SWQU Workshop Day 3:
Solar Wind Generator (SWiG)

Ronald M. Caplan, Miko Stulajter, Jon Linker,
Cooper Downs, Nick Arge, Shaela I. Jones, Carl Henney,
Lisa Upton, Raphael Attié, and Bibhuti Jha

Predictive Science Inc.

NASA NSF

SWiG

Outline

- SWiG Overview:
  - Models
  - Numerical Methods
  - Code implementations
- BREAK
- How to run SWiG
- How to Install SWiG
  - Mac (homebrew/macports)
  - Windows (10 or 11 with WSL)
  - Linux
- Assignment
- BREAK

Re-cap of Empirical Solar Wind Model

Wang-Sheeley-Arge (WSA):

$$V = V_0 + \frac{V_m}{(1+f_{exp})^{C_1}}\left(1 - C_2 \exp\left[-\left(\frac{\Theta_B}{C_3}\right)^{C_4}\right]\right)^{C_5}$$

V_0	V_m	C_1	C_2	C_3	C_4	C_5
286 m/s	625 m/s	2/9	0.8	1	2	3

Expansion Factor at Rss:

$$f_{exp} = (R_\odot / R_{ss})^2 (B(R_\odot)/B(R_{ss}))$$

Distance to open field boundaries at r0: Θ_B

(both of the quantities are traced out to r1)

Density and temperature: $\rho = \rho_{\mathrm{f}} \left(\frac{\max(V)}{V}\right)^2$ $t = t_{\mathrm{f}}\,\frac{\rho_{\mathrm{f}}}{\rho}$

$t_{\mathrm{f}} = 1.85e6\,\mathrm{K}$

$\rho_{\mathrm{f}} = 152\,\mathrm{cm}^{-3}$

Re-cap of PFSS+CS Model for Coronal Magnetic Field

Maxwell equation: $\nabla \times \vec{B} = \mu_0 \vec{J}$

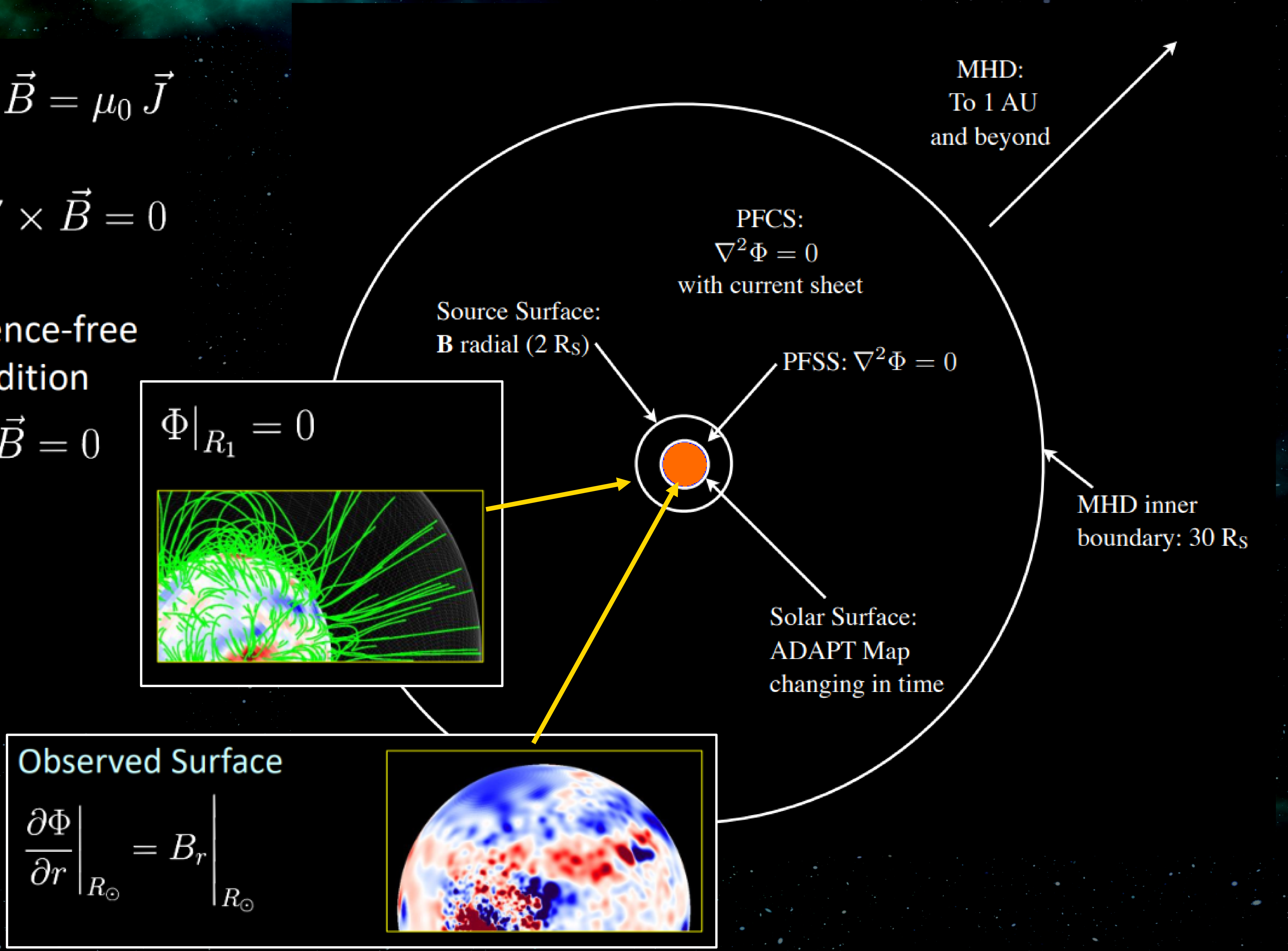
Assume no current: $\cancel{\vec{J}} \longrightarrow \nabla \times \vec{B} = 0$

Solution form: $\vec{B} = \nabla \Phi$

Divergence-free condition: $\nabla \cdot \vec{B} = 0$

$$\boxed{\nabla^2 \Phi = 0}$$

Laplace equation



MHD:
To 1 AU
and beyond

PFCS:
$\nabla^2\Phi = 0$
with current sheet

Source Surface:
B radial (2 R$_S$)

PFSS: $\nabla^2\Phi = 0$

MHD inner
boundary: 30 R$_S$

Solar Surface:
ADAPT Map
changing in time

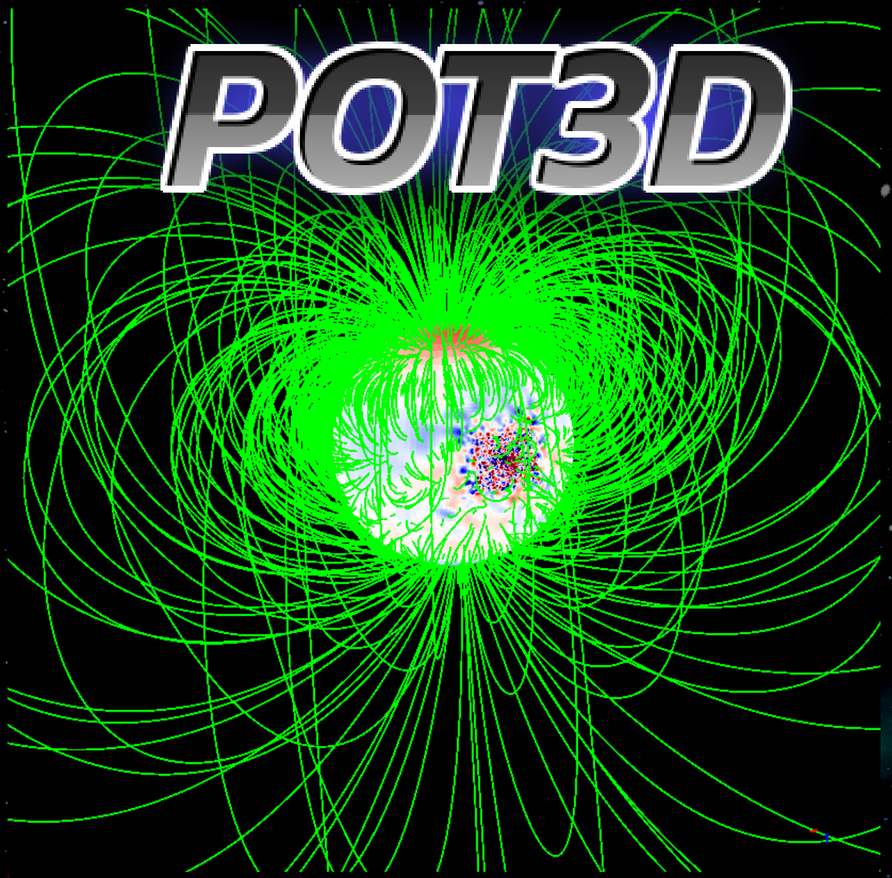
$$\Phi\big|_{R_1} = 0$$

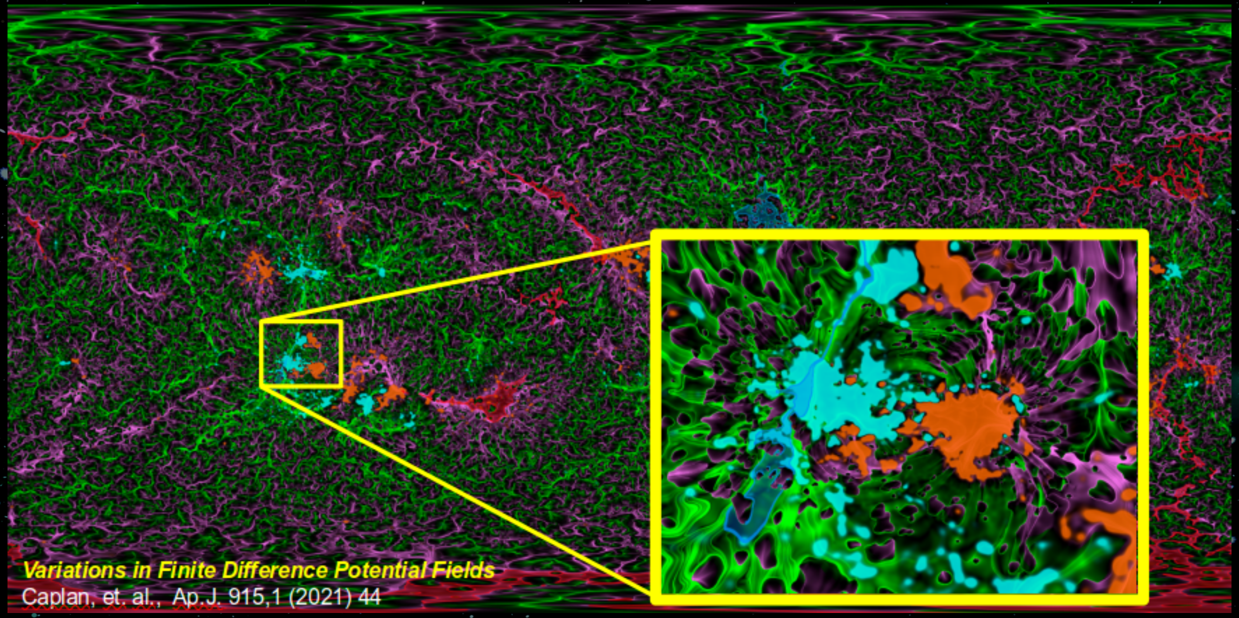
Observed Surface

$$\left.\frac{\partial \Phi}{\partial r}\right|_{R_\odot} = B_r\Big|_{R_\odot}$$

POT3D

- POT3D is a code that computes potential field approximations of the solar coronal magnetic field using observations of the solar surface magnetic field as a boundary condition
- The code is parallelized for use on CPUs and GPUs using MPI+OpenACC and StdPar (Standard Parallelism)
- The HDF5 file format is used for input/output

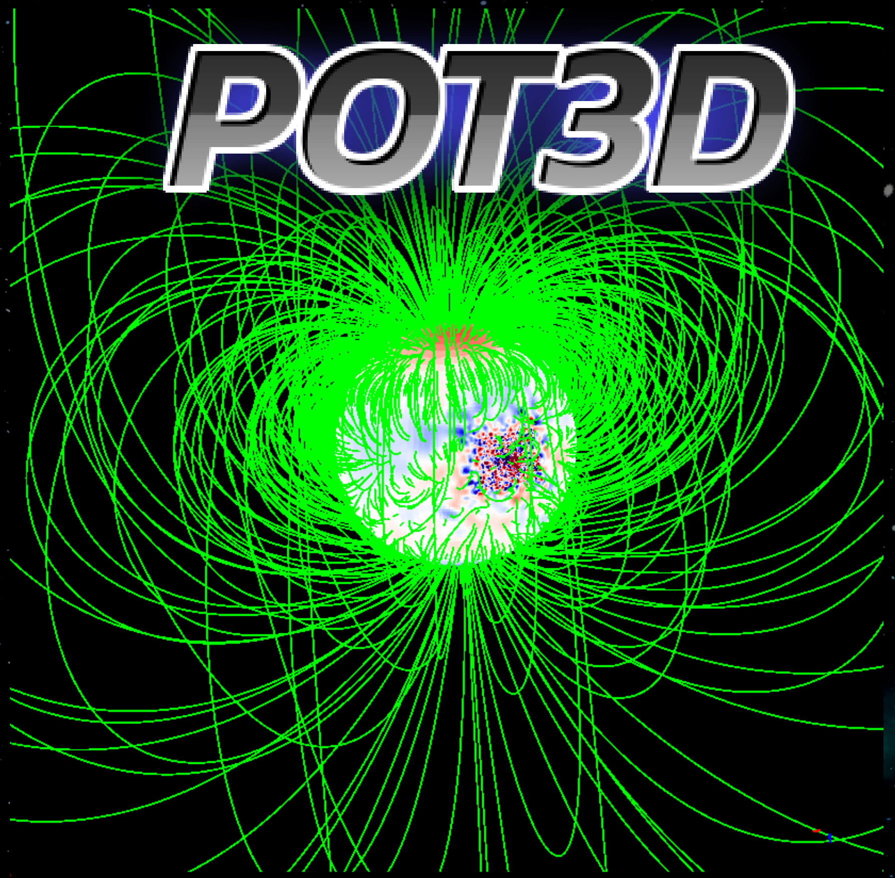



Variations in Finite Difference Potential Fields
Caplan, et. al., Ap.J. 915,1 (2021) 44

github.com/predsci/pot3d

POT3D

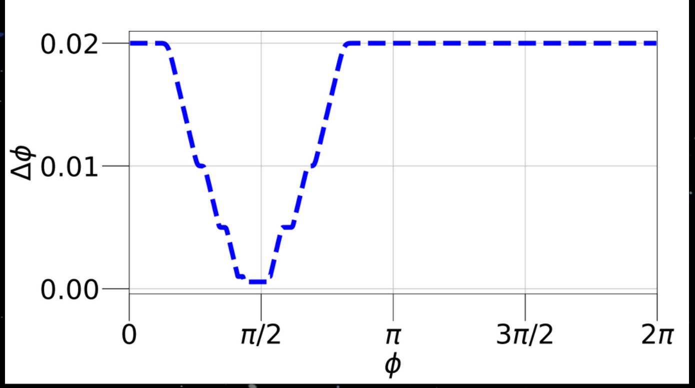
- POT3D is included in the Standard Performance Evaluation Corporation's (SPEC) SPEChpc(TM) 2021 benchmark suite
- POT3D was one of the codes used in the ISC2023 Student Cluster Competition
- Publications describing POT3D:
- ***Variations in Finite Difference Potential Fields***
  Caplan, et. al., Ap.J. 915,1 (2021) 44
- ***From MPI to MPI+OpenACC: Conversion of a legacy FORTRAN PCG solver for the spherical Laplace equation***
  Caplan, et. al., arXiv:1709.01126 (2017)

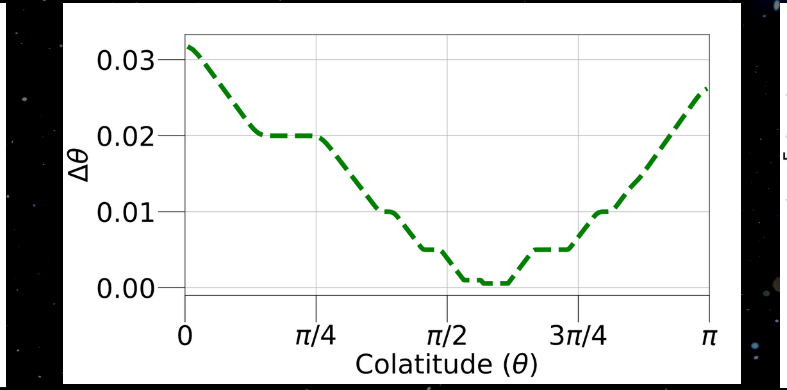


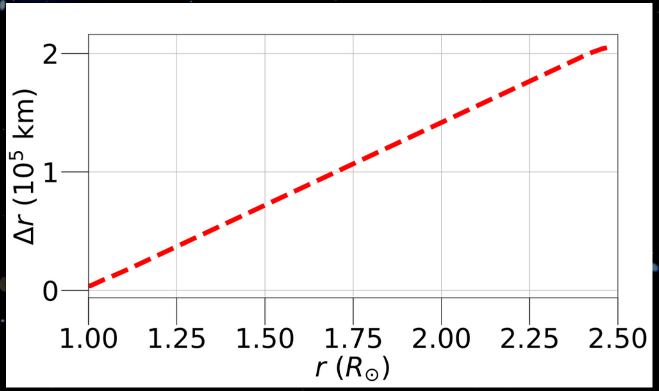
github.com/predsci/pot3d

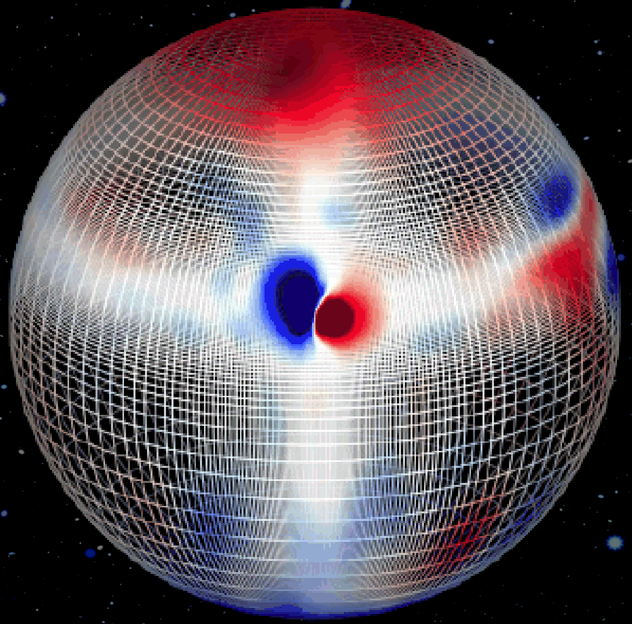
POT3D: Grid

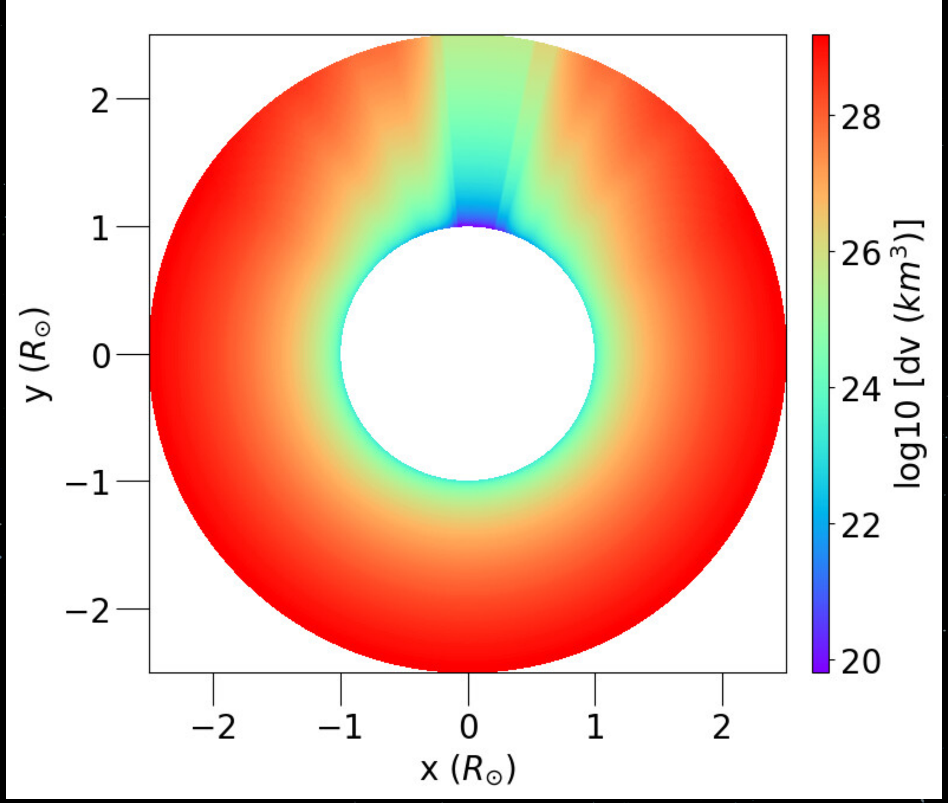
Non-uniform logically-rectangular spherical staggered grid

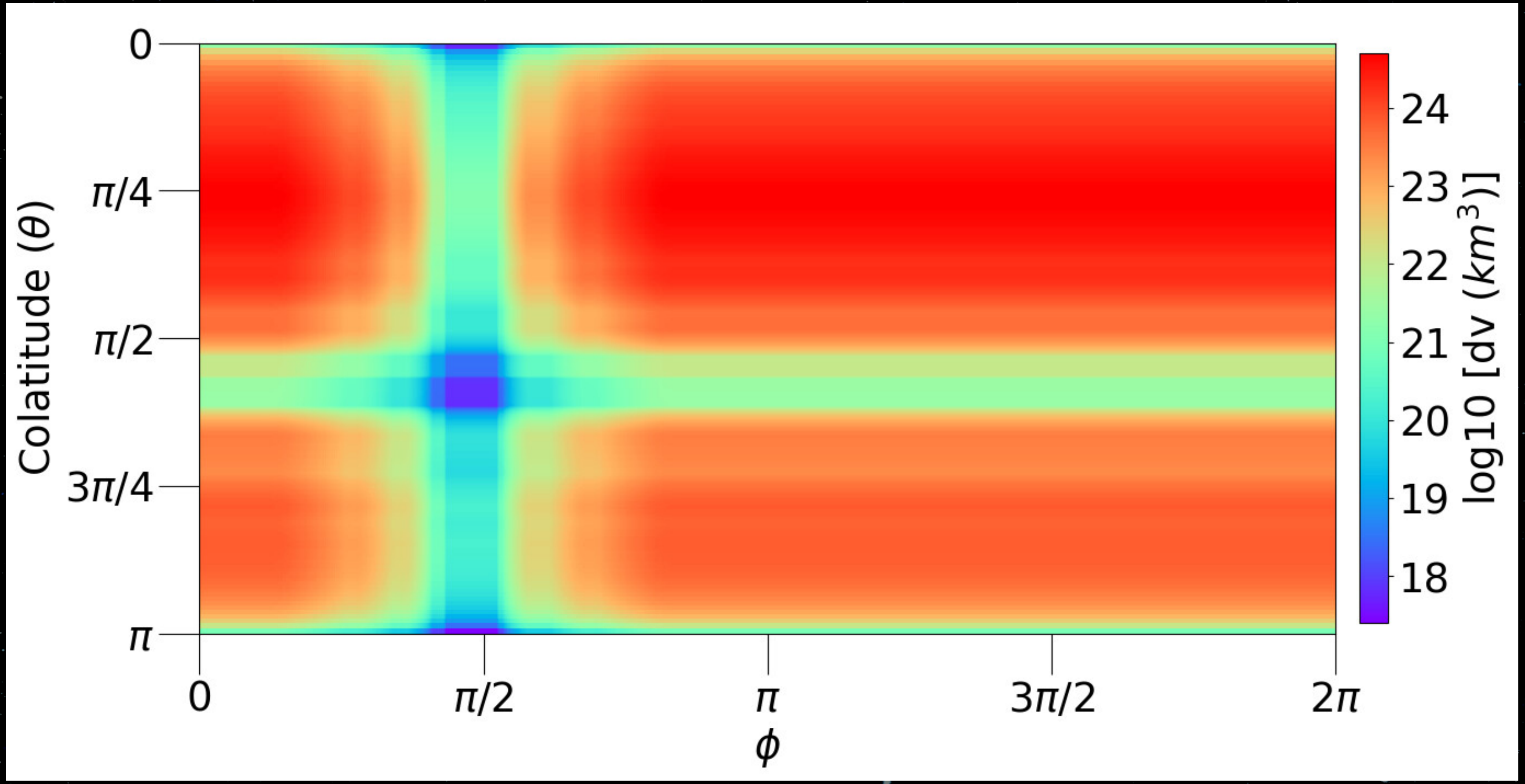




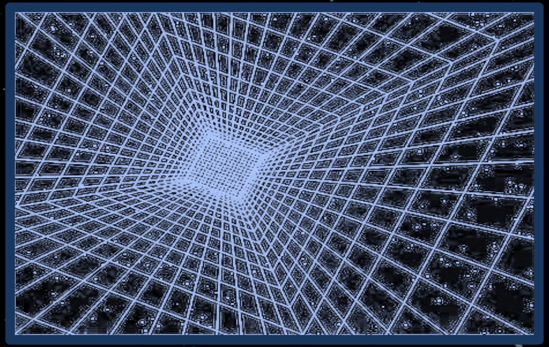








POT3D: Numerical Methods



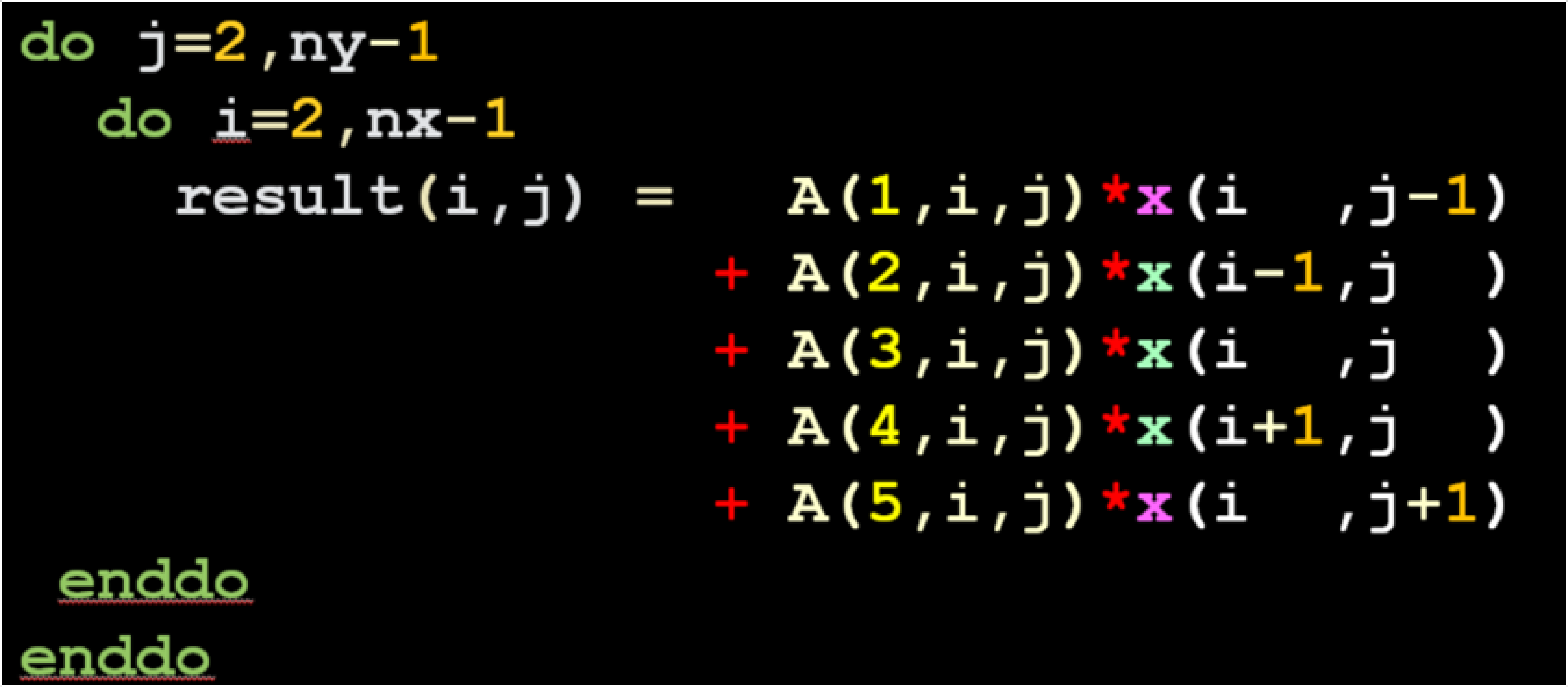
2nd-order Central Difference

$$\nabla^2 \Phi_{i,j,k} \approx \frac{1}{\Delta r_i}\left[\frac{\Phi_{i+1,j,k}-\Phi_{i,j,k}}{\Delta r_{i+\frac{1}{2}}}-\frac{\Phi_{i,j,k}-\Phi_{i-1,j,k}}{\Delta r_{i-\frac{1}{2}}}\right]$$
$$+\frac{1}{\sin\theta_j\,\Delta\theta_j}\left[\sin\theta_{i,j+\frac{1}{2}}\frac{\Phi_{i,j+1,k}-\Phi_{i,j,k}}{\Delta\theta_{j+\frac{1}{2}}}-\sin\theta_{i,j-\frac{1}{2}}\frac{\Phi_{i,j,k}-\Phi_{i,j-1,k}}{\Delta\theta_{j-\frac{1}{2}}}\right]$$
$$+\frac{1}{\sin^2\theta_j\,\Delta\phi_k}\left[\frac{\Phi_{i,j,k+1}-\Phi_{i,j,k}}{\Delta\phi_{k+\frac{1}{2}}}-\frac{\Phi_{i,j,k}-\Phi_{i,j,k-1}}{\Delta\phi_{k-\frac{1}{2}}}\right]=0.$$

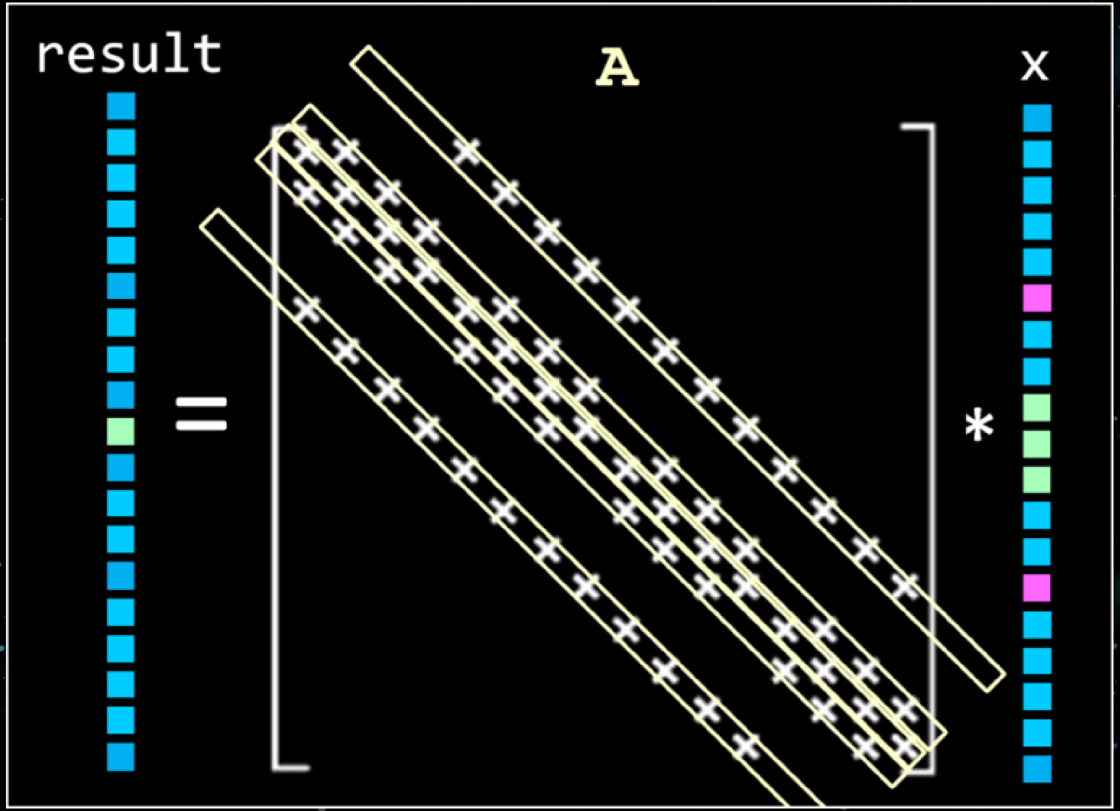
POT3D: Numerical Methods

- The coefficients for all grid points' local stencils are stored as a sparse 7-banded matrix in a modified "DIA" storage format

- The equation is then solved using an iterative Preconditioned Conjugate Gradient (PCG) solver, consisting of array operations (axpy), matrix-vector products, and dot products



```
do j=2,ny-1
  do i=2,nx-1
    result(i,j) =   A(1,i,j)*x(i  ,j-1)
                  + A(2,i,j)*x(i-1,j  )
                  + A(3,i,j)*x(i  ,j  )
                  + A(4,i,j)*x(i+1,j  )
                  + A(5,i,j)*x(i  ,j+1)
 enddo
enddo
```

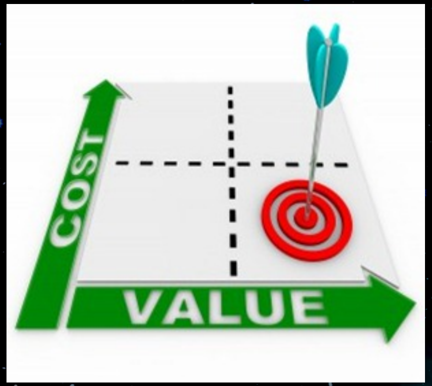


POT3D: Numerical Methods

$$\mathbf{A}\,\vec{x} \qquad \vec{x}\cdot\vec{y}$$

$$a\,\vec{x}+b\,\vec{y}$$

- PCG consists of matrix-vector products, vector operations, dot products, and preconditioner (PC) application
- Applying the PC approximates applying the matrix inverse, but much less expensive to compute
- The PC reduces the number of iterations required for convergence
- Choosing a PC not simple; balance between cost and effectiveness



- For our solver, we use two *communication free* preconditioning options:

PC1

Point-Jacobi / Diagonal scaling

Cheap, not very effective

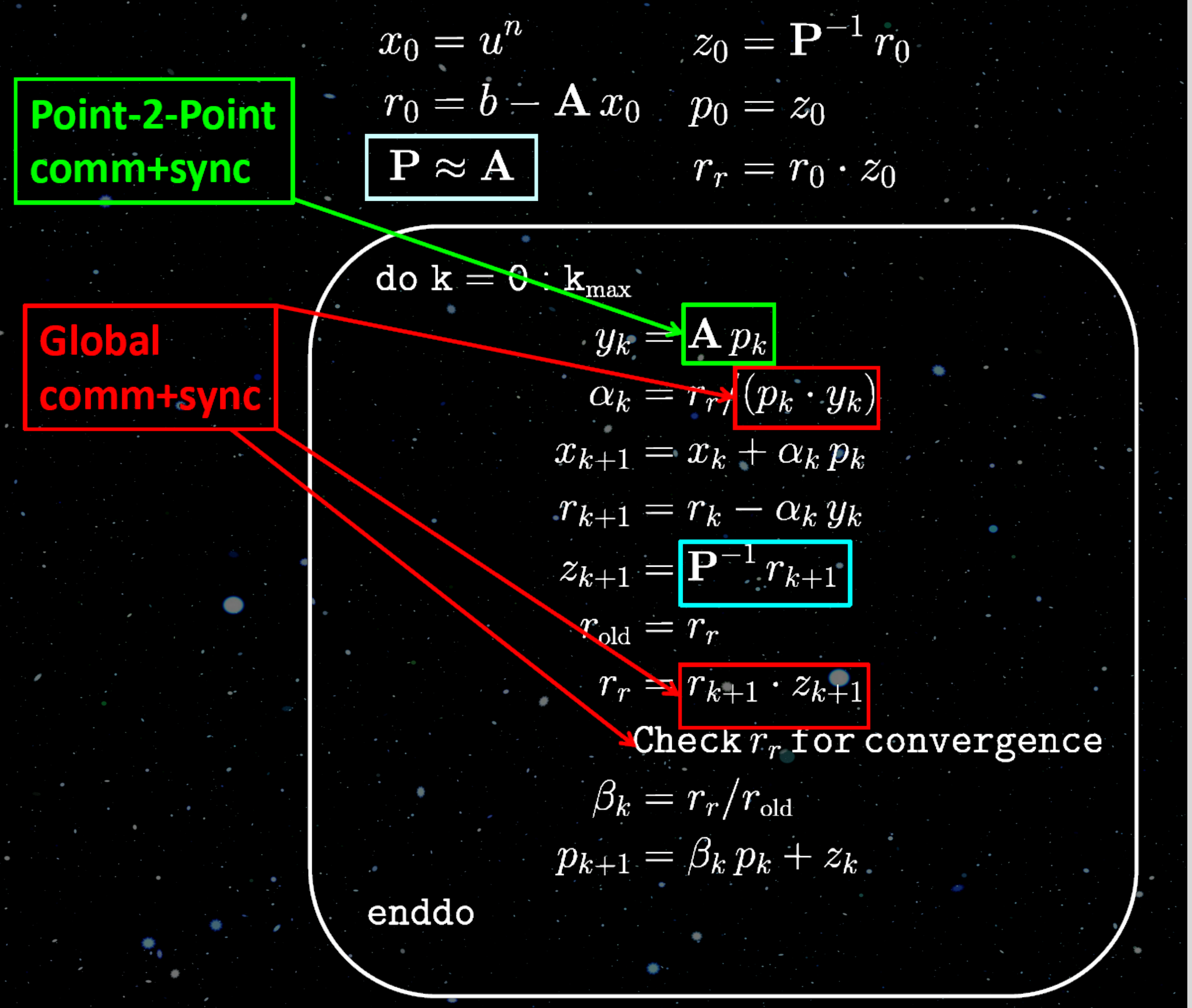
PC2

Non-overlapping domain decomposition zero-fill incomplete LU factorization

Expensive, much more effective!

POT3D: Numerical Methods

PCG



$x_0 = u^n$ $z_0 = \mathbf{P}^{-1} r_0$

$r_0 = b - \mathbf{A}\, x_0$ $p_0 = z_0$

$\boxed{\mathbf{P} \approx \mathbf{A}}$ $r_r = r_0 \cdot z_0$

Point-2-Point comm+sync

Global comm+sync

```
do k = 0 : k_max
        $y_k = \boxed{\mathbf{A}\, p_k}$
        $\alpha_k = r_r / \boxed{(p_k \cdot y_k)}$
        $x_{k+1} = x_k + \alpha_k\, p_k$
        $r_{k+1} = r_k - \alpha_k\, y_k$
        $z_{k+1} = \boxed{\mathbf{P}^{-1}\, r_{k+1}}$
        $r_{\text{old}} = r_r$
        $r_r = \boxed{r_{k+1} \cdot z_{k+1}}$
        Check $r_r$ for convergence
        $\beta_k = r_r / r_{\text{old}}$
        $p_{k+1} = \beta_k\, p_k + z_k$
enddo
```

- **PC1:** Simple vector operation, GPU implementation straight-forward
- **PC2:** Sequential in nature (setup and application); alternative algorithms needed for GPU implementation

PC1

LOAD

```
do j = 1 : N
    $P_{jj} = A_{jj}$
enddo
```

SOLVE

```
do i = 1 : N
    $z_i = P_{ii}\, r_i$
enddo
```

PC2

LOAD

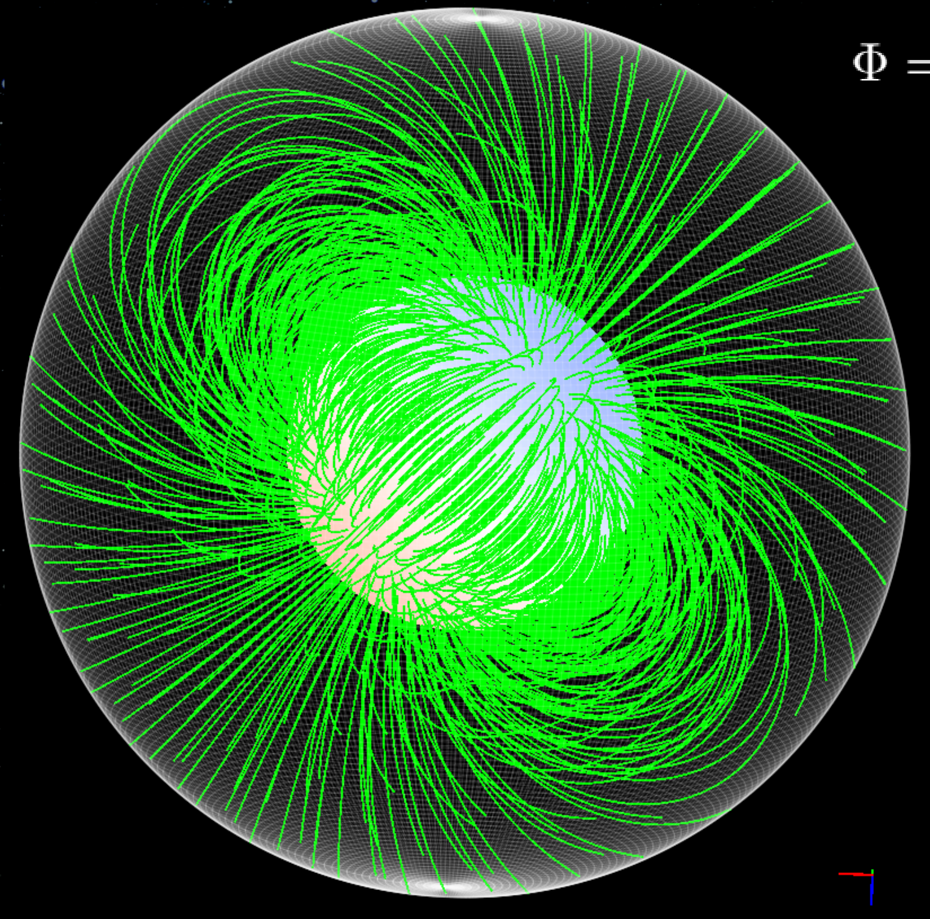
```
$\mathbf{LU} = \mathbf{A}$
do i = 2 : N
    do k = 1 : i − 1        $(LU_{ik} \neq 0)$
        $LU_{ik} = LU_{ik} / LU_{kk}$
        do j = k + 1 : N        $(LU_{ij} \neq 0)$
            $LU_{ij} = LU_{ij} - LU_{ik}\, LU_{kj}$
        enddo
    enddo
enddo
$\mathbf{P} = \mathbf{LU}$
```

SOLVE

```
do i = 1 : N
    $z_i^* = r_i$
    do j = 1 : i        $(LU_{ij} \neq 0)$
        $z_i^* = z_i^* - LU_{ij}\, z_j^*$
    enddo
enddo
do i = N : 1
    $z_i = z_i^*$
    do j = i + 1 : N        $(LU_{ij} \neq 0)$
        $z_i = z_i - LU_{ij}\, z_j$
    enddo
    $z_i = z_i / LU_{ii}$
enddo
```

POT3D: Validation

Tilted Dipole Solution



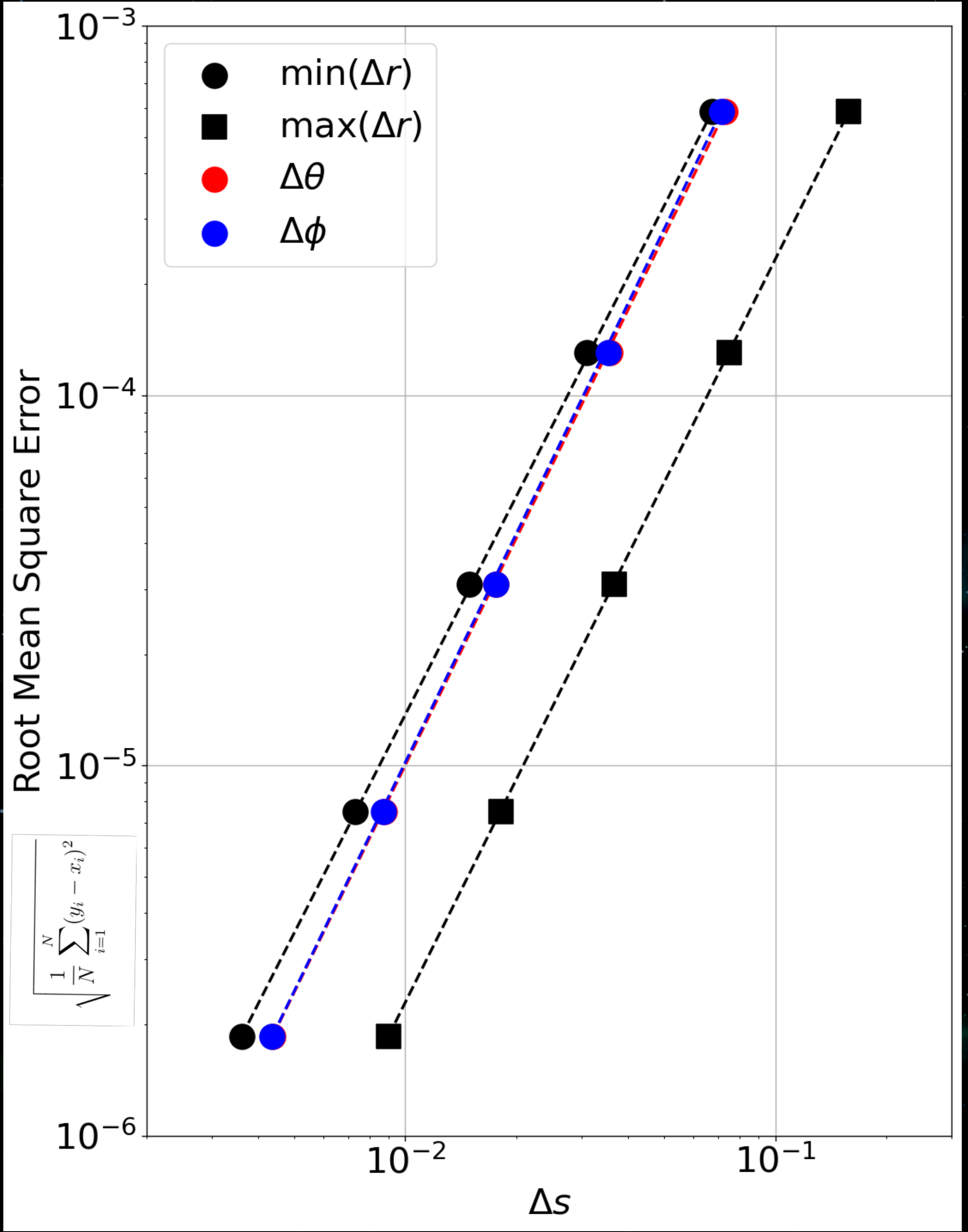
$$\Phi = -\frac{B_0}{r^2}\,\left(\cos\theta\,\cos\alpha + \sin\theta\,\cos\phi\,\sin\alpha\right)$$

$$B_r = \frac{2\,B_0}{r^3}\,\left(\cos\theta\,\cos\alpha + \sin\theta\,\cos\phi\,\sin\alpha\right),$$

$$B_\theta = \frac{B_0}{r^3}\,\left(\sin\theta\,\cos\alpha - \cos\theta\,\cos\phi\,\sin\alpha\right),$$

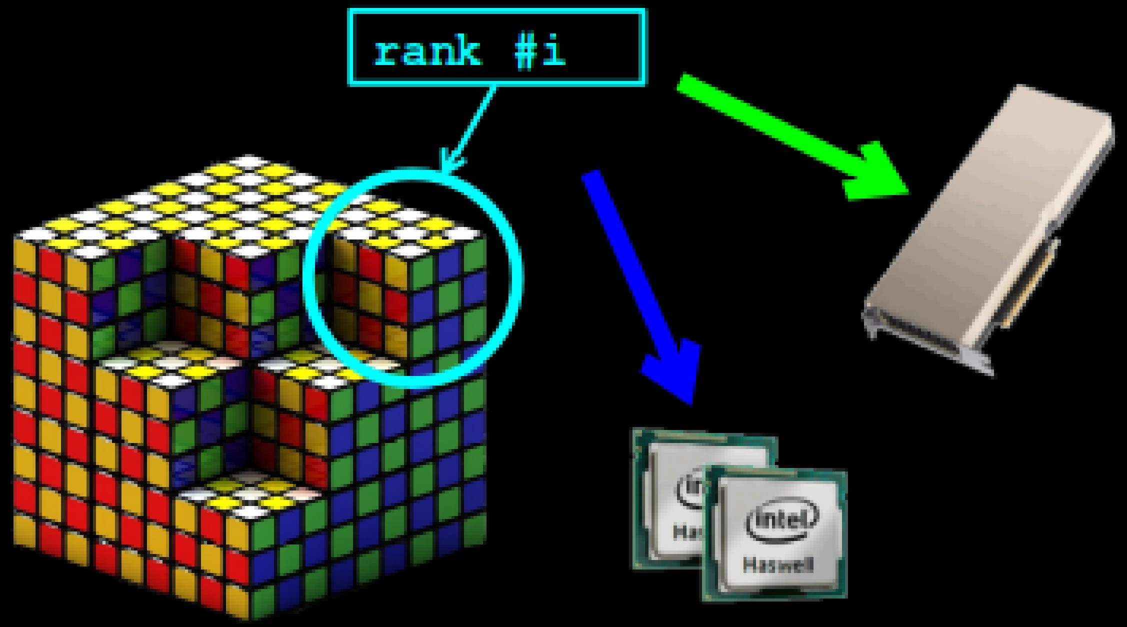
$$B_\phi = \frac{B_0}{r^3}\,\sin\phi\,\sin\alpha,$$

N_r	N_t	N_p	Δr_{min}	Δr_{max}	$\Delta\theta$	$\Delta\phi$	RMSE	Order	Order	Order	Order
								$(\Delta r_{\mathrm{min}})$	$(\Delta r_{\mathrm{max}})$	$(\Delta\theta)$	$(\Delta\phi)$
16	45	90	6.76e-2	1.58e-1	7.31e-2	7.14e-2	5.86e-4	-	-	-	-
32	90	180	3.10e-2	7.52e-2	3.57e-2	3.53e-2	1.31e-4	1.92	1.97	1.98	1.99
64	180	360	1.49e-2	3.67e-2	1.76e-2	1.76e-2	3.09e-5	2.02	2.01	2.00	2.00
128	360	720	7.30e-3	1.81e-2	8.78e-3	8.75e-3	7.52e-6	2.09	2.04	2.03	2.01
256	720	1440	3.61e-3	9.00e-3	4.38e-3	4.37e-3	1.85e-6	2.13	2.07	2.02	2.02



POT3D: Code Implementation: Parallelism

- The logical grid is broken up into 3D blocks split as evenly as possible across all MPI ranks
- For operations that require neighbors (matrix-vector products), asynchronous point-to-point MPI communication is used (iSend/iRecv)
- For dot products and other collectives, global MPI “Allgather” routines are used
- Each MPI rank’s local block of grid points computed by
  - 1 CPU thread (MPI-only)
  - 1 GPU (multi-GPUs)
  - Many CPU threads (hybrid-CPU)
- The local block parallelism is achieved through the use of Fortran’s standard parallelism (DC) along with OpenACC for loops that are not yet supported with DC as well as manual GPU-CPU data movement



3D Logical Domain Decomposition
[MPI_Cart_create()]

```
do concurrent (i=1:N)
  y(i) = a*x(i) + y(i)
enddo
```

```
!$acc parallel default(present)
!$acc loop
  do i=1,n
    y(i) = a*x(i) + y(i)
  enddo
```

POT3D: Code Implementation: Parallelism

POT3D is *highly* memory bandwidth bound

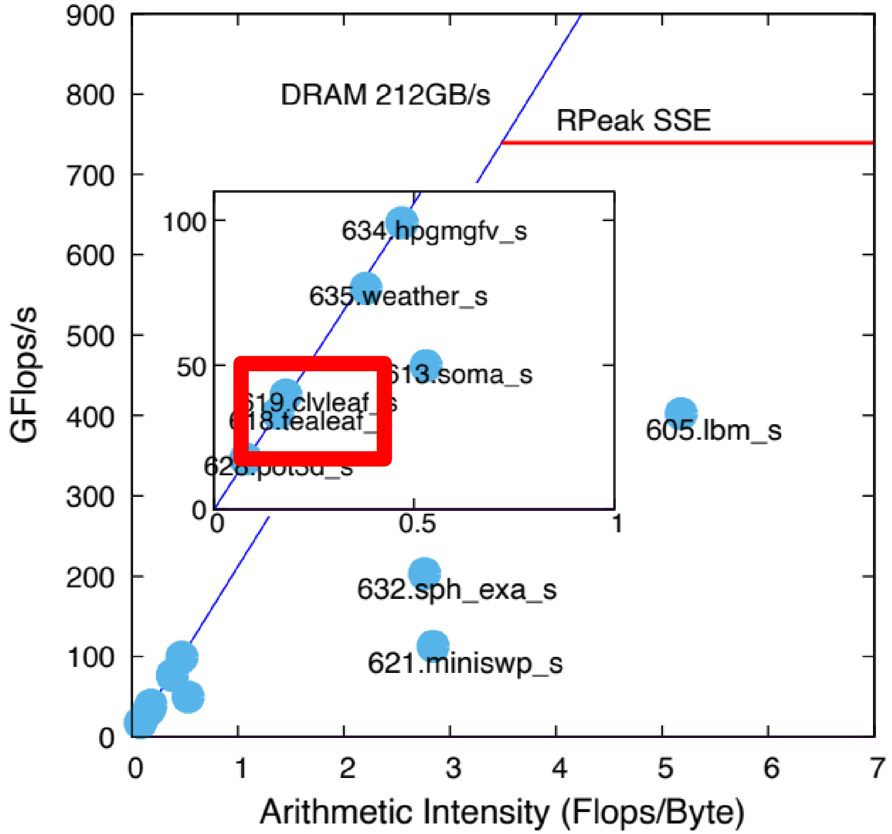
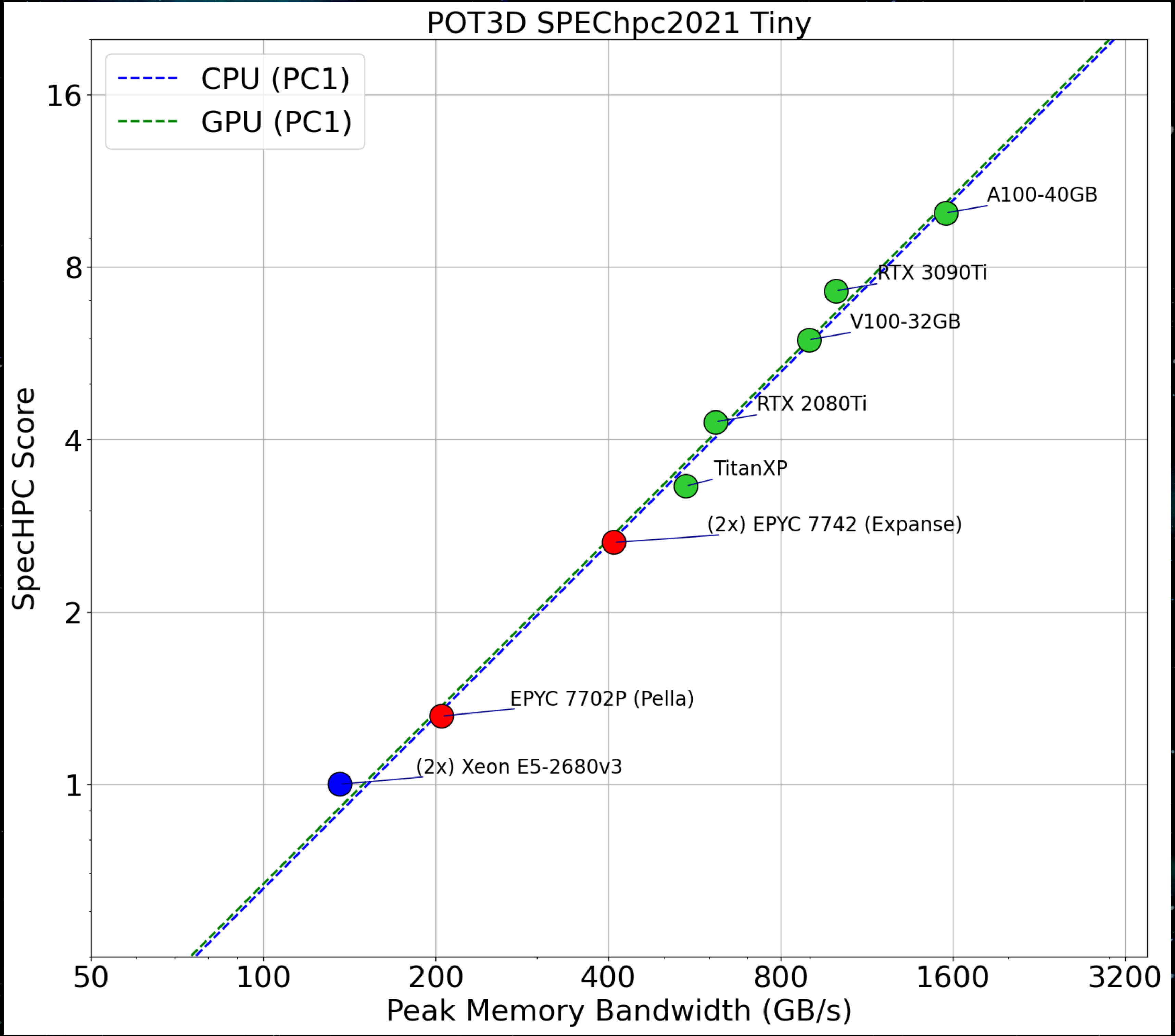


Fig. 4: Roofline plot for the *small* suite. Data collected for MPI-only versions using 4 nodes (224 ranks on Frontera). The roofline plots for the tiny, medium, and large suites are similar. Arithmetic intensity and memory bandwidth are collected for the entire duration of each program.

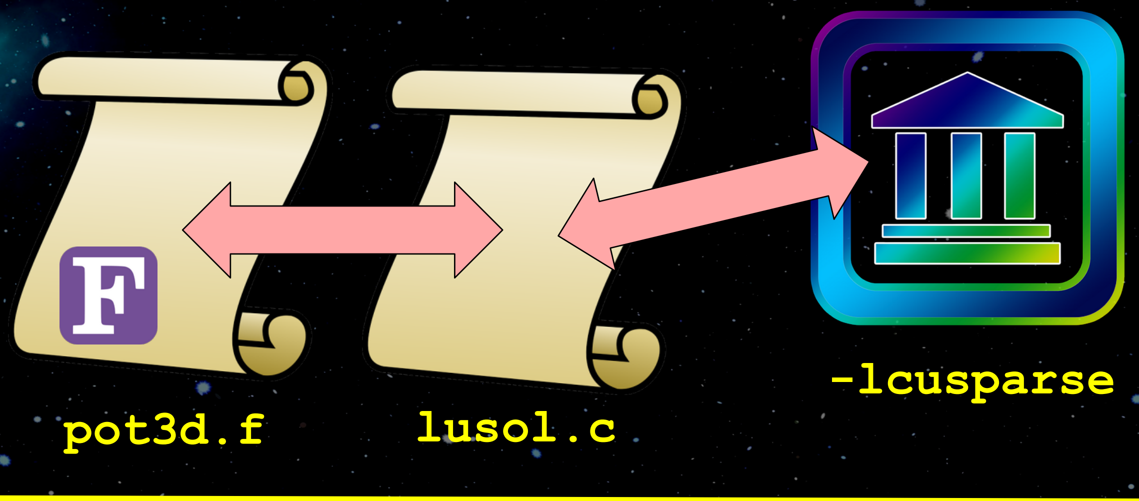
[Brunst et. al. (2022)]



POT3D: Code Implementation: PC2 on GPUs

- cuSparse contains native Fortran bindings
- For portability, we instead call C code from Fortran (minimal **`#ifdef`** pre-processing)

```
nvcc –c [FLAGS] lusol.c
nvfortran [FLAGS] lusol.o [LIBS] pot3d.f
```



lusol.c

```
void load_v3(double* CSR_LU,int* CSR_I,int* CSR_J,int N,int M)
{ ...
cusparseCreate(&cusparseHandle);
...
cusparseDcsrilu02(cusparseHandle,N,M,M_desc,CSR_LU,CSR_I,
                  CSR_J,M_alyz,M_pol,Mbuf);
...}
```

LOAD

```
void lusol_v3(double* x){...
// Forward solve (Ly=x)
  cusparseSpSV_solve(cusparseHandle, L_trans,
            &alpha_DP, L_mat, DenseVecX, DenseVecY, CUDA_R_64F,
            CUSPARSE_SPSV_ALG_DEFAULT, L_described);
  cudaDeviceSynchronize();
// Backward solve (Ux=y)
  cusparseSpSV_solve(cusparseHandle, U_trans,
            &alpha_DP, U_mat, DenseVecY, DenseVecX, CUDA_R_64F,
            CUSPARSE_SPSV_ALG_DEFAULT, U_described);
  cudaDeviceSynchronize();
...}
```

SOLVE

pot3d.f

```
module cusparse_interface
  interface
    subroutine lusol_v3(x) BIND(C, name="lusol")
      use, intrinsic :: iso_c_binding
      type(C_PTR), value :: x
    end subroutine lusol
  end interface
end module
```

```
use, intrinsic :: iso_c_binding
use cusparse_interface
integer(c_int) :: cN
```

```
!$acc host_data use_device(x)
   call lusol(C_LOC(x(1)))
!$acc end host_data
```

POT3D: Code Implementation: Mixed Precision

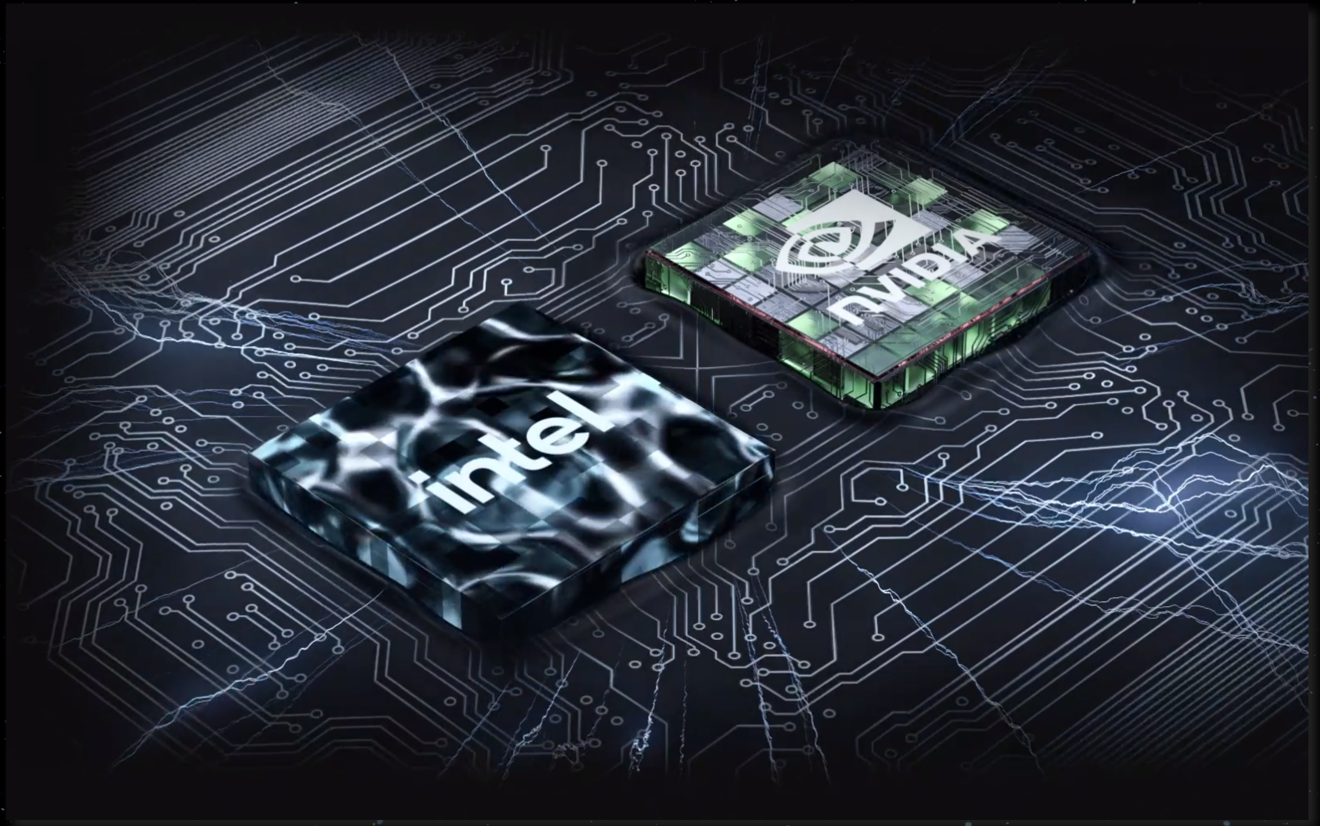
- Single precision
  - Half the memory footprint
  - Can use faster GPU compute cores

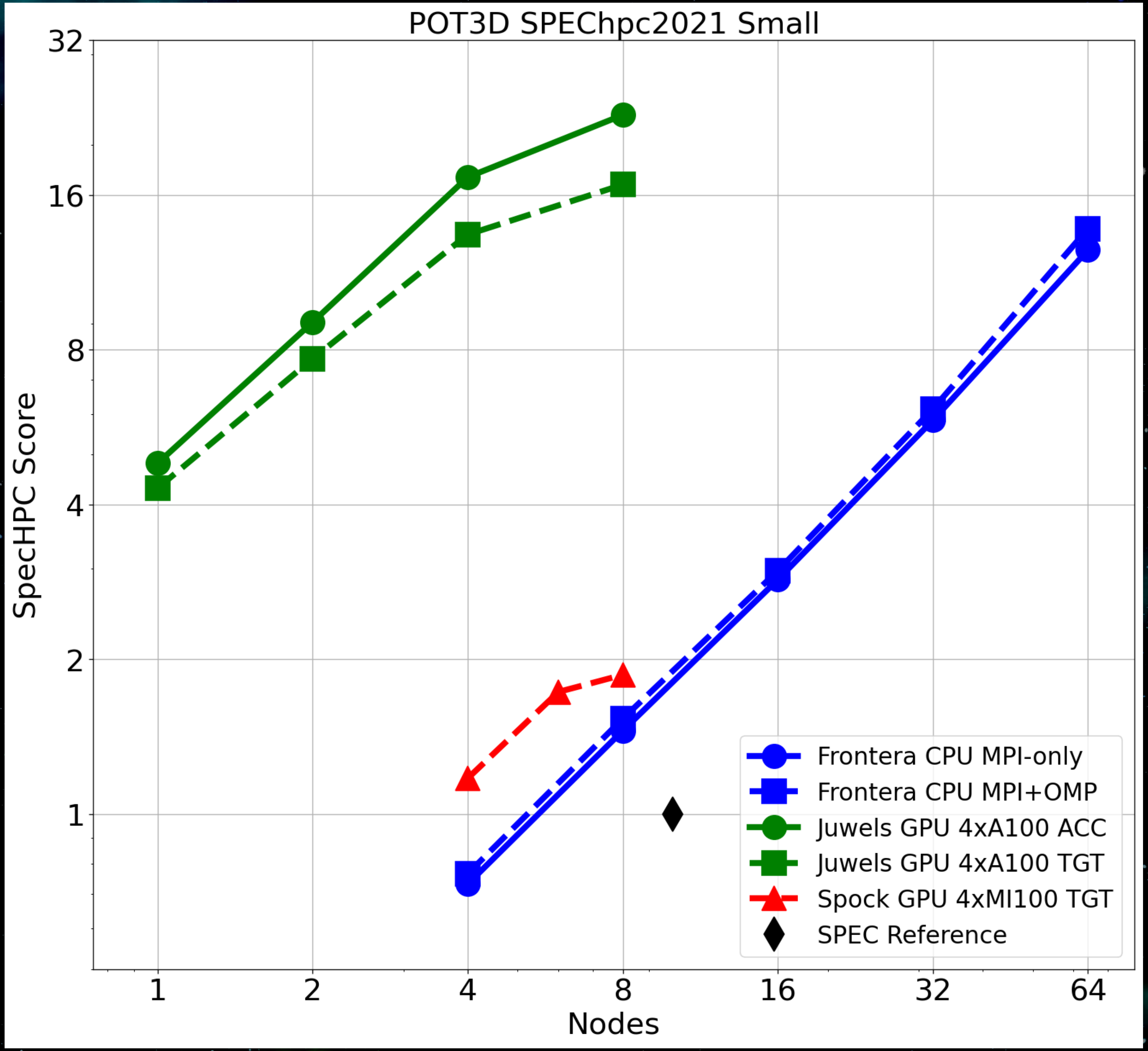
- Can not be used for the overall solve
  - May not converge
  - Solution required to be double precision

- Use only for the preconditioner!
  - PC an approximation, so could speed up the solve while yielding equivalent results
  - Requires casting arrays in and out
  - Number of iterations may go up

POT3D: Code Implementation: Performance

- POT3D scales well to many MPI ranks on both CPUs and GPUs





SPEChpc 2021 “Small” test (300 million points)
[Brunst et. al. (2022)]

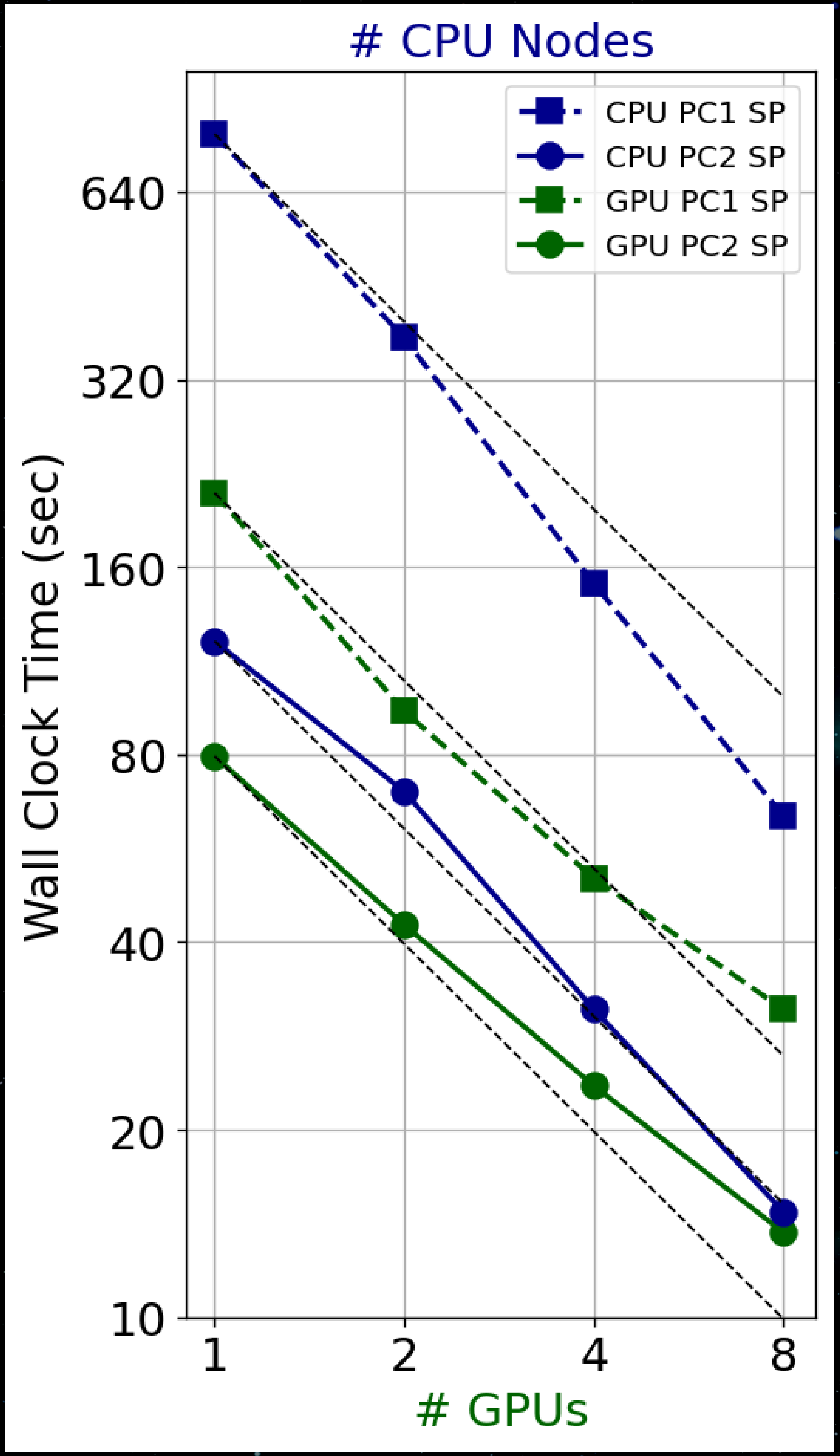
POT3D: C erformance

EXPANSE
COMPUTING WITHOUT BOUNDARIES

CSRC@SDSU DGX A100

# CPUs x Model	**(2x)** EPYC 7742
# GPUs x Model	**8x** A100-40GB SXM4
Peak DP FLOP/s / GPU	9.8 TFLOP/s
Memory / GPU	40 GB
Memory Bandwidth/GPU	1555 GB/s
Compiler Flags	`-O3 -tp=zen2 -acc=gpu -gpu=cc80,cudaXX.Y`

Peak FLOP/s	7.0 TFLOP/s
Memory	256 GB
Total Memory Bandwidth	381.4 GB/s
Compiler Flags	`-O3 -tp=zen2`
OpenMPI	`v4.04`

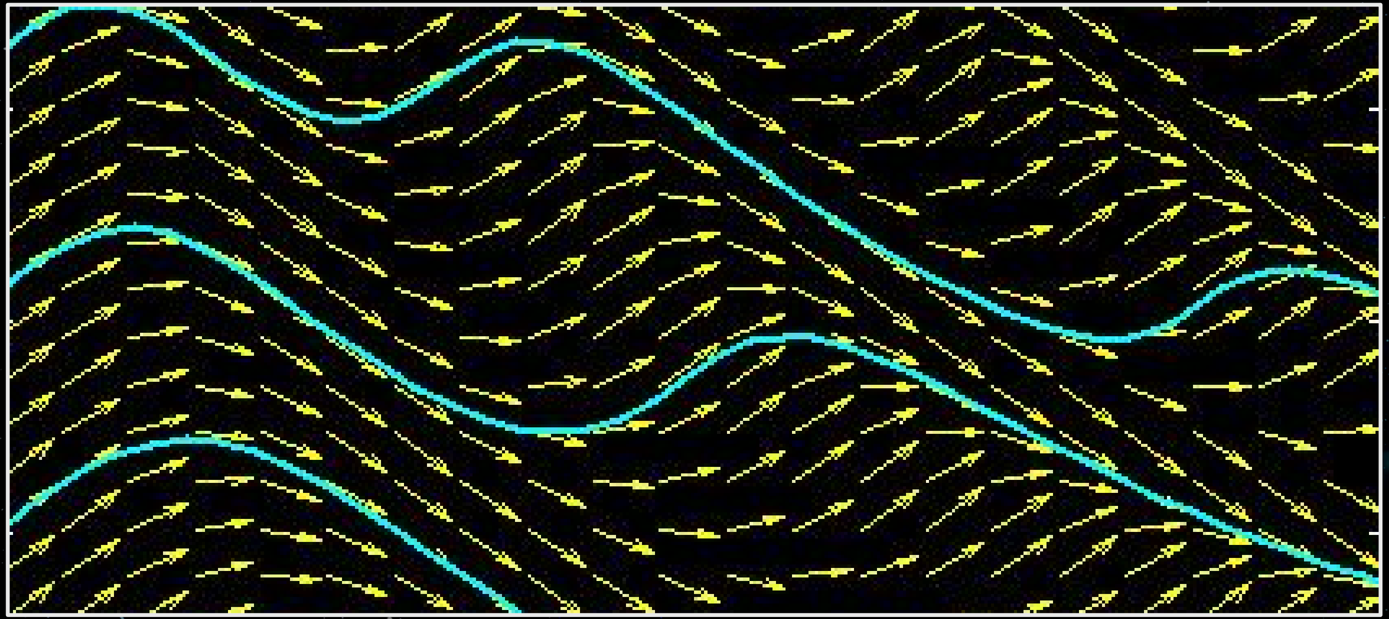


MAPFL

- Accurate field line tracing is very important
- MapFL is a Fortran code that traces field lines through a 3D field defined on a non-uniform spherical gird
- MapFL uses an adaptive tracing step size with a 2nd-order predictor-corrector scheme

github.com/
predsci/mapfl

MAPFL



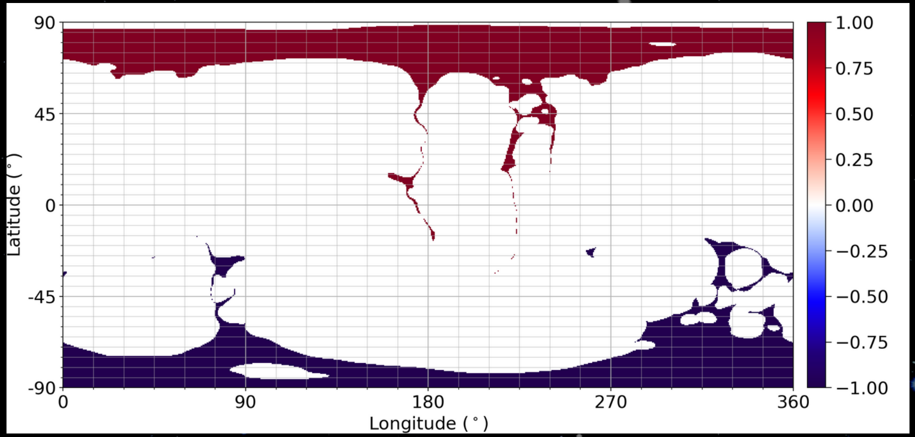
Parallelized across multiple CPU threads using OpenMP

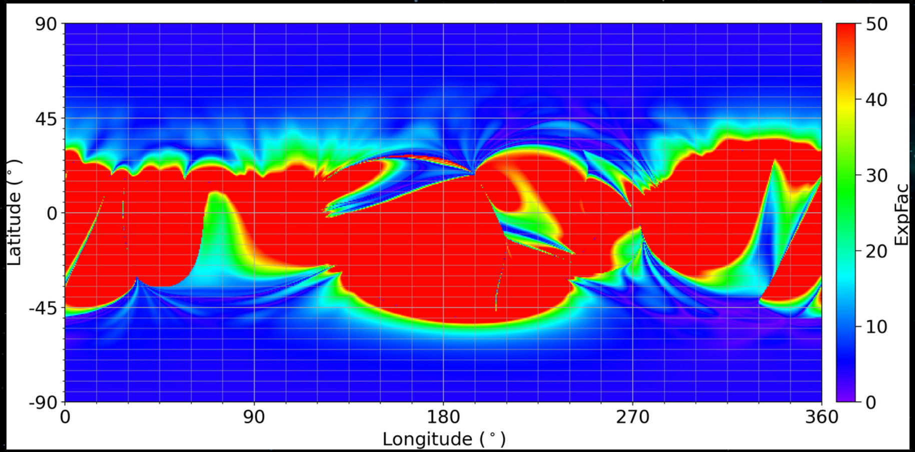
OpenMP®

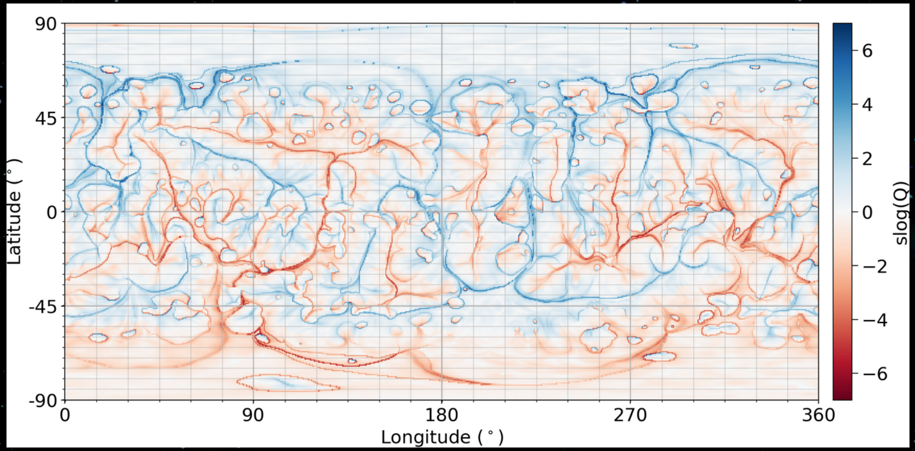
MAPFL: Outputs

- MapFL can trace forwards and backwards, outputting coordinate mappings
- It can also automatically calculate useful quantities, the three most relevant here being:
  - Open field map (needed for WSA)
  - Expansion factors (needed for WSA)
  - Squashing factor, Q (useful for analyzing magnetic structure)

MAPFL

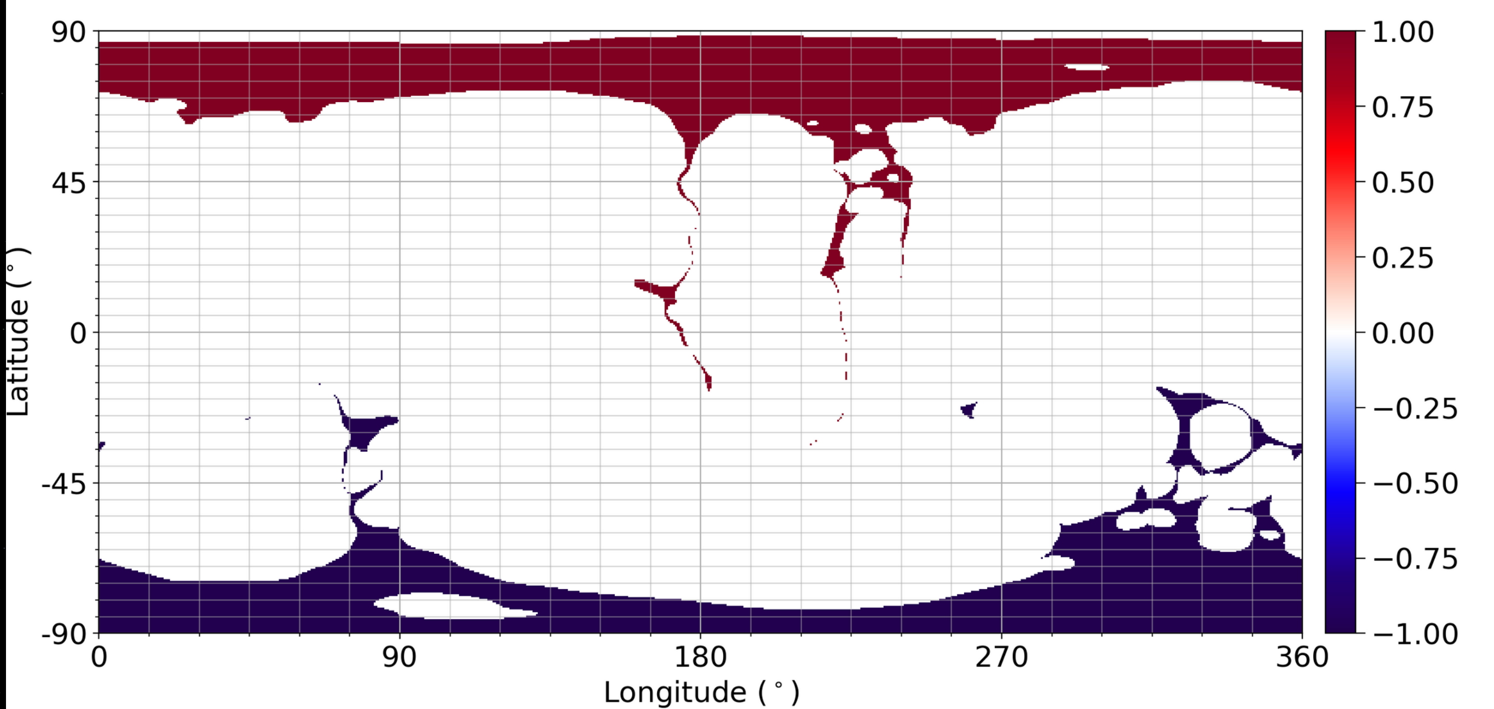


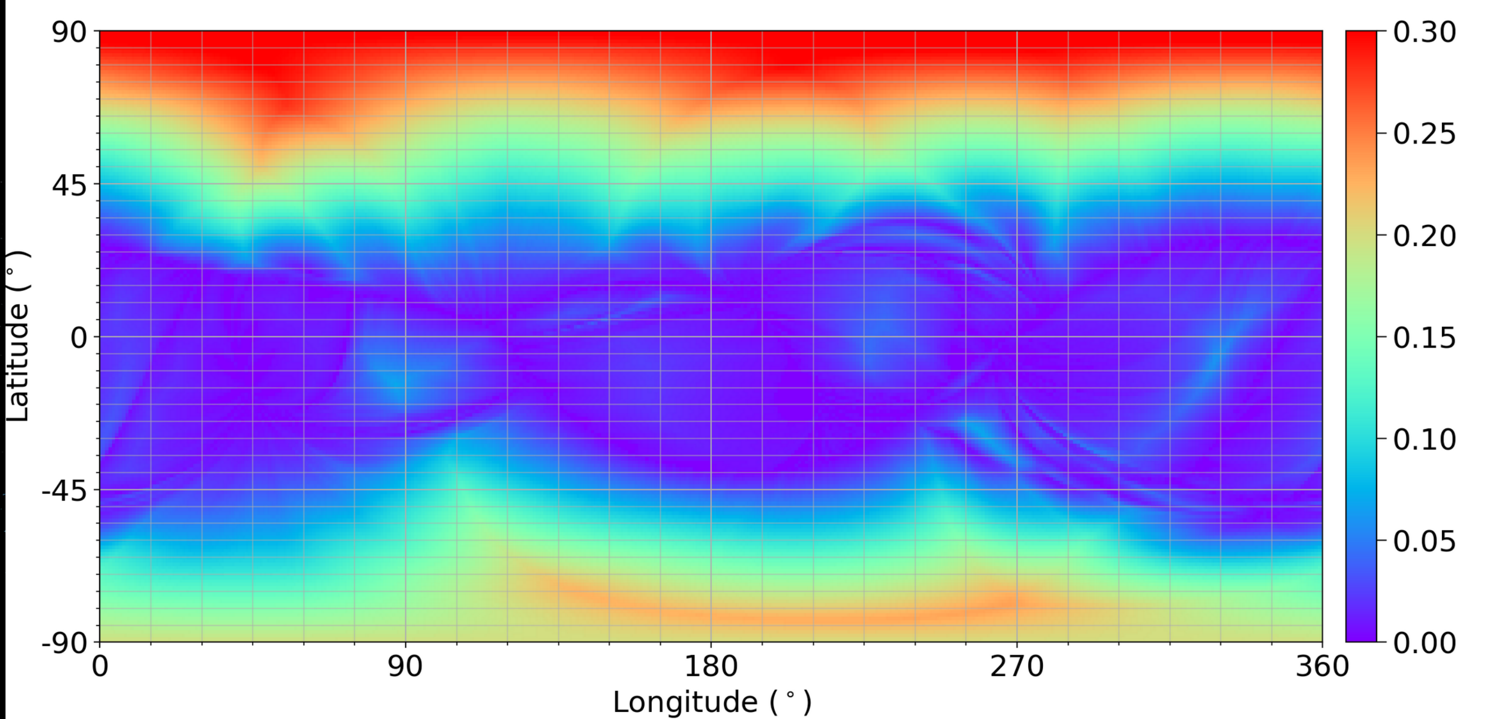




Distance to Coronal Hole Boundaries (DCHB)

- For every point within an open field region, we calculate the distance to the open field (coronal hole) boundary denoted as DCHB
- As WSA needs the DCHB at the outer CS boundary, we use the MapFL tracings from **`r1→rss`** and **`rss→r1`** to find the values of DCHB at every point at r1





Empirical Solar Wind Models

- Once we have the DCHB at 1 and the expansion factor at rss traced out to r1, we can insert them into an empirical solar wind model (e.g. WSA)

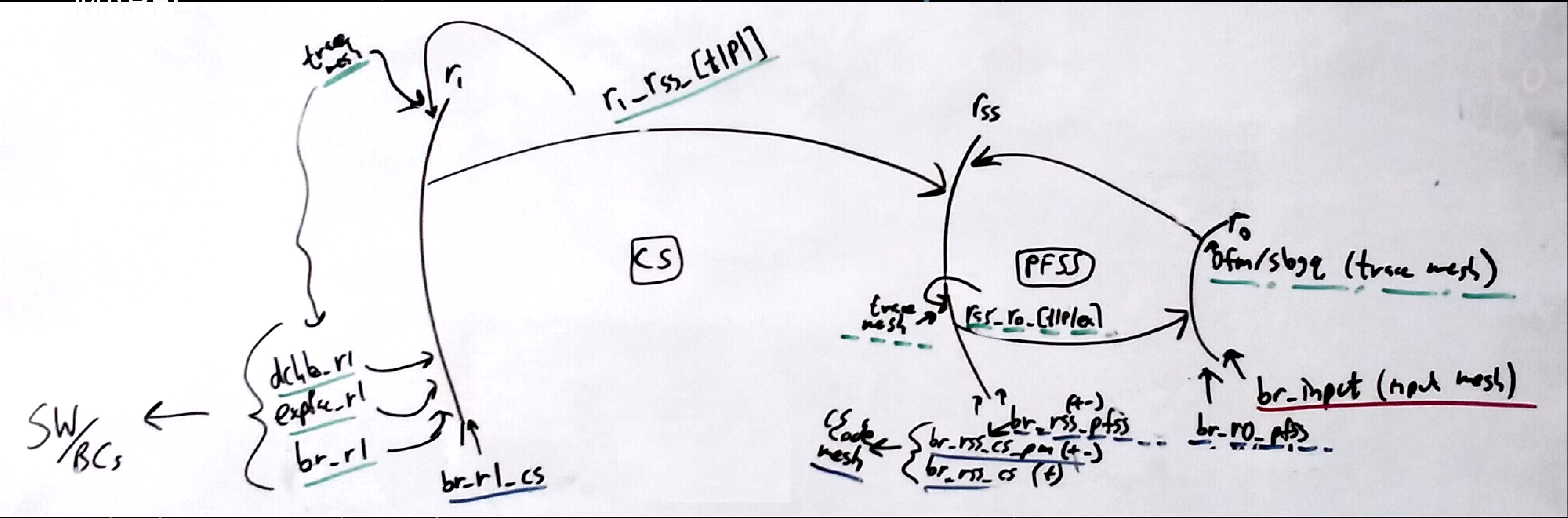
$$V = V_0 + \frac{V_m}{(1 + f_{exp})^{C_1}} \left(1 - C_2 \exp\left[-\left(\frac{\Theta_B}{C_3}\right)^{C_4}\right]\right)^{C_5}$$

$$\rho = \rho_{\mathrm{f}} \left(\frac{\max(V)}{V}\right)^2$$

$$t = t_{\mathrm{f}} \frac{\rho_{\mathrm{f}}}{\rho}$$

Solar Wind Generator (SWiG)

- **`swig.py`** is a python control script that reads in a Br magnetogram and produces an empirical solar wind solution using the PFSS+CS model computed by POT3D traced by MAPFL



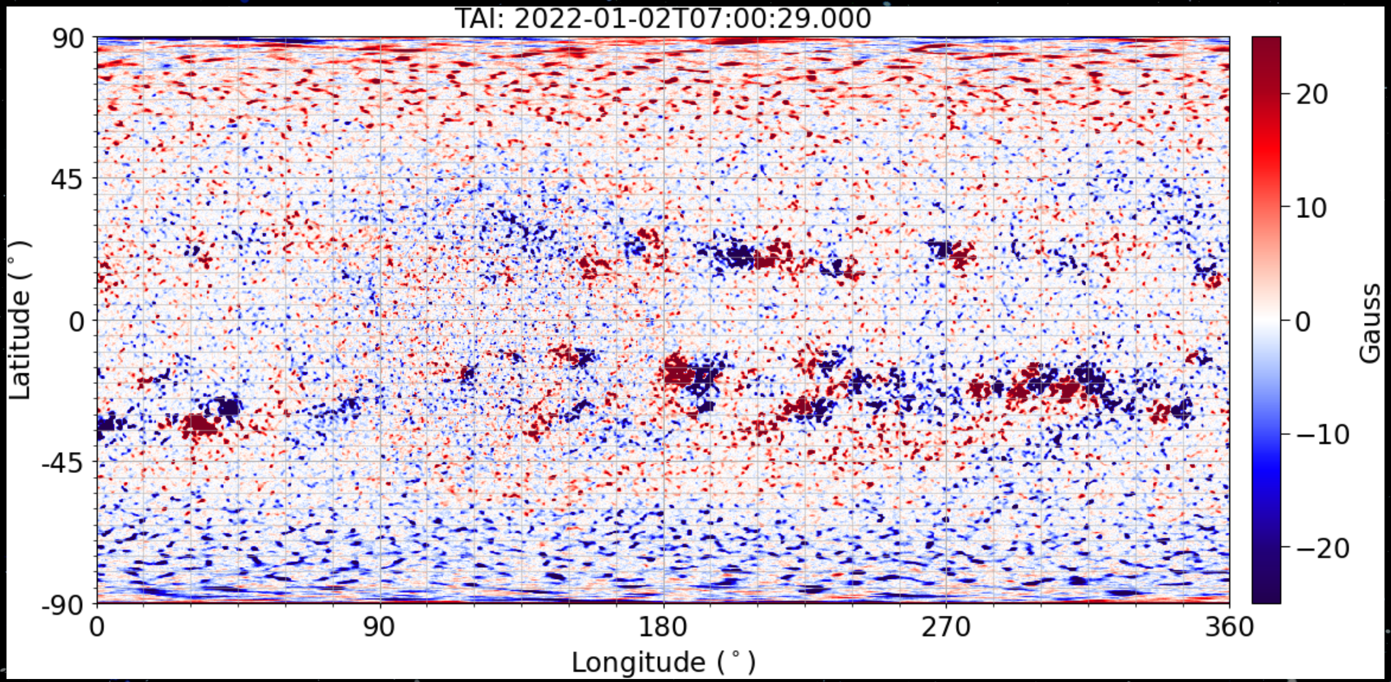
github.com/
predsci/swig

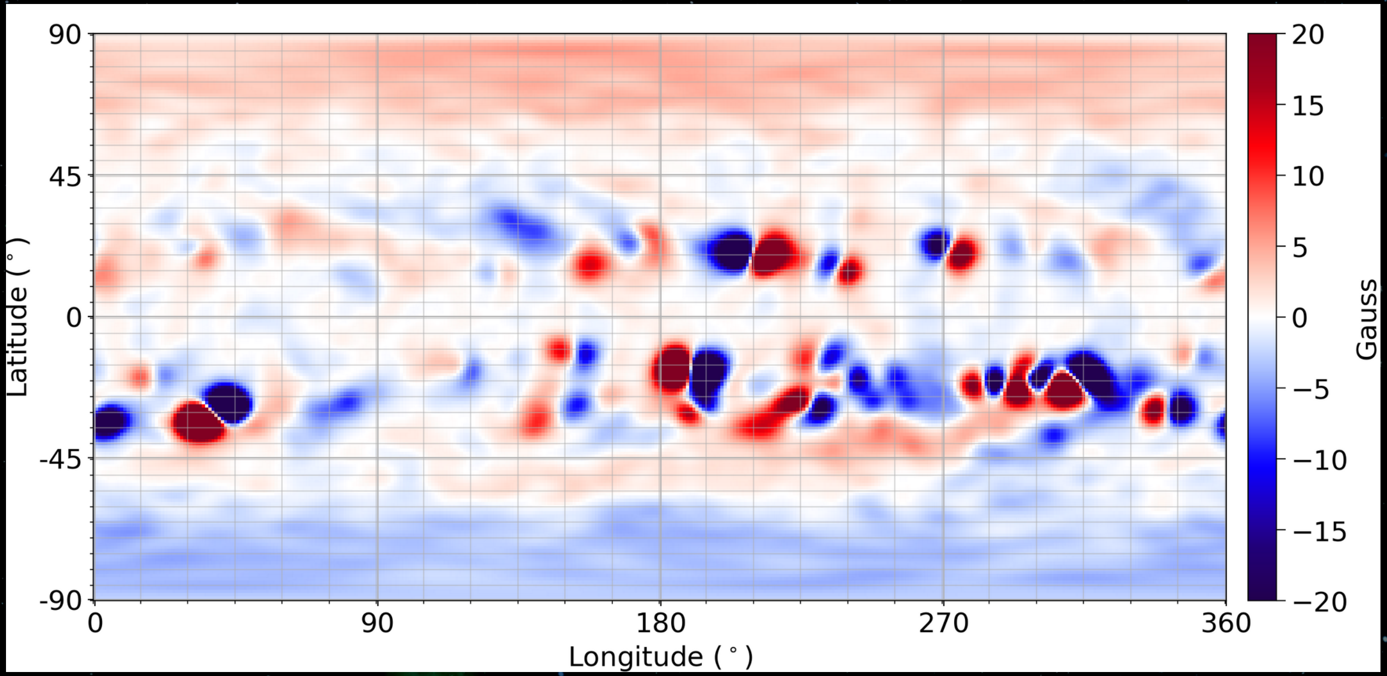
SWiG

BREAK
TIME!!!

Running SWiG

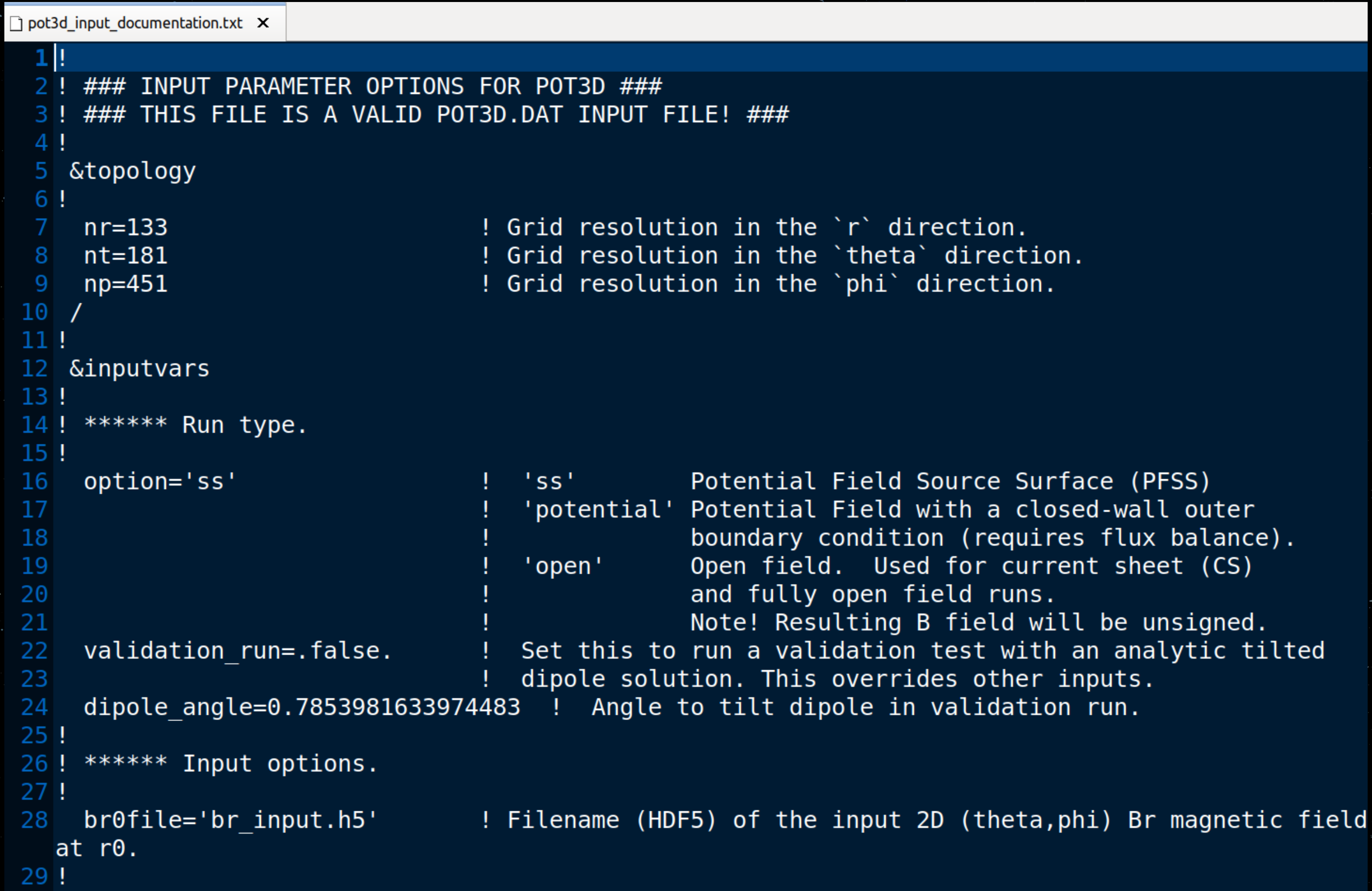
- IMPORTANT! Make sure to process the map first!
  (If map is too pixelated, field line tracing may fail)
- SWiG includes the necessary POT3D and MAPFL input file templates and does all the work for you
- However, it is still illustrative to show how these codes are run individually





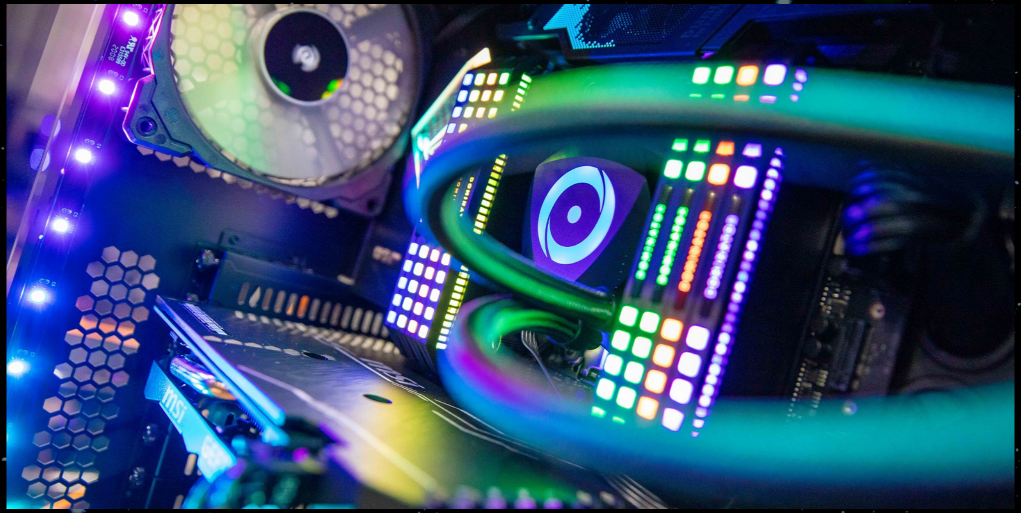
POT3D Input File [pot3d.dat]

- Sample full namelist input file with descriptions of all inputs and sample values: **pot3d/pot3d_input_documentation.txt**



```
!
! ### INPUT PARAMETER OPTIONS FOR POT3D ###
! ### THIS FILE IS A VALID POT3D.DAT INPUT FILE! ###
!
 &topology
!
  nr=133                      ! Grid resolution in the `r` direction.
  nt=181                      ! Grid resolution in the `theta` direction.
  np=451                      ! Grid resolution in the `phi` direction.
 /
!
 &inputvars
!
! ****** Run type.
!
  option='ss'                 !  'ss'        Potential Field Source Surface (PFSS)
                              !  'potential' Potential Field with a closed-wall outer
                              !              boundary condition (requires flux balance).
                              !  'open'      Open field.  Used for current sheet (CS)
                              !              and fully open field runs.
                              !              Note! Resulting B field will be unsigned.
  validation_run=.false.      !  Set this to run a validation test with an analytic tilted
                              !  dipole solution. This overrides other inputs.
  dipole_angle=0.7853981633974483  !  Angle to tilt dipole in validation run.
!
! ****** Input options.
!
  br0file='br_input.h5'       ! Filename (HDF5) of the input 2D (theta,phi) Br magnetic field
at r0.
!
```

Running POT3D: Command line



```
nohup mpirun -np <NUM_CPUS> pot3d
        1>pot3d.log 2>pot3d.err &
```

CPU

```
nohup mpirun --map-by ppr:<NGPUS/SOCKET>:socket
      pot3d 1>pot3d.log 2>pot3d.err &
```

GPU

Running POT3D: Outputs

- Output files:
  - **`pot3d.out`** - `Various output logs`
  - **`timing.out`** - `Timing information about the run`

- Output data (all optional):
  - **`br, bt, and bp`**
    `- 3D hdf5 magnetic field components`
  - **`phi`**
    `- 3D hdf5 scalar potential`
  - **`br_photo_file`**
    `- 2D hdf5 br boundary (after interpolation, flux balancing, and on POT3D staggered grid)`

POT3D Post processing scripts/tools

- **psi_data_reader_2d.py** – Script showing how to read in 2D pot3d data into python
- **psi_data_reader_3d.py** - Script showing how to read in 3D pot3d data into python
- **psi_interp2d_tp.py** – Script to find interpolated values on a map at a series of theta-phi points
- **psi_get_2D_grid_info.py**
- **psi_get_3D_grid_info.py**
- **psi_plot2d** – Used to plot 2D h5 files

POT3D Examples and Testsuite

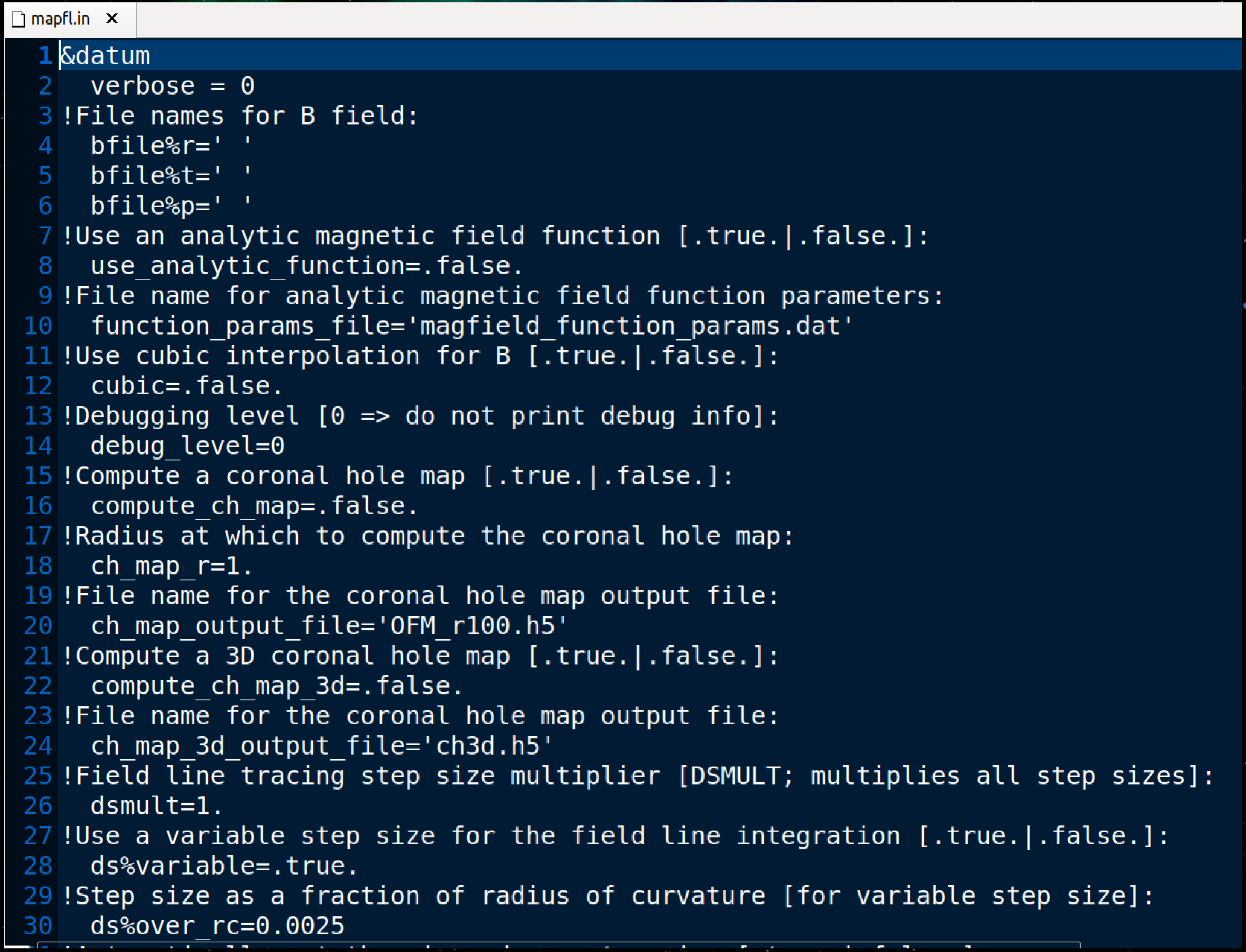
In `pot3d/run_examples`:

```
open_field
potential_field_current_sheet
potential_field_source_surface
```

In `pot3d/testsuite`:

```
isc2023
large
medium
small
validation
run_test.sh <TESTNAME> <NP>
```

MapFL Input File [mapfl.in]



```
&datum
  verbose = 0
!File names for B field:
  bfile%r=' '
  bfile%t=' '
  bfile%p=' '
!Use an analytic magnetic field function [.true.|.false.]:
  use_analytic_function=.false.
!File name for analytic magnetic field function parameters:
  function_params_file='magfield_function_params.dat'
!Use cubic interpolation for B [.true.|.false.]:
  cubic=.false.
!Debugging level [0 => do not print debug info]:
  debug_level=0
!Compute a coronal hole map [.true.|.false.]:
  compute_ch_map=.false.
!Radius at which to compute the coronal hole map:
  ch_map_r=1.
!File name for the coronal hole map output file:
  ch_map_output_file='OFM_r100.h5'
!Compute a 3D coronal hole map [.true.|.false.]:
  compute_ch_map_3d=.false.
!File name for the coronal hole map output file:
  ch_map_3d_output_file='ch3d.h5'
!Field line tracing step size multiplier [DSMULT; multiplies all step sizes]:
  dsmult=1.
!Use a variable step size for the field line integration [.true.|.false.]:
  ds%variable=.true.
!Step size as a fraction of radius of curvature [for variable step size]:
  ds%over_rc=0.0025
```

MapFL Outputs

```
rffile
tffile
pffile
effile
qffile
slogqffile
rbfile
tbfile
pbfile
ebfile
qbfile
slogqbfile
```

All outputs are optional.

Forward and backward tracing information

Can also output traces themselves, 3D volume tracings, length of field lines, and much more
(see sample input file for details)

SWiG: Current options

```
swig.py [-h] [-oidx OIDX] [-rundir RUNDIR] [-np NP] [-gpu]
        [-sw_model SW_MODEL] [-rss RSS] [-r1 R1] [-noplot]
        input_map
```

```
cor_pfss_cs_pot3d.py [-h] [-np NP] [-gpu] [-rss RSS]
[-r1 R1] br_input_file
```

```
mag_trace_analysis.py [-h] rundir
```

```
eswim.py [-h] -dchb DCHB -expfac EXPFAC -model MODEL [-wsa_vslow WSA_VSLOW]
                [-wsa_vfast WSA_VFAST] [-wsa_vmax WSA_VMAX] [-wsa_ef_power WSA_EF_POWER]
                [-wsa_chd_mult_fac WSA_CHD_MULT_FAC] [-wsa_chd_arg_fac WSA_CHD_ARG_FAC]
                [-wsa_chd_power WSA_CHD_POWER] [-wsa_c5 WSA_C5] [-psi_vslow PSI_VSLOW]
                [-psi_vfast PSI_VFAST] [-psi_eps PSI_EPS] [-psi_width PSI_WIDTH]
                [-rhofast RHOFAST] [-tfast TFAST]
```

SWiG: Current options

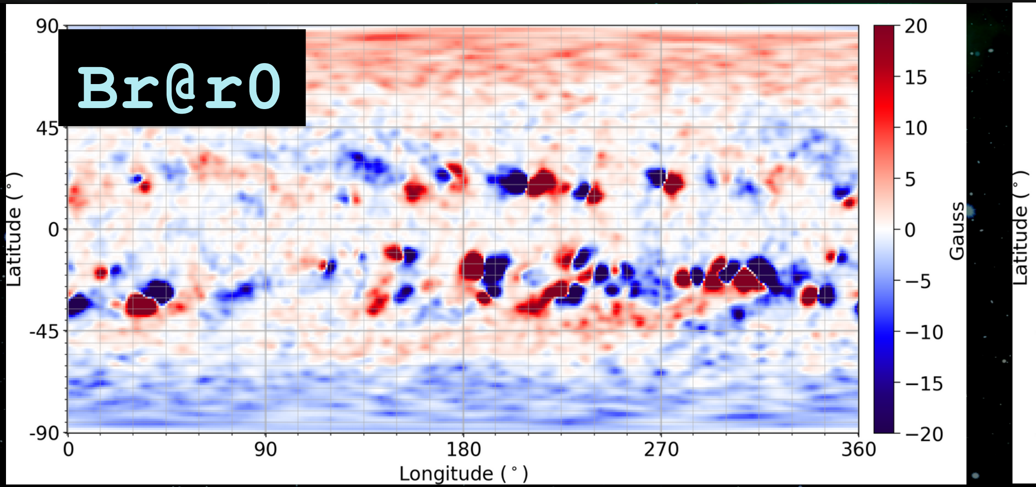
```
swig.py [-h] [-oidx OIDX] [-rundir RUNDIR] [-np NP] [-gpu]
        [-sw_model SW_MODEL] [-rss RSS] [-r1 R1] [-noplot]
        input_map
```

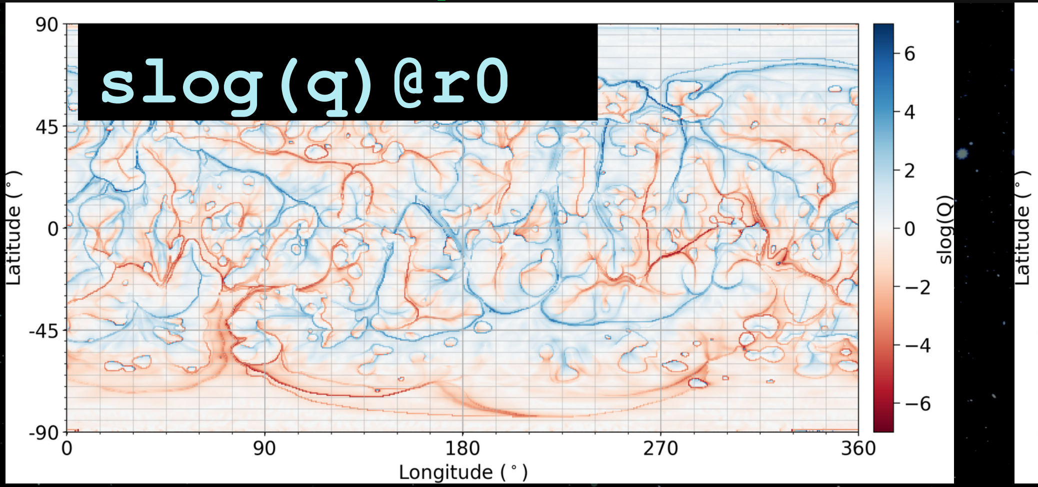
```
swig_run_multiple_maps.py [-h] [-outdir OUTDIR]
                          [-swig_path SWIG_PATH] [-np NP] [-gpu]
                          [-sw_model SW_MODEL] [-rss RSS]
                          [-r1 R1] [-noplot] input_directory
```

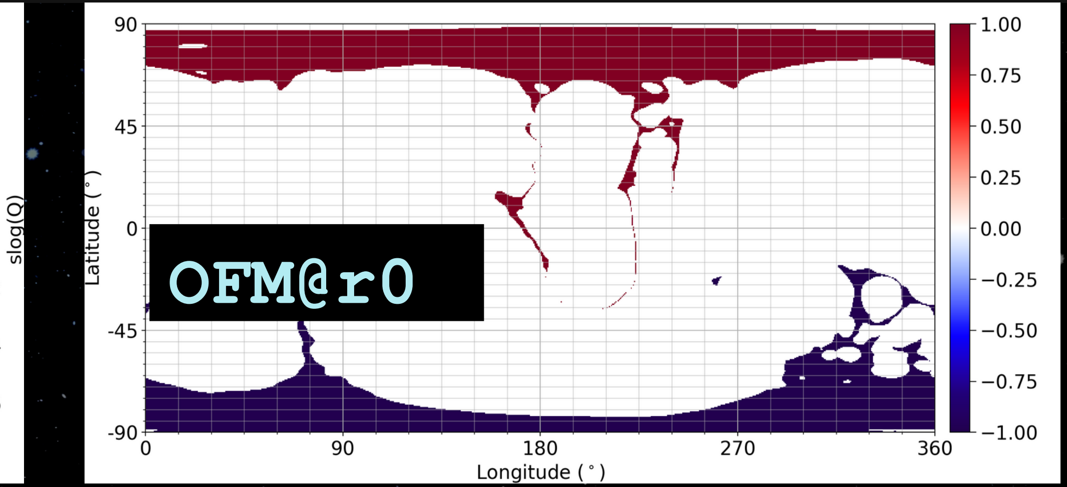
More options can be added later (pull requests welcome!):

- Resolution (pf, tracings), rss overlap, etc.
- Solar wind model options:

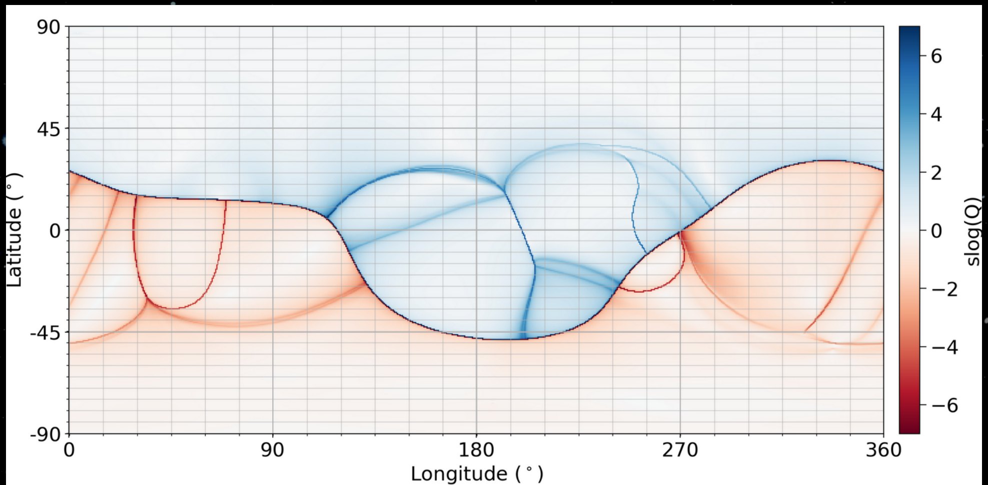
SWiG: Outputs and Plots


Br@r0

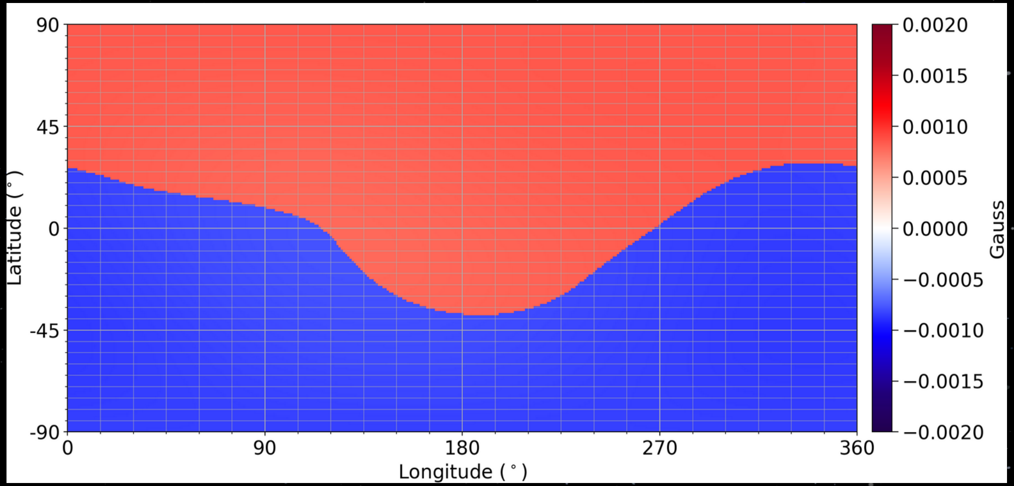

slog(q)@r0


OFM@r0

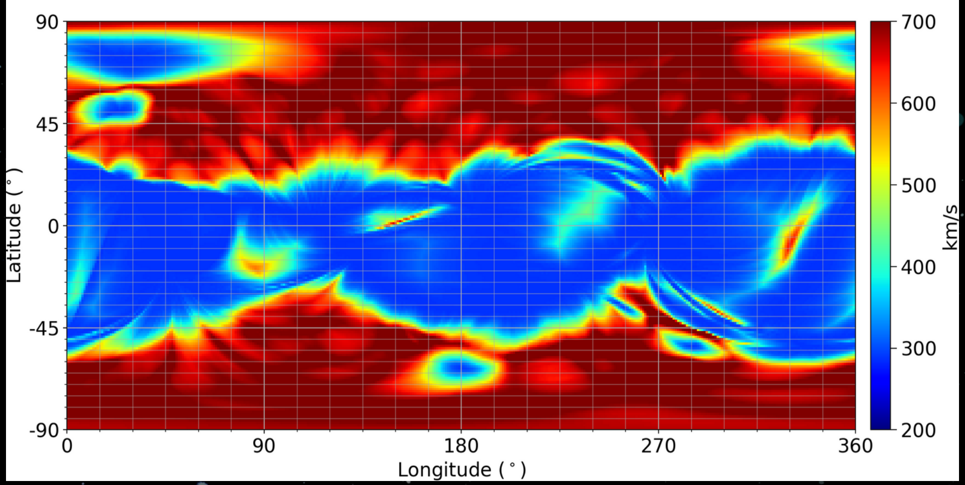
slog(q)@rss



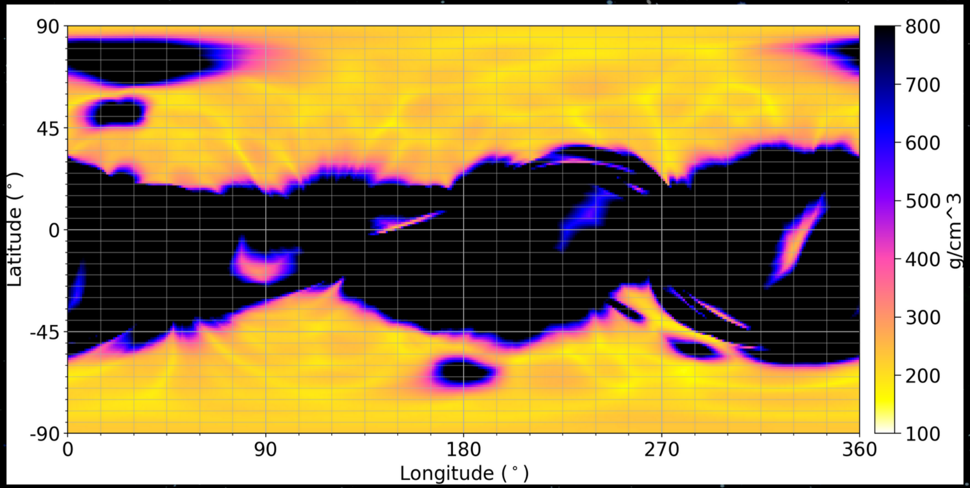
Br@r1



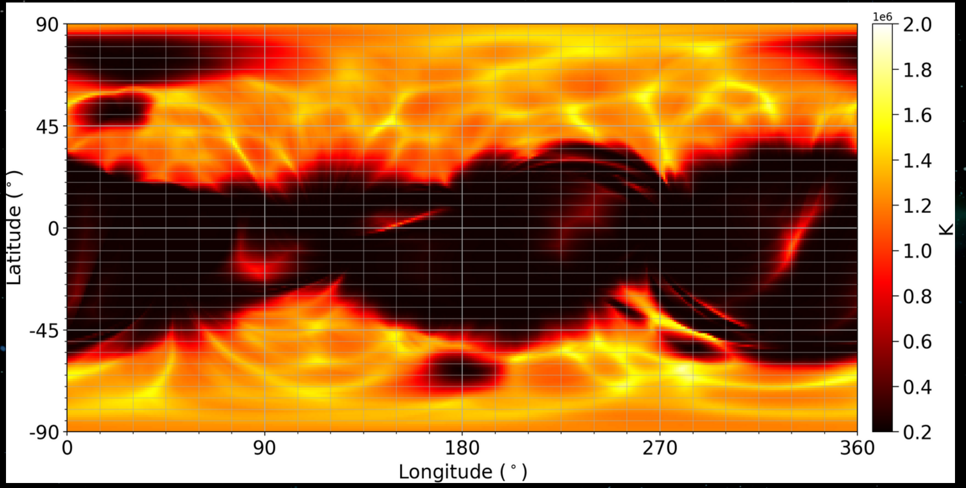
Vr@r1



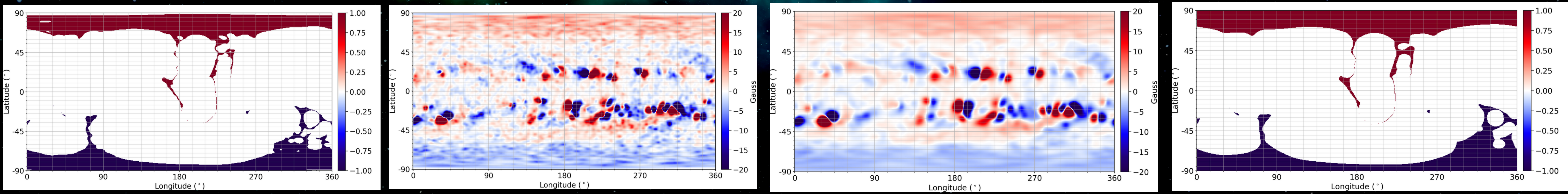
rho@r1



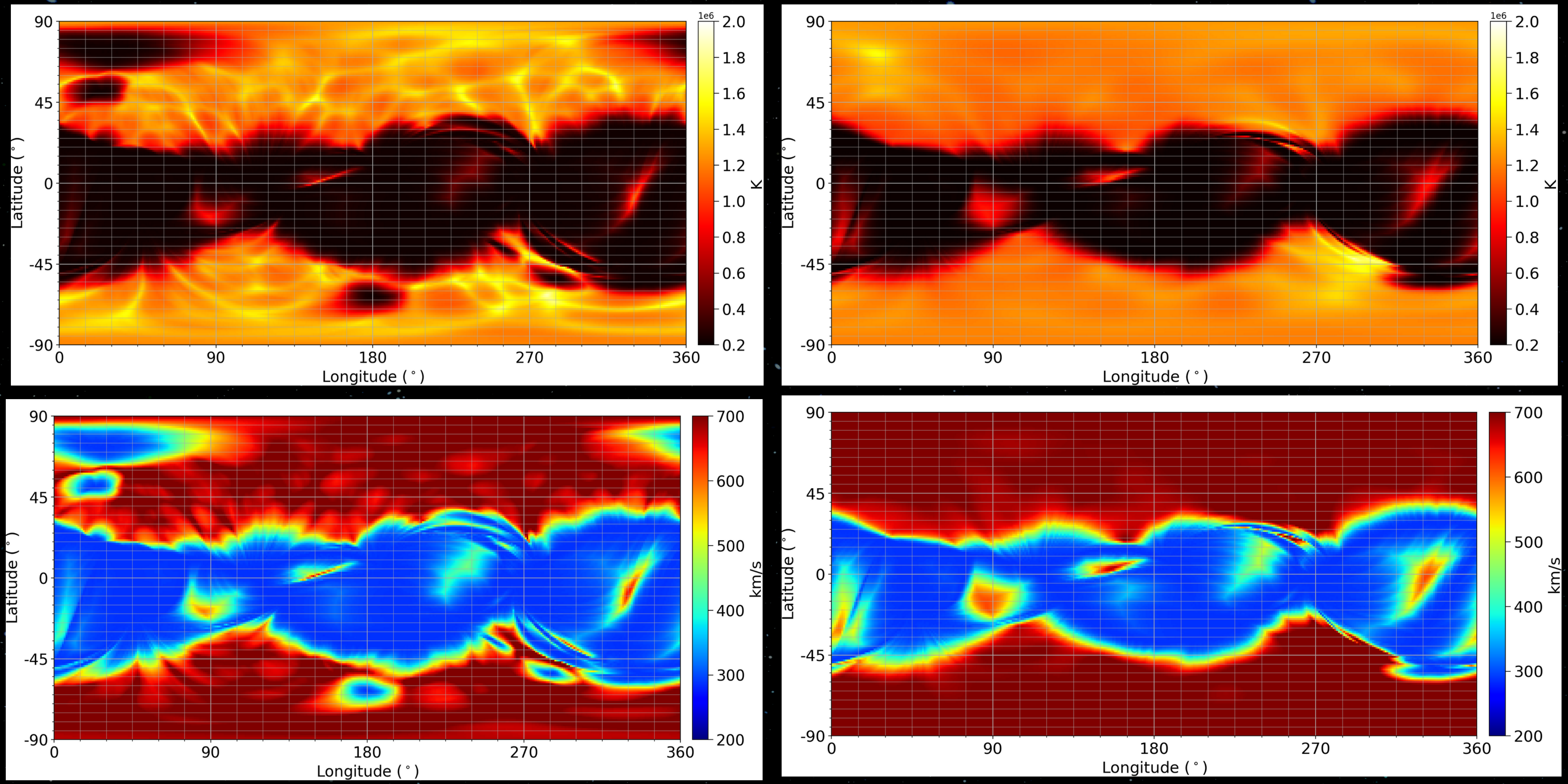
t@r1



SWiG: Effects of Smoothing



Smoothfac = 0.5



Smoothfac = 2.0

SWiG: Installation

- Installation guides for:
  - Linux
  - Mac
  - Windows with WSL

SWiG Assignment

- Run SWiG.py on at least one processed map from yesterday to create solar wind boundary conditions for tomorrow

predsci.com/~caplanr/swqu_workshop

BREAK
TIME!!!