GPU-accelerating Established Science Applications with OpenACC: Experiences, Results, and Development Tips

Ron Caplan,
J. A. Linker, and Z. Mikić
Predictive Science Inc
www.predsci.com

Predictive Science Inc.

Slides available at:
predsci.com/~caplanr

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

Twitter @OpenACCorg
Facebook @OpenACCorg
LinkedIn OpenACC Developers

GPU Computing – Not just for AI!

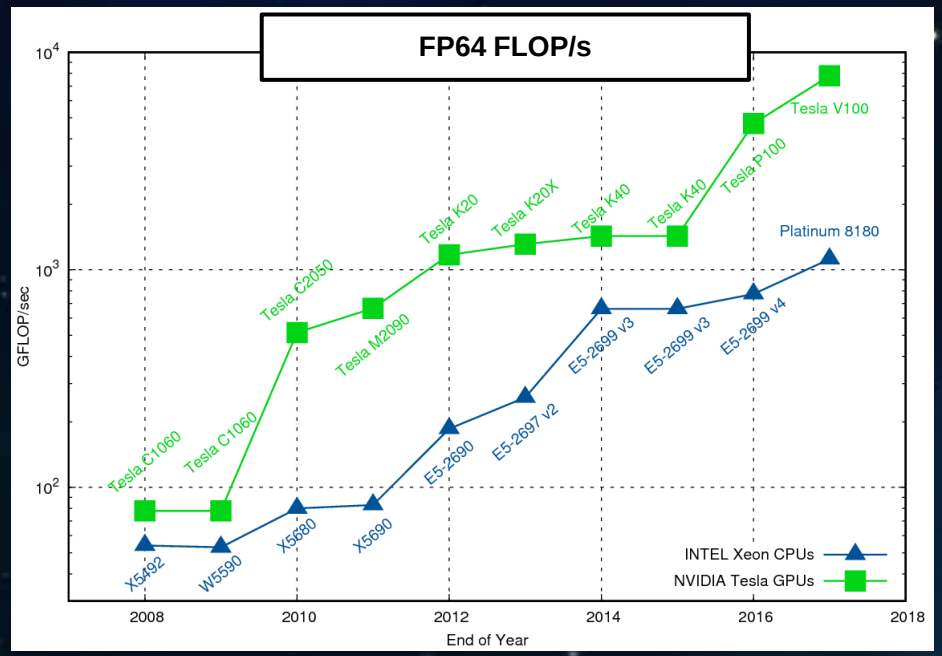
Why use GPUs?

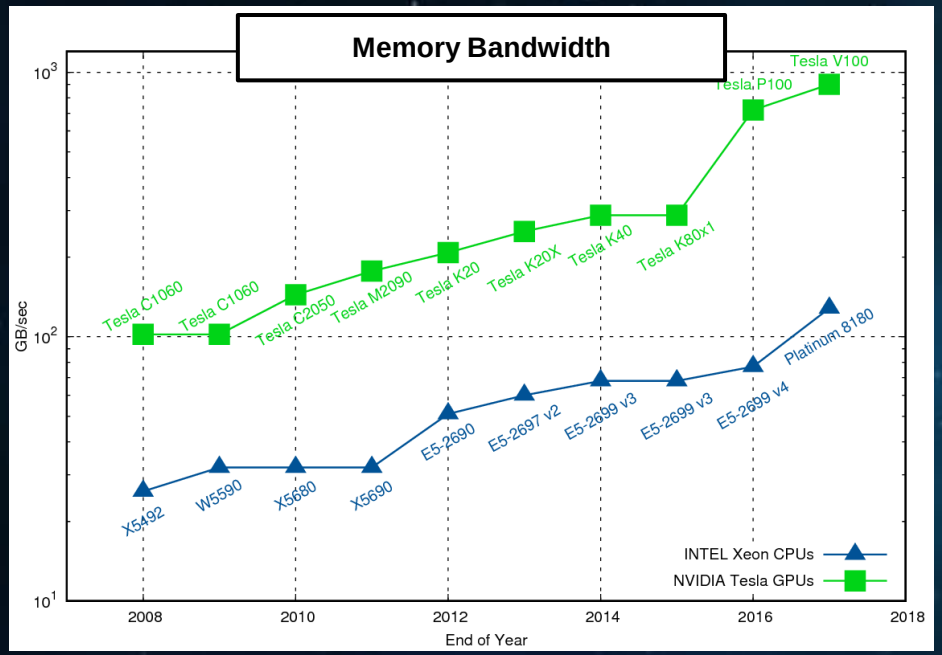
1) Performance
   (FLOP/s and Memory Bandwidth)
2) Compact Performance
3) Saves Energy
4) Saves Money

4xGPU

8xGPU

16xGPU





Why use OpenACC?

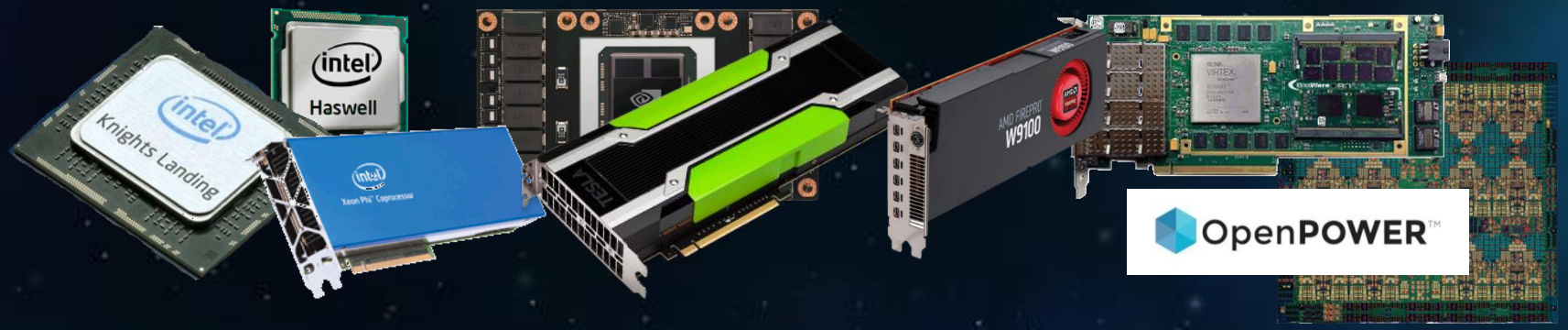
(1) Directive-based
(single source code, low risk)

(2) Fast development

(3) Maintains legacy code

(4) Can compile to multiple architectures (portable)

One code to run them all...



Example: Accelerating SAXPY

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

```
#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)
```

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);
```

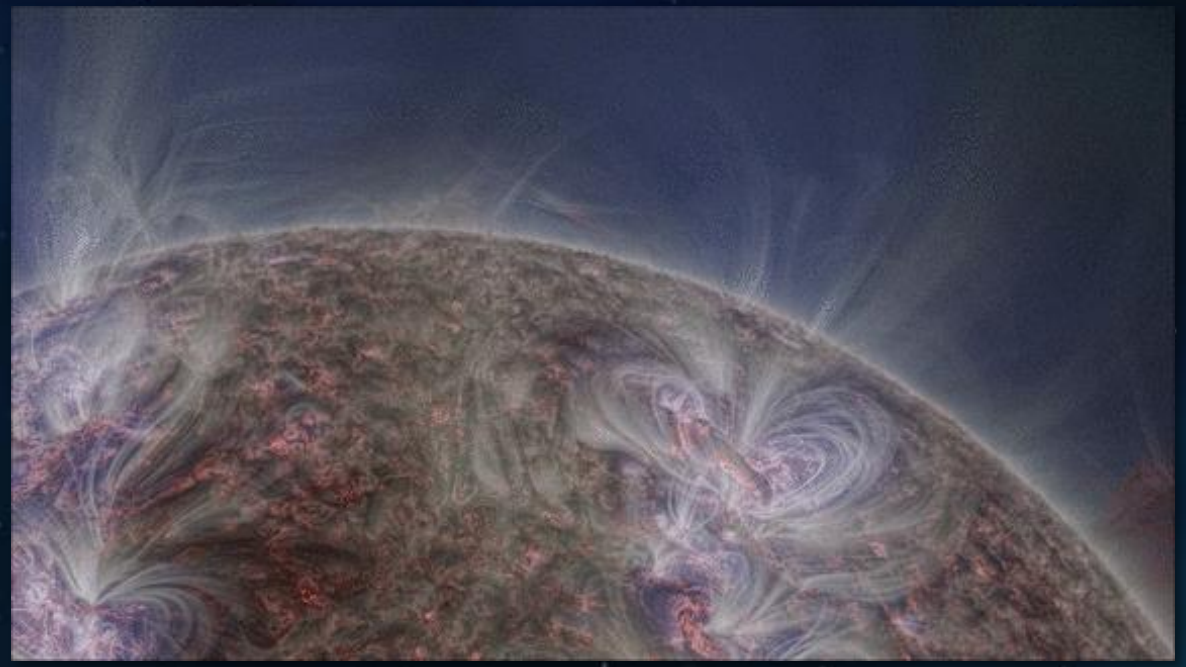
OpenACC

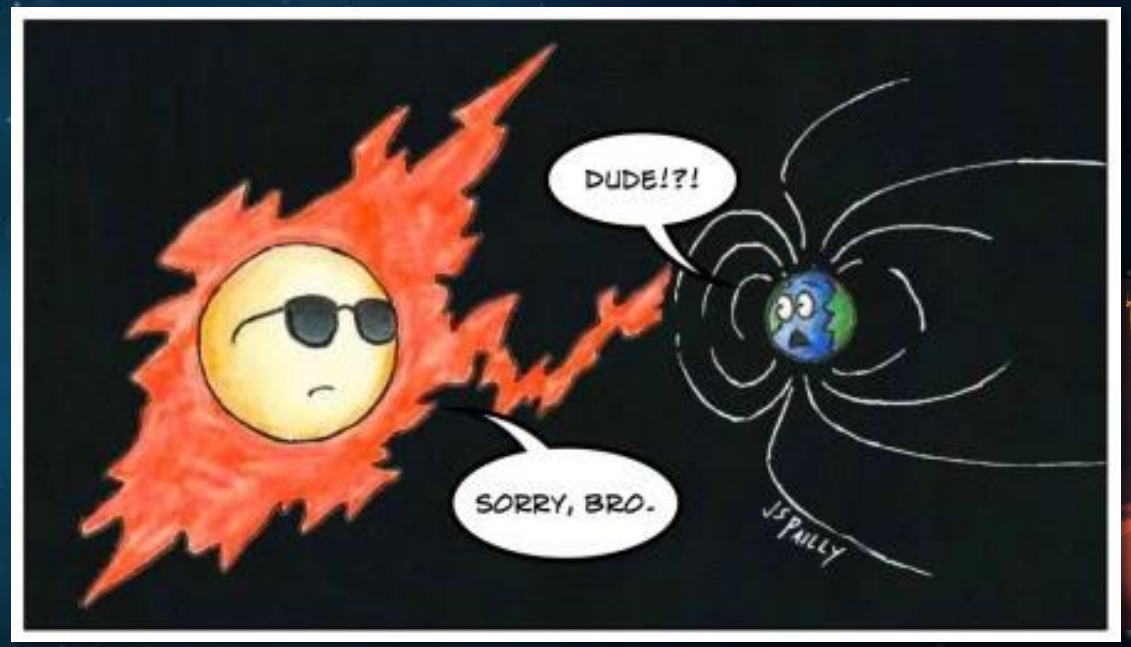
```
#pragma acc kernels
for (i=0; i<N; i++)
    y[i] = a*x[i] + y[i];
```

“Parallel Programming is Hard” – Michael Wolfe

Application: Solar Storm Modeling

- Large explosive events on the Sun including solar flares and coronal mass ejections (CME)
- CMEs can eject billions of 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





Interested? Check out my talk in the exhibit hall!

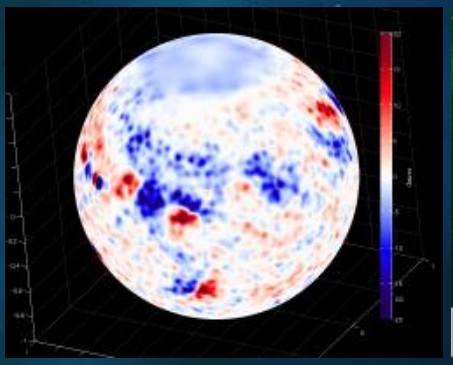
NASA booth #2609

- **WED 4:30 PM**
- **THURS 1:00 PM**

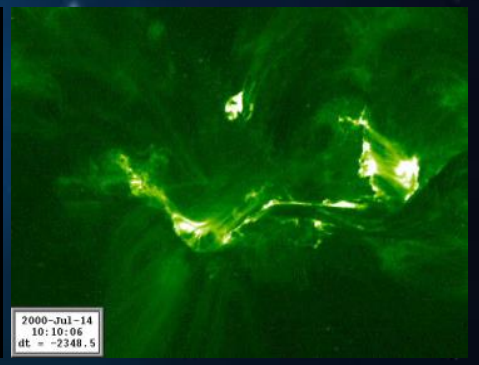
How We Model a Coronal Mass Ejection

Observations

Satellite Observations



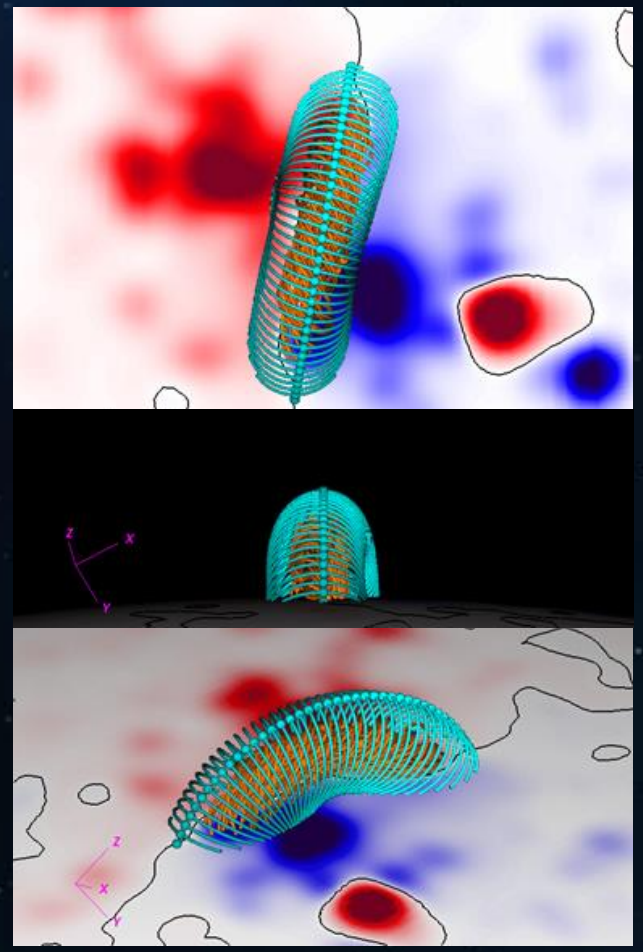
Surface Magnetic Field



EUV images

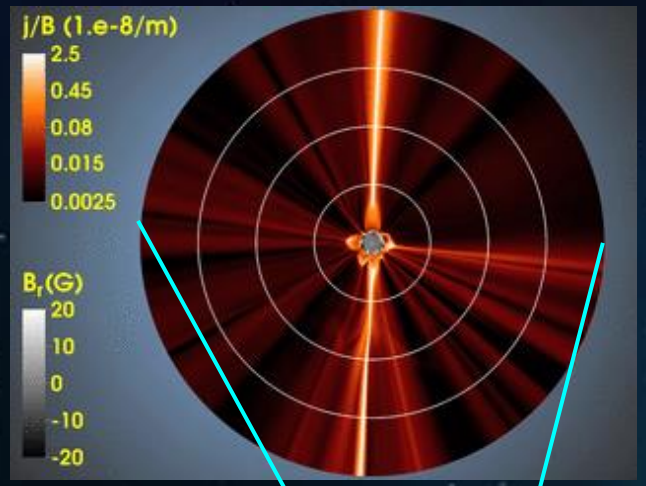
CME Initial Condition

Design and compute pre-CME structure embedded in global approximate magnetic field

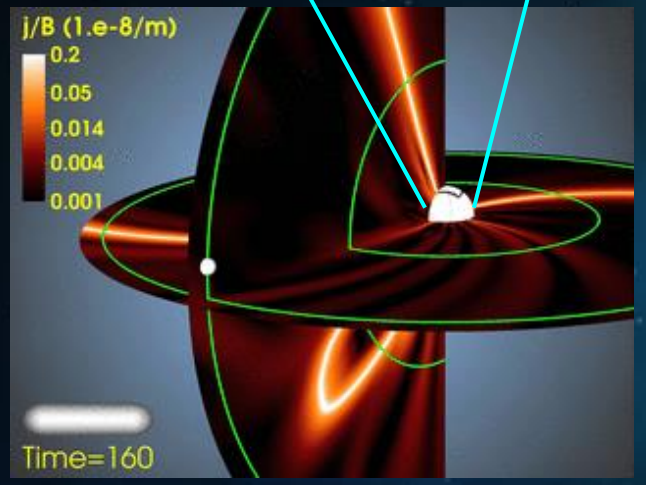


CME Eruption Simulations

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



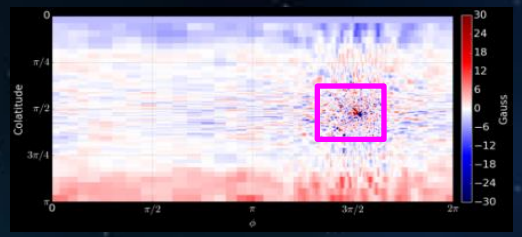
Coronal Simulation



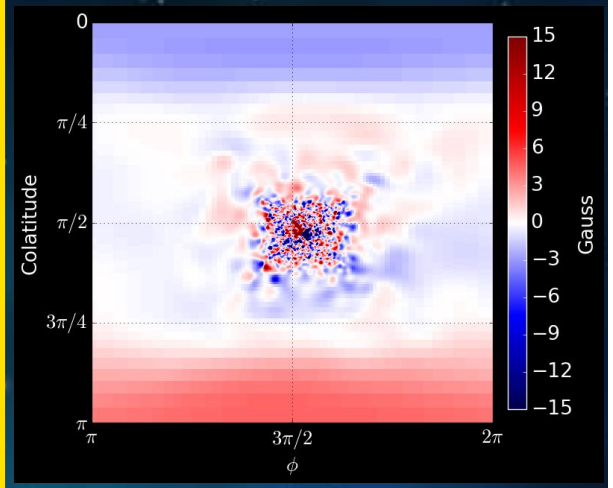
Heliospheric Simulation

Recipe for a pre-launch CME Structure

Isolate launch location, set grid and interpolate



Smooth data to resolve

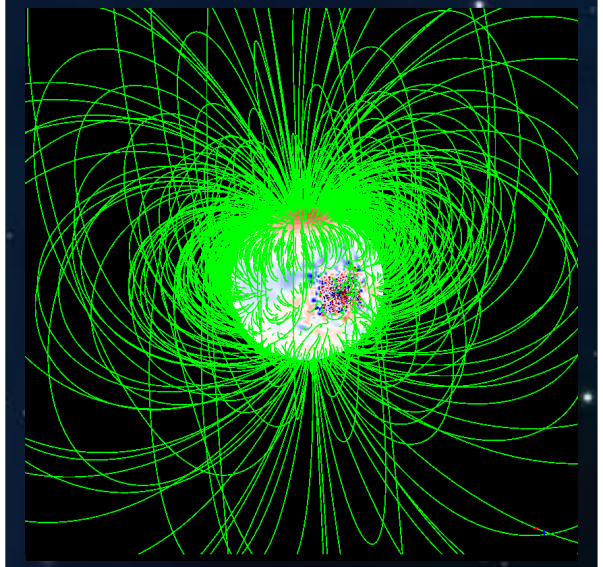


DIFFUSE

GPU TECHNOLOGY CONFERENCE 2016

Compute approximate 3D magnetic field

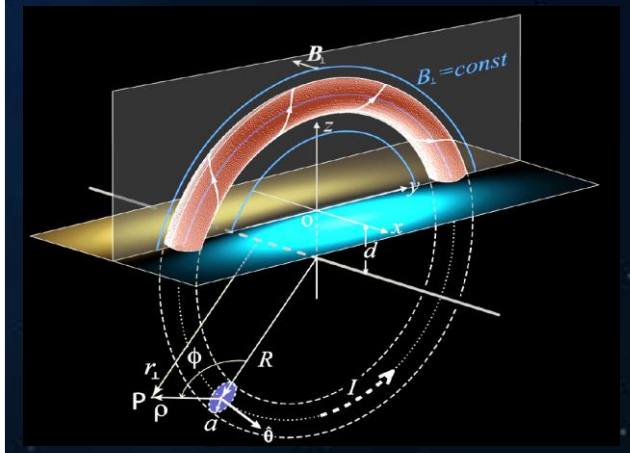
Potential Field

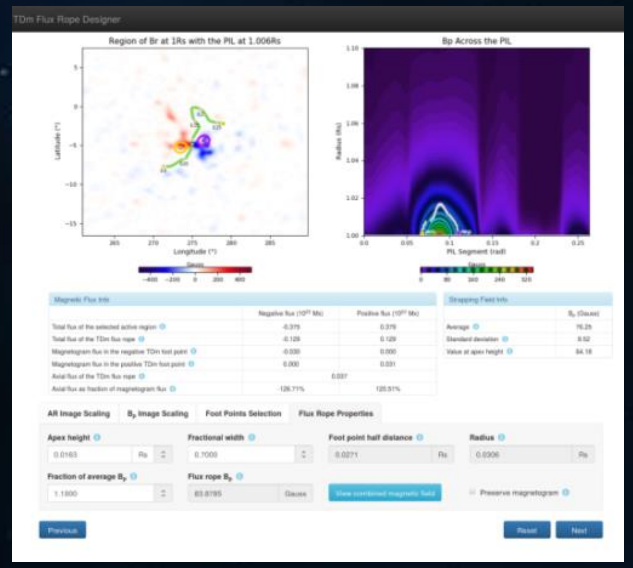


POT3D

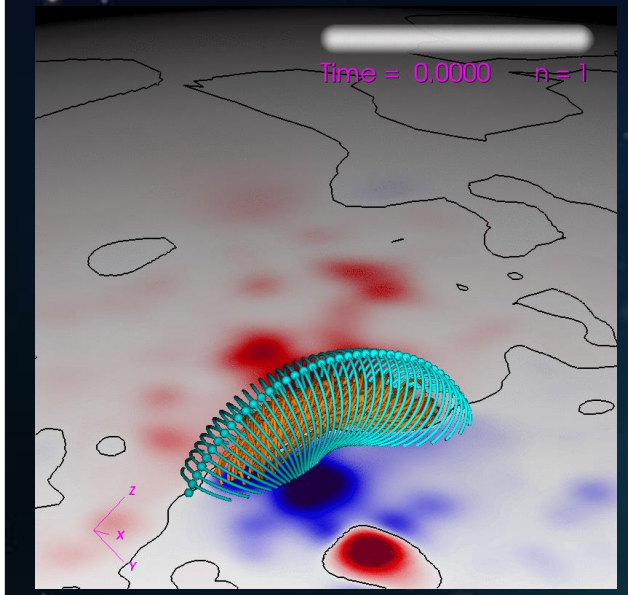
GPU TECHNOLOGY CONFERENCE 2017

Design and insert analytic flux rope





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



MAS (0-Beta)

GPU TECHNOLOGY CONFERENCE 2018

DIFFUSE: Description

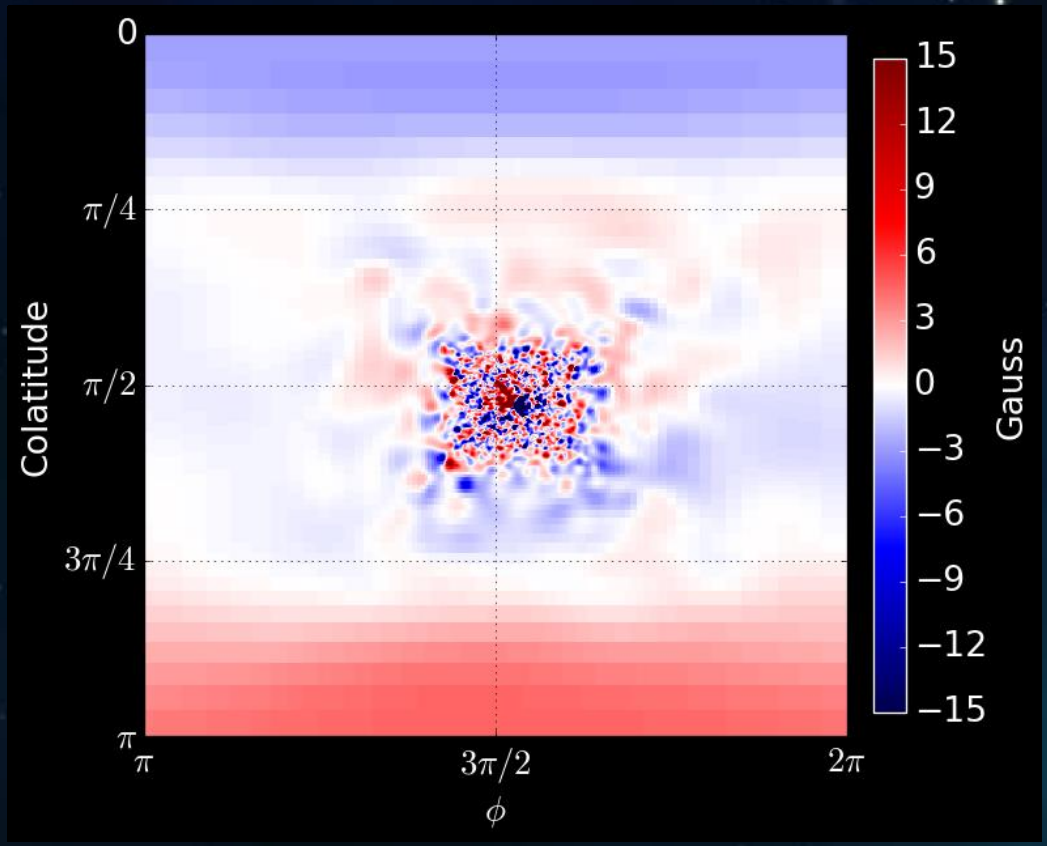
Purpose: Smooth solar surface magnetic field data

Model: Time-dependent spherical heat equation

Algorithm: Explicit finite-difference (stencil)

Code: ~**2000** lines of FORTRAN

Original Parallelism: OpenMP



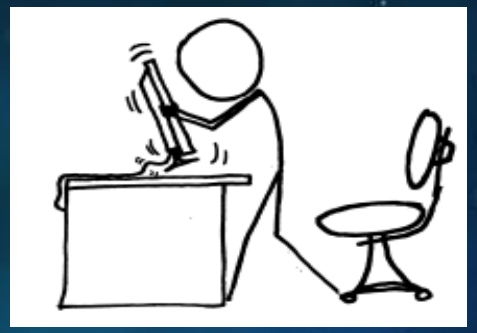
$$\frac{\partial B_r}{\partial t} = \nabla_\perp^2 B_r$$

DIFFUSE: Implementation

Highlights

- Basic array operations
  (explicit loops and array syntax)
- Unstructured data regions to maintain residency
- Collectives for polar boundary conditions

Issues



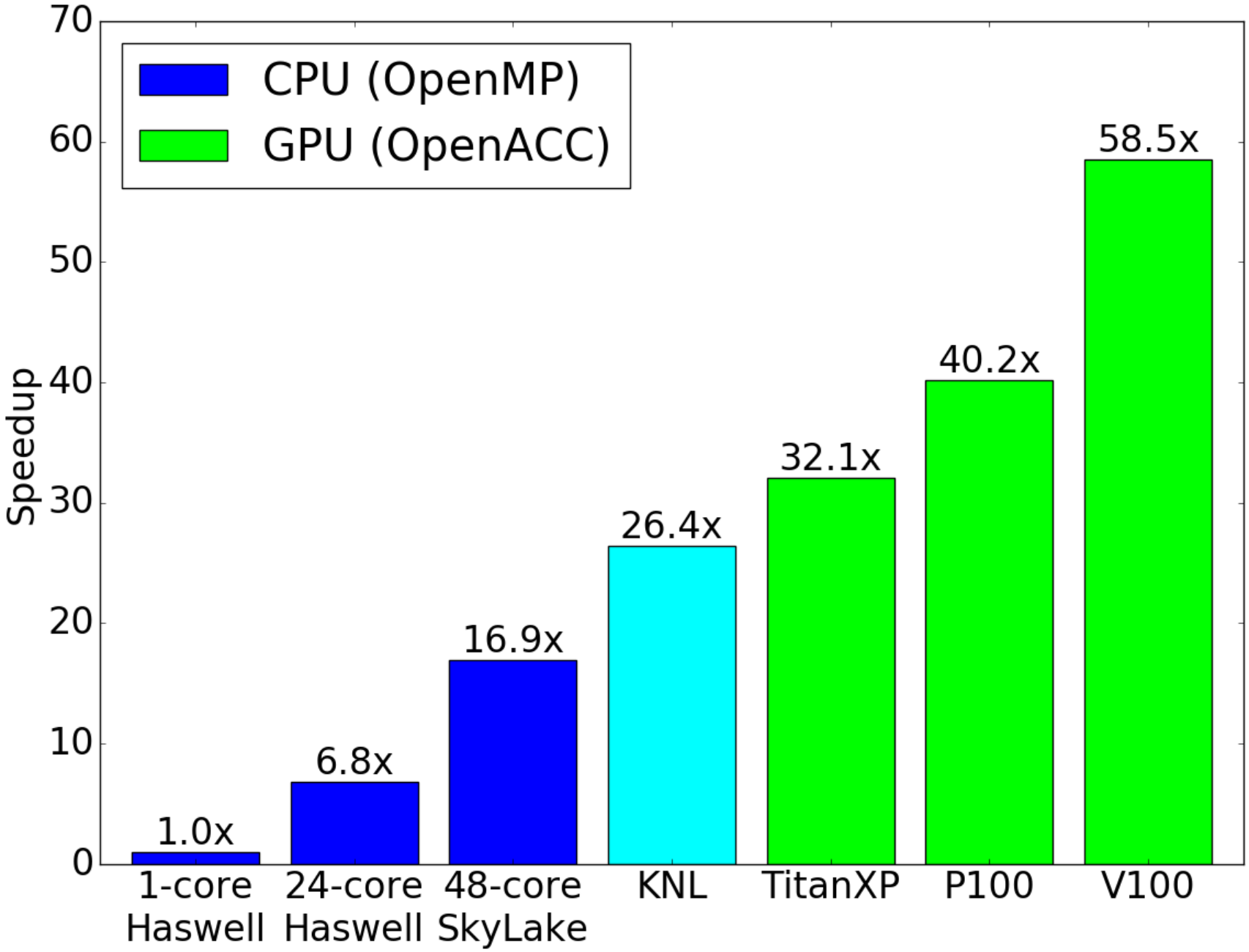
- Problem size often too small to show impressive speedup (many iterations of small loops)

<2%

OpenACC comment lines

DIFFUSE: Results

Timing Results
(large problem size
~3.5 million pts)



POT3D: Description

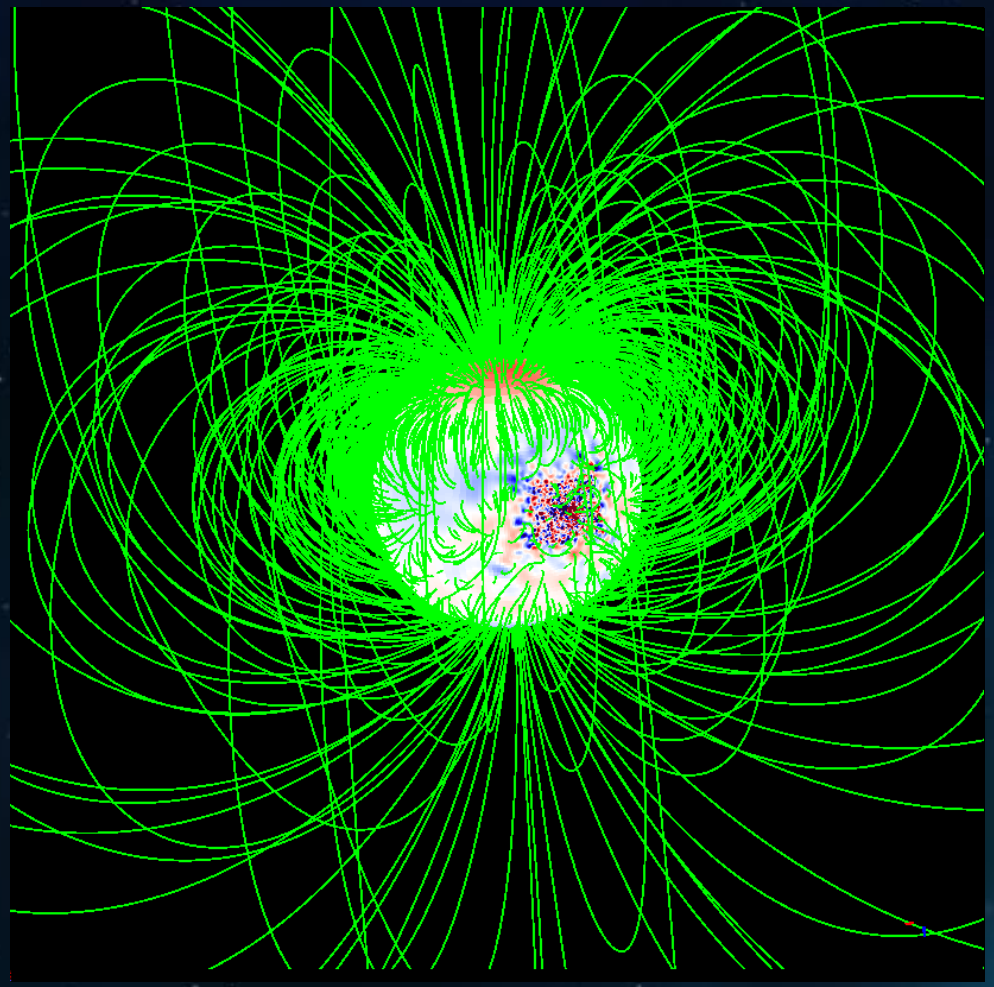
Purpose: Generate approximate magnetic field (potential field) using surface field as boundary condition

Model: Spherical 3D Laplace equation

Algorithm: Sparse matrix preconditioned iterative solver with finite-difference stencil

Code: ~**8000** lines of FORTRAN

Original Parallelism: MPI



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

POT3D: Implementation

- Multi-GPU with CUDA-aware MPI using `set device_num` and `host_data use_device`

Issues

- Two preconditioning algorithms – the "faster" one not parallelizable with OpenACC ☹
  Implemented it by linking to cuSparse library – broke portability!
- Compiler did not yet support all needed clauses
- MPI libraries on HPC systems did not yet support CUDA-aware MPI (or had bugs)

<1%

OpenACC comment lines

POT3D: Results

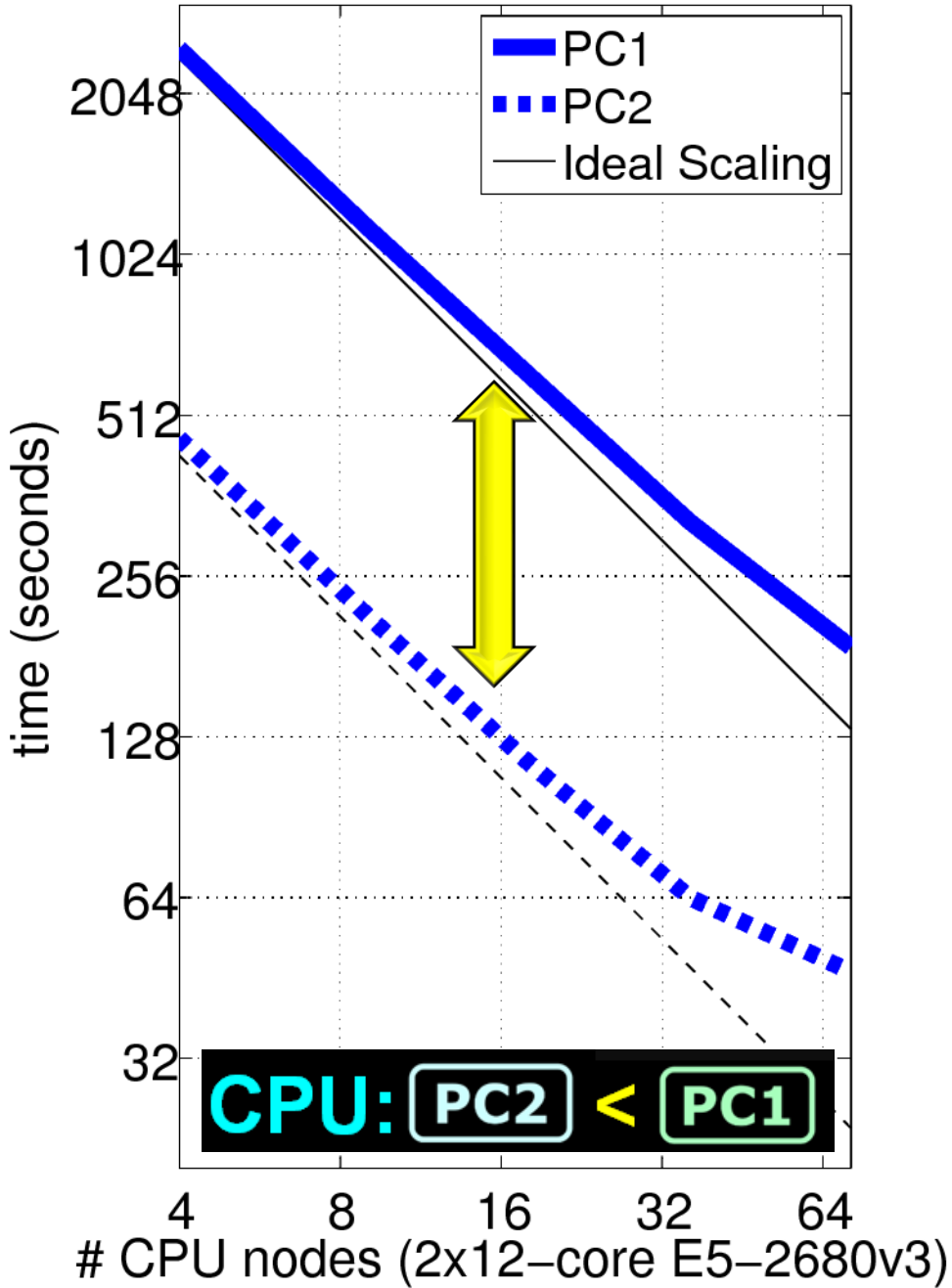
Timing Results (large problem size ~200 million pts)

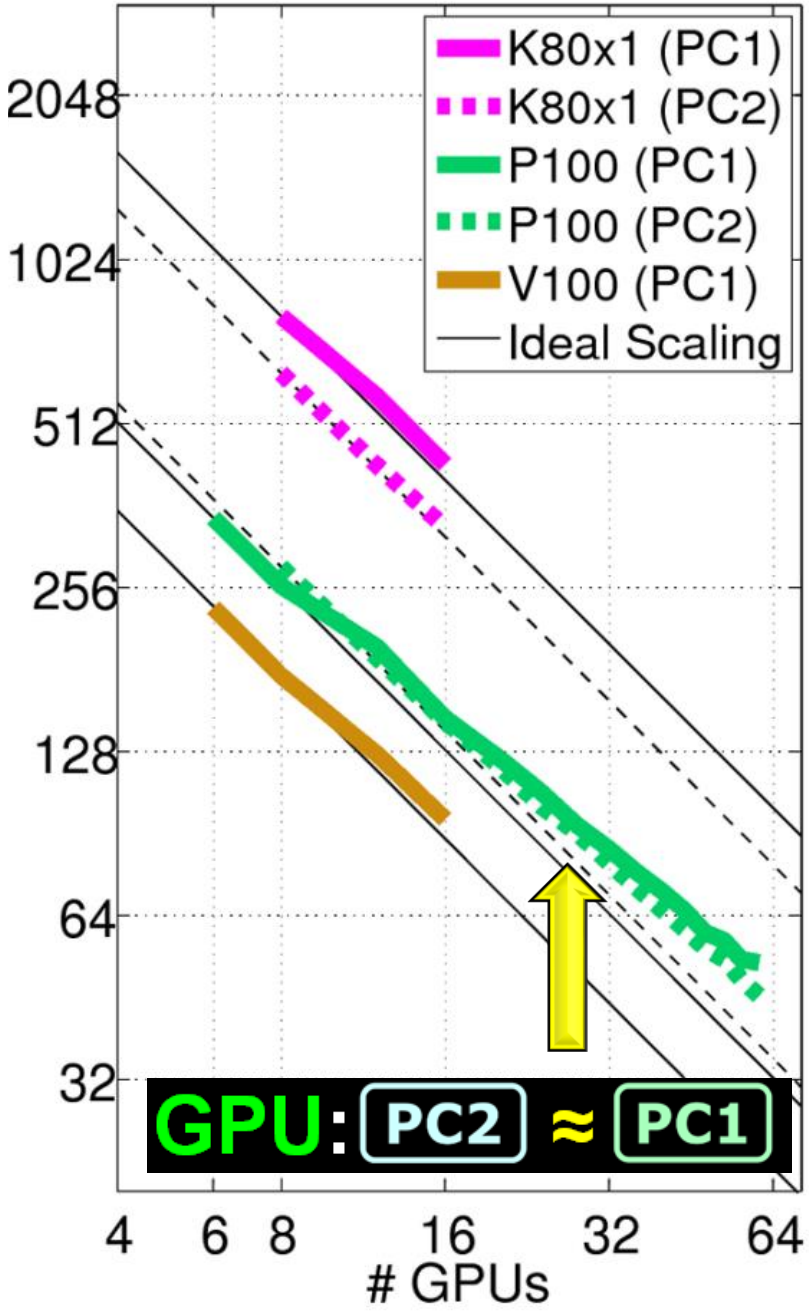
PC2 with cuSparse ~ PC1 on GPU so we can get portability back!

But now our timings compare...

vs.

... so what?





MAS: Description

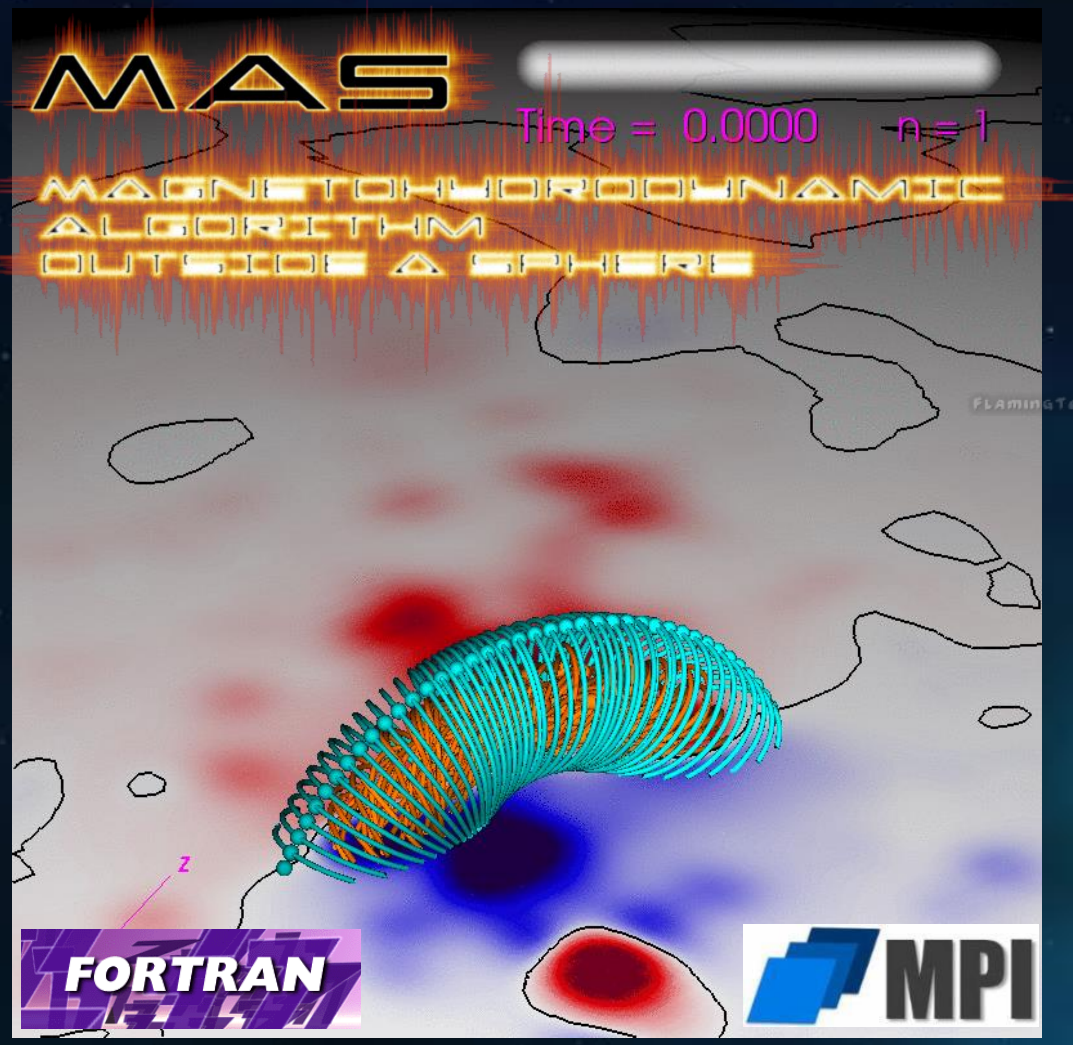
Purpose: General-purpose simulations of the corona and heliosphere for use with solar physics research

Model: Spherical 3D resistive thermodynamic MHD equations - reduced “0-beta” physics model used here

Algorithm: Implicit and explicit time-stepping with finite-difference stencils. Implicit steps use sparse matrix preconditioned iterative solver

Code: ~**57000** lines of FORTRAN

Original Parallelism: MPI



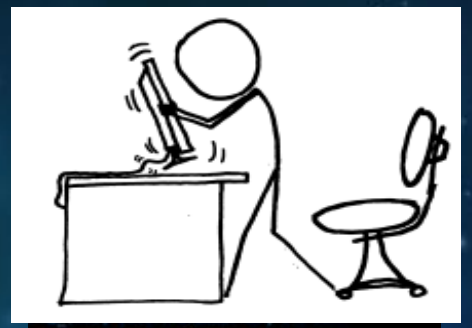
$$\frac{\partial \mathbf{A}}{\partial t} = \mathbf{v} \times (\nabla \times \mathbf{A}) - \frac{c^2 \eta}{4\pi} \nabla \times \nabla \times \mathbf{A}$$

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

MAS: Implementation

- Derived types – manual deep copy, allocatable vs pointer, `host_data`
- Cache-friendly vs vector-friendly loops
- Routines called in accelerator regions
- Array reductions using atomic operations
- Async clause for extra parallelism

Issues



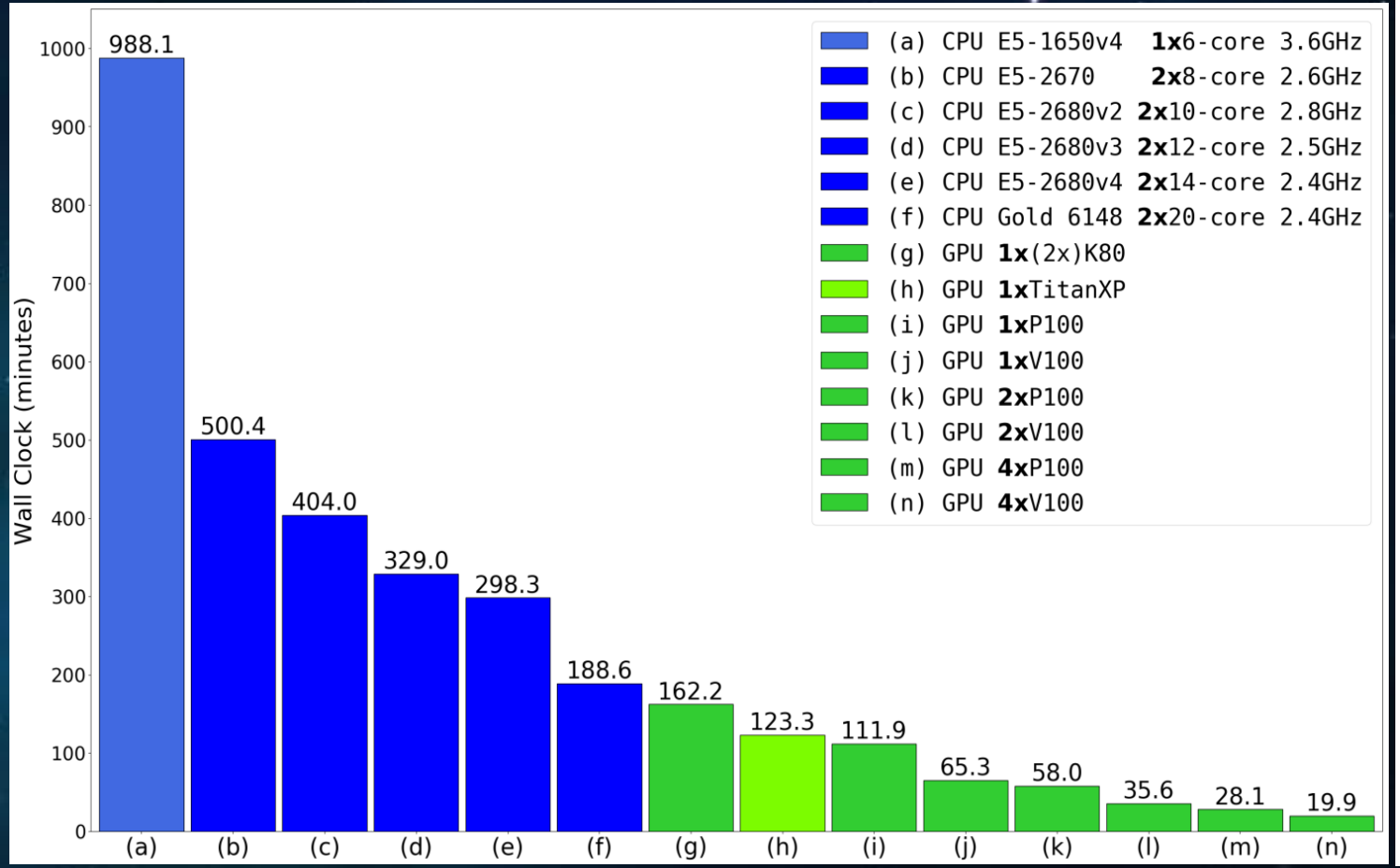
- Lots of small routines needed to be accelerated for data residency
- Memory footprint
- Missing features
- Bugs, bugs, and bugs

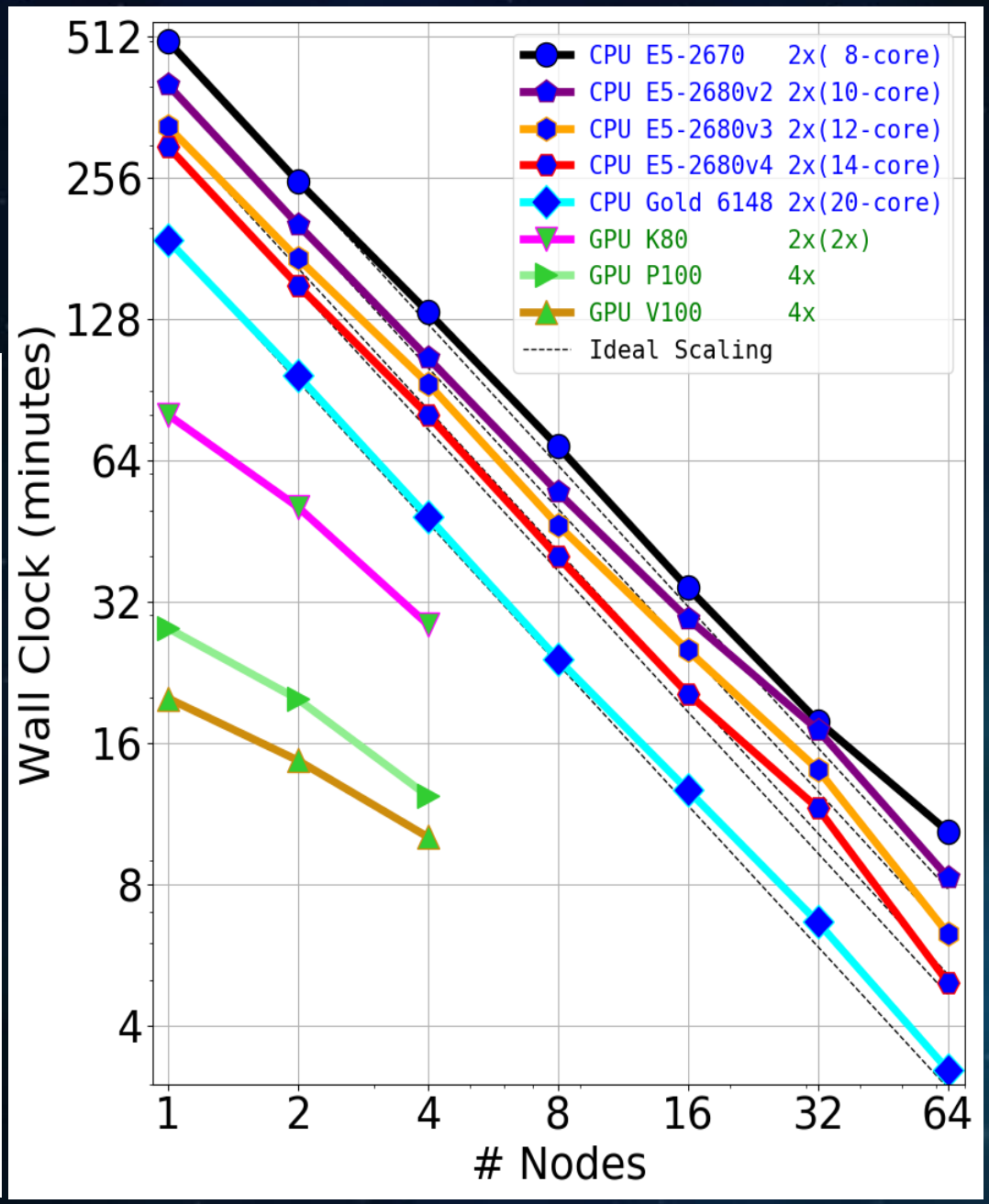
<2%

OpenACC comment lines

MAS: Results

Timing results for small test
(~10.5 million pts)





Some General Tips

- Read release notes and new documentation
- PGI user forum should be your second home
- Use unstructured data regions
- Check compiler "minfo" for "implicit xxxx"
- Profile, profile, profile!
- Don't use new features right away – branch off to test first
- Work with HPC staff to set up software stack before testing
- Know your problem size – too small=weak/no performance, too big=won't fit in memory

Consumer cards more useful then they may seem

- **Memory-bound algorithms perform nearly as good as server cards (even with FP64)**
- **FP32 flop-bound algorithms also perform nearly as good**
- **You get to have some awesome gaming on breaks ☺**
- **You can develop and run on your laptop (see next talk!)**

References

Slides available at:
predsci.com/~caplanr

DIFFUSE: "Non-uniform diffusion of the Solar surface magnetic field: Code acceleration using OpenACC for both GPUs and x86"
http://on-demand.gputechconf.com/gtc/2016/posters/GTC_2016_Astronomy_and_Astrophysics_AA_05_P6259_WEB.pdf

POT3D: "Potential Field Solutions of the Solar Corona: Converting a PCG Solver from MPI to MPI+OpenACC"
http://on-demand.gputechconf.com/gtc/2017/video/s7535-ronald-caplan-potential-field-solutions-of-the-solar-corona-converting-a-pcg-solver-from-mpi-to-mpi+openacc.mp4

"From MPI to MPI+OpenACC: Conversion of a legacy FORTRAN PCG solver for the spherical Laplace equation"
https://arxiv.org/abs/1709.01126

MAS: "Solar Storm Modeling using OpenACC: From HPC cluster to `in-house'"
http://on-demand.gputechconf.com/gtc/2018/video/S8847

"Using OpenACC to Port Solar Storm Modeling Code to GPUs"
https://devblogs.nvidia.com/solar-storm-modeling-gpu-openacc

"GPU Acceleration of an Established Solar MHD Code using OpenACC"
https://arxiv.org/abs/1811.02605