

Enabling Human Exploration of the Red Planet

Bill Jones Ashley Korzun Eric Nielsen Aaron Walden
NASA Langley Research Center

Chris Henze Pat Moran Tim Sandstrom
NASA Ames Research Center

Justin Luitjens
NVIDIA Corporation

Mohammad Zubair
Old Dominion University

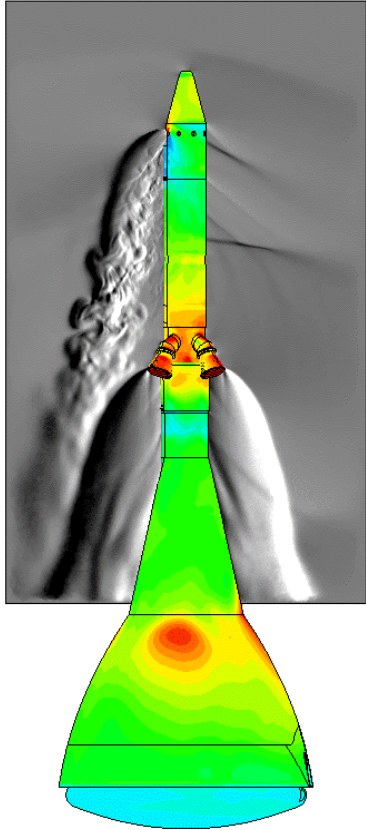
<https://fun3d.larc.nasa.gov>



This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.



Some Exascale Drivers



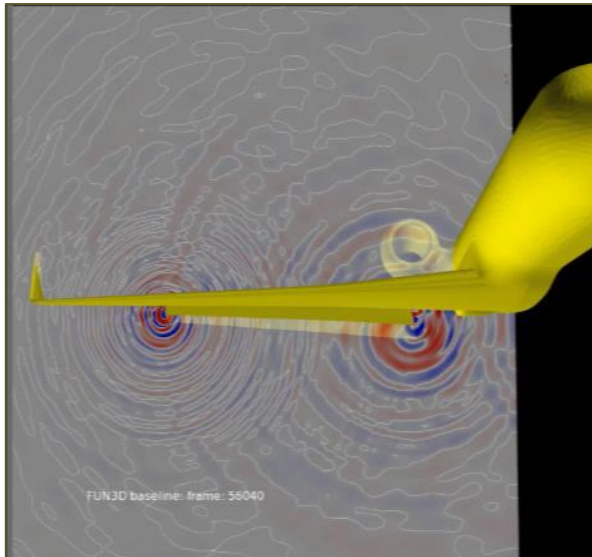
Launch Abort System

**Adjoint for
Chaotic
Systems
(NASA/MIT)**



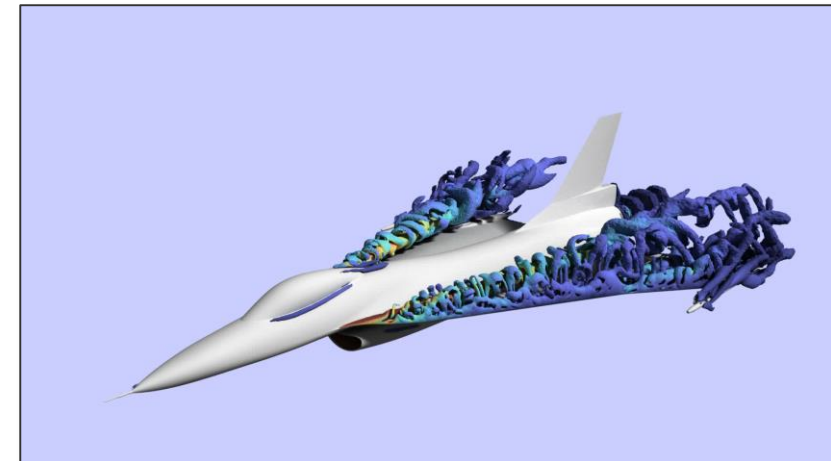
Rotorcraft

**NASA/Boeing
Truss-Braced Wing**



**Aeroacoustics:
Gulfstream
G550**

**Separated
Flows**

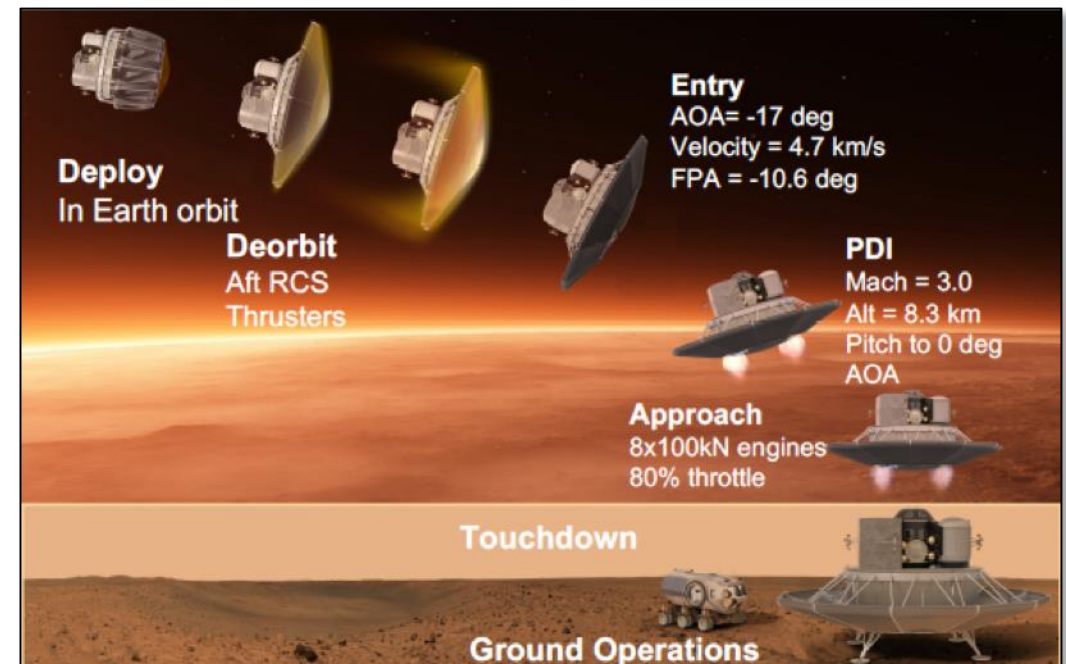


“Enabling Human Exploration of the Red Planet”

- Allocations for CY2019 through Summit Early Science and INCITE programs
 - Total award of 305,000 Summit node-hours; FUN3D equivalent of ~305,000,000 Xeon Skylake core-hours
- Team members include NASA Langley, NASA Ames, NVIDIA, and Old Dominion University
 - LaRC: Science and computational expertise
 - ARC: Large-scale visualization
 - NVIDIA, ODU: Kernel optimizations

Goals

- *Science*: Better understanding of retropropulsion flow physics during Mars entry of human-scale lander
- *Computational*: Demonstrate production readiness and efficiency advantages of GPU implementation at scale








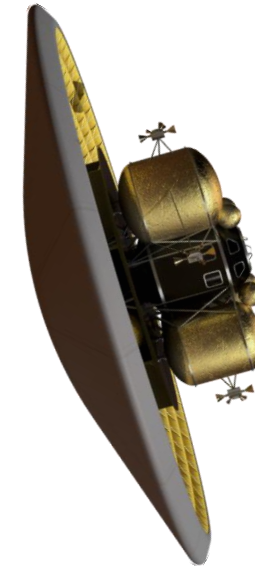


Retropropulsion for Human Mars Exploration

Human-scale Mars landers require new approaches to all phases of Entry, Descent, and Landing

- Cannot use heritage, low-L/D rigid capsules → deployable hypersonic decelerators or mid-L/D rigid aeroshells
- Cannot use parachutes → retropropulsion, from supersonic conditions to touchdown
- No alternative to an extended, retropropulsive phase of flight

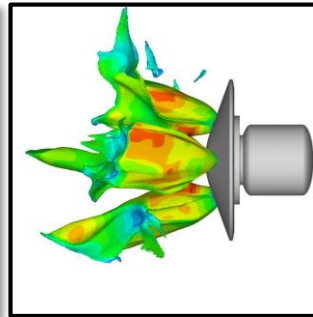
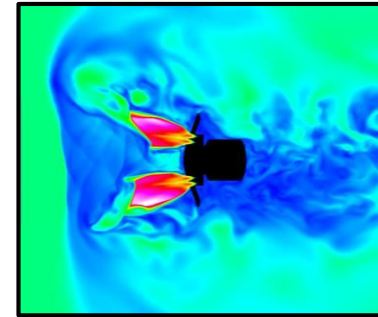
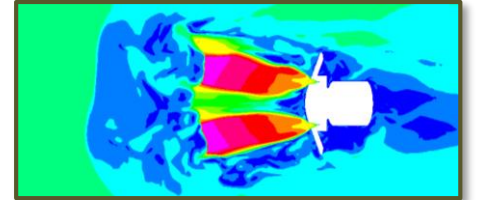
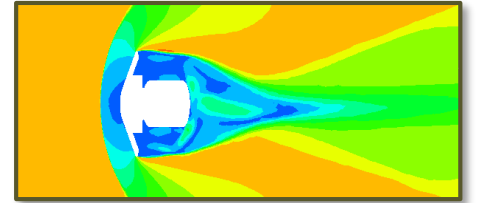
	Viking	MPF	MER	PHX	MSL	Human-Scale Lander (Projected)
Vehicles to Scale						
Diameter (m)	3.505	2.65	2.65	2.65	4.5	16 - 19
Entry Mass (t)	0.930	0.585	0.840	0.602	3.151	47 - 62
Landed Mass (t)	0.603	0.360	0.539	0.364	1.541	36 - 47
Landing Altitude (km MOLA)	-3.5	-1.5	-1.3	-3.5	-4.4	+/- 2.0
Peak Heat Rate (W/cm²)	24	106	48	56	~120	~120 - 350



Steady progression of “in family” EDL

Powered Flight in an Atmosphere

- Aerodynamic effects can be significant during powered descent
- Retropropulsion environment can significantly impact vehicle performance
 - Large variations in aero forces/moments challenge the ability to maintain control of the vehicle and accurately reach the landing target
 - Sensitive to engine operating conditions, start-up transients, atmospheric conditions, engine configuration and vehicle integration
- Highly unsteady flow field behavior, broad range of length scales, very large computational domains requiring fine resolution, strong shocks and massively separated flows all must be addressed to accurately simulate retropropulsion in an atmosphere



Examples of unsteady RANS solutions with insufficient spatial resolution, while stressing available conventional computational resources

Vehicle-level design decisions are directly impacted by the ability to characterize and bound aerodynamic-propulsive interference effects



Why Summit?

- Simulating interactions between atmosphere and retropropulsion plumes at sufficient spatial resolution to resolve governing phenomena with a high level of confidence not feasible with conventional computational capabilities
 - Single solution requires 200,000+ CPU hours with severe limitations on spatial resolution
 - Thousands of solutions eventually required to model flight performance

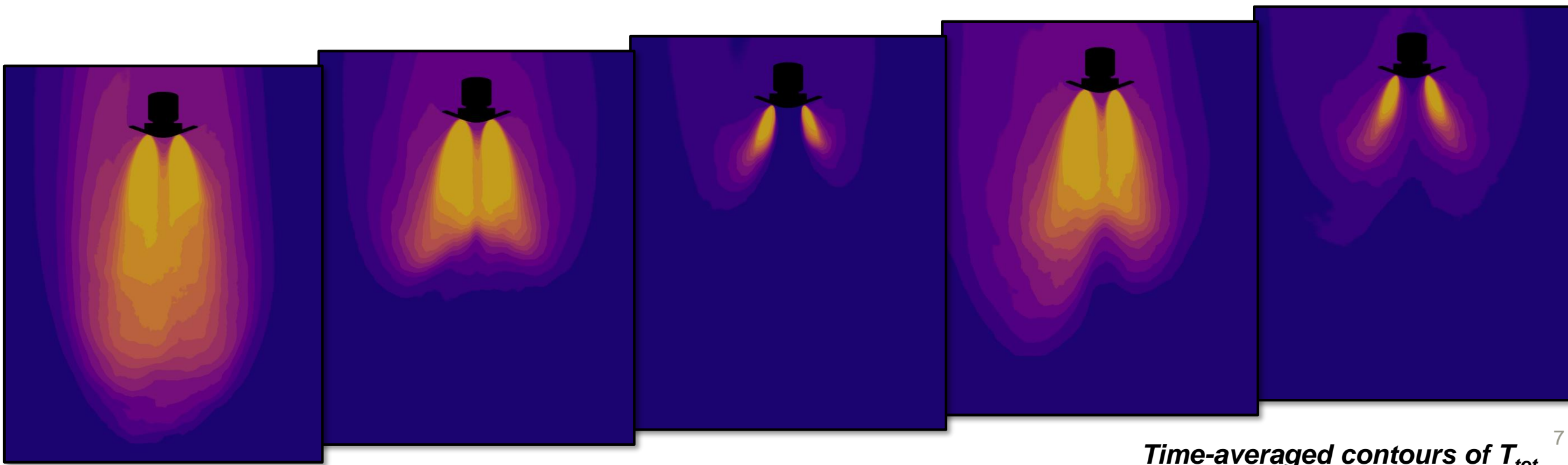
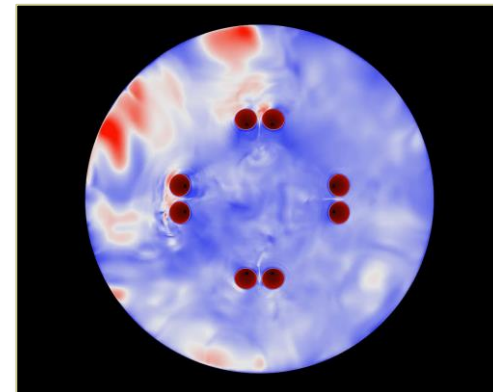
Enabling Capabilities Provided by Summit

- Application of Detached Eddy Simulation methods with resolution of relevant length scales
- Meaningful statistics and characterization of unsteady flowfield behaviors
- Domain dimensions in kilometers with the ability to resolve flow features on the order of centimeters
- Complete redefinition of the state-of-the-art for powered descent aerodynamics characterization for both requisite accuracy and computational environment/implementation



Summit Campaign

- Campaign aligns closely with 2020 wind tunnel entry
- Rather than pursue small number of “hero” simulations, exploring large ensemble of asymmetric throttle conditions across freestream Mach numbers from 0.8 to 2.4
- Spatial mesh sizes ranging from ~1-10 billion elements
- Long temporal duration (~1.6 sec real time) to capture diverse transients and statistics
- Individual runs can reach 200 TB of output; entire project will exceed 1 PB



Time-averaged contours of T_{tot} ⁷



Game-Changing Performance

Typical Job of 6.5B Elements, 200K Time Steps

Conventional system with capacity policy

- 5,000 Xeon Skylake cores (125 nodes)
- 3.5 months compute time
- 22 5-day queue submissions + waits

Summit

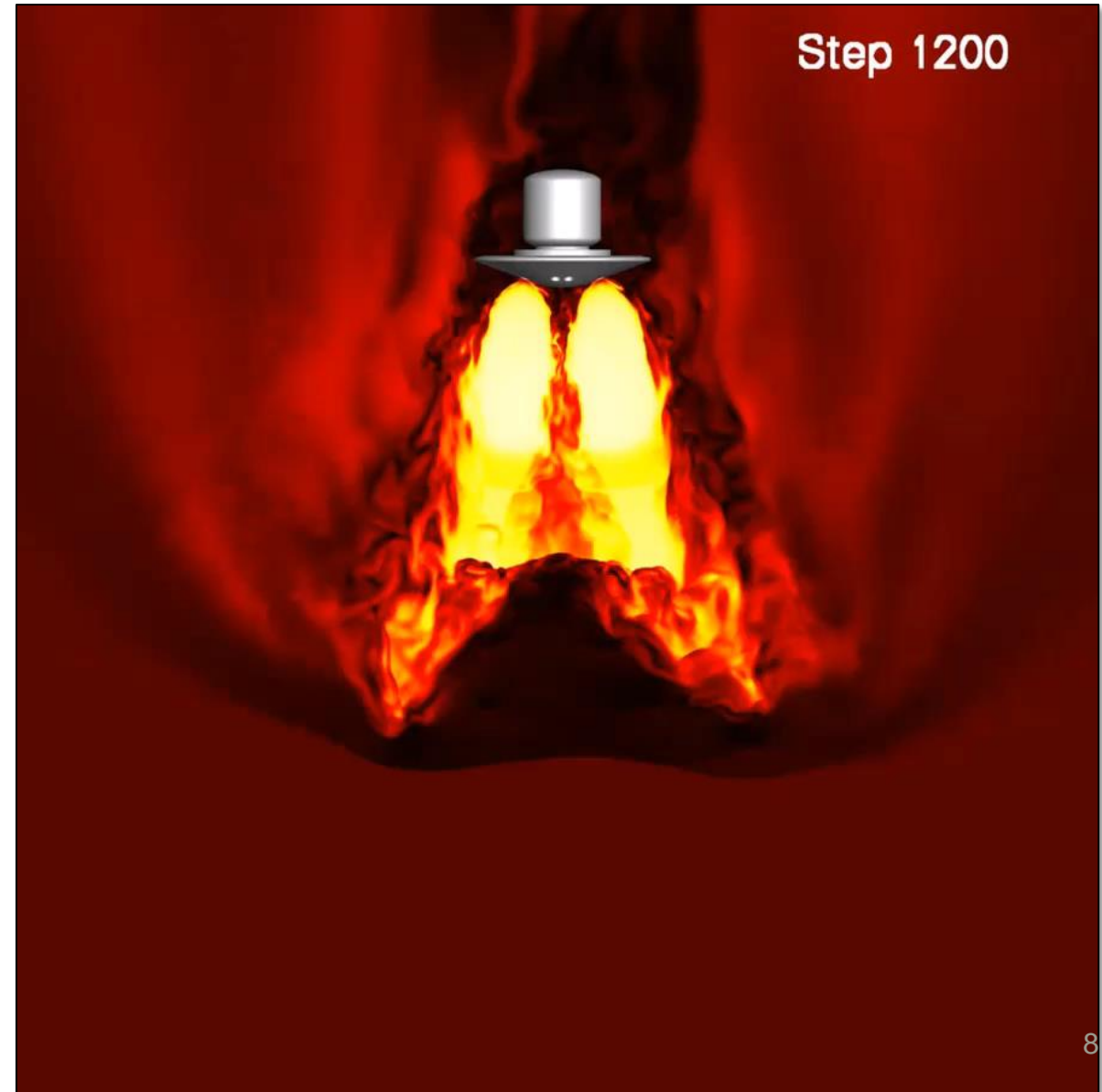
- 552 Tesla V100s (92 nodes)
- 5 days compute time
- 10 12-hour queue submissions
- Usually no queue wait, 1-2 hours at most

Conventional system with capability policy

- 106,500 Xeon Skylake cores (2,663 nodes)
- 5 days compute time
- 5-10 queue submissions

We are running 4-5 such jobs simultaneously:

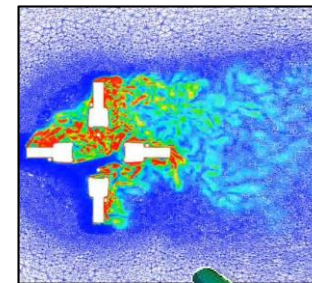
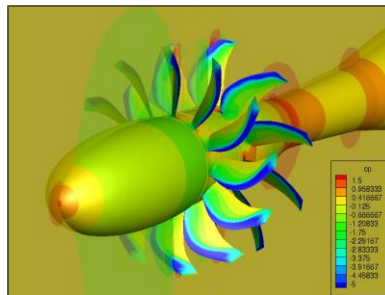
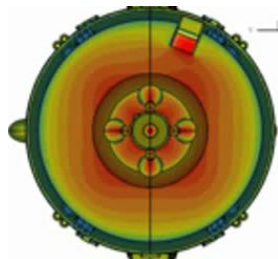
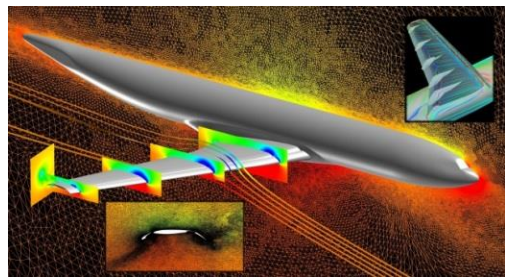
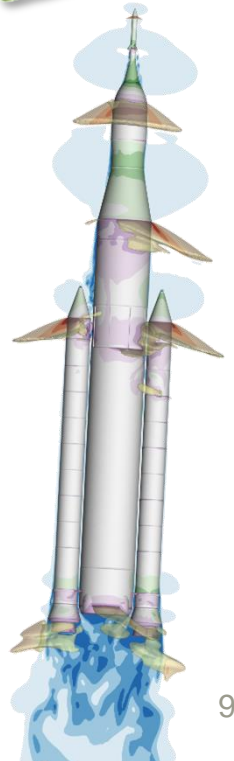
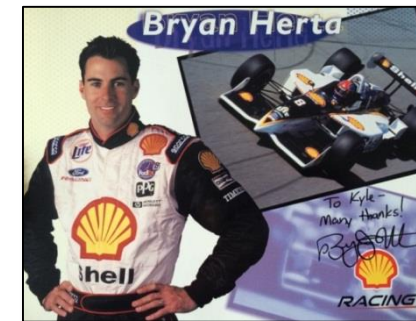
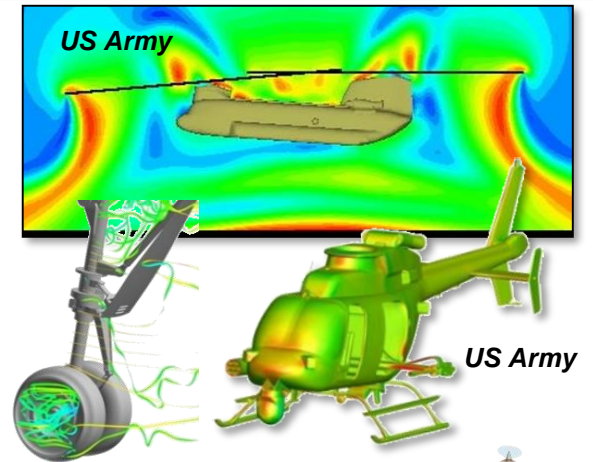
Leadership class HPC is reducing our learning cycle from years to days



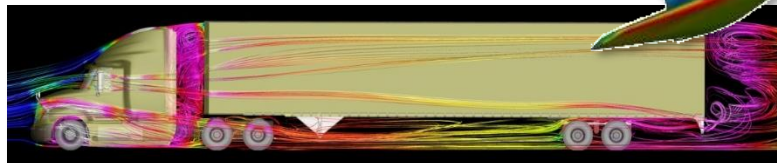
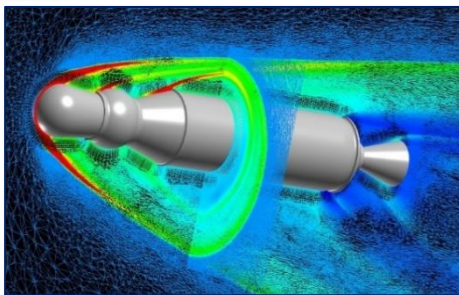
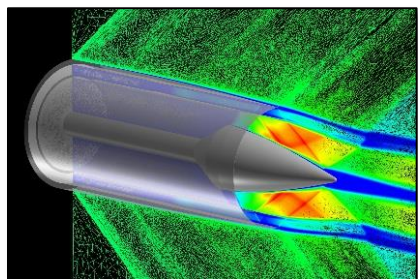
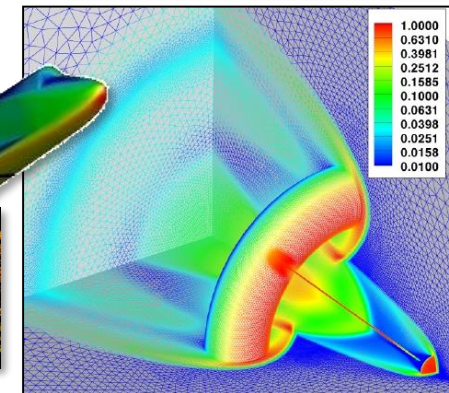


FUN3D Overview

- Established as a research code in late 1980s; now supports numerous internal and external efforts across the speed range
- Solves 2D/3D steady and unsteady Euler and RANS equations on node-based mixed element grids for compressible and incompressible flows
- General dynamic mesh capability: any combination of rigid / overset / morphing grids, including 6-DOF effects
- Aeroelastic modeling using mode shapes, full FEM, CC, etc.
- Constrained / multipoint adjoint-based design, mesh adaptation
- Distributed development team using agile/extreme software practices including 24/7 regression, performance testing
- Capabilities fully integrated, online documentation, training videos, tutorials



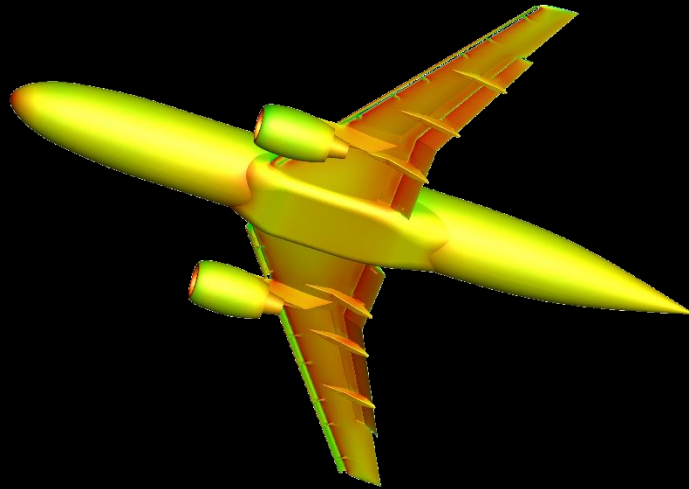
Georgia Tech



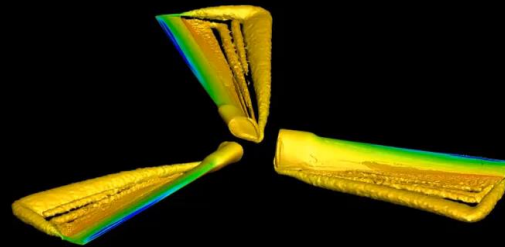


Early GPU-Based Simulations

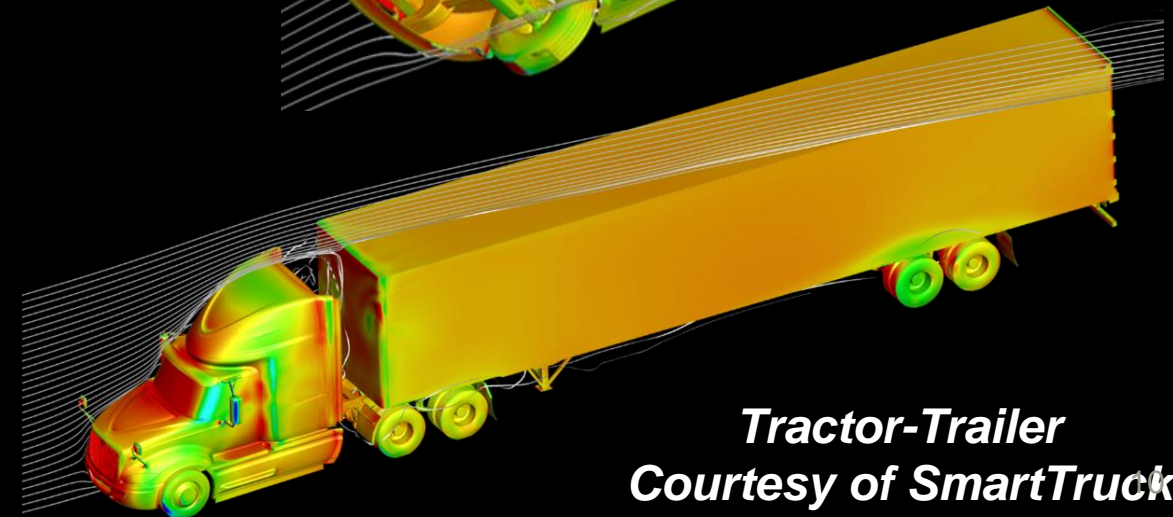
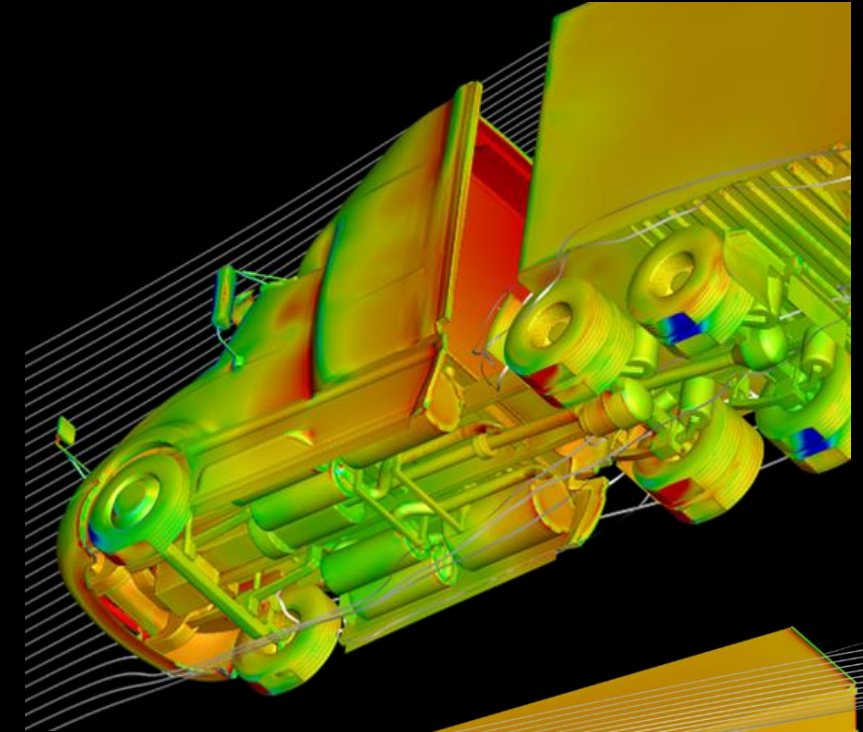
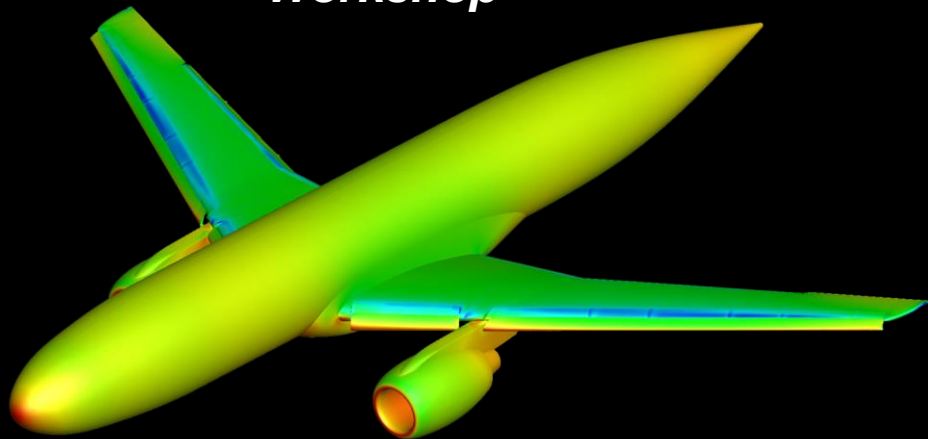
Titan and Summit



***AIAA High-Lift
Workshop***



TRAM Rotor in Hover



***Tractor-Trailer
Courtesy of SmartTruck***



FUN3D Primary Motifs

- FUN3D solves the Navier-Stokes equations of fluid dynamics using implicit time integration on general unstructured grids
- This approach gives rise to a large block-sparse system of linear equations that must be solved at each time step
- Two kernels are generally the largest contributors to run time:
 - Kernel 1: Construction and storage of the compressible viscous flux Jacobians
 - Kernel 2: Multicolor point-implicit linear solver used to solve $Ax=b$

```
for i = 1 to n_time_steps do  
  Form Right Hand Side  
  Form Left Hand Side  
  Solve  $Ax = b$   
  Update Solution  
end for
```



History of GPU Efforts

Nov 2010			Initial discussions with Stan Posey/NVIDIA at SC10
ca. 2011	GTX 470 Also GTX 480, Tesla M2050	CUDA C	Early work with Austen Duffy (FSU)* -- ~1.5x on point solver (linear algebra) *and EM Photonics via NAVAIR
Nov 2013	K20	OpenACC	Began OpenACC with Dave Norton (PGI) at SC13 – 2x on point solver
ca. 2014	K40	OpenACC	Worked with Justin Luitjens to put OpenACC throughout FUN3D – many issues, compiler bugs, poor performance
ca. 2014	K40	OpenACC	Extended FUN3D MPI layer to accommodate device data – MPT bugs
ca. 2014	K40	OpenACC / CUDA Fortran	Worked with Justin/Dominik Ernst to extend point solver using OpenACC and CUDA Fortran – 4x speedup
May 2016	K40	OpenACC / CUDA Fortran	ORNL/UDel hackathon: Continued to struggle with OpenACC approach, Zubair has good success with CUDA Fortran for point solver (~7x over cuBLAS)
Nov 2016	K40 / P100	CUDA C	Zubair et al. publish CUDA C point solver at SC16, eventually incorporated into cuSPARSE
Aug 2017	V100	CUDA C	ORNL/LaRC hackathon: Large speedups (~6x) on early access V100 for linear algebra and LHS, convinced to go fully CUDA and abandon OpenACC
July 2018	V100	Kokkos	Implemented point solver in Kokkos, decent speed, though cumbersome



Implementation Overview

Goals:

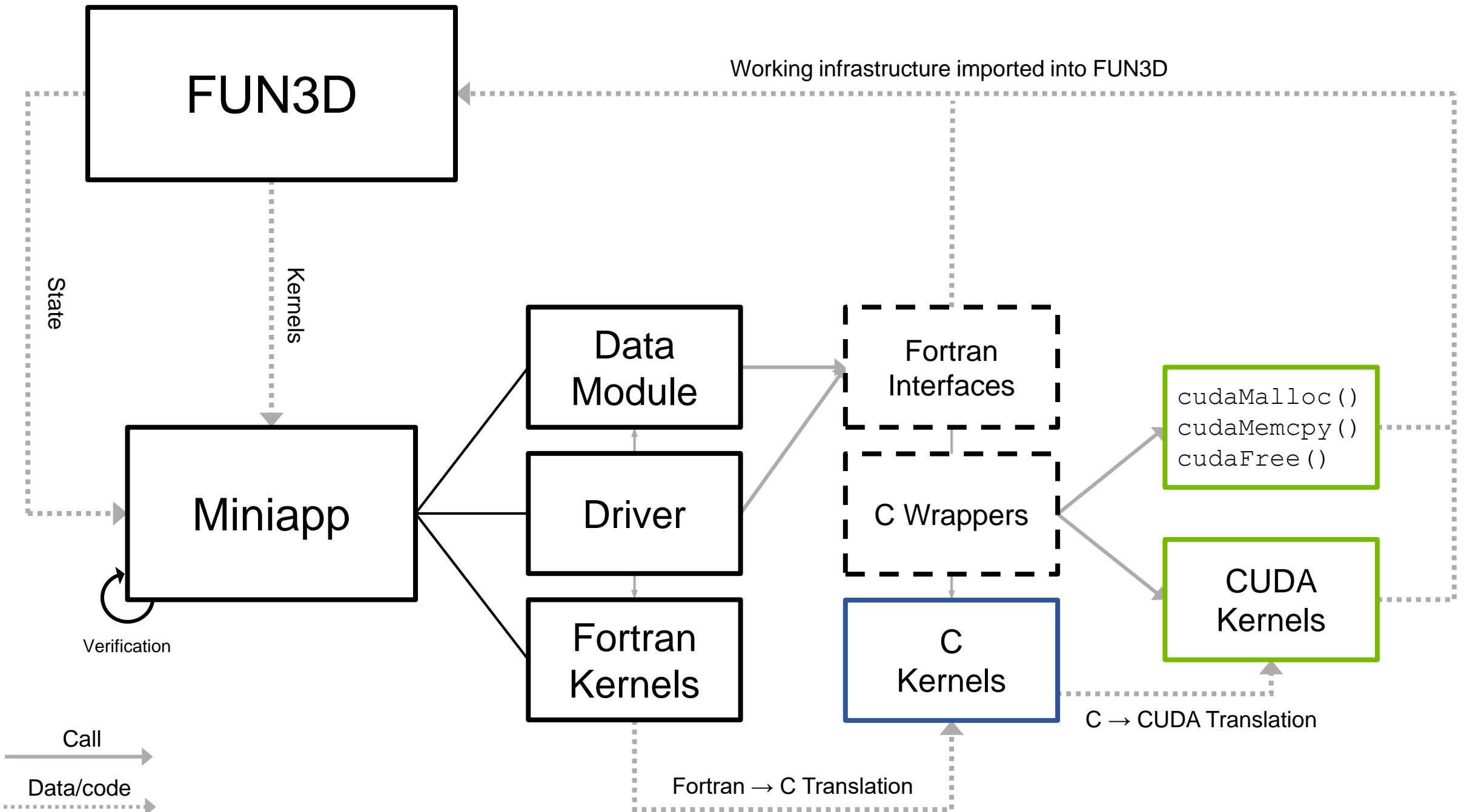
- Perform entirety of FUN3D's PDE solve on device using CUDA
- Minimal data movement between host and device
- Use FUN3D's existing Fortran MPI-based front end
- Change as little of FUN3D as possible (esp. data structures)



Implementation Overview

Strategy:

- Translate ~110 computational kernels using miniapp
- Use `iso_c_binding` to create device mirrors of Fortran variables
- Push necessary data to device before time-stepping loop
- Call interfaces which bind C wrappers around CUDA kernels
- Use CUDA-aware MPI with device pointers
- Data extraction/visualization: field data pulled from device to asynchronous Fortran buffer on host; disk I/O completely hidden





C/Fortran Interoperability Concerns

A very brief summary of our findings:

- **Use** `iso_c_binding`
- `storage_size` **seems** to be portable
- Pointer arithmetic with `transfer`
- Be careful with logicals
- OpenMPI using Intel compiler does not like `c_ptr`
- Create interoperable mirror types to use in CUDA



Data Module: Derived Type Example

Original Derived Type

```
type, public :: elem_type
  integer :: a
  integer, dimension(:, :), &
    allocatable :: b
  integer, dimension(:, :, :), &
    allocatable :: c
end type elem_type
```

Interoperable Mirror Derived Type

```
type, bind(c) :: elem_type_p
  integer(c_int) :: a
  type(c_ptr) :: b
  type(c_ptr) :: c
end type elem_type_p
```

ORDER MATTERS

Interoperable C Struct

```
typedef struct elem_type_p {
  int a;
  int* b;
  int* c;
} elem_type_p;
```

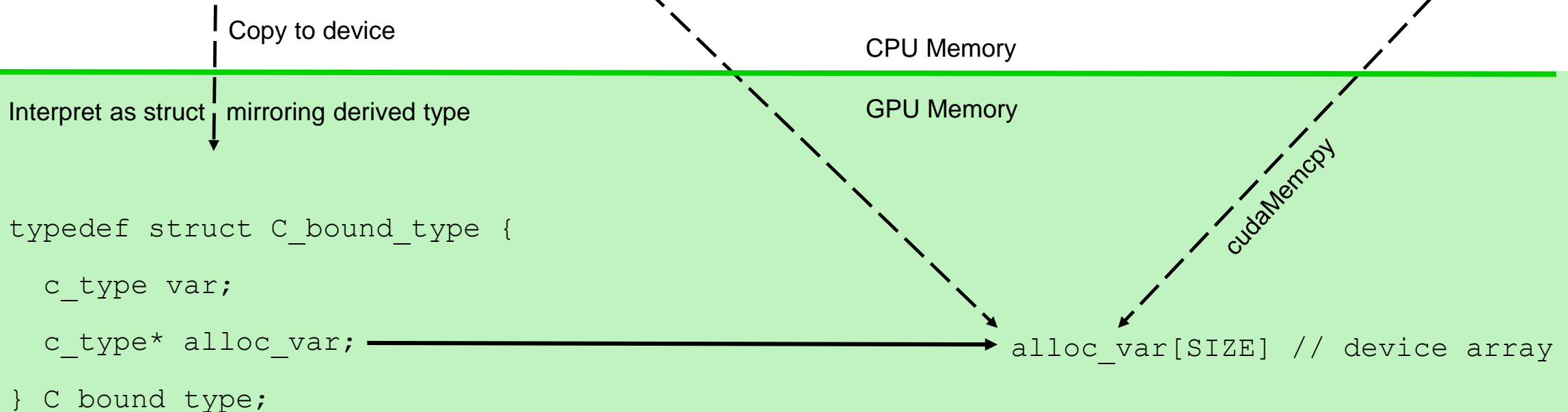


Data Module: Derived Types

Mirror original derived type with C-bound derived type

```
type, bind(c) :: C_bound_type
  type(c_type) :: var
  type(c_ptr) :: alloc_var_d
end type C_bound_type
```

```
type F_derived_type ! original
  f_type :: var
  f_type, allocatable :: alloc_var
end type F_derived_type
```



We can use this method to access arrays of derived types with allocatables on the device



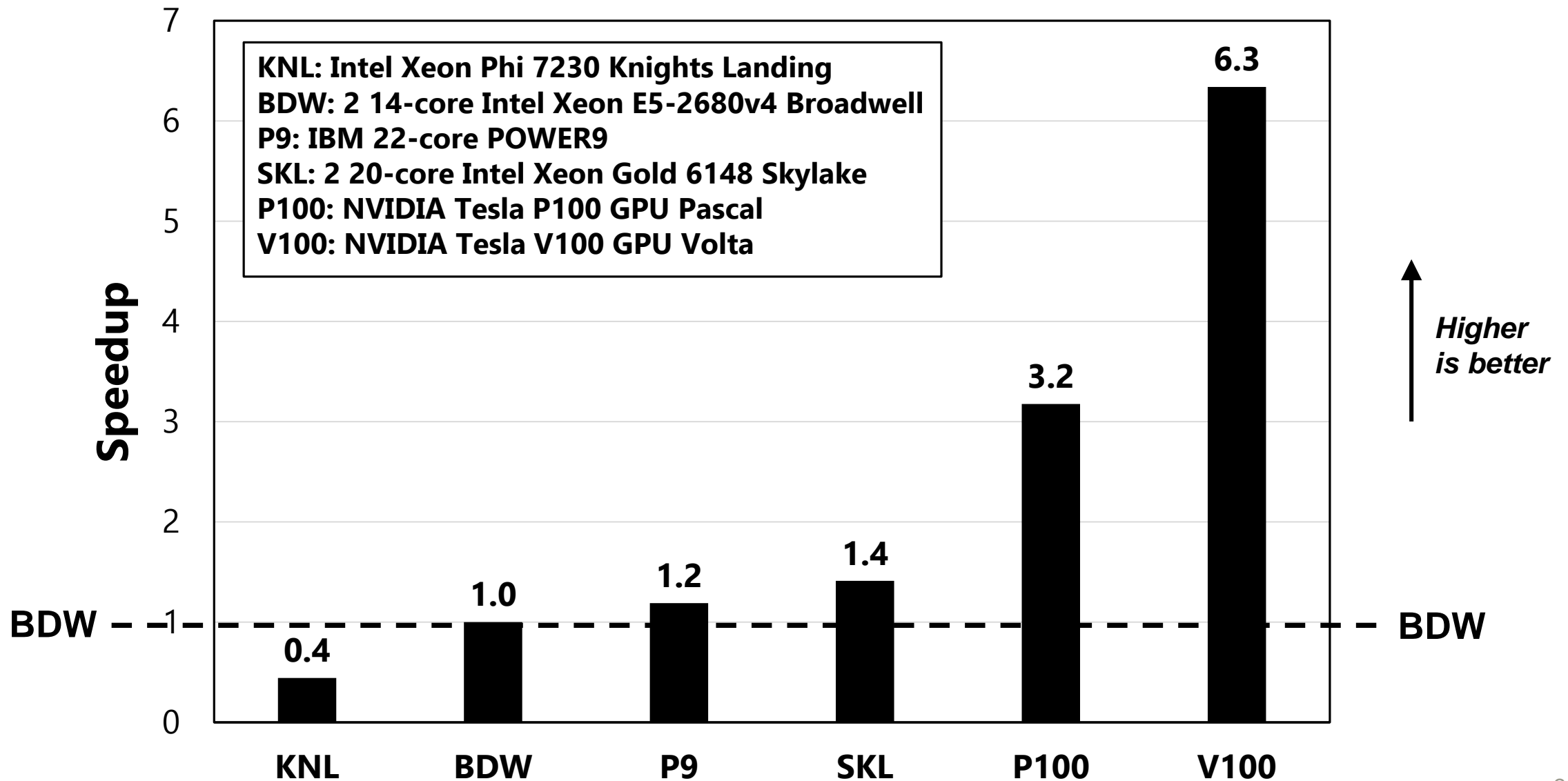
CUDA Optimization Strategies

Our heuristics:

- Maximize concurrency while maintaining memory coalescing
 - Node-based loop: one thread per variable if possible
 - Edge-based loop: 2× threads per node, use atomics
 - Cell-based loop: 1 warp per cell, atomics
 - Favor more blocks over more threads/block
 - Favor edge and cell-based loops
- Minimize temporary variables, seems to alleviate register pressure
- Atomics greatly outperform other race-avoidance strategies (for our code)
- Hard-code or template every scalar possible, but test performance



Device-Level Performance Relative to BDW



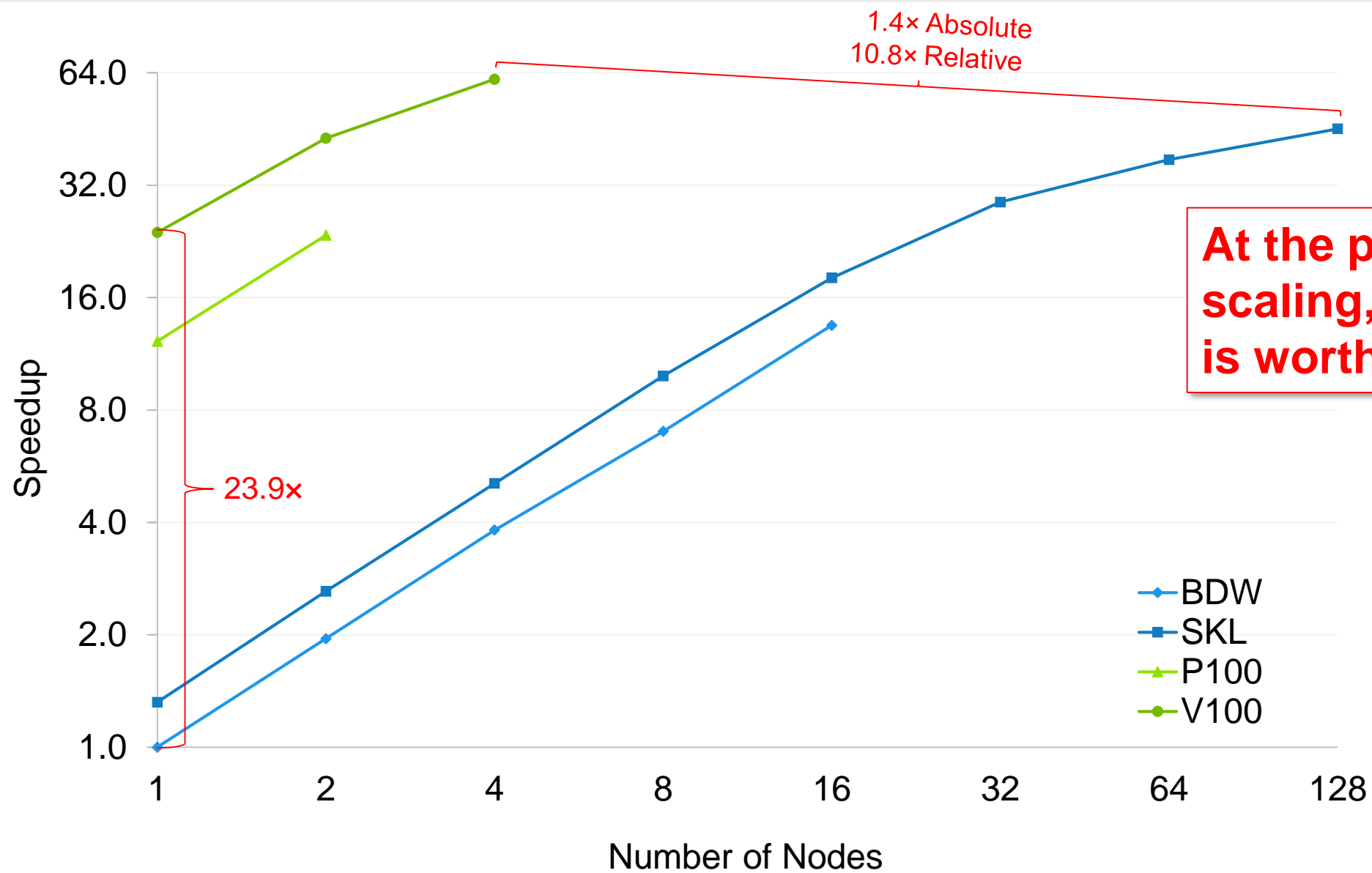


- Several small test cases from last year (2018)
- GPUs on NVIDIA PSG cluster, 4×V100 per node, CPUs on NASA Pleiades/Electra, Intel Xeon Broadwell/Skylake dual-socket nodes
- FUN3D strong scaling most dependent on multicolor point-implicit solver performance, $O(200)$ MPI transfers overlapped by interior computation
- MPI cost fixed, GPU solver 4-6× faster, less work to overlap comm., which poses a challenge for strong scalability
- **Absolute speedup:** compare node to node (dual-socket vs 4×V100)
- **Relative speedup:** speedup divided by number of devices



Strong Scaling

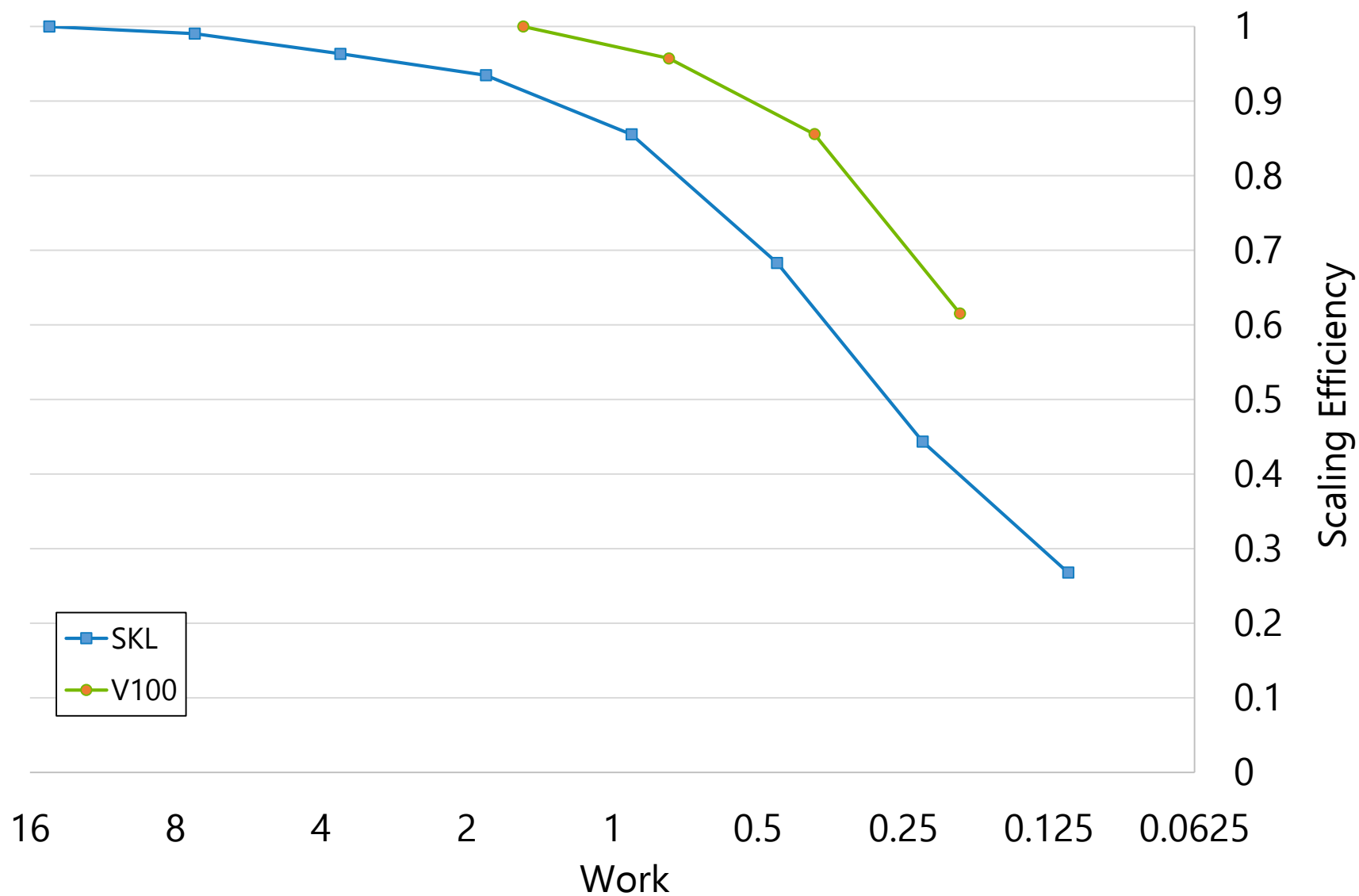
14.6 Million Grid Points





Strong Scaling

14.6 Million Grid Points

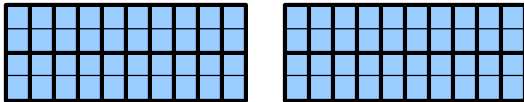




Typical x86 Node

~1 Teraflop

192 GB RAM



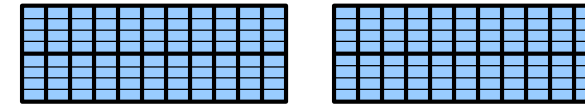
Dual Socket, 20-Core
Xeon Skylake
2 Threads/Core, AVX-512

ORNL Summit Node

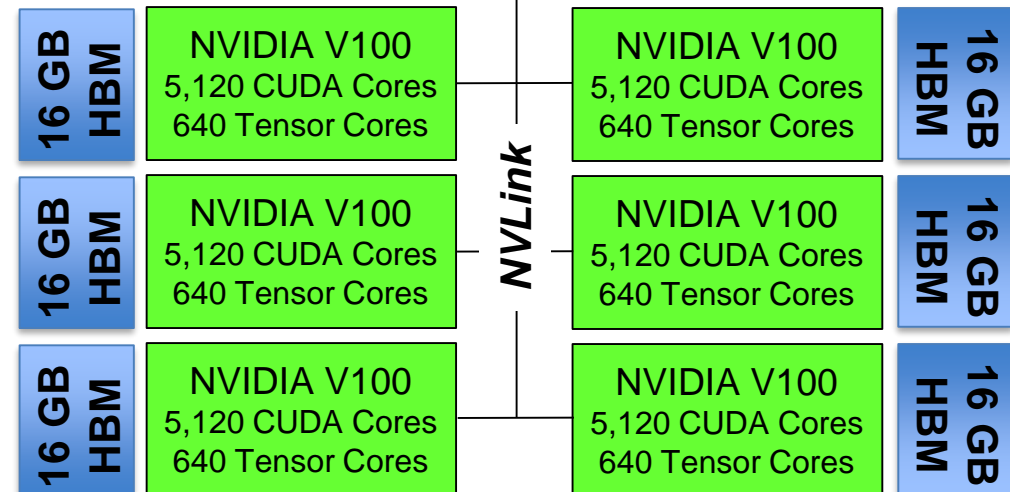
~40 Teraflops

1600 GB NVRAM

512 GB RAM



Dual Socket, 22-Core
IBM Power9
4 Threads/Core

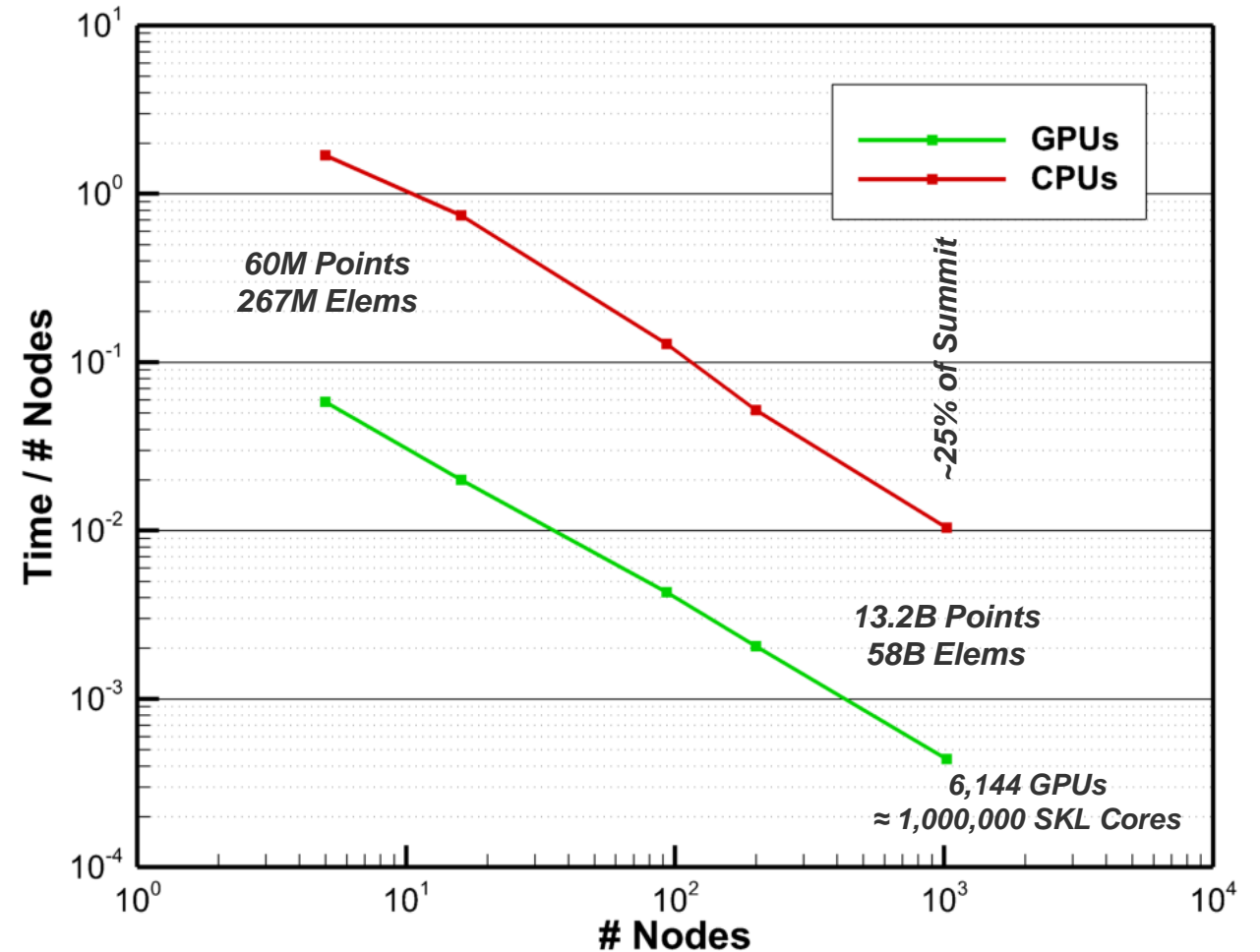




- Generally very good, with several caveats
- Occasional file system issues
- CUDA-aware MPI/GPUDirect performs 2× slower for our code as of December 2018, not the case earlier that year
- Pinning host MPI buffers worsens MPI performance after several timesteps to ~2× slower for the majority of a job
- Lack of pinning costs us >10% performance

Early Performance on Summit

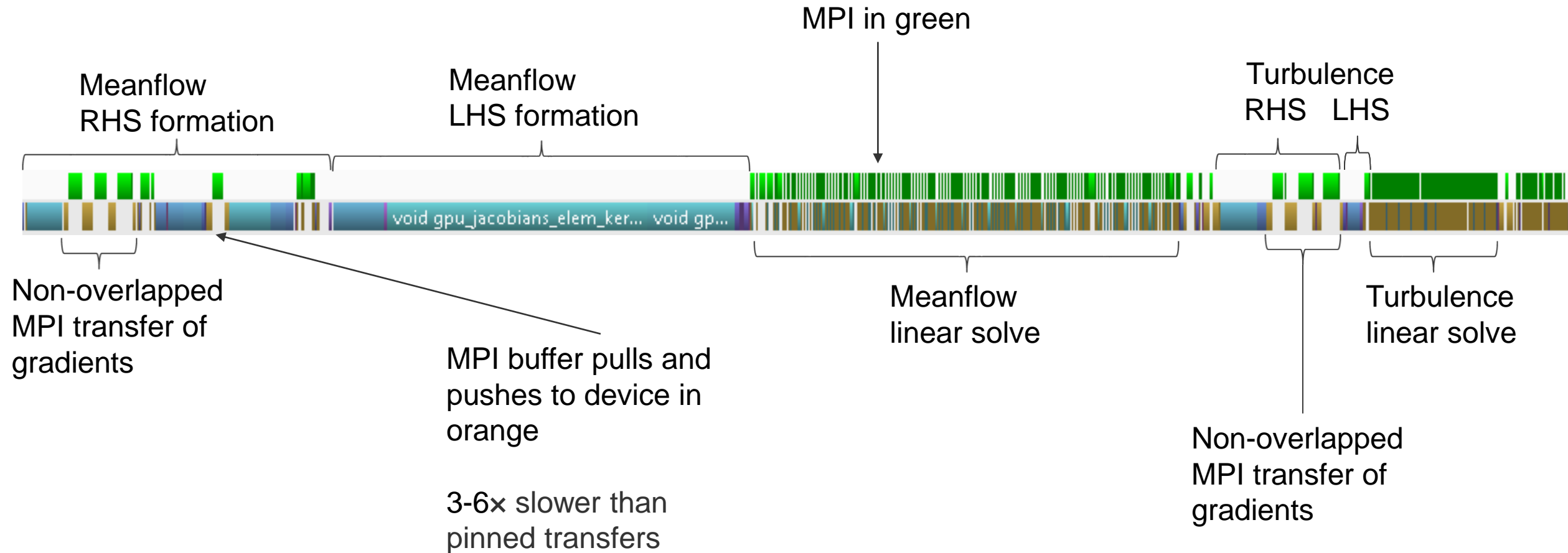
- Data taken during early access 4/2018
- Plot captures weak scaling:
 - Each node solves ~12.8M grid points
 - 5 nodes: 60M points (267M elems)
 - 1,024 nodes: 13.2B points (58B elems)
- **CPU curve** is MPI+OpenMP with 3 ranks/socket (total of 6 per node) with 168 total OpenMP threads per node (smt4 on Power9)
- **GPU curve** is MPI+CUDA: 3 ranks/socket shepherding 1 GPU each (total of 6 per node); all MPI via GPUDirect
- Nearly linear performance for both
- GPU node-level performance is 23x-37x faster at scale, correlates well with node-level studies





Current Summit Performance

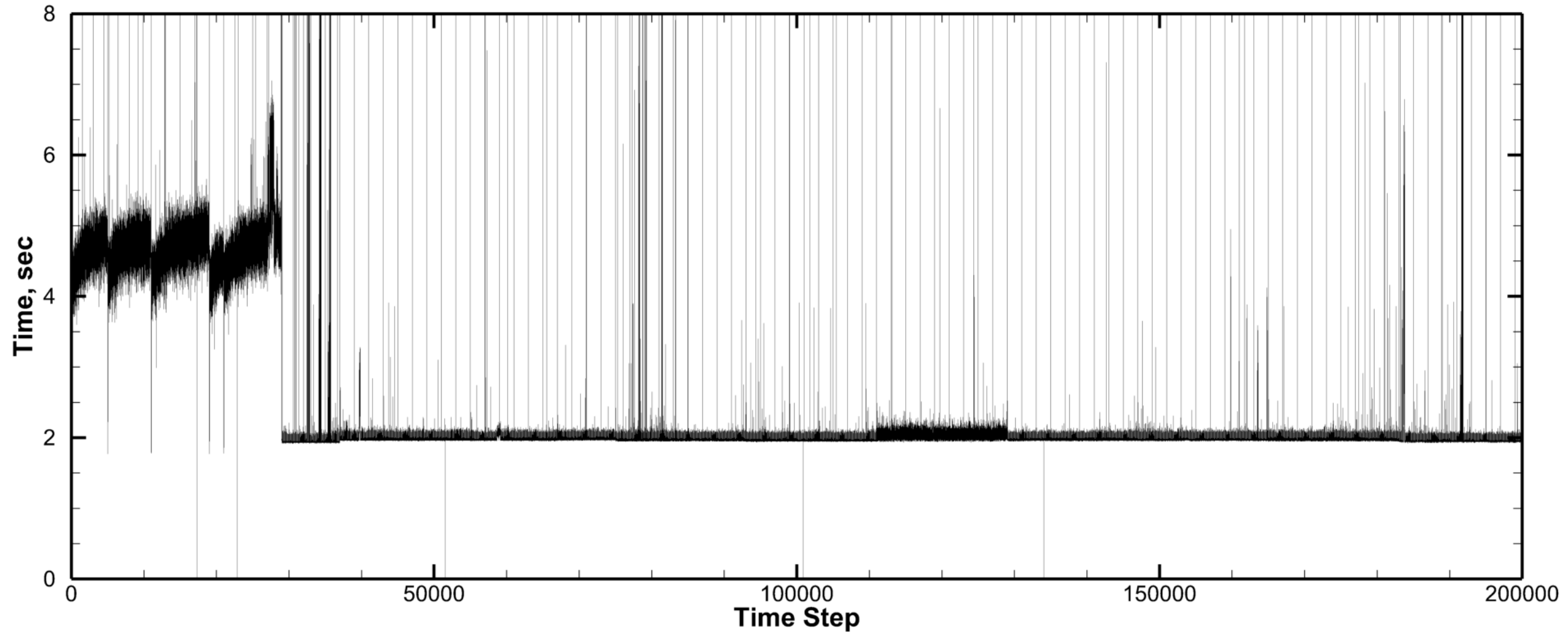
Case: 577 million grid points on 46 nodes
2.1M per device, apx 0.4s per subiteration



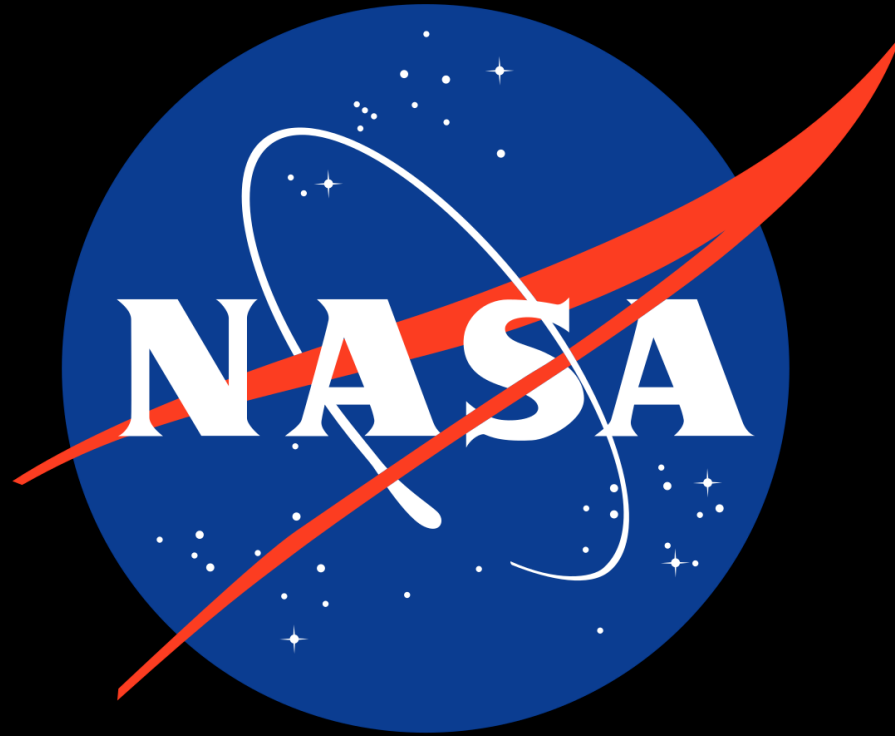


Current Summit Performance

1.14 Billion Grid Points



Thank you for having us!



Always seeking new collaborations
<https://fun3d.larc.nasa.gov>