Computational Aspects of OpenGGCM

Kai Germaschewski

Space Science Center & Department of Physics
University of New Hampshire

January 17, 2012

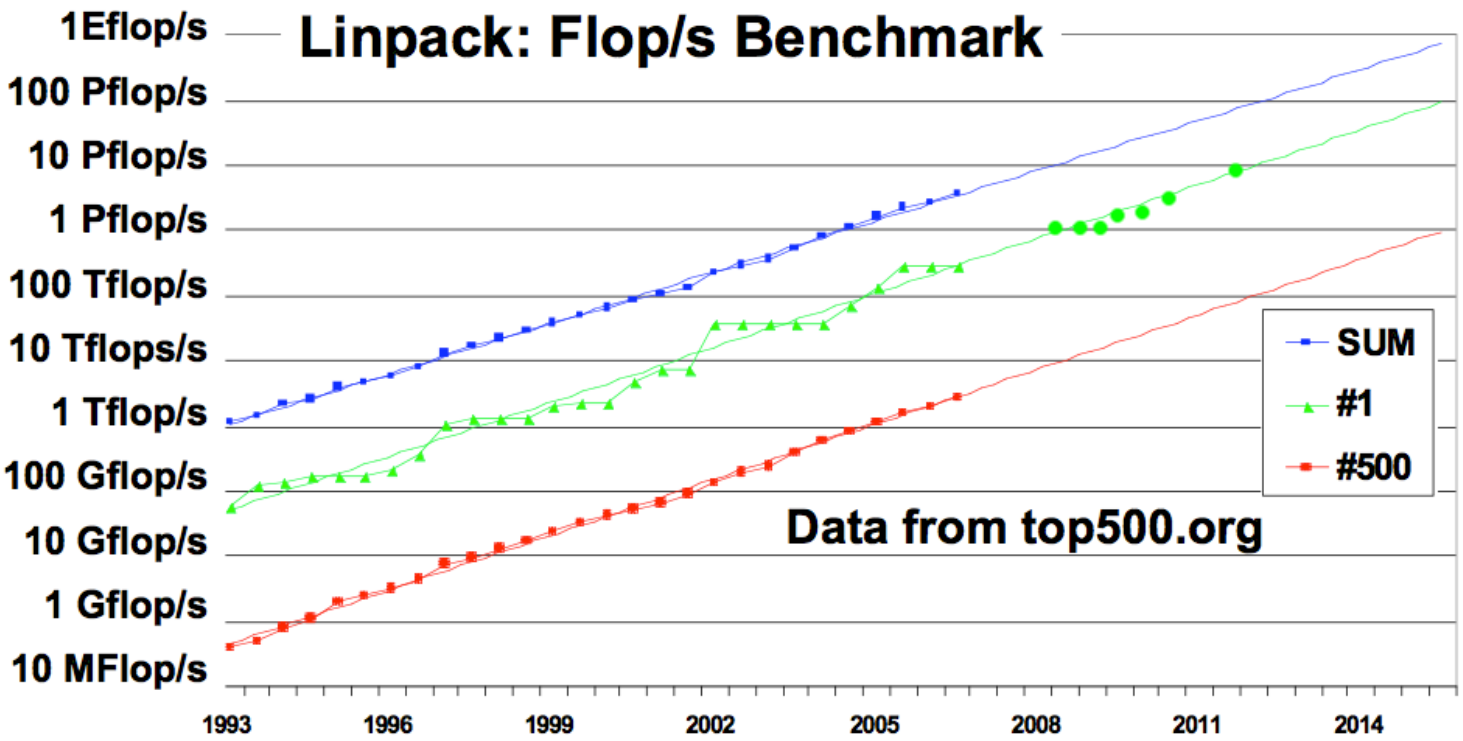
Collaborators: A. Bhattacharjee, M. Kuznetsova, J. Raeder, B. Sullivan

UNIVERSITY of NEW HAMPSHIRE

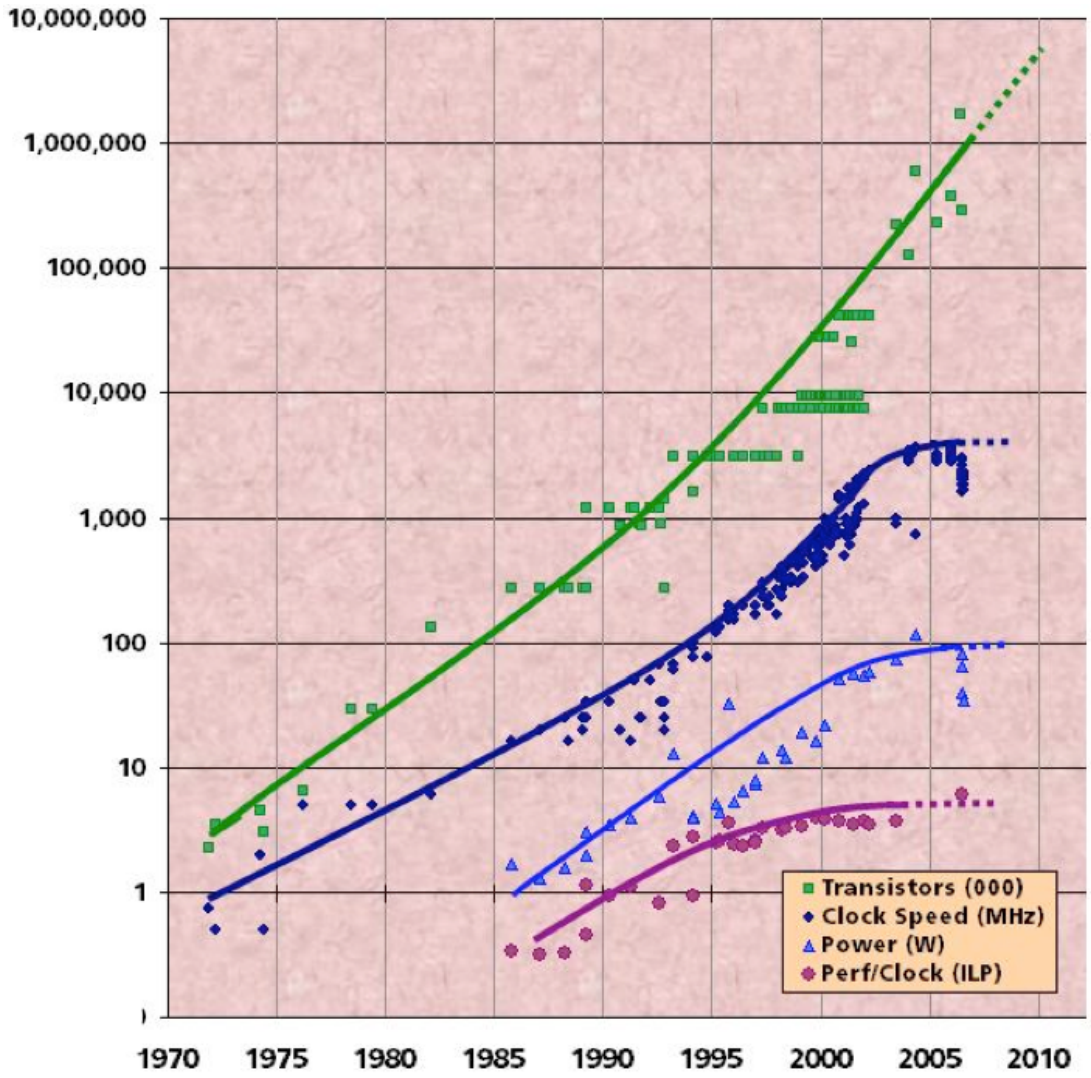
Outline

1. High Performance Computation
   - Introduction
   - Automatic code generation in OpenGGCM
2. Magnetopause reconnection in global models
   - Uniform resistivity
   - Hall MHD
3. Summary

Computer performance keeps growing exponentially

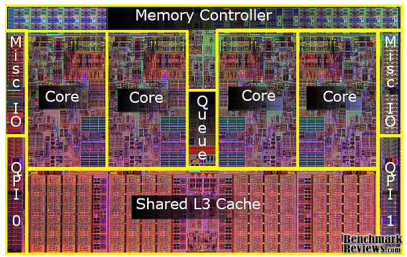


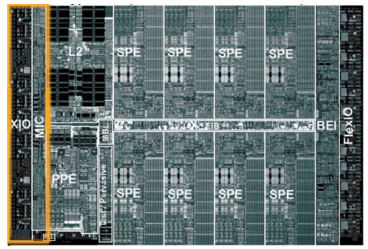
Moore's Law is alive and well, but...

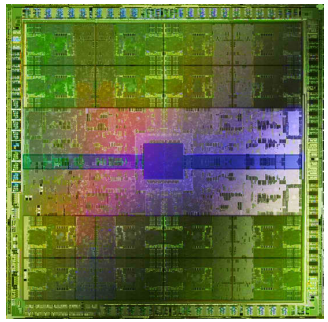


single-core performance has saturated!

Heterogeneous computing







	multi-core	Cell	GPGPU
OS runs on	any core	PPU	host CPU
main code on	any core	PPU	host CPU
comput. kernels on	any core	SPUs	GPGPU MP
memory architecture	main memory caches	main memory per-SPU local store	main memory GPGPU memory shared memory
data movement	transparent	DMA for moving data	explicit
SIMD	explicit	explicit	automatic
threads	few	no (?)	many
efficient programming	hard	hard	hard

Heterogeneous computing

Twelve Ways to Fool the Masses When Giving Performance Results on Parallel Computers
David H. Bailey
June 11, 1991
Ref: Supercomputing Review, Aug. 1991, pg. 54--55

6. Compare your results against scalar, unoptimized code on Crays.

It really impresses the audience when you can state that your code runs several times faster than a Cray, currently the world's dominant supercomputer. Unfortunately, with a little tuning many applications run quite fast on Crays. Therefore you must be careful not to do any tuning on the Cray code. Do not insert vectorization directives, and if you find any, remove them. In extreme cases it may be necessary to disable all vectorization with a command line flag. Also, Crays often run much slower with bank conflicts, so be sure that your Cray code accesses data with large, power-of-two strides whenever possible. It is also important to avoid multitasking and autotasking on Crays --- imply in your paper that the one processor Cray performance rates you are comparing against represent the full potential of a $25 million Cray system.

Automatic code generation

Lesson learned from porting OpenGGCM to the Cell processor:

Programming heterogeneous architectures and achieving high performance is *hard*.

$$\Longrightarrow \text{Just let the computer do the work.}$$

Automatic code generation lets you input your finite-difference / finite-volume equations in near symbolic form as a stencil computation, and then does all the work the generate efficient code for Cell / SSE2 / GPUs.

Example: $\partial_t \rho = -\nabla \cdot (\rho \vec{v})$

```
rhs_RHO = - Divg(ZIP(RHO, V))
```

What does code generation do?

Example: $\partial_t \rho = -\nabla \cdot (\rho \vec{v})$

```
V = 1./RHO * P
rhs_RHO = - Divg(ZIP(RHO, V))
```

```
  FLD3(r,jx,jy,jz,RHO) =
(
  -((0.5*(RHO(x,jx+0,jy+0,jz+0)*
          (P0(x,jx+1,jy+0,jz+0) / RHO(x,jx+1,jy+0,jz+0)) +
          RHO(x,jx+1,jy+0,jz+0) *
          (P0(x,jx+0,jy+0,jz+0) / RHO(x,jx+0,jy+0,jz+0))) -
     0.5*(RHO(x,jx-1,jy+0,jz+0) *
          (P0(x,jx+0,jy+0,jz+0) / RHO(x,jx+0,jy+0,jz+0)) +
          RHO(x,jx+0,jy+0,jz+0) *
          (P0(x,jx-1,jy+0,jz+0) / RHO(x,jx-1,jy+0,jz+0)))) /
    (CRD0f(jx+1) - CRD0f(jx+0))
    +
   ...
```

Automatic code generation

Symbolic manipulation

Having the equations available in their natural (discretized) form allows for easy symbolic manipulation, e.g. find non-zero structure of the Jacobian, etc.

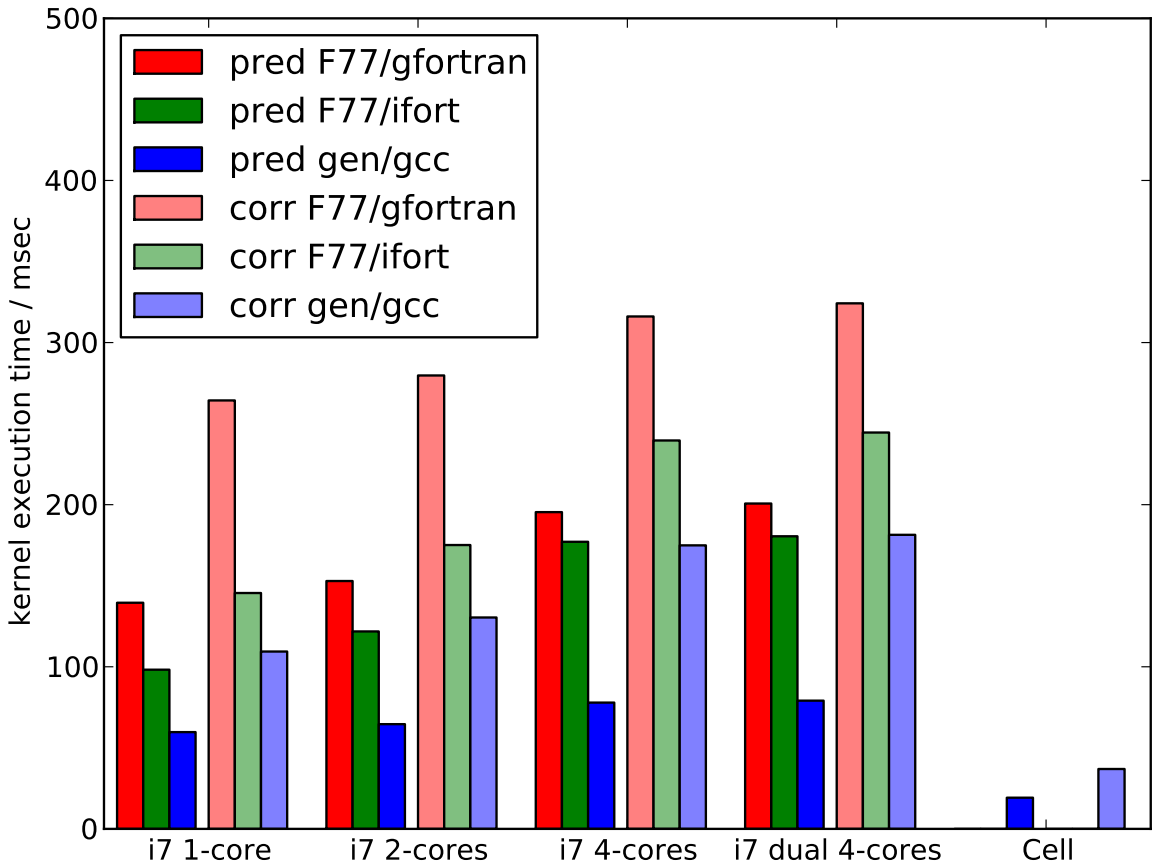
High performance

Conversion to actual computer code can automatically adapt to geometry, parameters, hardware architectures to obtain optimal performance.

Productivity

Only one version of the code needs to be maintained. Changing the underlying equations (physics) and numerics is simplified because they are abstracted out in near-mathematical form from their actual implementation.

Performance gains (SIMD, Cell)



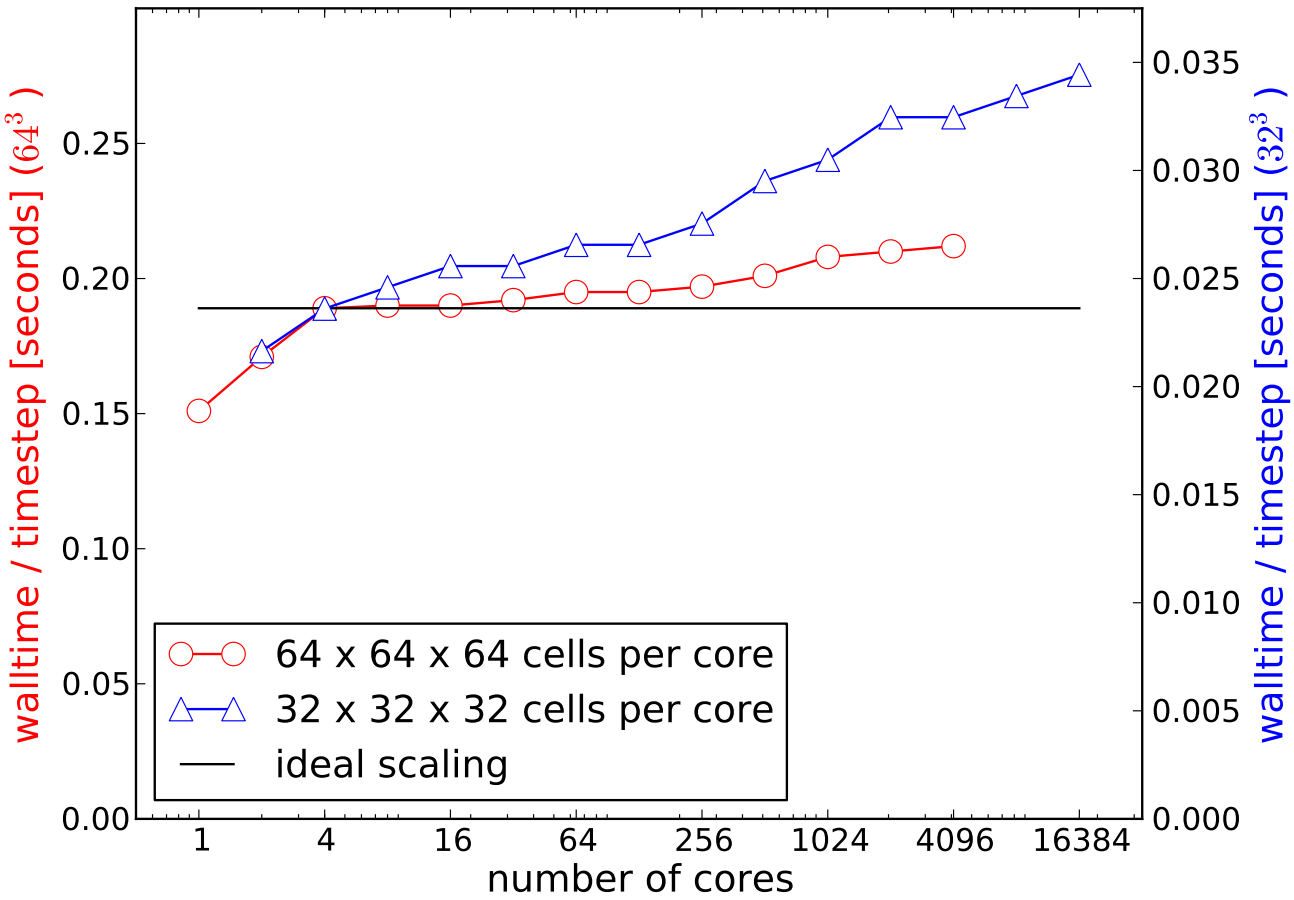
speed-up: 2.3× on multi-core, 200× (?!?) on the Cell (predictor)

Performance gains (GPU, prelim)

```
ymaskn_f              57.964   11 5.26945
primvar1_f           206.048   11 18.7316
primbb_f              29.344   11 2.66764
zmaskn_f              47.821   11 4.34736
pushpred_f           1898.37   10 189.837  <--
pushpred_b           222.927   11 20.2661  <--
pushfluid1_0b         11.553   11 1.05027
pushfluid1_123b       60.427   11 5.49336
pushfluid1_4b         51.231   11 4.65736
bcc_xj_b              17.935   11 1.63045
yzmaskn_b              12.91   11 1.17364
calce1_b              59.634   11 5.42127
bpush1_b               8.982   11 0.81654
pushpred_b_all       355.949   10 35.5949
```

The generated GPU code achieves a speed-up of ~ 9.5 over the original code on a nvidia Fermi card vs. intel i7 single core. Considering this was achieved without any attempt at specific GPU optimizations (e.g., using shared memory), that's quite promising.

Parallel scalability



parallel efficiency: 89% for 64^3 local problem, 69% for 32^3.

Magnetopause Reconnection

Magnetospheric plasma is rather tenuous and collisionless, but numerical models typically use resistive MHD.

Questions

- How does resistive 3D reconnection scale in a global code?
- Do we observe the plasmoid instability in a global code?

OpenGGCM runs were performed at UNH, BATSRUS runs at NASA's CCMC (Community Coordinated Modeling Center).

Parameters:

solar wind: $B_z = -5\,nT$, $v_x = -400\,km/s$, $n = 5\,cm^{-3}$

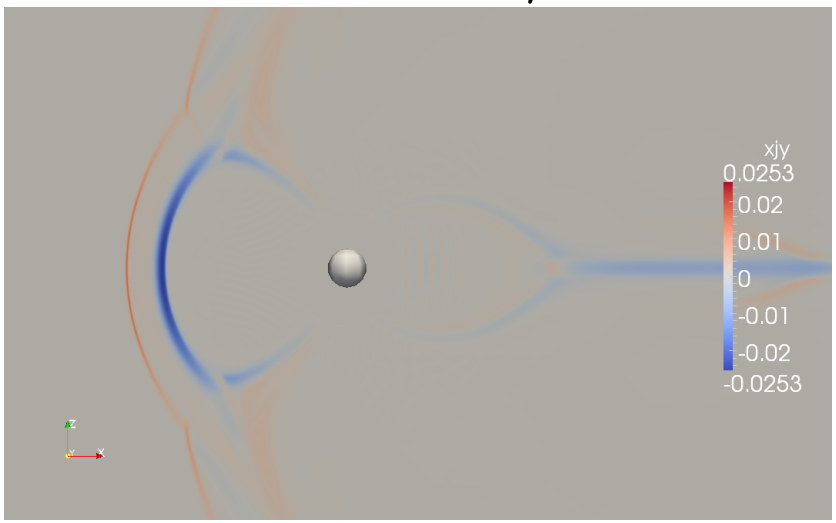
uniform resistivity: Lundquist number $S = 500 \ldots 10000$

Time evolution of the magnetosphere pressure

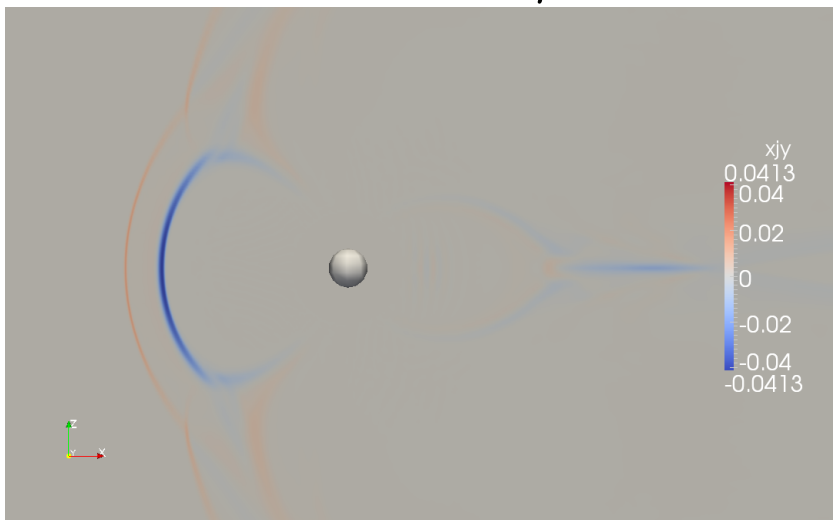
(Loading...)

Magnetopause with varying resistivity

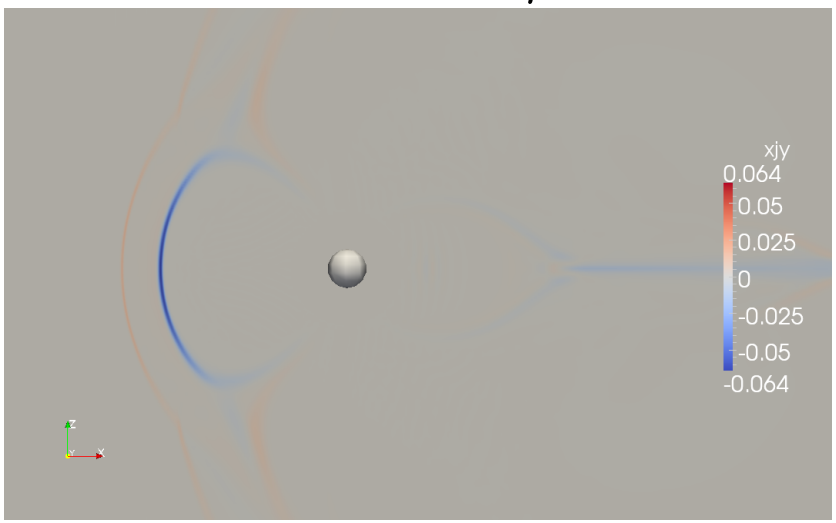
$\eta = 5 \times 10^4$



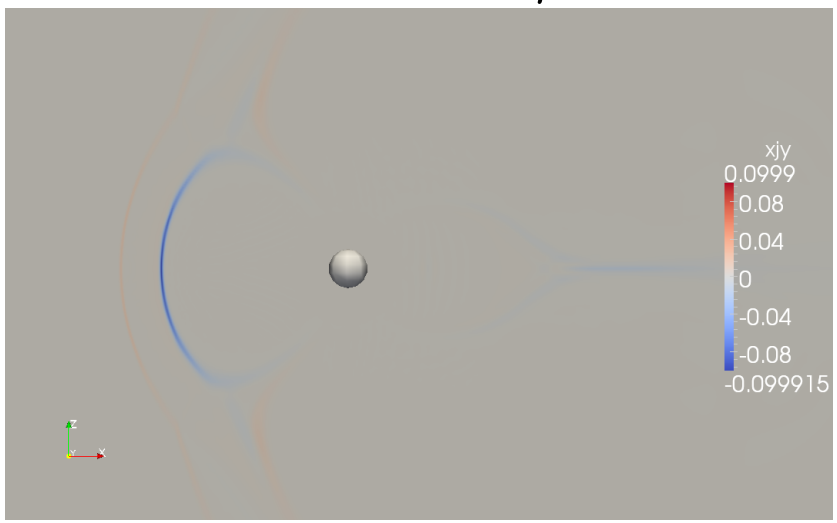
$\eta = 2 \times 10^4$



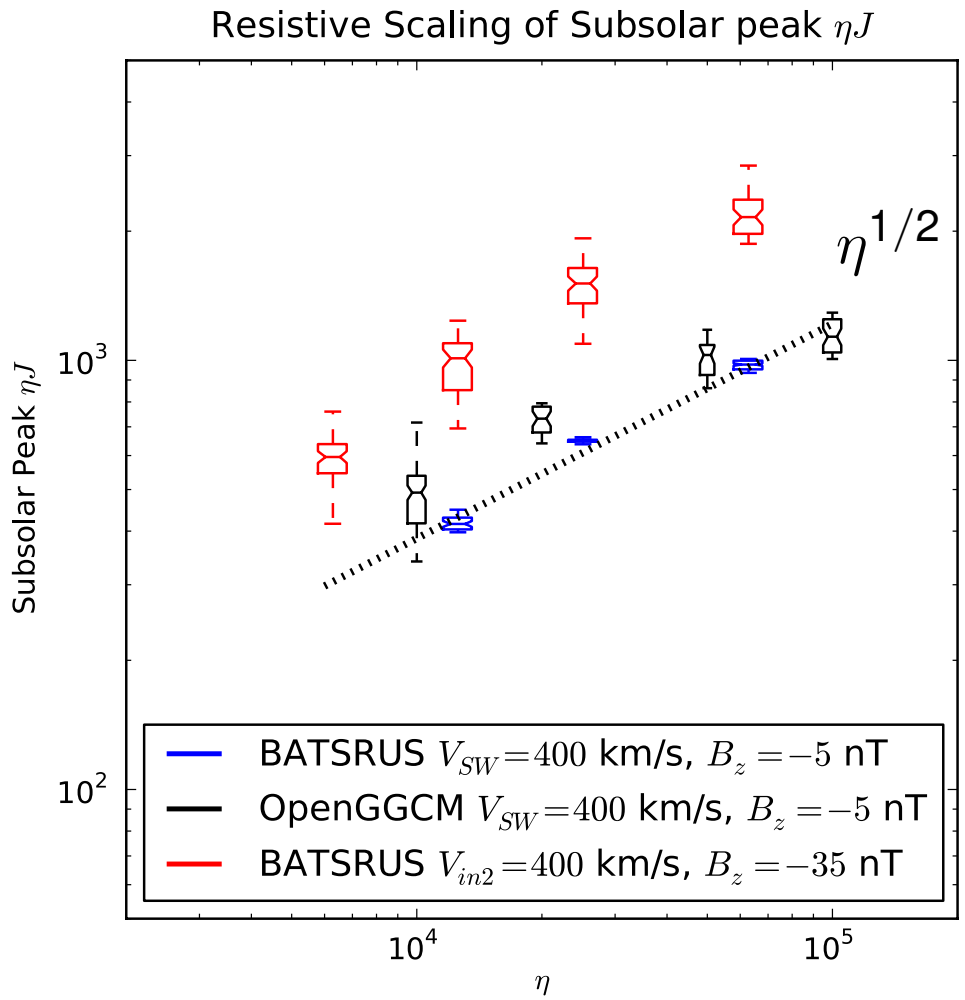
$\eta = 1 \times 10^4$



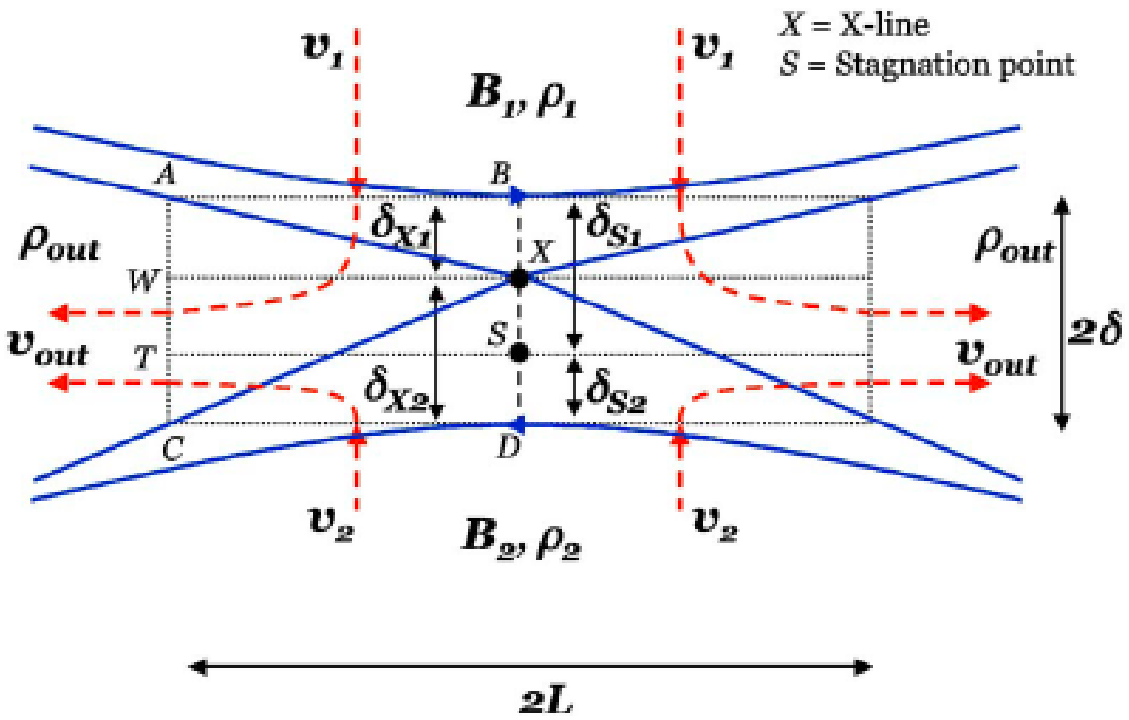
$\eta = 5 \times 10^3$



Sweet-Parker scaling for electric field



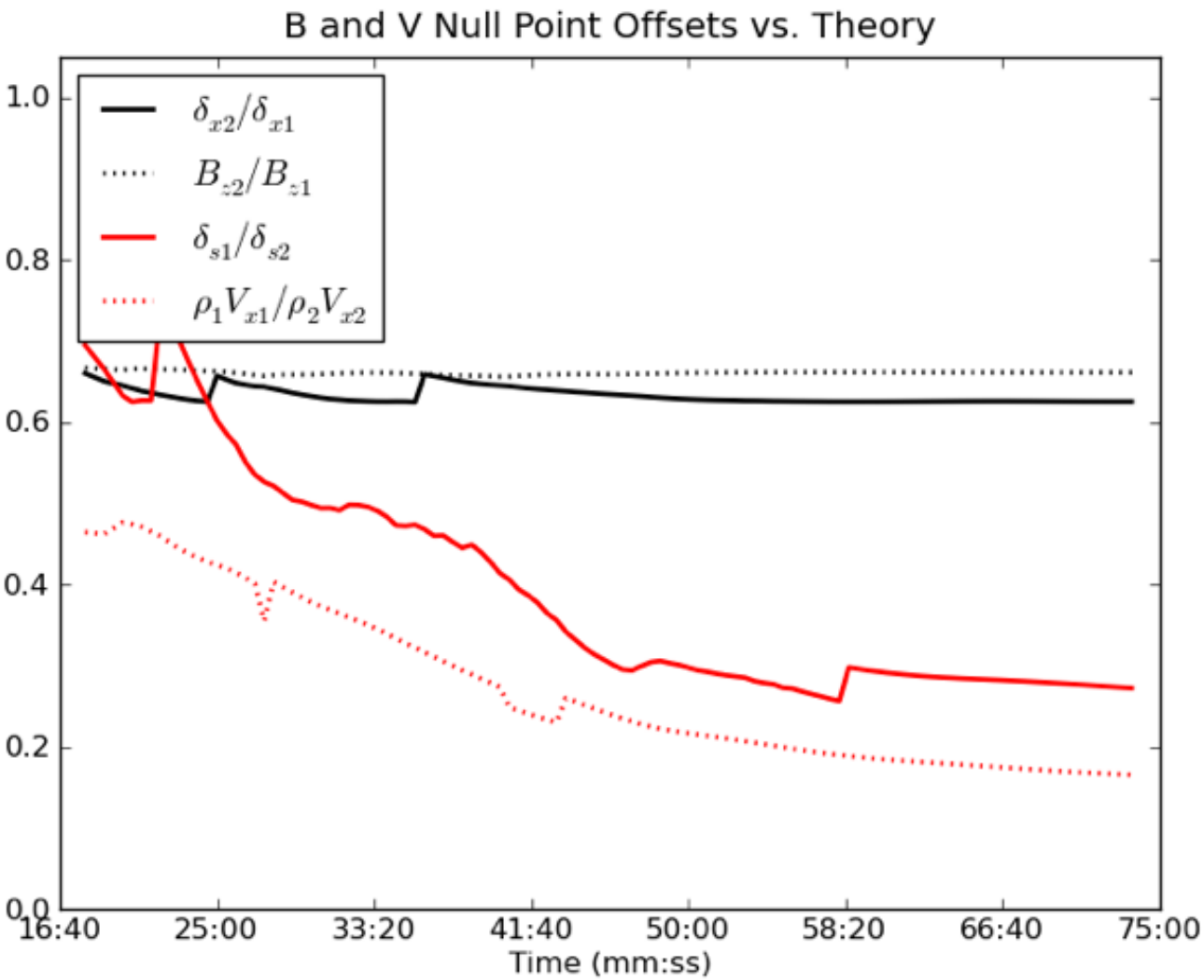
Theoretical predictions of X-point and flow stagnation point offset



$$\frac{\delta_{X2}}{\delta_{X1}} \approx \frac{B_2}{B_1} \qquad \frac{\delta_{S2}}{\delta_{S1}} \approx \frac{\rho_1 v_{x1}}{\rho_2 v_{x2}}$$

Cassak, Shay 2007

Results vs. 2D Cassak-Shay theory

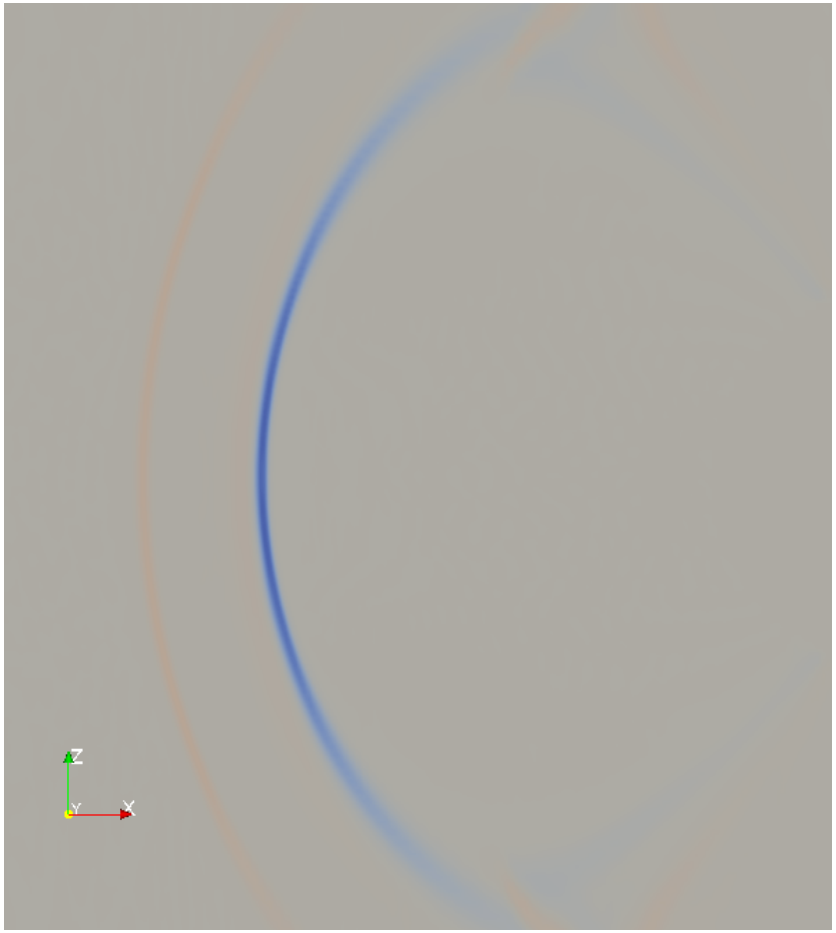


Hall Reconnection

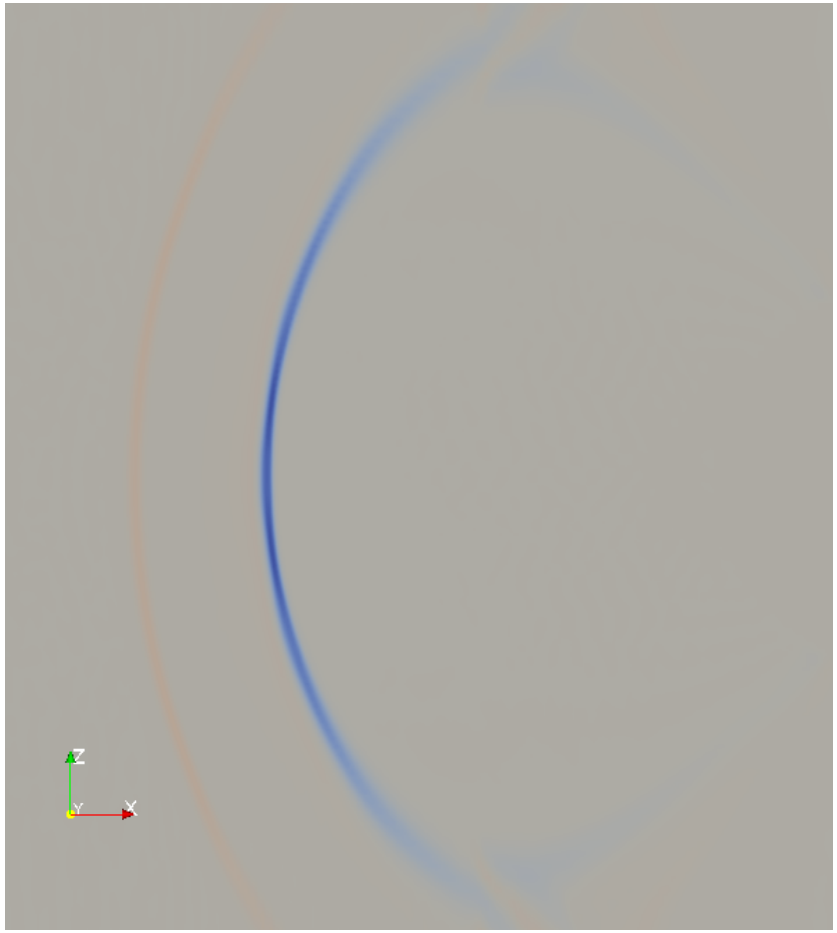
Loading c1.mp4

Current sheet structure resistive vs Hall MHD

$d_i = 0$

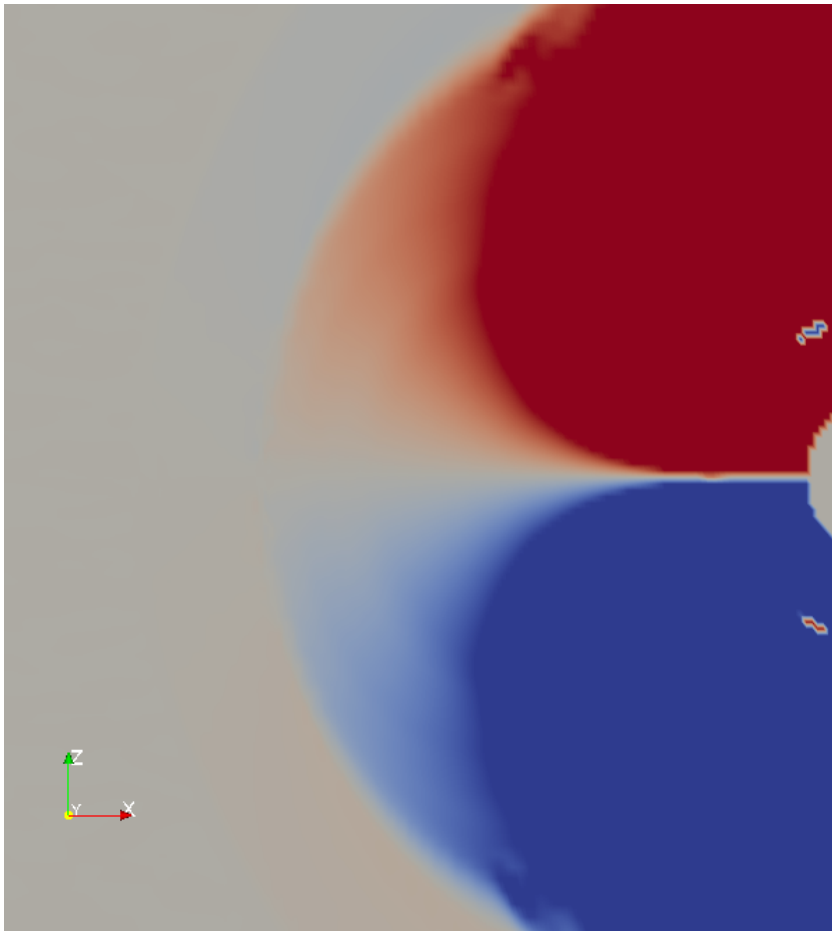


$d_i = 0.5 R_E$

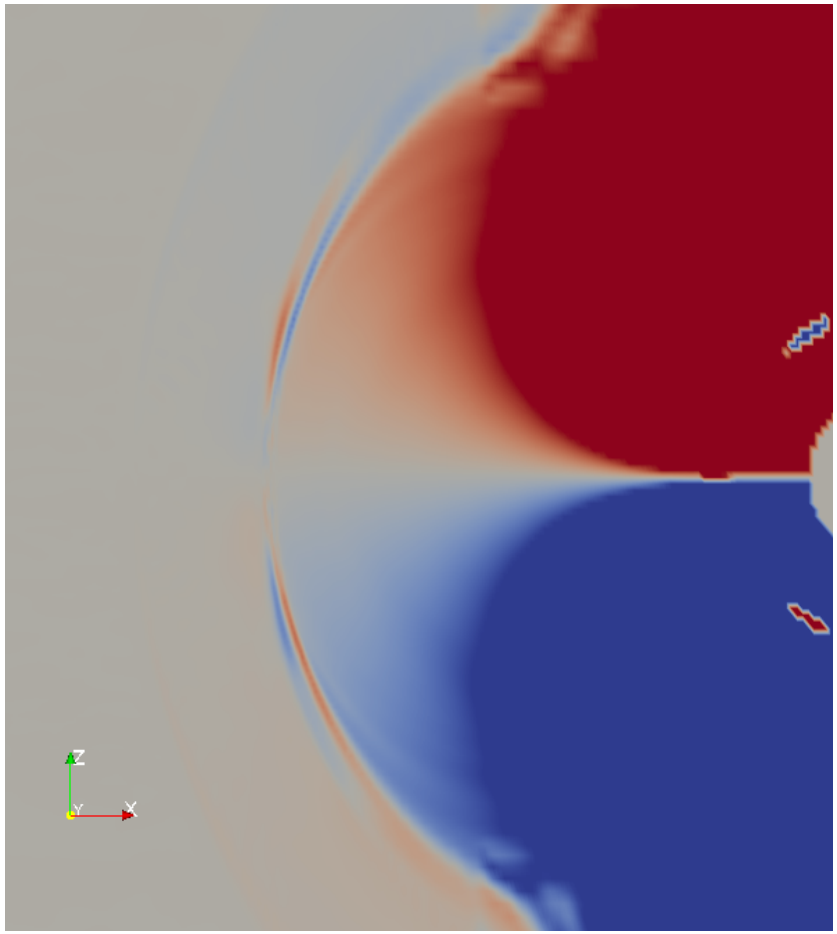


Out-of-plane magnetic field resistive vs Hall MHD

$d_i = 0$

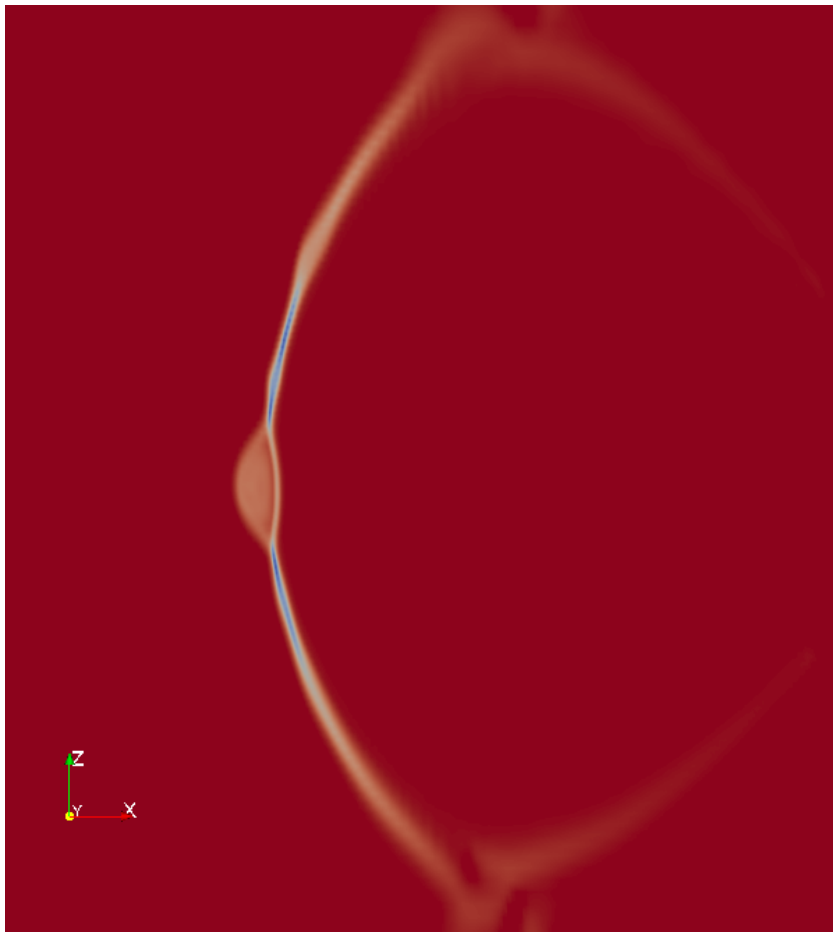


$d_i = 0.5 R_E$

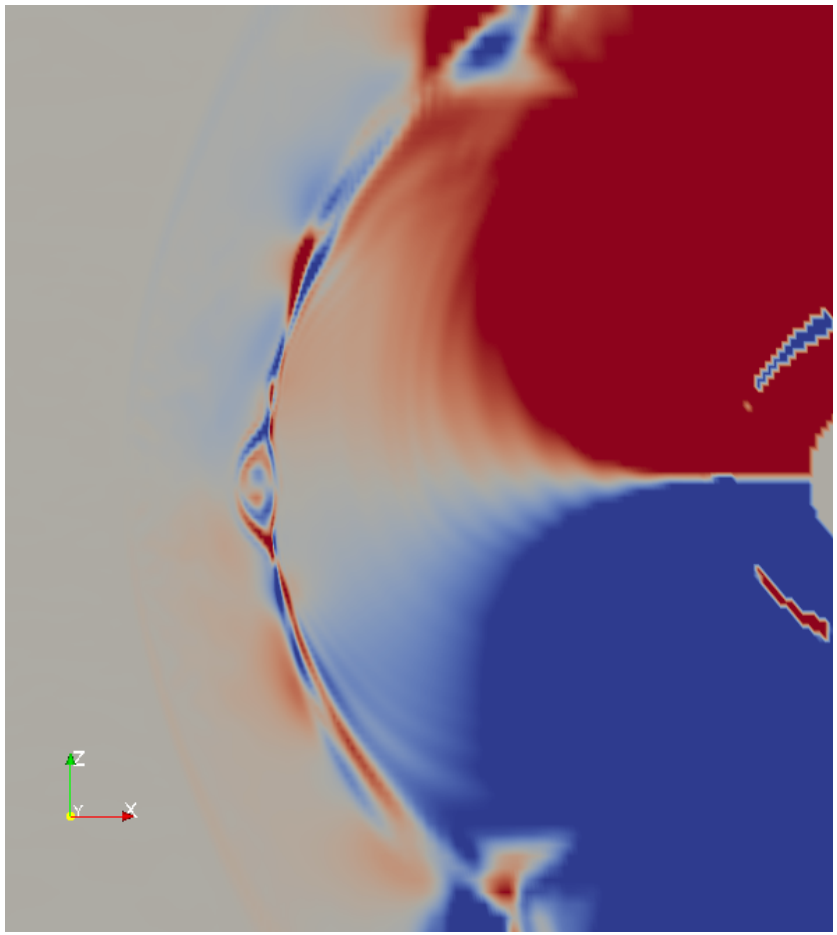


Quadrupolar out-of-plane magnetic field with Hall

J_y



B_y



Hall MHD – Current density

Loading sw6c.mp4

$d_i = 0.5 R_E$, $\delta_{SP} \approx 0.1 R_E$

Summary

- Advances in computing capabilities allow unprecedented opportunities (and challenges) for numerical modelling.
- Automatic code generation is a promising approach to reap benefits while avoiding some of the pain.
- Reconnection in global magnetosphere models requires improved models and poses many open questions.