CHIBA UNIVERSITY

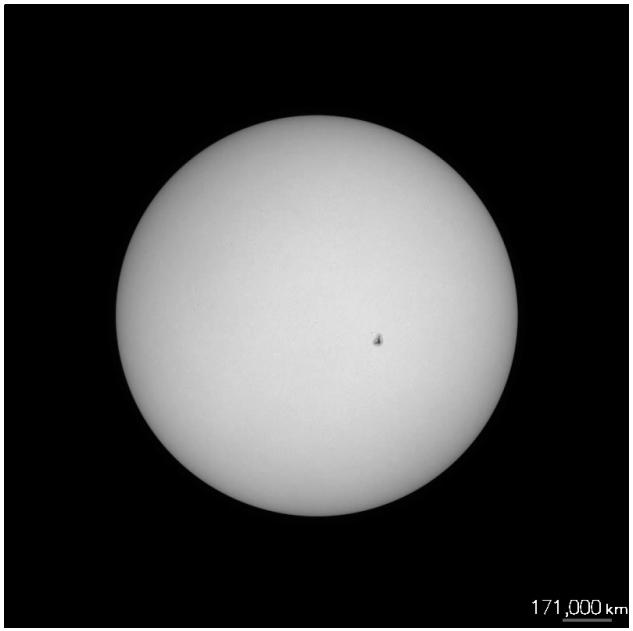
富岳における太陽内部計算

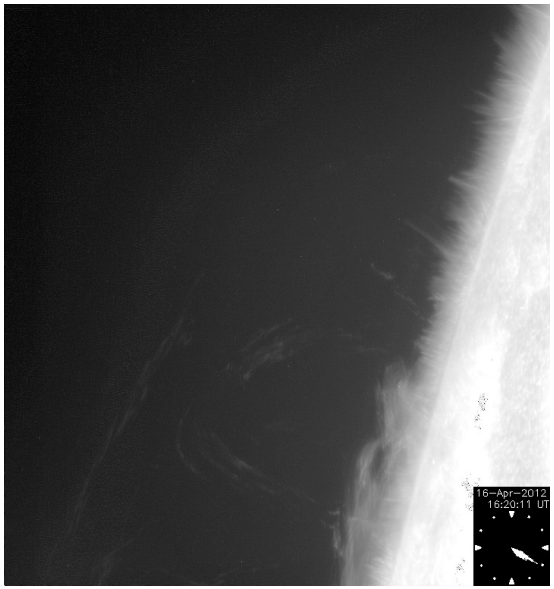
Hotta & Kusano, 2021, Nature Astronomy accepted

堀田英之(千葉大学)

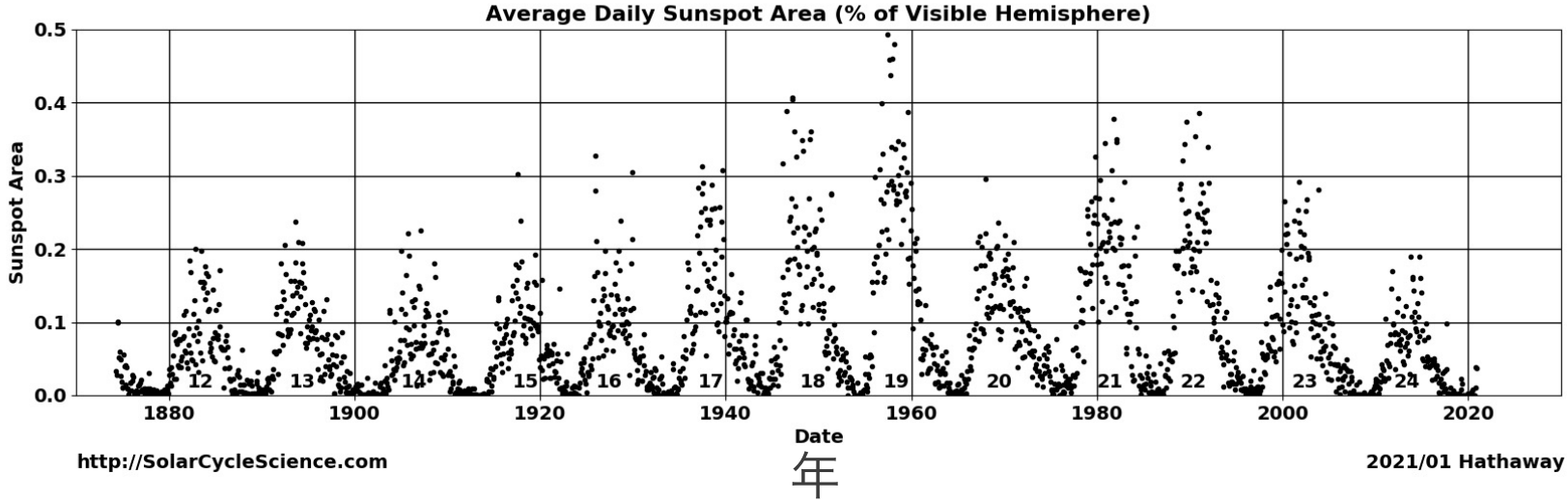
共同研究者：草野完也(名古屋大学)

太陽





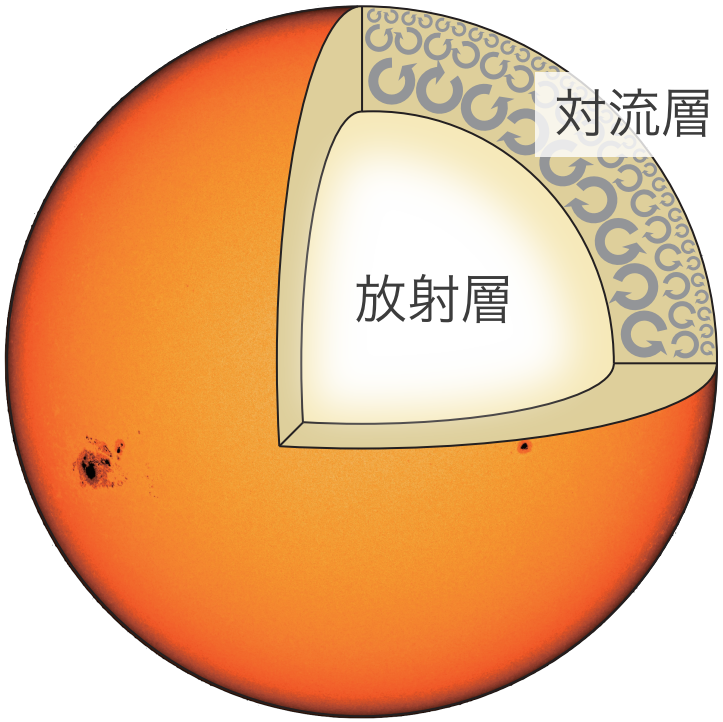
NAOJ/JAXA/NASA



黒点の面積

年

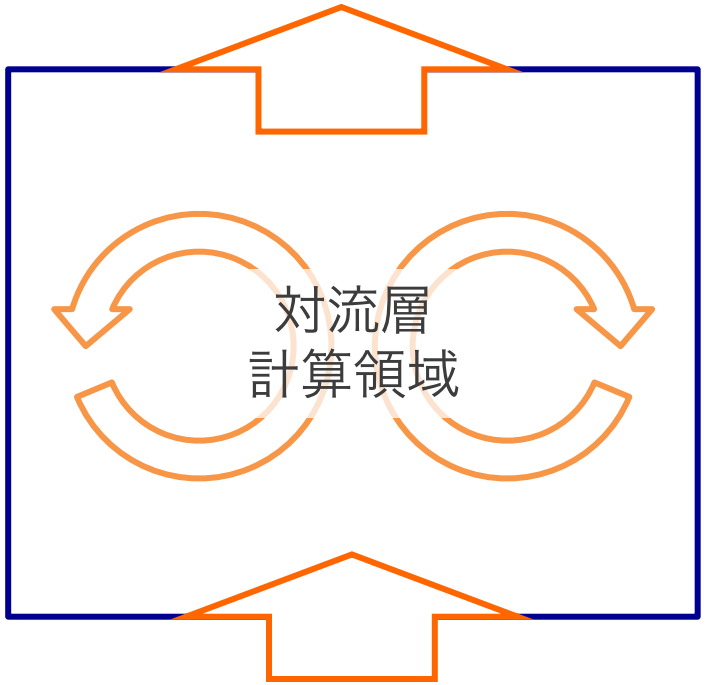
太陽内部構造



$$L_\odot = 3.84 \times 10^{33}\ \mathrm{erg\ s^{-1}}$$

11年周期で0.1%の変動あり

輻射エネルギーフラックス



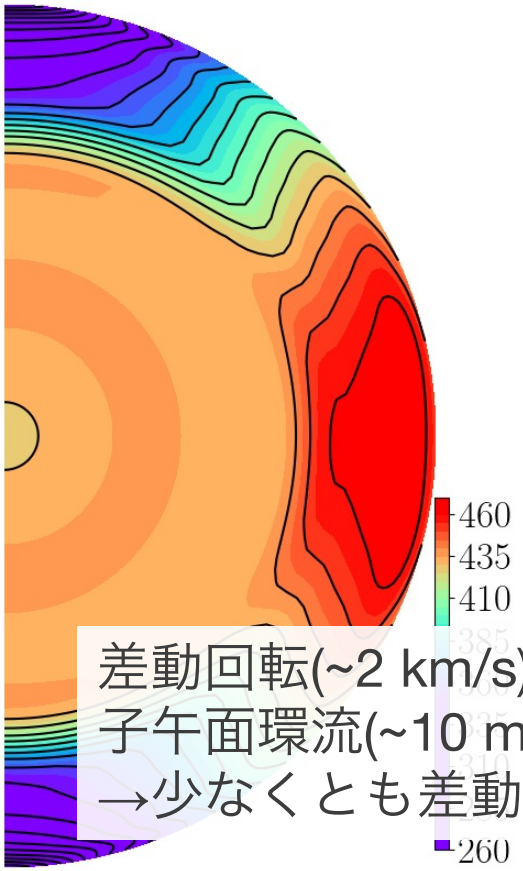
輻射エネルギーフラックス
(よく知られた量)

内部構造(密度、圧力、温度...)などは理論観測により、非常に精密に決定している。
運ぶべきエネルギー量も精密に決定。
初期条件・パラメタの不定性なし。「正しく計算」すれば「正しい答え」が出るはず。

大規模流れ

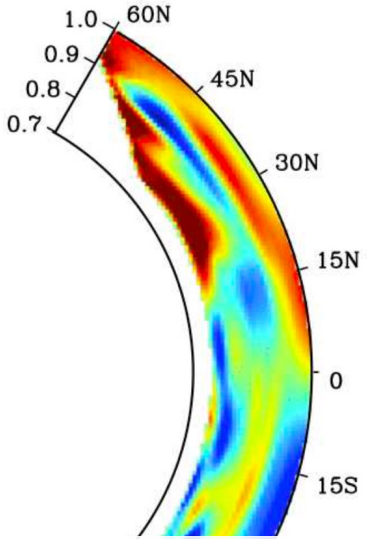
観測がしっかりしているので、11年周期だけを再現できてもだめ

差動回転の観測
(R. Howeデータ)

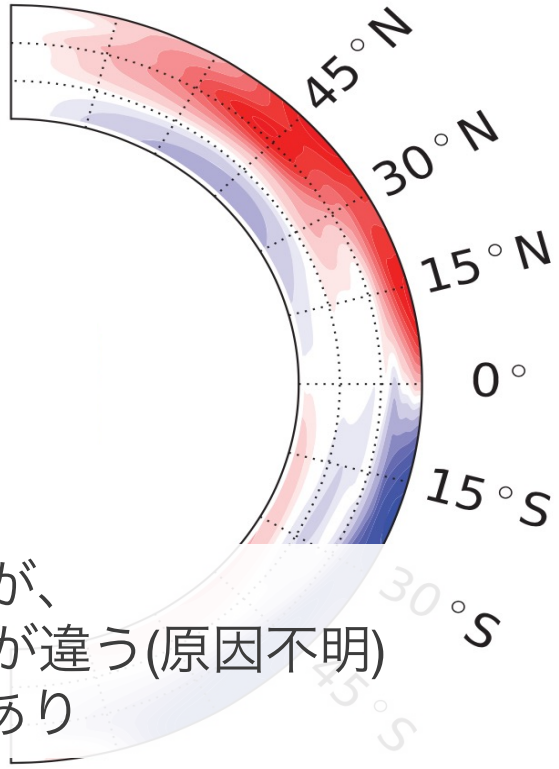


子午面環流

Zhao+2013
SDO



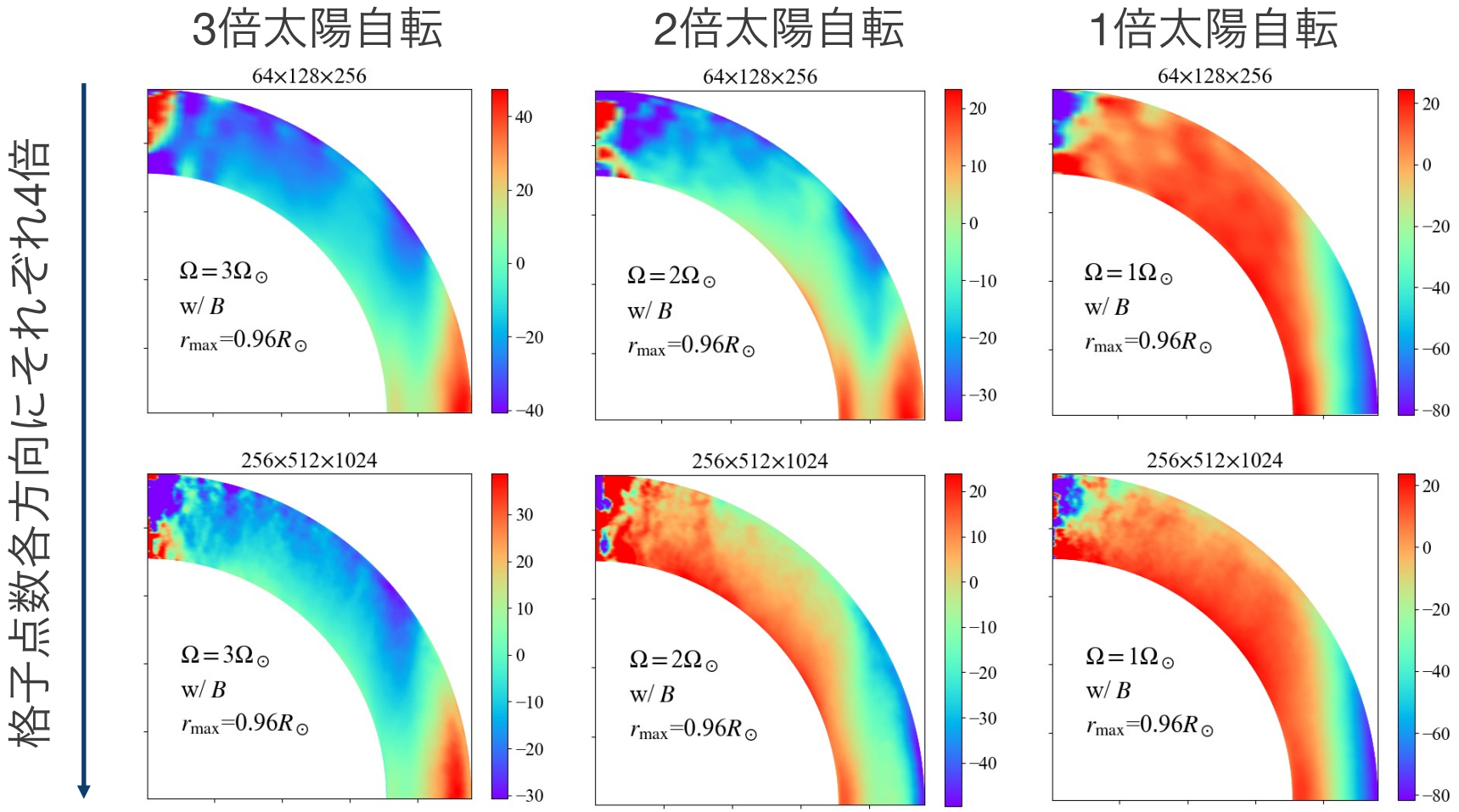
Gizon+2020
SOHO, GONG



差動回転(~2 km/s)の構造はほぼ確定しているが、
子午面環流(~10 m/s)は観測装置によって結果が違う(原因不明)
→少なくとも差動回転は正しく再現する必要あり

太陽差動回転理論の問題

太陽自転速度では、赤道を速くできない：Convective conundrum



高解像度化(=現実に近づける)は問題を悪くするセンス

研究の目的

富岳を用いた高解像度計算で熱対流の難問(Convective conundrum)を解決する。

つまり高解像度化で、赤道が速く自転するという実際の太陽の差動回転分布を目指す。

どこまで解像度を上げればいいのかわからないが、富岳でできるところまでを目指す。

太陽内部計算で解く方程式系

磁気流体力学の方程式
Magnetohydrodynamics (MHD)

$$\frac{\partial \rho}{\partial t} = -\nabla \cdot (\rho \boldsymbol{v})$$ **質量保存則**

重力

$$\frac{\partial}{\partial t}(\rho \boldsymbol{v}) = -\nabla \cdot (\rho \boldsymbol{v}\boldsymbol{v}) - \nabla p + \rho \boldsymbol{g}$$ **運動方程式**

ローレンツ力 $+\frac{1}{4\pi}(\nabla \times \boldsymbol{B}) \times \boldsymbol{B}$ $+ 2\rho \boldsymbol{v} \times \boldsymbol{\Omega}$ **コリオリ力**

$$\rho T \frac{\partial s}{\partial t} = -\rho T (\boldsymbol{v} \cdot \nabla) s + Q_{\mathrm{rad}}$$ **エントロピーの方程式**
or エネルギー保存則

輻射

$$\frac{\partial \boldsymbol{B}}{\partial t} = \nabla \times (\boldsymbol{v} \times \boldsymbol{B})$$ **磁場の誘導方程式**

$$p = p(\rho, s)$$ **状態方程式**(電離の効果を含めるOPAL)

これらの方程式を格子法で解く

コード開発

R2D2コード(Hotta+2019, Hotta+2020)を富岳用にチューニング
京では実行効率20%超程度。

音速抑制法(Hotta+2012)、アルフベン速度抑制(Rempel+2009)などを使って全体通信のないアルゴリズム

京で使っていたコードをそのまま富岳で実行すると実行効率4.5%程度。ある程度のチューニングが必要と考えられる。

$$\frac{\partial \rho_1}{\partial t} = -\frac{1}{\xi^2}\nabla \cdot (\rho \boldsymbol{v})$$

富岳の特徴

値はノードあたり
性能　京：128 GFLOPS/node、A64FX：3 TFLOPS/node

	L1$	L1$ BW	L1$ B/F	L2$	L2$ BW	L2$ B/F
京	256 KB	512 GB/s	4	5 MB	256 GB/s	2
A64FX	3 MB	12 TB/s	4	32 MB	4 TB/s	1.2

また、A64FXは京に比べてレジスタが少なくキャッシュのレイテンシも多い。(数値は公式資料で確認できなかったが、牧野さん資料参照)

富岳のチューニング方針

1. キャッシュのレイテンシを隠蔽するソフトウェアパイプラインを促進する→レジスタ少ない→最内ループの中の演算量を減らす→ループ分割

2. 低いキャッシュ幅をごまかすためにレジスタを有効に使う→最内ループになるべく多くの物理量を登場させる→ループ内の演算を増やす

富岳におけるMHD計算の困難

1. 変数が多く、色々な演算に登場
   密度、内部エネルギー、速度三成分、磁場三成分、圧力＝8+1成分
   ループ分割をするとデータ非局所化の影響が大きい

2. 解像度が重要なので、ステンシルを大きくして多くの要素を参照するがある。R2D2コードでは空間微分は４次の中央差分を利用

$$\left(\frac{\partial q}{\partial x}\right)_i = \frac{-q_{i+2} + 8q_{i+1} - 8q_{i-1} + q_{i-2}}{12\Delta x}$$

   L2キャッシュにも3次元データは乗らないのでナイーブにコードを書くと、一度読み込んだデータを有効に活用できない

ループ分割とデータ局在化両立のアイディア

$$-\nabla \cdot [(\rho_0 + \rho_1)\, \boldsymbol{v}]$$

$$\left(\frac{\partial q}{\partial x}\right)_i = \frac{-q_{i+2} + 8q_{i+1} - 8q_{i-1} + q_{i-2}}{12\Delta x}$$

微分の各要素を一つのループにまとめる。→結構速くなる

```
dro(i) = - (&
     & + rkx1(i)*(ro0(i+i2) + qqp(1,i+i2,j,k))*qqp(2,i+i2,j,k) &
     & + rkx2(i)*(ro0(i+i1) + qqp(1,i+i1,j,k))*qqp(2,i+i1,j,k) &
     & + rkx3(i)*(ro0(i-i1) + qqp(1,i-i1,j,k))*qqp(2,i-i1,j,k) &
     & + rkx4(i)*(ro0(i-i2) + qqp(1,i-i2,j,k))*qqp(2,i-i2,j,k) &
     & + rky1(j)*(ro0(i   ) + qqp(1,i,j+j2,k))*qqp(3,i,j+j2,k) &
     & + rky2(j)*(ro0(i   ) + qqp(1,i,j+j1,k))*qqp(3,i,j+j1,k) &
     & + rky3(j)*(ro0(i   ) + qqp(1,i,j-j1,k))*qqp(3,i,j-j1,k) &
     & + rky4(j)*(ro0(i   ) + qqp(1,i,j-j2,k))*qqp(3,i,j-j2,k) &
     & + rkz1(k)*(ro0(i   ) + qqp(1,i,j,k+k2))*qqp(4,i,j,k+k2) &
     & + rkz2(k)*(ro0(i   ) + qqp(1,i,j,k+k1))*qqp(4,i,j,k+k1) &
     & + rkz3(k)*(ro0(i   ) + qqp(1,i,j,k-k1))*qqp(4,i,j,k-k1) &
     & + rkz4(k)*(ro0(i   ) + qqp(1,i,j,k-k2))*qqp(4,i,j,k-k2) &
     & )
```

```
! y derivative
!OCL FULLUNROLL_PRE_SIMD
do l = 1,2*marginy
   jp = ipa(l)
   !OCL LOOP_NOFUSION
   do i = 1+marginx,nxg-marginx
      dro(i) = dro(i) - rky(j,jp)*qqp(i,j+jp,k,mtype2)*qqp(i,j+jp,k,3) !equation of continuity

      drx(i) = drx(i) + rky(j,jp)*( &
           & - qqp(i,j+jp,k,mtype2)*qqp(i,j+jp,k,2)*qqp(i,j+jp,k,3) &
           & + qqp(i,j+jp,k,5)*qqp(i,j+jp,k,6)*cafac(i,j,k) & ! magnetic tension
           & )
```

ループ分割とデータ局在化

ナイーブにループ分割しても、コードが遅くなることがほとんどだった。MHDには色々な演算があるので、分割・融合を繰り返して、最適な演算順序を考える。

ループ一重化

```
do k = 1,nzg
do j = 1,nyg
```

```
do jk = 0,nyzg - 1
   k = jk/nyg
   j = jk - nyg*k

   j = j + 1
   k = k + 1
```

12 openMP並列が基本なので、最外ループの並列化のみでは足りないことが多い。

ループ一重化で並列化効率をアップ。

配列の構造(旧バージョンでの計測)

配列の要素の並び順は、チューニングの大きな要素。
四次元配列については全て試した

	(m,i,j,k)	(i,m,j,k)	(i,j,m,k)	(i,j,k,m)
実行効率	7.3%	8.2%	8.8%	8.9%
実行時間	52.4秒	46.3秒	43.8秒	43.2秒

(i,j,k,m)が一番速かった。京の時は、(m,i,j,k)だった。この前の説明会で聞いた話によると京の時は、このようにしても速くする仕組みがあったとのこと
説明会では(i,j,m,k)が一番速いと言っていた

二次元の一時配列についても(m,i)とするより(i,m)とした方が早い

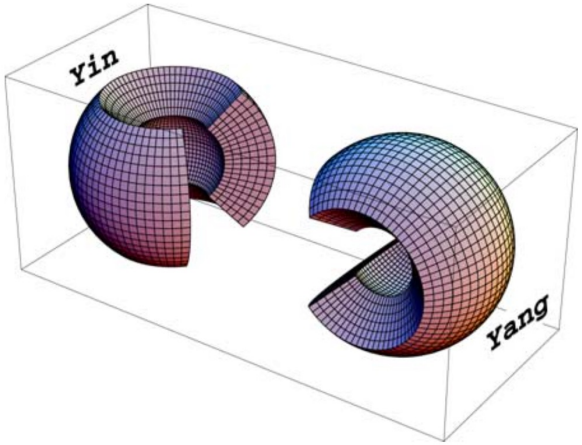
チューニング結果

MPI Processあたり128×64×512 100 step

	チューニング前(実行時間)	チューニング前(実行効率)	チューニング後(実行時間)	チューニング後(実行効率)
全体	110.1 秒	4.57%	44.5 秒	9.3%
Runge-Kutta ループ	32.9 秒	4.87%	18.9 秒	8.2%
人工粘性	68.0 秒	4.26%	19.9 秒	10.5%
輻射輸送	5.27 秒	7.68%	4.2 秒	8.4%

3.8×10^7 cell update/sec/nodeを達成
4 step Runge-Kutta使っていてCFL numberが2くらいまでいけるので、
時間2次精度のMHDスキームと比較するとさらに2倍くらい速い
(と思ってもらって良い)

MPIチューニング



Tofuトポロジーでの MPIスレッドの
配置を最適化する。

```
#PJM -L ”node=3x4x2"
```

```
include "mpif-ext.h"
call fjmpi_topology_get_dimension(fjmpi_dim,merr)
call fjmpi_topology_get_shape(ix0,jx0,kx0,merr)
```

ノードがどこにあるかしか確かめることができない。
CMGがどこにあるかわわからない。
富岳では1ノードにつき4MPIスレッド使うので、y方向に展開する。
Yin-Yang gridでは、3次元構造をy方向に二つに割って、YinとYangを割り当てているこのようにすれば少なくとも2次元内で相手を探せるので際立った性能の劣化はない(最適ではない)

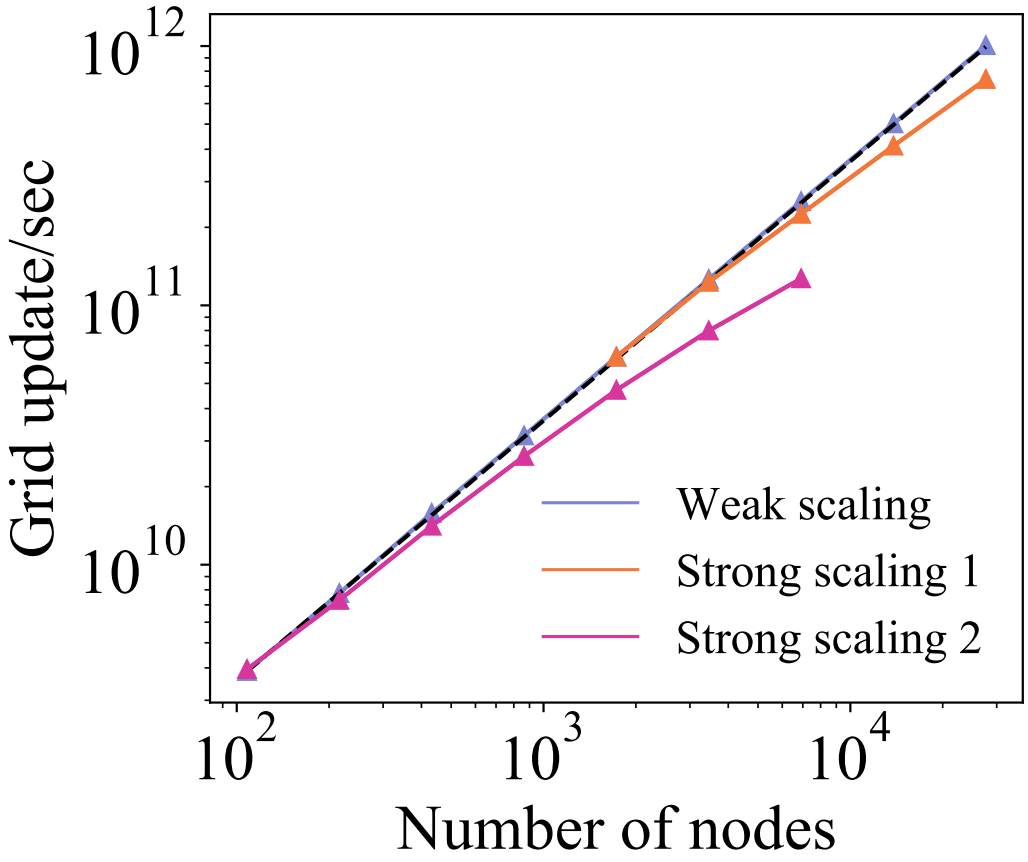
チューニング結果

MPI Processあたり128×64×512 100 step

	チューニング前(実行時間)	チューニング前(実行効率)	チューニング後(実行時間)	チューニング後(実行効率)
全体	110.1 秒	4.57%	44.5 秒	9.3%
Runge-Kutta ループ	32.9 秒	4.87%	18.9 秒	8.2%
人工粘性	68.0 秒	4.26%	19.9 秒	10.5%
輻射輸送	5.27 秒	7.68%	4.2 秒	8.4%

3.8×10^7 cell update/sec/nodeを達成
4 step Runge-Kutta使っていてCFL numberが2くらいまでいけるので、
時間2次精度のMHDスキームと比較するとさらに2倍くらい速い
(と思ってもらって良い)

スケーリング

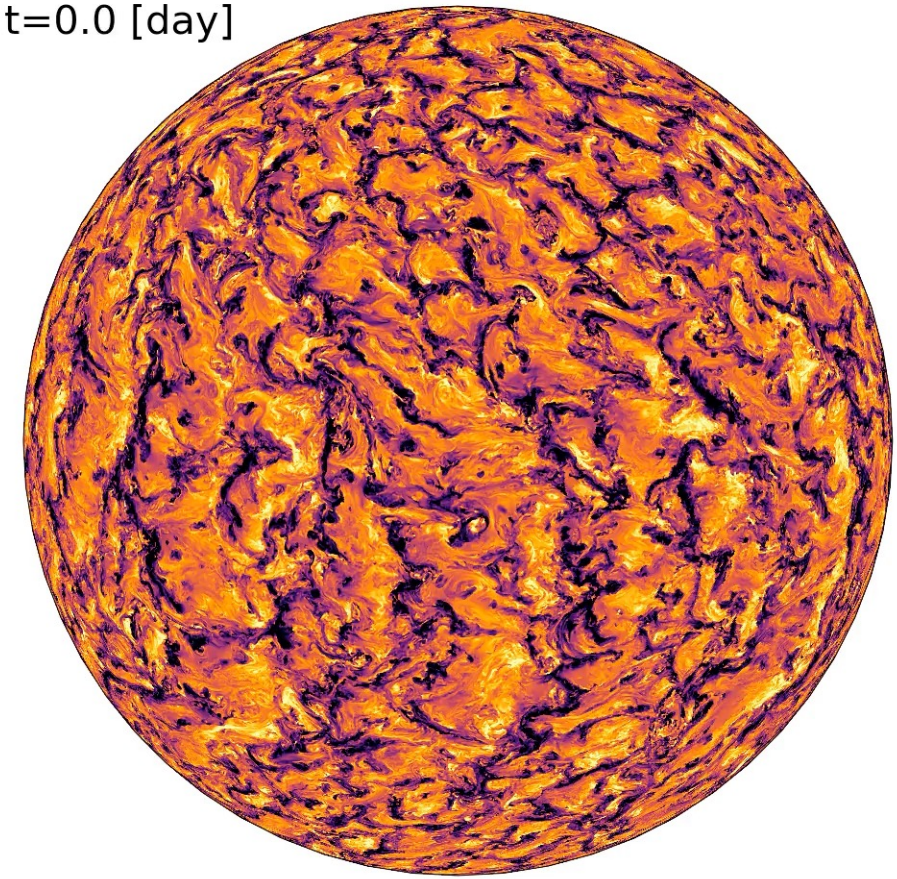


100万超えてもweak scalingならば、99%を超える並列化効率を示している。大規模計算も滞りなくできそう

細かい富岳利用Tips

- 言語関連マニュアル・言語使用手引書をよく読む
- 不老などと一緒に使うのが吉
  - ✓ 384node=18432core以下混んでる？
- on the fly解析推奨
  - ✓ 大規模計算は状況を知るだけでも一苦労
- ストレージ圧倒的に足りない。1ケース100TBくらい利用

富岳でおこなった計算設定



$r = 0.9R_\odot$での動径方向速度

計算領域：$0.71R_\odot < r < 0.96R_\odot$

格子点数：

Low	96×768×1536
Middle	192×1536×3072
High	384×3072×6144

計算時間：4000日
Highの場合350万ステップほど

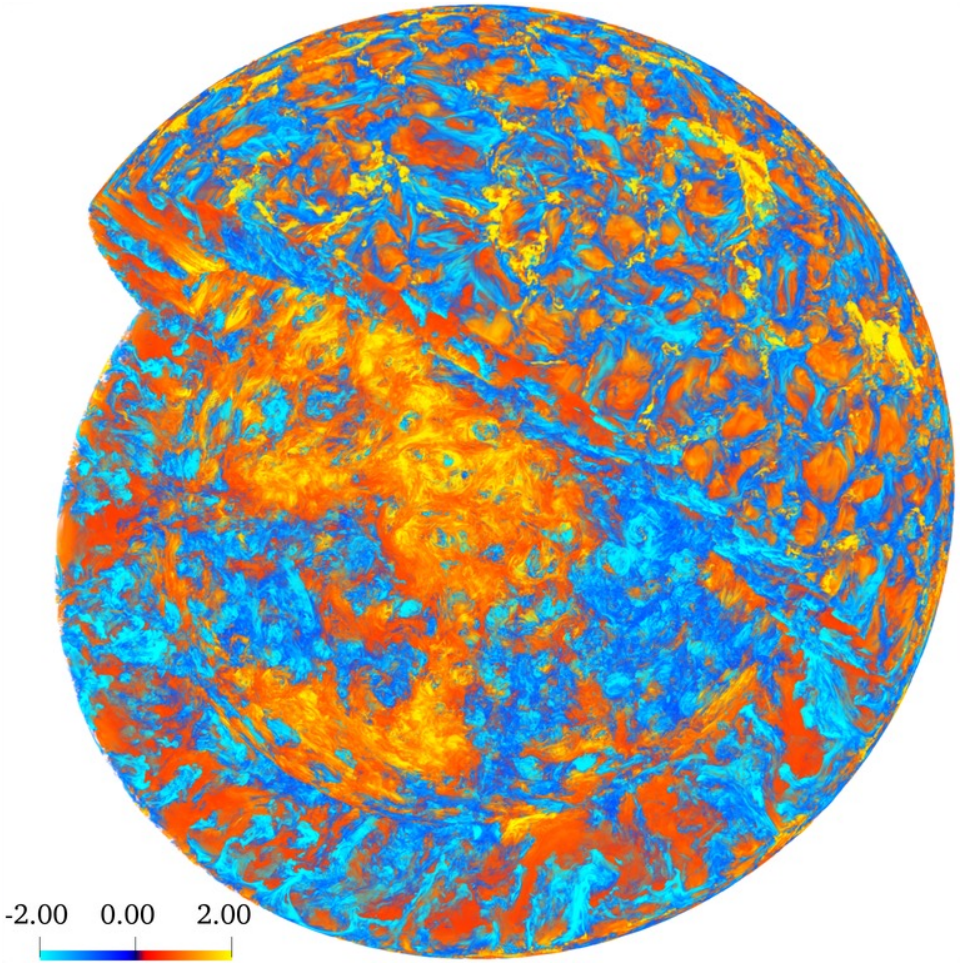
3000日ほどでほぼ統計的定常に遷移

3600-4000日でのデータを平均して
結果を示す

熱対流と磁場

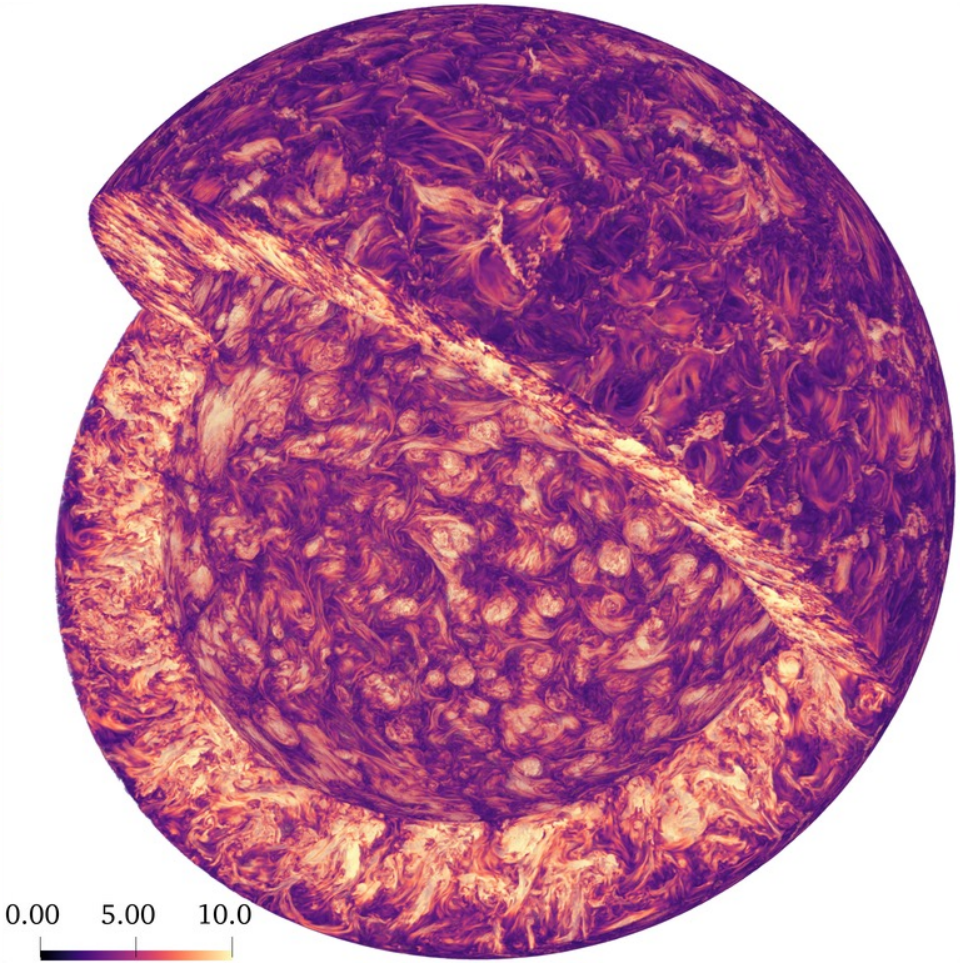
Normalized entropy

$(s - \langle s \rangle)/s_{\mathrm{rms}}$



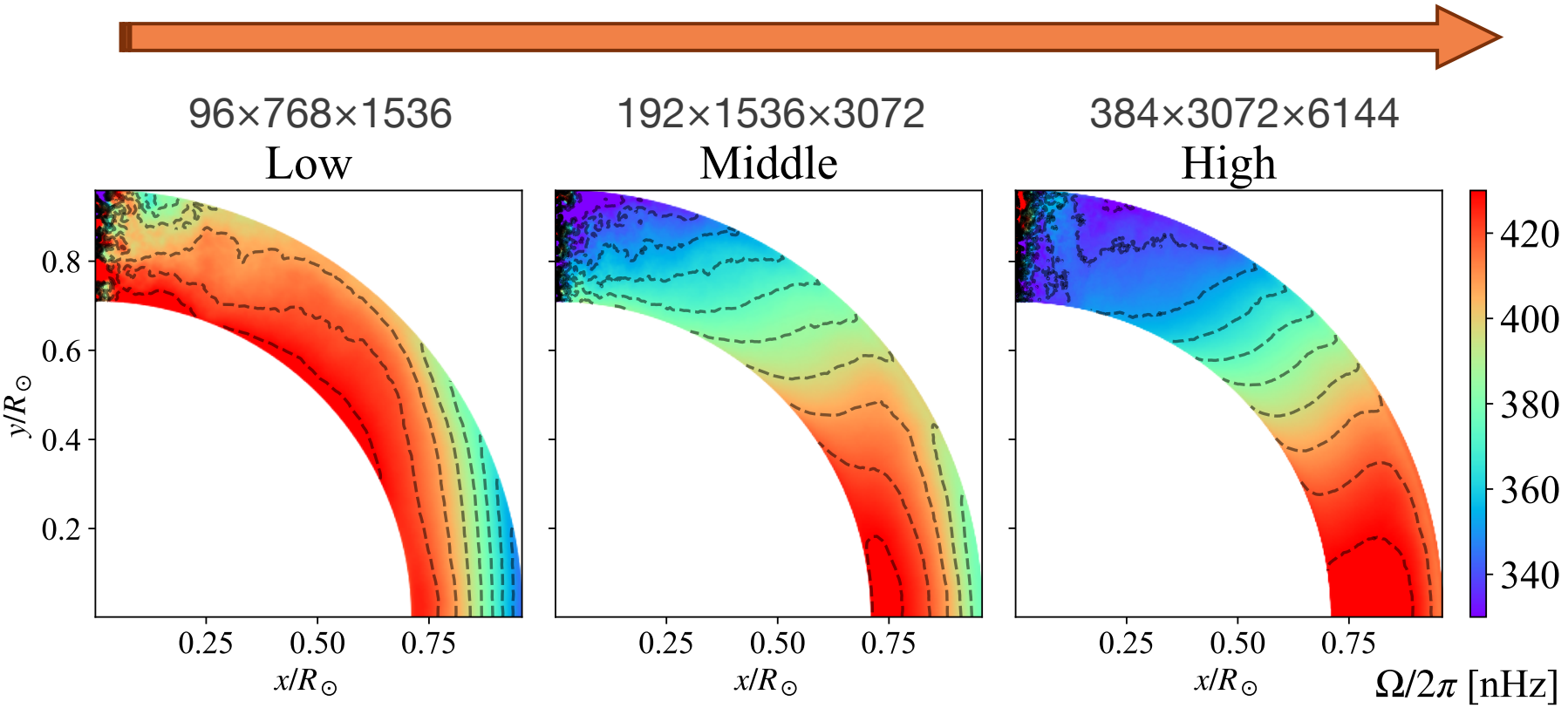
Magnetic field strength

$|B|$ [kG]



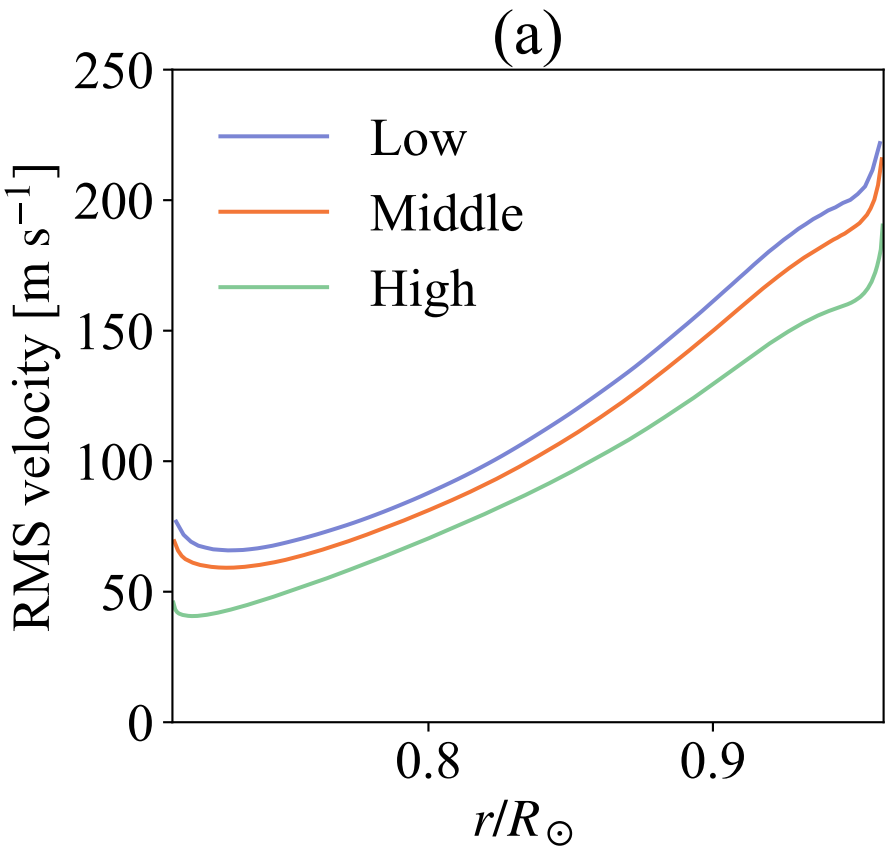
差動回転の角速度依存性

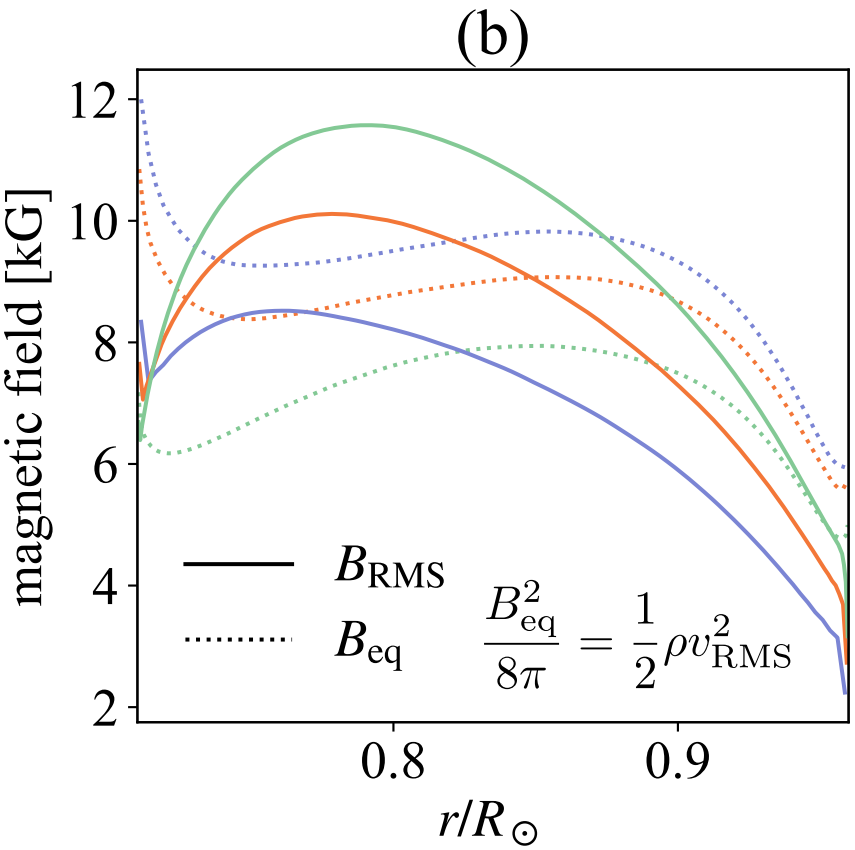
解像度アップ



Low caseでは、これまでの計算と同じく太陽回転角速度だと赤道よりも極が速い差動回転が実現してしまっているが、High caseではほぼ太陽と同じ差動回転を実現することができている。

熱対流速度とダイナモ生成磁場



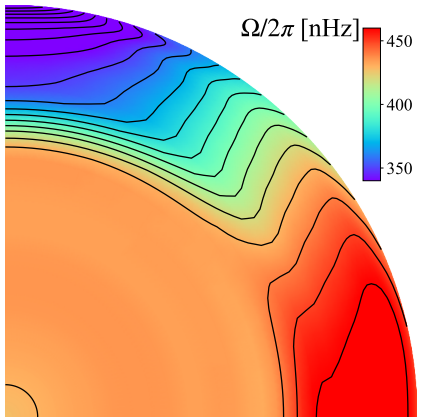


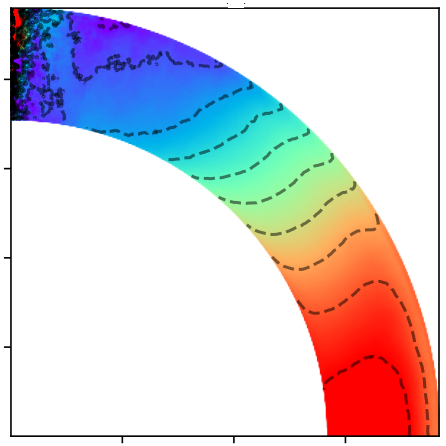
解像度が高くなると、熱対流速度は遅くなり、磁場は強くなる。
Low caseの時は、対流層内部の全ての場所で$E_{\rm kin} > E_{\rm mag}$だが、High caseの場合は対流層内部全てで$E_{\rm kin} < E_{\rm mag}$となり状況が全く変わっている。
詳しくは説明しないが、磁場が角運動量を運ぶことがエッセンス

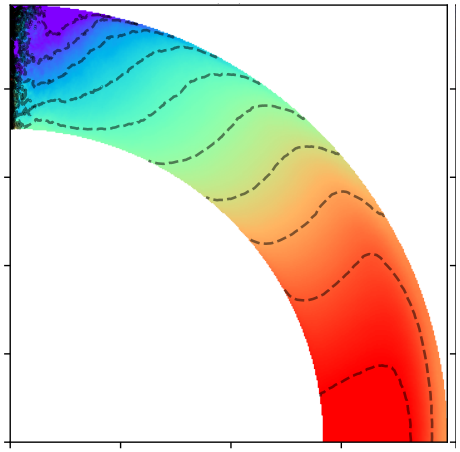
さらに高解像度へ

実際の太陽　Hotta & Kusano　新しい計算

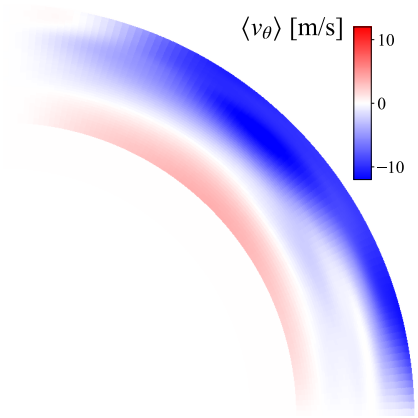
差動回転

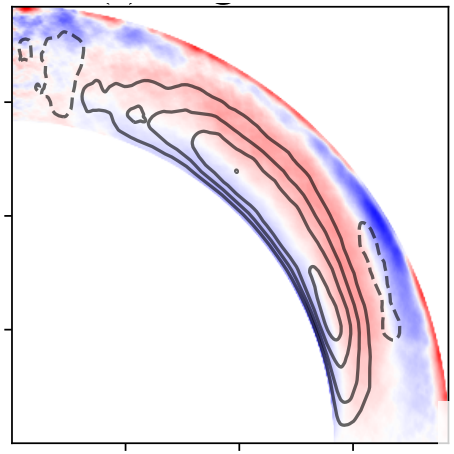


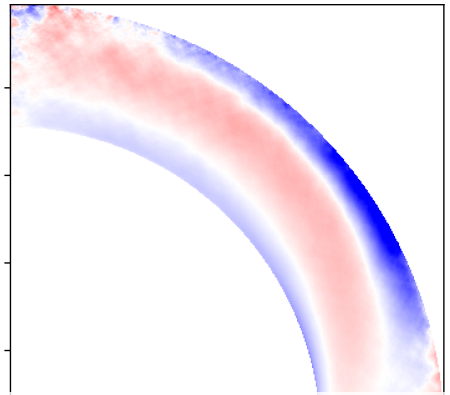




子午面環流







太陽表面近くの構造を再現できていない。
(子午面環流は深い場所では感度なし)

$r_{\max} = 0.96R_\odot$, 384×3072×6144
350万時間ステップ→
$r_{\max} = 0.99R_\odot$, 512×4096×8192
1200万時間ステップ

まとめ

- 太陽内部の構造はよくわかっているが、数値計算では太陽の差動回転を再現できないという「熱対流の難問」が長らく課題となっていた。
- 富岳を用いた大規模計算で解決を試みた
- MHD計算を富岳で行うとき、複雑な方程式系・多い変数種類が要因で、ループ分割・データ局所化が困難

- 色々な試みの結果、チューニングはある程度の成果をみせ、富岳による大規模計算が可能に

- これまでにない大規模計算で赤道が速い差動回転を実現。「熱対流の難問」解決

- 高解像度化はまだまだ価値があり、高解像度化によりさらに観測される太陽の特徴を再現できるフェーズに入ってきている。