GPU-acceleration of an Established Solar MHD code using OpenACC

Ronald M. Caplan, Jon A. Linker,
Zoran Mikić, Cooper Downs, and Tibor Török

Predictive Science Inc.

Slides available at:
predsci.com/~caplanr

ASTRONUM 2018

13TH INTERNATIONAL CONFERENCE ON NUMERICAL MODELING OF SPACE PLASMA FLOWS

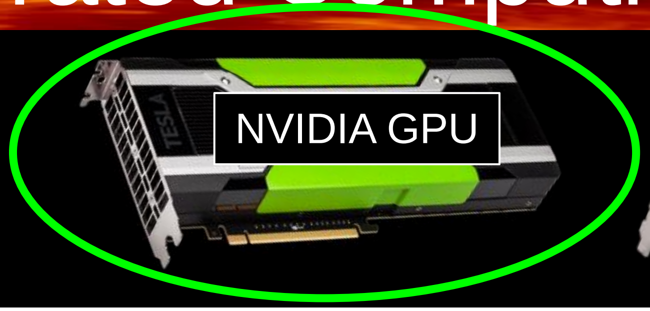
JUNE 25 – 29, 2018, PANAMA CITY BEACH, FL

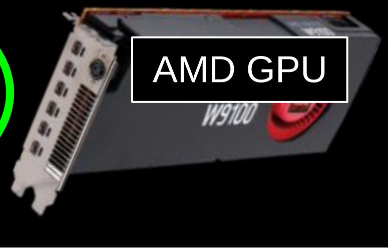
Outline

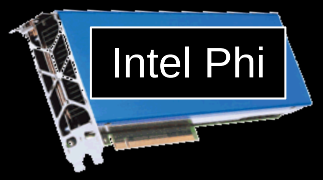
- Accelerated Computing
- OpenACC
- The MAS Code
- OpenACC Implementation
- Results
- Outlook

Accelerated Computing

An accelerator is a discrete piece of hardware designed for massively parallel computations







Many brands/types of accelerators, here we focus on NVIDIA GPUs.

Why use accelerators?

1) Performance (FLOP/s and Memory Bandwidth)

2) **Compact Performance**

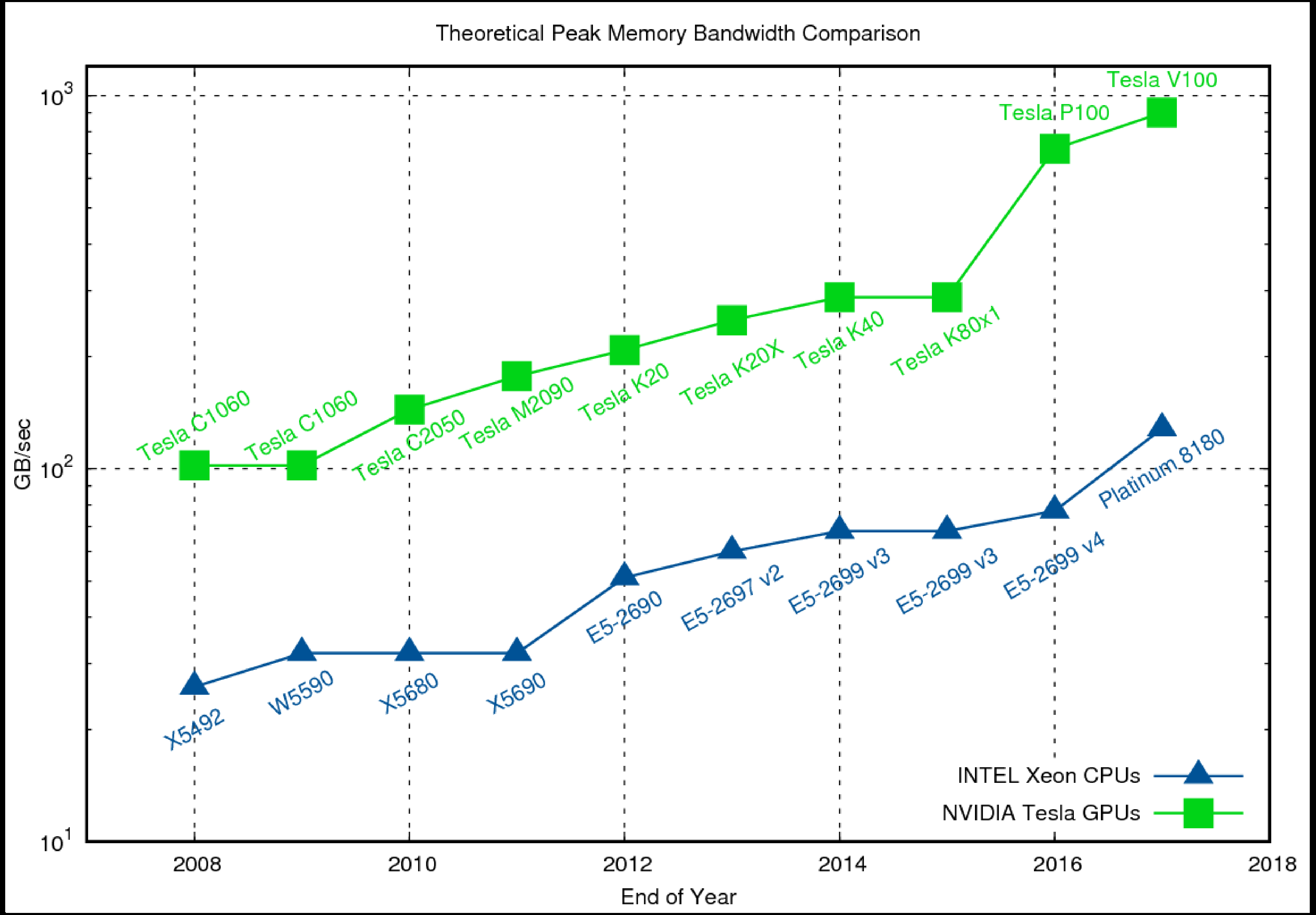
4xGPU

8xGPU

16xGPU

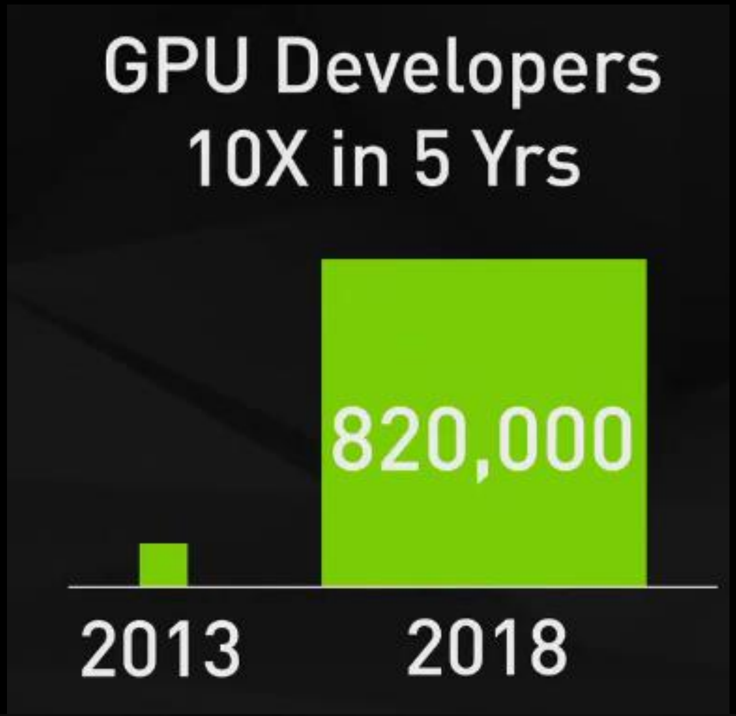
3) Saves Energy

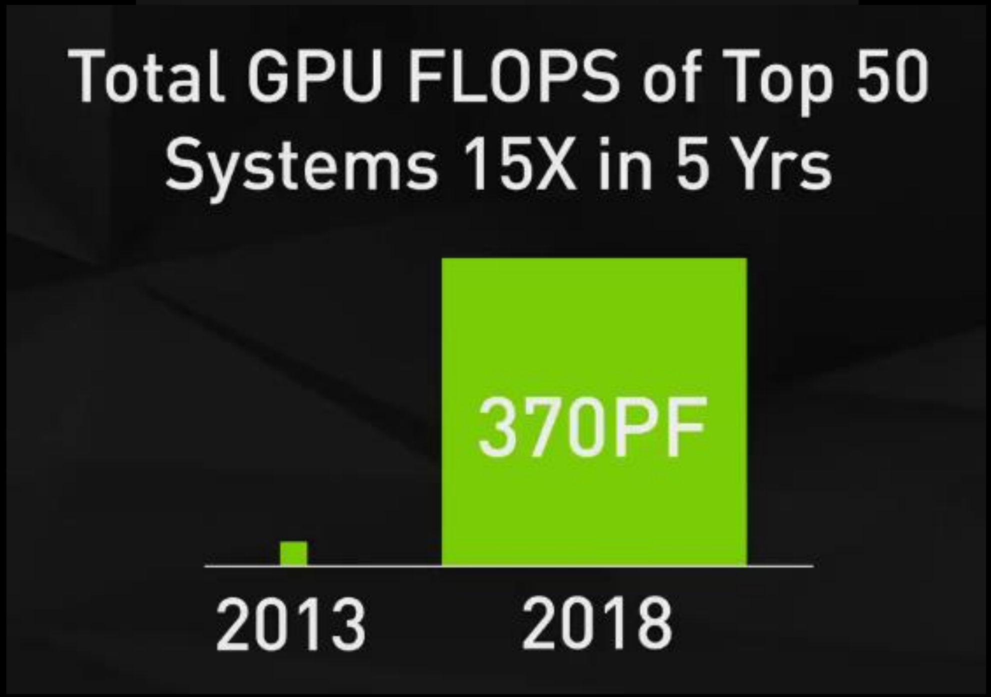
4) Saves Money

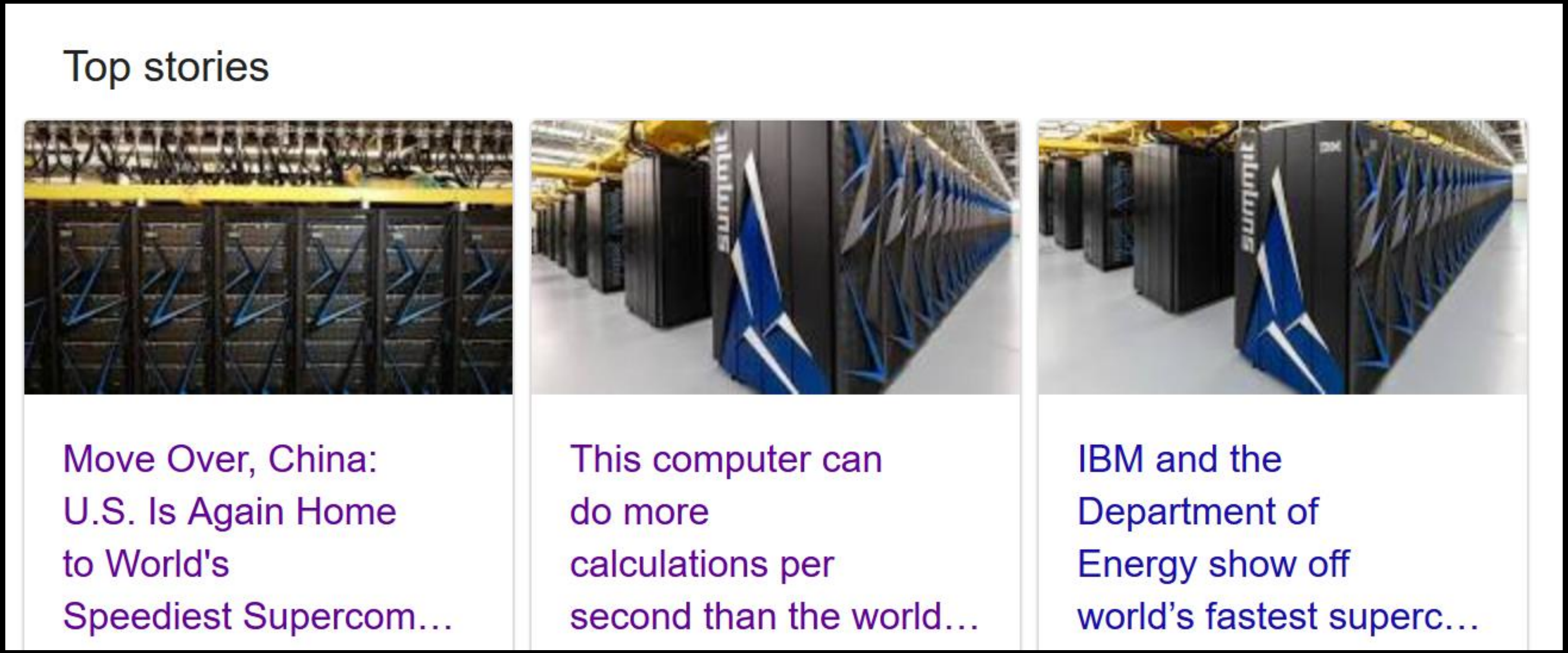


Accelerated Computing

Who uses accelerators?


GPU Developers 10X in 5 Yrs


Total GPU FLOPS of Top 50 Systems 15X in 5 Yrs


Top stories

Move Over, China: U.S. Is Again Home to World's Speediest Supercom…

This computer can do more calculations per second than the world…

IBM and the Department of Energy show off world's fastest superc…

"… consists of 4,608 compute servers, each containing two 22-core IBM Power9 processors and six NVIDIA Tesla V100 GPU accelerators …"

ASTRONUM 2018

• Tues.	9:40 AM	M. Zingale	
• Tues.	3:30 PM	M. Zhang	
• Wed.	8:25 AM	N. Pogorelov	- MS-FLUKSS
• Thurs.	1:55 PM	P. Woodward	- PPMStar

Accelerated Computing

Why *not* use accelerators?

- Not all algorithms suitable
- Hard to program
  Originally, only option was language extension APIs
  - CUDA (NVIDIA-specific)
  - OpenCL (more general)
  - This involves rewriting large sections of code and maintaining at least two code bases.

```
for (i=0; i<N; i++)
   y[i] = a*x[i] + y[i];
```

- Portability and longevity risk
  What if GPUs go away?

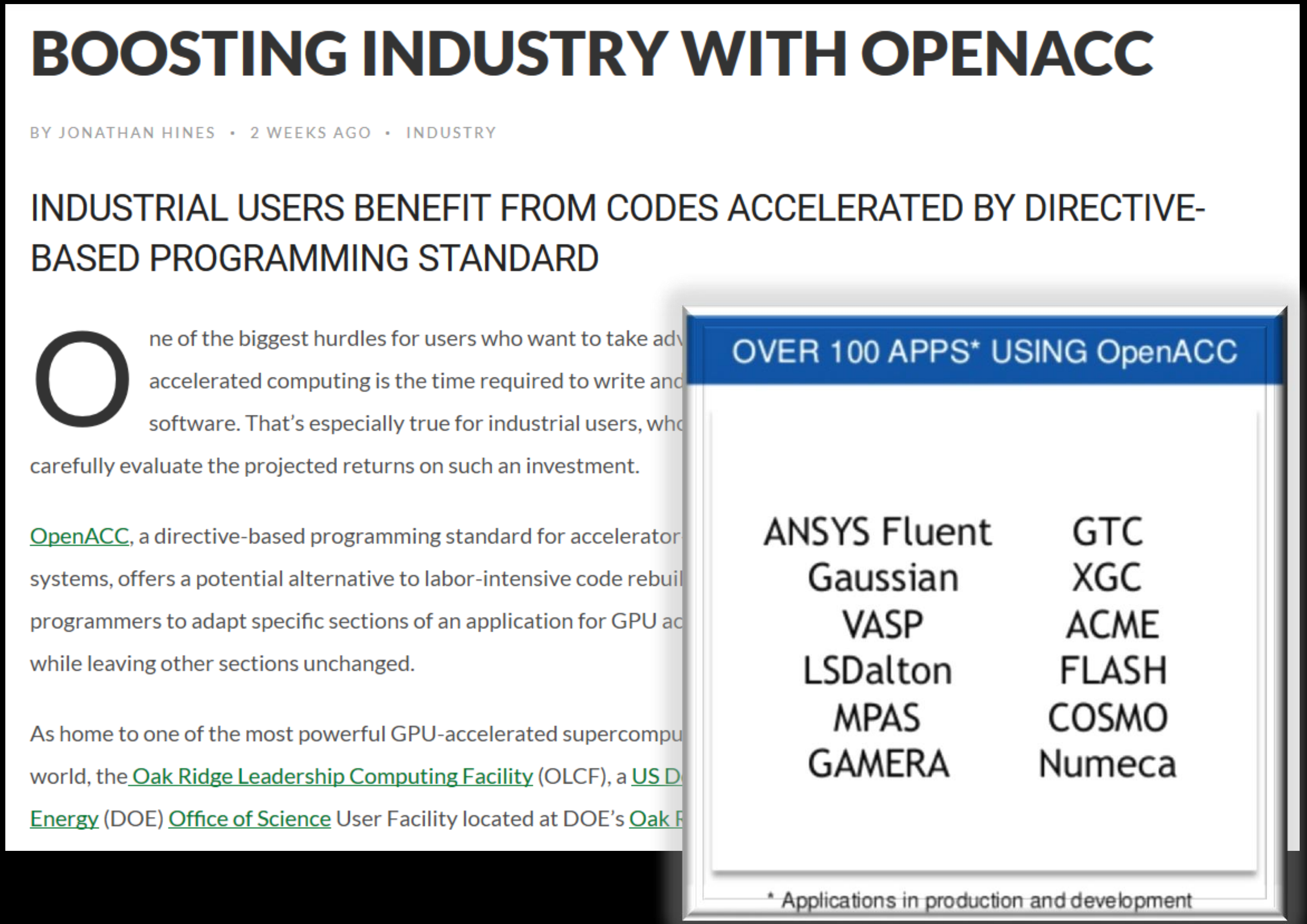
```
__global__ void saxpy(int N, float a,
                      float * restrict x,
                      float * restrict y){
  int i = blockIdx.x*blockDim.x + threadIdx.x;
  if (i < N) y[i] = a*x[i] + y[i];
}
...
const int BLOCK_SIZE=2048;
float *d_x,*d_y;

dim3 dimBlock(BLOCK_SIZE);
dim3 dimGrid((int)ceil((N+0.0)/dimBlock.x));

cudaMalloc( (void **) &d_x, sizeof(float)*N);
cudaMalloc( (void **) &d_y, sizeof(float)*N);
cudaMemcpy(d_x, x, N, cudaMemcpyHostToDevice);
cudaMemcpy(d_y, y, N, cudaMemcpyHostToDevice);

saxpy<<<dimGrid,dimBlock>>>(N, a, d_x, d_y);

cudaMemcpy(y, d_y, N, cudaMemcpyDeviceToHost);
cudaFree(d_x);
cudaFree(d_y);
```

OpenACC

More Science, Less Programming

```
C:          #pragma acc
FORTRAN:  !$acc
```

- Directive-based API, began as off-shoot of OpenMP
  - Uniform source code (no branches!)
  - Low-risk (can compile to CPU as before)
- Vendor-independent (**PGI**, CRAY, GNU, OMNI, SunWay)
- Multiple Target Architectures (**GPU**, Multicore x86, FPGA, SunWay)
- Designed for rapid development, especially for pre-existing codes
- Used by >90% of GPU Industry codes run on Titan at ORNL



BOOSTING INDUSTRY WITH OPENACC

BY JONATHAN HINES • 2 WEEKS AGO • INDUSTRY

INDUSTRIAL USERS BENEFIT FROM CODES ACCELERATED BY DIRECTIVE-BASED PROGRAMMING STANDARD

One of the biggest hurdles for users who want to take adv accelerated computing is the time required to write and software. That's especially true for industrial users, who carefully evaluate the projected returns on such an investment.

OpenACC, a directive-based programming standard for accelerator systems, offers a potential alternative to labor-intensive code rebuil programmers to adapt specific sections of an application for GPU ac while leaving other sections unchanged.

As home to one of the most powerful GPU-accelerated supercompu world, the Oak Ridge Leadership Computing Facility (OLCF), a US D Energy (DOE) Office of Science User Facility located at DOE's Oak R

OVER 100 APPS* USING OpenACC

ANSYS Fluent
Gaussian
VASP
LSDalton
MPAS
GAMERA

GTC
XGC
ACME
FLASH
COSMO
Numeca

* Applications in production and development

OpenACC

Example:
Accelerating SAXPY

```
for (i=0; i<N; i++)
    y[i] = a*x[i] + y[i];
```

NVIDIA CUDA

```
__global__ void saxpy(int N, float a,
                      float * restrict x,
                      float * restrict y){
  int i = blockIdx.x*blockDim.x + threadIdx.x;
  if (i < N) y[i] = a*x[i] + y[i];
}
...
const int BLOCK_SIZE=2048;
float *d_x,*d_y;
dim3 dimBlock(BLOCK_SIZE);
dim3 dimGrid((int)ceil((N+0.0)/dimBlock.x));
...
cudaMalloc( (void **) &d_x, sizeof(float)*N);
cudaMalloc( (void **) &d_y, sizeof(float)*N);
cudaMemcpy(d_x, x, N, cudaMemcpyHostToDevice);
cudaMemcpy(d_y, y, N, cudaMemcpyHostToDevice);

saxpy<<<dimGrid,dimBlock>>>(N, 2.0, d_x, d_y);

cudaMemcpy(y, d_y, N, cudaMemcpyDeviceToHost);
cudaFree(d_x);
cudaFree(d_y);
```

OpenACC

```
#pragma acc enter data copyin(x,y)
#pragma acc parallel present(x,y)
{
#pragma acc loop gang vector(32)
for (i=0; i<N; i++)
   y[i] = a*x[i] + y[i];
}
#pragma acc update_self(y)
#pragma acc exit data delete(x,y)
```

```
#pragma acc kernels
for (i=0; i<N; i++)
    y[i] = a*x[i] + y[i];
```

MAS

MAGNETOHYDRODYNAMIC
ALGORITHM
OUTSIDE A SPHERE

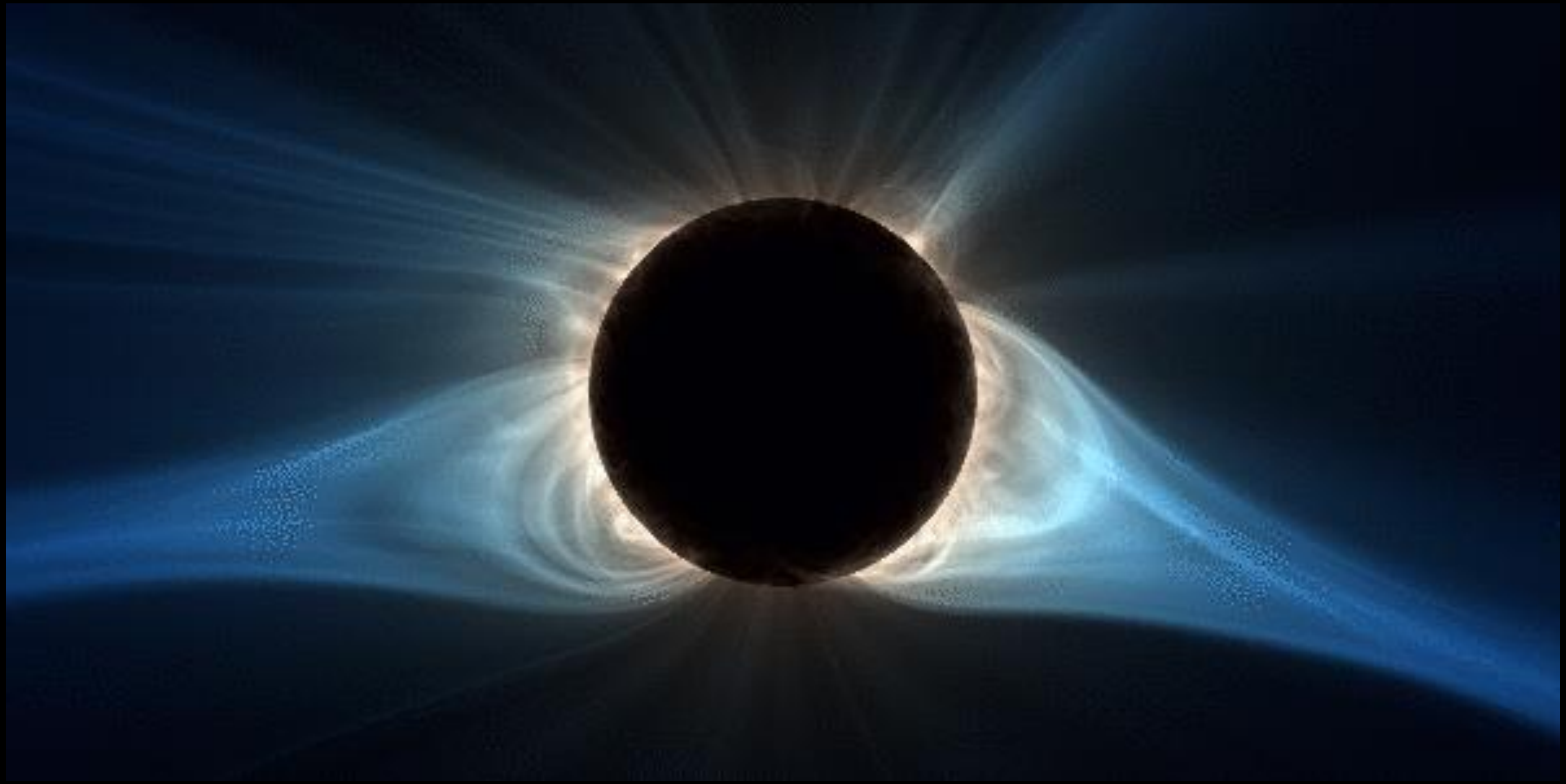
Predictive Science Inc.

- 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)

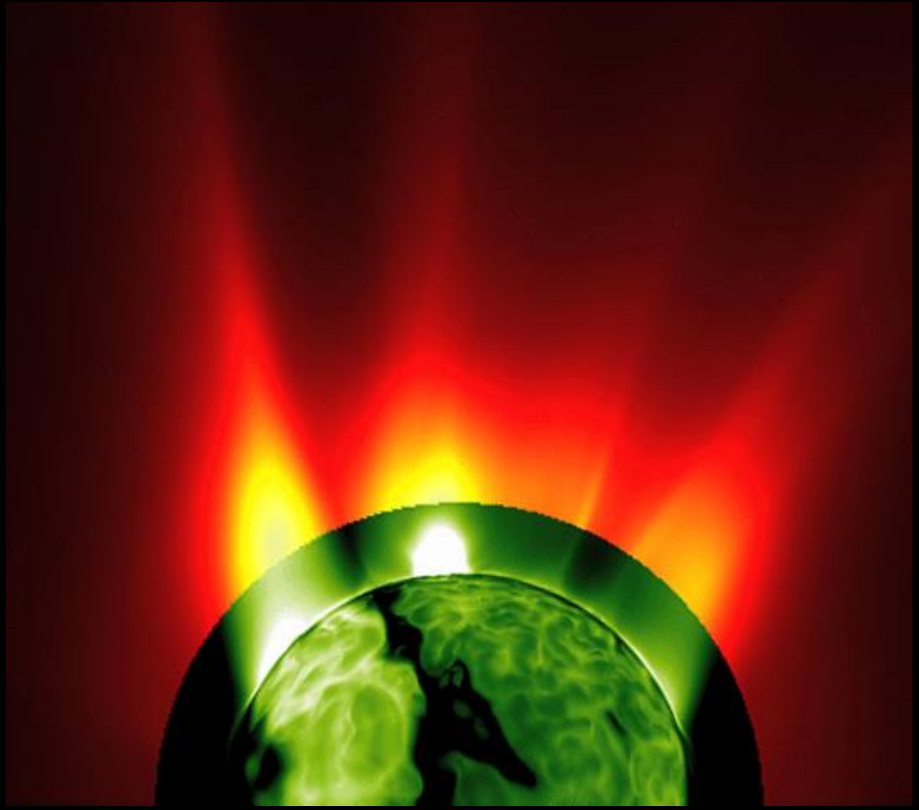
FORTRAN

MPI

Community
Coordinated
Modeling
Center



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



Simulation of the Feb. 13th, 2009 CME

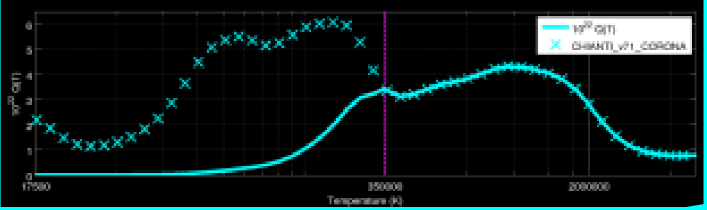
MAS: `Full' MHD Model Equations

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

RESISTIVITY

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

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



RADIATIVE COOLING

CORONAL HEATING

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

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

THERMAL CONDUCTION

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

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

ALFVÉN WAVES

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

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

VISCOSITY

SEMI-IMPLICIT OPERATOR

WAVE TURBULENCE

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

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

$\mathbf{B} = \nabla \times \mathbf{A}$

$\mathbf{J} = \frac{c}{4\pi}\nabla \times \mathbf{B}$

$p = 2\,k\,T\,\rho/m_p$

$\mathbf{g} = -g_0\,R_\odot^2\,\hat{\mathbf{r}}/r^2$

$\gamma = 5/3$

$\hat{\mathbf{b}} = \mathbf{B}/\,|\mathbf{B}|$

$\mathbf{v}_\mathbf{A} = \mathbf{B}/\sqrt{4\,\pi\,\rho}$

$B_w = 6.09\,G$

$v_c^2 = \gamma\,p/\rho$

$$\beta_{\text{Tcut}}(T) = \begin{cases} (T/T_{\text{cut}})^{-5/2} & T < T_{\text{cut}} \\ 1, & T \geq T_{\text{cut}} \end{cases}$$

$T_{\text{cut}} = 3.5 \times 10^5\,K$

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

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

$C_f = \Delta t\,\tilde{k} \cdot \mathbf{v}$

$C_w^2 = 0.25\,\Delta t^2\,\tilde{k}^2\,(v_c^2 + |\mathbf{v}_\mathbf{A}|^2)$

$\tilde{k}^2 = 4\,\left(\Delta r^{-2} + (r\,\Delta\theta)^{-2} + (r\,\Delta\phi\,\sin\theta)^{-2}\right)$

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)
- This allows a simplified “zero-beta” model to be useful in many cases (e.g. modeling the initial configuration and onset dynamics of a CME eruption)

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

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

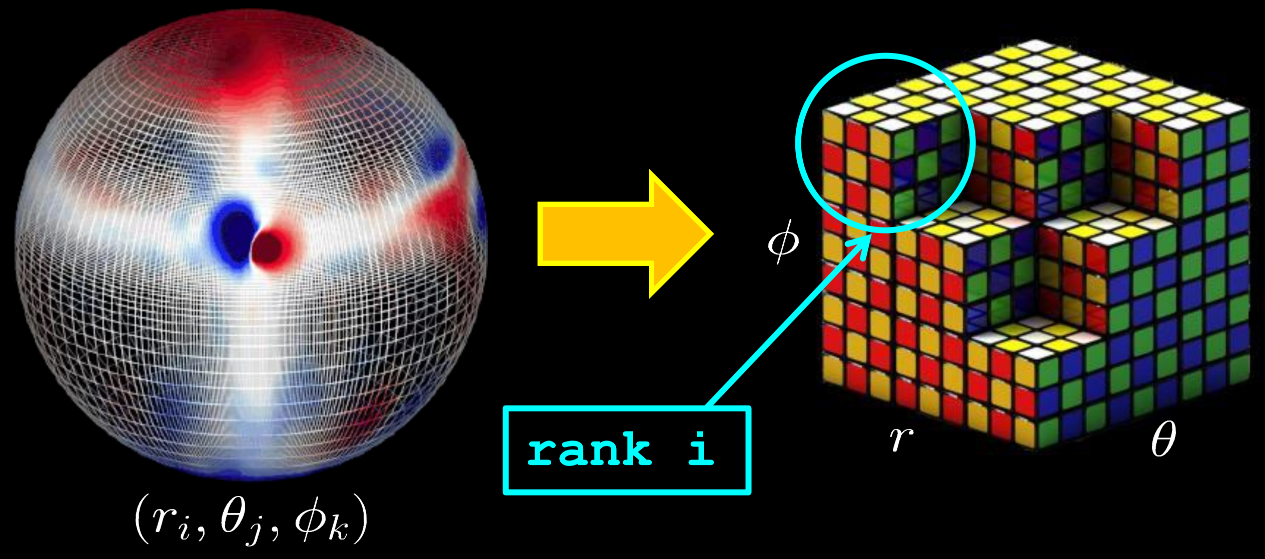
$$\mathbf{B} = \nabla \times \mathbf{A} \qquad \nabla \cdot \mathbf{B} = 0 \qquad v_A^2 = |\mathbf{B}|^2/(4\,\pi\,\rho) \qquad S = (\Delta t^2\,\tilde{k}^2)^{-1}\left(C_w^2/(1-C_f)^2 - 1\right) \qquad C_w^2 = 0.25\,\Delta t^2\,\tilde{k}^2\,(v_c^2 + v_A^2)$$

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

- Since the core algorithms are the same as the full model, this makes an ideal target for our initial OpenACC implementation (stepping stone)

MAS: Algorithm Summary

- Finite difference on non-uniform spherical grid
- Parallelized with MPI
- Explicit and implicit time-stepping algorithms
- Implicit time-step (backward-Euler) solved with Preconditioned Conjugate Gradient
- Two communication-free preconditioners: **PC1** and **PC2**
- For ‘hard’ solves, **PC2** faster than **PC1** for ‘easy’ solves, **PC1** faster than **PC2**



PCG

- **Resistivity**
- **Semi-Implicit Pred**
- **Semi-Implicit Corr**
- **Viscosity**

PC1

Point-Jacobi

PC2

Block-Jacobi with ILU0

Production Test Run

Zero-Beta Unstable Flux Rope Eruption

Run information

Physical code time duration: **198 seconds**

Number of time-steps: **695**

$$160 \times 267 \times 246 \sim 10.5\,\text{million points}$$

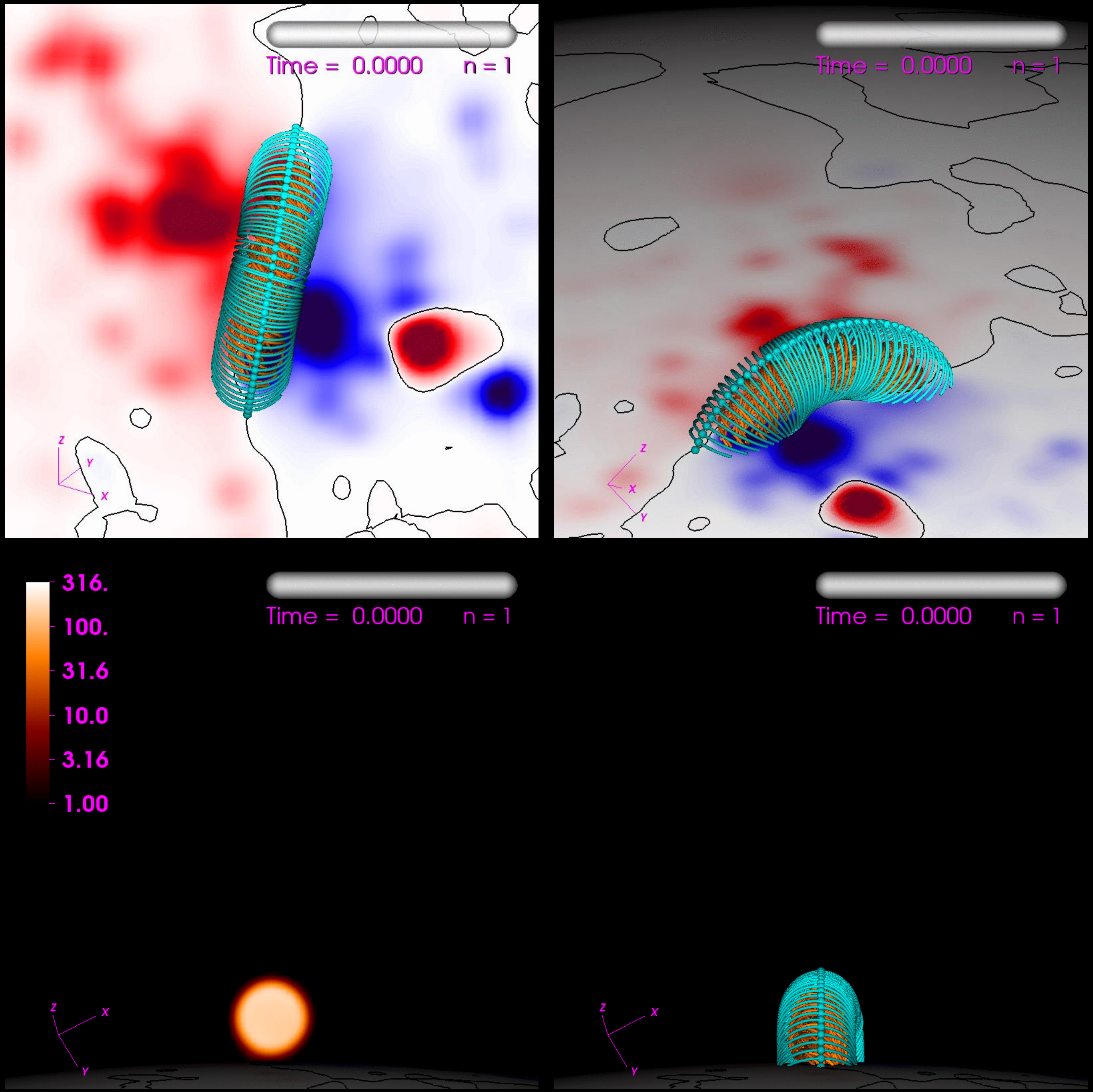
Spherical Domain with $r_{\max} = 10\ R_{\odot}$

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	0.066°	9.45°	11%
ϕ	246	0.067°	14.61°	10%
t	695	0.001 sec	0.17 sec	11%

PCG Solver Iterations per Time Step (mean)

	SI Predictor	SI Corrector	Viscosity
PC1	72	75	554
PC2	$30 \to 34$	$31 \to 35$	$104 \to 197$

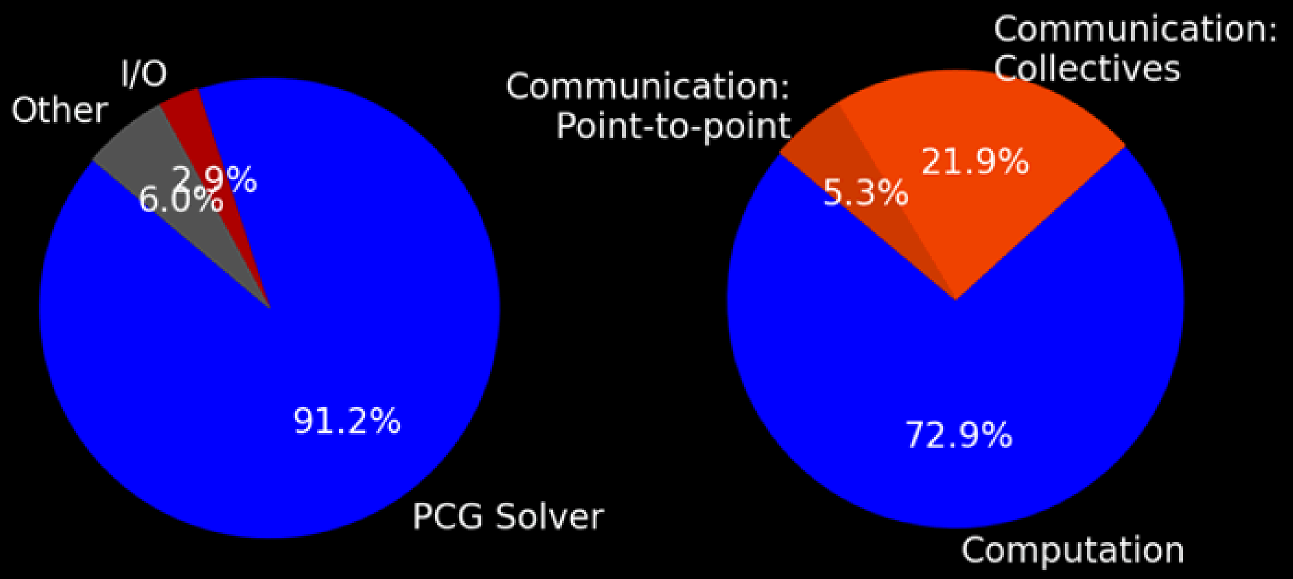


OpenACC Implementation: Preliminaries

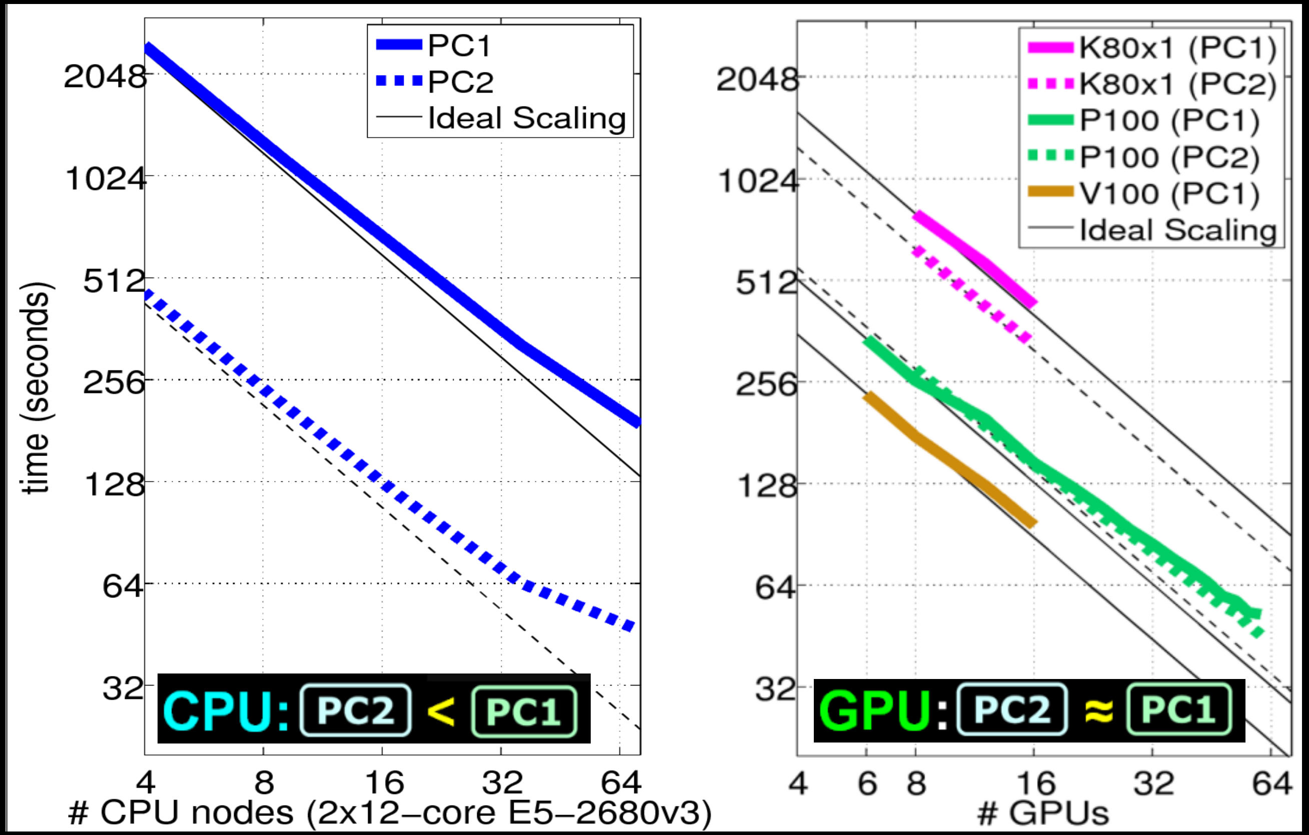
- Profile code
  - PCG over 90% of run-time
  - Viscosity is hardest solve
- Analyze algorithms for GPU-compatibility
  - Most PCG steps and explicit time-stepping “vector-friendly”
  - Preconditioners

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

- Test performance through “Proof-of-concepts”
  - DIFFUSE: Explicit finite-difference
  - POT3D: PCG+PC1/PC2
- Based on results of POT3D, we only accelerate PC1 in MAS



Run using 16 nodes of 24-core Haswell CPUs (PC2)



OpenACC Implementation: Examples

CPU↔GPU Data transfers

```
    allocate and initialize “y” …
!$acc enter data copyin (y)
    use “y” in OpenACC compute regions …
!$acc update self (y)
    CPU version of “y” updated for I/O, etc. …
!$acc exit data delete(y)
```

Basic Loop

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

Reductions

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

FORTRAN Array-syntax

```
!$acc kernels default(present)
    y(:) = a*x(:) + y(:)
!$acc end kernels
```

OpenACC Implementation: Multi-GPU

Multiple GPUs with MPI

MPI-2

(assumes linear affinity)

```
call MPI_Comm_rank (MPI_COMM_WORLD, iprocw, ierr)
ngpus_per_node = 4
igpu = MODULO(iprocw, ngpus_per_node)
!$acc set device_num(igpu)
```

MPI-3

(code shown assumes #GPUs/node = #ranks/node)

```
call MPI_Comm_split_type (MPI_COMM_WORLD,MPI_COMM_TYPE_SHARED,
&                              0,MPI_INFO_NULL,comm_shared,ierr)
call MPI_Comm_size (comm_shared, nprocsh, ierr)
call MPI_Comm_rank (comm_shared, iprocsh, ierr)
igpu = MODULO(iprocsh, nprocsh)
!$acc set device_num(igpu)
```

Use GPU data directly with MPI calls (“CUDA-aware MPI”)

```
!$acc host_data use_device(y) if_present
  call MPI_Allreduce (MPI_IN_PLACE,y,n,MPI_DOUBLE,MPI_SUM,MPI_COMM_WORLD,ierr)
!$acc end host_data
```

OpenACC Implementation: Effort Summary

<2%

OpenACC comment lines added

<5%

Total modified lines of code

Details

Total lines in original code	52,600
Total lines in accelerated code	55,460
Total !$acc/!$acc& lines added	776 (1.5%)
Total modified lines	2451 (4.7%)

Factors to consider:

- Optional CPU code simplifications
- Some CPU changes are temporary compiler bug work-arounds, or waiting for future OpenACC features
- Full code not accelerated (zero-beta only!)

Single portable source for GPU and CPU!

OpenACC Implementation: Difficulties

Difficulties...

- Compiler Issues
  - Documentation lag
  - Implementation lag
  - Bugs

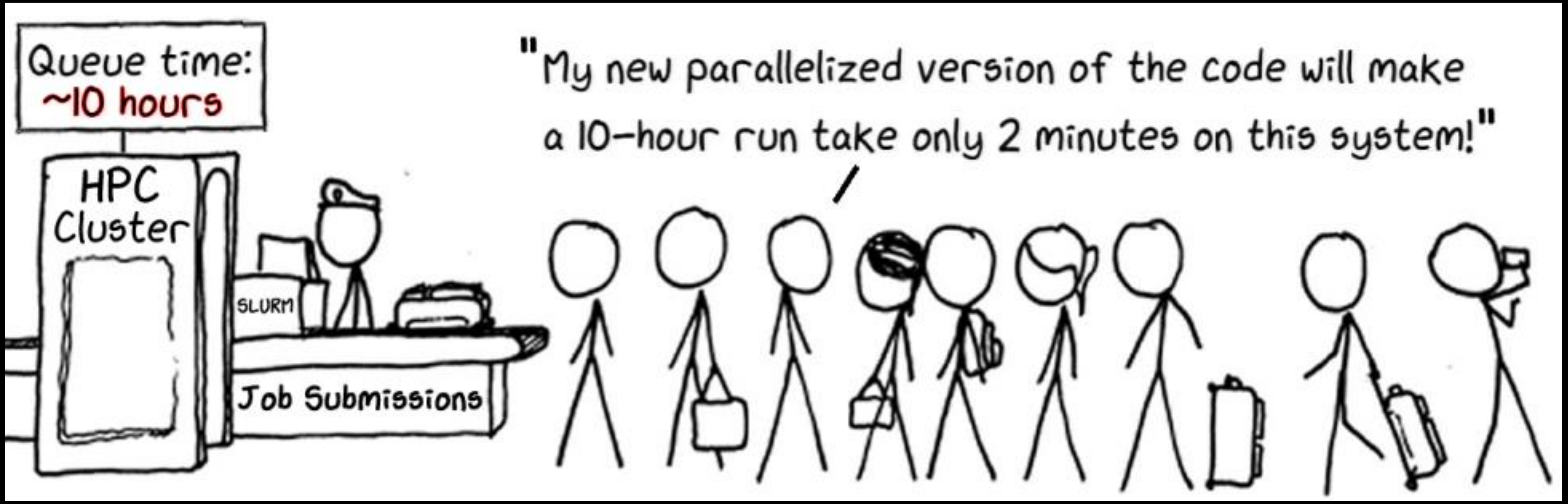
!$acc cache(a%y(i-1:i+1))

>I'm sorry, I'm afraid
I can't do that... yet

- System issues
  - Compiler licenses/updates
  - Library versions and setup
  - Hardware setups

Timing Procedures

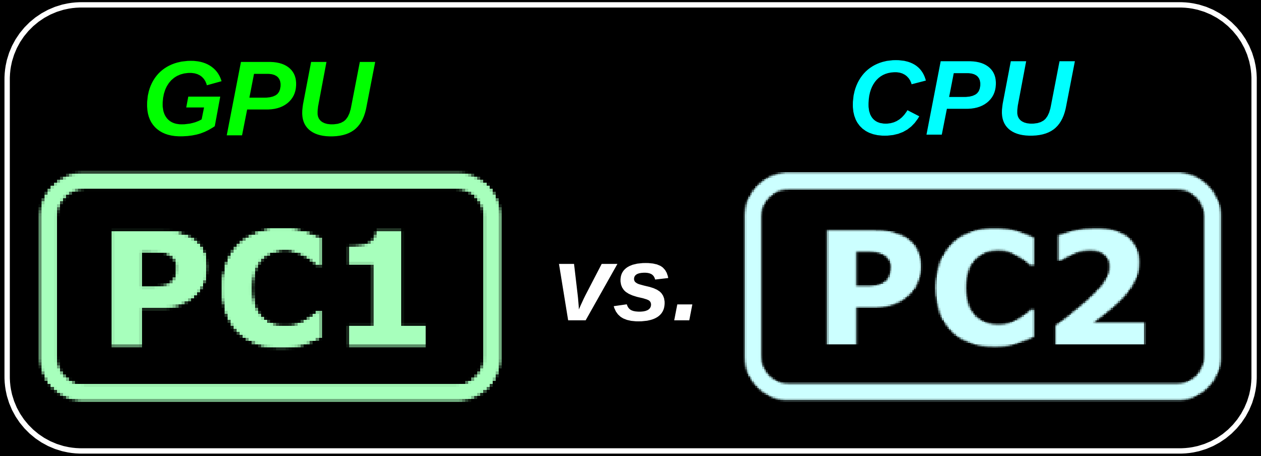
- “Time-to-solution” Includes I/O, comm, setup, etc. (Queue times excluded, but important!)
- We use best available compiler, compiler version, instruction sets, library versions, and ***algorithm*** for each hardware


Queue time:
~10 hours
"My new parallelized version of the code will make a 10-hour run take only 2 minutes on this system!"
HPC Cluster
SLURM
Job Submissions

Why is this fair?

We’re not benchmarking hardware

Want to test the maximum “effective” performance on each system for solving our problem, using our code


GPU

PC1

vs.

CPU

PC2

vs.

Hardware and Environments

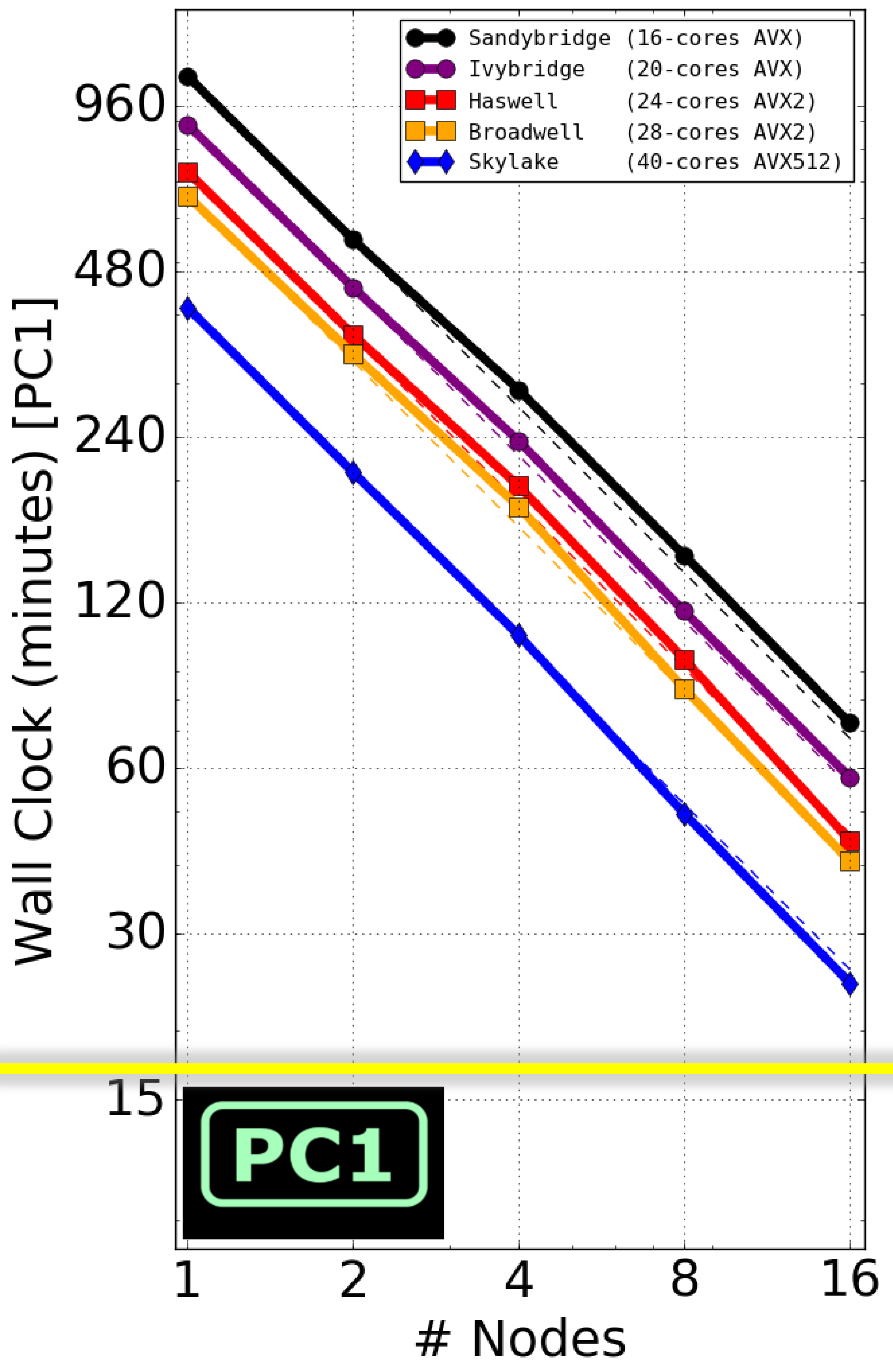
	NASA NAS Pleiades & Electra					Local Workstation	Local Desktop
Compiler	Intel 2018 .0.128					GNU 5.4.0	
MPI Library	SGI MPT 2.15r20					OpenMPI 1.10.2	
Family	**Sandy Bridge**	**Ivy Bridge**	**Haswell**	**Broadwell**	**Skylake**	**Haswell**	**Broadwell**
Instruction Set	AVX		AVX2		AVX512	AVX2	
Model	E5-2670	E5-2680v2	E5-2680v3	E5-2680v4	Gold 6148	E5-2680v3	E5-1650v4
Clock Rate	2.6 GHz	2.8 GHz	2.5 GHz	2.4 GHz	2.4 GHz	2.5 GHz	3.6 GHz
#Sockets x #Cores	**2**x8	**2**x10	**2**x12	**2**x14	**2**x20	**2**x12	**1**x6
Total Mem Bandwidth	**51.2 GB/s**	**59.7 GB/s**	**68 GB/s**	**76.8 GB/s**	**128 GB/s**	**68 GB/s**	**76.8 GB/s**

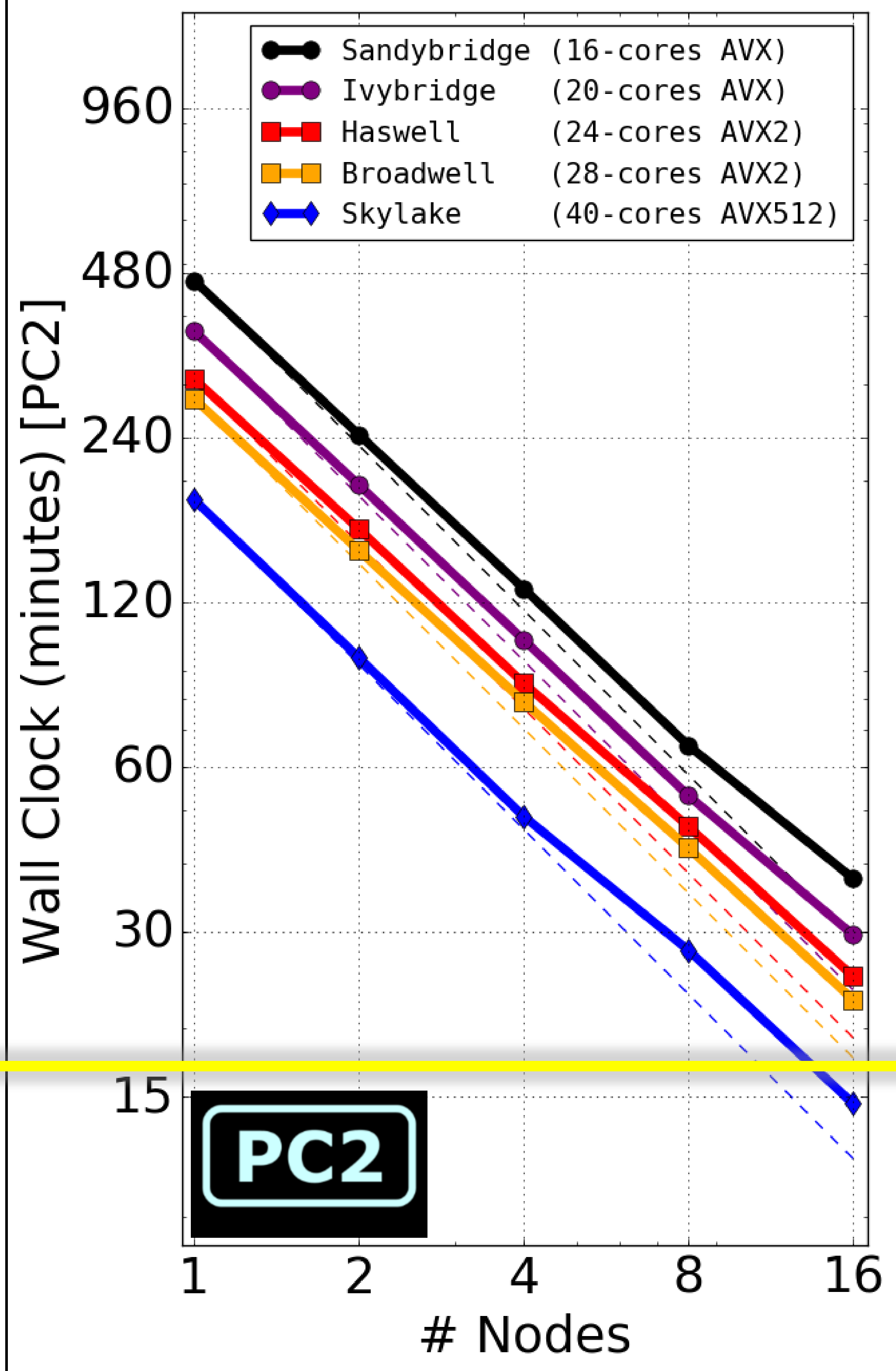
	NVIDIA PSG	SDSC Comet	Local Desktop
Compiler	PGI 18.3	PGI 18.4	
MPI Library	OpenMPI 1.10.7	OpenMPI 2.1.2	
CUDA Library	CUDA 9.1		
Driver Version	396.26	367.48	396.26
# GPUs x Model	**4x**V100	**4x**P100	**1x**TitanXP
Clock Rate	1.38 GHz	1.33 GHz	1.58 GHz
# CUDA DP Cores/GPU	2560	1792	120
Mem Bandwidth/GPU	**900 GB/s**	**732 GB/s**	**547.6 GB/s**

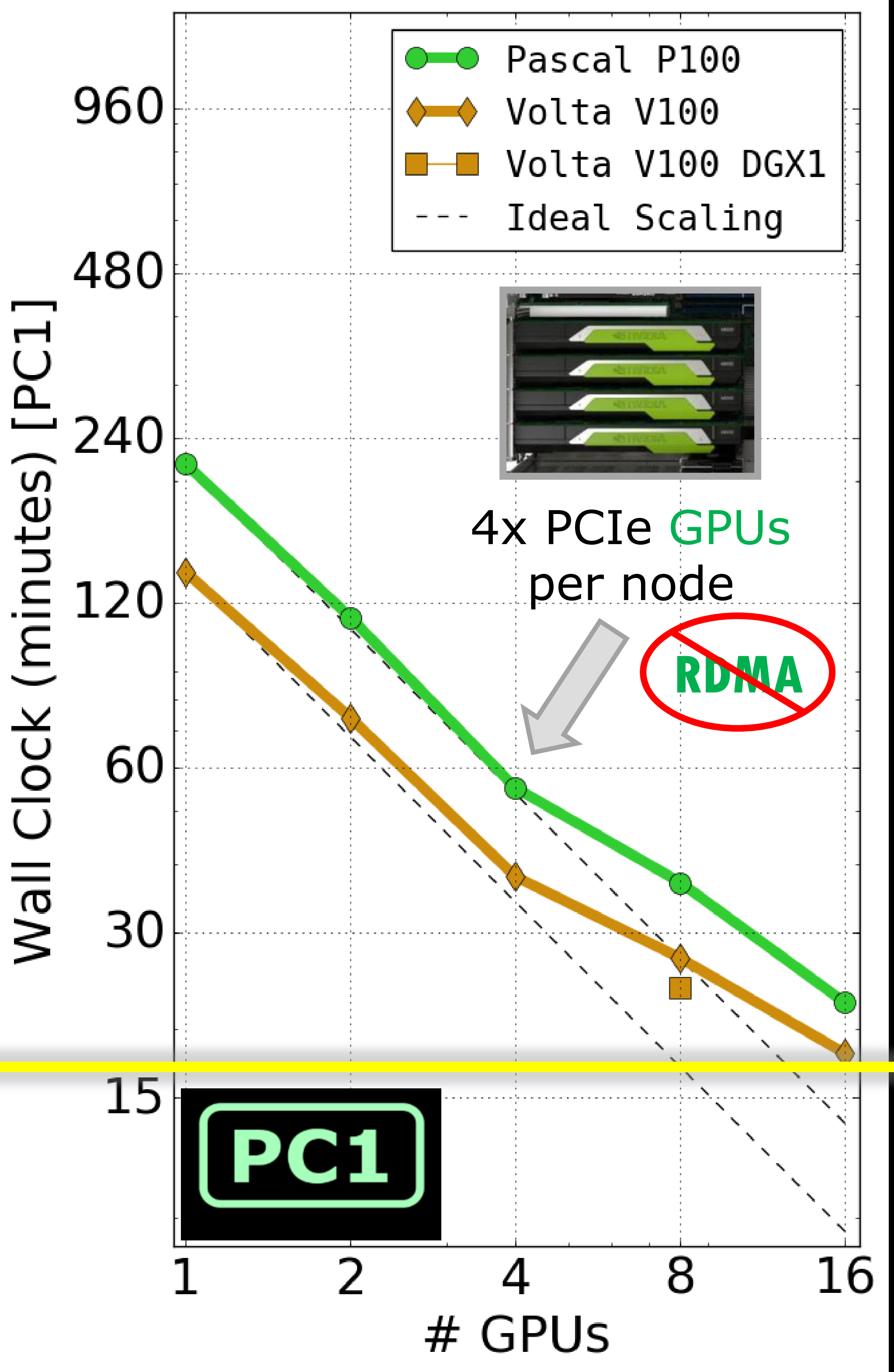
Compiler Flags:

```
Intel (CPU): -O3 -heap-arrays
             -fp-model precise
             -xCORE_AVX#

GNU (CPU): -O3 –mtune=native

PGI (GPU): -O3
           -ta=tesla:cuda9.1,cc##
```

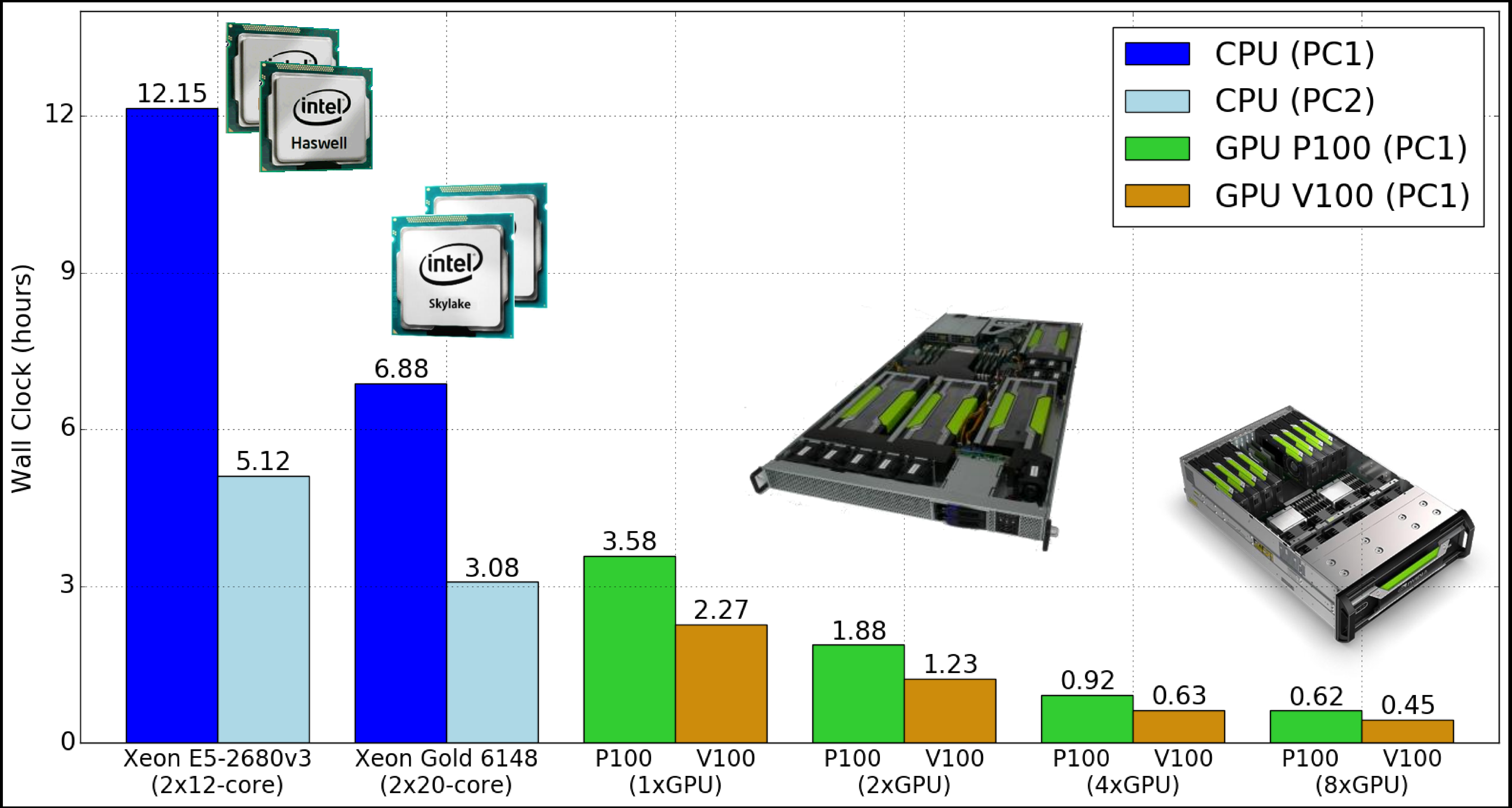
Timing Results



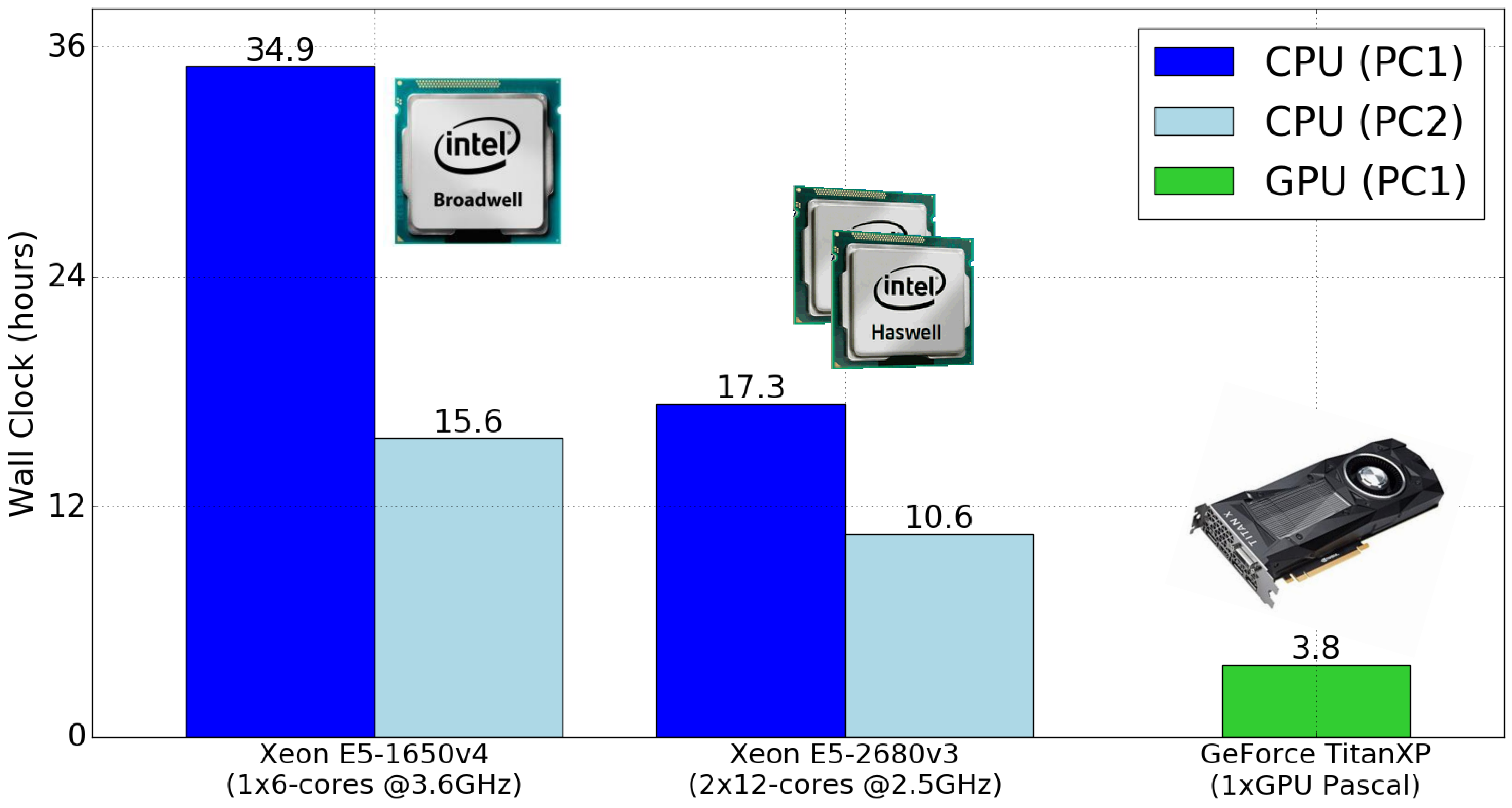




Timing Results “In-house” Single Server



Timing Results “In-house” Single Desktop



~$3000

~$7000

+$1200

~$9000
Wall Clock:
(est) ~1 hour

<1.5x Cost
>10x Speed

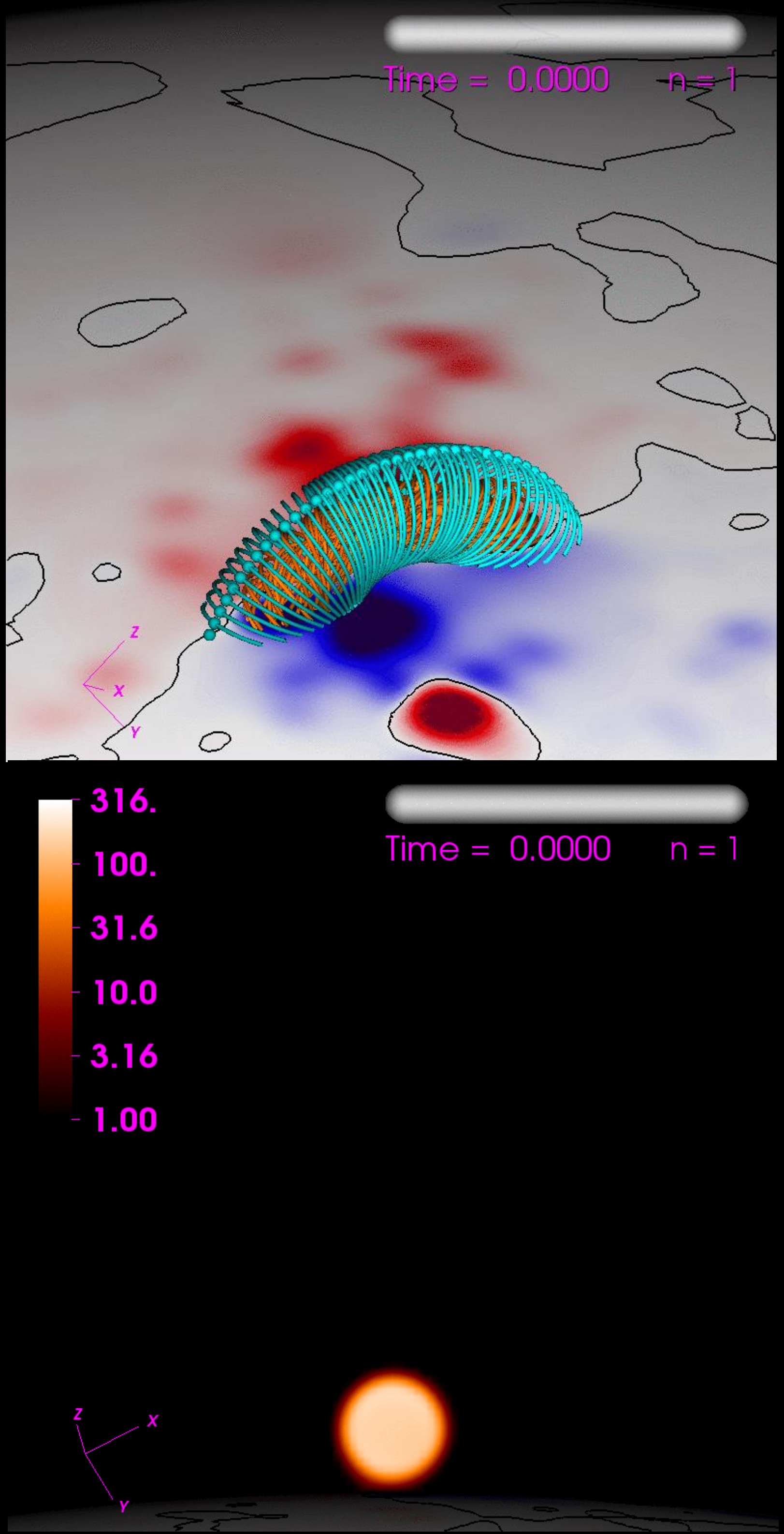
Alternative Algorithms: Super Time-Stepping

- Want vectorizable PC as good as **PC2** in reducing iterations
- Geometric/algebraic multigrid attractive choice but requires massive code changes
- At ASTRONUM 2016 we tested RKL2 Super Time-Stepping (**STS**) (Meyers et al 2014) in MAS as an alternative to PCG for viscosity
- Performance of the **STS** method was great, but had accuracy issues
- Since the **STS** algorithm is highly vectorizable, its worth testing an OpenACC implementation for the current problem (where viscosity is most time-consuming)

STS

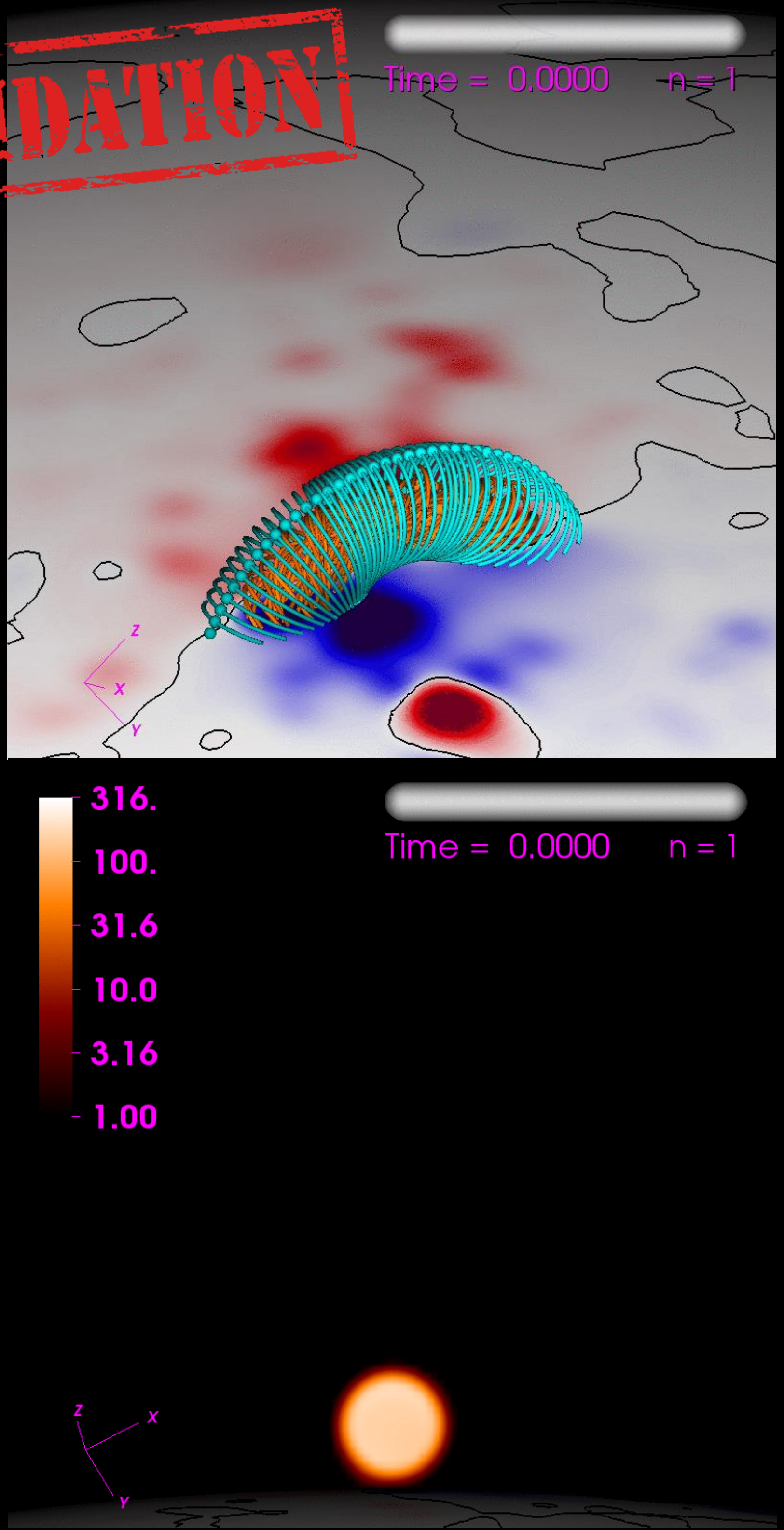
Alternative Algorithms: Super Time-Stepping

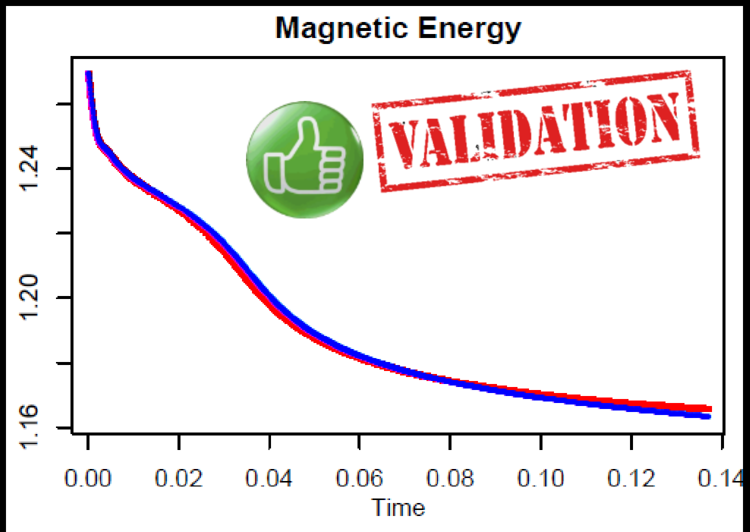
Viscosity: **PCG+PC2**

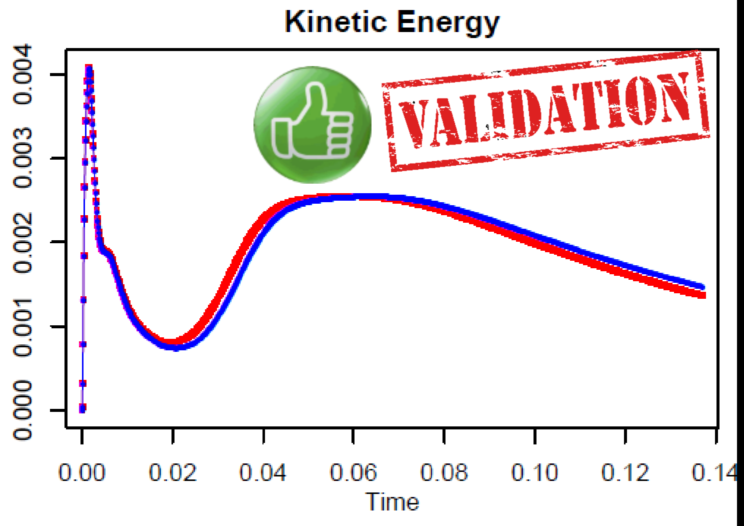


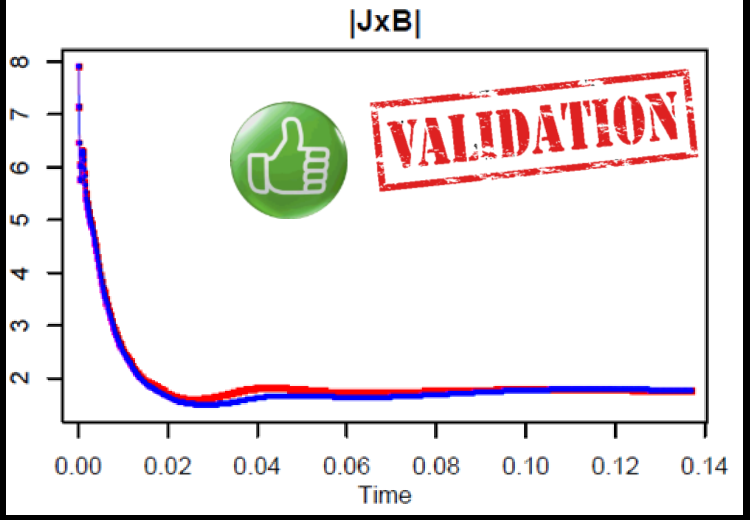
VALIDATION

Viscosity: **STS**

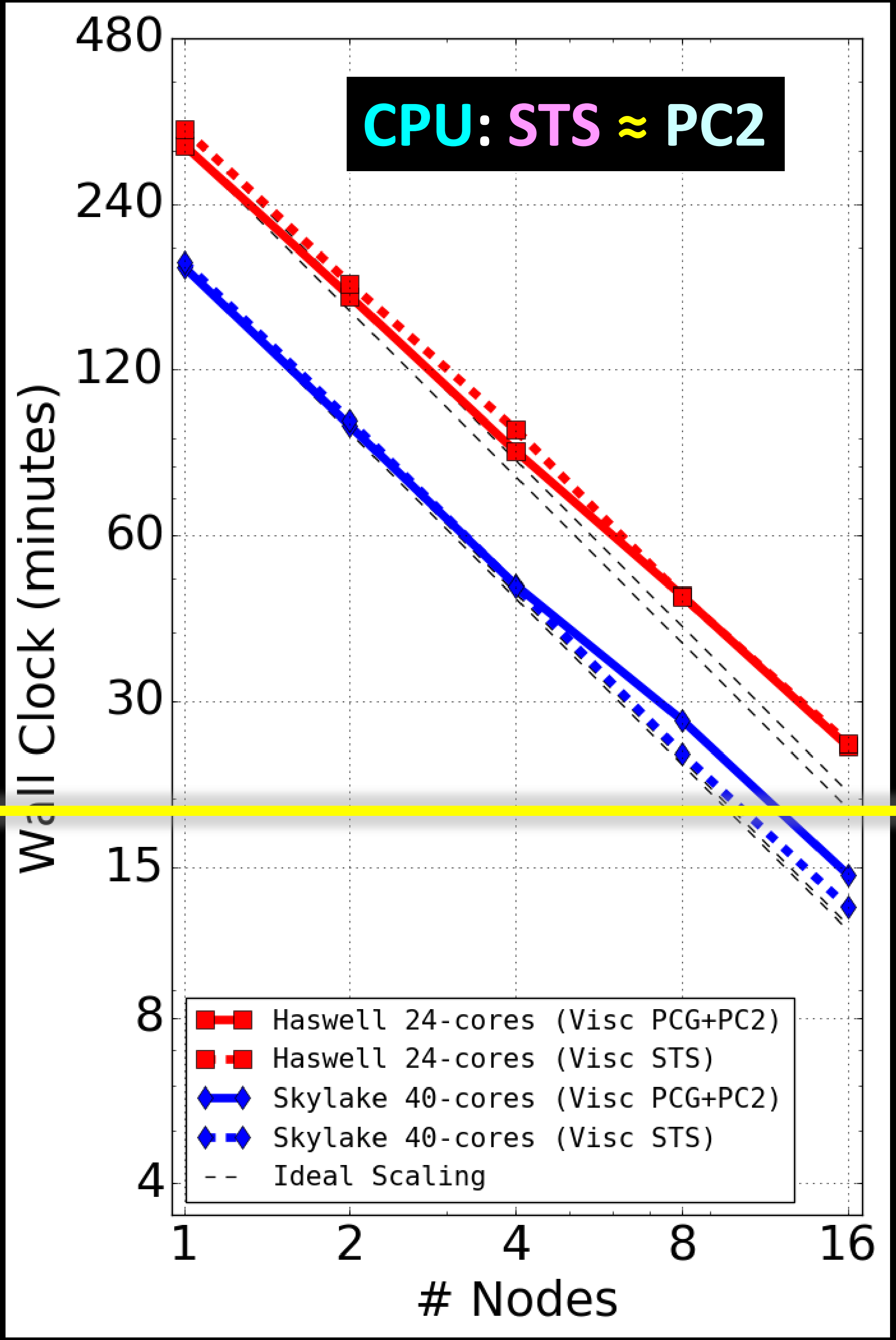


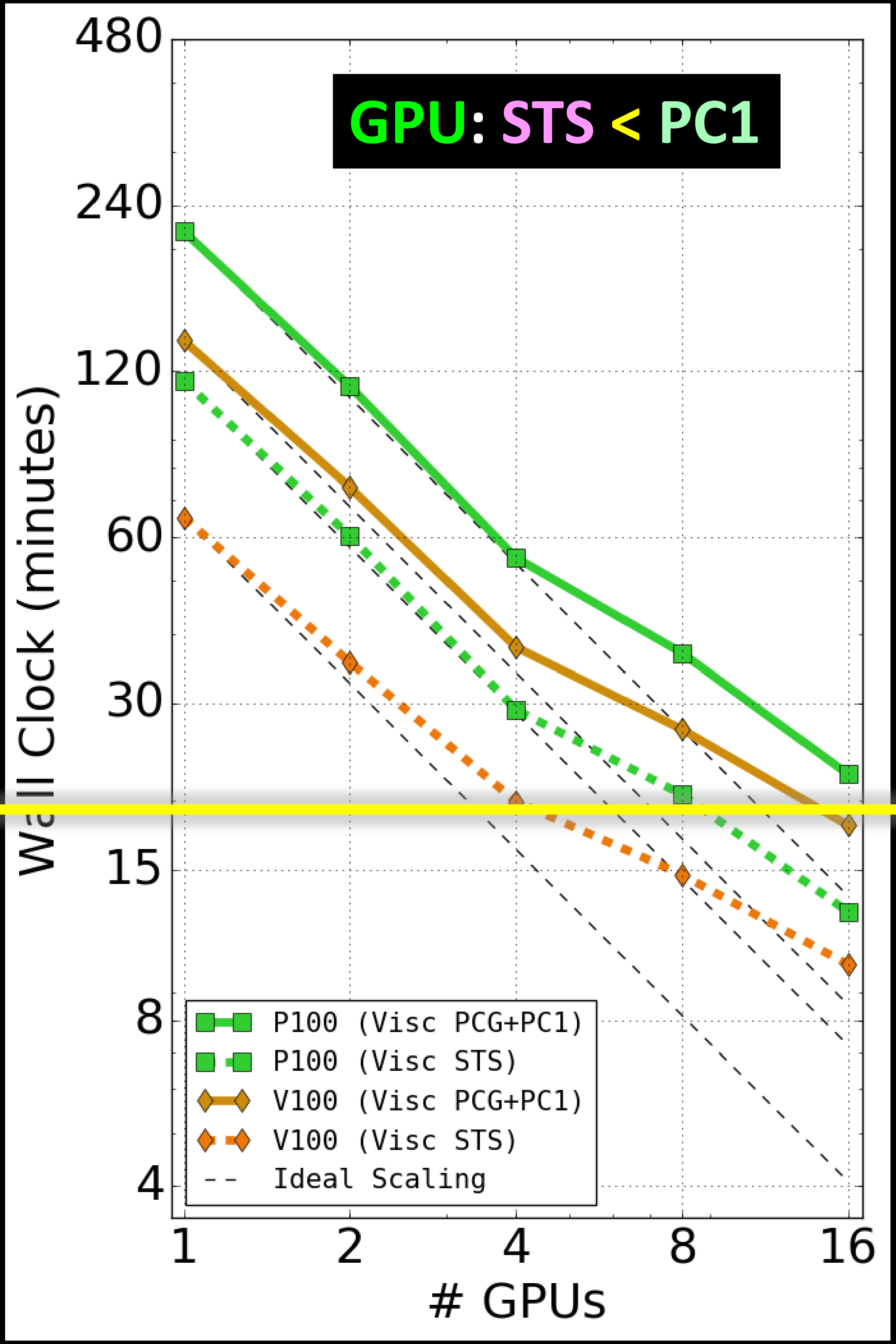






Alternative Algorithms: Super Time-Stepping





- **CPU**
  STS exhibits better scaling, but similar run times to **PC2**
- **GPU**
  STS ~ twice as fast as **PC1**, but similar scaling

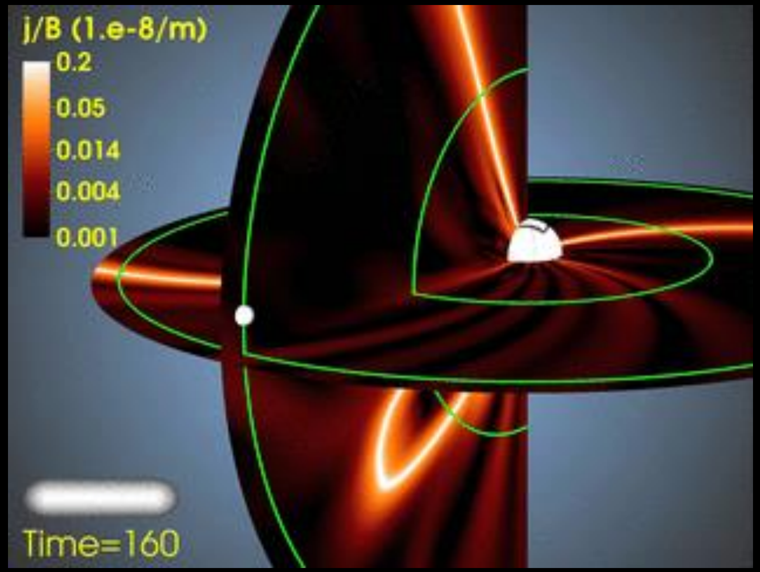
Performance Summary of Equivalent Wall Time

PC1 (2.3 hours)	**1x** V100	**=**	**1.6x** P100	**=**	**2.9x** 40 cores	**=**	**5.3x** 24 cores
CPU: PC2 **GPU: PC1** (2.3 hours)	**1x** V100	**=**	**1.6x** P100	**=**	**1.4x** 40 cores	**=**	**2.5x** 24 cores
STS$_{visc}$ (1.1 hours)	**1x** V100	**=**	**1.7x** P100	**=**	**2.9x** 40 cores	**=**	**5.7x** 24 cores

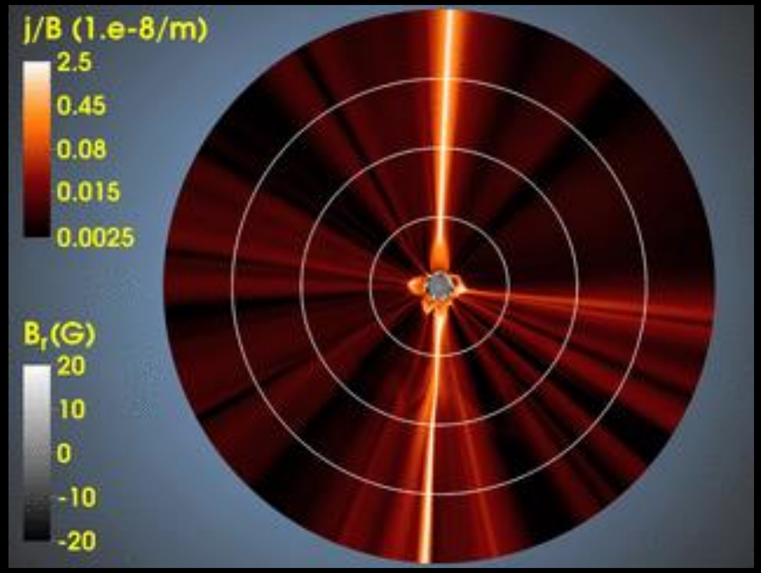
Can fit **4** of these in one desktop (~20 min)!

Summary and Outlook

- For this run (representative of many similar cases), we can move from HPC cluster to "in-house"
- Future improvements
  - Vectorizable Preconditioners
  - **PC2** with single-precision
  - Make **STS** method accuracy-robust
- Next steps
  - Heliospheric runs (**PC1** faster than **PC2** on CPU!)
  - Thermodynamic (coronal) runs (on GPU-cluster like Summit)



Heliospheric CME Simulation



Thermodynamic CME Simulation

Questions?

OpenACC User Group

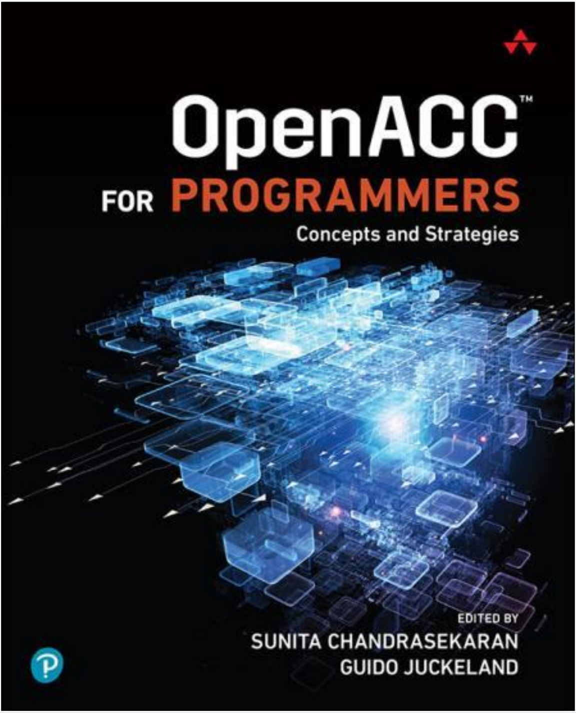
Directives for Accelerators —> More Science. Less programming

Twitter @OpenACCorg
Facebook @OpenACCorg
LinkedIn OpenACC Developers

PGI Community Edition 18.4

Latest CPUs
NVIDIA Volta
OpenACC 2.6

FREE DOWNLOAD



OpenACC™ FOR PROGRAMMERS

Concepts and Strategies

EDITED BY
SUNITA CHANDRASEKARAN
GUIDO JUCKELAND

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

Computer allocations provided by
- NASA NAS (Pleiades/Electra)
- NVIDIA Cooperation (PSG)
- XSEDE/SDSC (Comet)

NVIDIA.

NSF

NASA

XSEDE
Extreme Science and Engineering
Discovery Environment

Predictive Science Inc.

Contact: caplanr@predsci.com

Slides available at:
predsci.com/~caplanr