Solar Storm Modeling using OpenACC: From HPC cluster to "in-house"

Ronald M. Caplan,
Jon A. Linker, Cooper Downs, Tibor Török, Zoran Mikić,
Roberto Lionello, Viacheslav Titov, Pete Riley, and Janvier Wijaya

Predictive Science Inc.
www.predsci.com

Predictive Science Inc.

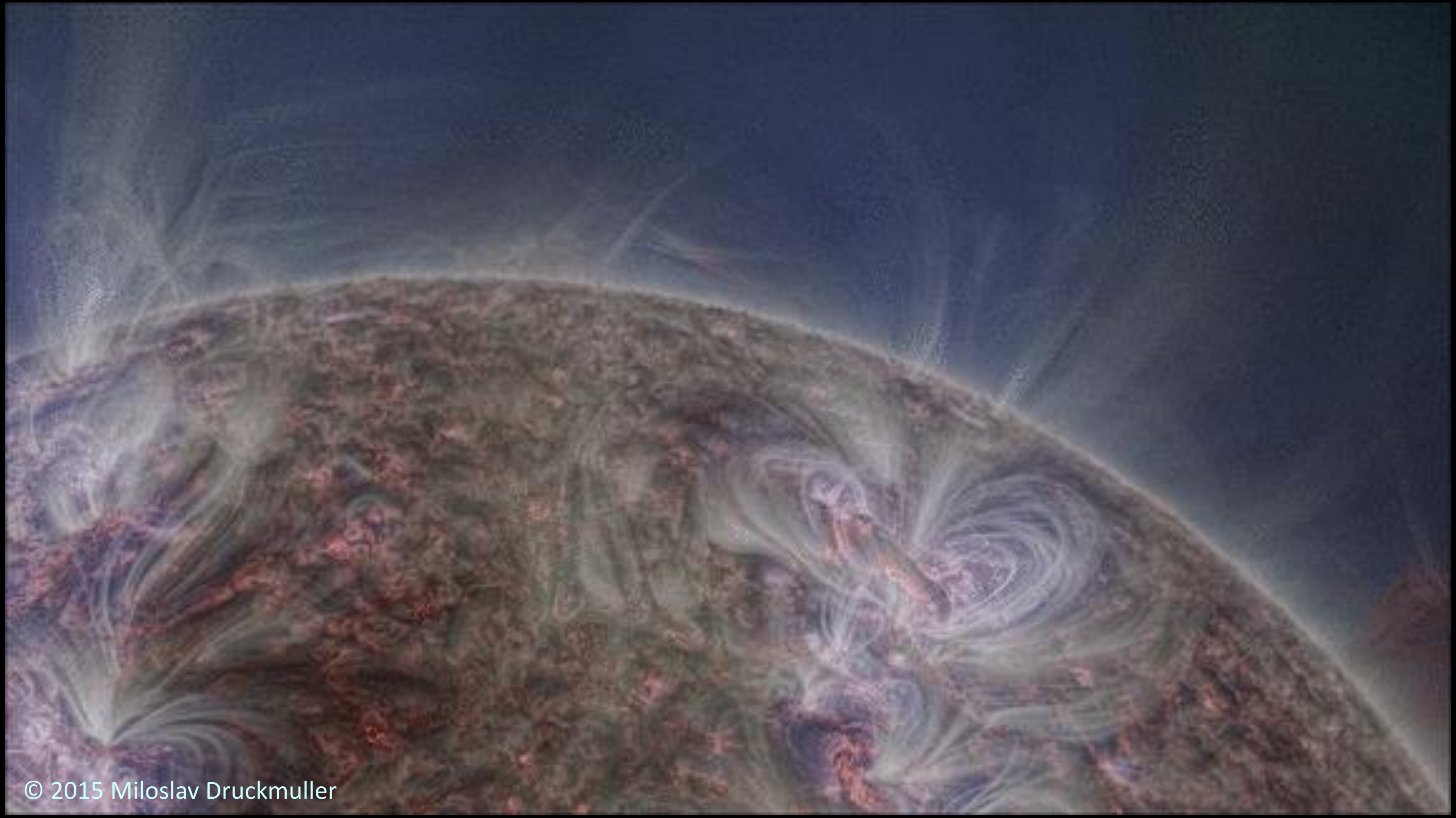
GPU TECHNOLOGY CONFERENCE

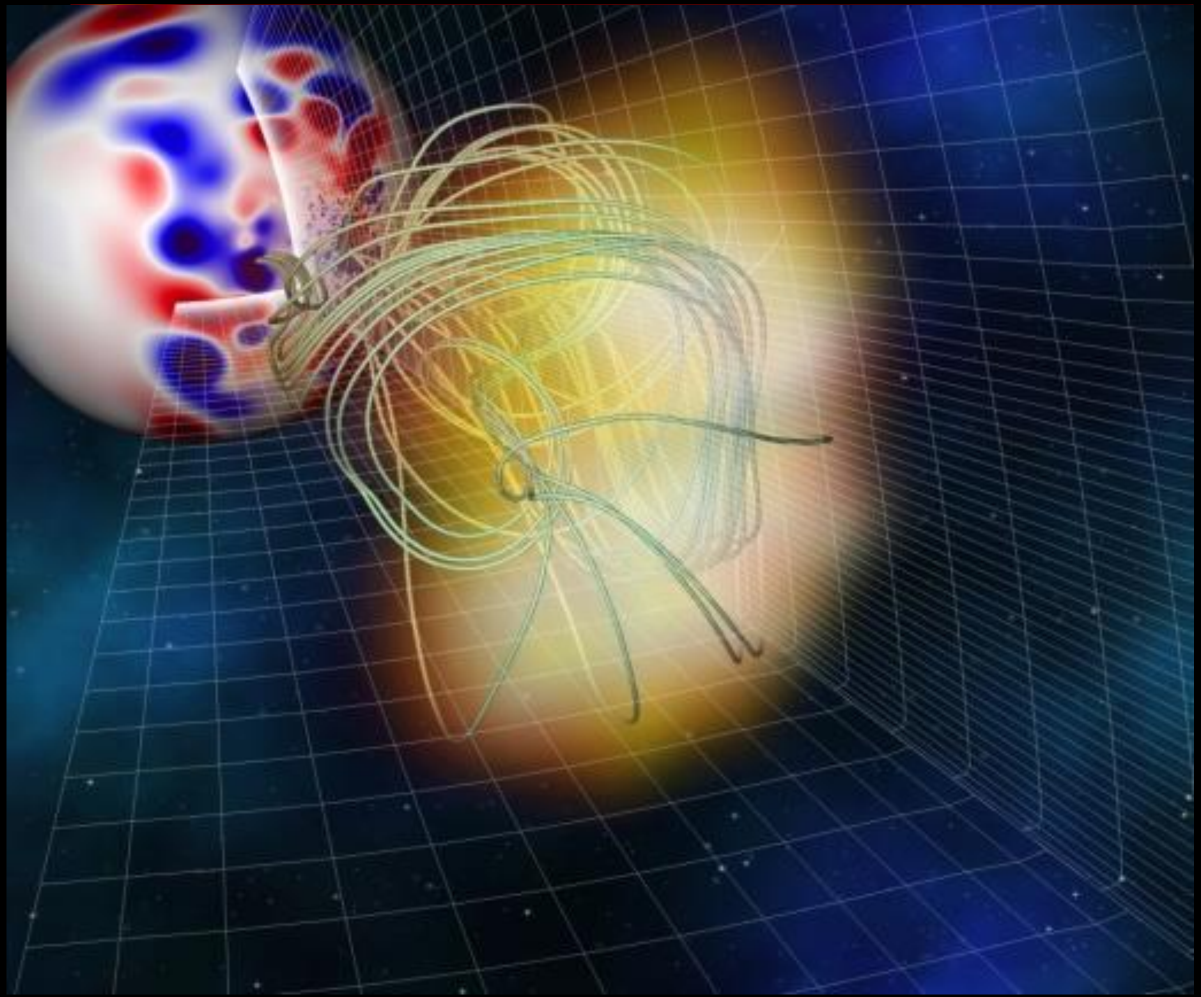
Outline

- Solar Storms
- Modeling a Coronal Mass Ejection
- Why add OpenACC?
- Recap of previous OpenACC implementations
- MAS: **M**agnetohydrodynamic **A**lgorithm outside a **S**phere
- Initial OpenACC Implementation of MAS
- "Time-to-solution" results
- Summary and Outlook

Solar Storms

- Solar storms include coronal mass ejections (CMEs): large explosive events capable of ejecting a billion tons of magnetized million-degree plasma out into space
- CME impacts on Earth can cause interference and damage to electronic infrastructure including GPS satellites and the power grid
- **The first step in forecasting CME impacts is the ability to accurately model their initiation and propagation**

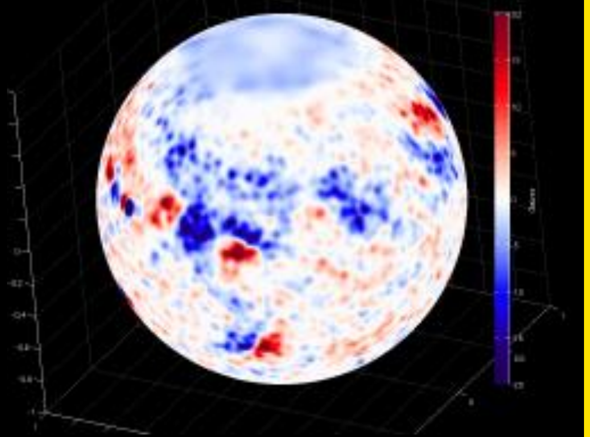




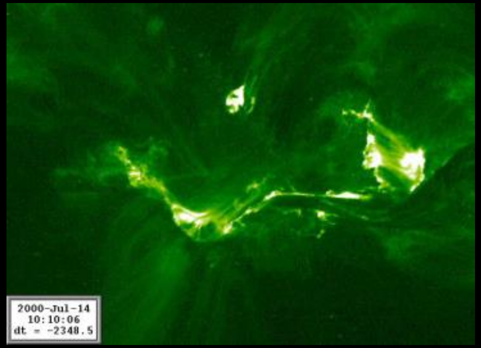
How We Model a Coronal Mass Ejection

Observations

Satellite Observations



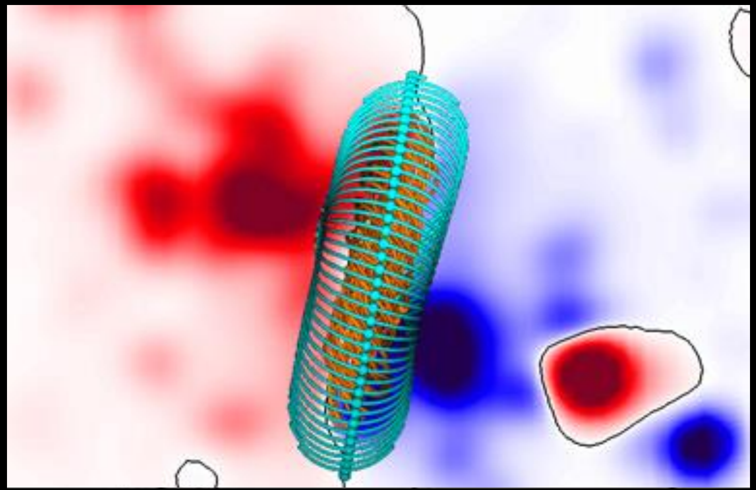
Surface Magnetic Field

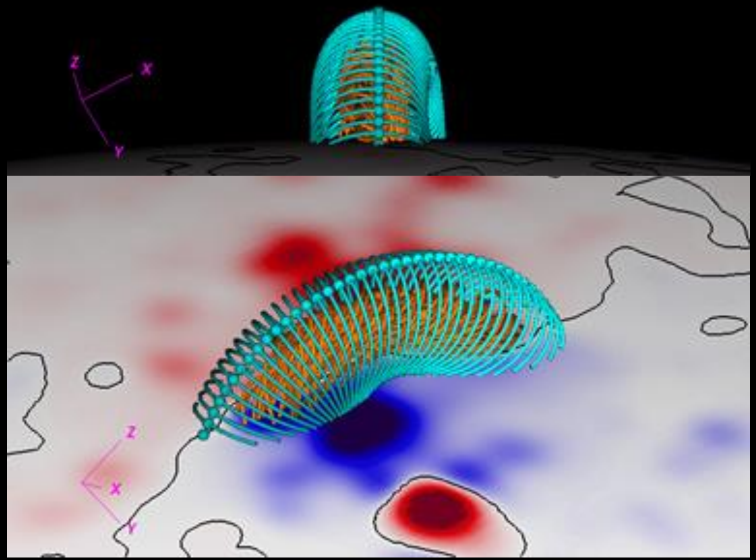


EUV images

CME Initial Condition

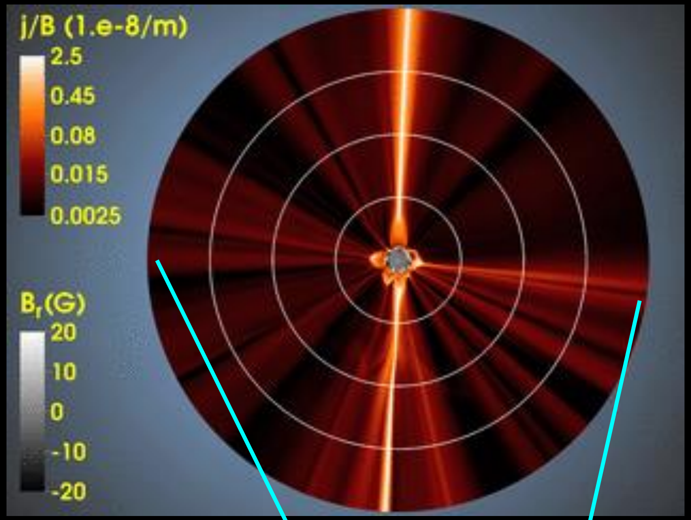
Design and compute stable "Flux Rope" in "Active Region" embedded in global approximate magnetic field



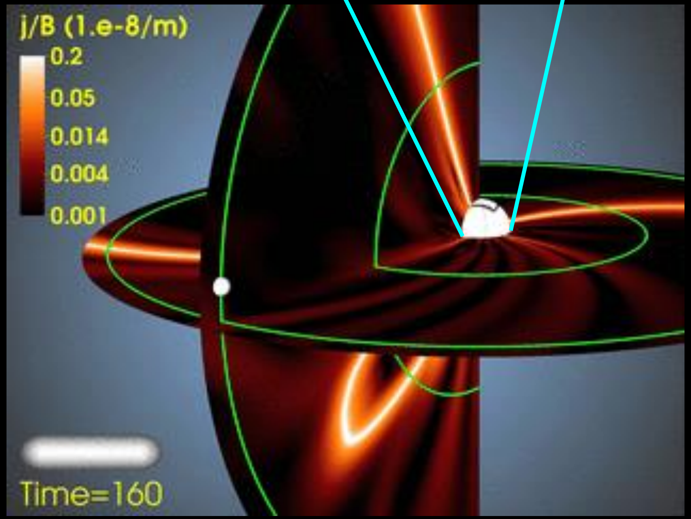


Global TMHD Simulations

Manipulate surface field/flow to erupt CME and propagate to Earth

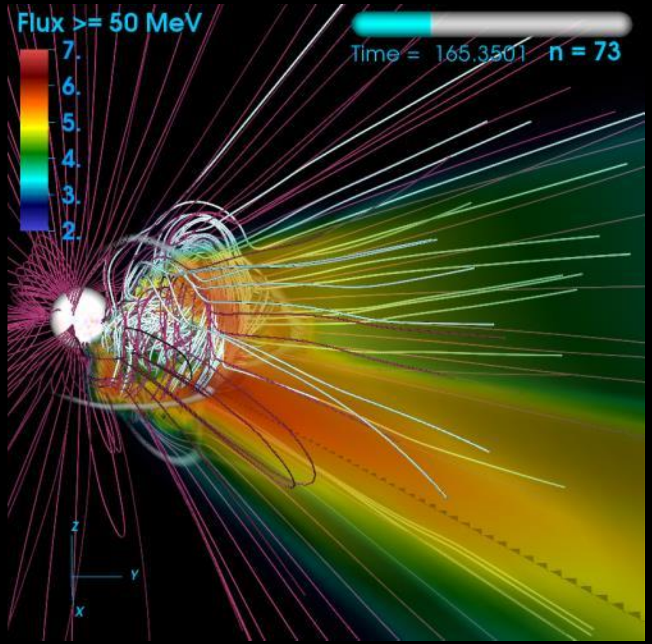


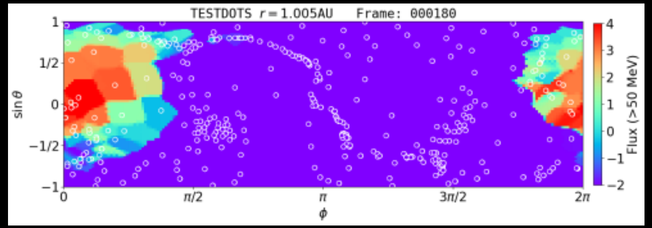
Coronal Simulation



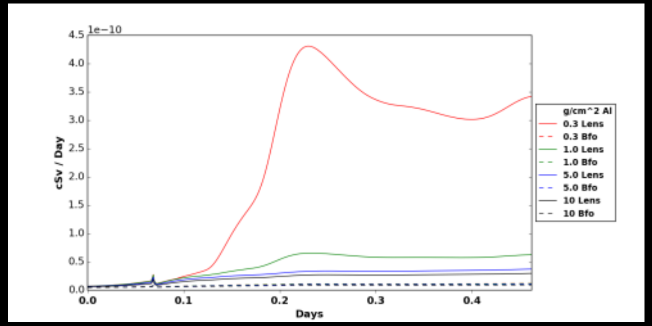
Heliospheric Simulation

Post Analysis





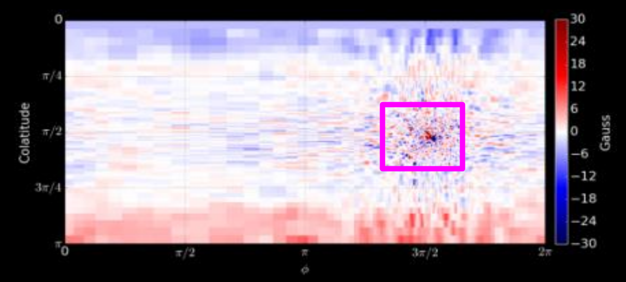
Energetic Particle Fluxes



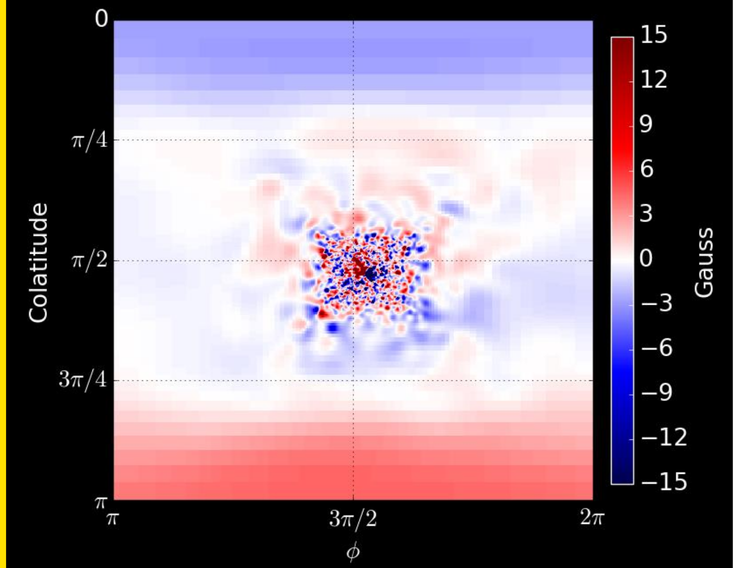
Radiation Dose Levels

Flux Rope Modeling Pipeline (CME Generator)

Isolate CME location, set grid and interpolate



Smooth data to resolve

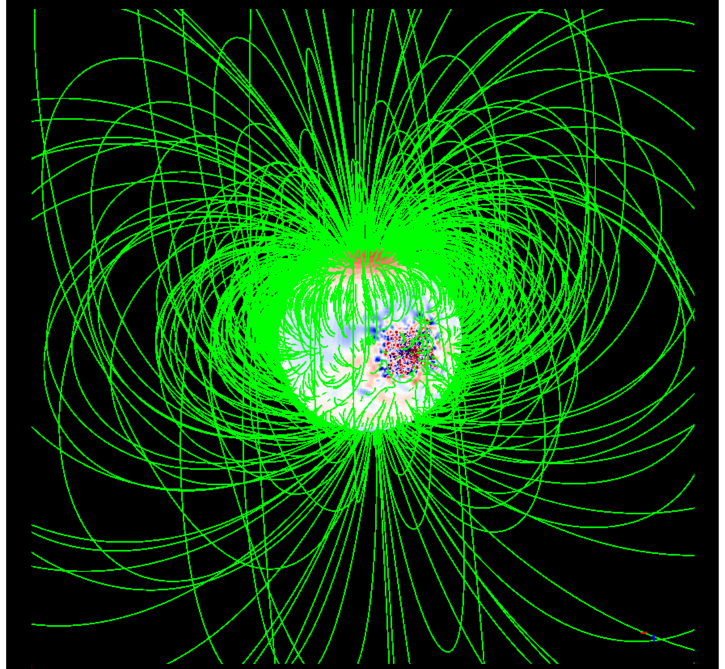


DIFFUSE

GPU TECHNOLOGY CONFERENCE 2016

Compute approximate 3D magnetic field

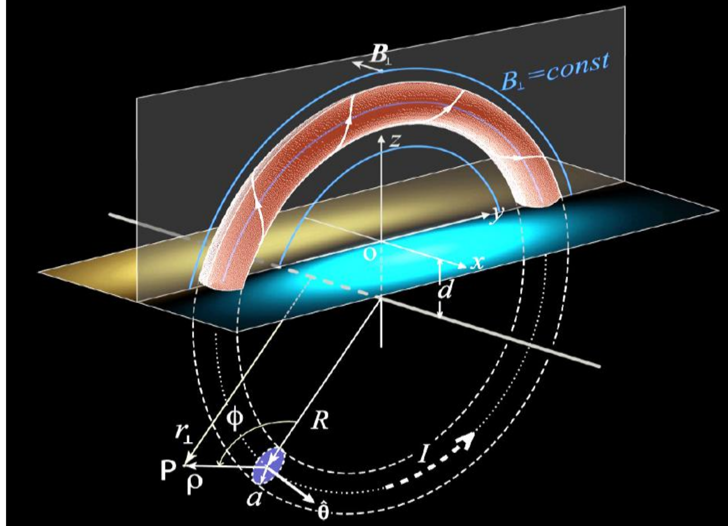
Potential Field



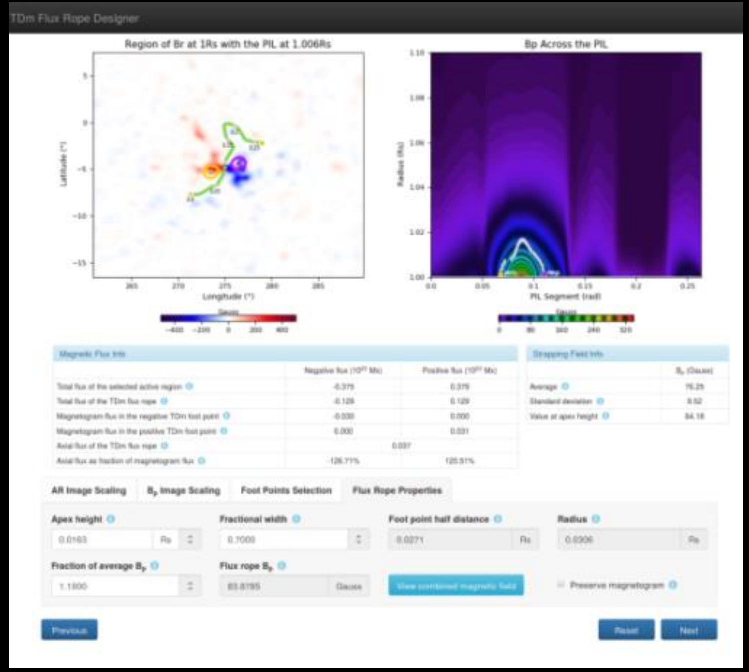
POT3D

GPU TECHNOLOGY CONFERENCE 2017

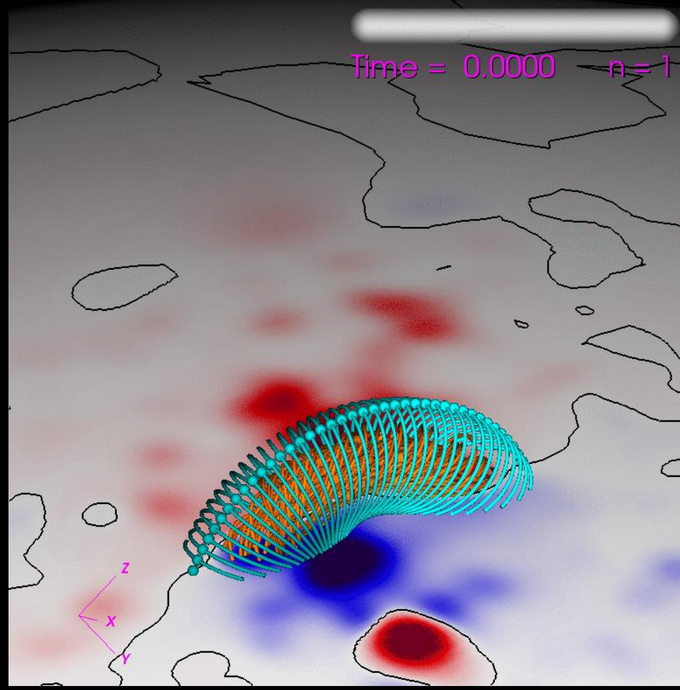
Design and insert analytic flux rope



Titov, V.S., et. al. Ap.J. 790,163 (2014)



Relax to Steady-State with "0-Beta" MHD Simulation



MAS (0-Beta)

GPU TECHNOLOGY CONFERENCE TODAY!

Production Test Run (TEST1)

TEST1: Stable rope (default resolution)

Run information

Physical time duration: **211 sec**

Number of time-steps: **200**

$$160 \times 267 \times 602 \sim 26 \,\text{million points}$$

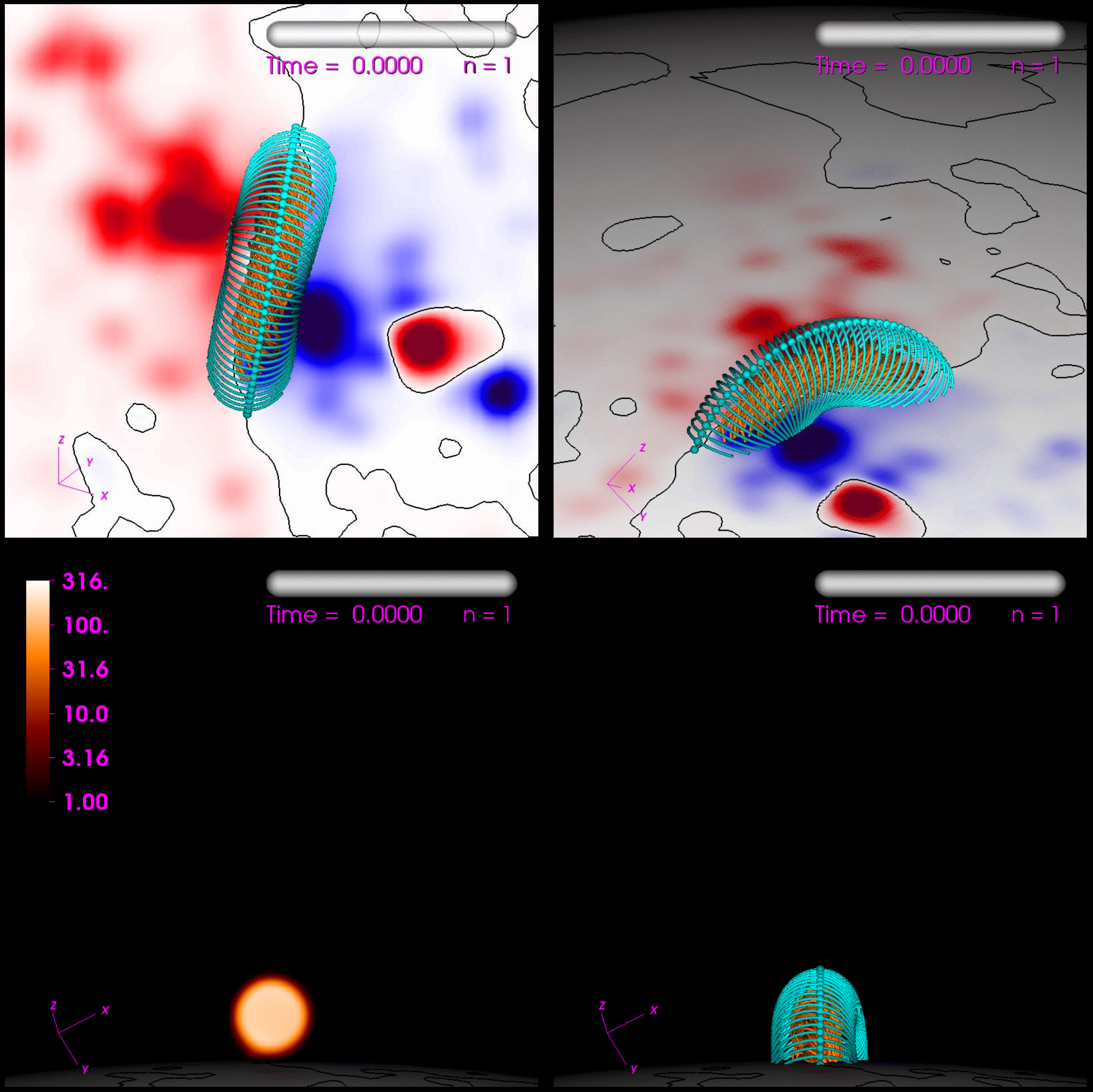
Acceptable time-to-solution: 20 min

Detailed run information

	N	Δ_{min}	Δ_{max}	$\max \frac{\Delta_{i+1}-\Delta_i}{\Delta_{i+1}}$
r	136	64 km	540000 km	9%
θ	450	$0.052°$	$7.45°$	11%
ϕ	543	$0.052°$	$14.32°$	10%
t	887	0.001 sec	0.13 sec	10%

PCG Solver Iterations per Time Step (mean)

	SI Predictor	SI Corrector	Viscosity
PC1	186	195	963
PC2	$55 \to 65$	$57 \to 67$	$174 \to 310$



Production Test Run (TEST2)

TEST2: Eruptive Rope (high resolution)

Run information

Physical time duration: **118 sec**

Number of time-steps: **887**

$$136 \times 450 \times 543 \sim 33\,\text{million points}$$

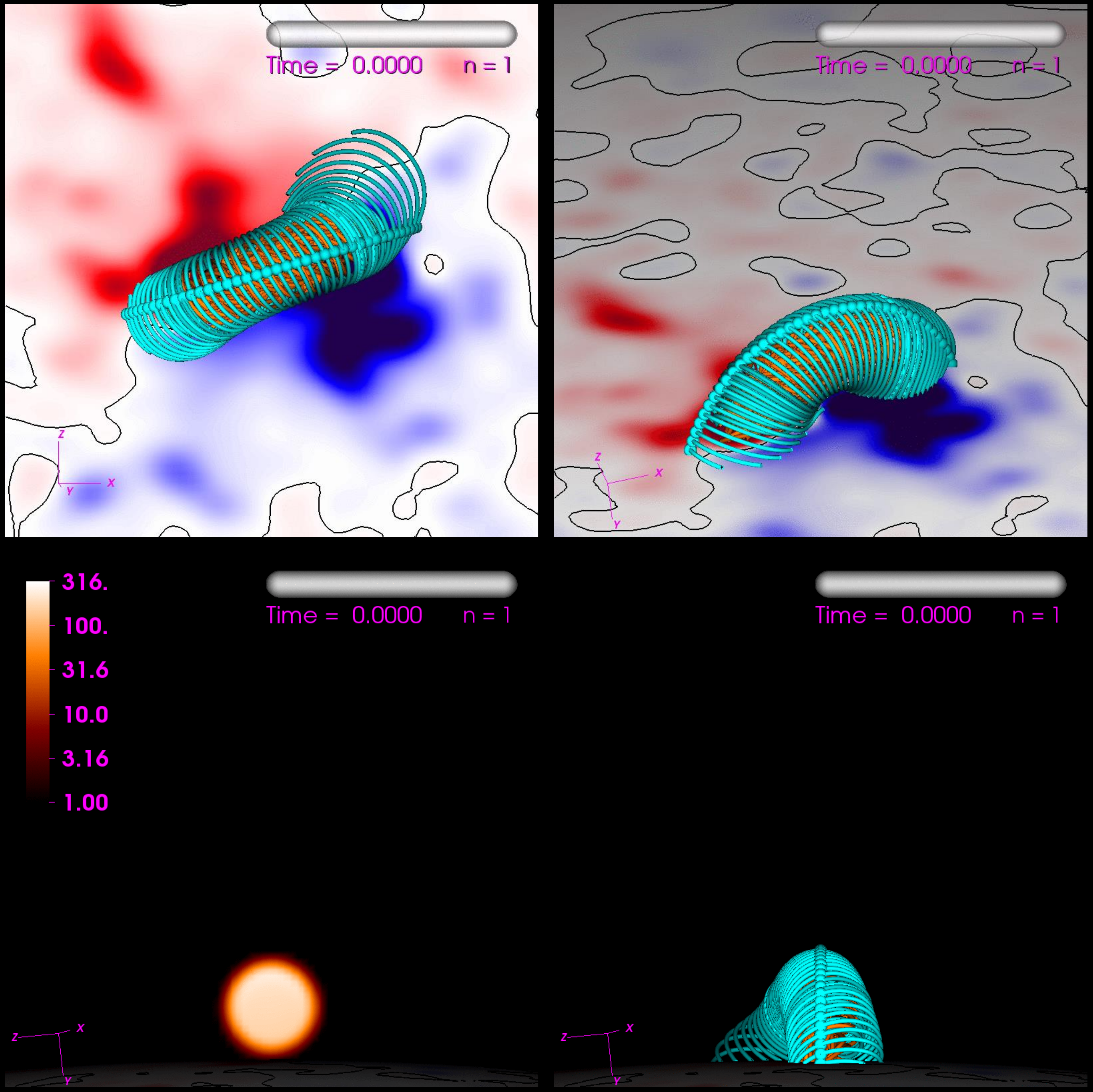
Acceptable time-to-solution: 90 min

Detailed run information

	N	$\Delta_{\min}$	$\Delta_{\max}$	max $\frac{\Delta_{i+1}-\Delta_i}{\Delta_{i+1}}$
r	160	800 km	530000 km	9%
θ	267	0.086°	9.74°	11%
ϕ	602	0.097°	14.90°	10%
t	200	0.001 sec	0.17 sec	4%

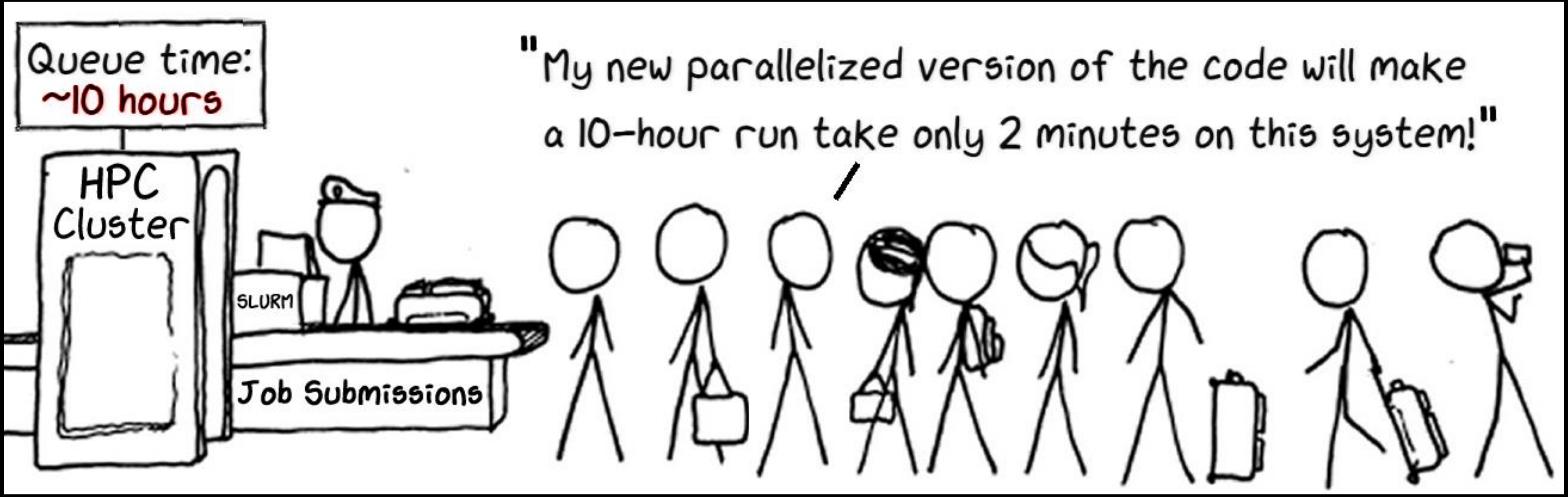
PCG Solver Iterations per Time Step (mean)

	SI Predictor	SI Corrector	Viscosity
PC1	32	34	516
PC2	$13 \rightarrow 15$	$14 \rightarrow 16$	$92 \rightarrow 149$



Motivation for OpenACC Implementation

- **MAS** run currently requires an HPC cluster for acceptable "time-to-solutions"
- Would rather run "in-house" to avoid wait queues, allocation usage, and have control of software stack



THE BIG IDEA: Can we achieve the same acceptable "time-to-solutions" on a single multi-GPU node using OpenACC in a *portable*, *single-source* implementation?

4xGPU Workstation

8xGPU Server

16xGPU Server

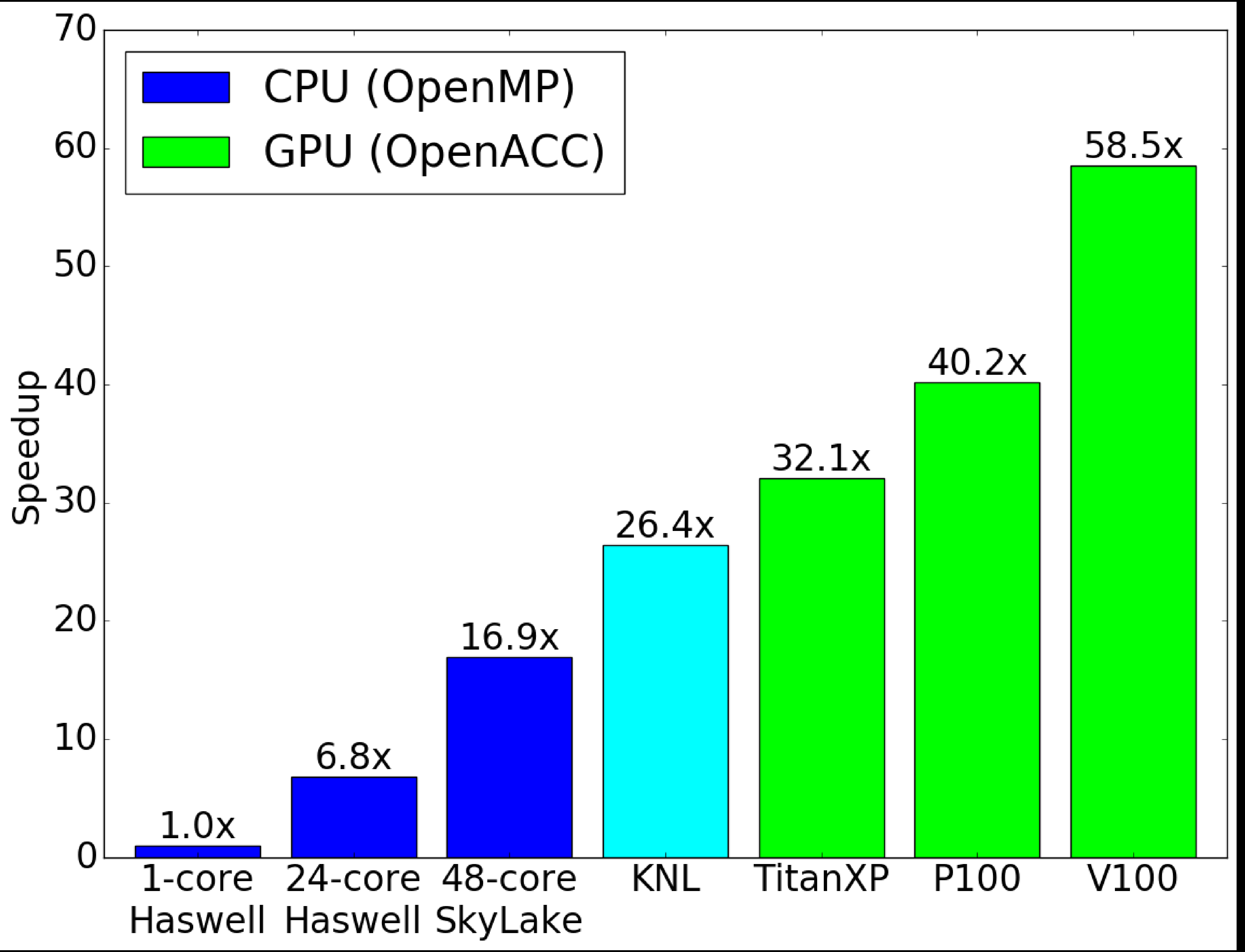
DIFFUSE Recap (3.5 million pt test)

GPU TECHNOLOGY CONFERENCE 2016

- Smooths unresolvable structure
- Integrates

$$\frac{\partial B_r}{\partial t} = \nabla_{\perp}^2 B_r$$

  with explicit super time-stepping
- Parallelized with OpenMP and OpenACC



POT3D Recap (200 million pt. test)

- Solves potential field:

$$\nabla^2\Phi = 0, \quad \mathbf{B} = \nabla\Phi$$

- MPI+OpenACC
- Preconditioned Conjugate Gradient

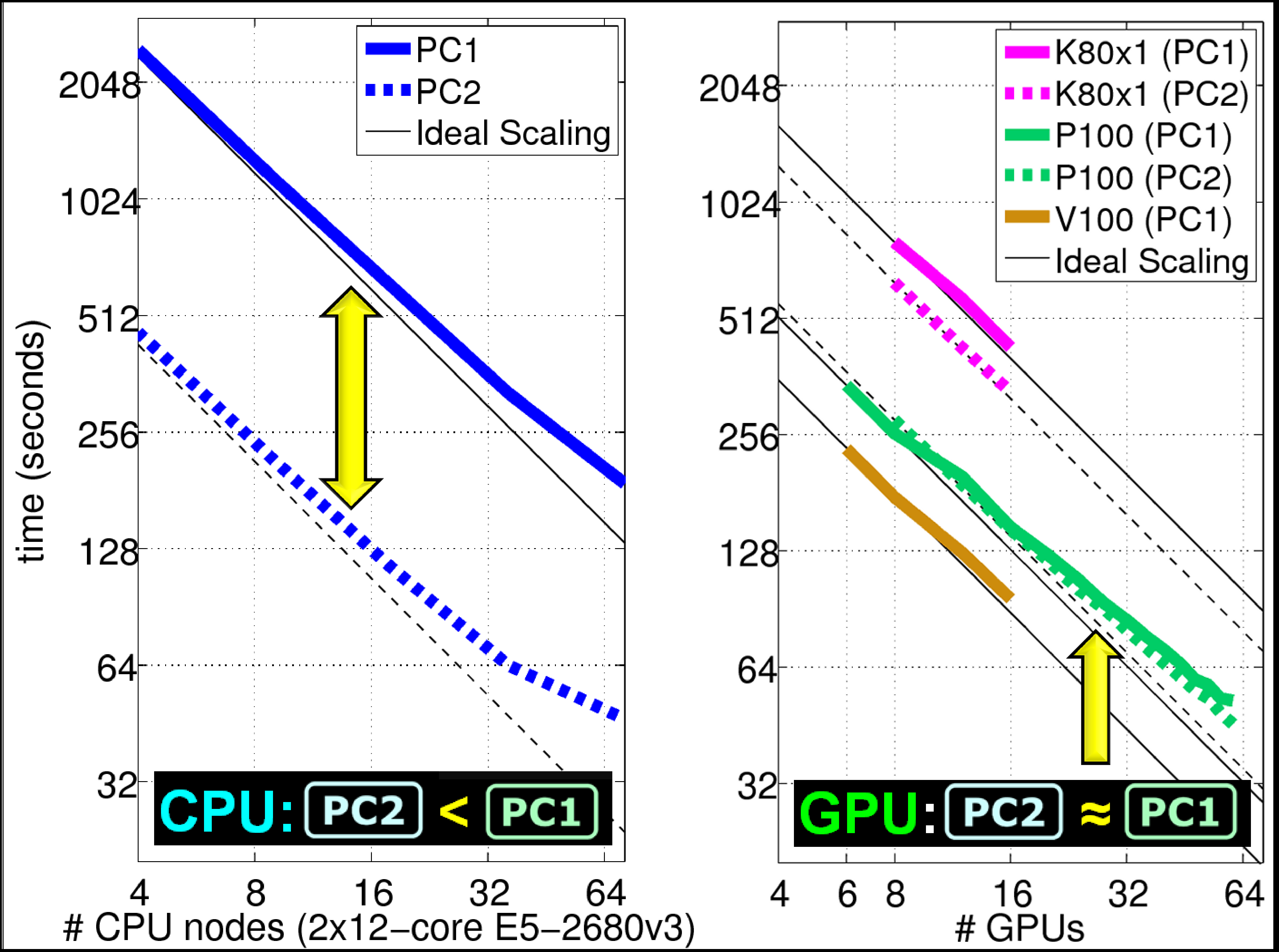
Two preconditioners:

PC1 — **Point-Jacobi**

PC2 — **Block-Jacobi with ILU0**

GPU Implementations:

PC1: pragmas only (**portable**)
PC2: cuSparse (**not portable**)

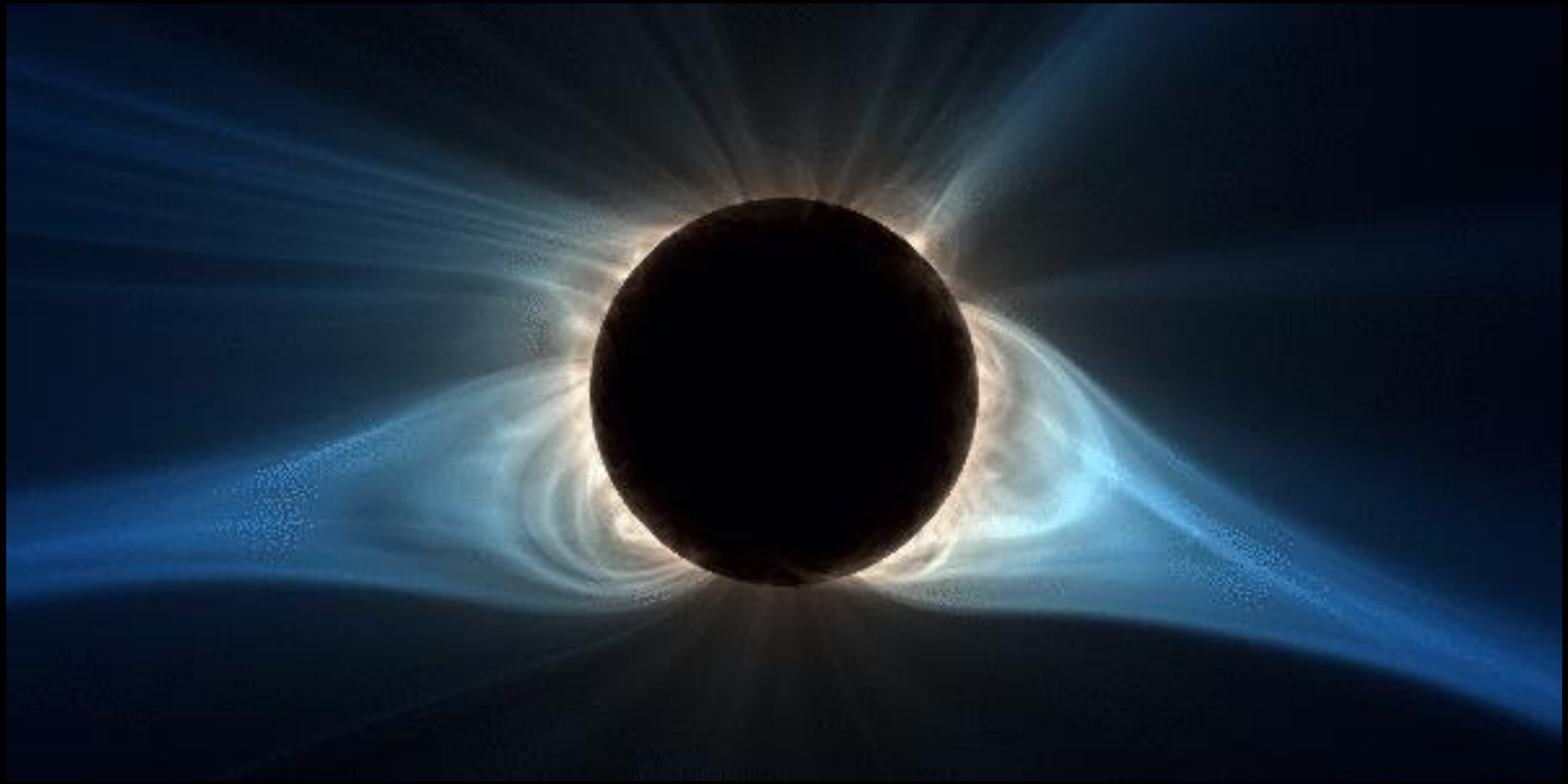


MAS

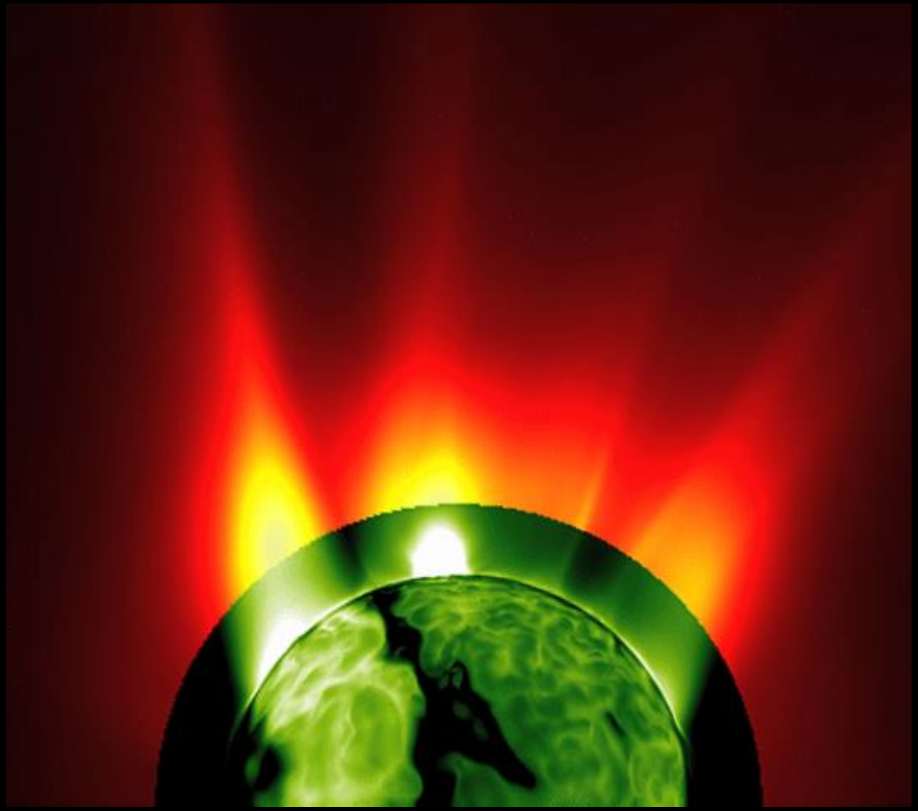
Magnetohydrodynamic Algorithm Outside a Sphere

Predictive Science Inc.

- Established MHD code with over 15 years of development used extensively in solar physics research
- Written in FORTRAN 90 (~50,000 lines), parallelized with MPI
- Available for use at the **C**ommunity **C**oordinated **M**odeling **C**enter (CCMC)



Predicted Corona of the August 21st, 2017 Total Solar Eclipse



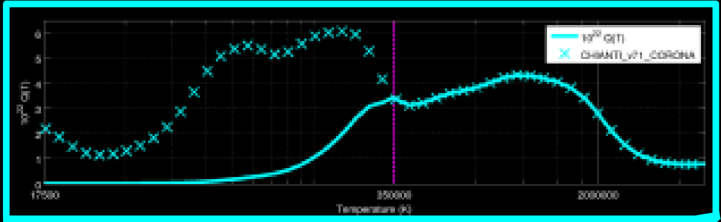
Simulation of the Feb. 13th, 2009 CME

MAS: Full MHD Model Equations

$$\frac{\partial \mathbf{A}}{\partial t} = \mathbf{v} \times (\nabla \times \mathbf{A}) - \boxed{\frac{c^2 \eta}{4\pi} \nabla \times \nabla \times \mathbf{A}}$$

RESISTIVITY

$$\frac{\partial \rho}{\partial t} = -\nabla \cdot (\rho \mathbf{v})$$



RADIATIVE COOLING

$$\frac{\partial T}{\partial t} = -\nabla \cdot (T\mathbf{v}) - (\gamma - 2)(T \nabla \cdot \mathbf{v}) + \frac{\gamma - 1}{2k} \frac{m_p}{\rho} \left[\boxed{-\nabla \cdot (\mathbf{q}_1 + \mathbf{q}_2)} - \boxed{\frac{\rho^2}{m_p^2} Q(T)} + \boxed{H} \right]$$

CORONAL HEATING

THERMAL CONDUCTION

$$\mathbf{q}_1 = -f(r)\, \beta_{\mathrm{Tcut}}(T)\, \kappa_0 T^{5/2}\, \hat{\mathbf{b}}\hat{\mathbf{b}} \cdot \nabla T$$

$$\mathbf{q}_2 = (1 - f(r)) \frac{k}{(\gamma - 1)} \frac{\rho}{m_p} T\, \mathbf{v}\, \hat{\mathbf{b}}\hat{\mathbf{b}}$$

$$H = H^* + \frac{\rho}{4\lambda_\perp} \left[|z_-|\, z_+^2 + |z_+|\, z_-^2 \right]$$

$$\lambda_\perp = \boxed{\lambda_0} \sqrt{\frac{B_w}{|\mathbf{B}|}} \qquad |z_\pm(r = R_\odot)| = \boxed{z_0}$$

ALFVÉN WAVES

$$\frac{\partial \epsilon_\pm}{\partial t} = -\nabla \cdot (\epsilon_\pm [\mathbf{v} \pm \mathbf{v}_\mathbf{A}]) - \frac{\epsilon_\pm}{2} \nabla \cdot \mathbf{v}$$

$$\frac{\partial \mathbf{v}}{\partial t} = -\mathbf{v} \cdot \nabla \mathbf{v} + \frac{1}{\rho} \left[\frac{1}{c} \mathbf{J} \times \mathbf{B} - \nabla p - \boxed{\nabla \left(\frac{\epsilon_+ + \epsilon_-}{2} \right)} + \rho \mathbf{g} \right] + \boxed{\frac{1}{\rho} \nabla \cdot (\nu \rho \nabla \mathbf{v})} + \boxed{\frac{1}{\rho} \nabla \cdot \left(S \rho \nabla \frac{\partial \mathbf{v}}{\partial t} \right)}$$

VISCOSITY

SEMI-IMPLICIT OPERATOR

WAVE TURBULENCE

$$\frac{\partial z_\pm}{\partial t} = -(\mathbf{v} \pm \mathbf{v}_\mathbf{A}) \cdot \nabla z_\pm - \frac{z_\pm |z_\mp|}{2\lambda_\perp} + \frac{z_\pm}{4} (\mathbf{v} \mp \mathbf{v}_\mathbf{A}) \cdot \nabla (\ln \rho) + \frac{z_\mp}{2} (\mathbf{v} \mp \mathbf{v}_\mathbf{A}) \cdot \nabla (\ln |\mathbf{v}_\mathbf{A}|)$$

$$\nabla \cdot \mathbf{B} = 0 \qquad \mathbf{B} = \nabla \times \mathbf{A} \qquad \mathbf{J} = \frac{c}{4\pi} \nabla \times \mathbf{B}$$

$$p = 2kT\rho/m_p \qquad \mathbf{g} = -g_0 R_\odot^2 \hat{\mathbf{r}}/r^2 \qquad \gamma = 5/3$$

$$\hat{\mathbf{b}} = \mathbf{B}/|\mathbf{B}| \qquad \mathbf{v}_\mathbf{A} = \mathbf{B}/\sqrt{4\pi\rho} \qquad B_w = 6.09\,G \qquad v_c^2 = \gamma p/\rho$$

$$\beta_{\mathrm{Tcut}}(T) = \begin{cases} (T/T_{\mathrm{cut}})^{-5/2} & T < T_{\mathrm{cut}} \\ 1, & T \geq T_{\mathrm{cut}} \end{cases} \qquad T_{\mathrm{cut}} = 3.5 \times 10^5\,K$$

$$f(r) = 1 - 0.5 \tanh\left[(r - 10 R_\odot)/R_\odot \right]$$

$$S = (\Delta t^2 \tilde{k}^2)^{-1} \left(C_w^2/(1 - C_f)^2 - 1 \right) \qquad C_f = \Delta t\, \tilde{k} \cdot \mathbf{v}$$

$$C_w^2 = 0.25\, \Delta t^2 \tilde{k}^2 (v_c^2 + |\mathbf{v}_\mathbf{A}|^2) \qquad \tilde{k}^2 = 4 \left(\Delta r^{-2} + (r\Delta\theta)^{-2} + (r \Delta\phi \sin\theta)^{-2} \right)$$

MAS: MHD Model Equations ("Zero-Beta")

- In the low corona outside of active regions, the plasma beta is very small (i.e. dynamics dominated by magnetic field)
- Therefore, one can approximate the magnetic field and onset dynamics of the CME eruption with a simplified "zero-beta" form of the MHD equations

RESISTIVITY

$$\frac{\partial \mathbf{A}}{\partial t} = \mathbf{v} \times (\nabla \times \mathbf{A}) - \boxed{\frac{c^2\,\eta}{4\,\pi} \nabla \times \nabla \times \mathbf{A}}$$

$$\frac{\partial \mathbf{v}}{\partial t} = -\mathbf{v} \cdot \nabla\,\mathbf{v} + \frac{1}{\rho}\left[\frac{1}{c}\mathbf{J} \times \mathbf{B}\right] + \boxed{\frac{1}{\rho}\nabla \cdot (\nu\rho\nabla\mathbf{v})} + \boxed{\frac{1}{\rho}\nabla \cdot \left(S\,\rho\,\nabla\frac{\partial \mathbf{v}}{\partial t}\right)}$$

VISCOSITY

SEMI-IMPLICIT OPERATOR

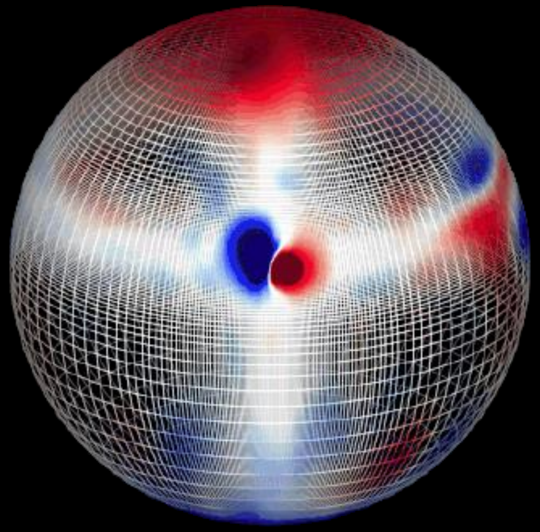
$$\boxed{\begin{aligned} p &= 0 \\ \rho &= \rho_0(\mathbf{r}) \end{aligned}}$$

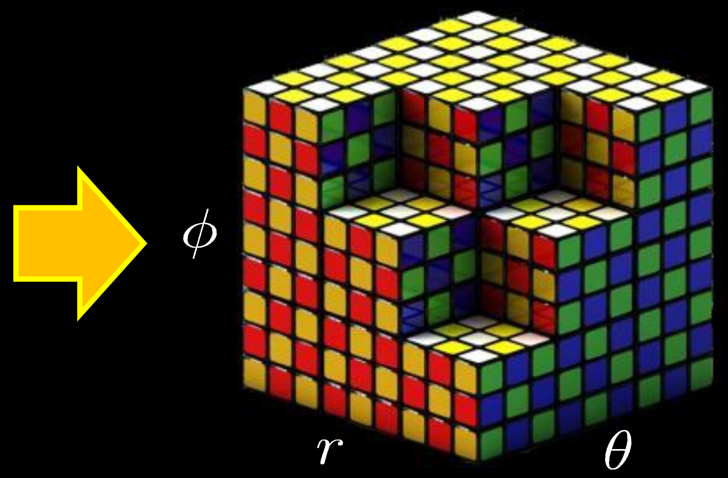
$$\mathbf{B} = \nabla \times \mathbf{A} \qquad \nabla \cdot \mathbf{B} = 0 \qquad v_A^2 = |\mathbf{B}|^2/(4\,\pi\,\rho) \qquad S = (\Delta t^2\,\tilde{k}^2)^{-1}\left(C_w^2/(1-C_f)^2 - 1\right) \qquad C_w^2 = 0.25\,\Delta t^2\,\tilde{k}^2\,(v_c^2 + v_A^2)$$

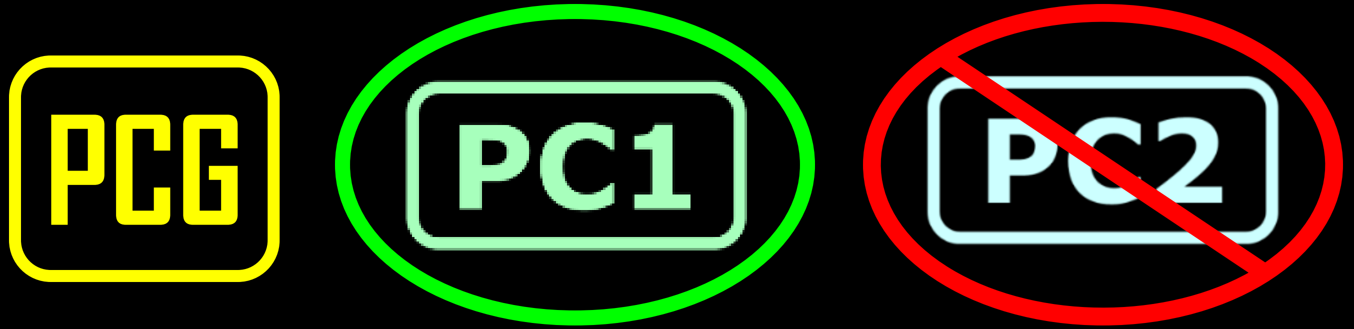
$$\mathbf{J} = \frac{c}{4\,\pi}\nabla \times \mathbf{B} \qquad \mathbf{g} = 0 \qquad C_f = \Delta t\,\tilde{k} \cdot \mathbf{v} \qquad v_c = 0 \qquad \tilde{k}^2 = 4\left(\Delta r^{-2} + (r\,\Delta\theta)^{-2} + (r\,\Delta\phi\,\sin\theta)^{-2}\right)$$

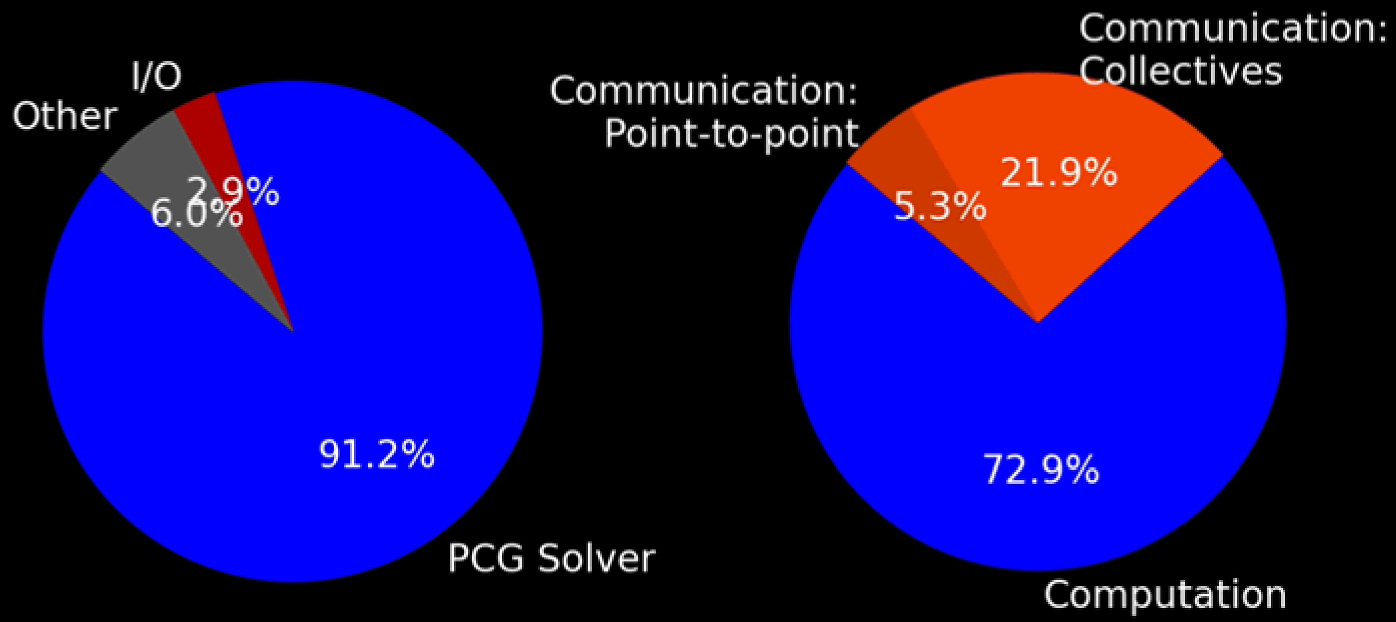
MAS: Algorithm Summary and Profile

- Finite difference on non-uniform spherical grid
- Explicit and implicit time-step algorithms
- PCG used to solve implicit steps
- Sparse matrix operators stored in mDIA format, PC2 ILU0 matrix stored in CSR
- PCG solvers use the same PCs in **POT3D**. Since GPU results showed PC1~PC2, we only implement PC1 in MAS (**portable!**)
- PCG solvers are ~90% of run-time!









TEST1 run on 16 nodes of 24-core Haswell CPUs (PC2)

OpenACC Implementation: Quick Picks

Most implementation details the same as POT3D (see our GTC17 talk)

Don't use cutting edge features if you're afraid of getting cut!

`deepcopy, managed, zeroinit, cache, tile`

Managed Memory
Transition from managed memory to manual memory can be a BIG, all-or-nothing step

CPU Redundant Routines
Some calls use GPU, some don't. OpenACC "`if/if_present`" conditional clauses to the rescue! (PGI >18.1)

Valuable PGI ENVs

```
PGI_ACC_DEBUG
PGI_ACC_NOTIFY
PGI_ACC_TIME
PGI_ACC_PROFILE
PGI_ACC_FILL
```

GPU Data Residency
Avoiding GPU-CPU data transfers can involve increased development time due to many small (possibly awkward) routines

OpenACC Implementation: Derived Types

Fortran Derived Types

```
type :: vvec
  real, dimension(:,:,:), allocatable :: r
  real, dimension(:,:,:), allocatable :: t
  real, dimension(:,:,:), allocatable :: p
end type
type :: vvec_bc
  type(vvec) :: r0
  type(vvec) :: r1
end type
```

```
type(vvec),target :: v
type(vvec_bc),target :: v_bc
```

(Allocations…)

```
!$acc parallel loop collapse(2)
!$acc& default(present)
do j=2,ntm1
  do i=2,nrm-1
    v%r(i,j,2)=v%r(i,j,2)+v_bc%r0%r(i,j,2)
  enddo
enddo
```

"Manual" Deep-copy

```
!$acc enter data create(v,v%r,v%t,v%p)

!$acc enter data create(
!$acc&    v_bc,v_bc%r0,v_bc%r1,
!$acc&    v_bc%r0%r,v_bc%r0%t,v_bc%r0%p,
!$acc&    v_bc%r1%r,v_bc%r1%t,v_bc%r1%p)
```

"True" Deep-copy (PGI: *–ta:tesla,deepcopy*)

```
!$acc enter data create(v)
!$acc enter data create(v_bc)
```

- "True" Deep-copy + CUDA-aware MPI weren't playing nicely, so we used manual deep-copy
- Due to compiler bug (fixed in PGI ≥17.10), had to change code to only use single-level types
- Due to compiler bug (PGI ≥17.10) with CUDA-aware MPI + types, used PGI 17.9 (work-around found)

OpenACC Implementation: Array Reductions

Array Reductions

OpenACC scalar reductions

```
real(r_typ) :: sum
!$acc kernels loop
!$acc& reduction(+:sum)
do j=1,m
  sum=sum+a(j)
enddo
```

OpenACC does *not* directly support array reductions

```
allocate(sum(n))
do j=1,m
  sum(:)=sum(:)+a(:,j)
enddo
```

Two example options

(1)

```
!$acc kernels
!$acc loop
do j=1,m
!$acc loop
   do i=1,n
!$acc atomic update
     sum(i)=sum(i)+a(i,j)
enddo
```

(2)

```
!$acc kernels loop
do i=1,n
  sum(i)=SUM(a(i,1:m))
enddo
```

Timing results of 1 step of TEST1 on TitanXP

	GPU	CPU	CPU (full routine)
(1)	0.25	2.3	21.3
(2)	0.46	2.5	17.6

We use option (2) for code simplicity

Full routine only 0.03% of total run time

OpenACC Implementation: Performance Tuning

For TEST1, ~60% of wall-time in computing velocity matrix multiply routine

Cache-friendly vs Vector-friendly

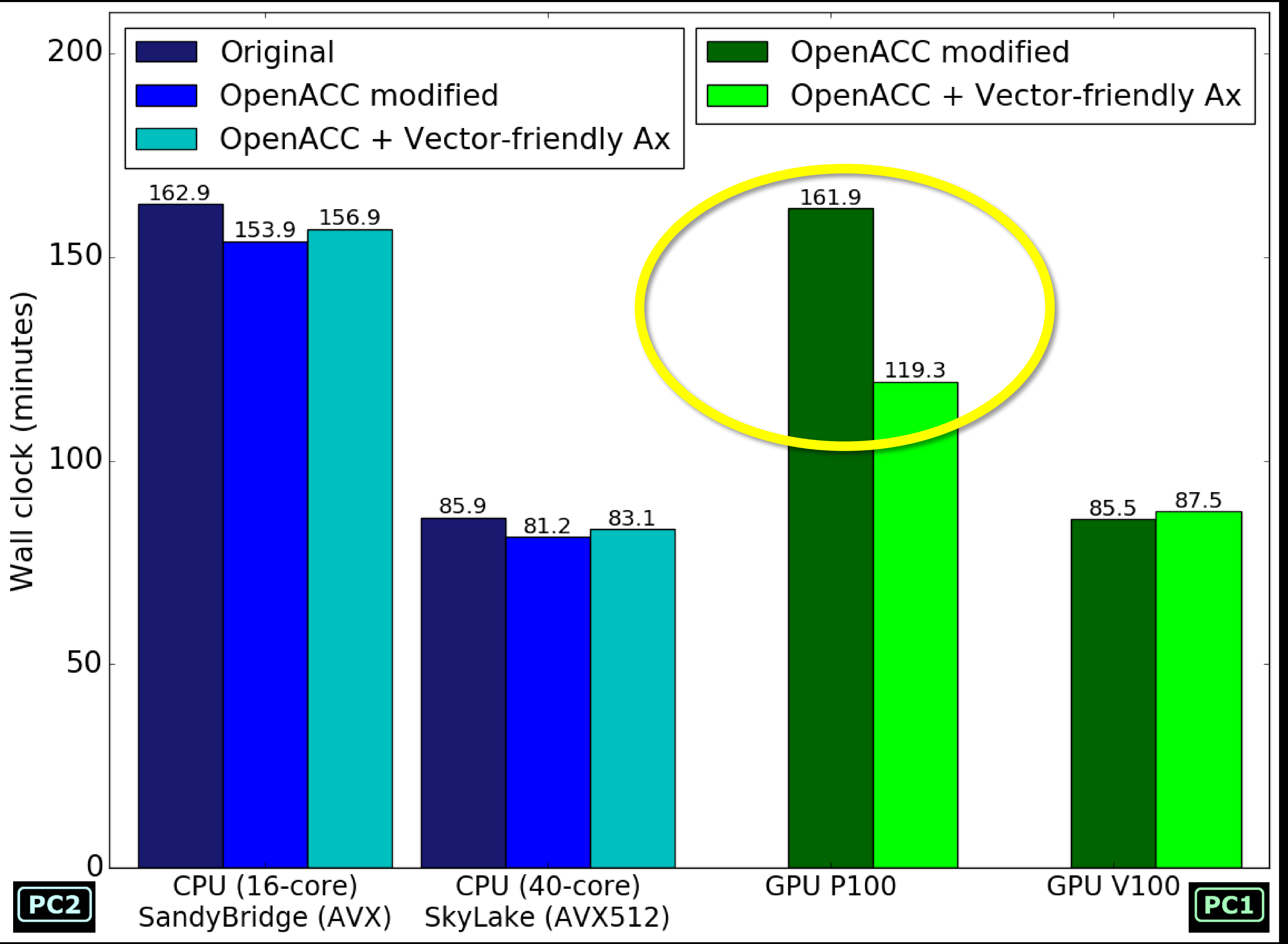
```
do k=2,npm1
 do j=2,ntm1
  do i=2,nrm-1
   ii=ntm2*(nrm-2)*(k-2)
      +(nrm-2)*(j-2)+(i-1)
   q(ii)=a_r(1,i,j,k)*ps%r(i  ,j  ,k-1)
        +a_r(2,i,j,k)*ps%r(i  ,j-1,k  )
        . . .
        . . .
        . . .
        +a_r(14,i,j,k)*ps%p(i  ,j  ,k  )
        +a_r(15,i,j,k)*ps%p(i+1,j  ,k  )
  enddo
 enddo
enddo
```

```
do k=2,npm1
 do j=2,ntm1
  do i=2,nrm-1
   ii=ntm2*(nrm-2)*(k-2)
      +(nrm-2)*(j-2)+(i-1)
   q(ii)=a_r( i,j,k,1)*ps%r(i  ,j  ,k-1)
        +a_r( i,j,k,2)*ps%r(i  ,j-1,k  )
        . . .
        . . .
        . . .
        +a_r(i,j,k,14)*ps%p(i  ,j  ,k  )
        +a_r(i,j,k,15)*ps%p(i+1,j  ,k  )
  enddo
 enddo
enddo
```

OpenACC Implementation: Performance Tuning

Cache vs Vector Results (TEST1)

- CPU: Vector-friendly version slower, but still faster than original code
- GPU: Vector-friendly version much faster on P100, little change on V100



Predictive Science Inc.

OpenACC Implementation: Performance Tuning

parallel, kernels, gangs, workers, vectors … oh my!

- Many configuration options (hardware narrows choices a bit)

```
    (kernels)
!$acc parallel default(present) present(ps) async(1)
!$acc loop
      do k=2,npm1
!$acc loop
        do j=2,ntm1
!$acc loop
          do i=2,nrm-1
           …
```

vr,vt,vp computed asynchronously

num_gangs(#), num_workers(#), num_vectors(#)
gang, worker(#), vector(#), seq

- We test various clause options with 1 step of TEST1 on a TitanXP GPU (timing routine using PGI_ACC_TIME=1)

OpenACC Implementation: Performance Tuning

parallel

kernels

Source	PGI 17.9 Output	Time (s)

`!$acc parallel`
`!$acc loop`
`!$acc loop`
`!$acc loop`
	`!$acc loop gang ! blockidx%x`
	`!$acc loop seq`
	`!$acc loop vector(128) ! threadidx%x`
		60.3

`!$acc parallel vector_length(32)`
`!$acc loop`
`!$acc loop`
`!$acc loop`
	`!$acc loop gang ! blockidx%x`
	`!$acc loop seq`
	`!$acc loop vector(32) ! threadidx%x`
		55.7

`!$acc parallel vector_length(16)`
`!$acc loop`
`!$acc loop`
`!$acc loop`
	`!$acc loop gang ! blockidx%x`
	`!$acc loop seq`
	`!$acc loop vector(16) ! threadidx%x`
		76.8

`!$acc loop independent`
`!$acc loop independent`
`!$acc loop independent`
	`!$acc loop gang ! blockidx%y`
	`!$acc loop gang, vector(4) ! blockidx%z threadidx%y`
	`!$acc loop gang, vector(32) ! blockidx%x threadidx%x`
		45.7

`!$acc loop independent gang worker vector`
`!$acc loop independent gang worker vector`
`!$acc loop independent gang worker vector`
	`!$acc loop gang ! blockidx%z`
	`!$acc loop gang, vector(4) ! blockidx%y threadidx%z`
	`!$acc loop gang, worker(2), vector(64) ! blockidx%x threadidx%y threadidx%x`
		47.7

`!$acc loop independent gang`
`!$acc loop independent gang worker`
`!$acc loop independent gang vector`
	`!$acc loop gang ! blockidx%z`
	`!$acc loop gang, worker(4) ! blockidx%y threadidx%y`
	`!$acc loop gang, vector(32) ! blockidx%x threadidx%x`
		49.1

`!$acc loop independent gang`
`!$acc loop independent vector(8)`
`!$acc loop independent vector(8)`
	`!$acc loop gang ! blockidx%x`
	`!$acc loop gang, vector(8) ! blockidx%z threadidx%y`
	`!$acc loop gang, vector(8) ! blockidx%y threadidx%x`
		151.84

`!$acc loop independent gang`
`!$acc loop independent gang vector(8)`
`!$acc loop independent gang vector(8)`
	`!$acc loop gang ! blockidx%x`
	`!$acc loop gang, vector(8) ! blockidx%z threadidx%y`
	`!$acc loop gang, vector(8) ! blockidx%y threadidx%x`
		91.93

OpenACC Implementation: Effort Summary

≈1%

OpenACC comment lines

≈8%

Total added, deleted, and changed lines

Details

Total lines in original code	51,591
Total lines in accelerated code	54,191
Total !$acc/!$acc& lines added	671 (1.0%)
Total modified lines	844 (1.6%)
Total # of additional lines	2600 (5.0%)
Total # of different lines	4314 (8.0%)

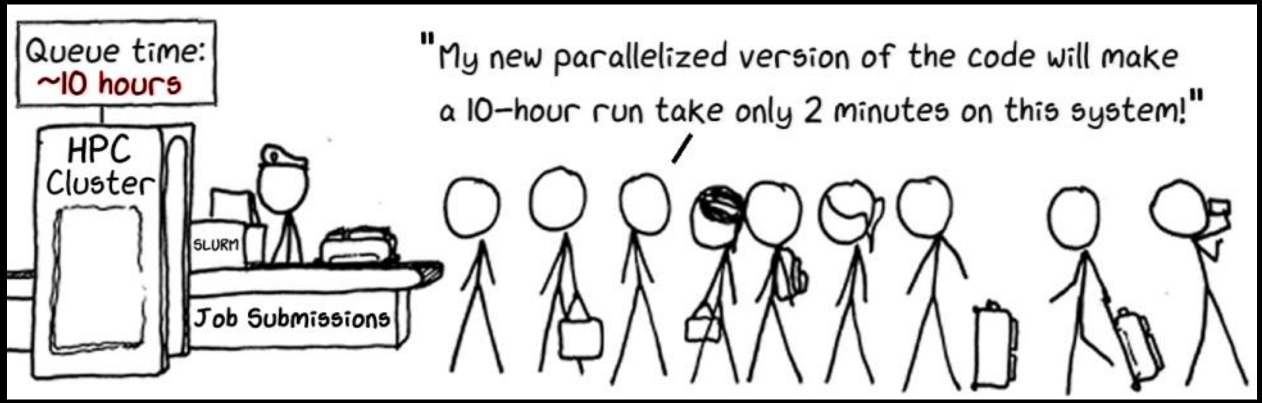
Factors to consider

- **Added Lines:** Duplicate CPU routines (can remove with OpenACC 2.6 conditionals)
- **Deleted Lines:** Optional CPU code simplifications
- **Modified Lines:** CPU changes for array reductions, vector-friendly matrix multiply, and single-level derived types (temporary)
- **OpenACC Comment Lines:** Full code not accelerated (zero-beta parts only!)

Single portable source for GPU and CPU!

Timing Procedures

- "Time-to-solution" includes I/O, comm,setup, etc. (Queue times excluded, but important!)
- Acceptable "time-to-solution" for TEST1 & TEST2 set by current pipeline (not cherry picked!)
- We use best available compiler, compiler version, instruction sets, library versions, and ***algorithm*** for each system



GPU **CPU**

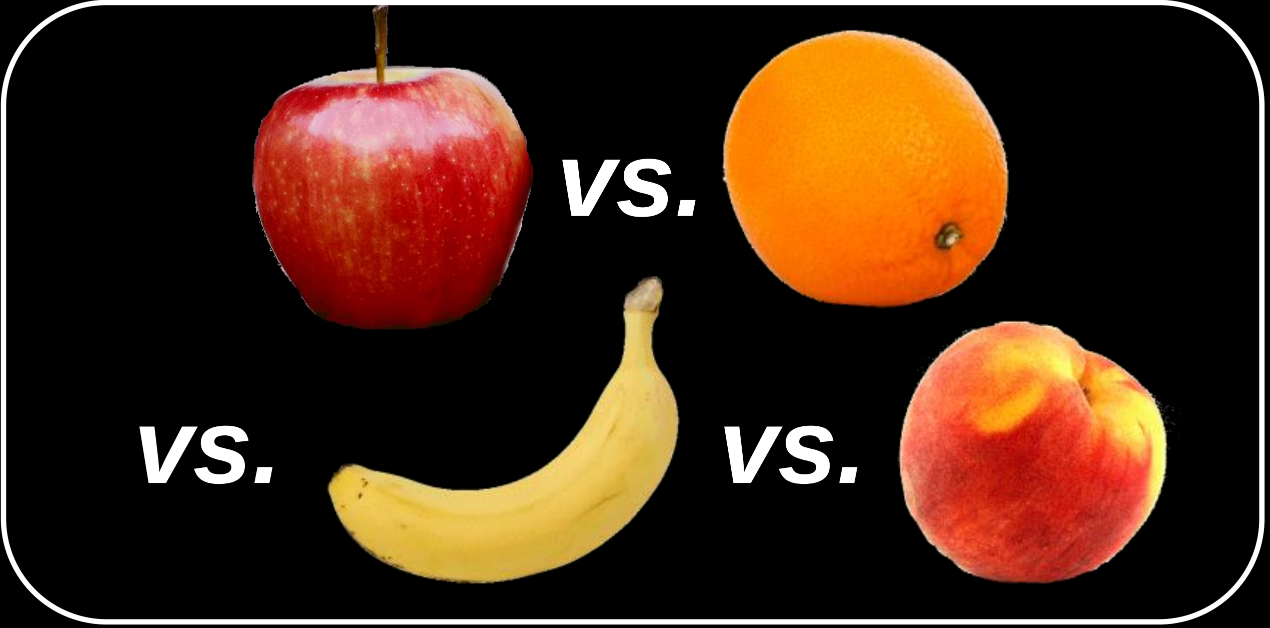
PC1 ***vs.*** **PC2**

Why is this fair?

We care about the "real" world

We are not benchmarking hardware

We want to test the maximum performance on each system for solving our problem, using our code



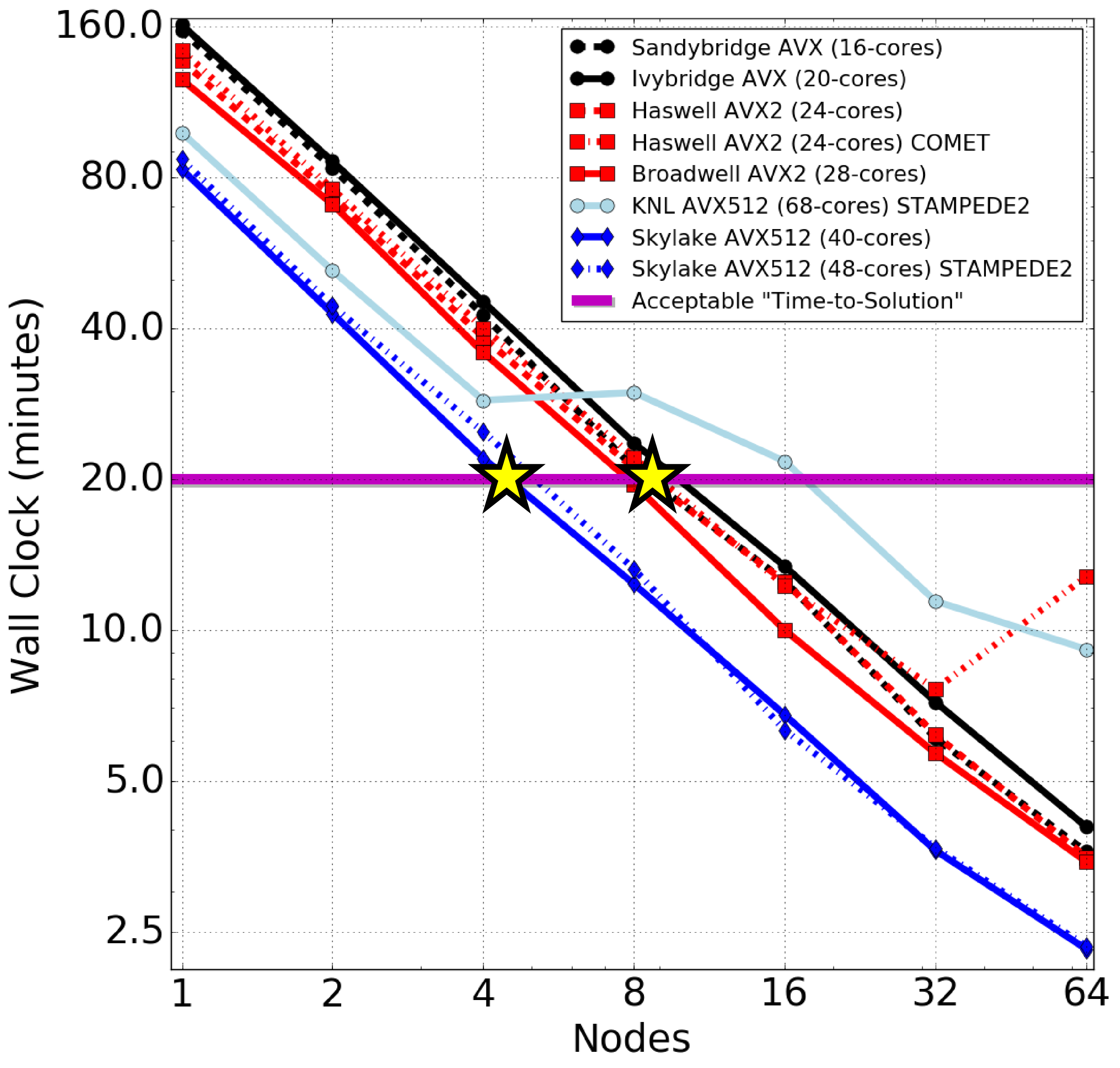
Hardware and Environments

	NASA NAS Pleiades & Electra					SDSC Comet	TACC Stampede2	
Compiler	Intel 2018 .0.128					Intel 2016.3.210	Intel 18.0.0	
MPI Library	SGI MPT 2.15r20					MVAPICH2 v2.1	Intel MPI 18.0.0	
Intel Family	**Sandy Bridge**	**Ivy Bridge**	**Haswell**	**Broadwell**	**Skylake**	**Haswell**	**KNL**	**Skylake**
Instruction Set	AVX	AVX	AVX2	AVX2	AVX512	AVX2	AVX512	AVX512
Processor	E5-2670	E5-2680v2	E5-2680v3	E5-2680v4	Gold 6148	E5-2680v3	Phi 7250	Platinum 8160
Clock Rate	2.6 GHz	2.8 GHz	2.5 GHz	2.4 GHz	2.4 GHz	2.5 GHz	1.4 GHz	2.1 GHz
# Cores	16	20	24	28	40	24	68	48
Memory Bandwidth	**51.2 GB/s**	**59.7 GB/s**	**68 GB/s**	**76.8 GB/s**	**128 GB/s**	**68 GB/s**	**115.2 GB/s**	**128 GB/s**

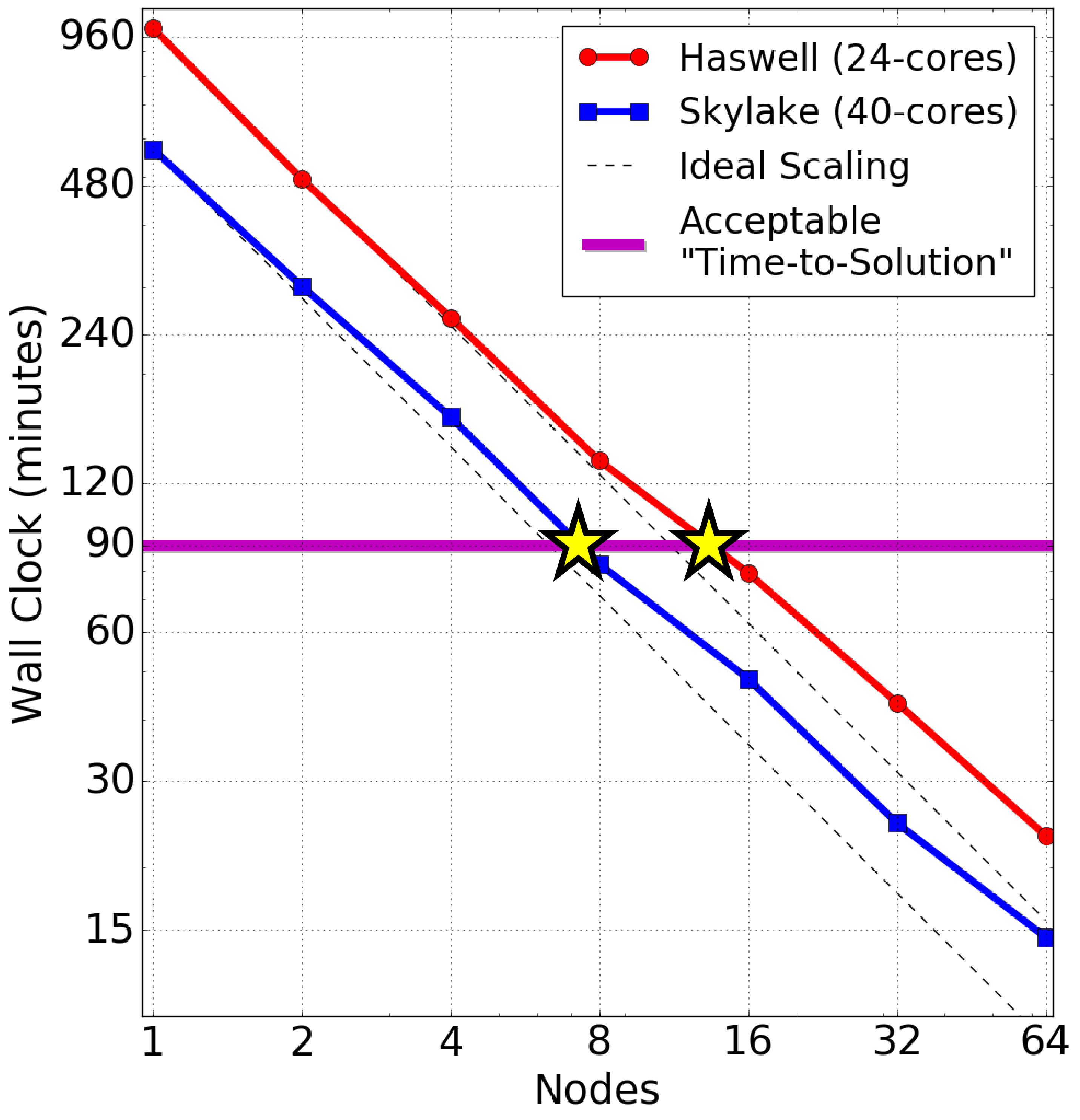
	NVIDIA PSG		SDSC Comet
Compiler	PGI 17.9		PGI 17.10
MPI Library	OpenMPI 1.10.7		OpenMPI 2.1.2
CUDA Library	CUDA 9.0.176		CUDA 8.0
Driver Version	387.26		367.48
Model (# GPUs/node)	**P100 PCIE (4)**	**V100 PCIE (4)**	**P100 PCIE (4)**
Compute Capability	6.0	7.0	6.0
Clock Rate	1.33 GHz	1.38 GHz	1.33 GHz
# CUDA DP Cores	1792	2560	1792
Memory Bandwidth	**732 GB/s**	**900 GB/s**	**732 GB/s**

Timing Results CPU (PC2)

TEST1



TEST2

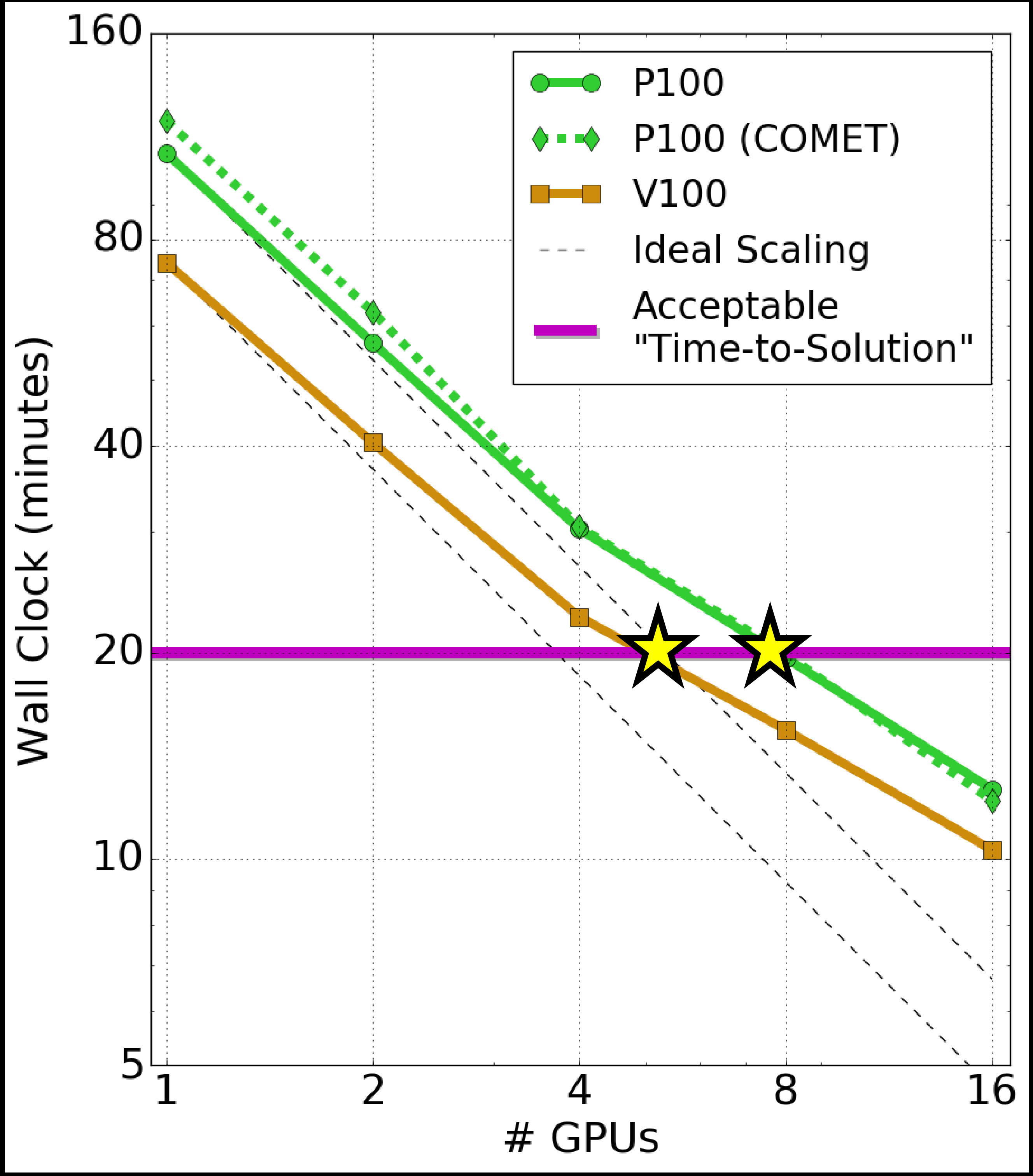


Timing Results GPU (PC1)

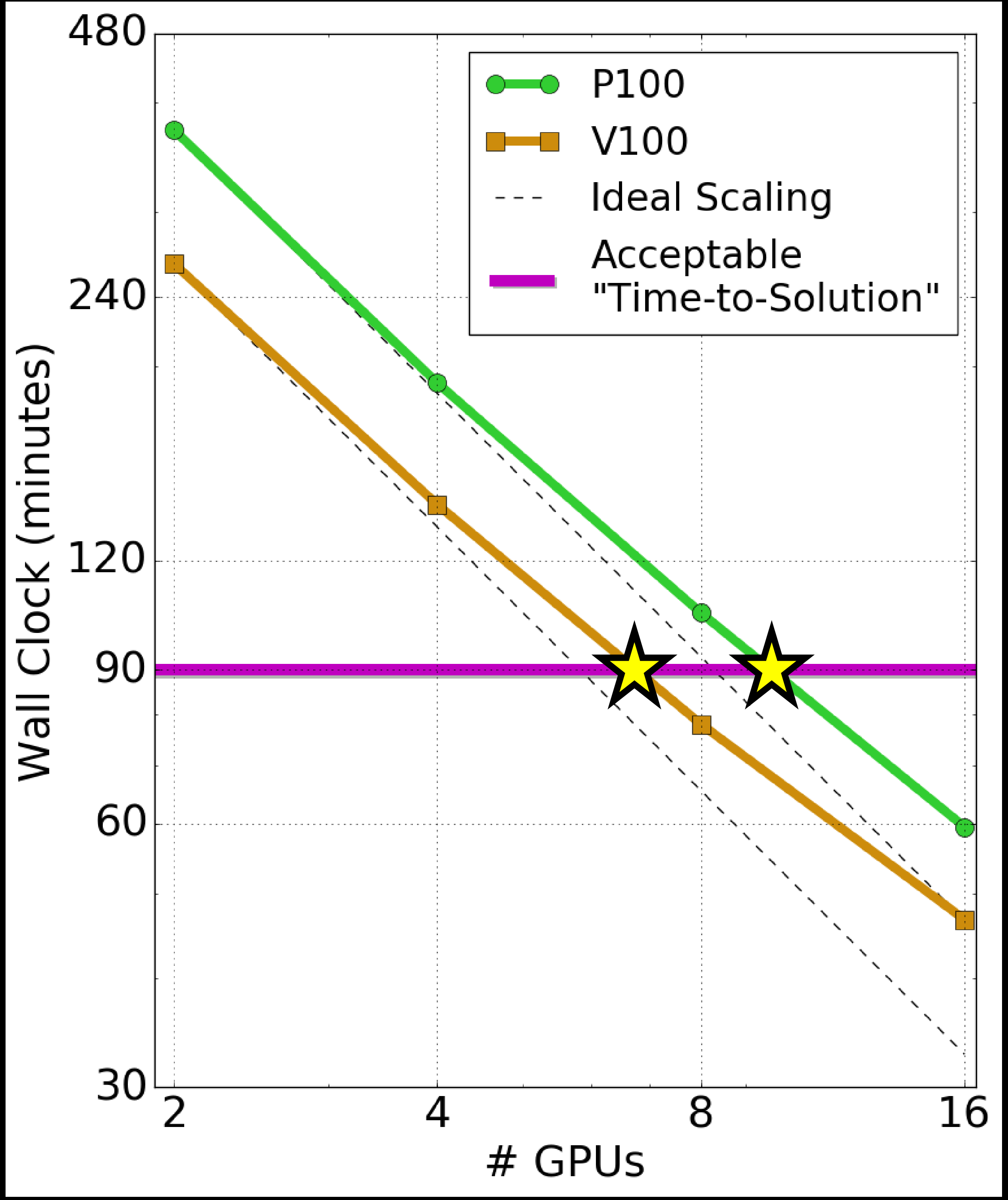
4x PCIe GPUs
per node

NVlink

TEST1

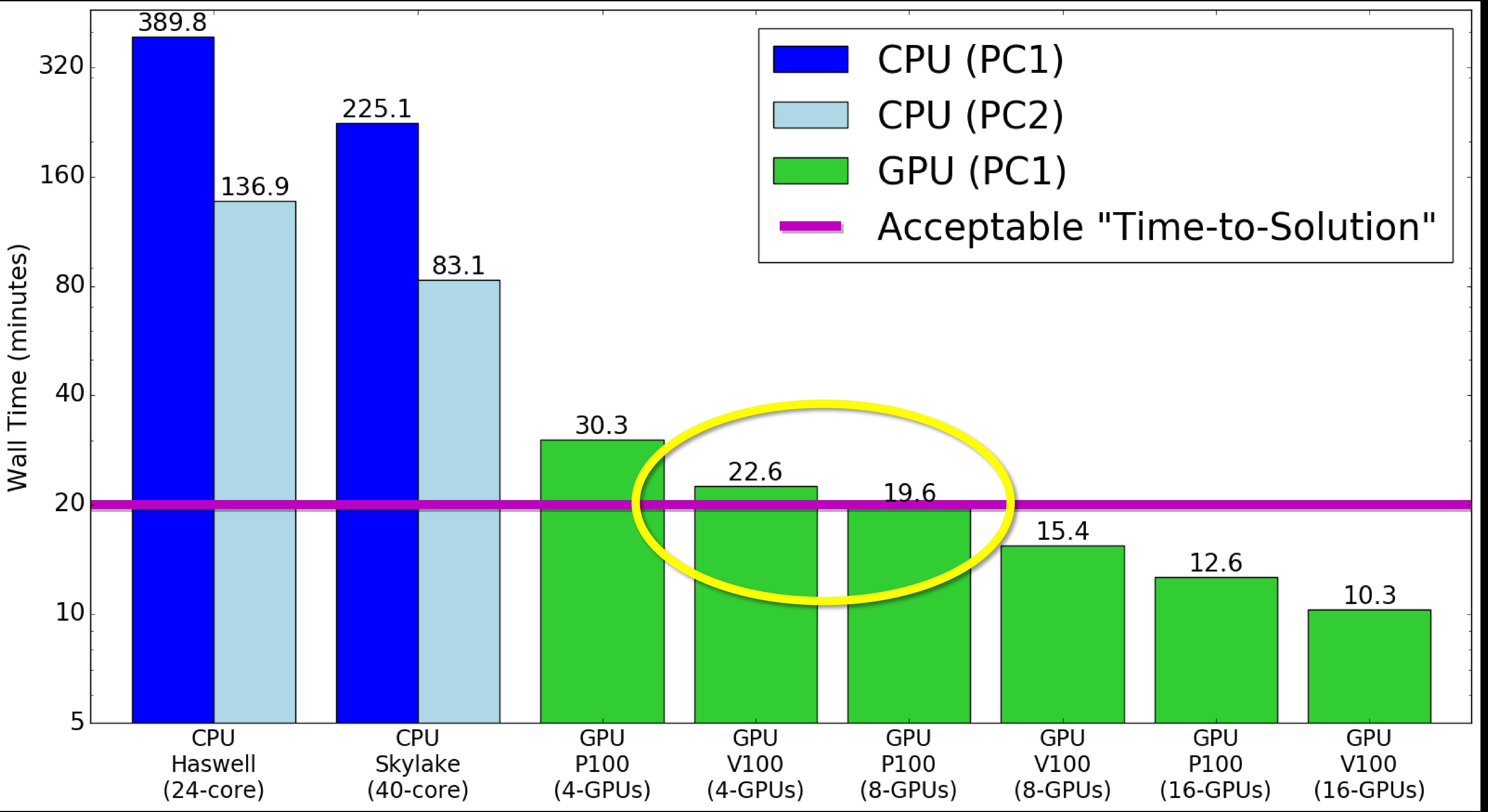


TEST2



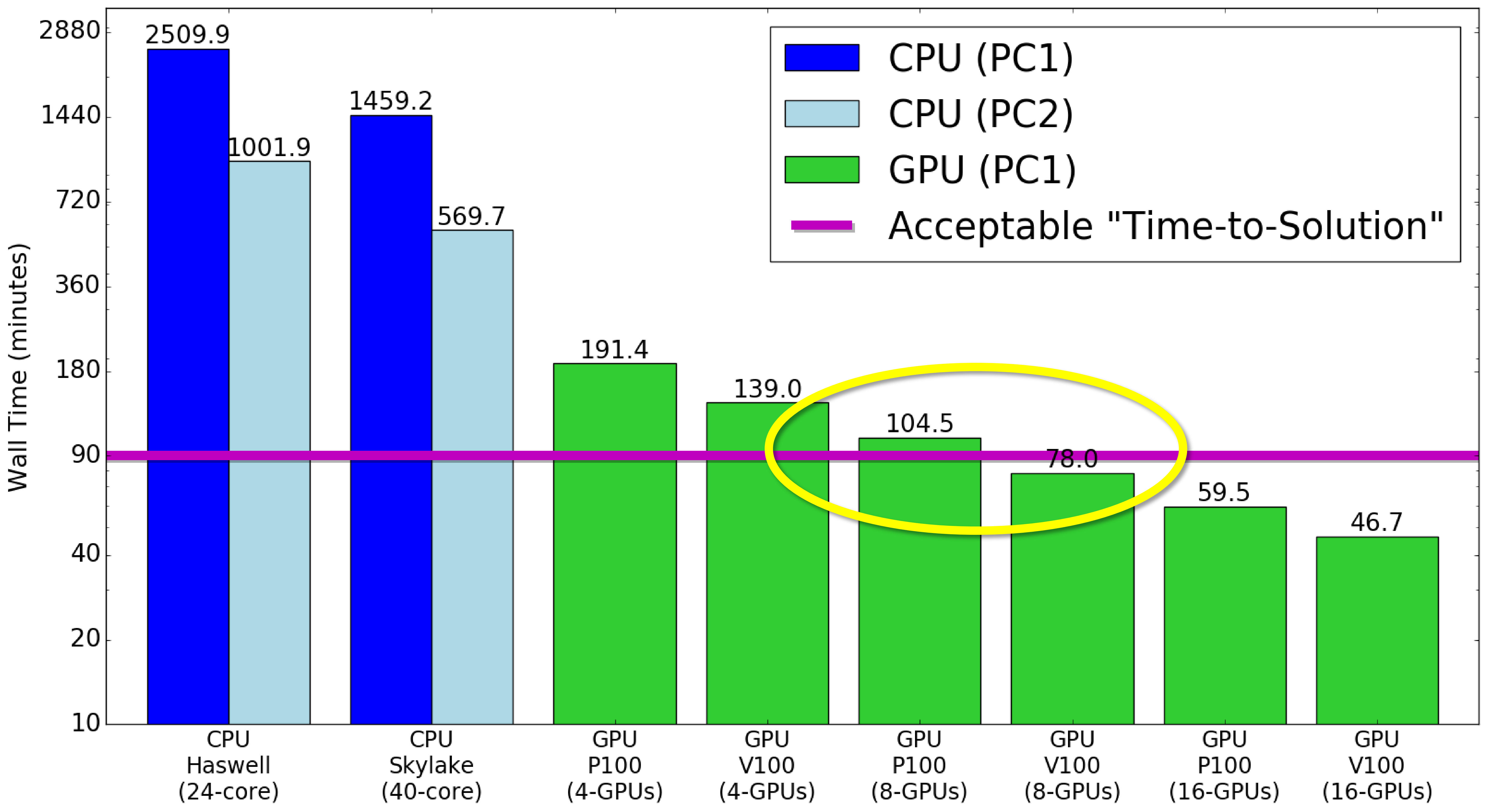
Timing Results Single Node ("In-house")

TEST1



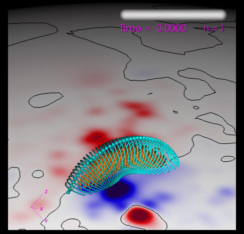
Timing Results Single Node ("In-house")

TEST2

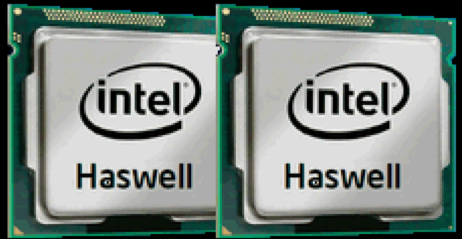


Performance Summary

TEST1:
Acceptable time-to-solution: 20 min



≈ 8x



2x12-core **Haswell** Nodes

≈ 4x

2x20-core **Skylake** Nodes

PC2

≈ 8x

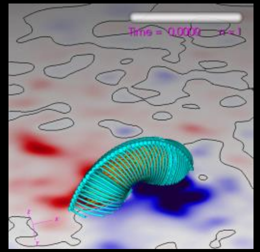
P100

≈ 4x

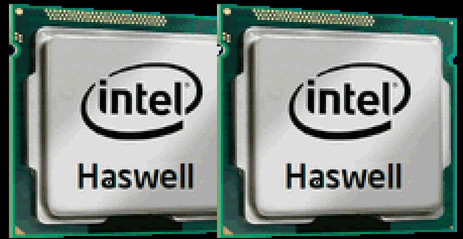
V100

PC1

TEST2:
Acceptable time-to-solution: 90 min



≈ 16x



2x12-core **Haswell** Nodes

≈ 8x

2x20-core **Skylake** Nodes

PC2

≈ 8x

P100

≈ 8x

V100

PC1

Summary and Outlook

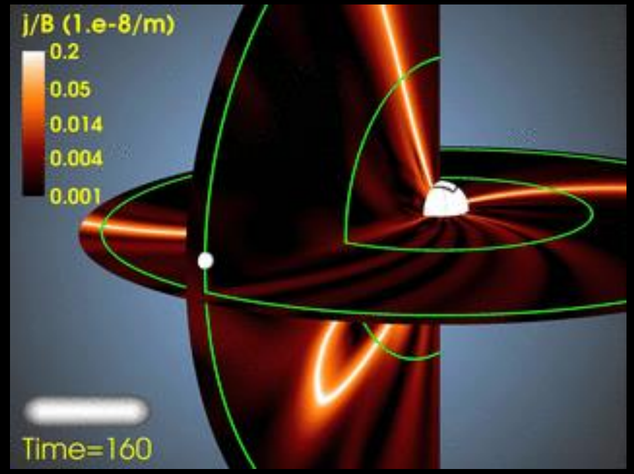
THE BIG IDEA:
Can we achieve the same acceptable "time-to-solutions" on a single multi-GPU node using OpenACC in a *portable*, *single-source* implementation?

Yup!

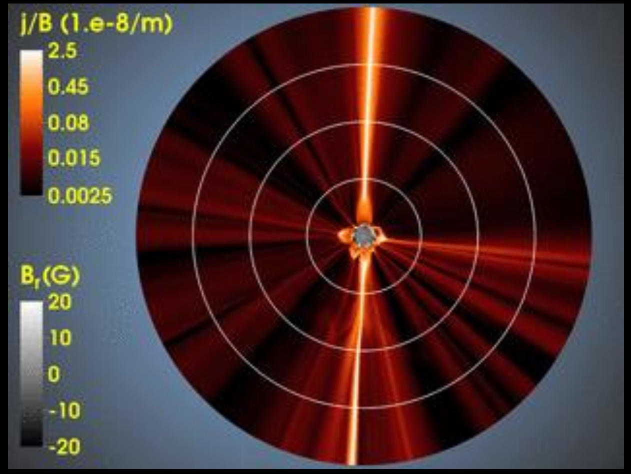
4xGPU Workstation

8xGPU Server

- For TEST1 and TEST2 (representative of many cases), we can move from HPC cluster to "in-house"!
- Future improvements
  (Give PC2 another go? Mixed-precision?)
- Next steps in OpenACC implementation of MAS:
  - Heliospheric runs
    (where PC1 is most efficient on the CPU runs)
  - Thermodynamic runs
    (Using many multiple-GPU nodes)



Heliospheric CME Simulation



Thermodynamic CME Simulation

Questions?

OpenACC User Group
Directives for Accelerators —> More Science. Less programming

Twitter @OpenACCorg
Facebook @OpenACCorg
LinkedIn OpenACC Developers

This work was supported by
- NSF's Frontiers in Earth System Dynamics program
- NASA's Living with a Star program
- Air Force Office of Scientific Research

We gratefully acknowledge NVIDIA Cooperation for donating allocation use of their PSG Cluster for GPU timings.

Predictive Science Inc.