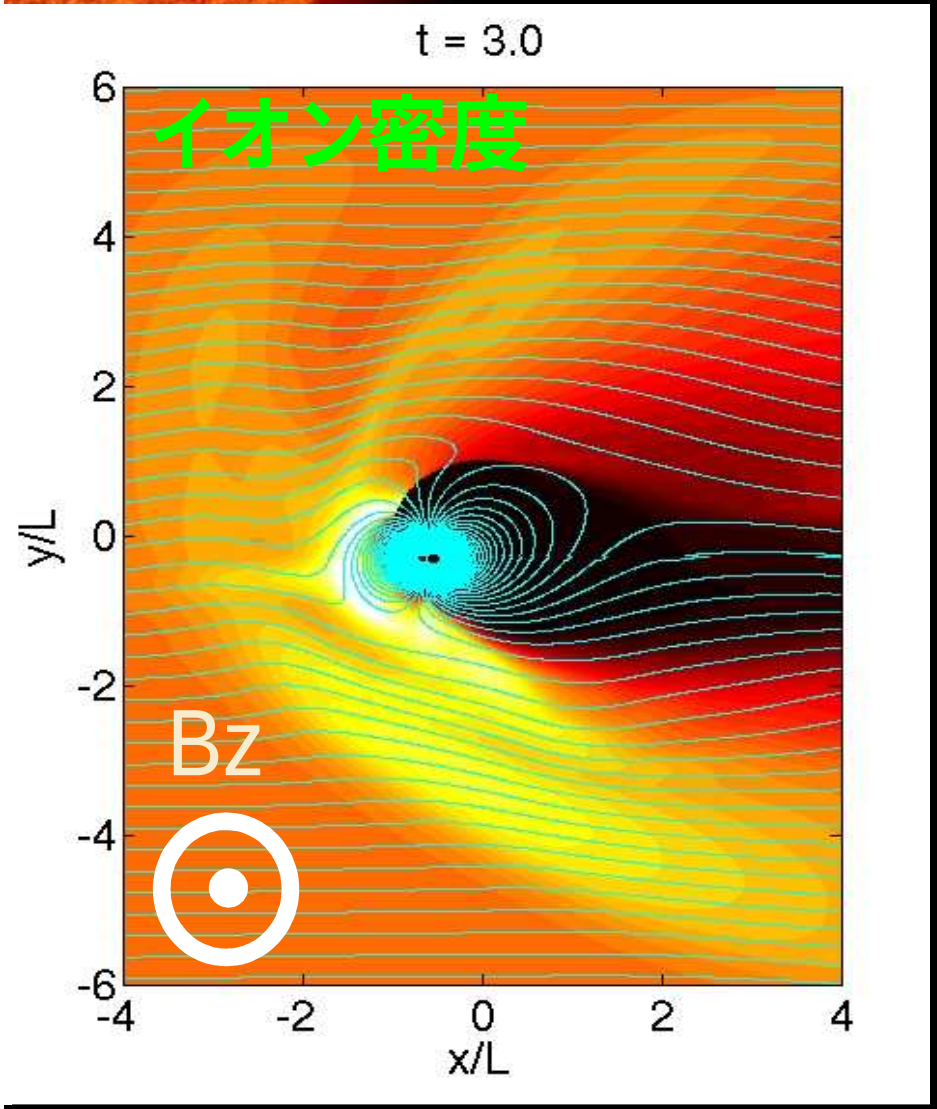
計算の途中結果：初期値

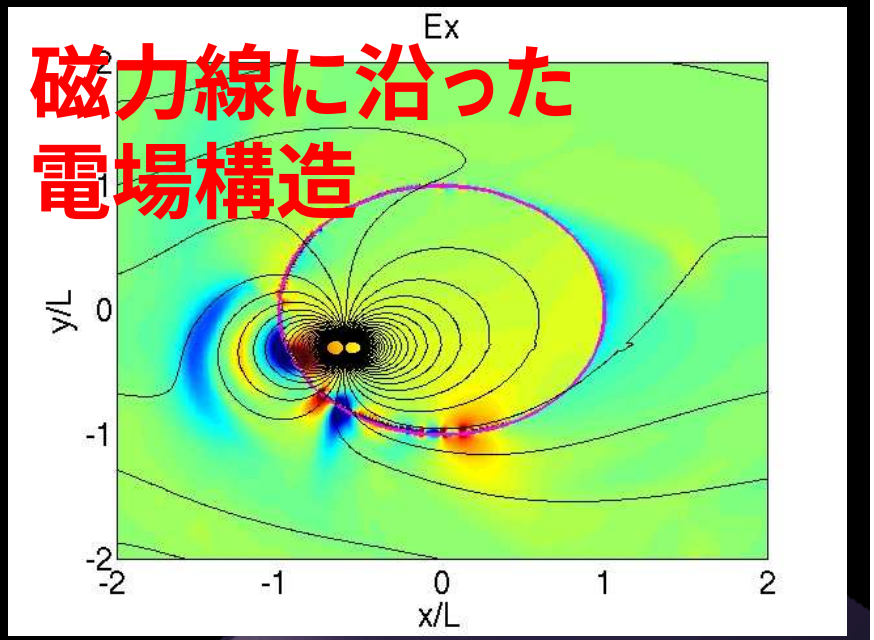
磁力線とイオン密度

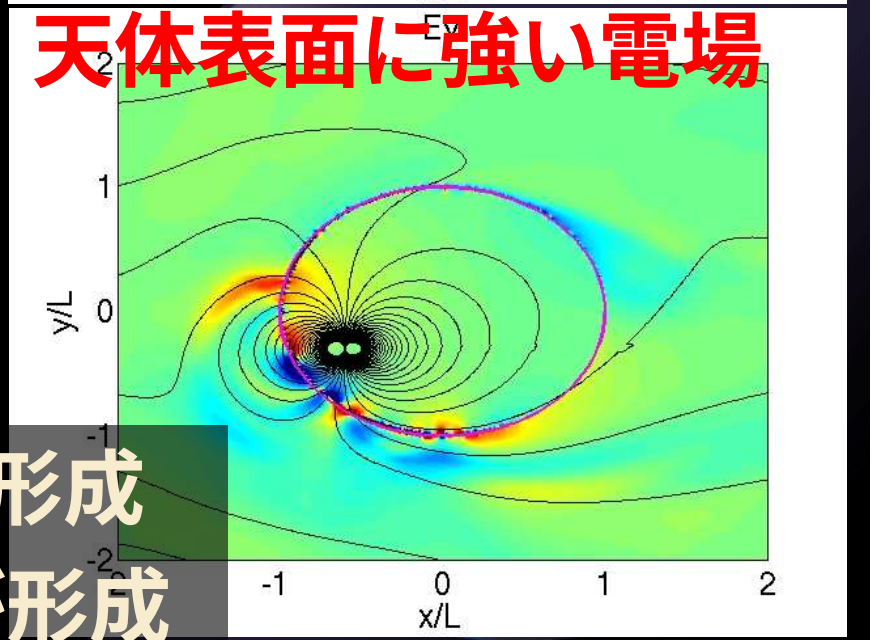


- 昼側（左側）境界から太陽風プラズマが入射
- 昼側に弱いダイポール磁場を配置
- プラズマは天体表面に吸着、電磁波は透過
- 天体半径＝イオンジャイロ半径の2倍
- 昼側に、イオンジャイロ半径スケールの磁気圏（磁力線の閉じた領域）がある
- 夜側の磁力線は、基本的には開いている

計算の途中結果：$\omega_{ci}t=3.0$







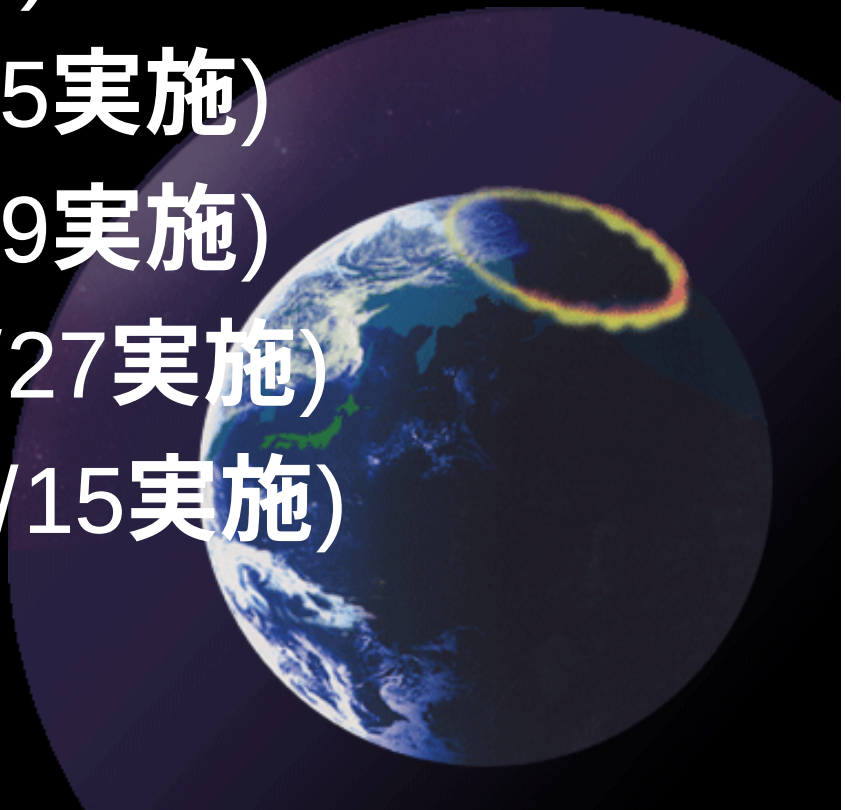
- 昼側に、高密度領域（衝撃波）が形成
- 夜側に、低密度領域（ウェイク）が形成

性能評価

- コアあたりのメモリ使用量を～1GB
  (26 x 46 x 30 x 30 x 30)
  に固定した、弱いスケーリング
  - 実際の計算に近い設定
- コンパイラオプション
  - FX10/K：–Kfast,openmp,visimpact,simd=2 –x250
  - FX1：–Kfast,OMP,impact –x250
  - CX400：–Kfast,openmp,parallel,AVX –x250
    : –ip –ipo –O3 –xAVX –openmp

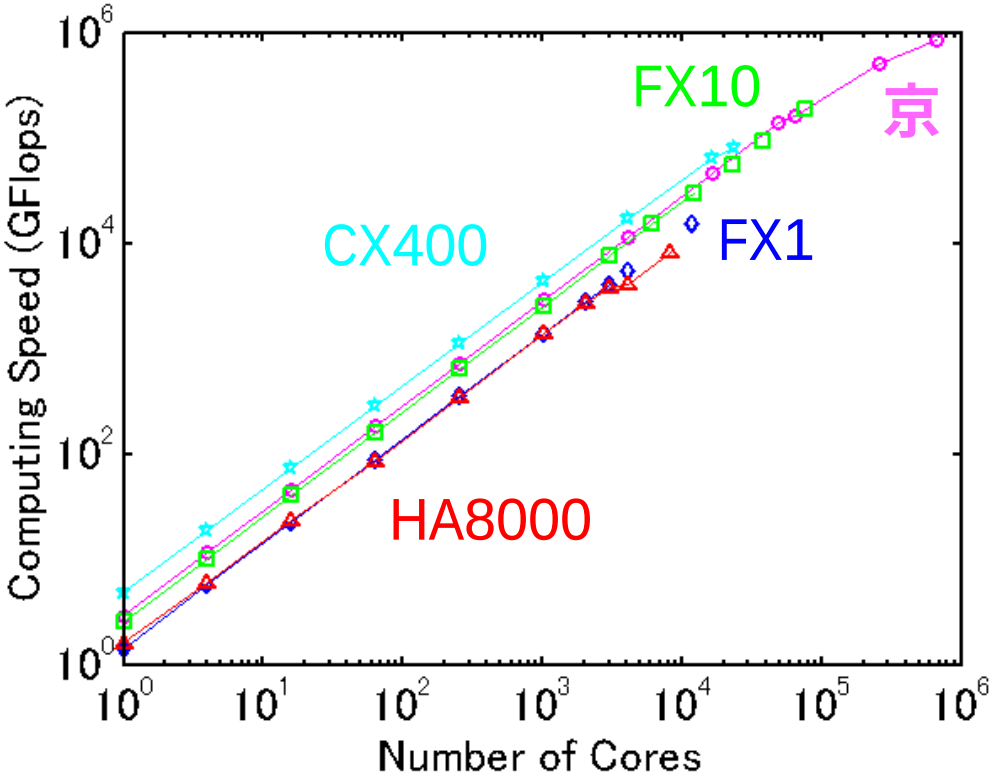
性能評価に用いたマシン

- 東京大学 HA8000 (H22/10/21実施)
- 名古屋大学 FX1 (H24/01/04実施)
- JAXA FX1 (H21/03/11実施)
- 東京大学 FX10 (H24/10/25実施)
- 九州大学 FX10 (H25/03/29実施)
- 九州大学 CX400 (H25/06/27実施)
- 理化学研究所 京 (H25/08/15実施)

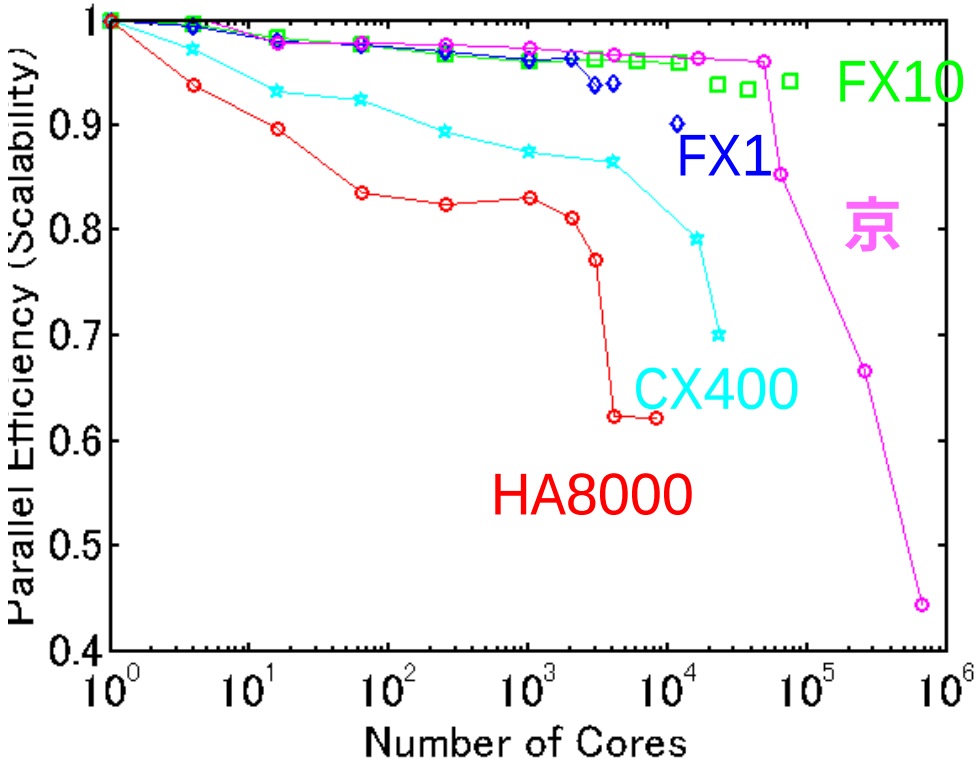


性能評価

実効速度



実効並列効率

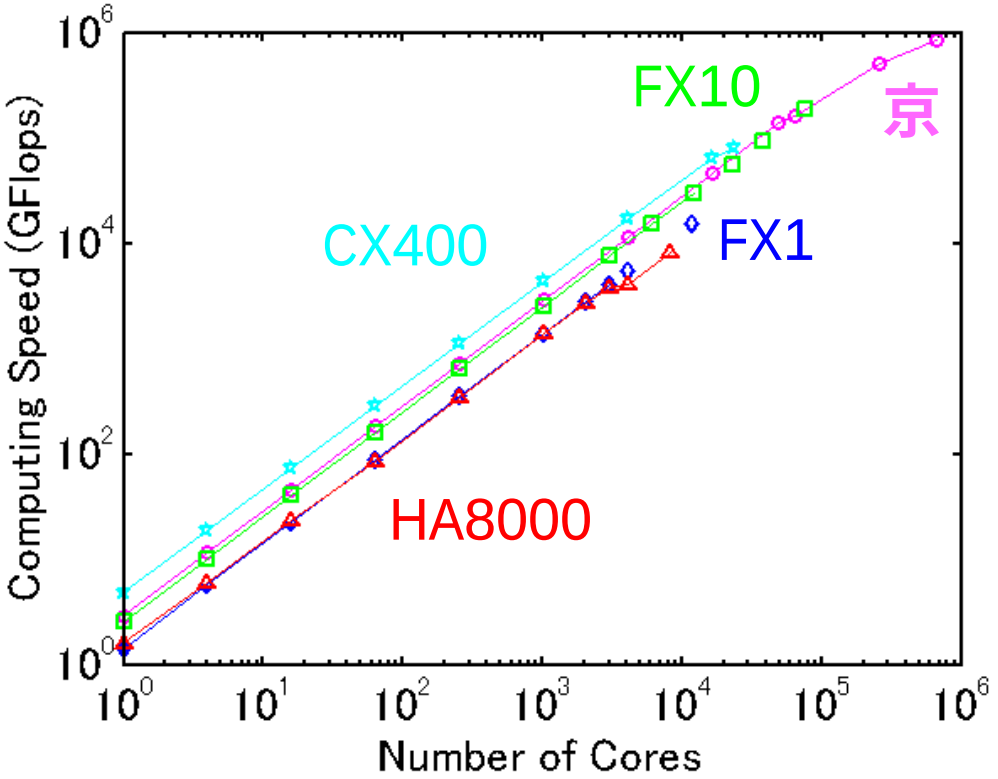


HA8000(@東大、日立): 実効効率~14%

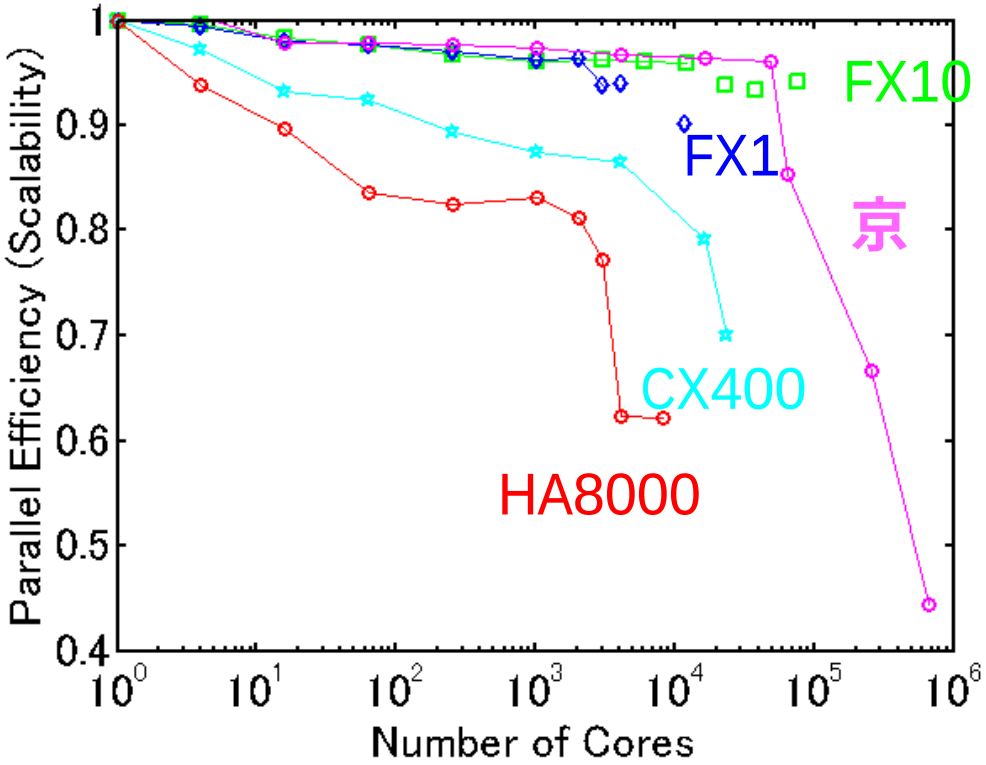
- 日立コンパイラ+フラットMPIが最速、
  ただしOpenMPが　うまく動かなかった
- 256ノード(4096コア)以上で性能が低下、
  ただし512ノードと効率が変わらず　⇒ネットワークの飽和

性能評価

実効速度



実効並列効率

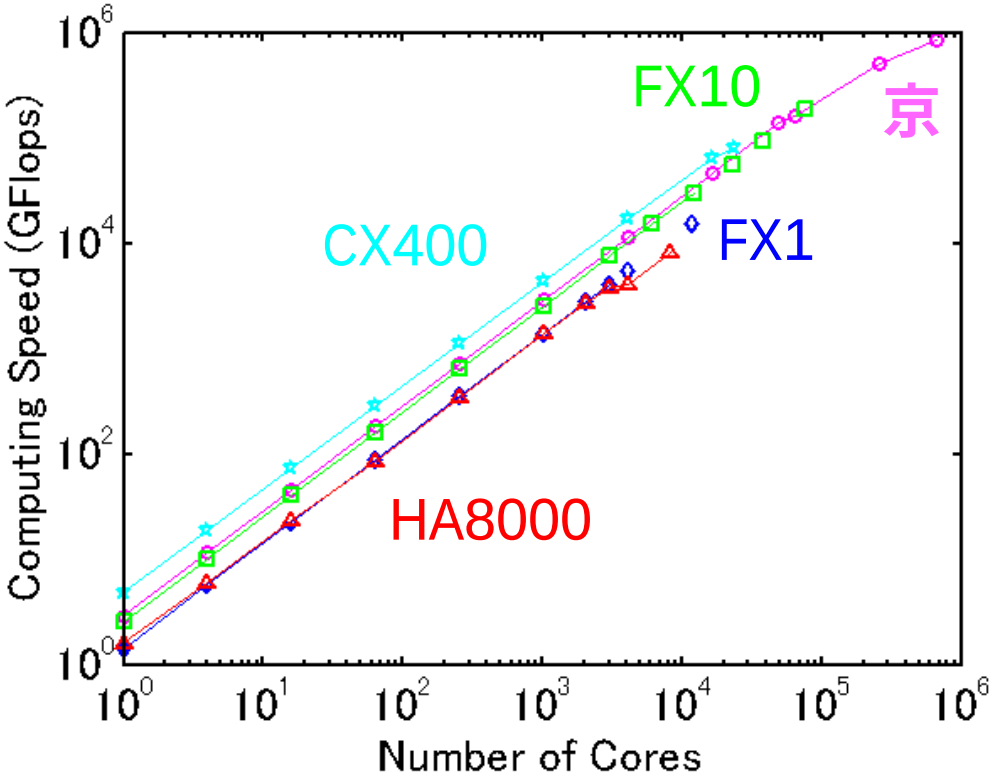


CX400@九大: 実効効率~21%

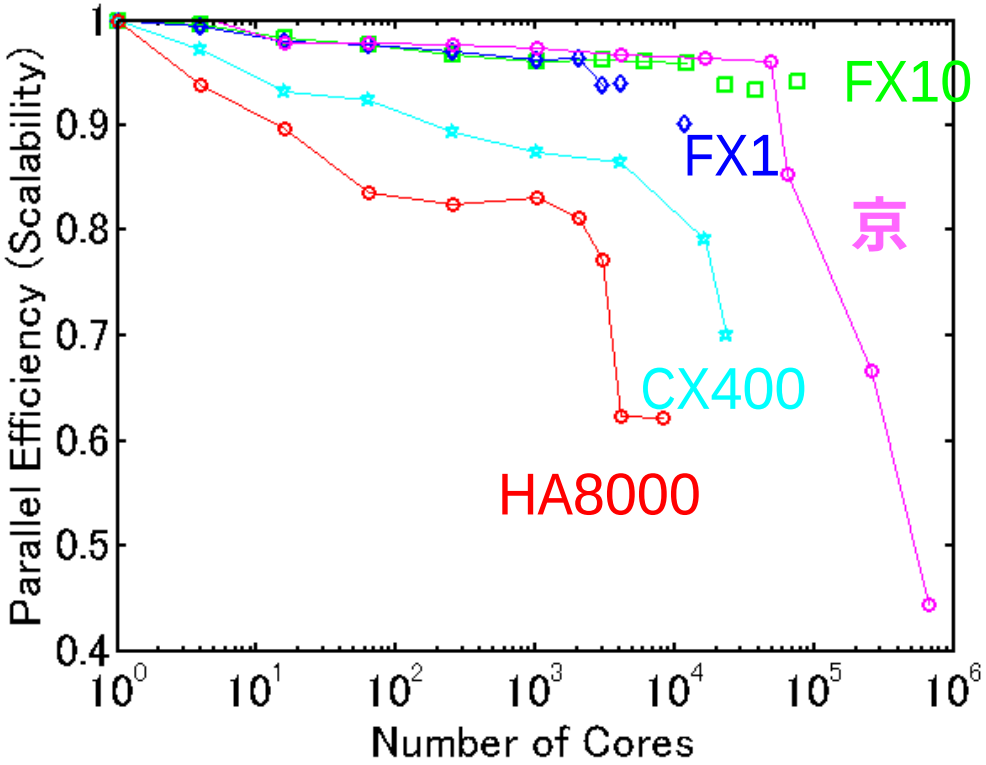
- Flat MPIが最速
- インテルコンパイラが富士通コンパイラよりも1.5倍高速
- 256ノード (4096コア) を超えると急激に性能が低下
  ⇒ネットワークトポロジーの問題 (全ノードFBBでない)

性能評価

実効速度



実効並列効率

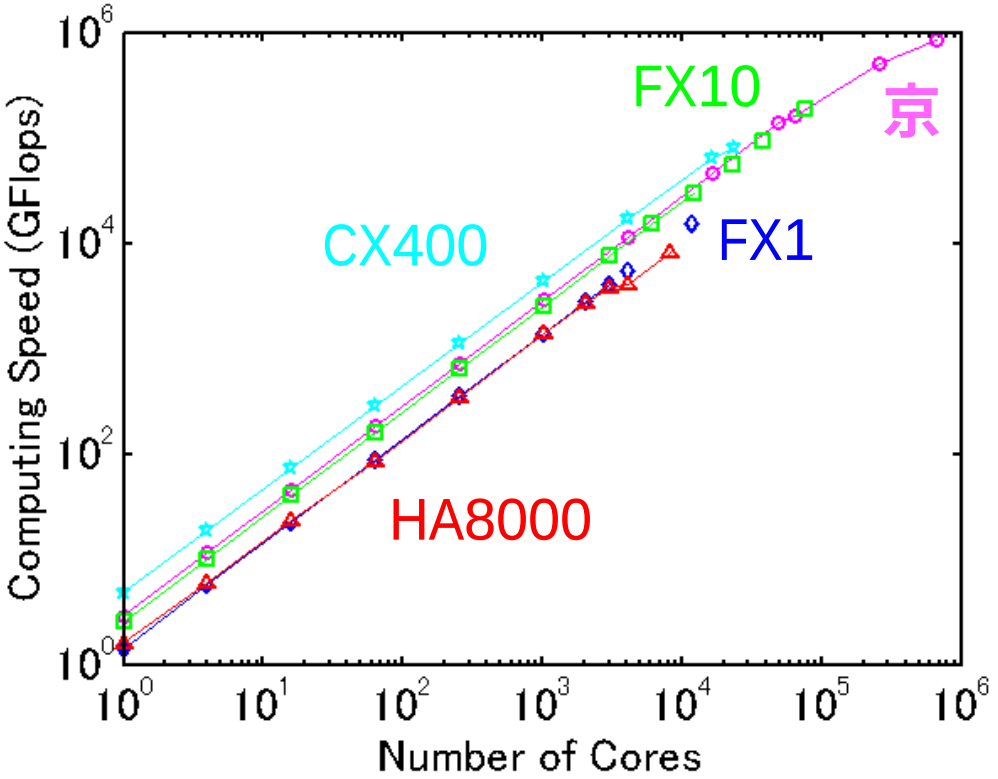


FX1@名大・JAXA：実効効率~15%

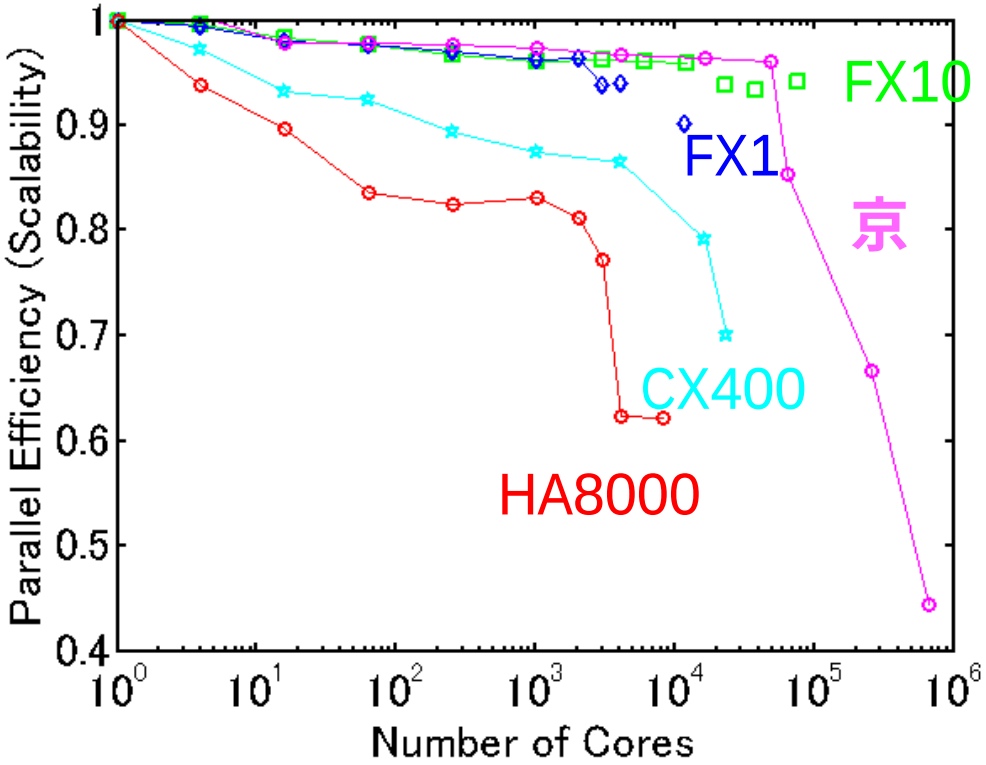
- Flat MPIが<u>若干</u>ハイブリッド並列よりも速い
- 全ノードを使用した場合（3008ノード@JAXA, 768ノード@名大）に、低下(実効効率~13%)

性能評価

実効速度



実効並列効率



FX10/京：実効効率~18%

- 2 or 4スレッドのハイブリッド並列が速い
- FX10では、4800ノード（76,800コア）まで性能を維持
- 京では、8192ノード（65,536コア）から急激に性能が低下
  ⇒陰解法FDTDで使っているMPI_Allreduceの問題

性能チューニング

協力：SS研マルチコアクラスタ性能WG、富士通関係者様

- As-is コード：Xeon(Woodcrest-Nehalem)で開発
  - 768ノード@FX1 (名大) で実効効率13%
  - 512ノード@HA8000 (東大) で実効効率11%
  - ノード数があまり多くないOpteronで実効効率 14~15%
  - ノード数があまり多くないXeon(Westmere)で実効効率20%
- FX10にそのまま移植すると…
  - 1440ノード@FX10 (東大) で実効効率11%
  - 全ノード(4800ノード)計測に失敗

性能チューニング①

- **配列のマージ**

```
real(kind=8)::ax(ix,iy,iz),ay(ix,iy,iz),az(ix,iy,iz)
```

Type A

```
real(kind=8) :: array(ix,iy,iz,3)
#define array(i,j,k,1) ax(i,j,k)
#define array(i,j,k,2) ay(i,j,k)
#define array(i,j,k,3) az(i,j,k)
```

Type B

```
real(kind=8) :: array(3,ix,iy,iz)
#define array(1,i,j,k) ax(i,j,k)
#define array(2,i,j,k) ay(i,j,k)
#define array(3,i,j,k) az(i,j,k)
```

配列 ax,ay,az **に対して連続的、**i,j,k**に対してストライド**

性能チューニング①

- Type A　`array(i,j,k,3)`
  - 配列アクセスのSIMD化を促進
  - 1次元目のサイズに注意
    2のn乗付近でパディングが必要
- Type B　`array(3,i,j,k)`
  - ストリーム数（配列(ax,ay,az)へのアクセス命令数）が減るが、SIMD化は無し
  - パディング不要
  - キャッシュデータの再利用性に注意
- **どちらが速いか、全ての配列について試す！**

性能チューニング②

```
do nn=2,nvzp1
  uz1=vz(nn,ss)
  do mm=1,nvyp1
    uy1=vy(mm,ss)
    do ll=2,nvxp1
      ux1=vx(ll,ss)

      vv =ux1*bbx+uy1*bby+uz1*bbz
      mv =sign(1.0d0,vv)
      mp0=mm-floor(vv)
      mp2=min(max(mp0+mv+mv,1),nvyp2)
      mp1=min(max(mp0+mv   ,1),nvyp2)
      mm1=min(max(mp0-mv   ,1),nvyp2)
      mm2=min(max(mp0-mv-mv,1),nvyp2)

      hp2=ff(ll,mp2,nn,ii,jj,ss)
      hp1=ff(ll,mp1,nn,ii,jj,ss)
      hp0=ff(ll,mp0,nn,ii,jj,ss)
      hm1=ff(ll,mm1,nn,ii,jj,ss)
      hm2=ff(ll,mm2,nn,ii,jj,ss)
      dfy(ll,mm)=flux(hp2,hp1,hp0,hm1,hm2,vv)
    end do
  end do
```

As-isコード：
FX1,RX200でチューニング済
実効効率:13%, 20%

ループ内で複数の組み込み関数とユーザ定義関数を呼ぶ

外側にさらに3重ループ

性能チューニング②

```
do mm=1,nvyp1
  uy1=vy(mm,ss)
  do ll=2,nvxp1
    ux1=vx(ll,ss)
    vv(ll,mm) =ux1*bbx+uy1*bby+uz1*bbz
    mv =sign(1.0d0,vv(ll,mm))
    mp0(ll,mm)=mm-floor(vv(ll,mm))
    mp2(ll,mm)=min(max(mp0+mv+mv,1),nvyp2)
    mp1(ll,mm)=min(max(mp0+mv   ,1),nvyp2)
    mm1(ll,mm)=min(max(mp0-mv   ,1),nvyp2)
    mm2(ll,mm)=min(max(mp0-mv-mv,1),nvyp2)
  end do
end do
do mm=1,nvyp1
  do ll=2,nvxp1
    hp2=ff(ll,mp2(ll,mm),nn,ii,jj,ss)
    hp1=ff(ll,mp1(ll,mm),nn,ii,jj,ss)
    hp0=ff(ll,mp0(ll,mm),nn,ii,jj,ss)
    hm1=ff(ll,mm1(ll,mm),nn,ii,jj,ss)
    hm2=ff(ll,mm2(ll,mm),nn,ii,jj,ss)
    dfy(ll,mm)=flux(hp2,hp1,hp0,hm1,hm2,vv(ll,mm))
  end do
end do
```

ループを分割
floorとfluxを分ける

データ受け渡し用配列は2次元
（1次元より若干速い
3次元にするとかなり遅くなる）
CX400とFX1で実効効率が1%向上
（FX10/京は0.5%向上）

性能チューニング③

```
do mm=1,nvyp1
  uy1=vy(mm,ss)
  do ll=2,nvxp1
    ux1=vx(ll,ss)

    vv(ll,mm) =ux1*bbx+uy1*bby+uz1*bbz
    sv =sign(1.0d0,vv(ll,mm))
    mp0(ll,mm)=mm-floor(vv(ll,mm))
    wmp0  =mp0(ll,mm)
    wnvyp2=nvyp2
    mp2(ll,mm)=min(max(wmp0+sv+sv,1.0d0),wnvyp2)
    mp1(ll,mm)=min(max(wmp0+sv   ,1.0d0),wnvyp2)
    mm1(ll,mm)=min(max(wmp0-sv   ,1.0d0),wnvyp2)
    mm2(ll,mm)=min(max(wmp0-sv-sv,1.0d0),wnvyp2)
  end do
end do
```

max, min を整数から
倍精度実数演算に変更

CX400とFX1：型変換の演算が増えたため、
実効効率が1%低下

FX10と京：実効効率が3%向上!! (~18%)
“特殊系”チューニング（整数関数が遅すぎる…）

「一般ユーザ」として京を使ってみて…

- 待ち時間が長い…
  - 他のシステムだと、全リソースの10分の1以下なら1日待ちぐらいで走る
- 大量ファイル操作のストレス

⇒6万個のデータファイルと2万個のリスタート用ファイル

フロントエンドがlsで固まる…

- 市販の（非並列）ソフトウェアを用いると、図を描くだけでかなり時間がかかる

（ある物理量＋磁力線の図で3時間）

今後：6次元計算

- $40^3 \times 40^3$ ~ 160GB
- $30^3 \times 30^3$ ~ 40GB
- $20^3 \times 40^3$ ~ 20GB –32GBで動くサイズ

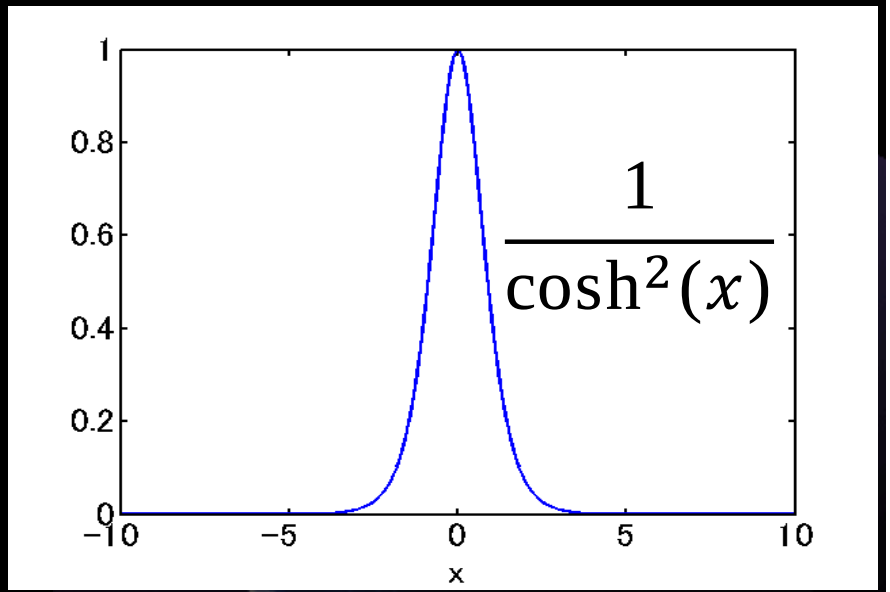
⇒このサイズをどうノード内並列するか？？

- 既存のノード内スレッド並列（ループ1次元のみをOpenMP/自動並列）では、メニーコア環境（>32スレッド??）に対応できない

⇒ノード内スレッド＋ノード間MPIのハイブリッド並列は不可

大規模計算ならではの「これは困った」

- 引数が大きくなって関数がオーバーフローする
- (コンパイラ) オーバーフローの回避
  - 既にオプション有あり？

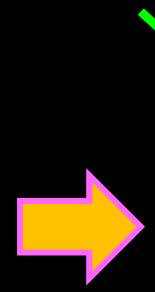


- x → 大
- $\cosh^2(x)$ → NaN (1.0E+300)
  $1/\cosh^2(x)$ もNaN
- インテルCPUだと、$\cosh^2(x)$ → Inf
  (富士通コンパイラでも) 1/Inf → 0.0

6次元計算に向けて①

- (ハード) ノード内共有メモリはある程度欲しい
  - プロセス数 (ノード間通信量) を減らす
  - ループ長を伸ばす
  - データファイル数を減らす
- (コンパイラ) 多次元ループに対応した
  スレッド並列化機能が欲しい

```
do nn=1,30
do mm=1,30
do ll=1,30
```



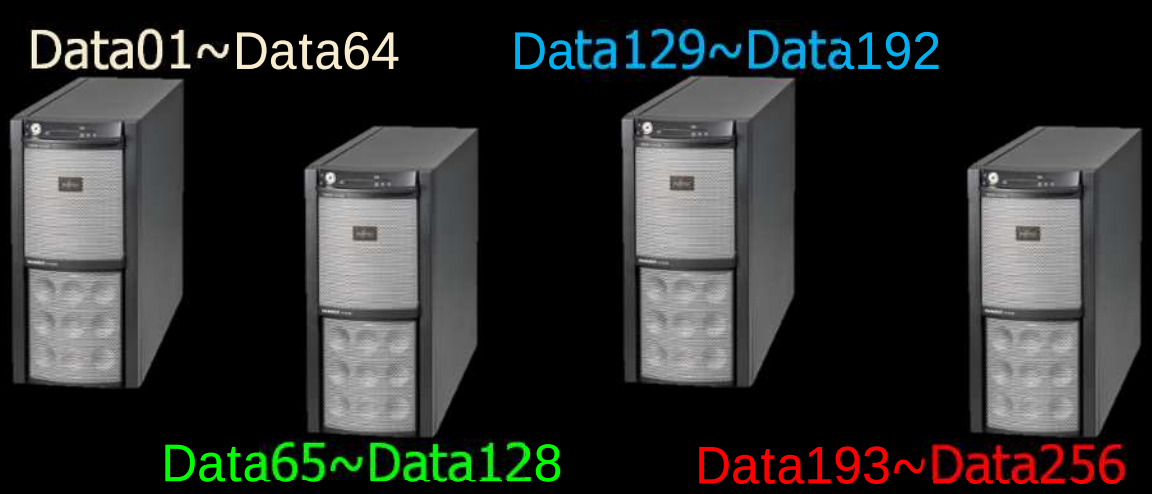
```
do nn=n1,n2
do mm=1,30
do ll=1,30
```

```
do nn=n1,n2
do mm=m1,m2
do ll=l1,l2
```

6次元計算に向けて②

- 1変数あたり数千ファイルのデータ解析処理が思った以上に大変…
  - 市販の解析ソフト（非並列）で、1つの図を描くのに数時間…

  ⇒プロセス数をできるだけ減らしたい
- (フリーな)分散データ解析環境？



Group1	Group2
Group3	Group4

まとめ

- 20〜30年後の地球磁気圏の第一原理6次元シミュレーションに向けて、「京」を用いた5次元シミュレーションを実行中
- 既存のハード・コンパイラ・データ解析環境の延長では、エクサでさえ厳しい…
  - ノード内メモリの増加
  - 多次元ループのスレッド化
  - 大量ファイルのデータ解析処理

  ⇒プロセス数はできるだけ減らしたい