M27c

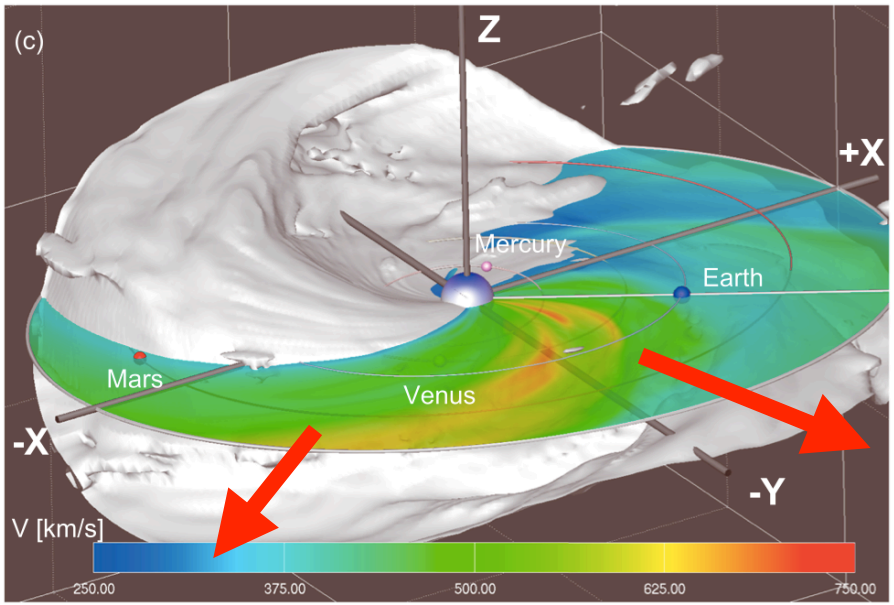
超並列磁気流体シミュレーションコード OpenMHD-GPU の開発

銭谷誠司（神戸大学）

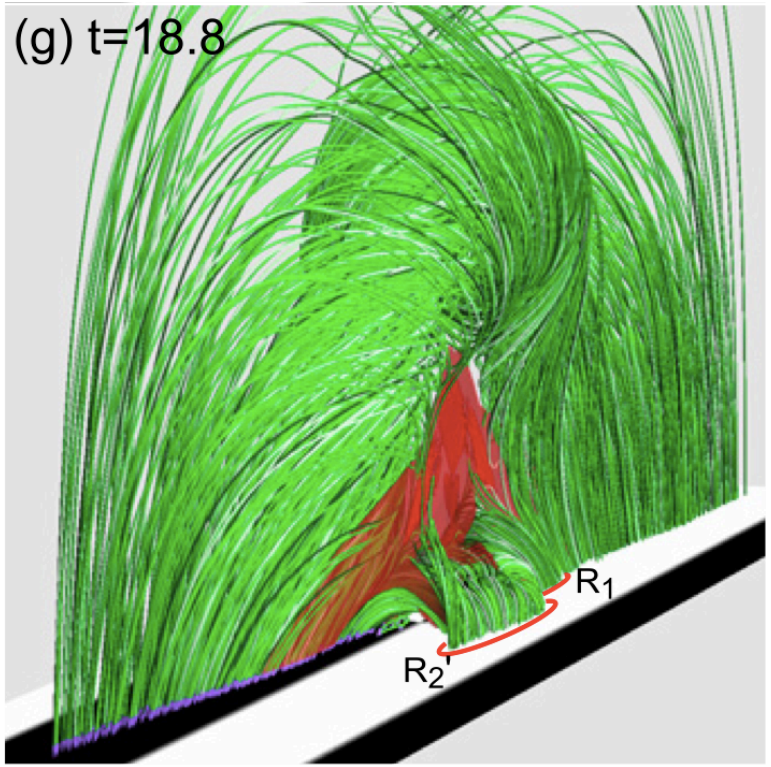
- 1. OpenMHD の概要
- 2. GPU 対応
- 3. クラウド利用
- 4. 利用実績

磁気流体（MHD）シミュレーション

- 太陽・宇宙プラズマ研究の必須ツール
- 高度なシミュレーション技法：日進月歩で改良が進む



Shiota et al. 2014 Space Weather



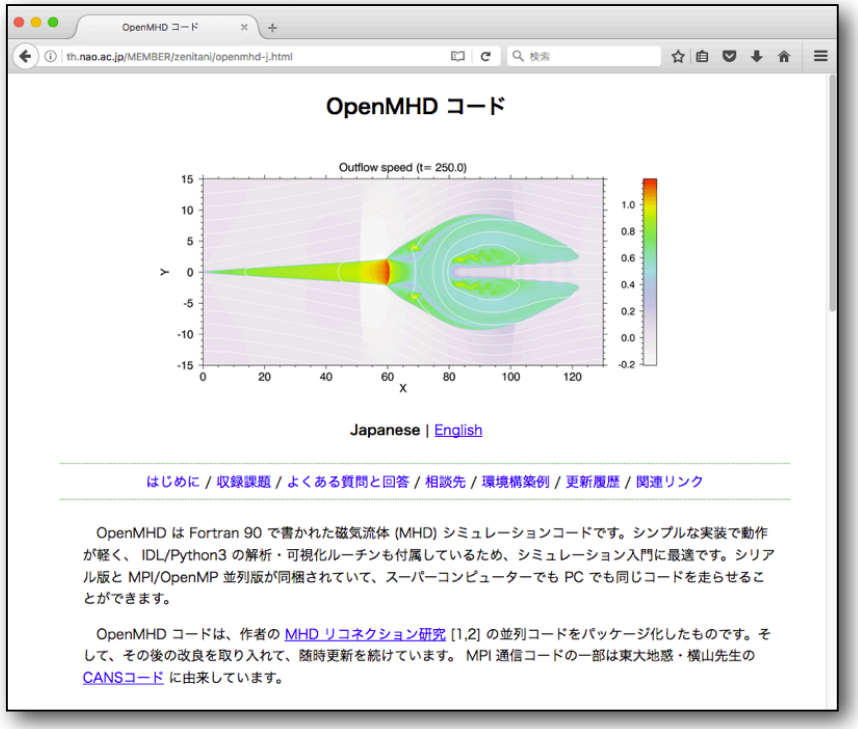
Kusano et al. 2012 ApJ

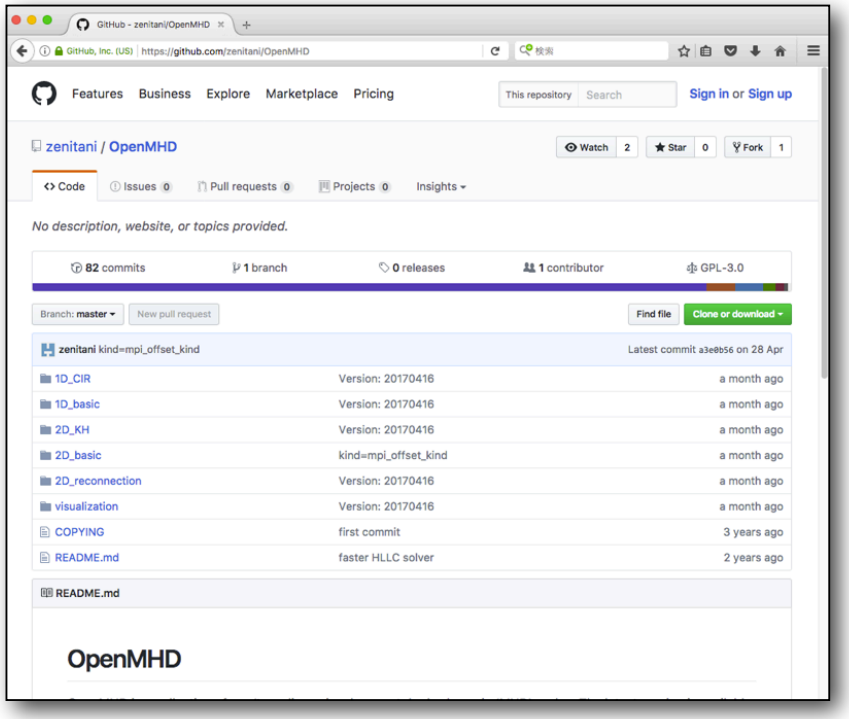
公開MHDコード

- CANS（2002頃〜）
  - https://www.kwasan.kyoto-u.ac.jp/~yokoyama/etc/cans/
  - 東京大学（現・京都大学）横山央明先生
  - Fortran 77, MPI, IDL可視化
- CANS+（2014〜）
  - http://www.astro.phys.s.chiba-u.ac.jp/cans/doc/
  - 千葉大学 松本洋介さんら
  - Fortran 90, MPI+OpenMP, IDL可視化
  - 千葉大サマースクール
- OpenMHD（2010〜）
  - https://sci.nao.ac.jp/MEMBER/zenitani/openmhd-j.html
  - Fortran 90, MPI+OpenMP & CUDA Fortran, MPI-GPU
  - Python, IDL可視化

OpenMHD

- 2010年から公開、技術詳細→次々頁
- 日本語・英語のオンラインドキュメント
- 開発中のコードも github で公開
- サポートメーリングリスト（41名参加）

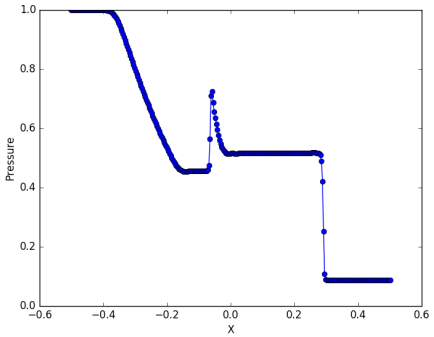




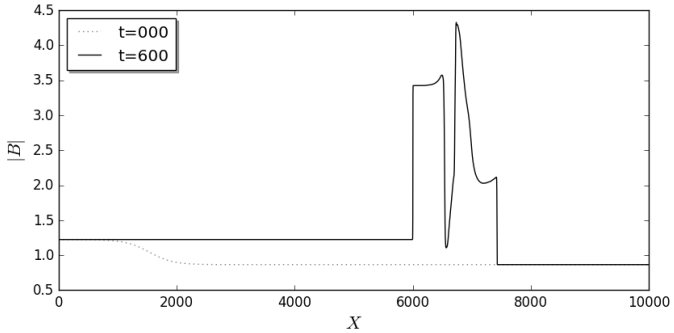
https://sci.nao.ac.jp/MEMBER/zenitani/openmhd-j.html

収録問題

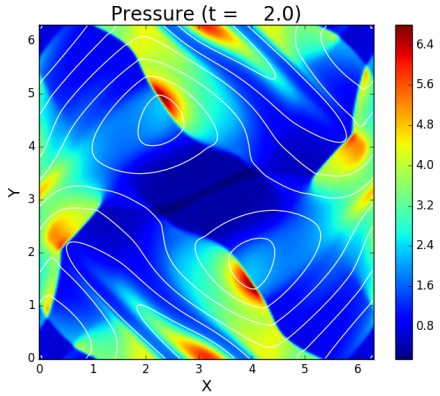
- リーマン問題



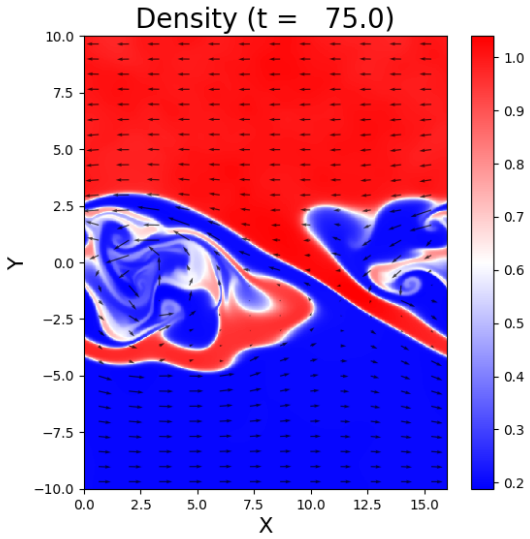
- 太陽風問題



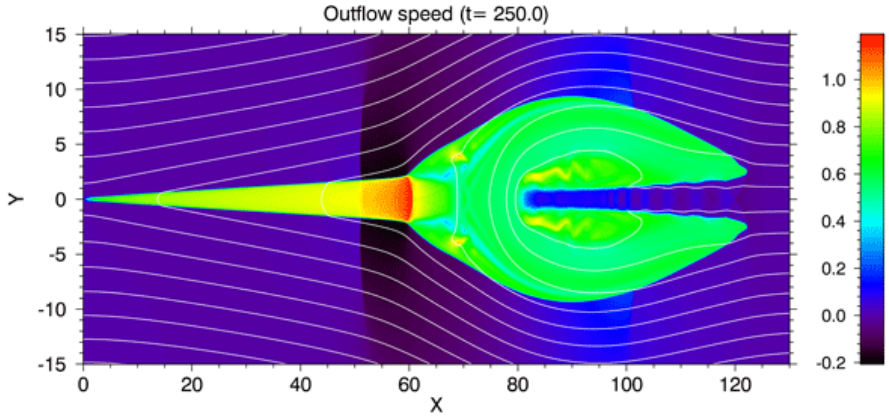
- Orszag-Tang 渦



- Kelvin-Helmholtz 不安定



- 磁気リコネクション



OpenMHD 技術詳細

- 数値解法
  - 時間：　2次精度SSP-RK法
  - 空間：　TVD法、2次MUSCL補間
  - 数値流束：　HLLD法（HLLC法を併用）
  - div-B 補正：Hyperbolic divergence cleaning
- 並列化
  - プロセス並列：MPI（MPI通信 or SHM通信）
  - スレッド並列：OpenMP or GPU (CUDA)
- 可視化ツール
  - Python (matplotlib, Jupyter Notebook対応)
  - IDL

現在の標準技術の組み合わせ

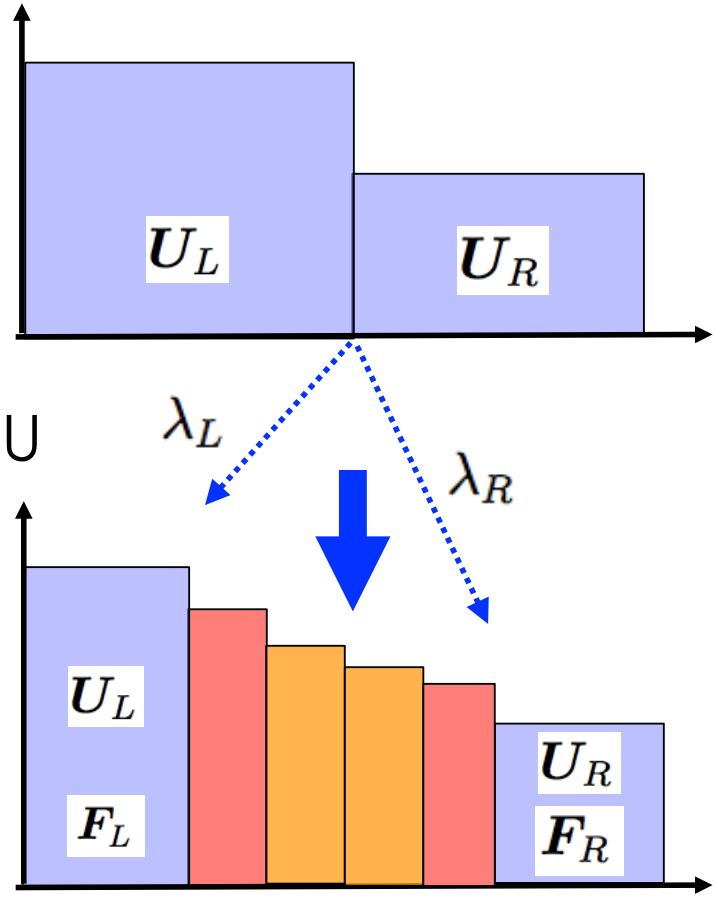
セールスポイント

いちはやく Python 3 を取り入れた

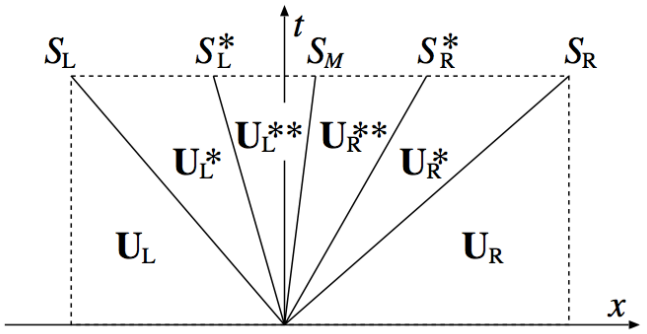
数値流束計算：HLLD法

- セル間の時間発展をリーマン問題で近似
- ４つの中間状態を想定（Miyoshi & Kusano 2005）

U



$$\mathbf{F}_{\mathrm{HLLD}} = \begin{cases} \mathbf{F}_{\mathrm{L}} & \text{if } S_{\mathrm{L}} > 0, \\ \mathbf{F}_{\mathrm{L}}^{*} & \text{if } S_{\mathrm{L}} \leqslant 0 \leqslant S_{\mathrm{L}}^{*}, \\ \mathbf{F}_{\mathrm{L}}^{**} & \text{if } S_{\mathrm{L}}^{*} \leqslant 0 \leqslant S_{M}, \\ \mathbf{F}_{\mathrm{R}}^{**} & \text{if } S_{M} \leqslant 0 \leqslant S_{\mathrm{R}}^{*}, \\ \mathbf{F}_{\mathrm{R}}^{*} & \text{if } S_{\mathrm{R}}^{*} \leqslant 0 \leqslant S_{\mathrm{R}}, \\ \mathbf{F}_{\mathrm{R}} & \text{if } S_{\mathrm{R}} < 0. \end{cases}$$

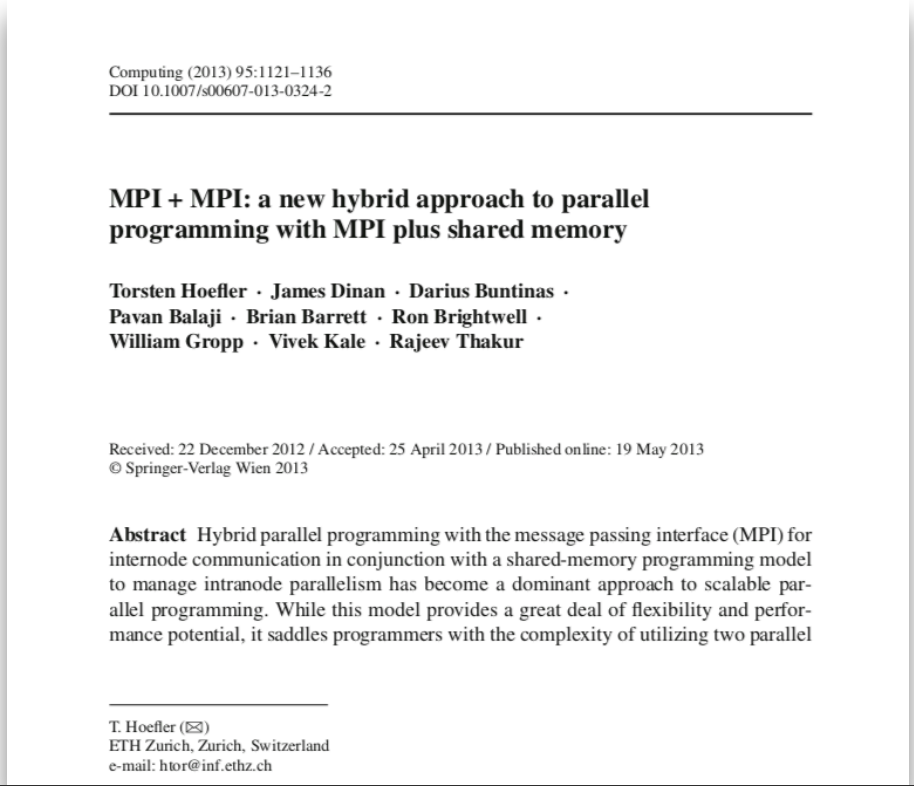


共有メモリ通信（SHM通信）

- MPI で
  - ノード外 → MPI 通信
  - ノード内 → 共有メモリ経由でデータをやりとり
  - MPI-3 shared memory window

- OpenMHDでの試験実装：
  - MPI 通信の方がやや高速
  - チューニング不足？

- GPU-MPI 時代に
  役に立つ可能性



Computing (2013) 95:1121–1136
DOI 10.1007/s00607-013-0324-2

MPI + MPI: a new hybrid approach to parallel programming with MPI plus shared memory

Torsten Hoefler · James Dinan · Darius Buntinas · Pavan Balaji · Brian Barrett · Ron Brightwell · William Gropp · Vivek Kale · Rajeev Thakur

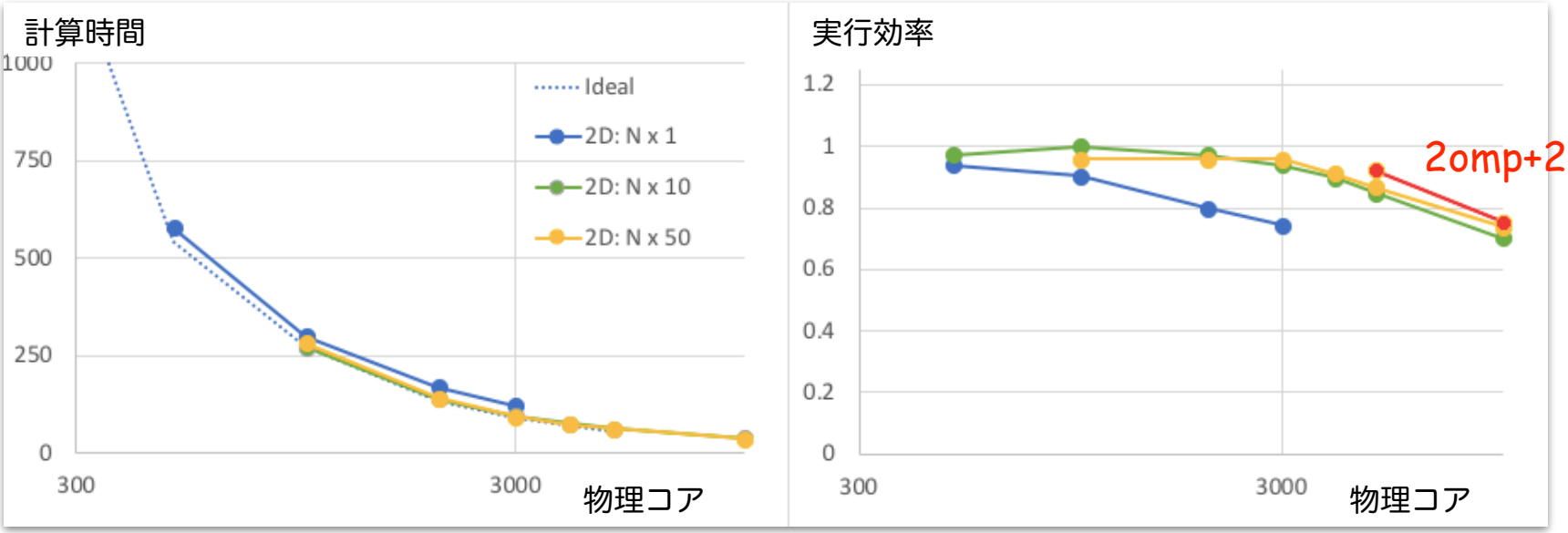
Received: 22 December 2012 / Accepted: 25 April 2013 / Published online: 19 May 2013

Abstract Hybrid parallel programming with the message passing interface (MPI) for internode communication in conjunction with a shared-memory programming model to manage intranode parallelism has become a dominant approach to scalable parallel programming. While this model provides a great deal of flexibility and performance potential, it saddles programmers with the complexity of utilizing two parallel

T. Hoefler (✉)
ETH Zurich, Zurich, Switzerland
e-mail: htor@inf.ethz.ch

超並列性能（Strong scaling）

- 多次元領域分割とノンブロッキング通信を使い
  少なくとも 5000 コアまで良好な並列性能
- ハイパースレッディング ON + OpenMP 2 スレッドで最速
  （京大KDKコンピューターの特殊事情）

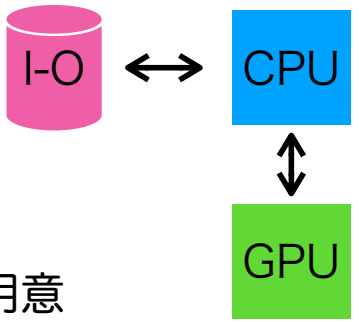


OpenMHD/2018

- スパコンGPU化の流れ
  - 電力性能的にも有利
  - TOP 500 スパコンの上位 8/10、全体の 342/500 が GPU を活用
- プラズマシミュレーション分野でのGPU利用
  - 2010 杉山さん（JAMSTEC） GPU PICコードの開発
  - 2014 齊藤慎司さん（名大） GPU, MIC を利用したPICコード
  - 2014 淵田さん・近藤さん（愛媛大） GPU-CPU 連結 MHDコード
  - 2020 銭谷ら（神戸大） OpenMHD-GPU を公開
- OpenMHD-GPU
  - CUDA Fortran を用いて NVIDIA GPU に完全対応
  - GPU-MPI にも対応
  - ⇒ OpenMHD (2020) に収録・公開中

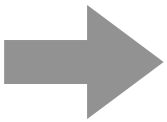
CUDA Fortran による GPU 移植（1/3）

- 移植方針
  - I-O 以外の主要計算ループを全て GPU 側に移動
- 移植手順
  - 1. ホスト (PC) 用・デバイス (GPU) 用の変数をそれぞれ用意
  - 2. サブルーチンにホスト・デバイス用の修飾子を追加
  - 3. デバイス用サブルーチンへの移植。
    ループの中身がデバイスルーチンになると思えば良い




```
call limiter(...)
```

```
subroutine limiter(...)
            ...
  do j=1,jx
    do i=1,ix
      ...
    enddo
  enddo
```



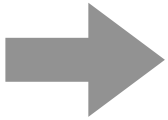
```
call <<<dim3,dim3>>>limiter(...)
```

```
attributes(global) &
subroutine limiter(...)
            ...
  j = (blockIdx%y-1)*blockDim%y + threadIdx%y
  i = (blockIdx%x-1)*blockDim%x + threadIdx%x
  if( (1<=j).and.(j<=jx) ) then
    if( (1<=i).and.(i<=ix) ) then
      …
    endif
  endif
```

CUDA Fortran による GPU 移植（2/3）

- 開発上の難点：
- 1) デバイス配列の部分指定（:）
  - コンパイルは通るが実行時に異常動作
  - 明示的にループを書いてディレクティブでGPUカーネル化

```
! left/right
  U(ix,:,:) = U(2,:,:)
  U(1,:,:)  = U(ix-1,:,:)
```



```
!$cuf kernel do(2) <<<*, *>>>
  do k=1,var1
     do j=1,jx
        U(ix,j,k) = U(2,j,k)
        U(1,j,k)  = U(ix-1,j,k)
     enddo
  enddo
```

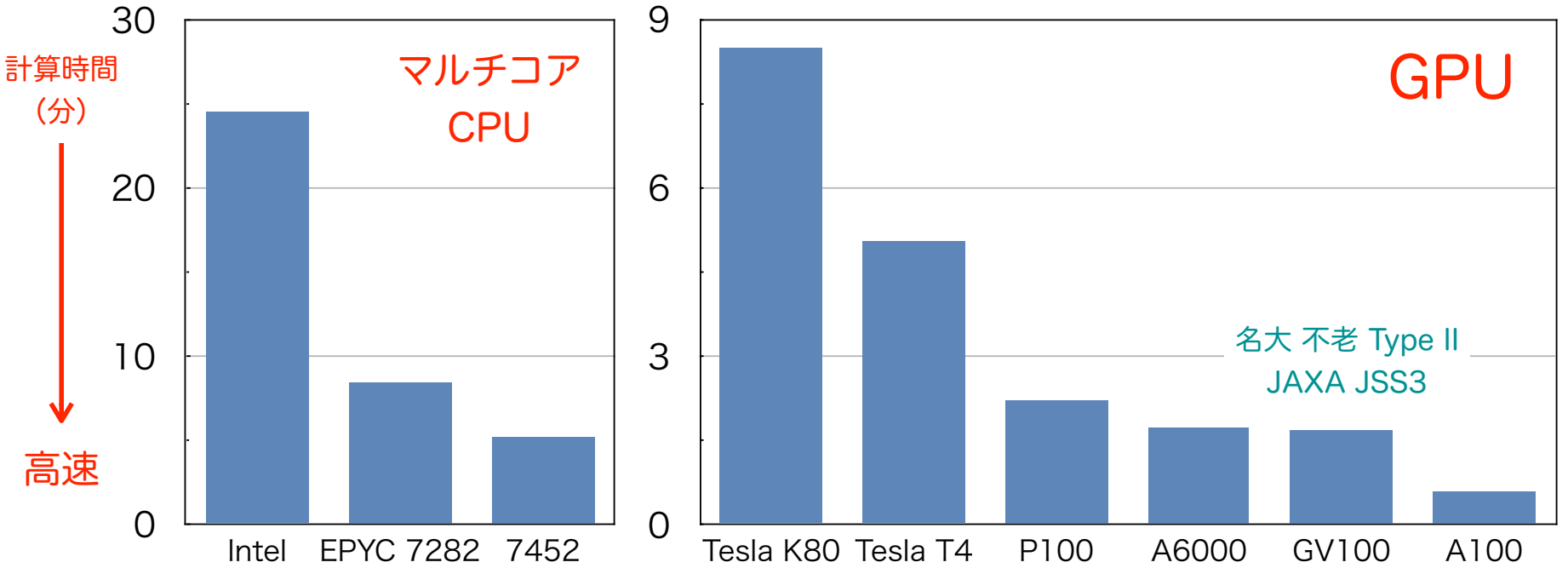
- 2) 非同期処理の混在
  - デバイス変数のパッキング処理と（ノンブロッキング）MPI 通信など、非同期処理機構が並走すると想定外のデータ混入が起きる
  - cudaDeviceSync, cudaStreamSync で明示的に同期
- 3) 古い情報？
  - デバイス変数・サブルーチンを module にまとめる必要はない
- 4) 開発ツールの変化
  - 新環境で旧ツール（プロファイラ）が利用できない

CUDA Fortran による GPU 移植（3/3）

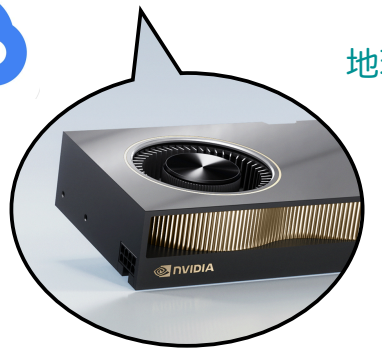
- 参考資料
  - NEC／九州大学・筑波大学の GPU講習会資料
    - https://www.cc.kyushu-u.ac.jp/scp/doc/users/forum/forum20100929/1.pdf
    - https://www.ccs.tsukuba.ac.jp/wp-content/uploads/sites/14/2012/12/gpulec4.pdf
  - NVIDIA CUDA Fortran Programming Guide
    - https://docs.nvidia.com/hpc-sdk/compilers/cuda-fortran-prog-guide/
  - CUDA Fortran for Scientists and Engineers【書籍】
    - https://www.amazon.co.jp/dp/0124169708

ベンチマーク: 良好なノード性能

- 基準CPU (Intel i9) 1コアで70-75分の２次元リコネクション計算

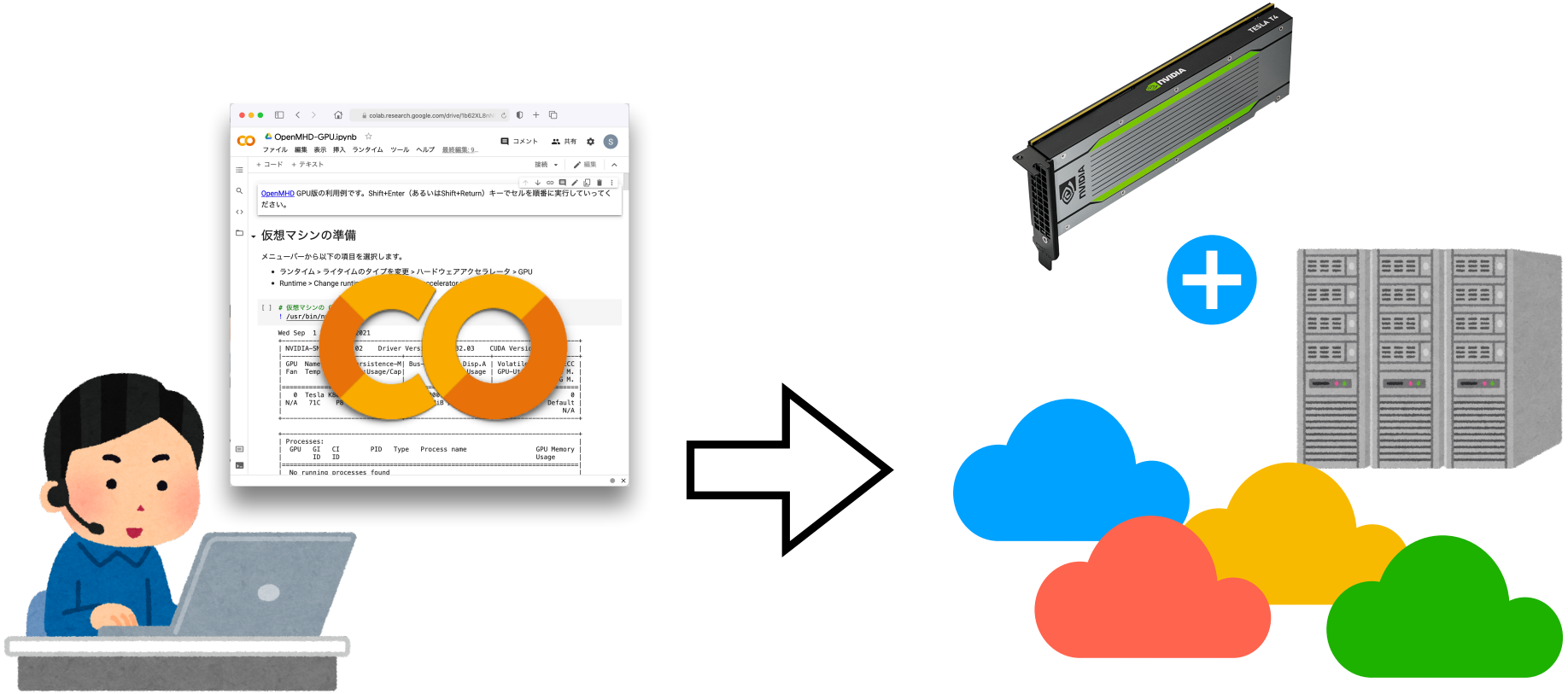


- Intel i9-9900K 8コア
- AMD EPYC 7282 16コア
- AMD EPYC 7452 32コア



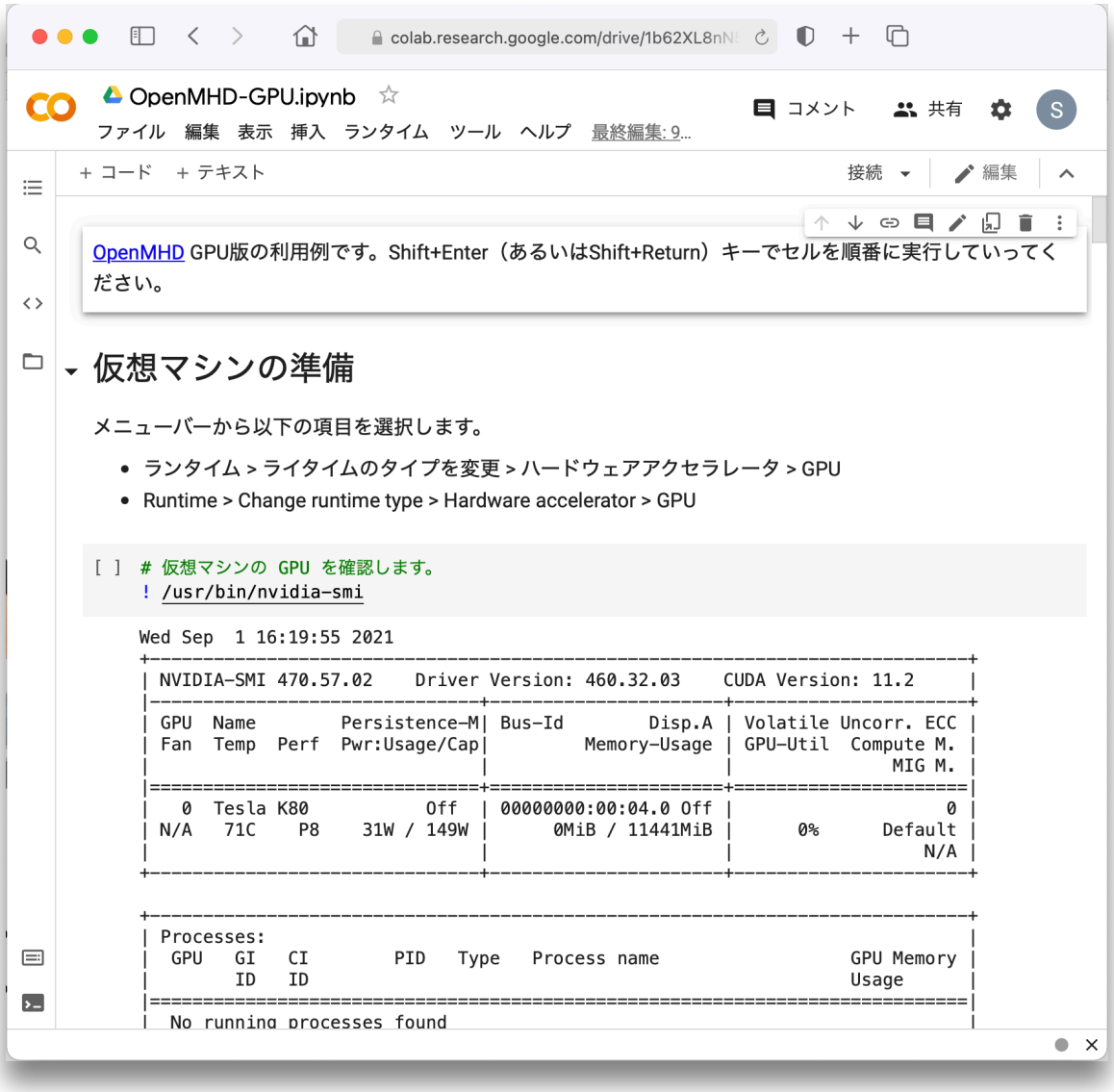
JAMSTEC
地球シミュレータ

OpenMHDのクラウド利用：Google Colaboratory



- ブラウザで操作
- Jupyter ノートブック
- Googleクラウド上の仮想マシン
- GPUインスタンスも利用可

Google Colaboratory in action

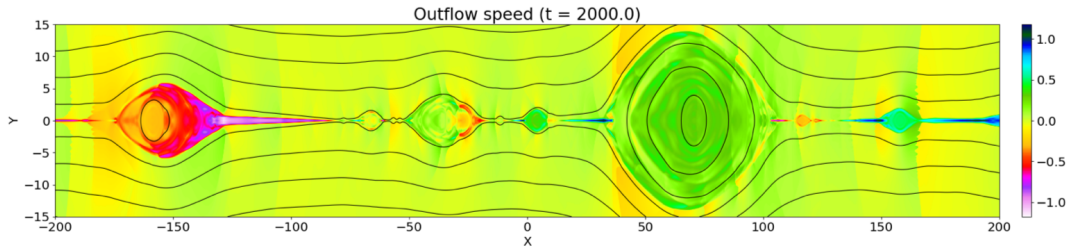


https://colab.research.google.com/drive/1b62XL8nN5W7oxPTCiYPH8M265JQ5gZla?usp=sharing

OpenMHD の利用実績

- 査読論文：12本
  - 銭谷筆頭 (3)、銭谷共著 (5)、独立論文 (4)
- 修士論文・卒業論文：3本
  - 神戸大学 (3)

- 磁気リコネクション問題

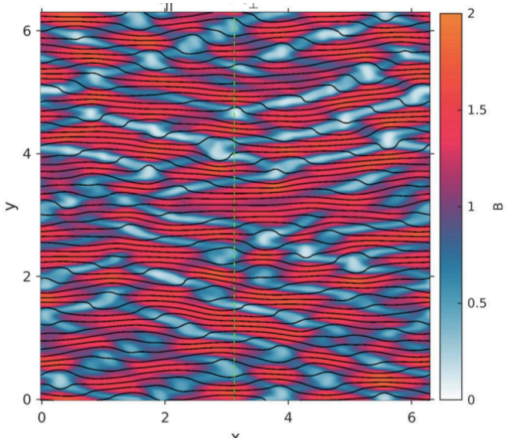


SZ & Miyoshi 2020 ApJL

M21a 山本さん発表

M24b 渡邊さん発表

- Double-polytropicプラズマ中のミラーモード揺らぎ構造



Teh & SZ 2019, 2020 ApJ

まとめ

- 1. OpenMHD の紹介
- 2. GPU対応
  - CUDA Fortranによるコード移植
  - 良好なノード性能
- 3. クラウド対応
  - Google Colaboratory
- 4. 利用実績
  - 査読論文 12本・卒業論文・修士論文3本
- URL
  - https://sci.nao.ac.jp/MEMBER/zenitani/openmhd-j.html