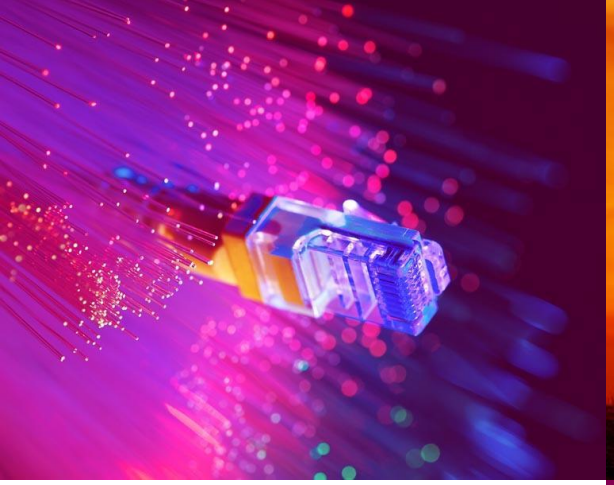


Vlasiator - enabling large scale hybrid-Vlasov simulations

Sebastian von Alfthan(1), Urs Ganse(2), Yann Pfau-Kempf (2), Minna Palmroth(2)

1) CSC- IT Center for Science
2) University of Helsinki

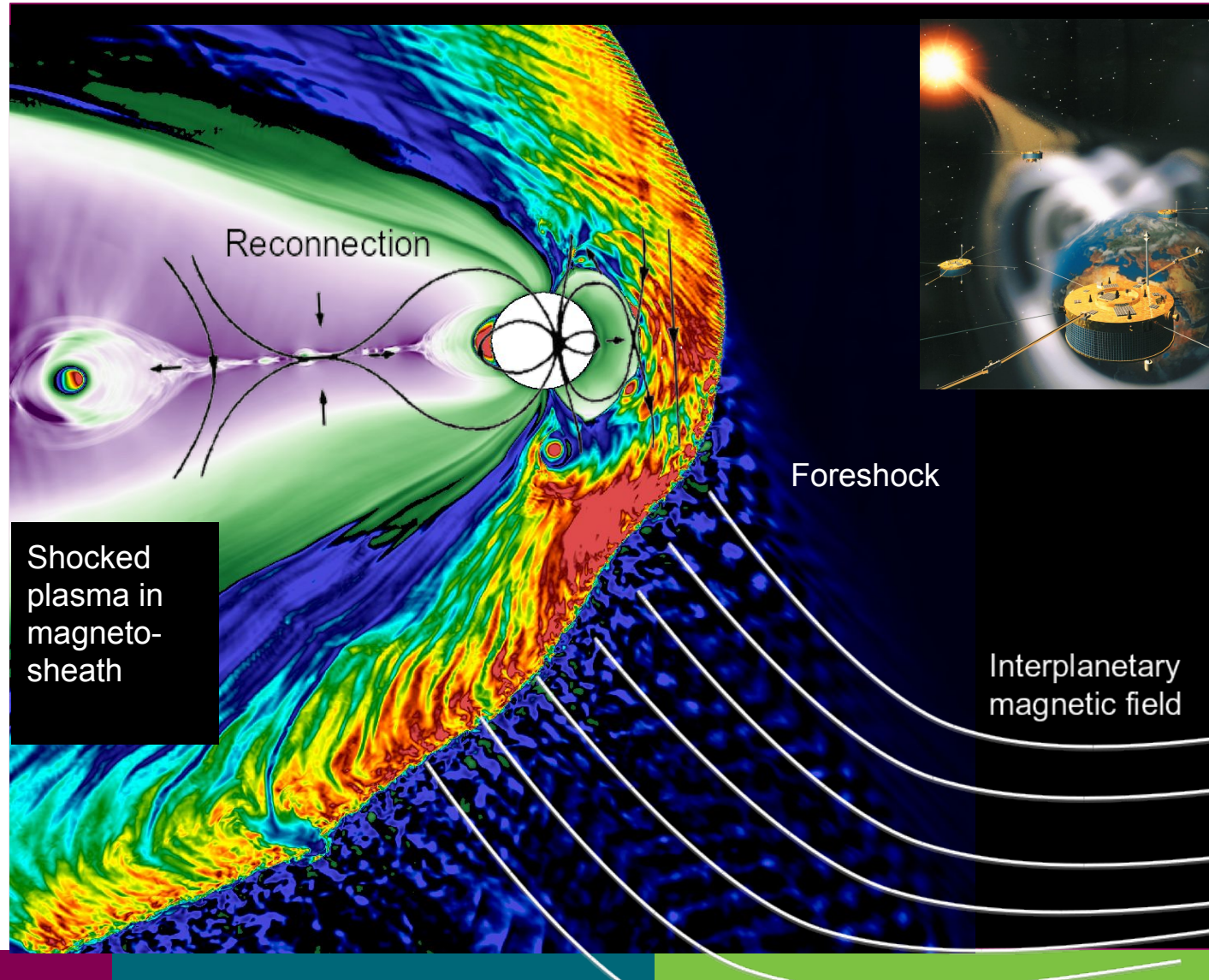
CSC – Suomalainen tutkimuksen, koulutuksen, kulttuurin ja julkishallinnon ICT-osaamiskeskus

Outline

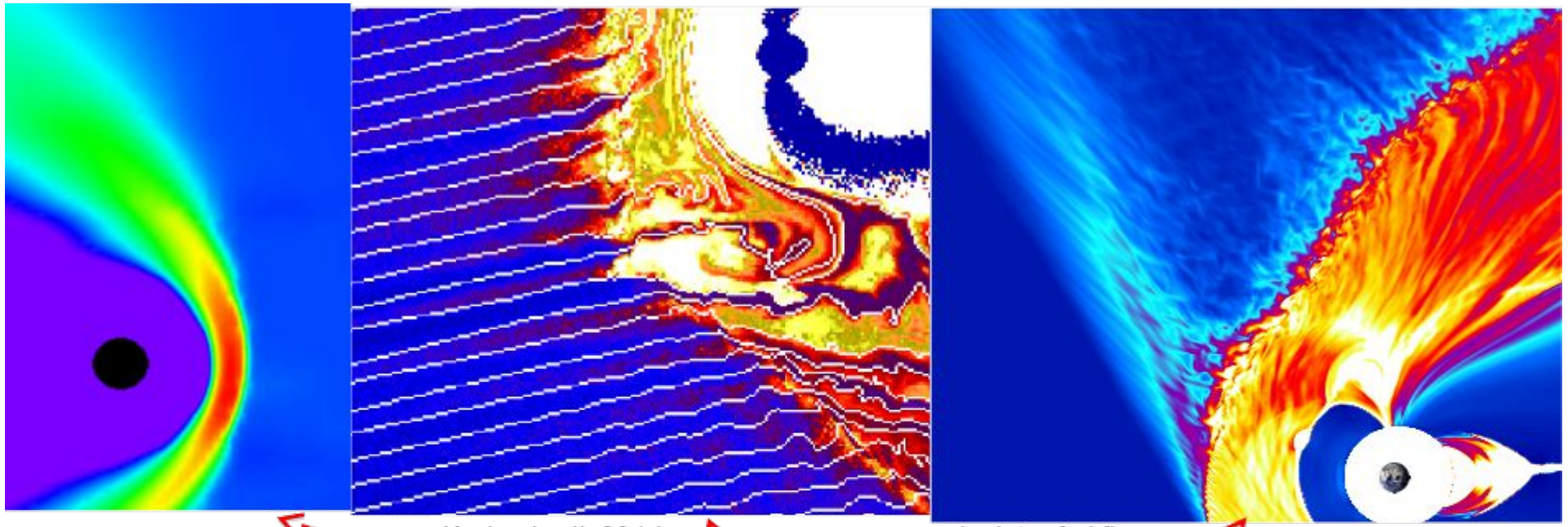
- Introduction
- Hybrid Vlasov &Computational challenges
- Vlasiator
  - 6D sparse grid
  - Conservative Semi Lagranging Vlasov solver
  - Performance
- KNL optimizations & experiences on KNL
- Recent developments

VLASIATOR

- Vlasiator (vlasiator.fmi.fi) simulates plasma in near-earth space
- Hybrid-Vlasov description
  - Electrons: Charge-neutralizing fluid
  - Ions: 6D distribution function
  - Supports multiple ion species
  - Multi-temperature ion physics
  - Noise-free
- Science
  - Simulations of Earth's magnetoshpere
  - Also shocks, and possibly other settings



Global modelling techniques

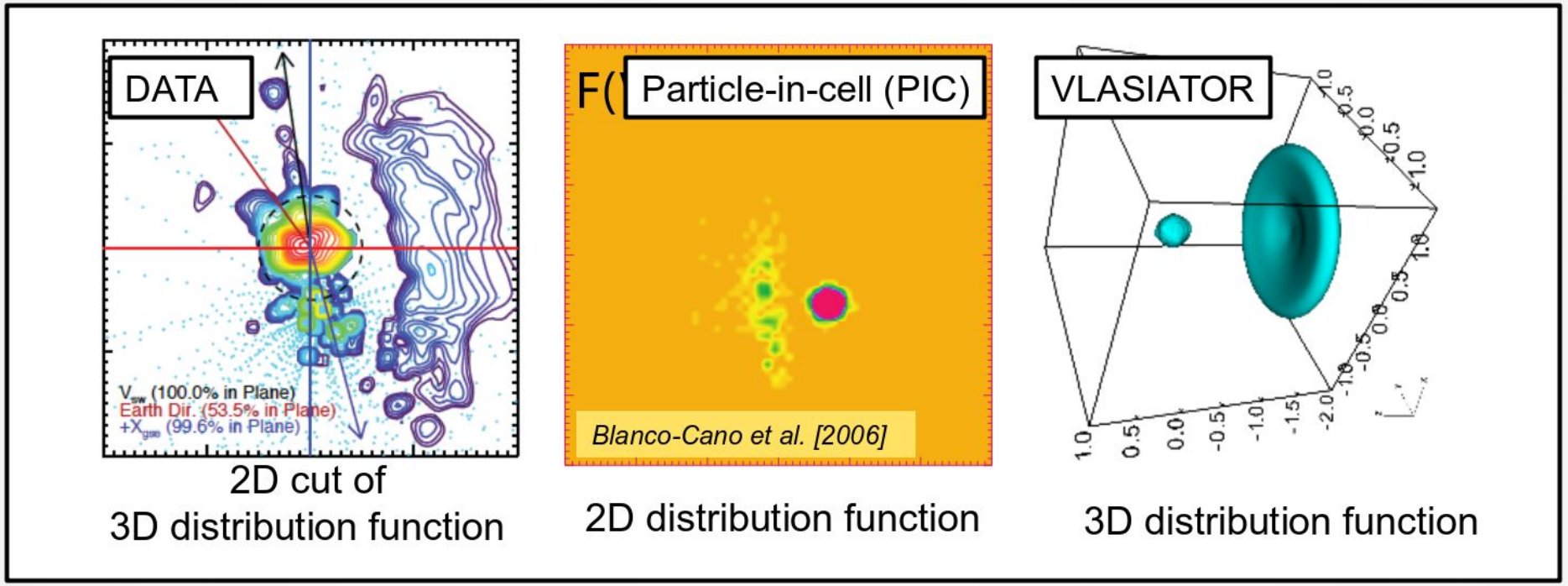


Karimabadi, 2014

vlasiator.fmi.fi

	Global fluid (MHD)	Hybrid particle-in-cell	Hybrid-Vlasov (Vlasiator)
Ions	Fluid	Particles	Distribution functions
Electrons	Fluid	Fluid	Fluid
Run time	Real-time to weeks	Weeks	Weeks – months
Scale of applications	Fluid scale (1000 km) Now: Solar system	Ion kinetic effects (10 – 1000 km) Global = Solar wind, magnetosphere (+ ionosphere)	

Global modelling techniques

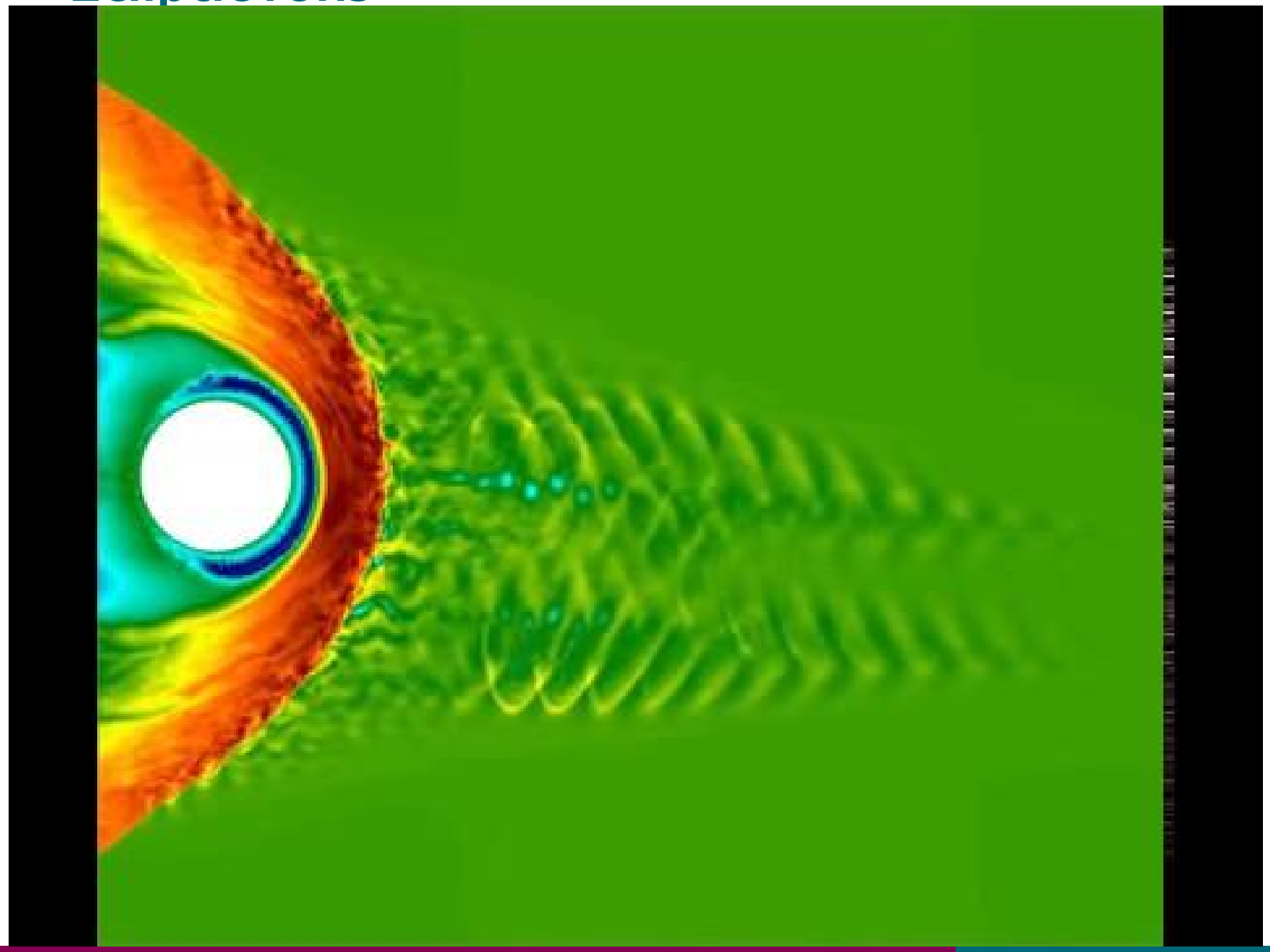


	Global fluid (MHD)	Hybrid particle-in-cell	Hybrid-Vlasov (Vlasiator)
Ions	Fluid	Particles	Distribution functions
Electrons	Fluid	Fluid	Fluid
Run time	Real-time to weeks	Weeks	Weeks – months
Scale of applications	Fluid scale (1000 km) Now: Solar system	Ion kinetic effects (10 – 1000 km) Global = Solar wind, magnetosphere (+ ionosphere)	

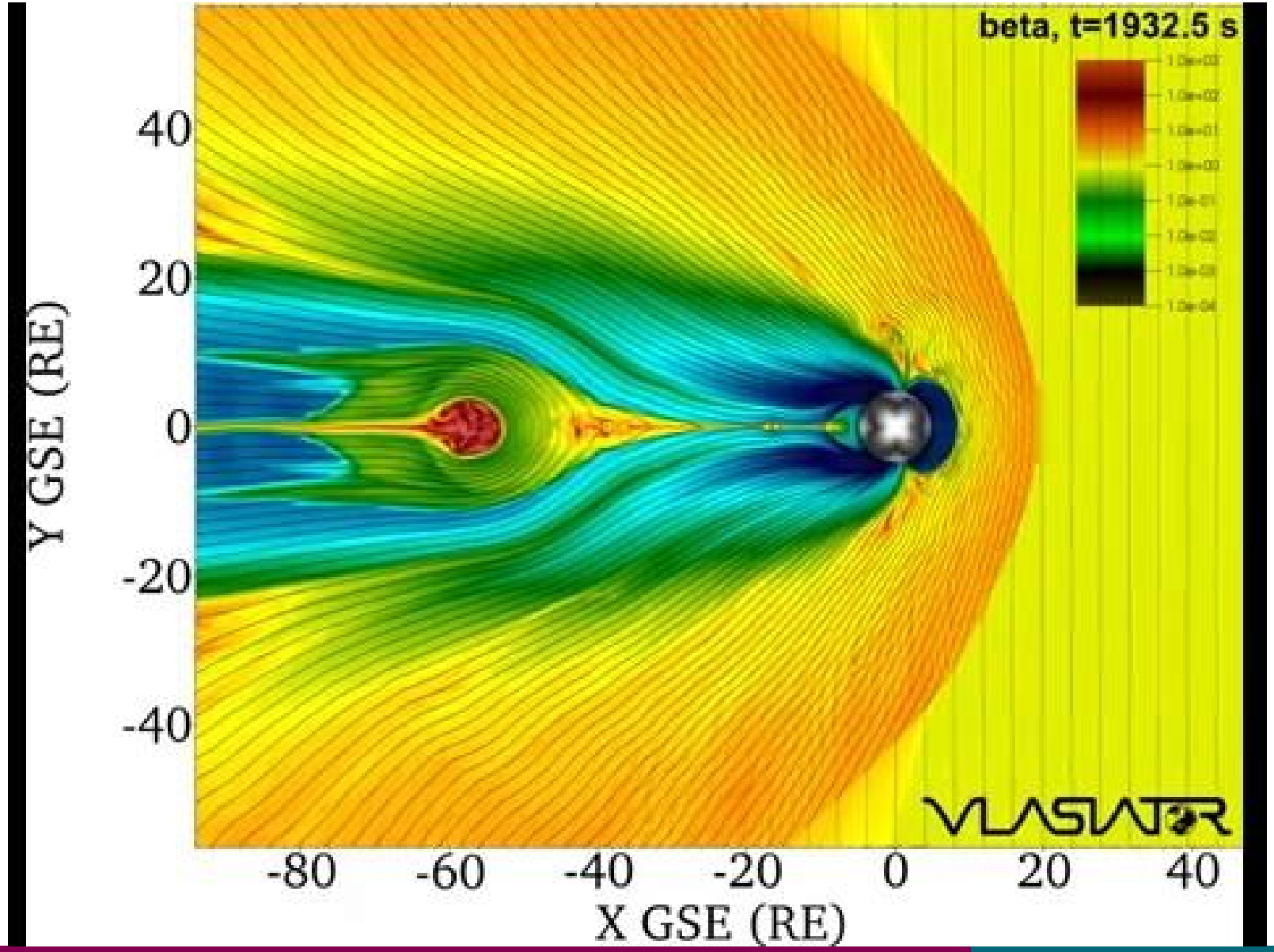
Recent compute campaigns

- 2013
  - PRACE access, 30 MCPUh on Hermit (Cray XE6)
  - European suprcomputing infrastructure: http://www.prace-ri.eu/
  - 5D simulations in XY plane
- 2014
  - Pilot usage of Sisu (Cray XC40)
  - Kinetic scale 5D simulation in XY and XZ plane
- 2015
  - PRACE : 24 MCPUh on Hornet (Cray XC40)
  - CSC: 12 MCPUh grand challenge
- 2017-18
  - PRACE: 1.1 MNodeh Marconi (KNL)

Ecliptic runs



Polar runs


beta, t=1932.5 s
40
20
0
-20
-40
Y GSE (RE)
-80
-60
-40
-20
0
20
40
X GSE (RE)
VLASIATOR

Hybrid Vlasov

- Protons: 6D distribution function $f(\boldsymbol{r}, \boldsymbol{v}, t)$ that advects according to the Vlasov equation:

$$\frac{\partial}{\partial t} f + \boldsymbol{v} \cdot \frac{\partial}{\partial \boldsymbol{r}} f + \boldsymbol{a} \cdot \frac{\partial}{\partial \boldsymbol{v}} f = 0$$

$$\mathbf{a} = \frac{q}{m}(\mathbf{v} \times \mathbf{B} + \mathbf{E})$$

- From *f* one can compute density, temperature, ...

$$\rho_q(\mathbf{r}) = q \int f(\mathbf{r}, \mathbf{v}, t) d^3 v$$

$$\mathbf{j_i}(\mathbf{r}) = q \int \mathbf{v} f(\mathbf{r}, \mathbf{v}, t) d^3 v$$

- Hybrid: electrons are a massless (ideal) MHD fluid, following Maxwell's equations

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

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

$$\nabla \times \mathbf{B} = \mu_0 \mathbf{j}$$

- Closed by an Ohm's law

$$\mathbf{E} = -\mathbf{V_i} \times \mathbf{B} + \frac{1}{\rho_q}\mathbf{j} \times \mathbf{B}$$

Computational challenge

- Size of distribution function
  - Spatial resolution
    - Ion-kinetic physics require sufficient real space resolution to be resolved, 200 - 300 km
    - Typically 2D simulations, order of 2000 x 2000 spatial cells
  - Velocity resolution
    - Need to resolve & limit diffusion, for current solvers 30 km/s
    - +- 2000 - 3000 km/s (at least) in each dimension.
    - Dense scheme up to 8M phase space cells per spatial cell, sparse scheme gives max average 200k phase space cells
  - In total order of 10^{12} phase space cells (10^{15} with dense implementation)
  - Both a challenge in terms of CPUh, as well as memory consumption

Computational challenge

- Time
  - Strong B close to Earth inner boundary => Large acceleration and high wave velocities (whistler, Alfvén) leading to short timesteps

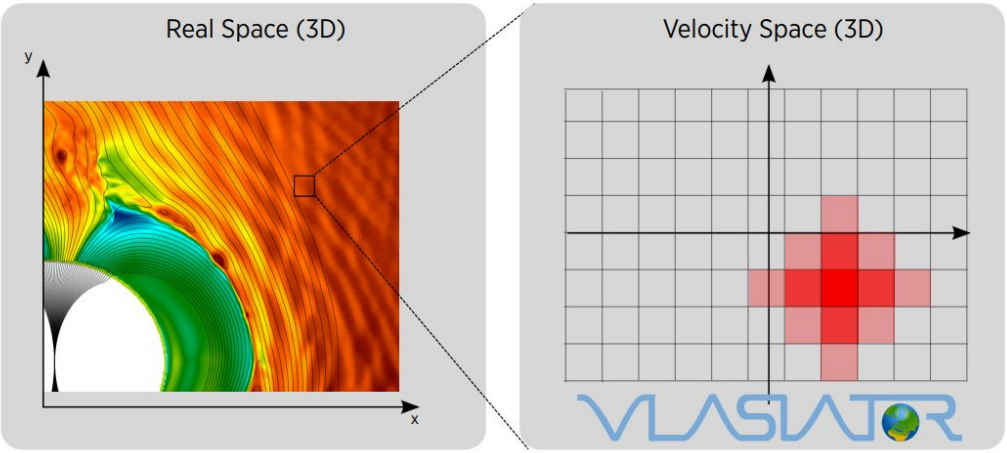
Targets

1. Minimize number of phase space cells
2. Accurate solvers with large dt
3. Efficient implementation – scalability and computational throughput

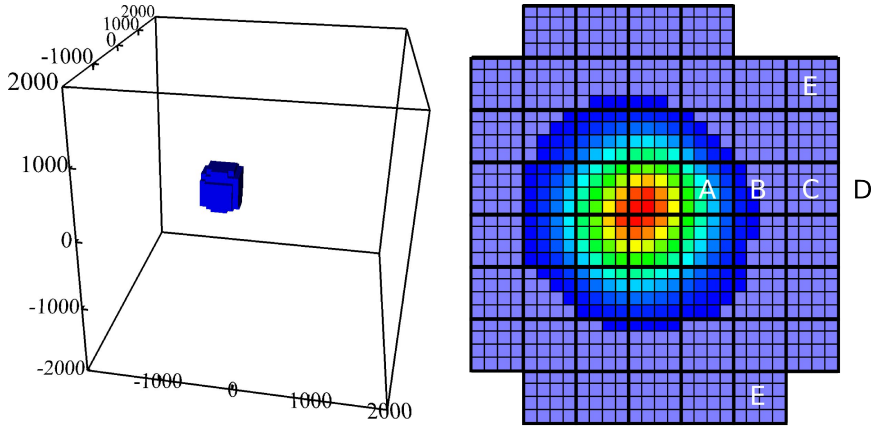
Vlasov propagation

Numerical method: discretization

- Proton distribution function discretized onto 6D cartesian mesh as volume average
- 3D real space mesh (x,y,z)
  - MPI parallel DCCRG :github.com/fmihpc/dccrg)
  - Each cell has a 3D velocity mesh
- 3D velocity space meshes (vx, vy, vz)
  - Comprises **blocks** of 4 x 4 x 4 phase space cells
  - Sparse - only blocks with content exist
  - Single precision in production runs



6D cartesian mesh: 3D real space + 3D velocity space



Sparse velocity space mesh

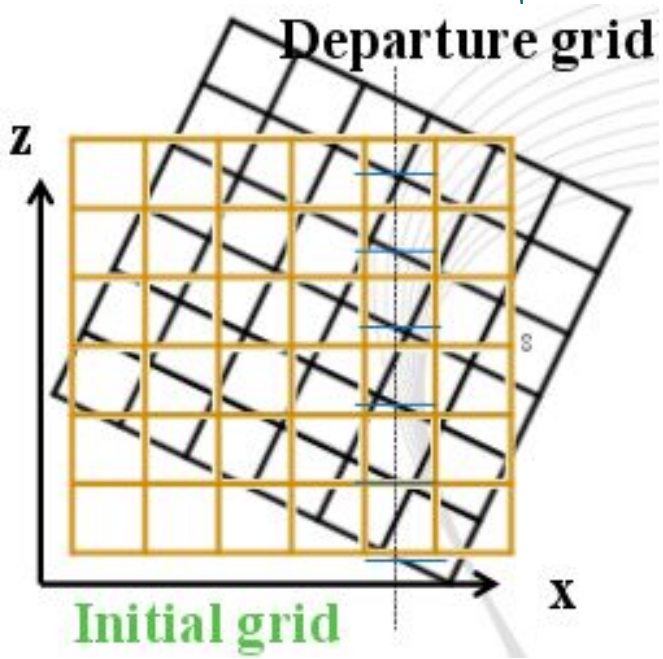
Propagation

- Vlasov solver
  - This is where ~90% of time is spent
  - Real space propagation and velocity space propagation split with Strang splitting in two 3D propagations

$$\widetilde{f}(t+N\Delta t) = \left[S_T(\frac{\Delta t}{2})S_{\mathrm{A}}(\Delta t)S_{\mathrm{T}}(\frac{\Delta t}{2})\right]^N \widetilde{f}(0).$$

  - 2011 – 2014: 2nd order accurate Finite Volume Method approach (Langseth & Leveque 2000).
  - 5th order accurate conservative Semi-Lagrangian scheme (SLICE-3D Zerroukat 2012)

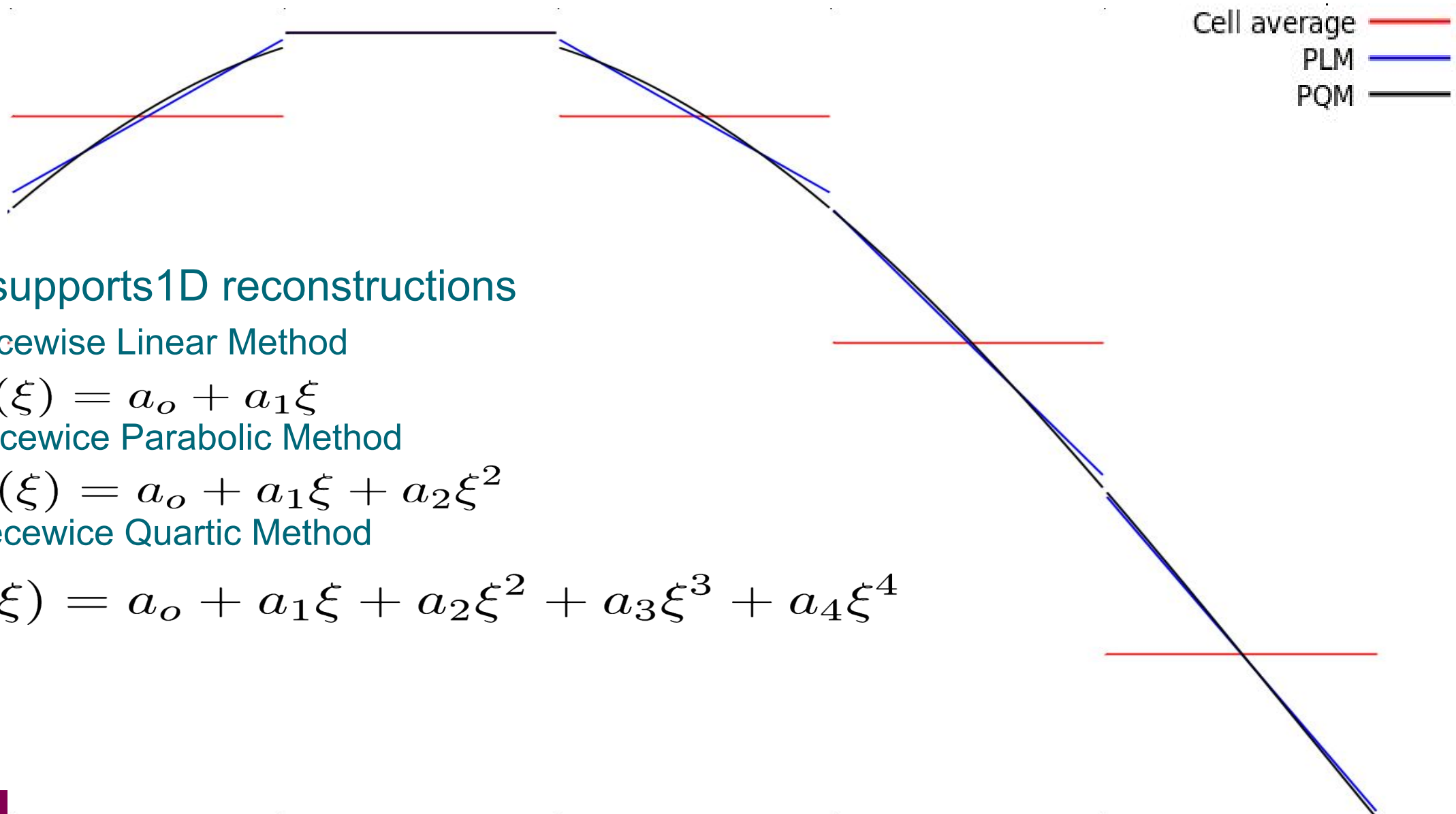
Vlasov propagation



- Propagation based on Slice3D algorithm
- Semilagrangian method, split in 3 1D propagations

1. Compute how the grid moves over one timestep. This is a euclidian transformation (rotation & translation)
2. For z,x,y sweep map along dimension
   a. Compute intersections of new grid with the original
   b. Compute 1D polynomial (4th order) reconstruction for each cell
   c. Map the value to the departure grid by integrating the reconstructed polynomial between the intersection points

1D reconstructions



- Vlasiator supports1D reconstructions
  - PLM: Piecewise Linear Method

$$R_j(\xi) = a_o + a_1\xi$$

  - PPM: Piecewice Parabolic Method

$$R_j(\xi) = a_o + a_1\xi + a_2\xi^2$$

  - PQM: Piecewice Quartic Method

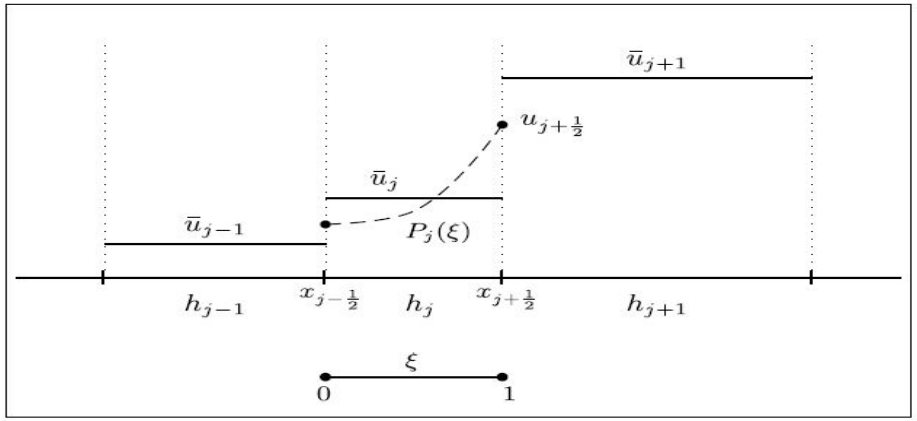
$$R_j(\xi) = a_o + a_1\xi + a_2\xi^2 + a_3\xi^3 + a_4\xi^4$$

1D reconstructions

- PQM (White et al. 2008)
  - Piecewice polynomials of order 4

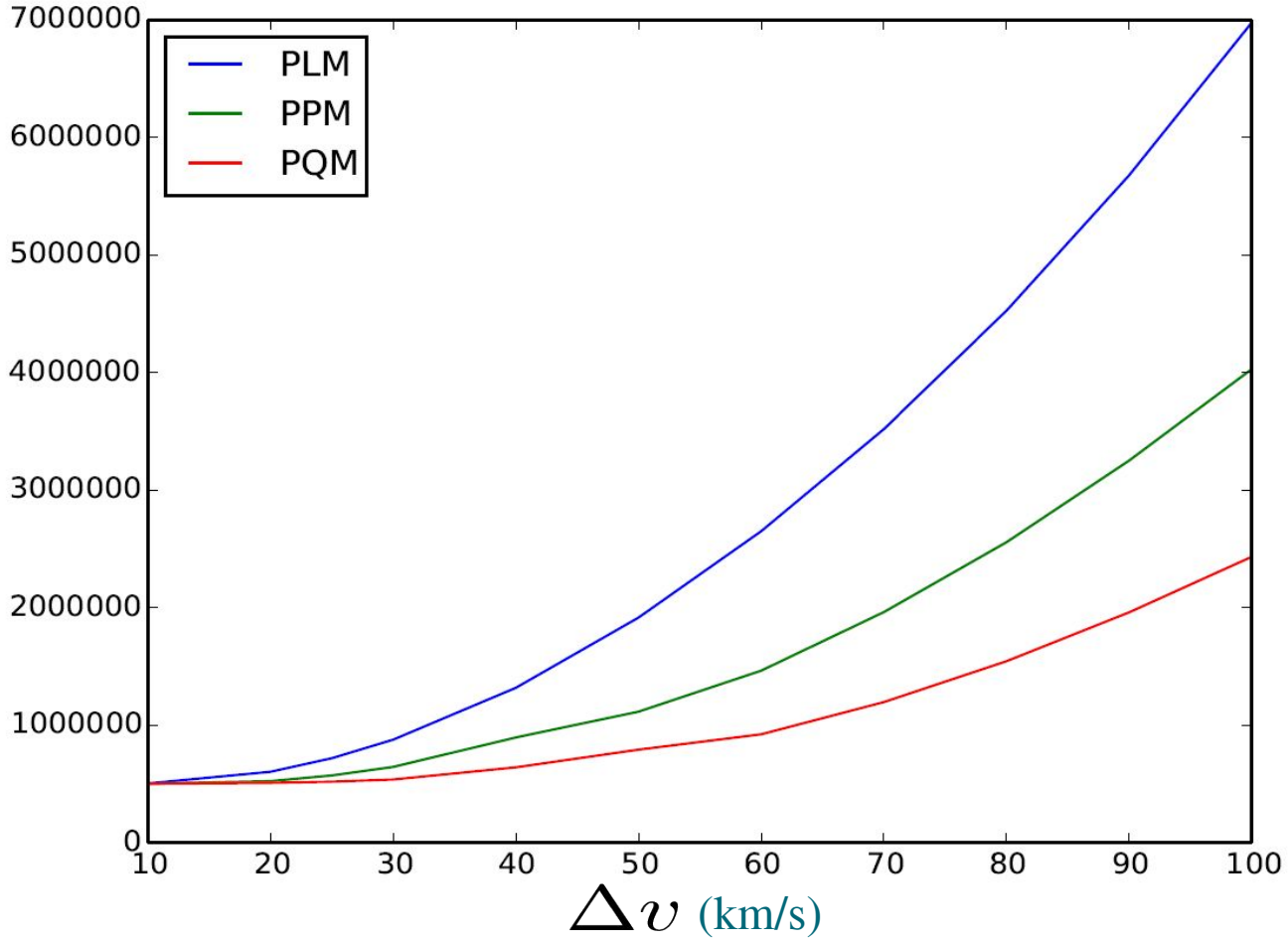
$$R_j(\xi) = a_o + a_1\xi + a_2\xi^2 + a_3\xi^3 + a_4\xi^4$$

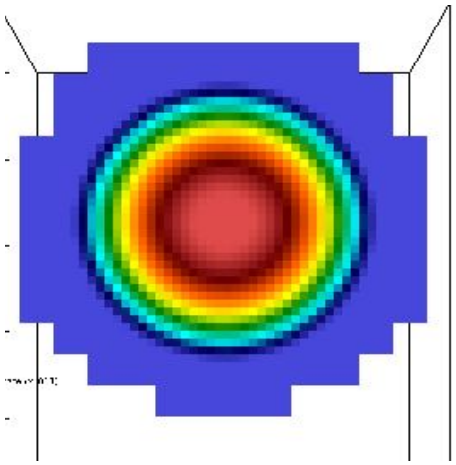
  - Five constraints used to determine *a*
    - 1. Conservative, integral over cell equals mass $\frac{1}{h_j}\int_{x_{j-0.5}}^{x_{j+0.5}} R_j(x)dx = \bar{u}_j$
    - 2 - 5: At cell edges the R_j matches edge values and slopes
  - Edge values and slopes estimated with 8th and 6th order explicit estimates – compact stencil
  - Limited to make sure the reconstruction is bounded and monotonic



Diffusion

Singel Maxwellian with T=500kK
Temperature after 10 gyroperiods, in total 3600 steps





Vlasov propagation

- Not CFL limited in velocity space
  - we typically set max dt to 5% of the gyroperiod
- Order
  - 1D splits makes it easier to add higher order reconstructions
  - PQM is significantly better than PPM, especially in velocity space
  - In ordinary space we still use PPM due to more compact stencils
- Performance
  - It is easy to vectorize efficiently due to the 1D nature
  - Algorithm maps very well to GPUs (work ongoing)
  - Solves 13 Mcells/s per 3D propagator (Haswell, 2.6GHz)

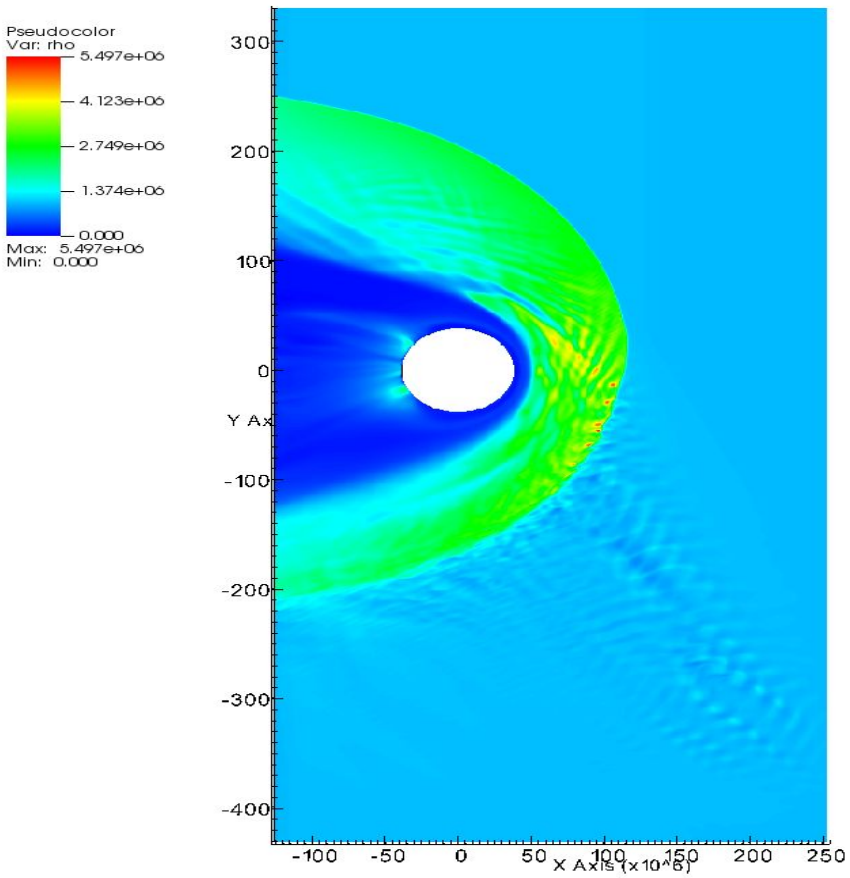
Parallelization on three levels

1. Across nodes on clusters & supercomputers - MPI
   - Domain decomposition of ordinary space (r)
   - Implemented by DCCRG library (*https://github.com/fmihpc/dccrg*)
   - Frequent load balancing due to sparse velocity mesh (RCB, Zoltan)
2. Across cores on nodes – OpenMP
   - Propagation in velocity space threaded over spatial space
   - Propagation in spatial space threaded over velocity space
3. Across core – Vectorization
   - Planes in 4 x 4 x 4 velocity blocks perpendicular to map dimension vectorized
   - Vlasov solvers written using templated vector data type: *http://www.agner.org/optimize/#vectorclass*
   - SSE2, AVX, AVX-512 supported (4, 8 or 16 single precision floats)
   - Main data arrays aligned

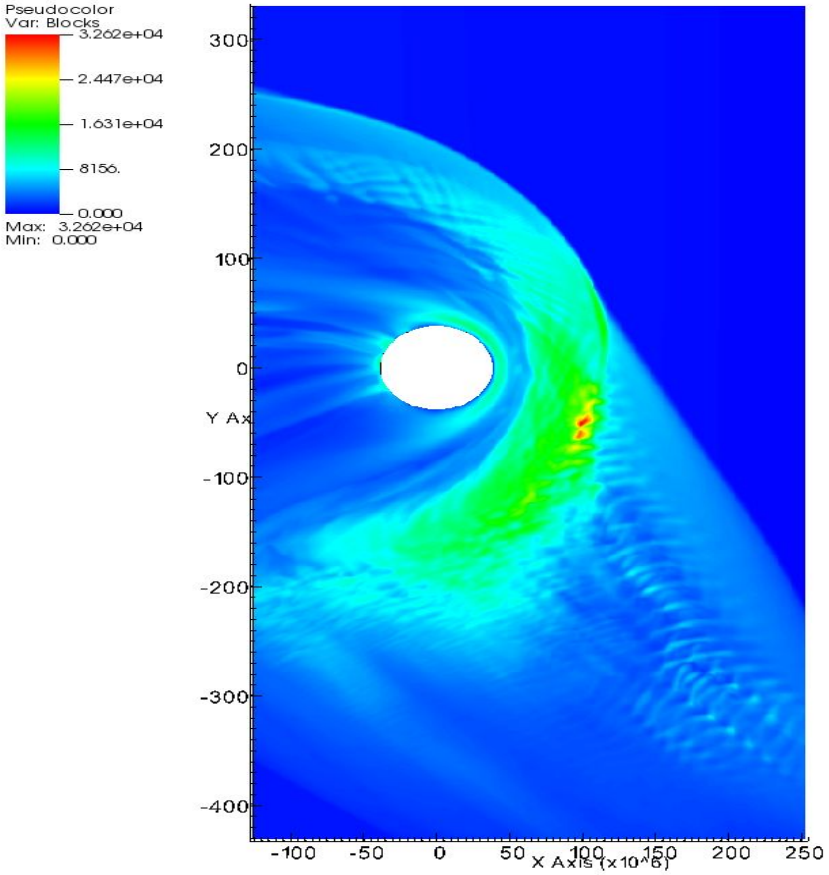
Vectorization example

```
void slope_limiter(const Vec& l,const Vec& m, const Vec& r, Vec& slope_abs, Vec& slope_sign) {
  const Vec two(2.0);
  const Vec half(0.5);
  const Vec zero(0.0);
  Vec sign;
  Vec a=r-m;
  Vec b=m-l;
  Vec minval=min(two*abs(a),two*abs(b));
  minval=min(minval,half*abs(a+b));

  //check for extrema, set absolute value
  slope_abs = select(a*b < 0, zero, minval);
  slope_sign = select(a + b < 0, minus_one, one);

}
```

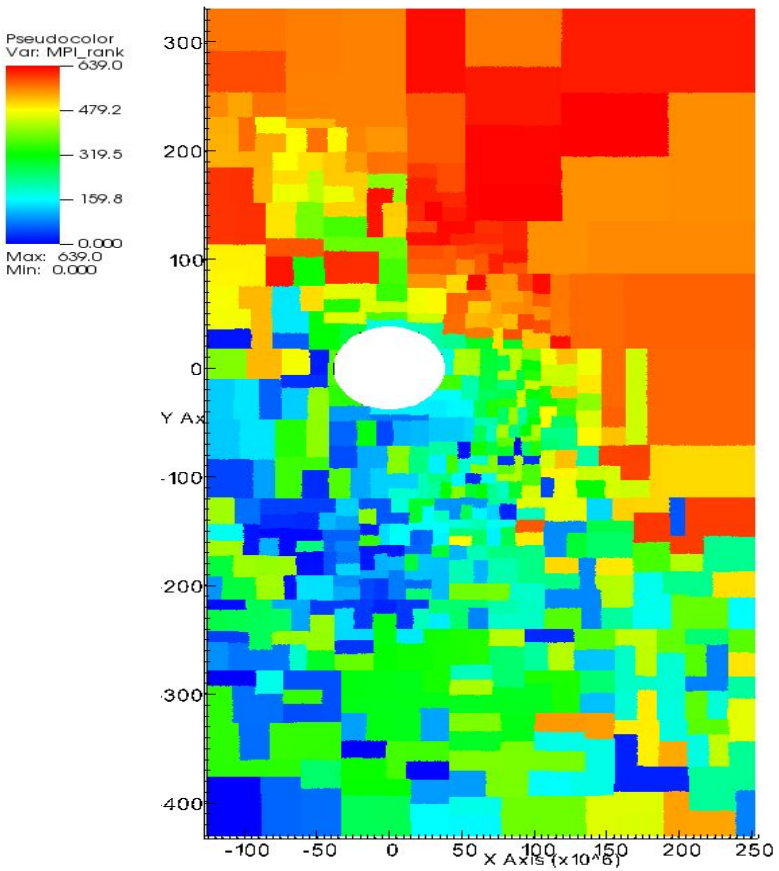
MPI parallelization



Density (1/m^3)



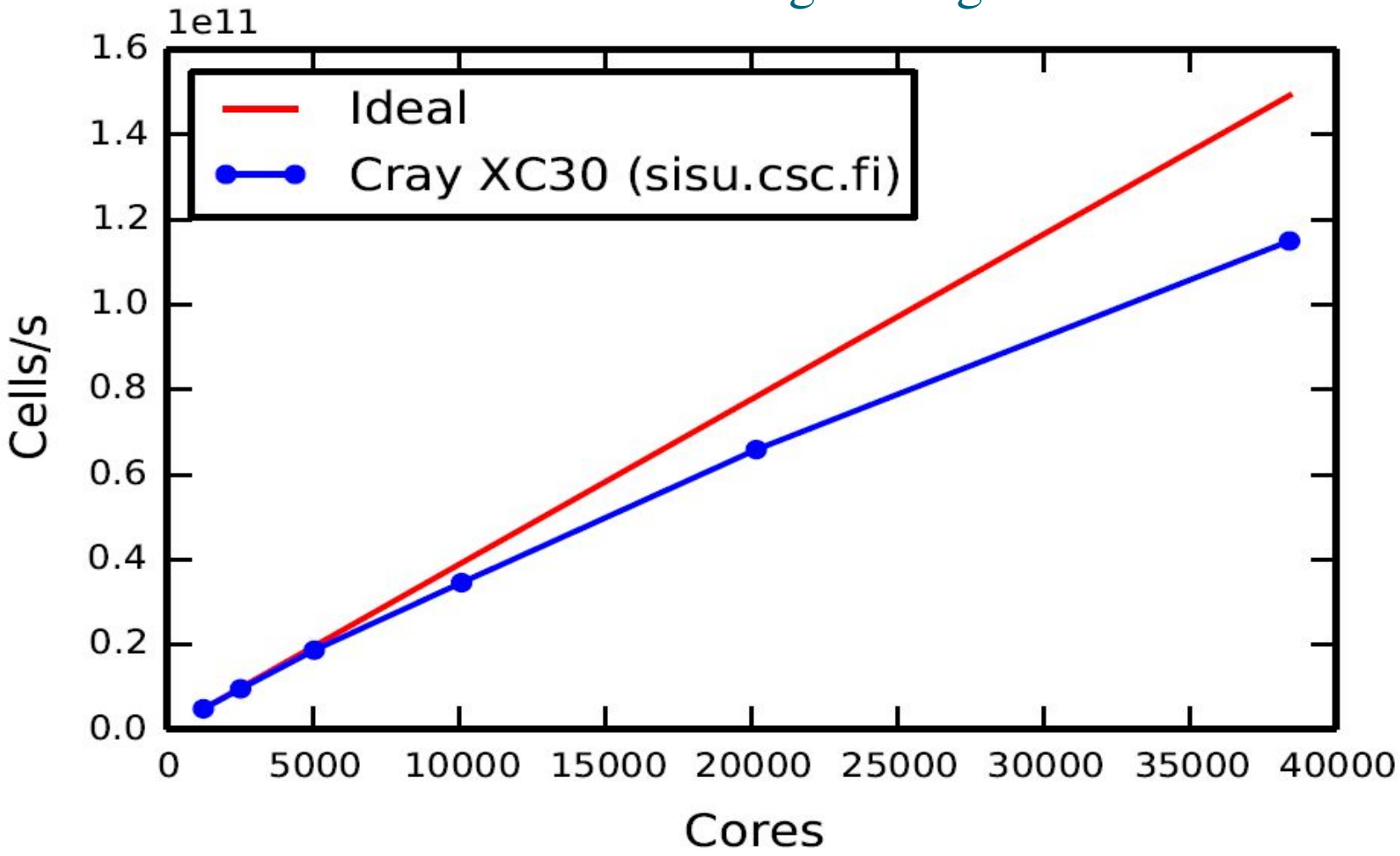
Computational load



MPI computational domains

Scalability

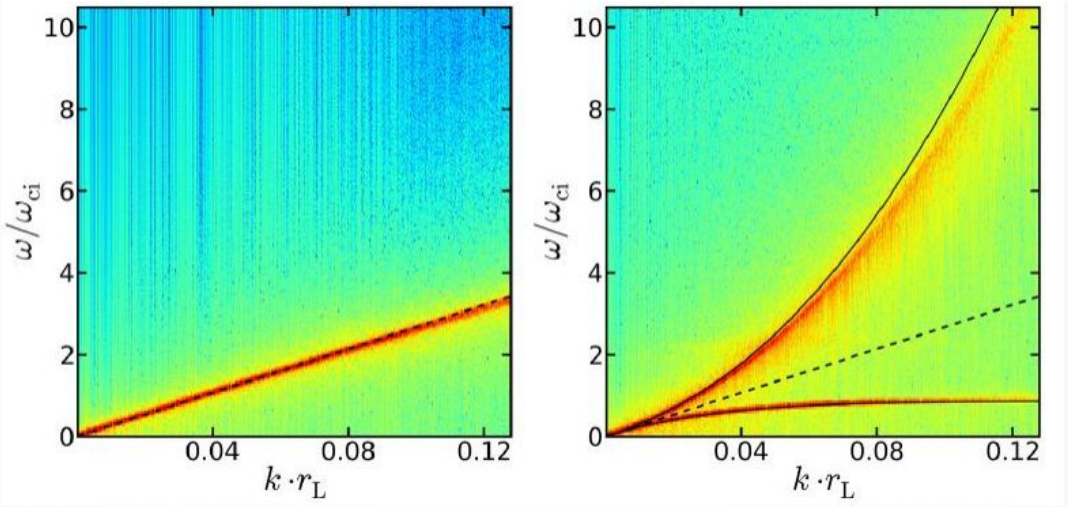
Strong scaling



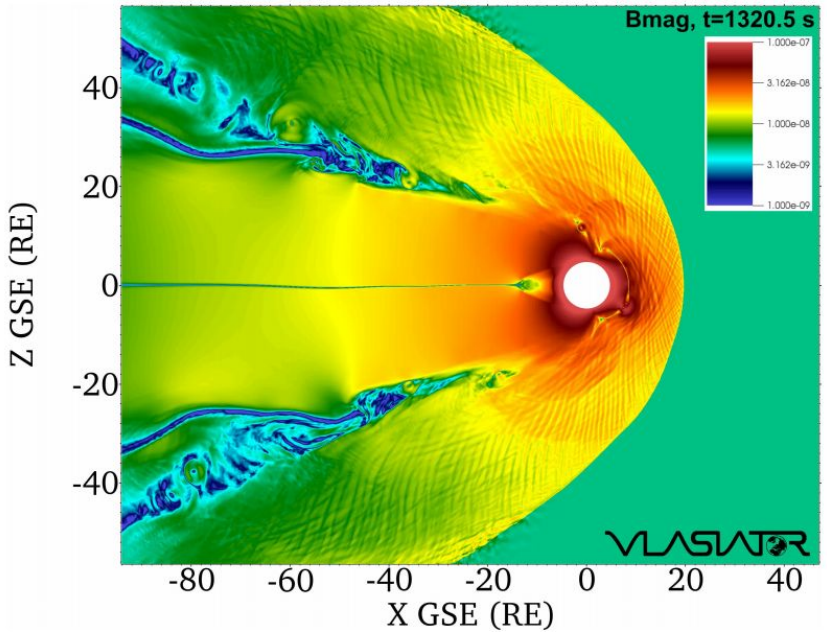
Fieldsolver

Propagation

- Fieldsolver
  - Second-order accurate upwind constrained transport method [Londrillo and del Zanna 2004]
  - Fieldsolver 2nd order accurate in time and space (Runge Kutta)
  - Divergence of B is preserved - no cleaning needed
  - Field Solver CFL depending on B (Alfven & Whistler speed) Large B near the inner boundary, especially near the poles. FS Subcycling is required

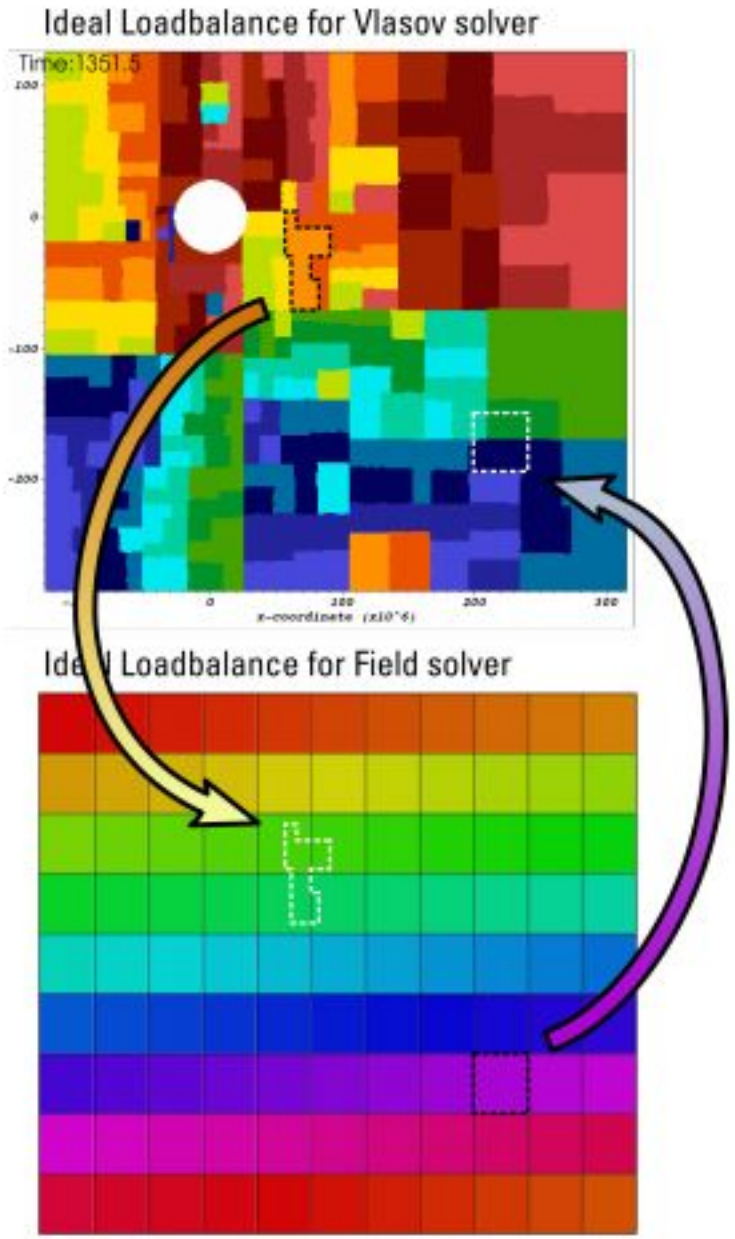


Kempf et al.: Wave dispersion in the hybrid-Vlasov model: Verification of Vlasiator, *PoP (2013)*



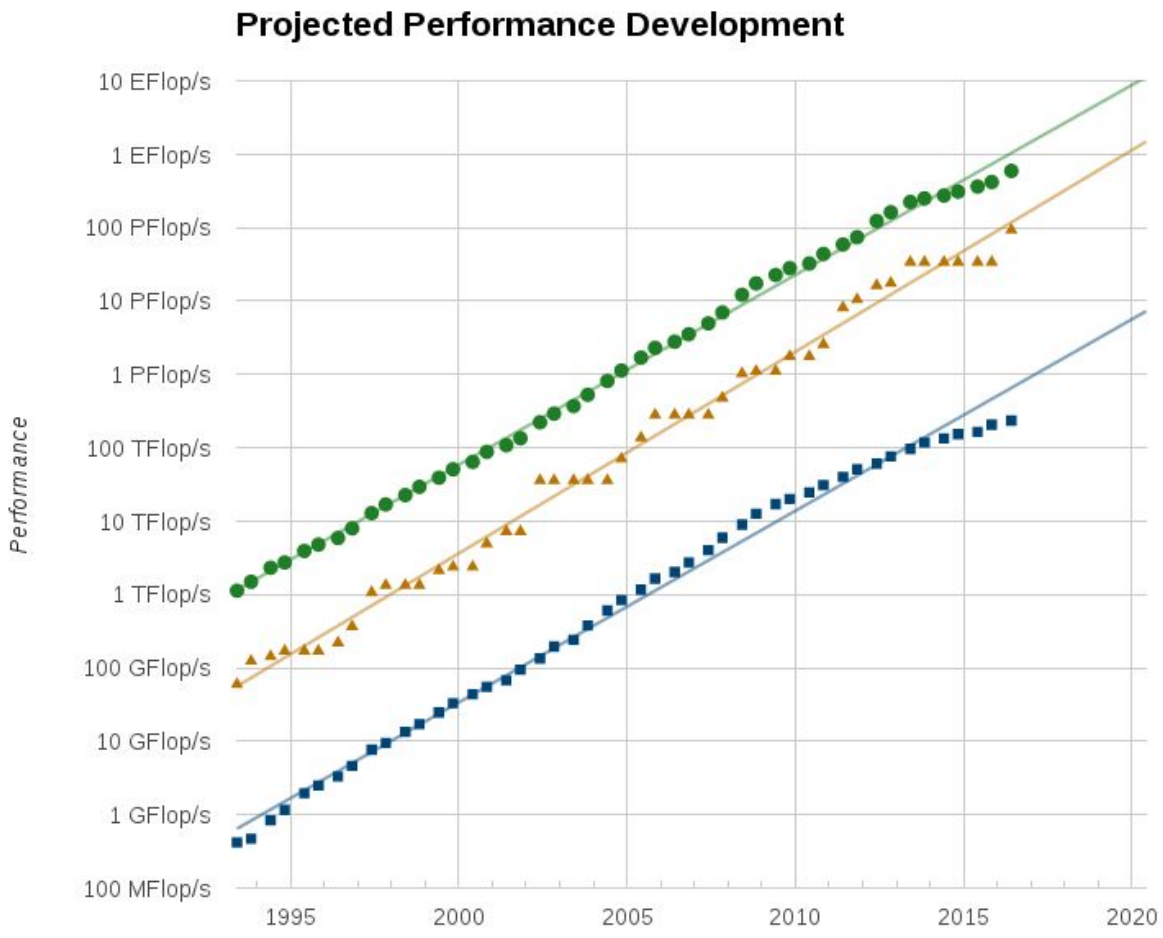
Fieldsolver & load imbalance

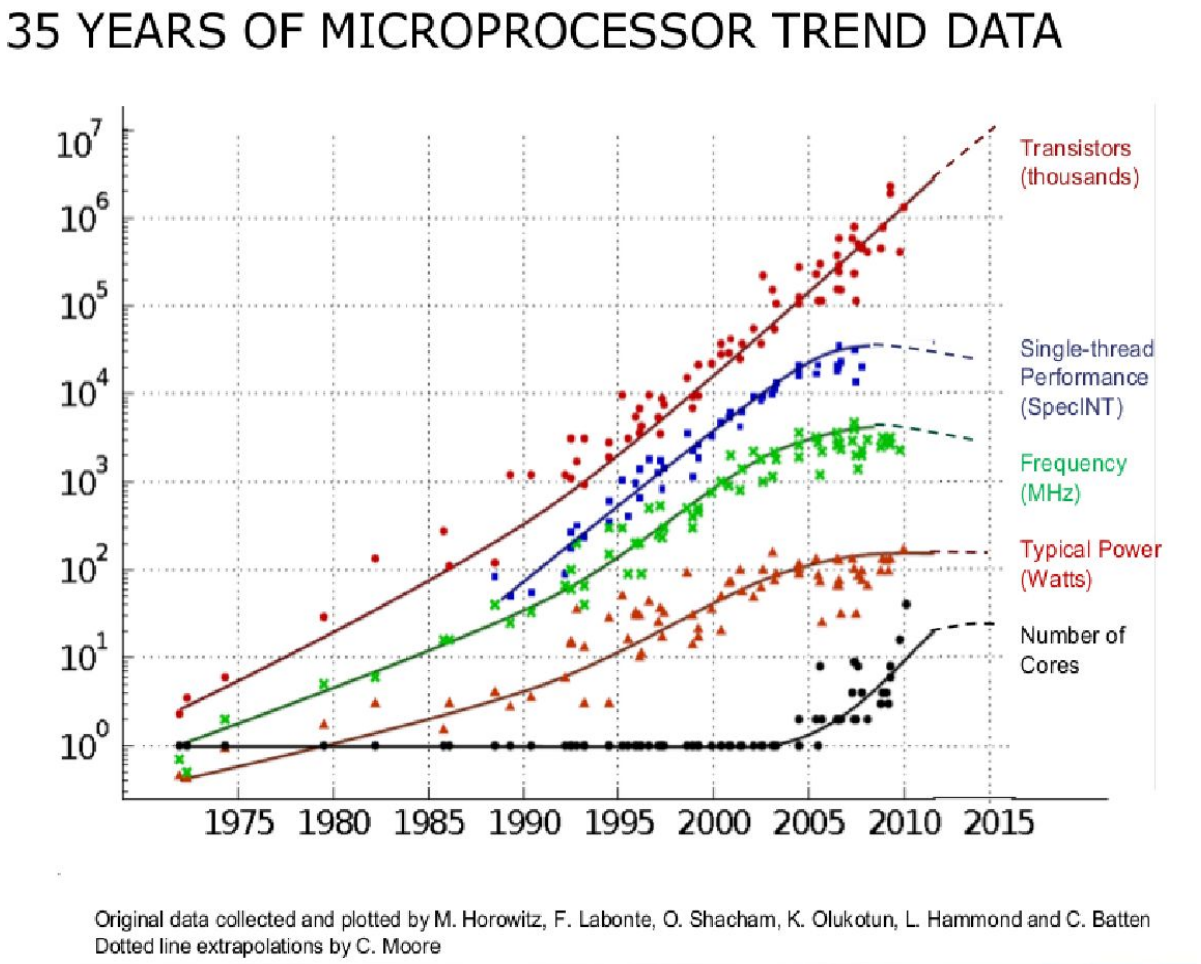
- Fieldsolver in principle cheap to compute, but can at scale take up to 20%
  - Requires several communication passes per cycle
  - Subcycling - tens of cycles per vlasov propagation
  - Bad load imbalance - load balance optimized for Vlasov fluid
- Heterologous domain decomposition
  - Field solver on simple, MPI-Cartcomm uniform spatial decomposition.
  - Moments from Vlasov solver get transferred into field solver (1 - to N)
  - Field quantities get transferred back after FS steps (1 - to - N)



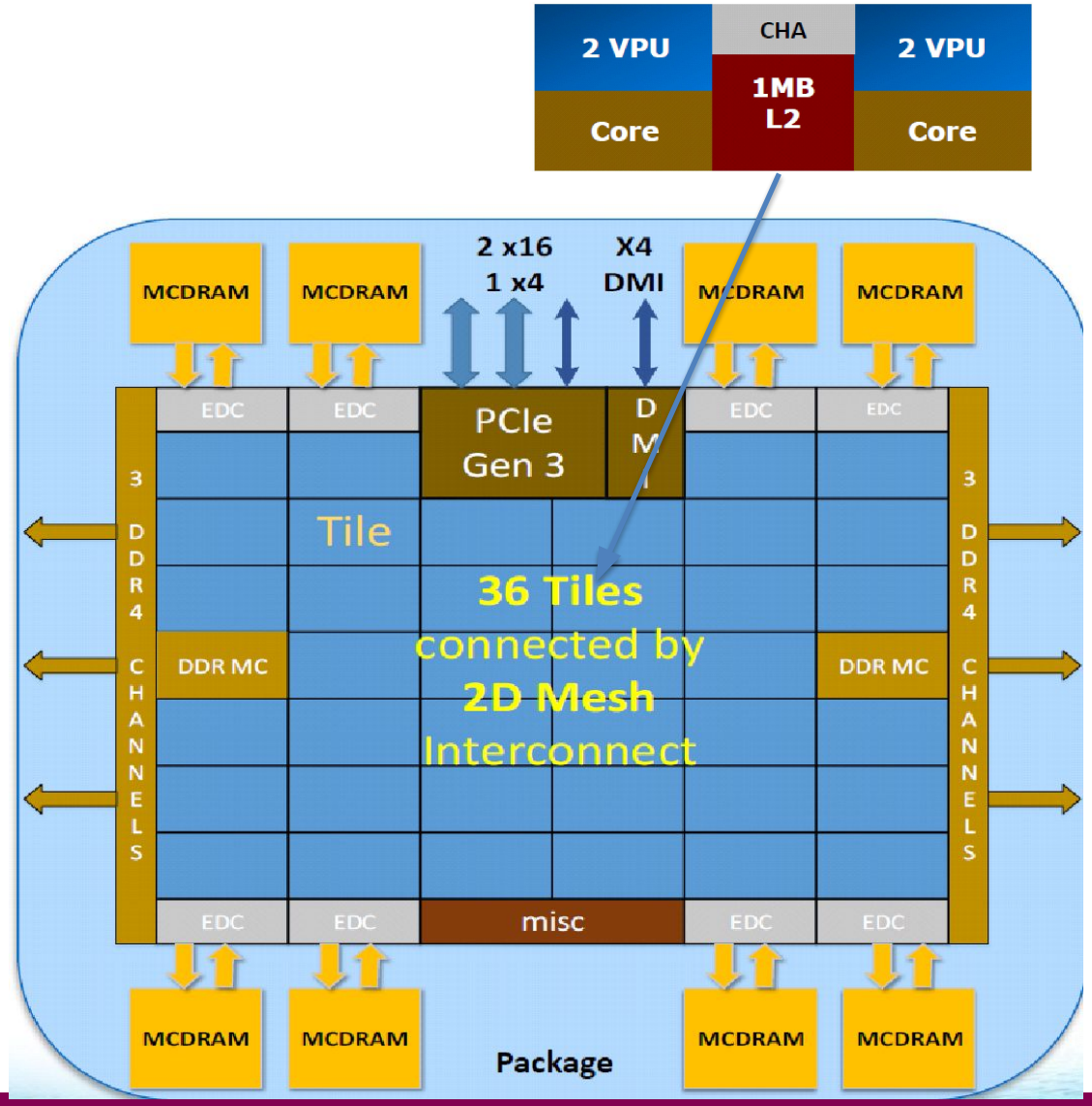
KNL optimizations

Performance development





Intel Xeon Phi - Knights Landing



- Compute
  - Up to 72 cores in 36 Tiles on 2D Mesh
  - **AVX512**
  - 3+TF DP, 6+TF SP Flops
- Memory
  - MCDRAM: 16 GB on-package; 400+GB/s
  - DDR
- Integrated Omnipath

Step 1 -porting

- KNL test system
  - Colfax Ninja development platform with Xeon Phi 7210 (1.3GHz, 64 cores)
  - Marconi - 3600 node cluster with 1.4GHz, 68 core KNLs, Omnipath
  - Comparisons against one node on Cray XC40 with dual Haswell 12c at 2.6 GHz
- Code uses Agner's vectorclass
  - AVX512 support - 3x performance compared to non-vectorized version
- Allocaters matter
  - TBBMalloc gives 30% boost
- 4 threads per core optimal - 2x better than 1
- Cache mode & Quadrant targeted
  - Mode used in Marconi
  - Working set much larger than MCDRAM

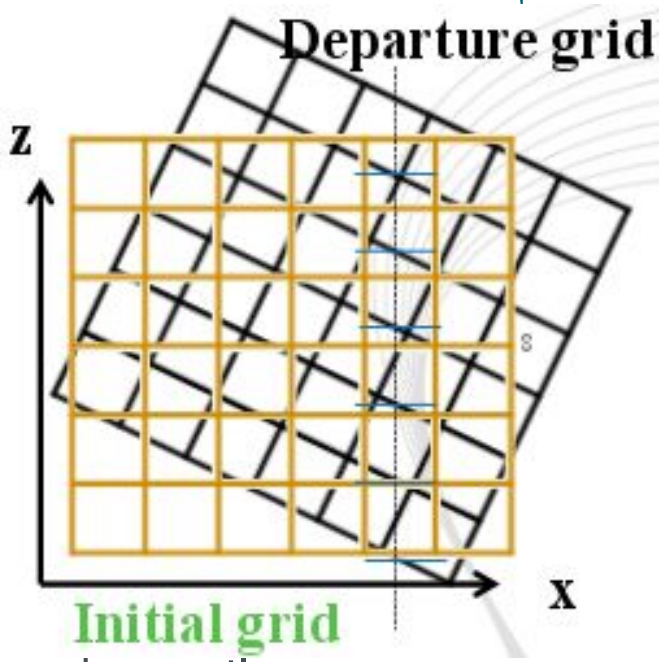
Internal profiling output - KNL

Timers with more than 1% of total time. Set of identical timers has 16 processes with up to 16 threads each.

				Count	Process time			Thread imbalances			Workunits
Id	Lvl	Grp	Name	Avg	Avg (s)	Time %	Imb %	No	Avg %	Max %	Avg
1	1		main	1	412.8	100	0.0006138	1			
2	2		Initialization	1	194.9	47.22	0.005305	1			
131	2		Simulation	1	217.1	52.59	0.01734	1			
138	3		Propagate	10	211.5	97.4	0.04377	1			9.164e+06 Cells/s
147	4		Spatial-space	10	73.23	34.63	0.8482	1			2.646e+07 Cells/s
202	4		Velocity-space	10	134.6	63.66	0.07267	1			1.44e+07 Cells/s
203	5		semilag-acc	10	131.5	97.65	0.06411	1			
205	6		cell-semilag-acc	351	98.65	75.05	3.006	16	1.576	1.927	
209	6	A	re-adjust blocks	10	29.41	22.37	16.59	1			
216	3		compute-timestep	9	4.901	2.258	0.02068	1			

- Performance on KNL not competitive out-of-the-box compared to HSW
  - Total Propagation: **x 0.67**
  - Kernel 1: Velocity space propagation: **x 0.67**
  - Kernel 2: Real space propagation: **x 0.77**
  - Initial focus on velocity space propagator - Mapping 47% of total propagation

Kernel 1: Velocity space



- Original
  - Threaded over velocity meshes (i.e. real space cells)
  - In-place propagation - blocks are sorted along pencils and each pencil is loaded into a temporary array
  - Writes out target values directly into original mesh
- Step 1
  - Removed dynamic block creation from loop, is now done in advance for each pencil
  - Removed all conditionals from loop
  - Loop changed to for loop and target cells are contiguous in memory
- Step 2
  - Storage loop now insignificant
  - 10% of time was spent in one sqrt call in interpolation filtering, replaced by AVX512ER call

Kernel 2: Real space

- Original
  - Threaded over blocks in inner loop
  - Translation uses separate target grid - the point where memory consumption is highest
  - Load block data
    - A unordered map maps block id to its actual location in data array
    - Block removal and addition keeps the data array packed, and thus the block positions are randomized over time
  - Hardware prefetching does not function well
- Step 3
  - Prefetch data
- Step 4 (in progress
  - Swap loops over real space cells and blocks and sort list-of-spatial cells into pencils as in acceleration
  - Can do translation in-place, halfs memory consumption

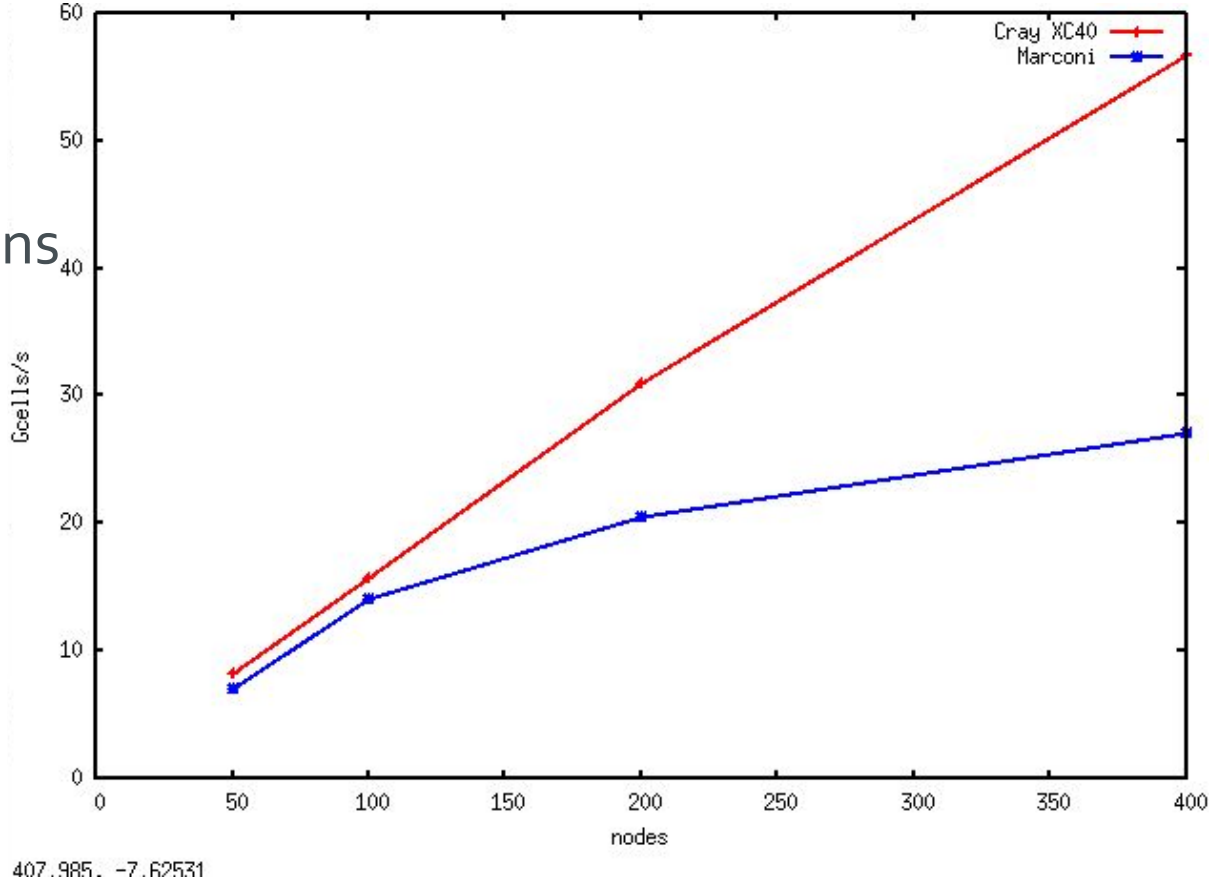
Speedup

	KNL (Intel 17) performance in seconds			Haswell (Gnu 6.2.0) performance in seconds		
	Propagation	**Velocity map.**	Spatial map.	Propagation	**Velocity map.**	Spatial map.
Original	211	**98.7**	73.2	144	**66.0**	56.5
Step 1	149	**44.5**	71.3	121	**42.2**	56.5
Step 2	144	**39.2**	71.3	121	**42.2**	56.5
Step 3	**138**	39.2	**64.7**	**111**	43.8	**47.2**

	Marconi KNL (Intel 17) performance in seconds			Haswell (Gnu 6.2.0) performance in seconds		
	Propagation	**Velocity map.**	Spatial map.	Propagation	**Velocity map.**	Spatial map.
Step 3	**104**	34.1	38.9	**111**	43.8	**47.2**

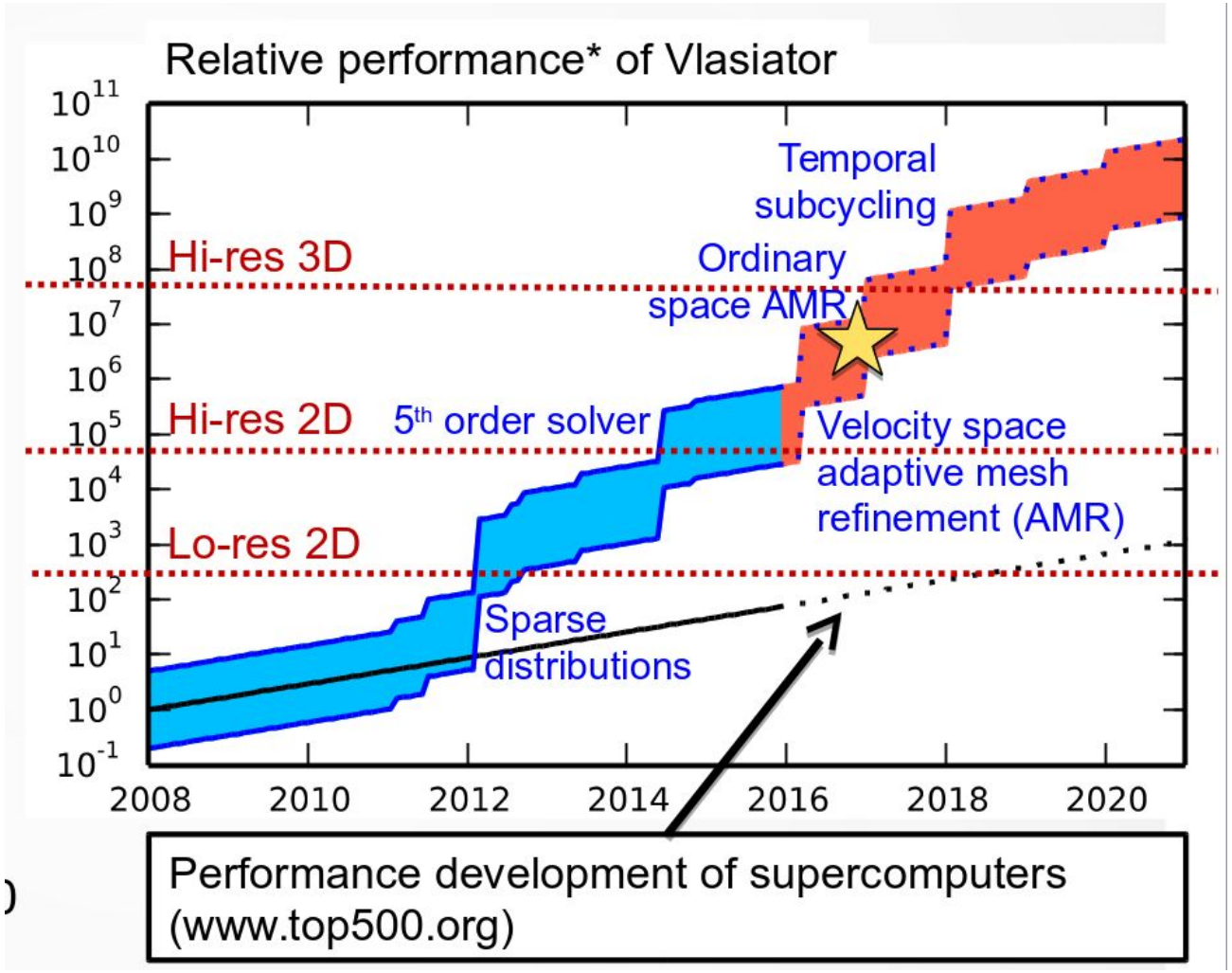
Marconi - issues

- Scalability not impressive
  - Also visible in MPI synthetic benchmarks
- Occasionally large variability between runs
  - Cleans up after reboots
- Poor IO latency
  - Rewrite IO library to buffer calls to MPI IO



Outlook

- 6D still target - requires much quicker improvement than through moores law
- Reduce memory consumption
  - No target in translation
  - Compression for phase space values
  - AMR in all 6 D
- Reduce computational load
  - AMR in all 6D
  - No global timestep



Q?